Universidad Zaragoza
1542

Trabajo Fin de Grado

Diseño y Desarrollo de un Sistema de Machine Learning para la Detección de Defectos Adaptable para los Procesos Industriales

Design and Development of an Adaptive System of Machine Learning of Defect's Detection for Industry Process

Autor

Jiahe Qiu

Director

Javier Esteban Escaño

Escuela Universitaria Politécnica La Almunia

Junio 2024

Escuela Universitaria Politécnica - La Almunia
Centro adscrito
Universidad Zaragoza

ESCUELA UNIVERSITARIA POLITÉCNICA
DE LA ALMUNIA DE DOÑA GODINA (ZARAGOZA)

MEMORIA

Diseño y Desarrollo de un Sistema de Machine Learning para la Detección de Defectos Adaptable para los Procesos Industriales

Design and Development of an Adaptive System of Machine Learning of Defect's Detection for Industry Process

424.22.13

Autor: Jiahe Qiu

Director: Javier Esteban Escaño

Fecha: Junio 2024

ÍNDICE DE CONTENIDO BREVE

ÍNDICE DE CONTENIDO

INDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

1.1. RESUMEN

La demanda y la importancia de las PCB en la electrónica aumentan cada día, y los retos que presenta la inspección visual tradicional de defectos, como la imprecisión y la lentitud de la detección, hacen que es imprescindible introducir nuevas tecnologías de inspección visual. Por consiguiente, el proyecto propone un sistema automatizado de inspección mediante el empleo de un modelo de Deep Learning para identificar PCB defectuosas.

En el presente proyecto se detallarán los procedimientos necesarios para la creación de un sistema básico de detección de objetos. Se investigará acerca de los mejores métodos de detección de objetos, tales como YOLO-Nas y Faster R-CNN. En última instancia, se implementará el modelo en diversos sistemas operativos y hardware.

1.2. PALABRAS CLAVE

Aprendizaje automático; Aprendizaje Profunda; Placas impresas PCB; defectos de PCB; Sistema de detección de defectuosidades; norma IPC; Raspberry Pi; Jetson Nano; YOLO-Nas; Faster R-CNN; VGGNet; Python; Google Colab; Visión computacional; Detección de objetos.

1.3. ABSTRACT

The demand and importance of PCBs in electronics increases every day, and the challenges presented by traditional visual inspection of defects, such as inaccuracy and slow detection, make it essential to introduce new visual inspection technologies. Therefore, the project proposes an automated inspection system by employing a Deep Learning model to identify defective PCBs.

This project will detail the necessary procedures for creating a basic object detection system. The best object detection methods, such as YOLO-Nas and Faster R-CNN, will be investigated. Ultimately, the model will be deployed on different operating systems and hardware.

1.4. KEY WORDS

Machine Learning (ML); Deep Learning (DL); Printed Circuit Board (PCB); PCB defects; PCB defects detection system; IPC Standard; Raspberry Pi; Jetson Nano; YOLO-Nas; Faster R-CNN; VGGNet; Python; Google Colab; Computer Vision; Object Detection.

1.5. PRE-INTRODUCCIÓN

Desde hace más de medio siglo las placas de circuito impreso se volvieron populares en la electrónica de consumo, son componentes esenciales de muchos sistemas electrónicos contemporáneos, desde dispositivos electrónicos de uso privado y de hogar hasta equipos médicos y militares en el sector público.

Las PCB son placas de sustrato no conductor que se emplean para el montaje e interconexión de componentes electrónicos a través de rutas o pistas de un material conductor grabadas sobre el sustrato. Todo ello conlleva una serie de ventajas sobre otros tipos de placas." (BravoJordana, 2020).

Las PCB tienen las ventajas de que son de tamaños reducidos, presentan un alto nivel de repetibilidad y uniformidad de las características eléctricas. Como resultado, se reducen el tiempo de inspección en el proceso de control de calidad, y se requieren que el equipo del diseño y de la fabricación posea habilidades técnicas específicas y una capacitación mínima.

A medida que la tecnología evoluciona, se integran más componentes en las placas de circuitos impresos, lo cual implica una mayor complejidad en el diseño y la fabricación. Cualquier defecto superficial de PCB, ya sea causado por factor ambiental, producción o del tiempo, puede afectar negativamente al sistema.

En consecuencia, el control de calidad es un paso fundamental y esencial en el proceso de fabricación de PCB. No obstante, los procedimientos de inspección tradicionales son lentos y costosos, debido al factor humano que es imprevisible.

Por lo tanto, este proyecto tiene como objetivo en diseñar y desarrollar un sistema automatizado de inspección de defectos superficiales de las placas PCB, mediante un sistema de adquisición de datos y procesados de imágenes mediante machine learning.
El proyecto de investigación se dividirá en tres partes principales: primero se explicará el problema sugerido, luego se propone una justificación, proponiendo los alcances y objetivos del proyecto y luego se llevarán a cabo los estudios sobre el estado de arte del proyecto.

En la segunda sección se muestran los fundamentos teóricos fundamentales relacionados con los componentes de las placas PCB, normativas, composición y procedimientos de fabricación.
Asimismo, se presentarán los conceptos generales de machine learning, así como los algoritmos fundamentales empleados para la identificación de defectos en las placas PCB y las tecnologías pertinentes.

Por último, se presentará el algoritmo apropiado para la aplicación propuesta, las razones de selección, el proceso de desarrollo de la aplicación y los resultados alcanzados.

1.6. PLANTEAMIENTO DEL PROBLEMA

Dado el rápido avance de los circuitos integrados y las tecnologías de semiconductores, el tamaño de una placa PCB puede disminuirse a una dimensión considerablemente reducida, lo cual puede posibilitar una mayor complejidad en el proceso de diseño y de fabricación, así como una mayor laboriosidad en el proceso de control de calidad.

Como comenta (Ling & Isa, 2023), en la era de la industria 4.0, la manufactura de PCB se enfrenta a nuevos retos en los cuales exigen alto nivel de calidad, precisión y fiabilidad del producto. De modo que, los criterios en la fabricación de PCB resultan ser complicados, debido a los tradicionales métodos de inspección y el factor humano. Si un defecto de PCB no ha podido ser detectado con rapidez y precisión, el tiempo del personal empleado para la revisión y la cantidad de los productos defectuosos desechados son factores considerablemente adversos para la empresa.

En las inspecciones visuales realizadas por el operario, empleaba un microscopio simple o calibrado para determinar si la placa PCB era defectuosa o no y si era necesario un retrabajo de ella. Las herramientas que emplean son la inspección visual y la experiencia cualificada. Por lo que, el proceso de control de calidad dependía de la destreza del operario y la empresa tenía que invertir una cantidad de presupuestos para su formación.
Entonces las desventajas de esto son que es costoso a largo plazo, la detección del defecto es discontinuo y la recolección de datos era complicado.

Por otra parte, grandes números de las empresas se han adaptado a las técnicas AOI (Automatic Optical Inspection), que son más rápidos y precisos que las técnicas manuales de revisión, pero tienen la desventaja de que son sensibles a la luminosidad del ambiente. Por lo que muchas veces presenta resultados erróneos.

Por esta razón, para mejorar la precisión y la velocidad de la detección, consiguiendo una mejora de calidad y reduciendo los costes, los investigadores presentan una técnica de visión de máquina usando métodos tradicionales de algoritmos para el procesamiento de imágenes. Se aplica principalmente en dos áreas: en los defectos superficiales y materiales de las placas PCB y en los defectos de componentes y de soldadura de las placas PCB.

Este método de sustracción de imágenes está categorizada como método de comparación de referencia, que consiste en comparar la imagen de entrada con una imagen de referencia y mediante una operación lógica XOR pixel por pixel determina los defectos del PCB si las tiene.
Con esta técnica en (Chauhan & Bhardwaj, 2011) lograron la detección de defectos típicos, como grabados excesivos, grabados insuficientes y agujeros.
Mediante procedimientos posteriores de procesamiento de imágenes, también se obtuvieron las posiciones y tamaños precisos de los defectos.
No obstante, este método presenta sus limitaciones debido a que la imagen de referencia debe tener el mismo tamaño y orientación que la imagen inspeccionada en términos de píxeles, y que son sensibles a la luminosidad del ambiente.

Por consiguiente, es fundamental establecer un sistema automatizado de inspección a través del entrenamiento de un modelo seleccionado de Deep Learning que pueda clasificar una placa PCB defectuosa de una ideal.

1.7. JUSTIFICACIÓN

En el ámbito de la manufactura de PCB, se emplean las normas IPC para clasificar el producto en función de su tipo, complejidad, desempeño y aspectos de producción, con el propósito de garantizar una uniformidad durante todo el proceso de diseño, producción y entrega al cliente.

Según las diferentes aplicaciones que destine la placa impresa, se clasifica desde clase 1 hasta 3, siendo la clase 3 la más compleja.

Se emplea la norma IPC-2221 para el diseño de PCB, IPC 600 para la fabricación PCB, IPC-7351 para el diseño de huellas o footprints e IPC 610 para aceptar o rechazar las PCB según la soldadura de tarjeta o productos electrónicos.

En este proyecto, la investigación se centrará en la inspección de los casos que incumplan la norma IPC-610. Y según los criterios de aceptación IPC-610 (Laverde, 2020a), los errores de soldadura, superficiales y materiales más comunes en PCB son:

- Áreas de contacto no soldadas
- Ampollas y delaminación
- Aréolas
- Delaminación, muescas, grietas según la IPC-600
- Quemaduras
- Depanelización
- Falta de Conductores/pistas
- Conductores/pistas-levantados
- Antisolder
- Residuos de flux

Bajo este contexto, el objetivo de la investigación consiste en el diseño y el desarrollo de un sistema de adquisición y procesado de imágenes mediante Deep Learning que sea capaz de detectar errores superficiales en las placas PCB tales como errores en las pistas de cobre, pérdida de lámina, mala soldadura y efectos en el estaño(Sanín Ramírez & Nates Huertas, 2021), con el fin de conseguir un proceso de calidad automatizada de alta precisión y rápida.

1.8. OBJETIVOS

El objetivo principal es:

Diseñar y desarrollar un sistema de detección de defectos en la fabricación de PCB, construir un sistema de adquisición de datos y procesados de imágenes mediante técnicas de Deep learning.

Los objetivos específicos:

1. Caracterizar los diferentes casos de defectuosidad según la norma IPC-610.
2. Identificar algoritmos de procesamiento de imágenes para la detección de defectuosidades en las PCB.
3. Implementar un sistema de software y hardware para detectar las defectuosidades de las placas PCB.
4. Efectuar una evaluación de precisión y control del prototipo.

1.9. ANTECEDENTES

Para llevar una investigación de una forma más integral, se llevará a cabo un estudio previo sobre las investigaciones existentes del tema y se recolectarán las diferentes propuestas junto con las ventajas e inconvenientes de cada una de ellas.

Acuerdo a los estudios realizados por (Ling & Isa, 2023), la evolución del proceso de control de calidad de PCB comenzó con las inspecciones visuales por operarios con experiencia, después se sustituyó por las técnicas de inspección automática óptica (AOI), más tarde se implementó las detecciones de defectos y procesado de imágenes mediante visión artificial y Machine Learning, mejorando la precisión y la velocidad de ejecución, y finalmente con las R-CNN de Deep Learning alcanzó un nivel de precisión y ejecución aún más alto.

A diferencia de los métodos de visión artificial, los enfoques basados en CNN (Convolutional Neural Network) pueden extraer automáticamente las características de la imagen, simplificando el proceso de preprocesamiento de la imagen y mejorando así la precisión y la velocidad de detección.
Además, los métodos basados en CNN son más robustos al entorno del trabajo y al ruido. Aunque puedan existir sombras o reflejos, aún se pueden lograr resultados significativos en la detección de objetos debido a la capacidad de extracción de características multicapa de estos métodos.

Las CNN más utilizadas y modificadas para detectar defectos de PCB son la red neuronal convolucional basada en regiones rápidas (Faster R-CNN) y las series "You Only Look Once" (YOLO).

1.9.1. TDD-net (Tiny Defect Detection Network

En el estudio (Ding et al., 2019) propone una pequeña red de detección de defectos (Tiny Defect Detection Network, TDD-net) basada en Faster R-CNN. Aplica la agrupación de k-medias a los cuadros delimitadores del conjunto de datos de entrenamiento de PCB para buscar automáticamente escalas de caja de anclaje o región propuesta razonables.
Luego, se implementaron métodos data augmentation, con el objetivo de generalizar el modelo creado, creando más datos de entrenamiento con la adición de ruido gaussiano, cambio de luz, rotación de imágenes, volteo, recorte y desplazamiento aleatorios.

Para mantener las características que tienen un nivel semántico bajo, se adoptó la arquitectura de Feature Pyramid Network (FPN). En FPN, las características con baja resolución y semántica fuerte están conectadas con características que poseen alta resolución y semántica débil mediante conexiones laterales de arriba a abajo, y predice características en cada nivel de la red piramidal.

Esta red sigue al modelo de R-CNN, pero trae tres novedades:
Primero, caja de anclaje razonable están diseñados mediante el uso de agrupaciones de k-medias.
En segundo lugar, TDD-Net fortalece la relación de los mapas de características de diferentes niveles y se beneficia de información estructural de bajo nivel, que se adecua para la detección de defectos pequeños.
Finalmente, teniendo en cuenta el conjunto de datos pequeño y desequilibrado, la minería de ejemplos duros en línea se adopta en toda la fase de capacitación para mejorar la calidad de las propuestas de regiones de interés (ROI) y hacer un uso más eficaz de la información de los datos.

Sin embargo, este algoritmo es de estructura de dos etapas (Two-Stage) y contiene grandes parámetros, lo que implica que la velocidad de detección es más lenta que la de la red de una etapa (One-Stage).

1.9.2. Faster R-CNN y FPN

En el estudio de investigación (Hu & Wang, 2020) propusieron un método de detección de defectos en la superficie de PCB basado en Faster R-CNN y FPN mejorados. Se adoptó un R-CNN más rápido utilizando como Backbone el Residual Neural Network (ResNet50) para la imagen de entrada como detector. Se utilizó FPN para fusionar características profundas y superficiales, lo que se considera una forma poderosa de promover la precisión de la detección de objetos pequeños.

Se implementaron unidades residuales Shuffle V2 para disminuir el cálculo de toda la red. Se utilizó el algoritmo de red de propuestas de región de anclaje guiada (GARPN) para producir anclajes más precisos y luego se ejecutaron capas de agrupación de RoI para obtener propuestas. Por último, las capas completamente conectadas se utilizaron para clasificar y realizar una regresión del cuadro delimitador para lograr la detección final de defectos.

El Data Augmentation fue aplicado para obtener un total de 12.000 imágenes de defectos. Este método logró un mAP (mean Average Precision) de detección del 94,2 %, que fue mejor que el de otros métodos de última generación, incluidos Faster R-CNN, RetinaNet y YOLOv3.

Este modelo logró una detección más rápida que el Faster R-CNN original mediante determinadas técnicas. Sin embargo, no estaba capacitada para la inspección a tiempo real.

También tenía la inconveniencia de detectar defectos desconocidos debido a que los defectos desconocidos en el conjunto de entrenamiento aparecían en diversas características.

1.9.3. Modelos YOLO (You Only Look Once)

Los autores de (Liao et al., 2021) utilizaron YOLOv4 con un Backbone mejorada para detectar defectos en la superficie de los PCB.

En este modelo, Darknet53, tiene un mayor consumo de memoria en YOLOv4, fue reemplazada por la red liviana MobileNetV3 para acelerar la detección. MobileNetv3 utiliza la convolución separable en profundidad para capturar mapas de características e incluye dos pasos: convolución en profundidad y convolución puntual.

La cantidad de parámetros en la convolución separable en profundidad es mucho menor que la de la convolución normal, lo que hace que los parámetros completos sean aproximadamente 40 % menos que el YOLOv4 original.

El modelo propuesto obtuvo una precisión de detección de 98,64 % mAP, mayor que el de Faster R-CNN, RetinaNet, Single Shot MultiBox Detector (SSD), YOLOv3 y YOLOv4.
Significativamente, la velocidad de respuesta fue satisfactoria a 56,98 fotogramas por segundo con la GPU RTX3080. Sin embargo, los números de cada tipo de defecto se mantuvieron manualmente para que fueran aproximadamente iguales para que el conjunto de datos estuviera equilibrado en muestras, lo que contribuye al buen rendimiento.

Sin embargo, este modelo presenta dos inconvenientes:

1. Las imágenes de datos de entrenamiento únicamente contenían un defecto por imagen. Lo que implica su incapacidad de detectar varios defectos en la misma imagen.
2. El tamaño de las imágenes de entrenamiento era 416*416 píxeles, mientras que la imagen de PCB en tamaño real es 70 veces más grande.

De esta forma, en el entorno real del trabajo, la velocidad de respuesta y la precisión de inspección se reducirían significativamente.

(Adibhatla et al., 2021) aplicaron YOLOv5 para detectar defectos en PCB gracia a su estructura mejorada, de alta velocidad y de menor tamaño respecto a las versiones anteriores de YOLO.
En este estudio, se utilizaron tres modelos YOLOv5 de diferentes tamaños (pequeño, mediano y grande). En este método se implementó el aumento de datos en mosaico, empalmando cuatro imágenes mediante escalado aleatorio, recorte y disposición aleatorios, para enriquecer el conjunto de datos y disminuir la carga de la GPU con un tamaño de conjuntos de datos pequeños.

El Backbone de este modelo se mejoró mediante la adición de la red Cross Stage Partial Network (CSPNet), lo que la hace más ligera a la vez que mantiene la precisión. Además, se combinó Path Aggregation Network (PANet) con FPN para fusionar características de varios niveles, lo que debería mejorar la precisión de la detección, especialmente para objetivos de pequeño tamaño.

En el trabajo, los tres modelos YOLO-v5 de diferentes tamaños consiguieron buenos resultados. El modelo YOLO-v5 de gran tamaño obtuvo la mejor precisión de detección con un 99,74 %. Sin embargo, este trabajo solo aplicó los modelos YOLOv5 para llevar a cabo la tarea de detección de defectos en PCB, sin modificación o innovación para el algoritmo.

Los inconvenientes de este modelo son:

1. Solo es capaz de la clasificación binaria
2. Las imágenes de entrenamiento y de prueba son de tamaño pequeño (400*400 píxeles)

Por lo tanto, para su aplicación en el ámbito real del trabajo se deben buscar un punto de equilibrio entre la precisión y la velocidad de detección.

1.9.4. Modified FCOS

(Y. Zhang et al., 2021)modificaron la red Fully Convolucional One Stage (FCOS) para lograr la detección de defectos superficiales en PCB.
Los autores reemplazaron el Backbone de ResNet101 con una red ligera de MobileNetV2 para disminuir los parámetros del modelo. El módulo de atención de bloques convolucionales se aplicó en MobileNetv2 para mejorar el rendimiento de extracción de características. Se implementó FPN basado en PANet en lugar del FPN tradicional en el modelo de FCOS. Además, la función de pérdida de regresión del cuadro delimitador se reemplazó con una pérdida de distancia-intersección sobre unión (IoU) para aumentar la precisión de la detección de defectos más pequeños.

Este modelo propuesto en (Y. Zhang et al., 2021) logró un mAP de detección del 44,3 %, que fue mucho mayor que el del modelo FCOS tradicional. El detector logra una propuesta libre y libre de anclajes, reduciendo significativamente el número de parámetros.
Además, al eliminar el marco de anclaje, el detector evita por completo cálculos complejos de IoU y la coincidencia entre las cajas de anclaje y las cajas de verdad del terreno durante el entrenamiento, lo que reduce la huella total de memoria de entrenamiento aproximadamente dos veces.
Por lo tanto, la velocidad aumentó en un 30 % a 37,6 milisegundos de tiempo de inferencia en comparación con la de Faster R-CNN en este trabajo.

Sin embargo, solo se recopilaron 1455 imágenes basadas en el conjunto de datos de (Ding et al., 2019) para el entrenamiento mediante este modelo. Como resultado, este modelo obtuvo un mAP del 44,3 % para imágenes de alta resolución, que fue un 5% menor en comparación con Faster R-CNN debido al abandono de regiones propuestas y la insuficiencia de datos.

1.9.5. CS-ResNet

En el entorno real del trabajo, la distribución de clases de defectos siempre está desequilibrada. (H. Zhang et al., 2021) propusieron una red neuronal convolucional residual sensible a los costos (CS-ResNet) para superar el problema de los datos desequilibrados.

En el estudio, se aplicó una capa de ajuste sensible al costo a la capa completamente conectada de ResNet. Después de aplicar la función softmax, se obtuvo una función de pérdida ponderada basada en el grado de desequilibrio del conjunto de datos de entrenamiento. El peso wt se calculó mediante la Ecuación:

$$w_t = \alpha * \frac{cnt_p}{cnt_r}$$

- α: factor de ajuste (constante >= 1.0)
- cntr: número de defectos reales
- cntp: número de pseudodefectos

Luego, el peso wt estuvo involucrado en el cálculo de la función de pérdida de entropía cruzada de softmax, lo que significa que el grado de desequilibrio del conjunto de datos se consideró parte de la función de pérdida.

La introducción del peso wt en(H. Zhang et al., 2021) mitigó los efectos del conjunto de datos desequilibrado.

El CS-ResNet propuesto logró un mejor rendimiento que el ResNet estándar y otros métodos basados en CNN. La sensibilidad de CS-ResNet fue de 0,89, lo que supone un aumento del 12 % con respecto a la de ResNet.

Uno de los inconvenientes de este método es que es necesario elegir un factor de ajuste α adecuado para diferentes conjuntos de datos con diversos grados de desequilibrio.

Puede llevar a cabo una clasificación binaria con éxito; no obstante, no está asegurado de detectar defectos de diversos tipos. Para tareas reales de inspección de PCB, habrá diferentes tipos de defectos. El parámetro wt basado en el total de defectos reales y el total de pseudodefectos no puede representar los grados de desequilibrio de cada tipo, lo que dificulta obtener un mejor rendimiento.

2. MARCO TEÓRICO

2.1. PRINTED CIRCUIT BOARD (PCB)

2.1.1. Definición

Los circuitos impresos, conocidos como Printed Circuit Board (PCB), es una placa donde se realiza las interconexiones eléctricas de distintos elementos que conforman el circuito.

Apareció en los años 30 del siglo pasado, gracias a un ingeniero llamado Paul Eisler, y se definen como placas de sustrato no conductor que se emplean para el montaje de interconexiones eléctricas a través de pistas de un material conductor grabadas en el sustrato(li, 2022), lo que permitió la reducción de dimensiones, peso y producción de masa de los sistemas.

2.1.2. Materiales y tipología de PCB

Los sustratos se definen como los compuestos que proporcionan el soporte sobre los que se produce el grabado de las pistas y ensamblajes de los componentes de las placas PCB. (Pacheco, 2023)

La mayoría de los materiales de las PCB son sustratos de resina epoxi reforzada, fibras de vidrio y sustrato de resina epoxi unido con una lámina de cobre. Existe una amplia variedad de sustratos implementados como FR-4, CEM-1, RF-35, etc.

A la hora de seleccionar el material adecuado, se debe tener en cuenta en la temperatura Tg.
La temperatura Tg es una de las propiedades primarias para clasificar un sustrato a la hora de fabricar PCB, es la temperatura de la cual el material transforma desde un estado relativamente rígido a un estado cristalino, convirtiéndose en sí un estado más deformable con una significante variación en módulo dinámica de viscoelasticidad. (Coombs & Holden, 2016)

Normalmente, las placas se clasifican según el número de capas con las cuales son fábricas. Actualmente, existen en el mercado las PCB de una sola capa, de dos capas y multicapas.

2.1.2.1. PCB de una sola capa

En esta configuración solo tiene una capa de sustrato, donde la pista conductora de cobre solo se encuentra en una de las caras y se

Ilustración 1 PCB de una sola capa (Pacheco, 2023)

realiza las interconexiones eléctricas de los componentes. Es la configuración más económica y sencilla a la hora de diseño y fabricación, pero a la vez las aplicaciones que se abarca son las más simples.

2.1.2.2. PCB de dos capas

En este caso también tiene una sola capa de sustrato; sin embargo, tiene en ambas caras recubiertas de cobre, por lo que permite interconexiones de componentes en ambos lados, permitiendo crear un sistema de mayor complejidad que el caso anterior.

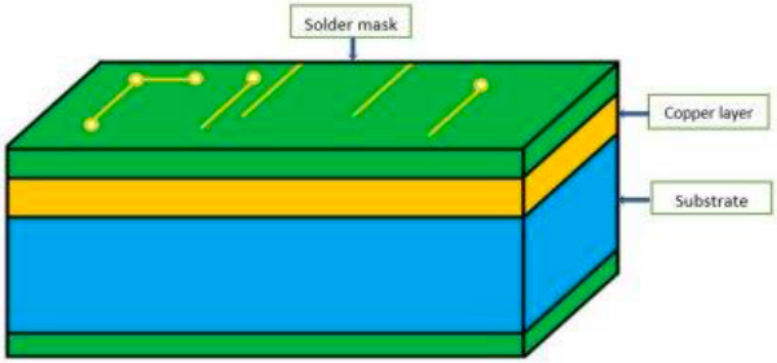


Ilustración 2 PCB de dos caras (Pacheco, 2023)

2.1.2.3. PCB multicapas

En este caso, la placa contiene más de tres caras conductoras interpoladas entre capas de aislamiento para garantizar que el exceso de calor no dañe a los componentes. Generalmente, en este tipo de PCB

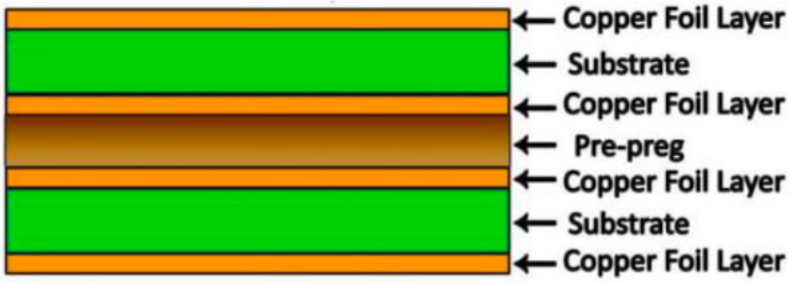


Ilustración 3 PCB Multicapas (Pacheco, 2023)

se utilizan planos de energía para evitar el sobre-ruteo de pistas de alimentación. También para mantener la integridad de señales se emplean vía ciegas, que comienza en una capa externa y termina en una interna.

2.1.3. Tipología de encapsulado de componentes

Actualmente, existen principalmente tres familias de encapsulados electrónicos, y en función de ello, la tecnología de ensamblado también varía.(Limitada, 2024)

2.1.3.1. Thru-Hole (A través de orificio)

Este tipo de encapsulado se caracteriza en que los componentes poseen pines para ser instalados en perforaciones metálicas (Thru-Hole Pad). Se sueldan por la capa opuesta y generalmente se sueldan en solo una de las caras de PCB

2.1.3.2. SMD (Dispositivos de Montaje superficial)

Este tipo de encapsulado permite que los componentes se monten superficialmente sobre las PCB. Pueden soldarse en ambos cara de PCB y debido a sus reducidas dimensiones, permite crear sistemas más pequeños y denso para aplicaciones de alta frecuencia.

2.1.3.3. BGA (Arreglo de bolas en rejilla)

Se emplean principalmente para chips que contiene grandes cantidades de pines (de 300 a 1000). Se requiere el uso de maquinaria para su instalación, ya que los pines son bolas de soldadura que

necesitan ser fundidas para conectarse con los pads, por lo tanto, la alineación es esencial. Es ideal para circuitos integrados de alta frecuencia, ya que la inductancia es mínima en este caso.

2.1.4. Diseño de PCB

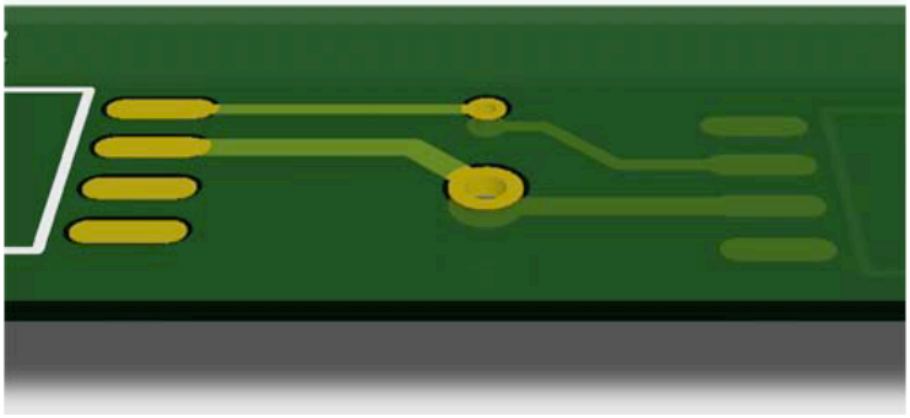
A continuación, se va a introducir sobre las diferentes tecnologías de ensamblaje de componentes sobre PCB y los detalles sobre las pistas y pads.

2.1.4.1. Tecnologías de ensamblajes

2.1.4.1.1. Tecnología de agujero pasante (THT)

Esta tecnología utiliza las perforaciones metálicas (THT pad) que se realizan en las PCB para el ensamble de los elementos con la finalidad de crear una interconexión entre las caras opuestas.

Ilustración 4 THT (Limitada, 2024)



Esta tecnología tiene la inconveniencia de que es sensible a la alta temperatura, por lo que si sobrepasa de los parámetros establecidos se puede generar un fallo en el funcionamiento del circuito.

Se emplean en diseños que requieren de una mayor robustez o que no tenga una restricción en sus dimensiones.

2.1.4.1.2. Tecnología de montaje superficial (SMD)

Este tipo de tecnología ensamblan los componentes electrónicos mediante contactos en la cara inferior de la placa, es decir, se sueldan sus terminales de conexión a Pads ubicados en la misma cara donde están colocados sin el uso de perforaciones.

Reduce los espacios e incrementa la eficacia de las PCB

2.1.4.2. Pad, Soldemask, Silkscreen y Track

Una Pad es una superficie de cobre en una placa PCB que permite soldar o fijar la componente a la placa, puede ser de agujero pasante (THT) o de montaje superficial (SMT).
Puede adquirir diferentes formas en función de la tecnología de ensamblaje que utiliza (rectangulares para SMT y ovaladas para THT).

El soldemask es una capa protectora contra la oxidación que recubre la PCB, excepto aquellas partes donde se soldaran los componentes y para evitar que la soldadura cortocircuite accidentalmente dos pistas de diferentes nodos. Se trata de barniz que se aplica en la etapa de fabricación y puede ser de colores variados (verde, rojo y azul).

Serigrafia o Silkscreen es el proceso de impresión sobre la máscara de soldado información conducente a facilitar la labor del ensamblado y de verificación. Indican puntos de prueba, posición de orientación y referencia de las componentes que conforman el circuito.

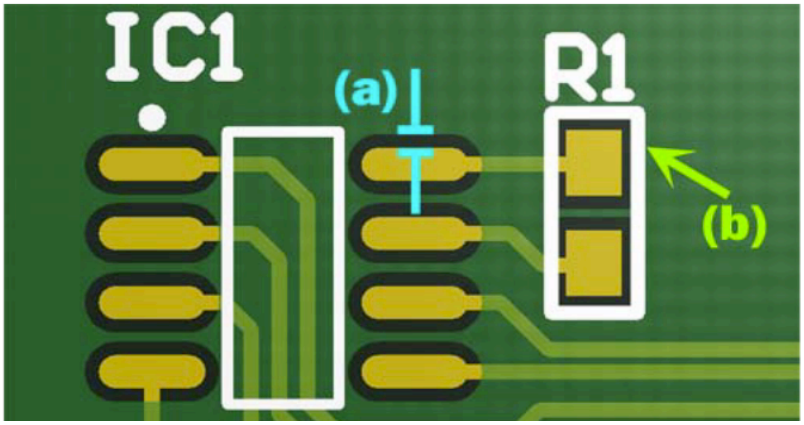


Ilustración 5 Silkscreen (Limitada, 2024)

Un track o pista es un camino conductor de cobre que tiene la función de conectar a diferentes Pads. Su grosor depende de la corriente que circule por ella, por lo que es necesario calcular el ancho con el objetivo de que exista una adaptación de impedancias durante todo el recorrido. (Limitada, 2024)

2.2. NORMAS IPC

Las normas IPC es un estandar técnica basada en la ciencia, tecnología y experiencia, aprobada por la Association Connecting Electronics Industries con el fin de conseguir niveles altos de calidad en las áreas de diseño, fabricación, ensamble, inspección de PCB y los elementos pasivos, activos eléctricos y electrónicos.

Las ventajas que traen estos criterios, (Laverde, 2016), son las siguientes:

- Trabajar con las mejores prácticas de la industria y sus necesidades.
- Comprometerse y demostrar la excelencia de la empresa y las personas, ganar reconocimiento.
- Son para uso de los diseñadores en su labor
- Ayudan a eliminar problemas de interpretación entre fabricantes y usuarios.
- Facilitan el intercambio de información.
- Ayudan en la mejora de los productos.
- Reducir demoras en el desarrollo de productos.
- Reducir el tiempo de los procesos.
- Ayudar a diseñar orientado a la manufactura, ambiente.
- Reducir el tiempo de lanzamiento al mercado.

Existen diferentes normas para cada etapa del procesado de PCB, como IPC-2220 para el diseño de PCB, IPC-7711/21 para el retrabajo de PCB, IPC-JSTD-001 para las soldaduras de los componentes eléctricos y electrónicos ensamblados, etc.

En este caso se centrará en la norma IPC-600, que es la norma de aceptabilidad de PCB.

2.2.1. IPC-600

Se trata de un control de calidad previo al ensamble, con el fin de detector las placas defectuosas, evitando la pérdida económica y reduciendo el retrabajo posterior sobre ello.

Los criterios de aceptación (Laverde, 2020b) son las siguientes:

2.2.1.1. Áreas de contacto no soldadas

Las zonas de PCB que deben estar despejadas de soldadura u otros contaminantes, porque van a ser utilizadas para conectar mecánicamente una pieza, como un tornillo para fijar la placa PCB a la caja y encerramiento.

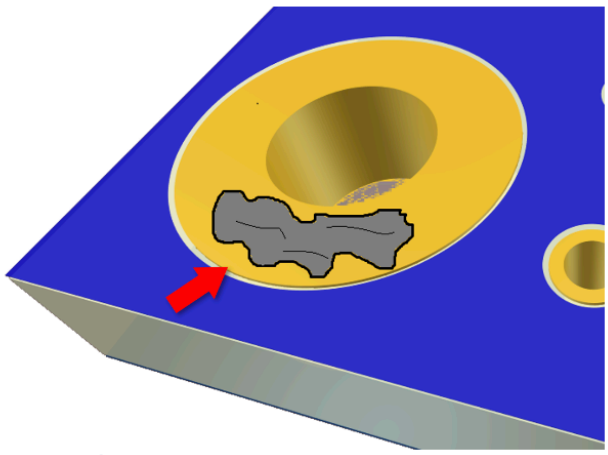


Ilustración 6 Áreas de contacto no soldadas

2.2.1.2. Ampollas y delaminación

Son resultados de debilidades en el material de laminado o fallas en el proceso, son aceptables mientras no sean conductivas y que las burbujas y delaminación no exceden el 50 % de la distancia entre conductores.

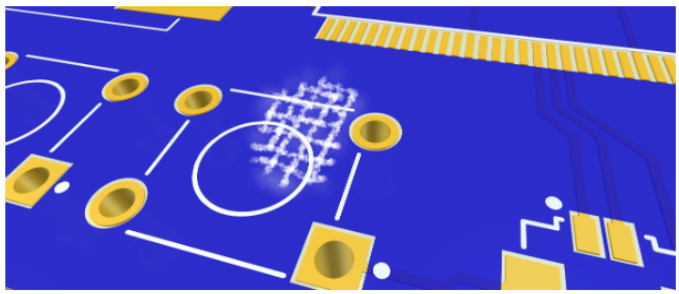


Ilustración 7 Ampollas y delaminación

2.2.1.3. Aréolas

Son una condición del laminado debido a presión por mecanizado. Defecto si la distancia entre la aureola y el elemento conductivo es menor a 0.1 mm.

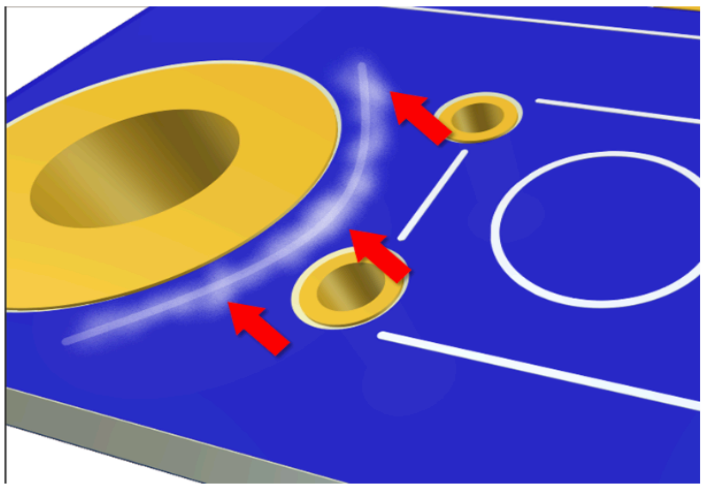


Ilustración 8 Aréolas

2.2.1.4. Delaminación, muescas, grietas

Es la separación entre capas del laminado, excede 0.25 mm, reducen el espacio de los conductores, hay facturas en el borde del laminado.

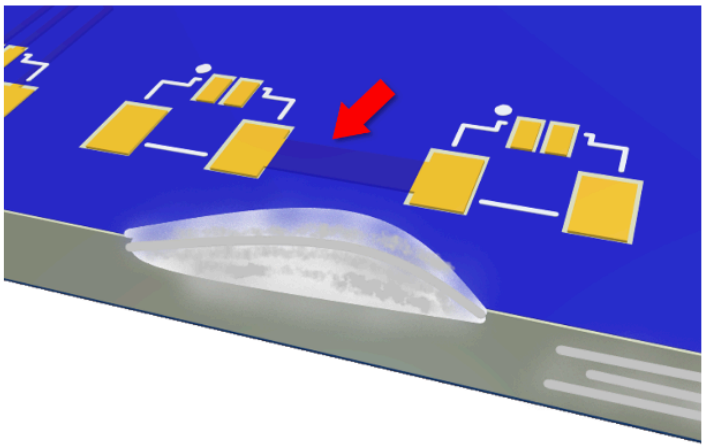


Ilustración 9 Delaminación, muescas, grietas

2.2.1.5. Quemaduras

La tarjeta tiene evidencia de quemado, lo que causa desprendimiento y daños posteriores.

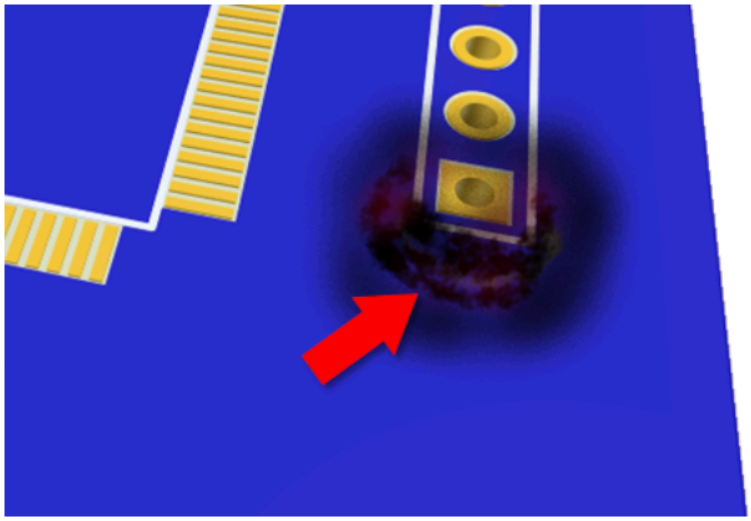


Ilustración 10 Quemaduras

2.2.1.6. Depanelización

Defectos del laminado en los bordes, ásperos y dañados, deshilachados, rebabas, muescas, resultado de la separación del panel, que sobresalen y afectan la forma, ajuste y función, en tareas de ensamble posterior.

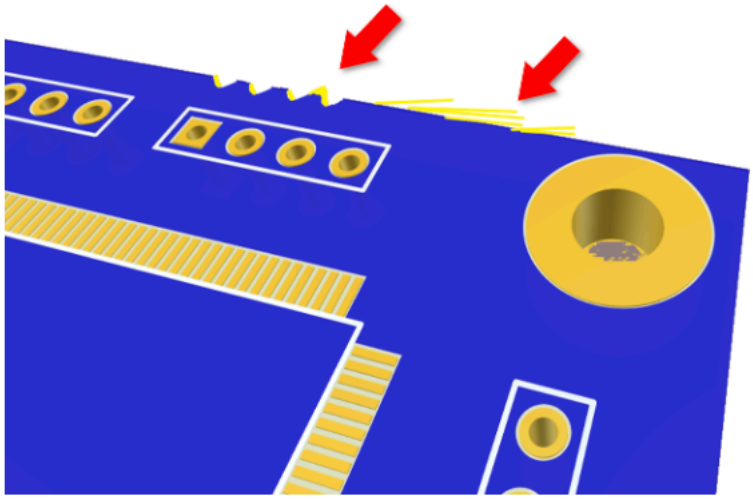


Ilustración 11 Depanelización

2.2.1.7. Conductores/pistas incompletas

Defectos de fabricación de PCB en las pistas, como muescas, orificios, bordes rugosos, rasguños, que pueden tener impacto en la

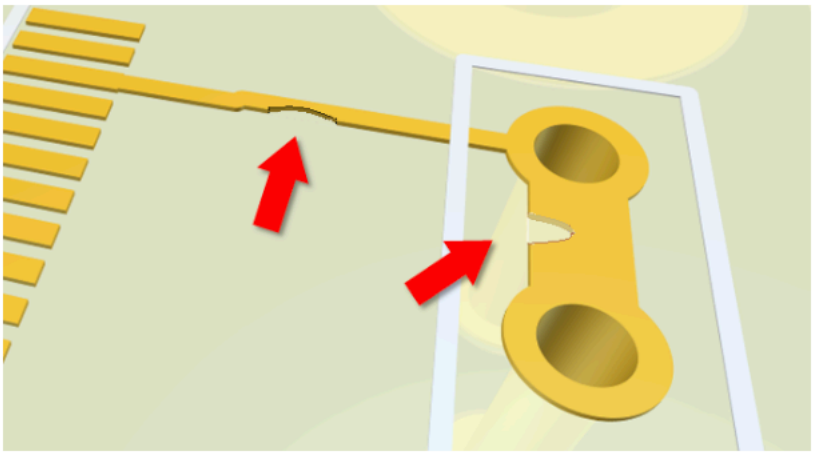


Ilustración 12 Conductores/pistas incompletas

corriente que circula por ella o en la impedancia en RF. Se debe a la contaminación en el proceso.

2.2.1.8. Conductores/pistas levantadas

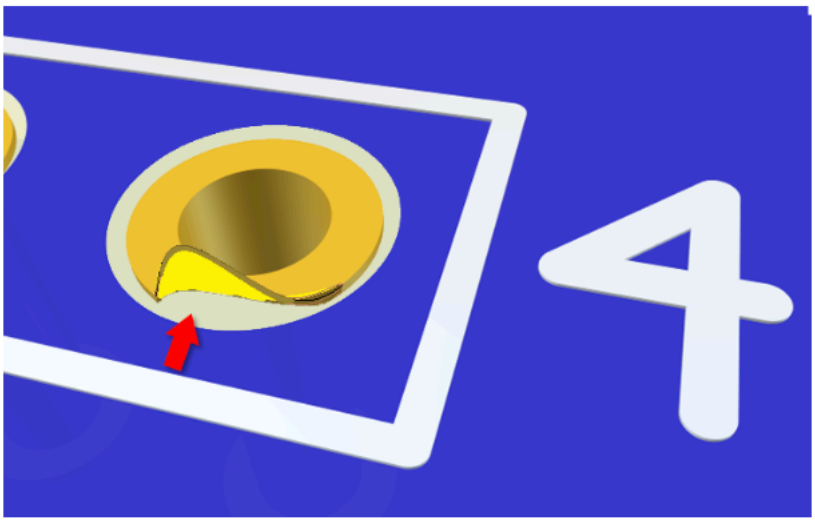


Ilustración 13 Conductores/pistas levantadas

Separación entre un pad y el laminado, mayor que el espesor de la pista misma. Vías separadas o levantadas. Se debe a la presión mecánica-térmica. El pad se puede desprender total o parcialmente.

2.2.1.9. Marcas o serigrafía

Screen, marcado, leyenda u overlay faltante, borroso, ilegible, duplicado, manchado. Screen en zonas prohibidas debido a problemas de diseño. El screen debe permanecer legibles antes y después del ensamble. Esto está presente en la IPC 600, IPC 610 e IPC 2221.

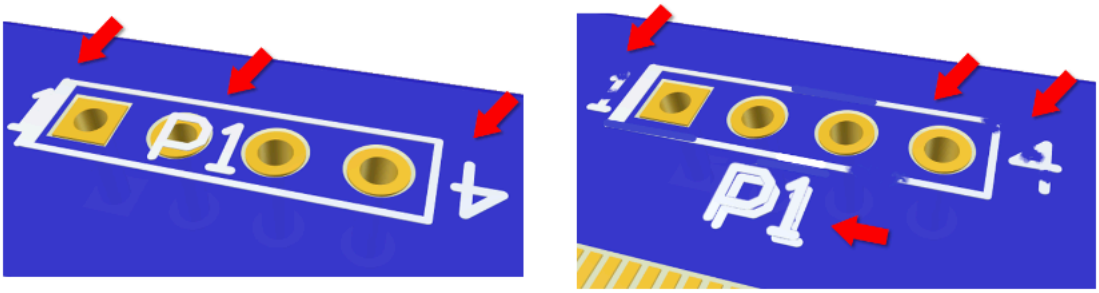


Ilustración 14 Marcas o serigrafía

2.2.1.10. Antisolder

Grietas, arrugas, **ampollas**, que pueden causar puentes de soldadura, partículas de máscara de soldadura, antisolder o solder mask desprendidas, que afectan el circuito.

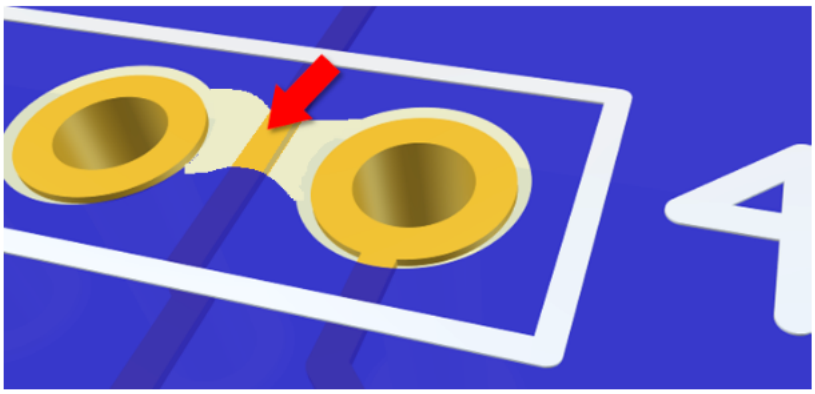


Ilustración 15 Antisolder

2.2.1.11. Residuos de flux

Defecto si se encuentran **residuos** de flux, tipo no clean o activado. Defecto si se encuentran residuos blancos. Residuos de flux activado RA o RMA que causa corrosión.

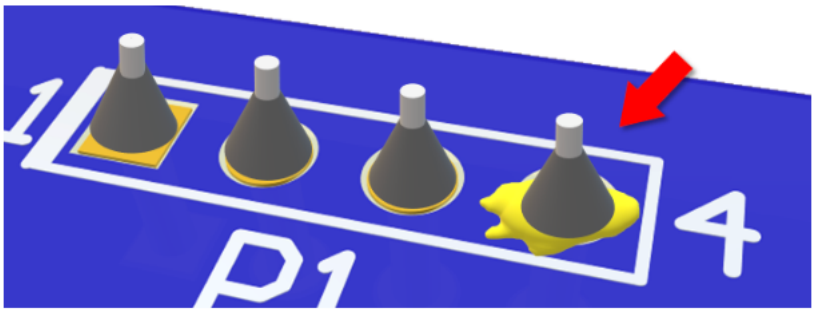


Ilustración 16 Residuos de flux

2.2.1.12. Restos de objetos extraños

Partículas extrañas, atrapadas en el antisolder o por fuera de él, residuos blancos en la **superficie** de la tarjeta o de una zona de **soldar**, que impiden la inspección o la conexión.

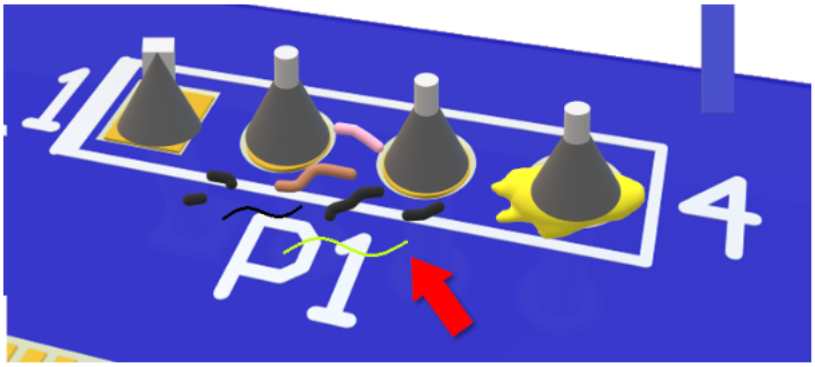


Ilustración 17 Restos de objetos extraños

2.2.1.13. Pandeo y torcedura

El pandeo y torceduras causan daños en el ensamble, la tensión generada por estos puede fracturar la conexión o dañar a los componentes. Puede dificultar el ensamble automatizado SMT, especialmente en componentes que requieren una superficie plana tipo QFP, QFN o BGA. Aquí es muy importante emplear la IPC 600.

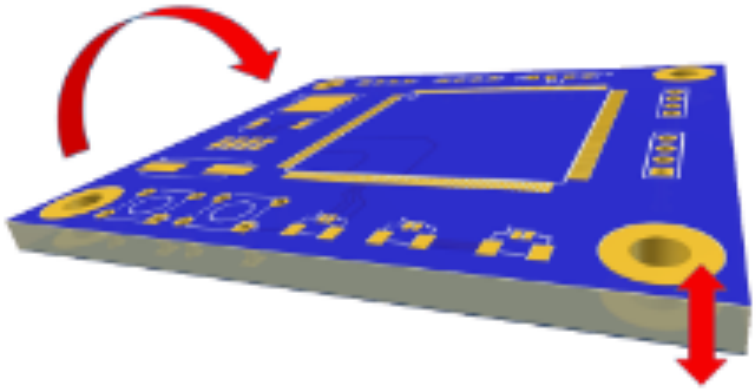


Ilustración 18 Pandeo y torcedura

2.3. MACHINE LEARNING

2.3.1. Definición

Machine Learning (ML) se puede entender como la capacidad de una máquina de encontrar un patrón o función adecuada para una aplicación determinada.

En (Murphy, 2012) define Machine Learning como algoritmo que puede detectar de forma automática patrones en el conjunto de datos de entrada y utilizar dichos patrones en nuevos datos para predecir nuevos resultados o tomar algún tipo de decisión según la aplicación.

En (Zhou, 2016) define el aprendizaje automático como una disciplina que se compromete a estudiar cómo usar los algoritmos a través de medios computacionales para mejorar el rendimiento del propio sistema.

2.3.2. Tipología

2.3.2.1. Aprendizaje supervisado

(Murphy, 2012) El ML se divide en dos categorías principales. En el predictivo o supervisado, donde el objetivo es determinar la relación entre los datos de entradas (x) con los datos de salida (y), dado conjuntos de datos (x, y) etiquetados. Se conocen tanto las variables predictoras como las variables objetivas.

$$D = \{(x_i, y_i)\}_{i=1}^{N} \qquad \text{Ecuación 1}$$

Donde D es el conjunto de datos de entrenamiento y N es el número de individuos, o número de datos.

En la configuración más simple, cada entrada de entrenamiento xi es un vector de números de D-dimensional, que representaría como la característica, atributos o covariables.
La variable de salida o respuestas, al igual que las variables de entrada, puede ser cualquier estructura compleja, sin embargo, la mayoría de las veces las variables de salida son categórico o nominal proveniente de un conjunto de finito o son escalares. En el caso de que la respuesta que se obtiene es una variable nominal, el problema que se enfrenta es de clasificación o patrones de reconocimiento; en el caso de que sea un escalar, se trata de un problema de regresión.

2.3.2.2. Aprendizaje no supervisado

También se conoce como Aprendizaje descriptivo, donde el objetivo es encontrar patrones interesantes en los datos, a veces conocido como "el descubrimiento de conocimientos" (Knowledge discovery).

$$D = \{(x_i)\}_{i=1}^{N} \quad \text{Ecuación 2}$$

2.3.2.3. Aprendizaje Reforzado

Hay un tercer tipo de aprendizaje automático, conocido como aprendizaje por refuerzo, que se usa con menos frecuencia. Esto es útil para aprender a actuar o comportarse cuando se le dan señales ocasionales de recompensa o castigo. Desafortunadamente, en este proyecto no se va a entrar en profundidad sobre este tema, ya que el objetivo principal es crear un sistema de detección de defectos mediante el Deep Learning.

2.4. RED NEURONAL (NEURAL NETWORK)

(Duan, 2018) Cuando se habla de Red Neuronal (RN) en ML se refiere al modelo computacional inspirado en la estructura y el funcionamiento de un cerebro humano.

La RN está formado por "neuronas", que funciona como las células nerviosas del cerebro, se encarga de procesar y transmitir información en 0 y 1.

En 1958, el informático Frank Rossenblatt propuso la primera regla de aprendizaje del perceptrón basada en el modelo M-P (McCulloch-Pitts neuron, MCP).
Su perceptrón constaba de dos capas de neuronas que formaban una red neuronal, una capa de entrada y otra de salida. Es la única red neuronal artificial del mundo que podía aprender y ya puede realizar un simple reconocimiento de imágenes.

Sin embargo, en 1969 el informático Marvin Minsky publicó un libro llamado "Perceptron", que detallaba las debilidades del perceptrón. El perceptrón de una sola capa es incapaz de completar muchas tareas simples, mientras que el perceptrón de doble capa limita la potencia de cálculo. Los requisitos eran altos y no había algoritmos de aprendizaje efectivos. Lo que llevó a la conclusión de que no tenía sentido estudiar redes neuronales más profundas, es decir, añadir más capas ocultas entre las capas de entrada y salida.

Pasado los 10 años, Geofey Hinton y otros propusieron el algoritmo de retro propagación (BP), que resolvió los complejos problemas de cálculo requeridos por la red neuronal de dos capas.

No obstante, lo que la gente nunca esperó es que, a mediados de la década de 1990, Apareció SVM, mostró capacidades poderosas, como la ausencia de necesidad de ajustar parámetros, mayor eficiencia, etc.

Durante esa época, Jeffrey Hinton varios otros académicos continuaron estudiando y le dieron al algoritmo de red neuronal multicapa un nuevo nombre: aprendizaje profundo (Deep Learning).

Actualmente, el aprendizaje profundo ocupa una posición dominante. Ya sea en el reconocimiento de imágenes, el reconocimiento de voz, el procesamiento del lenguaje natural, la conducción automática y otros campos, tiene una gama amplia de aplicaciones. Lo que se centra en este proyecto es el perceptrón multicapa (MLP) en redes neuronales.

Se define en (Zhou, 2016) "Una red neuronal es una red interconectada ampliamente paralela de unidades simples adaptables que está organizada para simular las respuestas interactivas de los sistemas nerviosos biológicos a objetos del mundo real".

2.4.1. Principio y corrección no lineal

Cuando se habla de un modelo lineal, se emplea la siguiente fórmula:

$$\hat{y} = w[0] * x[0] + w[1] * x[1] + \cdots + w[p] * x[p] + b \qquad \text{Ecuación 3}$$

Donde $\hat{y}$ representa el valor estimado de y, $x[0]$ y $x[p]$ son variables de entrada y w representa el peso de cada una de ellas. $\hat{y}$ puede considerarse como la suma ponderada de todos los valores propios.

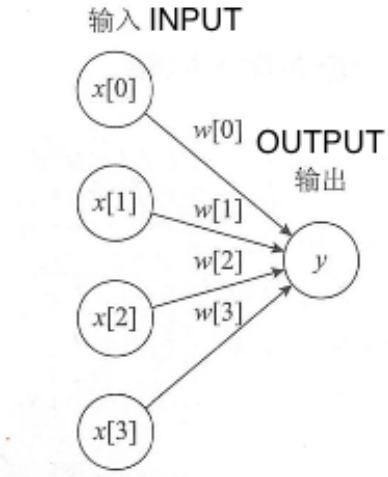


Ilustración 19 Regresión Lineal (Duan, 2018)

En Ilustración 19, las características de entrada y los resultados pronosticados están representados por nodos, y el vector de pesos, w, se utiliza para conectar estos nodos.

En el modelo MLP, el algoritmo agrega capas ocultas (Hidden Layers) en el proceso, las variables de entrada se combinan mediante pesos y se aplica una función de activación (h) sobre las propias combinaciones de las variables. Finalmente usa los resultados calculados de la capa oculta para generar el resultado final, como se muestra en la siguiente imagen:

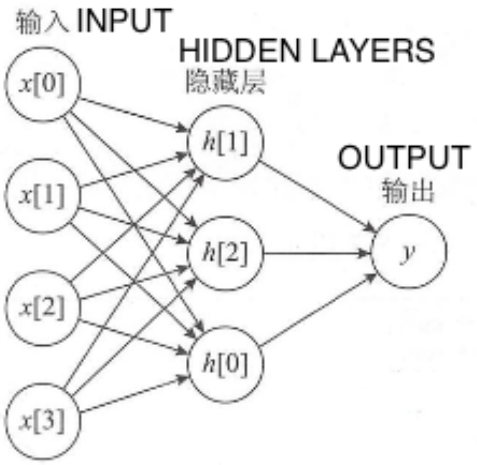


Ilustración 20 Modelo MLP con 1 Hidden layer (Duan, 2018)

De este modo, el modelo tendrá muchos más coeficientes de características o ponderaciones que aprender. Como puede observar, existe un coeficiente entre cada característica de entrada y la unidad oculta, es para generar estas unidades ocultas. También existe un coeficiente entre cada unidad oculta y el resultado final.

Desde un punto de vista matemático, si cada capa oculta solo realiza una suma ponderada, el resultado no será diferente del de un modelo lineal ordinario. Entonces, para que el modelo sea más potente que el modelo lineal ordinario, todavía necesitamos realizar un pequeño procesamiento.

Este procesamiento es conocido como la función de activación. Después de generar la capa oculta, se debe realizar una corrección (rectificación) no lineal del resultado, conocido como "relu" (unidad lineal rectificado) o un procesamiento tangente hiperbólico (tangens hyperbolicus), conocido como "tanh". Mediante estos dos tipos de procesamiento se procede a calcular el resultado final, y.

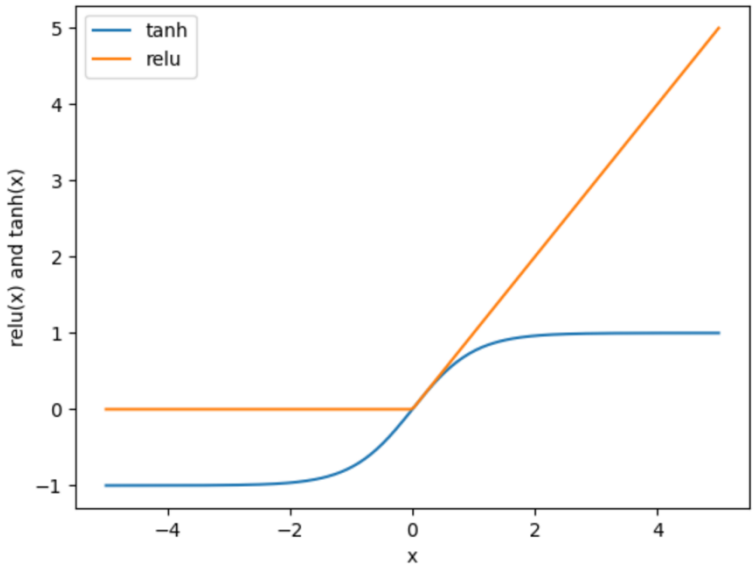


Ilustración 21 Procesamiento de atributos mediante tanh y relu

Se observa en la Ilustración 21, la función tanh comprime ha comprimido los valores de atributos entre -1 y 1, siendo -1 el valor más pequeño y 1 el valor más grande. Y la función relu directamente iguala los valores inferiores a 0 como 0.

Estos dos tipos de procesamiento no lineal simplifica los atributos de las muestras, con el fin de que la red neuronal pueda realizar aprendizaje sobre las muestras no lineales más complejas.

La Ecuación 3, una vez procesado por la función tanh, se convierte en la siguiente forma:

$$h[0] = \tanh\left(w[0] * x[0] + w[1] * x[1] + \cdots + w[p] * x[p] + b\right) \quad \text{Ecuación 4}$$

$$h[1] = \tanh\left(w[0] * x[0] + w[1] * x[1] + \cdots + w[p] * x[p] + b\right) \quad \text{Ecuación 5}$$

$$h[2] = \tanh\left(w[0] * x[0] + w[1] * x[1] + \cdots + w[p] * x[p] + b\right) \quad \text{Ecuación 6}$$

…

$$\hat{y} = v[0] * h[1] + v[1] * h[1] + \cdots + v[n] * h[n] \quad \text{Ecuación 7}$$

Además del coeficiente de peso w, tenemos otro coeficiente de peso v, que se utiliza para calcular el resultado de y-hat a través de la capa oculta h.

En el modelo, w y v se obtienen aprendiendo de los datos. El parámetro que el usuario debe configurar es la cantidad de nodos en la capa oculta. En términos generales, para conjuntos de datos a pequeña escala o conjuntos de datos simples, es suficiente establecer el número de nodos en 10. Sin embargo, para conjuntos de datos a gran escala o conjuntos de datos complejos, hay dos formas para elegir: una es

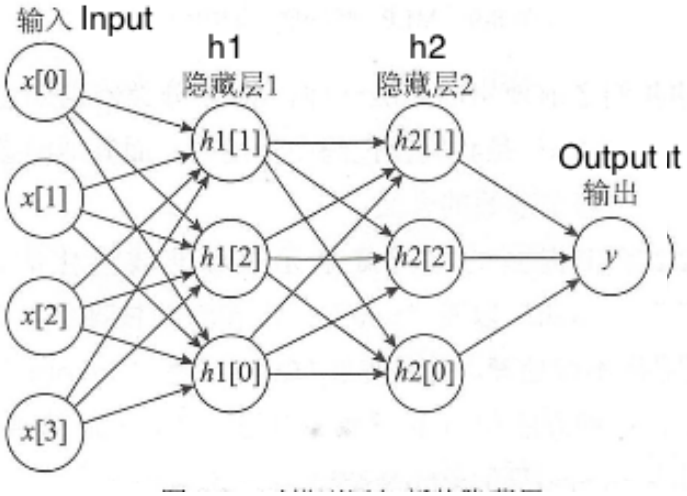


Ilustración 22 Agregar nuevas capas ocultas

aumentar el número de nodos ocultos o se agregan más capas ocultas, como se muestra en la Ilustración 22.

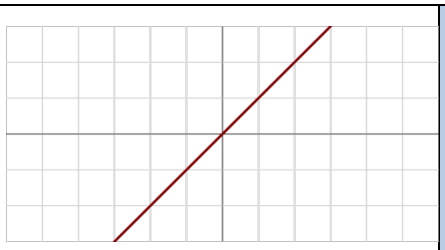
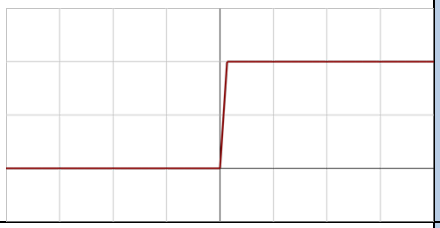
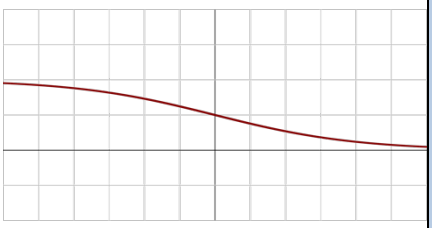
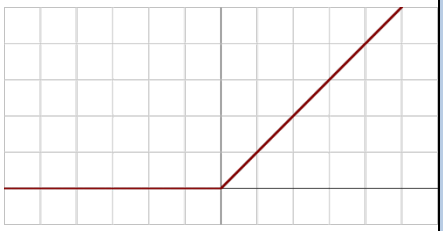
2.4.2. Funciones de activación

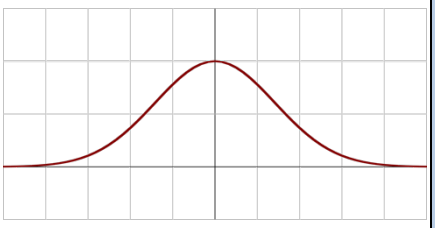
En (Tech, 2024) define la función de activación como una función que transmite la información generada por la combinación lineal de los pesos y las entradas. Actúa como un filtro a la salida de un nodo con función limitadora o umbral, que modifica el valor resultado o impone un límite que se debe sobrepasar para proseguir el nodo contiguo.

Las funciones de activación desempeñan un papel fundamental en la red neuronal, dado que permite que la relación de entrada y salida sea no lineal, lo cual posibilita la resolución de problemas más complejos.

Entre las funciones de activación más conocidas o usadas se encuentran:

Tabla 1 Funciones de activación

Identity		$g(x) = x$
Binary Step		$g(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$
Sigmoid		$\sigma(x) \doteq \frac{1}{1+e^{-x}}$
Tanh		$tanh(x) \doteq \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$
ReLu		$x^+ \doteq \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases} = max\,(0, x) = x1_{x>0}$

Gaussian		$g(x) = e^{-x^2}$
Softmax	$g_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \quad for\ i = 1, \dots, J$	
Maxout	$g_i(\vec{x}) = max_i x_i$	

2.4.3. Retro propagación (BackPropagation)

Es un algoritmo utilizado en las redes neuronales artificiales supervisadas para ajustar los pesos de las conexiones entre las neuronas, con el fin de reducir el error en la predicción de un modelo.

El algoritmo funciona propagando el error hacia atrás a través de la red neuronal, comenzando por la capa de salida y retrocediendo hacia las capas ocultas. Para cada capa, se calcula la contribución relativa de cada neurona a la función de costo, y se utilizan estas contribuciones para ajustar los pesos de las conexiones.

Se basa en la regla de la cadena de la derivada, que permite obtener la estimación del gradiente del vector peso (w). En el contexto de la retropropagación, la regla de la cadena se utiliza para calcular la contribución de cada nodo a la función de costo, en función de sus entradas y de los pesos de las conexiones que la conectan con las capas siguientes. El objetivo de todo esto es estimar el vector de pesos para la optimización de la función de pérdida (Loss) o error.

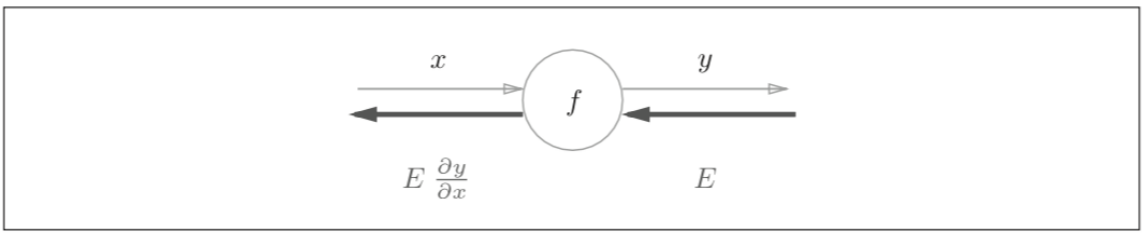


Ilustración 23 BackPropagation (斎藤康毅, 2018)

Como se observa en la Ilustración 23, la secuencia de cálculo para la propagación hacia atrás es multiplicar la señal E por la derivada local del nodo y luego pasar el resultado al siguiente nodo. La derivada local mencionada se refiere a la derivada de la función de la propagación directa

$$\frac{\partial y}{\partial x};\ siendo\ y\ = f(x)$$ Ecuación 8

Para explicar este proceso con más detalles se procede a la introducción de la regala de la cadena.

La regla, (斎藤康毅, 2018), dice lo siguiente:

"Si una función está representada como una función compuesta, la derivada de la función compuesta puede ser expresada como el producto de las derivadas de las funciones individuales de la función compuesta."

Matemáticamente dicha es así:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t} * \frac{\partial t}{\partial x}$$ Ecuación 9

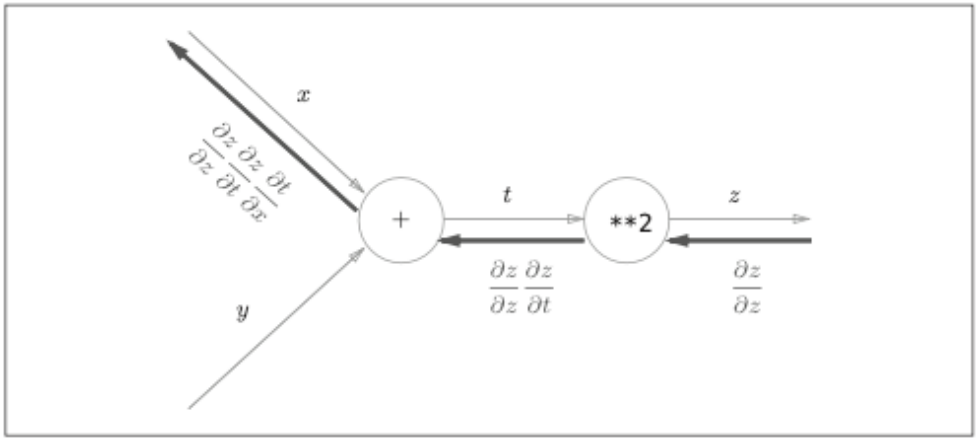


Ilustración 24 Regla de la cadena (斎藤康毅, 2018)

Como se observa en la Ilustración 24, el orden de la propagación hacia atrás es de derecha hacia izquierda. Multiplicar la señal de entrada del primer nodo partir de la derecha por la derivada local del mismo nodo y transmitirlo al siguiente nodo. En el ejemplo de la Ilustración 24, se observa que:

Durante el proceso de retropropagación:

La entrada del input "**2" es

$$\frac{\partial z}{\partial z}$$ Ecuación 10

La derivada local del mismo es

$$\frac{\partial z}{\partial t}$$ Ecuación 11

El resultado de la retropropagación en el nodo "**2" para x es

$$\frac{\partial z}{\partial z} * \frac{\partial z}{\partial t} \rightarrow \frac{\partial z}{\partial t}$$ Ecuación 12

La derivada en el nodo +

$$\frac{\partial t}{\partial x}$$ Ecuación 13

El resultado final de la retropropagación es:

$$\frac{\partial z}{\partial t} * \frac{\partial t}{\partial x} \rightarrow \frac{\partial z}{\partial x}$$ Ecuación 14

Existe otros ejemplos de retropropagación como el de la suma o el de la multiplicación.

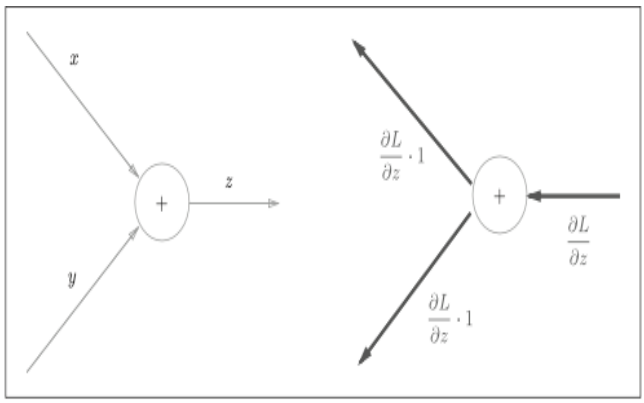


Ilustración 26 BP en suma (斎藤康毅, 2018)

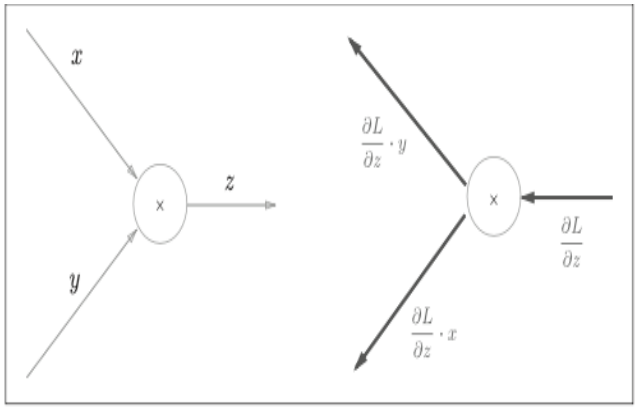


Ilustración 25 BP en multiplicación (斎藤康毅, 2018)

2.5. DEEP LEARNING (APRENDIZAJE PROFUNDO)

Se denomina Aprendizaje profundo a las redes neuronales multicapas. Según (Zhou, 2016), cuando mayor es el número de parámetros de un modelo, mayor será su complejidad y capacidad de aprender. Esto implica que podrá realizar tareas complejas.

Sin embargo, a la hora de entrenar el modelo complejo, el proceso es ineficiente y se puede sobre ajustar (Overfitting) el modelo.

Actualmente, en la era de Big Data, el aumento de datos para el entrenamiento puede mejorar la ineficiencia del entrenamiento y reducir el sobreajuste del modelo, por lo que el aprendizaje profundo (DL) vuelve a tomar importancia.

Una forma sencilla de aumentar la capacidad de aprendizaje en un modelo de aprendizaje profundo es aumentar el número de capas ocultas. O también aumentando los nodos de la capa oculta. Con el aumento de capas ocultas o nodos, aumenta los parámetros que tienen el modelo, por lo que aumenta la capacidad de aprendizaje.

Ahora bien, el aumento de números de capas ocultas es más efectivo para el modelo que es aumento de nodos, porque el aumento de capas, además de generar más capas que contienen funciones de activación, aumenta también los nodos de función de activación.
No obstante, es difícil utilizar el algoritmo BP (Backpropagation) para entrenar el modelo del DL, ya que a menudo los errores durante el entrenamiento se divergen y no vuelven a un estado estable.

El entrenamiento por capas no supervisado (Unsupervised layer-wise training) es un método eficaz para entrenar modelos de aprendizaje profundo. La idea básica es que cada vez que entrene una capa oculta, utiliza la salida de la capa anterior como entrada en la siguiente, esto también se conoce como pre-entrenamiento o pre-training; una vez que se haya completado el entrenamiento previo, se realiza un entrenamiento de "ajuste fino" al modelo.

Más aun, mediante el método de entrenamiento previo y ajuste fino se clasifican la gran mayoría de los parámetros, buscando en primer lugar las mejores configuraciones localmente para cada grupo y luego realiza conjuntamente una optimización global basada en las configuraciones anteriores. De modo que se aprovechan los grados de libertad proporcionados por los parámetros del modelo y al mismo tiempo reducen la sobrecarga de entrenamiento.

Otra estrategia para reducir la sobrecarga es compartir el peso (w) (weight sharing), dejar que un grupo de nodos utilice los mismos pesos de conexión. Esto juega un papel importante en la red neuronal convolucional (Convolutional Neural Network - CNN)

Por lo que, se puede entender DL como un conjunto de algoritmos de aprendizaje automático (Machine Learning) que mediante el mecanismo de apilar capas y usar la salida de la capa anterior procesada como entrada, conecta la entrada inicial con el objetivo de salida; en otras palabras, transforman los atributos de bajo nivel en alto nivel para resolver problemas complejos con un modelo sencillo. Por lo que también se conoce DL como aprendizaje de atributos o representación.

Las arquitecturas más populares son Deep Neural Network (DNN), Deep Belief Network (DBN), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) y Transformer.

2.5.1. CNN

La red convolucional, también llamada red neuronal convolucional (CNN), es una red neuronal especialmente diseñada para procesar datos con una estructura cuadricular, sobre todo aquellos que presentan dependencias locales espaciales. Así como indica (Aguilar Margalejo, 2023), las imágenes se consideran como matrices bidimensionales o también denominado como "mapas de características".

La arquitectura básica de CNN está estructurada en 3 capas:

- Capa de convolución
- Capa de agrupación (Pooling)
- Capa de salida (Fully connected)

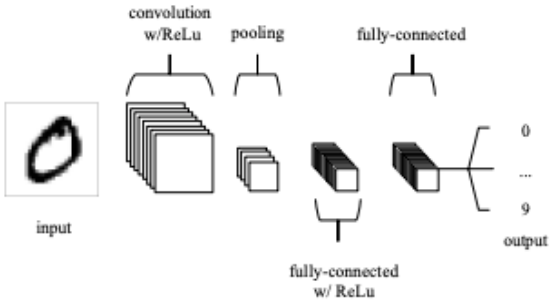


Ilustración 27 Arquitectura de CNN (O'Shea & Nash, 2015)

La capa de convolución se encarga de extraer características de la imagen, mientras que la capa de agrupación se encargar de reducir la dimensionalidad de la imagen. Tras los procesamientos en las capas anteriores obtenemos una matriz bidimensional reducida, que conserva las características de la imagen, y permite que la capa de salida pueda computar los valores producidos por las funciones de activación y clasificarlos.

Capa de convolución

El proceso de convolución utilizada en esta capa también es conocido como correlación cruzada.

En primer lugar, se define el tamaño del núcleo de convolución (la ventana del filtro) que se situará en la parte superior izquierda. El núcleo de convolución se desplaza progresivamente de izquierda a derecha un determinado número de casillas definido por el "stride" hasta llegar al final de la imagen. Sobre cada proporción escaneada se realiza el cálculo de convolución, obteniendo de esta forma un mapa de características.

Se observa que la imagen después del proceso de convolución se pierde información, y eso se debe a que el núcleo de convolución es incapaz de desplazar fuera de los bordes de la imagen o que los bordes permanezcan en el centro del núcleo de convolución. La solución consiste en incorporar información en los bordes con el fin de incrementar la dimensionalidad de la imagen y esta técnica es conocida como Padding.

Las técnicas de Padding más empleadas son:

1. Valid Padding: no se realizará ningún procesamiento sobre la imagen.
2. Same Padding: Se llevará a cabo el procedimiento de relleno con el fin de permitir que el núcleo de convolución supere los límites de la imagen, lo que lleva a la misma dimensionalidad en la salida que en la entrada.

Otro parámetro importante es el paso (stride), es el número de casilla que se desplaza en cada convolución. La función de stride es reducir la dimensionalidad de la imagen. El valor de este parámetro es el múltiplo específico de división, es decir, en caso de que la división sea de 2, las dimensiones de la imagen se reducen a mitad (1/2).

Hay que tener en cuenta que una imagen está compuesta por RGB tres colores, cada color es un canal el proceso de convolución y se expresa normalmente como:

$$(M * N) * n^{o}canal \qquad \text{Ecuación 15}$$

La convolución introduce 3 ideas fundamentales para mejorar el sistema de aprendizaje de Machine Learning:

1. Interacciones dispersas (sparse interaction): esto se logra haciendo que el tamaño del kernel sea mucho más pequeño que el tamaño de entrada.
2. Compartición de parámetros (parameter sharing): Se refiere al uso de los mismos parámetros en múltiples funciones de un modelo.

3. Representaciones equivalentes (equivariant representation): Se dice que una función es equivariante si su salida cambia de la misma manera que cambia su entrada.

Capa de agrupación (Pooling)

Esta capa recibe como entrada el mapa de características generada por la capa de convolución y se reduce las dimensiones de la imagen y se conserva las características esenciales de la imagen.

Los métodos de submuestreos más empleados son:

- Max-Pooling: toma el valor máximo del núcleo de convolución.
- Average Pooling: toma el valor promedio del núcleo de convolución.

Capa de activación ReLU (Unidades lineales rectificados)

Esta capa tiene la función de sustituir todos los valores de entrada negativos por ceros, haciendo que el modelo sea no lineal.

Capa de salida (Fully connected)

Se conecta todas las neuronas de la salida y después de haber recibido un vector en la entrada, se aplica una combinación lineal y una función de activación con el fin de clasificar la imagen, el resultado es un vector de tamaño correspondiente al número de clases.(Daniel, 2021)

2.5.1.1. VGGnet

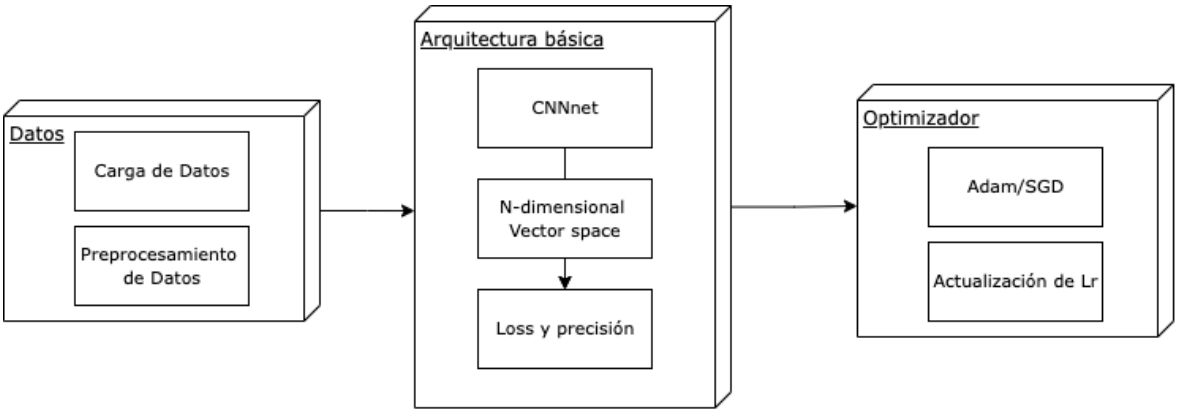


Ilustración 28 Bloques de diseño del entrenamiento del sistema

En (斎藤康毅, 2018) describe VGG como una CNN básica compuesta por capas convolucionales y capas de Pooling. Sin embargo, como se muestra en la Ilustración 29, se destaca en superponer capas ponderadas (capas convolucionales o capas completamente conectadas) a 16 capas (o 19 capas), lo que tiene profundidad (según la profundidad de la capa, a veces se le llama es "VGG16" o "VGG19").

Lo que hay que tener en cuenta en VGG es que la operación de la capa convolucional basada en el filtro pequeño de 3 × 3 se realiza de forma continua. Como se muestra en la Ilustración 29, se repite el proceso de "superponer la capa convolucional de 2 a 4 veces y luego reducir el tamaño a la mitad a través de la capa de Pooling y, finalmente, el resultado se genera a través de la capa completamente conectada.

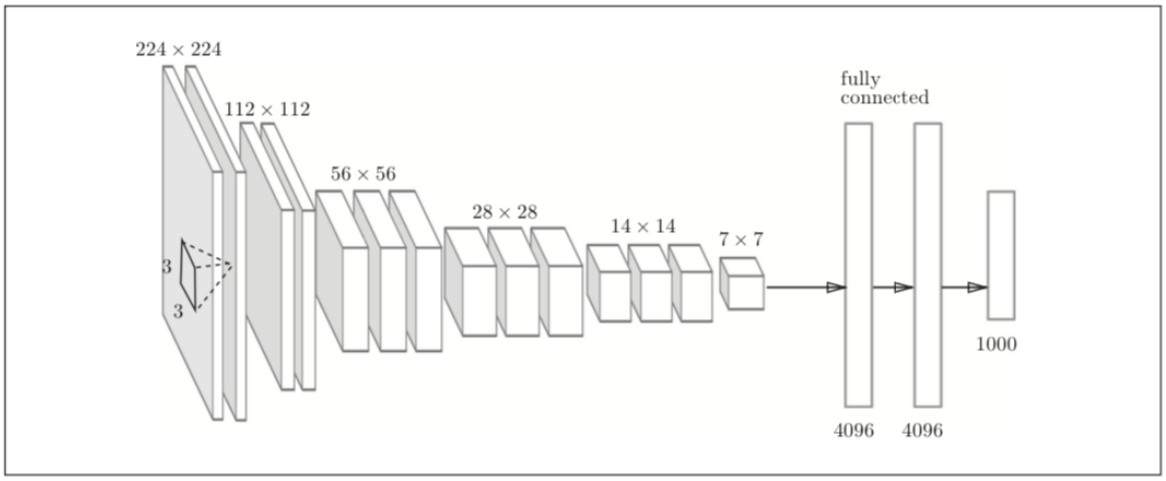


Ilustración 29 Estructura de VGGnet

2.5.1.2. Parámetros del VGGnet

- Profundidad: la red VGG tiene 16 capas como configuración predeterminada, divididas en 5 bloques. Existen redes más profundas como VGG-19 que es capaz de extraer más características de los datos.
- Tamaño del núcleo: 3*3. Es efectivo para capturar características locales en las imágenes.
- Número de canales: cuantos más canales presenten en el modelo, mejor se aprecian las características.
- Stride: operación básica que existe en CNN con el objetivo de reducir el tamaño del mapa de características después de una convolución. Cuando stride es 1, significa que reserva toda la resolución espacial de características, mientras que stride = 2, implica reducir la resolución espacial y aumentar la eficiencia computacional.
- Capa de Maxpooling: es una operación de agrupación que calcula el valor máximo para parches de un mapa de características y lo usa para crear un mapa de características reducido (agrupado). Reduce la resolución espacial y mejora la robustez de las características.
- Funciones de activación: explicado en Tabla 1
- Canales de salida: la red VGG utiliza la última capa totalmente conectada para realizar la clasificación de imágenes, el número de neuronas de la capa final depende del número de clases que existan.

2.5.2. Visión computacional

La visión artificial es un campo de la inteligencia artificial (IA) que permite a los ordenadores y sistemas extraer información significativa a partir de imágenes digitales, videos y otras entradas visuales, y tomar medidas o realizar recomendaciones en función de esa información.
(What is Computer Vision?, 2023)

El principio de funcionamiento consiste en que los modelos algorítmicos de Machine Learning enseñan al ordenador el contexto de los datos visuales, así que cuantas más muestras se introducen para el entrenamiento, será más rápido y preciso el dispositivo de reconocer y diferenciar una imagen de otra.

Las redes convolucionales que componen el modelo de Deep Learning desglosan las imágenes en píxeles y les asignan etiquetas. Utilizan estas etiquetas para realizar convoluciones y predicciones sobre las imágenes. De modo que estos modelos podrán identificar patrones

comunes en las muestras y aplicar esta información para predecir sobre nuevas imágenes.

2.5.2.1. Aplicación: Detección objetos

La detección de objetos consiste en buscar los objetos interesados en la entrada de imagen, determinando la clase y su posición.

Existe cuatro categorías principales de tareas de reconocimiento de imágenes en la detección de objetos:

- Clasificación: determinar la clase del objeto
- Localización: determinar la posición del objeto requerido.
- Detección: determinar la posición y clase del objeto
- Segmentación: determinar a qué clase (Instance-level) o escena (Scene-level) pertenece cada píxel.

Por lo que se podría decir que la detección de objeto es un problema conjunto de clasificación y regresión.

Los problemas claves de esta área de aplicación están relacionados con las categorías nombradas anteriormente:

- Problema de clasificación
- Problema de posición
- Problema de dimensión del objeto en la imagen
- Problema de forma del objeto.

Los algoritmos de detección de objetos basados en aprendizaje profundo se clasifican principalmente en dos categorías: Two Stage y One Stage.

- Two Stage: se registra previamente el conjunto de muestras entrantes con anotaciones, después se genera el área (Region Proposal-RP), un cuadro delimitador que podría contener el objeto, y finalmente se procesa mediante una red convolucional para realizar la clasificación de muestras.
  Los algoritmos más conocidos son: R-CNN, SPP-Net, Fast R-CNN, etc.

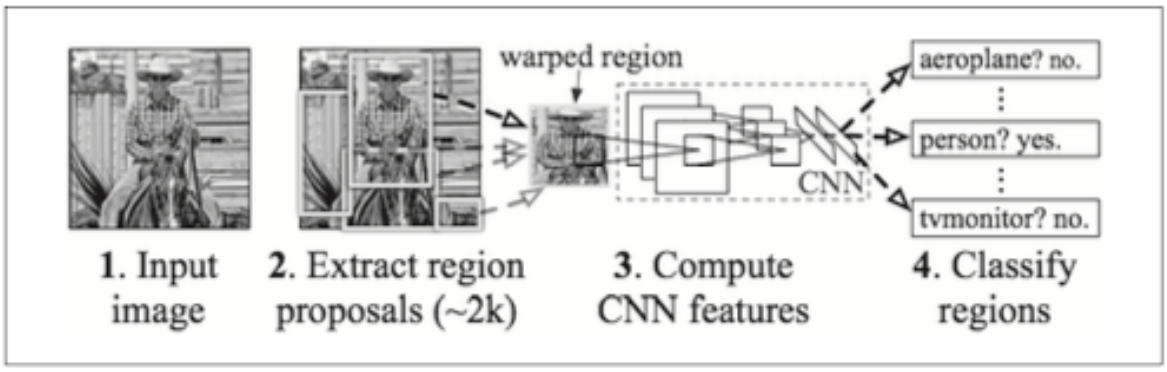


Ilustración 30 Arquitectua R-CNN

Primero se deben encontrar las áreas de imagen donde aparezca el objeto (extraer propuesta de región) y aplicar CNN (calcular atributos de CNN) a las áreas extraídas para clasificarla.

Para que el CNN se adapte a la aplicación, se debe realizar un entrenamiento previo y evaluarlo con una etapa de validación para asegurar el correcto funcionamiento. Para ello, se deben seguir los siguientes pasos:

1. Obtención de muestras
2. Preprocesamiento de las muestras
3. Extracción de atributos
4. Segmentación de la imagen
5. Procesamiento avanzado (Si es necesario)
6. Entrenar el modelo con las muestras procesadas
7. Comprobar la precisión del modelo
8. Optimización del modelo (si es necesario)

- One Stage: no emplea RP, se extrae directamente mediante la red las características y se realiza la clasificación de la muestra.
  Los algoritmos más conocidos son: OverFeat, YOLOv1, YOLOv2, YOLOv3, SSD, RetinaNet, etc.

2.5.2.2. YOLO

(YEGE, 2020) YOLO (You Only Look Once) es otro marco propuesto por Ross Girshick para abordar el problema de la velocidad de detección de objetos después de RCNN, RCNN rápido y RCNN más rápido.
Su idea central es reemplazar el algoritmo de Two Stage, donde se genera RoI y la detección de objetos por un algoritmo de una etapa de un conjunto de redes, que devuelve directamente la posición y categoría del cuadro delimitador en la capa de salida.

Los métodos tradicionales de detección de objetos generaban una gran cantidad de cuadros delimitadores mediante anotaciones a las posibles localizaciones del objeto, después se utiliza un clasificador que determina si los cuadros delimitadores generados por el modelo contienen el objeto a determinar y la probabilidad de que el objeto pertenezca a la clase correspondiente. Al mismo tiempo, se necesita un posterior procesamiento que corrige los cuadros delimitadores y filtran las que son de baja fiabilidad. Estos métodos generan unos resultados de alta precisión, pero su velocidad de ejecución es lenta.

Por lo que YOLO propone una nueva manera de visualizar este tipo de problemas, lo trata como un problema de regresión, combinando las dos etapas (Region Proposal y la detección) en una. De modo que pueda visualizar la clase del objeto y su correspondiente posición en la muestra.

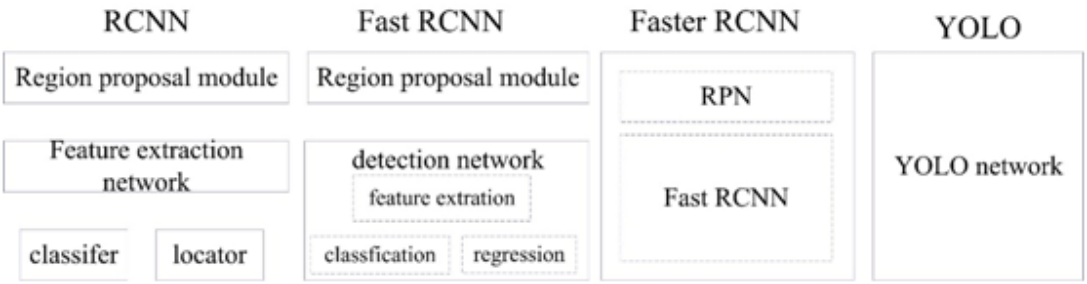


Ilustración 31 Diagrama de flujo de cada sistema de detección

Sin embargo, YOLO no elimina la región propuesta (RP), sino que lo predefine, es decir, divide la imagen en cuadrículas de 7*7. Cada cuadrícula permite predecir 2 bordes; un total de 49*2 cuadros delimitadores o también 98 regiones propuestas que cubre aproximadamente toda la imagen. De este modo, YOLO mejora la eficiencia del tiempo a expensas de reducir mAP (precisión).

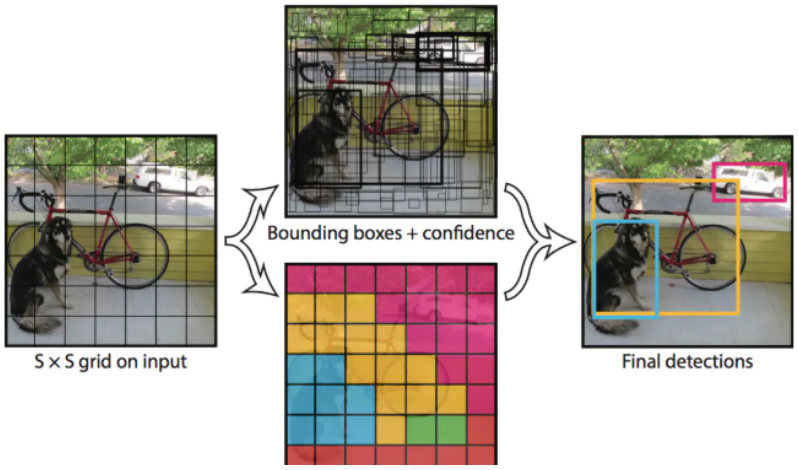


Ilustración 32 Algoritmo de YOLO

Cada celda de la cuadrícula predice 2 cuadros delimitadores y unas puntuaciones para dichos cuadros. Estas puntuaciones reflejan la fiabilidad del modelo acerca de si la caja contiene el objeto y con qué precisión predice la caja. La confianza se define como:

$$Conf = Pr(Object) * IOU_{truth}^{pred}$$ Ecuación 16

donde Pr(Objeto) es la probabilidad de que haya un objeto en la celda, e IoU es lo que se conoce como intersección sobre la unión, es la relación que existe entre el área de superposición y el área total de unión de ambos cuadros delimitadores.

$$IoU = \frac{Área\ de\ la\ superposición}{Área\ de\ la\ unión}$$ Ecuación 17

IoU se utiliza para evaluar el rendimiento de la detección de objetos mediante la comparación del cuadro delimitador anotado (ground-truth) al predicho.

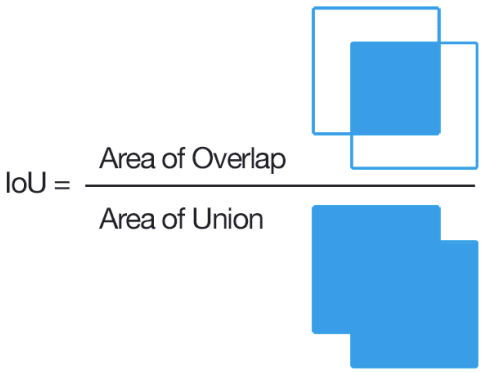


Figure 1 Representación gráfica de la ecuación

En el caso de que en la celda no está el objetivo, la puntuación de confianza debe ser cero. De lo contrario, se necesitaría que la puntuación de confianza sea igual a la intersección de partes de unión (IOU) entre el cuadro predicho y el valor de verdad fundamental (ground-truth).

Cada cuadro delimitador contiene 5 predicciones: x, y, el ancho (w), la altura (h) y la confianza.
La coordenada (x, y) representa el centro del cuadro delimitador en relación con el cuadro delimitador de la celda de la cuadrícula.
El ancho (w) y el alto (h) se predicen en relación con toda la imagen.
Finalmente, la predicción de confianza representa el IoU entre el cuadro predicho y el cuadro delimitador referente (grounth-truth).

Cada celda de la cuadrícula también predice C probabilidades de clase condicional a que la celda de la cuadrícula contenga el objeto: $PR(\ Clase_i|\ Objeto\ )\ con\ i = 1, \dots, C$. Solo se predecirá un conjunto de probabilidades de clase para cada celda de la cuadrícula, independientemente del número de cuadros delimitadores B.

En el momento de aplicación de CNN, multiplicará las probabilidades condicionadas y las predicciones individuales de confianza de la caja, de modo que se obtendrán índices de confianza específicos de clases para cada caja. Estos índices representan la probabilidad de la localización del objeto en el cuadro delimitador y la precisión del ajuste de la caja predicha sobre el objeto.

2.5.2.2.1. Arquitectura

La red YOLOv1 tiene 24 capas convolucionales, seguidas de 2 capas completamente conectadas. En este caso solo se utilizará una capa de reducción de dimensionalidad 1 × 1, seguida de una capa convolucional 3 × 3 como se muestra abajo:

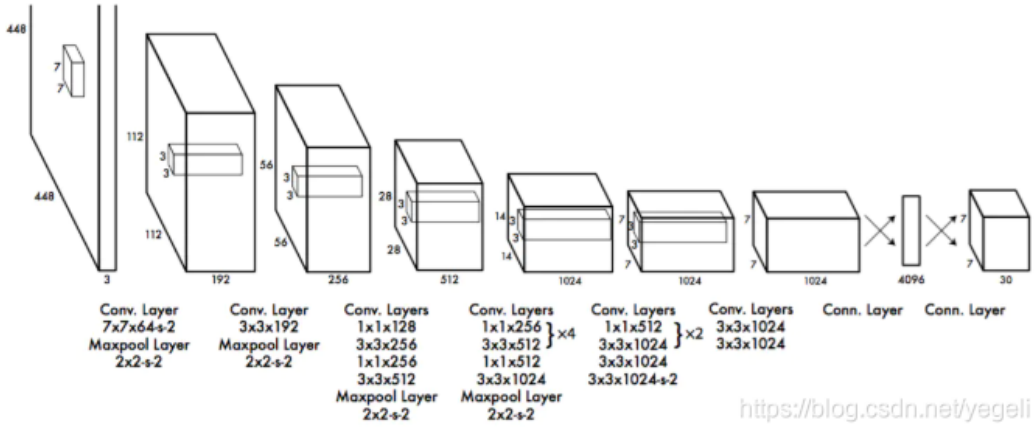


Ilustración 33 Arquitectura de la red YOLO

Para lograr rápidamente una detección rápida de objetos, YOLOV1 también entrena una versión rápida. Fast YOLO utiliza una red neuronal con menos capas convolucionales (9 en lugar de 24) y utiliza menos filtros en estas capas. Excepto por el tamaño de la red, todos los parámetros de entrenamiento y validación de YOLO y YOLO rápido son los mismos. La salida final de la red es un tensor de predicción de 7*7*30 (1470).

(Glenn, 2023) Actualmente la nueva versión de YOLO, YOLO-NAS desarrollado por Deci AI es un modelo innovador para la detección de objetos. Es producto de una tecnología avanzada de búsqueda de arquitectura neuronal y ha sido diseñado para resolver las limitaciones de los modelos YOLO anteriores. Con importantes mejoras en el soporte de cuantificación y compensaciones entre precisión y latencia.

YOLO-NAS utiliza bloques sensibles a la cuantificación y tecnología de cuantificación selectiva para un rendimiento óptimo. Este modelo muestra una pérdida mínima de precisión cuando se convierte a la versión cuantificada INT8, lo que supone una mejora significativa con respecto a otros modelos.

Las funciones innovadoras que se encuentran en este modelo son:

- Módulo básico que facilita la cuantificación: YOLO-NAS introduce un nuevo módulo básico que facilita la cuantificación, solucionando una limitación importante del modelo YOLO anterior.
- Entrenamiento y cuantificación avanzados: YOLO-NAS utiliza esquemas de entrenamiento avanzados y cuantificación posterior al entrenamiento para mejorar el rendimiento.
- Optimización y preentrenamiento de AutoNAC: YOLO-NAS adopta la tecnología de optimización AutoNAC y está preentrenado en conjuntos de datos famosos como COCO, Objects365 y Roboflow 100. Esta capacitación previa lo hace ideal para tareas posteriores de detección de objetos en entornos de producción.

2.5.2.3. R-CNN y Faster R-CNN

R-CNN (Regions with CNN features) es el algoritmo de primera generación de la serie R-CNN. Combina el conocimiento de "Aprendizaje profundo" y "visión por computadora" tradicional.

En el libro de (Shanmugamani & Moore, 2018), habla que RCNN fue propuesta por Girshick junto con otros, utilizan un método externo que propone varias áreas candidatas y comprueba si alguna de estas áreas coincide con valor verdadero.

Un ejemplo de estos métodos externo de propuestas de región es la búsqueda selectiva. La búsqueda selectiva propone regiones agrupando colores/texturas en ventanas de varios tamaños, buscando estructuras similares a manchas. Comienza con un píxel y produce una mancha a mayor escala. Generó aproximadamente 2.000 propuestas regionales.

Estas regiones propuestas se reajustan de tamaño para poder pasar a través de arquitecturas CNN estándar. La última capa de CNN se entrena utilizando SVM para identificar objetos que no tienen clases. El límite se mejora aún más ajustando el cuadro delimitador alrededor de la imagen.

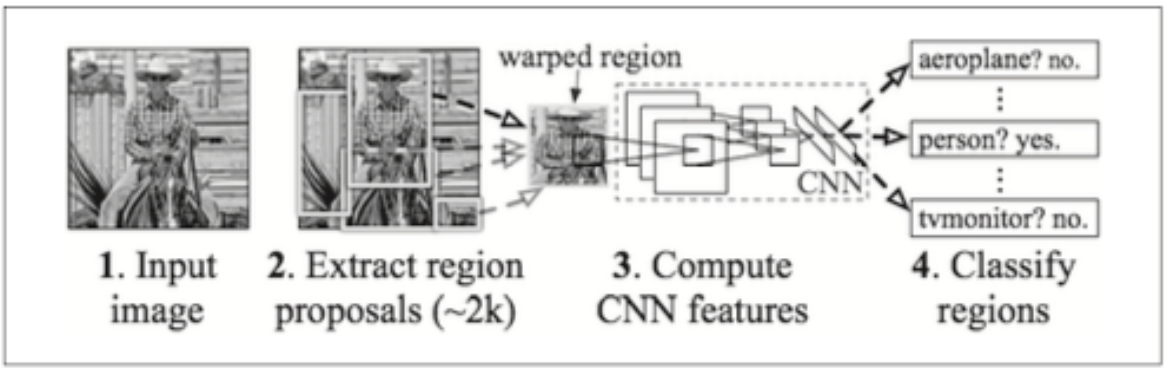


Ilustración 34 Arquitectua R-CNN

El codificador puede ser un modelo previamente entrenado de un modelo estándar de aprendizaje profundo.

En función de las características calculadas de los datos de entrenamiento. Se almacenan estas y se entrena el SVM. A continuación, las coordenadas normalizadas se utilizan para entrenar cuadros delimitadores. Puede haber algunas sugerencias fuera de las coordenadas de la imagen, por lo que el entrenamiento y la inferencia están normalizados.

Las desventajas de este algoritmo son:

- Si las regiones propuestas se generan mediante búsquedas selectivas, es necesario calcular muchas inferencias (del orden de 2000)

- Se debe entrenar los tres clasificadores, lo que aumentan el número de parámetros
- No existe entrenamiento de tipo End to End

Para mejorar el RCNN, Girschick junto con otros propuso el nuevo método de Fast-RCNN, que ejecuta la inferencia de CNN una sola vez y lo que consigue es reducir la cantidad de cálculos. Los datos obtenidos de la salida de CNN se emplean como parámetros de selección de la red y los cuadros delimitadores.

En este nuevo método se introduce una nueva tecnología llamada "Agrupación de regiones de interés", que agrupa las características de una CNN en función de las regiones. Después de la inferencia del CNN, las características obtenido mediante la agrupación se recolectan y eligen la región.

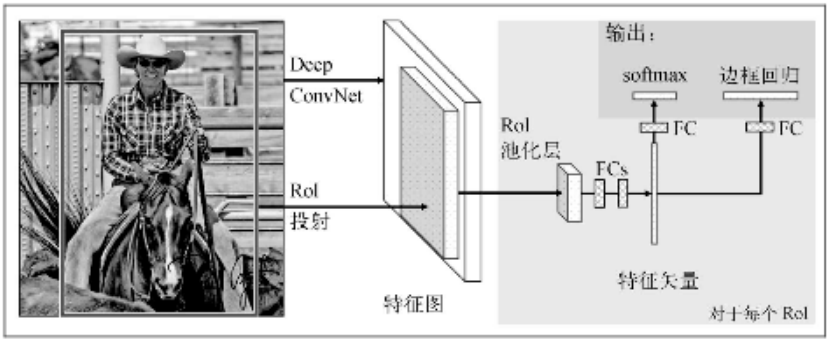


Ilustración 35 Arquitectura Fast-RCNN

De modo que se podrá realizar un entrenamiento de tipo extremo a extremo, evitando el uso de múltiples clasificadores. El SVM será reemplazada por una capa de softmax, y el regresor del cuadro por regresor del cuadro delimitador. Este método tiene la desventaja de la búsqueda selectiva, que requiere determinado tiempo de ejecución.

2.5.2.3.1. Faster R-CNN

En (Ren et al., 2016a) propusieron Faster R-CNN. La diferencia entre los métodos Faster R-CNN y Fast R-CNN es que el primero utiliza CNN con arquitecturas como VGG e Inception en lugar de búsqueda selectiva. Las características de CNN son procesadas posteriormente por una red de propuesta regional.

Estructuralmente, Faster RCNN ha incluido extracción de características, extracción de propuestas, regresión de cuadro delimitador, clasificación integrada en una red, lo que mejora significativamente el rendimiento general, especialmente la velocidad de detección.

1. Capas convolucionales: primero utiliza un conjunto de capas básicas conv + relu + pooling para extraer mapas de características de la imagen. Este mapa de características se comparte para capas RPN posteriores y capas completamente conectadas.
2. Redes de Propuestas Regionales (RPN): La red RPN se utiliza para generar propuestas regionales. Esta capa usa softmax para determinar si los anclajes son positivos o negativos, y luego usa la regresión del cuadro delimitador para corregir los anclajes y obtener propuestas precisas.

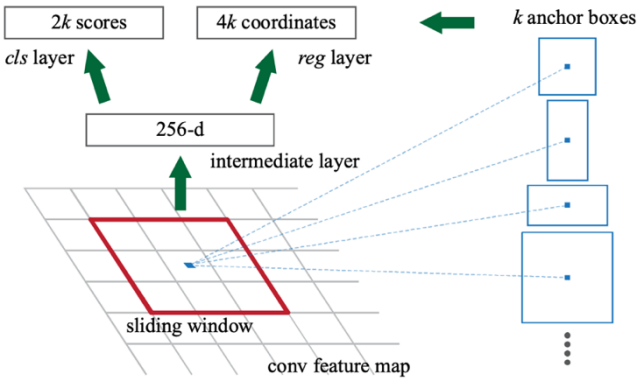


Ilustración 36 Region Proposal Network

3. Agrupación de ROI: esta capa recopila las propuestas y los mapas de características de entrada, sintetiza esta información, extrae los mapas de características de la propuesta y los envía a la capa posterior completamente conectada para determinar la categoría de destino.
4. Clasificación: utiliza mapas de características de la propuesta para calcular la categoría de la propuesta y, al mismo tiempo, vuelve a calcular la regresión del cuadro delimitador para obtener la posición final precisa del marco de detección.

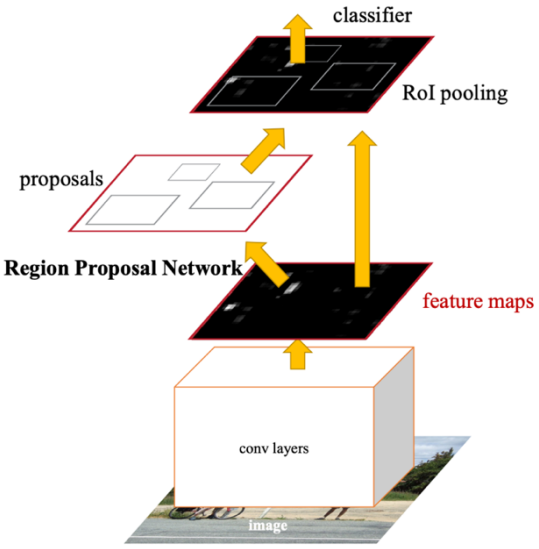


Ilustración 37 Arquitectura Faster R-CNN

2.5.2.4. Parámetros importantes

1. Parámetros de entrenamiento:
   - Batchsize: Similar a YOLOv5, define cuántas imágenes se procesan simultáneamente.
   - Subdivisiones: Divide el batch en sub-batches para GPUs con memoria limitada.
   - Epochs: Define cuántas veces se recorre el conjunto de datos durante el entrenamiento.
   - Learning rate: Controla cómo se actualizan los pesos del modelo.
   - Momentum: Técnica para suavizar las actualizaciones de los pesos.
   - Optimizador: Algoritmo utilizado para optimizar los pesos del modelo (Adam, SGD, etc.).
2. Parámetros de arquitectura:
   - Escalas: Define las diferentes escalas a las que se aplica la detección de objetos.
   - Anchors: cuadros delimitadores predeterminados que el modelo utiliza para predecir la ubicación de los objetos.
   - Filtros: Número de filtros en las capas convolucionales del modelo.
   - Ruta de búsqueda: Define el espacio de búsqueda de posibles arquitecturas que se exploran durante el entrenamiento.
3. Parámetros de cuantización:
   - Precisión: Nivel de precisión utilizado para la cuantización de los pesos del modelo (FP8, FP16, etc.).
   - Esquema de cuantización: Define cómo se realiza la cuantización (post-entrenamiento, cuantización consciente de la arquitectura, etc.).
4. Parámetros de inferencia:
   - Input size: Define la resolución a la que se procesan las imágenes durante la inferencia.
   - Confidence threshold: Probabilidad mínima que una predicción debe tener para ser considerada válida.
   - Non-maximum suppression (NMS): Técnica para eliminar detecciones redundantes.
5. Otros parámetros:

- Clases: Define la cantidad de clases que el modelo puede detectar.
- Dataset: Conjunto de datos utilizado para entrenar el modelo.
- Hardware: Define el hardware en el que se ejecutará el modelo (CPU, GPU, etc.).

2.5.2.5. Métricas de rendimiento

Una matriz de confusión es una herramienta de evaluación de rendimiento en el Machine Learning, representa el nivel de precisión del modelo. (Bhandari, 2020)

En la matriz se visualizan valores de verdaderos positivos, negativos y falsos positivos y negativos. El objetivo de esta es analizar el rendimiento del modelo, identificar las clasificaciones fallidas y mejorar la precisión de predicción.

En el caso de un problema de clasificación binaria, se obtiene una matriz 2*2, con 4 valores:

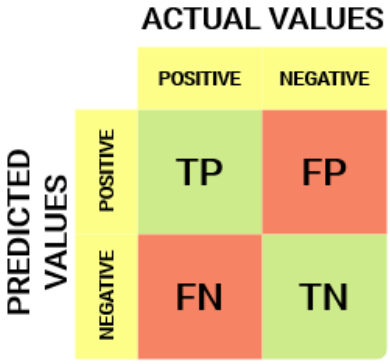


Figure 2Matriz de Confusión

Los valores de la columna representan los valores actuales y los de la fila representa los valores predichos, del objetivo variable.

Los términos importantes en la matriz son:

- Verdadero positivo (TP): para cuando el valor predicho coincide con el valor actual, o la clase predicha coincide con la clase actual. En este caso, el valor actual, igual que el valor predicho, es positivo.
- Falso positivo (FP): para cuando el valor predicho coincide con el valor actual, o la clase predicha coincide con la clase actual. En este caso, el valor actual, igual que el valor predicho, es negativo.
- Verdadero negativo (TN): el valor predicho es falso, porque el valor actual es negativo, pero el predicho es positivo.
- Falso negativo (FN): el valor actual es falso, porque el valor actual es positivo, pero el predicho es negativo.

Con los valores proporcionados en la matriz se puede evaluar el modelo entrenado con los siguientes índices:

La exactitud o Accuracy es el número de aciertos en total de casos:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$ Ecuación 18

La precisión es la relación de verdadero positivo y el número total de los positivos pronosticados:

$$Precision = \frac{TP}{TP + FP}$$ Ecuación 19

La sensibilidad o Recall es la relación de verdadero positivo y el total de positivos del ground-truth:

$$Recall = \frac{TP}{TP + FN}$$ Ecuación 20

El puntuaje F1 es una métrica que se utiliza para evaluar el rendimiento de un clasificador, se trata de una combinación de precisión y sensibilidad.

$$F1_{SCORE} = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}}$$ Ecuación 21

2.6. PYTHON

Python es un lenguaje de programación de alto nivel que combina secuencias de comandos interpretadas, compiladas, interactivas y orientadas a objetos. Tiene las siguientes características:

- Es legible
- Sencilla
- Multiplataforma
- Extensible
- Una biblioteca estándar extensa
- Fácil de mantener
- Integrable

3. DESARROLLO

El objetivo de la investigación consiste en el diseño y el desarrollo de un sistema de detección de defectos de las placas impresas PCB, para ello el trabajo está estructurada en 4 partes:

1. Recopilación y preprocesamiento de las muestras
2. Entrenamiento del modelo
3. Validación del modelo
4. Implantación del modelo

3.1. RECOPILACIÓN Y PREPROCESAMIENTO DE LAS MUESTRAS

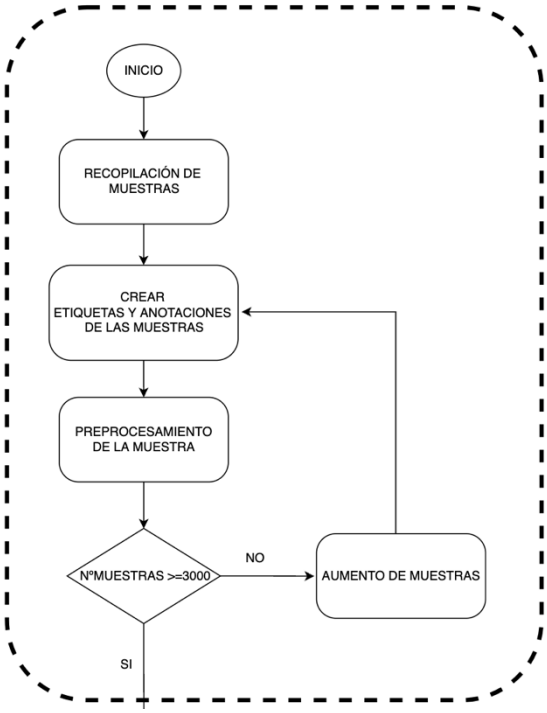


Ilustración 38 Recopilación y preprocesamiento de muestras.

Primero se obtiene el conjunto de imágenes de placas impresas con defectos en los repositorios libres de plataforma de ciencia de datos como Kaggle o Roboflow, en la plataforma de Github o en los repositorios libres de otras universidades.

En el caso nuestro, lo hemos conseguido en los recursos libres de la universidad de Peking (Dai, 2022), es una recopilación pública de imágenes de placas impresas defectuosas de 1386 imágenes y 6 clases de defectos: missing-hole, mouse-bite, open-circuit, short, spur y spurious copper.

La muestra contiene imágenes rotadas y todas están anotadas, en el caso de que las imágenes no estén anotadas, se debería etiquetarlas o anotarlas con las herramientas de anotación tales como labelimg.

Dado que la cantidad de imágenes que hemos conseguido es inferior a la mínima establecida, vamos a realizar un aumento de muestras, volteándolos horizontal y verticalmente y ajustando el brillo de la imagen. Para ello acudimos a la plataforma de Roboflow:

1. Entrar en la página de Roboflow, en la pestaña de Workplace, crear un nuevo proyecto:

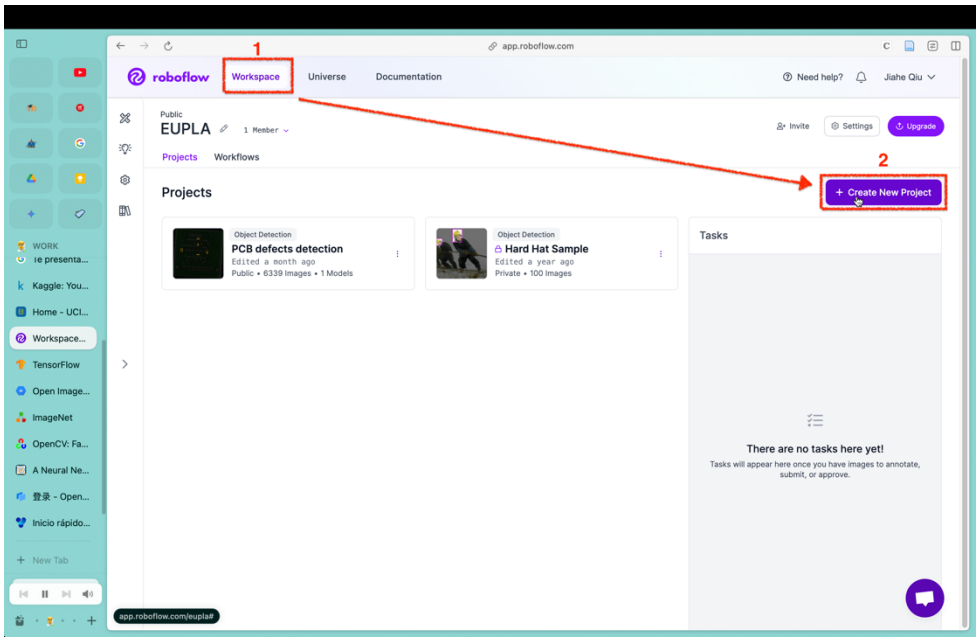


Ilustración 39 Roboflow-crear proyecto

2. Asignar un nombre al proyecto, el tipo de proyecto y de aplicación que queremos implementar y creamos el proyecto:

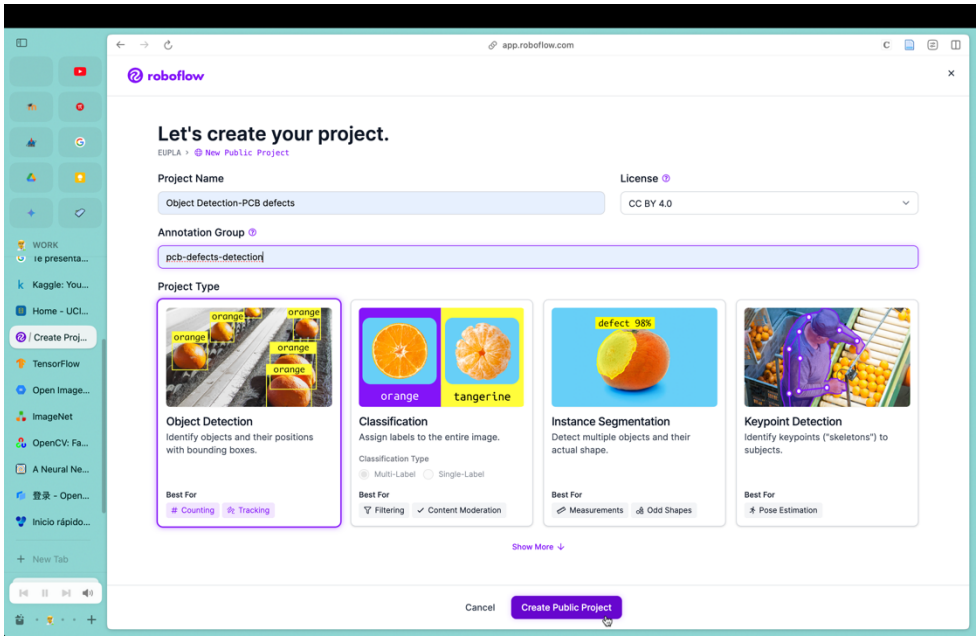


Ilustración 40 Roboflow-Asignación de nombre y tipo de aplicación

3. Subir los archivos de imágenes de la muestra y los documentos de anotación:

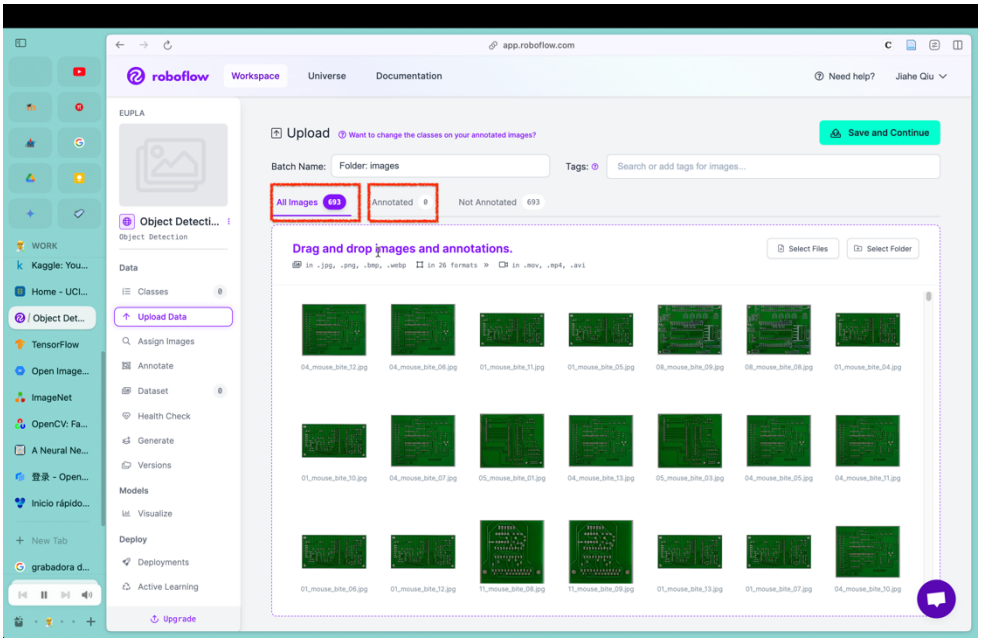


Ilustración 41 Roboflow-upload de imágenes y anotaciones

4. Dividir la muestra en conjuntos de entrenamiento, validación y comprobación:

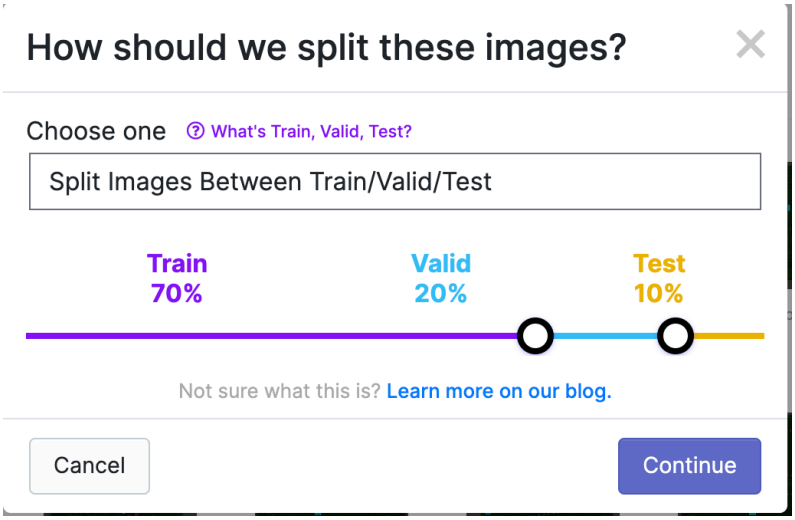


Ilustración 42 Roboflow-Dividir el dataset en conjuntos diferentes

5. Preprocesamos las imágenes. Por defecto hemos reajustado las dimensiones de las imágenes a 640*640 para facilitar el posterior entrenamiento y le hemos asignado orientaciones automáticas:

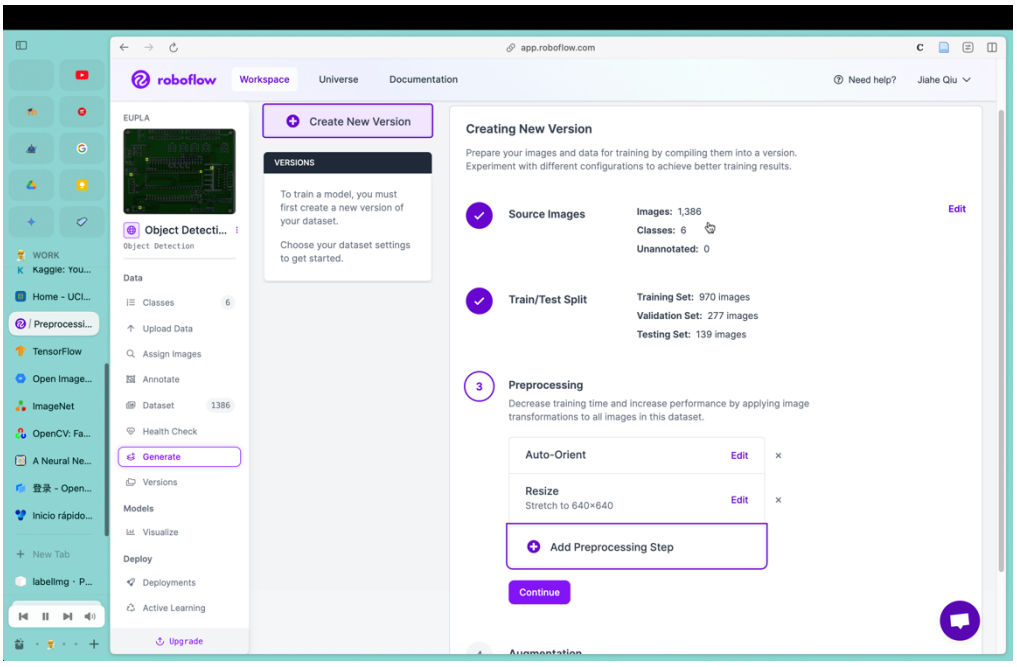


Ilustración 43 Roboflow-Preprocesamiento de las imágenes

Roboflow nos ayuda a entender las razones y los métodos de estos ajustes para que podamos maximizar nuestros resultados a la hora de entrenamiento.

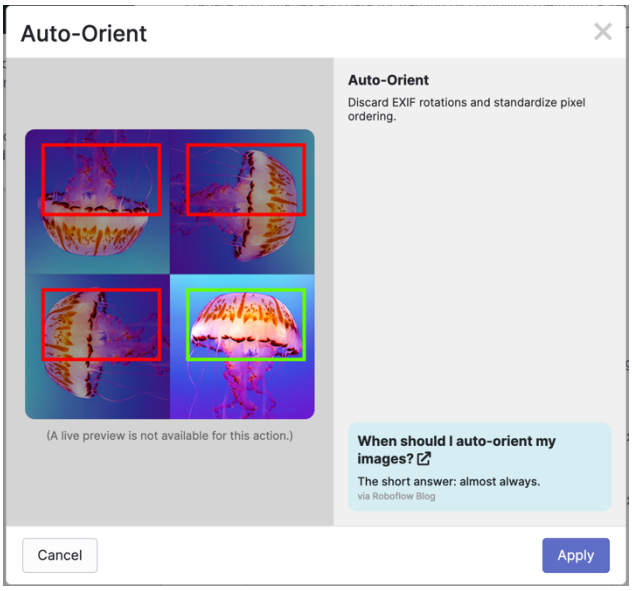


Ilustración 44 Orientación automática

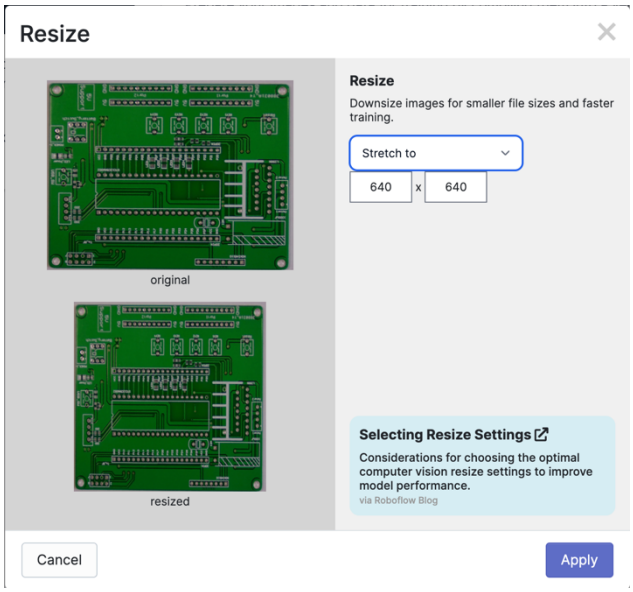


Ilustración 45 Reajuste de dimensiones

Otras opciones de preprocesamiento son las siguientes:

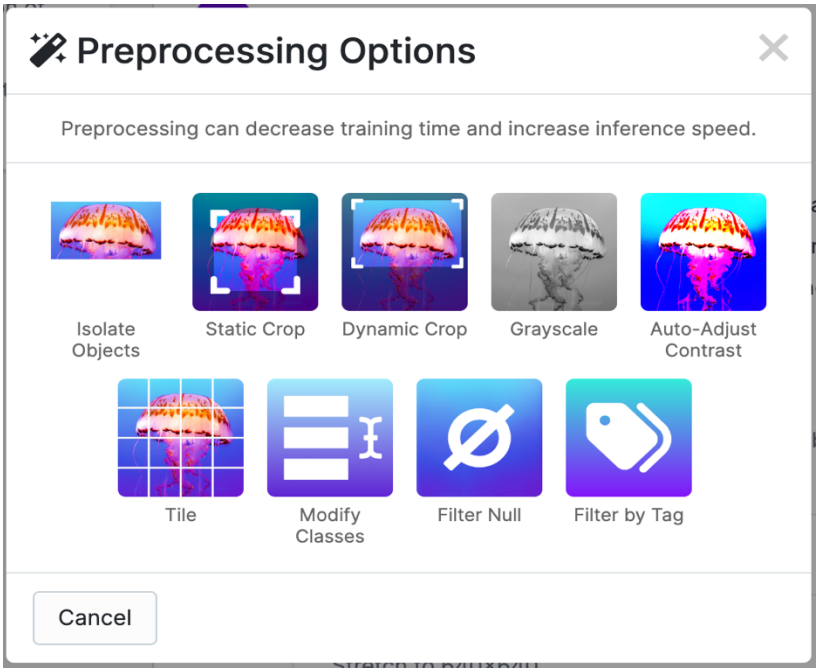


Ilustración 46 Opciones de preprocesamiento

6. Para obtener más cantidades de imágenes, acudimos a la opción de aumento:

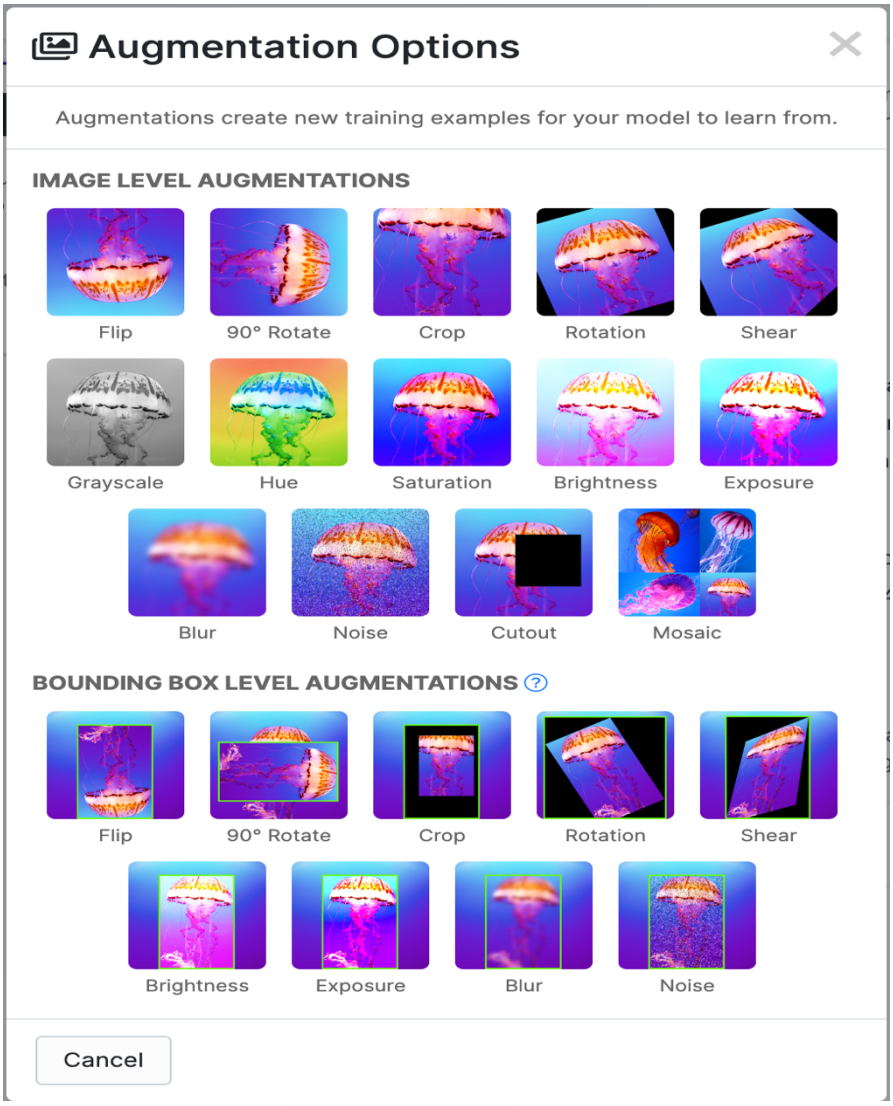


Ilustración 47 Opciones del aumento

Se deberían escoger las opciones del aumento que le sean conveniente u óptimo para su aplicación. Por ejemplo, en el caso de que su aplicación sea de clasificar objetos de diferentes colores, no es oportuno que escojan la opción de escala de grises, ya que el color resultado una de las características esenciales para su aplicación.

En el caso nuestro, por los posibles factores ambientales de luminosidad y de ángulo de posición de la placa de PCB, hemos acudido a las siguientes configuraciones:

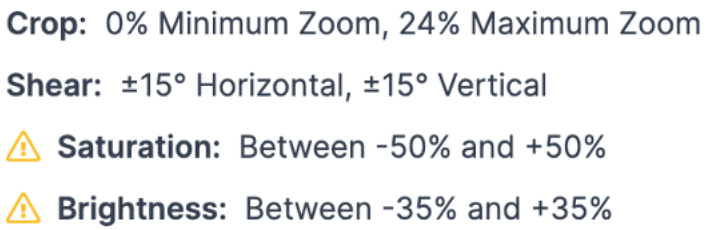


Ilustración 48 configuraciones de aumento

7. Una vez realizado las configuraciones anteriores, podemos seleccionar el tamaño del output que queramos que nos proporcione (de 1x a 50x). Podemos obtener hasta 50 variantes diferentes de cada una de las imágenes de nuestra muestra de entrenamiento. En la versión de prueba gratuita de Roboflow nos permite hasta un máximo de 3 variantes por cada imagen.

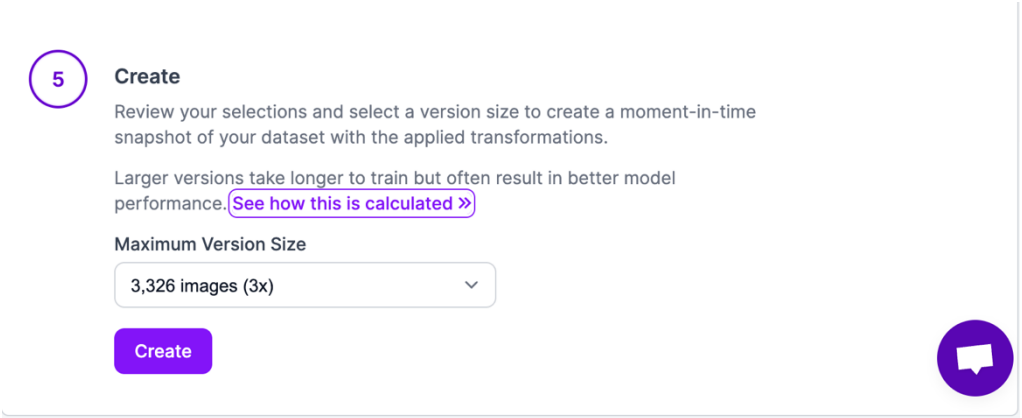


Ilustración 51 Roboflow-Seleccionar el tamaño de Output

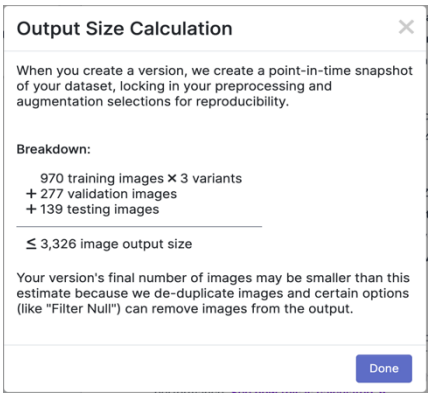


Ilustración 50 Cálculo de l tamaño de la muestra de salida

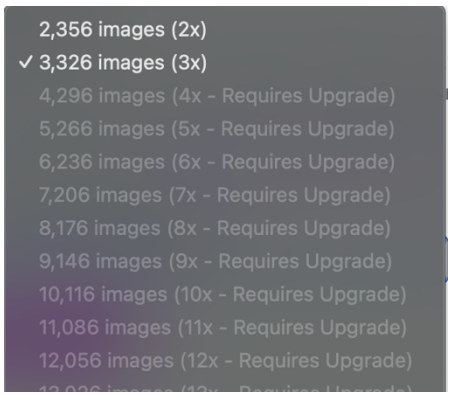


Ilustración 49 Opciones del tamaño

8. Finalmente obtendríamos lo siguiente:

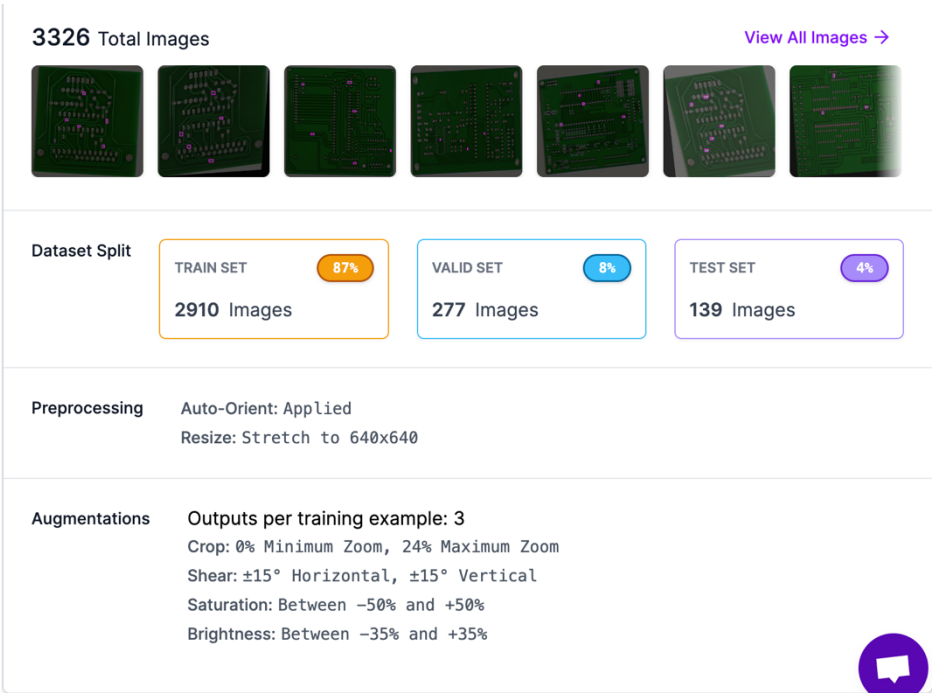


Ilustración 52 Roboflow-resultado

La plataforma de Roboflow tiene la ventaja de que puedas exportar tu conjunto de datos en el formato que requiera tu modelo de entrenamiento: coco, Pascal-VOC, YOLO, Tensorflow, etc.

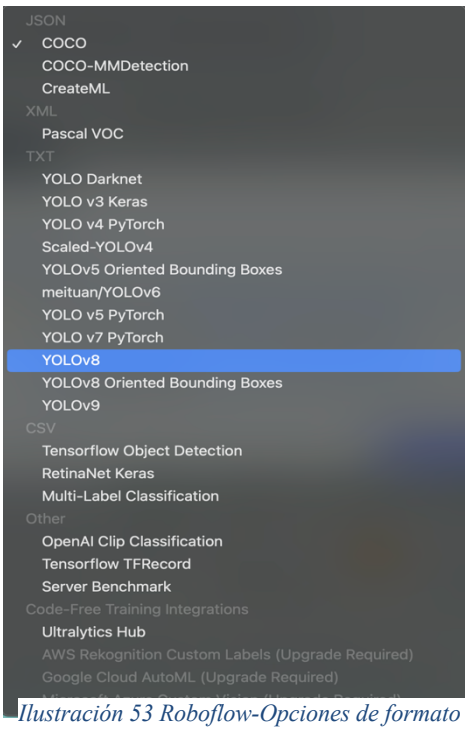


Ilustración 53 Roboflow-Opciones de formato

Puedes exportar, tanto en archivo zip para descomprimir y utilizarlo para el entrenamiento local, como en código para descargar y

entrenar en otras plataformas online de codificación y ciencias de datos como Kaggle o Google Colab.

Además, Roboflow permite también entrenamiento online con modelos de YOLO de diferentes versiones.

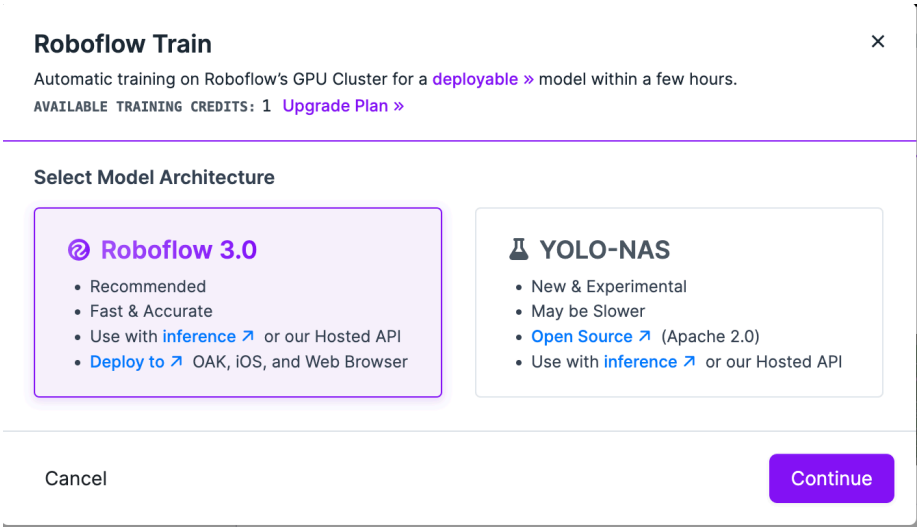


Ilustración 54 Roboflow-entrenamiento

La ventaja es que puedas entrenar fácilmente tu modelo para implementarlo en una aplicación, sin embargo, no sabes y tampoco puedes configurar los hiperparámetros para el modelo de entrenamiento. Un ejemplo de resultado de entrenamiento:

Si observamos en la Ilustración 55, en menos de 50 etapas habíamos conseguido una precisión promedia mayor del 90 %, y el Loss de clase, objeto y del cuadro delimitador está por debajo de 0.8 (cuando menor es el Loss, mejor será el resultado); sin embargo, en los posteriores entrenamientos con modelos de arquitecturas diferentes veremos que no es así.

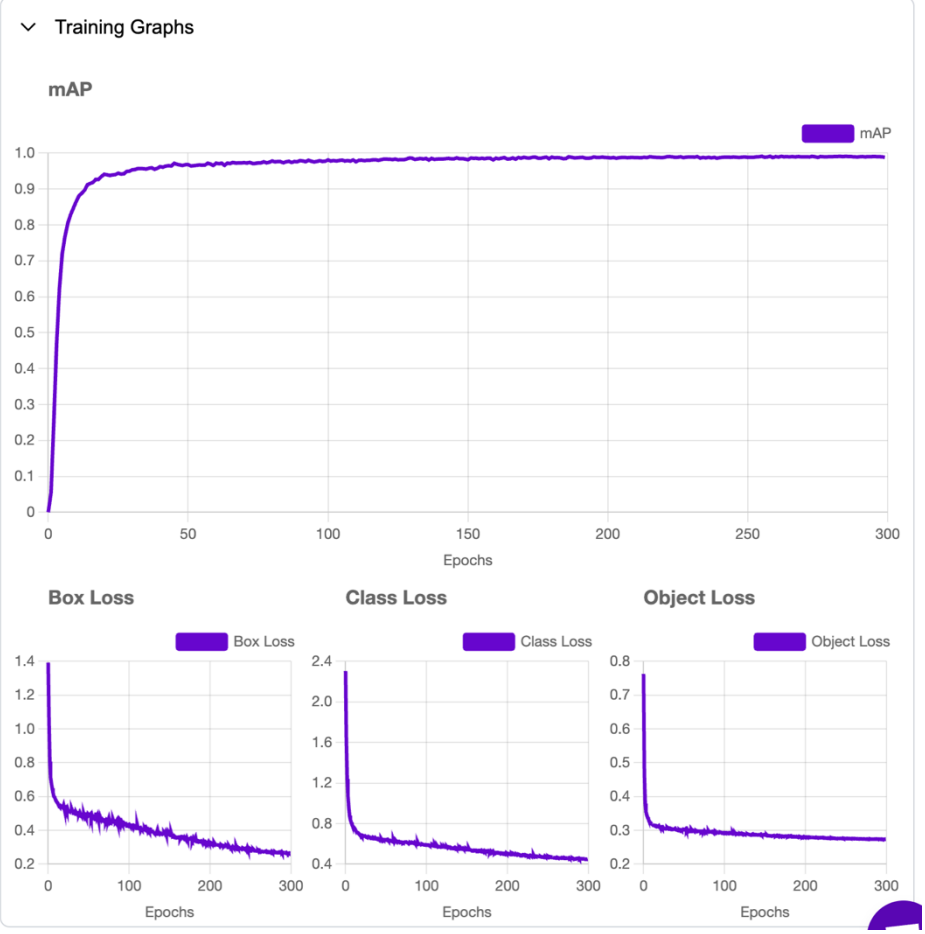


Ilustración 55 Roboflow-Gráfica del entrenamiento

La configuración de los hiperparámetros de los diferentes modelos es importante para la velocidad de entrenamiento y el resultado de validación.

3.2. ENTRENAMIENTO DEL MODELO

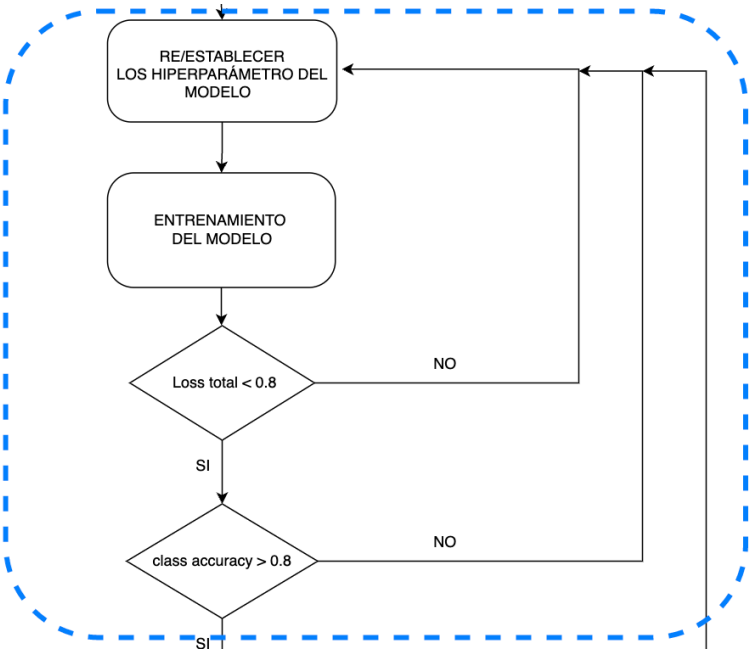


Ilustración 56 Entrenamiento del modelo

En este trabajo de investigación se va a trabajar con dos modelos con arquitecturas diferentes, por lo que es necesario preparar las muestras con los formatos requeridos por el modelo y se establecerá los mismos hiperparámetros para una posterior comparación de los resultados.

3.2.1. YOLO_NAS_S

Instalar e importar las librerías y los módulos necesarios:

Table 1 YOLO_Nas - Instalar librerias y módulos necesarios

```
%%capture
!pip install pyproject-toml
!pip install -q super-gradients
!pip install -q roboflow
!pip install -q supervision
!pip install ultralytics
```

Table 2 YOLO_Nas – Importar librerias y módulos necesarios

```
# instalando las librerias
import pandas as pd
import numpy as np
import os
import json
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image
from roboflow import Roboflow

from super_gradients.training import models
from super_gradients.training import Trainer
from super_gradients.training.dataloaders.dataloaders import
coco_detection_yolo_format_train
from super_gradients.training.dataloaders.dataloaders import
coco_detection_yolo_format_val
from super_gradients.training.losses import PPYoloELoss
from super_gradients.training.metrics import DetectionMetrics_050
from super_gradients.training.models.detection_models.pp_yolo_e import
PPYoloEPostPredictionCallback
```

Descargar el Dataset

Table 3 YOLO_Nas – Descargar el Dataset

```
from roboflow import Roboflow

# Usando el dataset prevamiente etiquetado y preprocesado en roboflow
rf = Roboflow(api_key="jQ9nQwgqzpT7KBd1f9ON")
project = rf.workspace("eupla").project("pcb-defects-detection-gxn7r")
version = project.version(1)
dataset = version.download("yolov5")
```

Asignar las rutas y definir las clases

Table 4 YOLO_Nas – Asignar las rutas y definir las clases

```
from typing import List, Dict

class config:
  # Project path
  #DATA_DIR: str = "/content/drive/MyDrive/objectoDetection_yolo/PCB-defects-detection-1"
  CHECKPOINT_DIR: str = "/content/drive/MyDrive/objectoDetection_yolo/PCB-defects-
detection-1"
  DATA_DIR: str = "/content/PCB-defects-detection-1"
  #CHECKPOINT_DIR: str = "/content/PCB-defects-detection-1"
  EXPERIMENT_NAME: str = "PCB-defects-detection"

  #Datasets
  TRAIN_IMAGE_DIR: str = "train/images"
  TRAIN_LABEL_DIR: str = "train/labels"
  VAL_IMAGE_DIR: str = "valid/images"
  VAL_LABEL_DIR: str = "valid/labels"
  TEST_IMAGE_DIR: str = "test/images"
  TEST_LABEL_DIR: str = "test/labels"

  #Classes
  CLASSES: List[str] = [ 'missing_hole', 'mouse_bite', 'open_circuit',
'short','spur','spurious_copper']
  NUM_CLASSES: int = len(CLASSES)

  #Model
  DATALOADER_PARAMS: Dict = {
      'batch_size':16,
      'num_workers':4
  }
  MODEL_NAME: str = 'yolo_nas_s'
  PRETRAINED_WEIGHTS: str = 'coco'
```

Definir los datos de entrenamiento, validación y comprobación

Table 5 YOLO_Nas – Definir los dataloaders

```
train_data = coco_detection_yolo_format_train(
    dataset_params={
        'data_dir': config.DATA_DIR,
        'images_dir': "train/images",
        'labels_dir': "train/labels",
        'classes': config.CLASSES
    },
    dataloader_params=config.DATALOADER_PARAMS
)

val_data = coco_detection_yolo_format_val(
    dataset_params={
        'data_dir': config.DATA_DIR,
        'images_dir': "valid/images",
        'labels_dir': "valid/labels",
        'classes': config.CLASSES
    },
    dataloader_params=config.DATALOADER_PARAMS
)

test_data = coco_detection_yolo_format_val(
    dataset_params={
        'data_dir': config.DATA_DIR,
        'images_dir': "test/images",
        'labels_dir': "test/labels",
        'classes': config.CLASSES
    },
    dataloader_params=config.DATALOADER_PARAMS
)
```

Definir los hiper parámetros

Table 6 YOLO_Nas – Definir los hiper parámetros

```
train_params = {
    "average_best_models":True,
    "warmup_mode": "linear_epoch_step",
    "warmup_initial_lr": 1e-6,
    "lr_warmup_epochs": 1,
    "initial_lr": 5e-4,
    "lr_mode": "cosine",
    "cosine_final_lr_ratio": 0.1,
    "optimizer": "Adam",
    "optimizer_params": {"weight_decay": 0.001},
    "zero_weight_decay_on_bias_and_bn": True,
    "ema": True,
    "ema_params": {"decay": 0.9, "decay_type": "threshold"},
    "max_epochs": 1,
    "mixed_precision": True,
    "loss": PPYoloELoss(
        use_static_assigner=False,
        num_classes=config.NUM_CLASSES,
        reg_max=16
    ),
    "valid_metrics_list": [
        DetectionMetrics_050(
            score_thres=0.1,
            top_k_predictions=300,
            num_cls=config.NUM_CLASSES,
            normalize_targets=True,
            post_prediction_callback=PPYoloEPostPredictionCallback(
                score_threshold=0.01,
                nms_top_k=1000,
                max_predictions=300,
                nms_threshold=0.8
            )
        )
    ],
```

```
    "metric_to_watch": 'mAP@0.50'
}
```

Crear el modelo e inicializar el entrenamiento

Table 7 YOLO_Nas – Seleccionar el modelo para el entrenamiento

```
# Creando el modelo
model = models.get(config.MODEL_NAME, num_classes=config.NUM_CLASSES,
pretrained_weights=config.PRETRAINED_WEIGHTS)
```

Table 8 YOLO_Nas – Inicializar Trainer

```
# Inicializando el trainer
trainer = Trainer(experiment_name=config.EXPERIMENT_NAME,
ckpt_root_dir=config.CHECKPOINT_DIR)
```

Table 9 YOLO_Nas – Iniciar el entrenamiento

```
# Entrenando el modelo
trainer.train(model=model, training_params=train_params, train_loader=train_data,
valid_loader=val_data)
```

Table 10 YOLO_Nas – Inicializar Trainer

```
# Inicializando el trainer
trainer = Trainer(experiment_name=config.EXPERIMENT_NAME,
ckpt_root_dir=config.CHECKPOINT_DIR)
```

Cargar el mejor modelo

Table 11 YOLO_Nas – Crear la ruta de checkpoint y el archivo de average_model

```
import os

# Assuming config.CHECKPOINT_DIR contains the directory where the checkpoint is located
checkpoint_dir = config.CHECKPOINT_DIR
experiment_name = config.EXPERIMENT_NAME
checkpoint_filename = 'average_model.pth'

# Construct the absolute path
ckpt_path = os.path.join(checkpoint_dir, experiment_name, checkpoint_filename)

# Now `ckpt_path` contains the absolute path to the
# checkpoint file
```

Table 12 YOLO_Nas – Visualizar las curvas de entrenamiento

```
# Look at training curves in tensorboard:
%load_ext tensorboard
#%reload_ext tensorboard
%tensorboard --logdir /content/drive/MyDrive/objectoDetection_yolo/PCB-defects-detection-1/PCB-defects-detection
```

Table 13 Evolución del Loss durante el entrenamiento

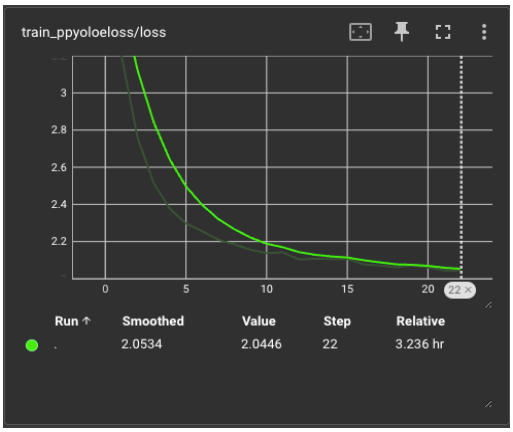
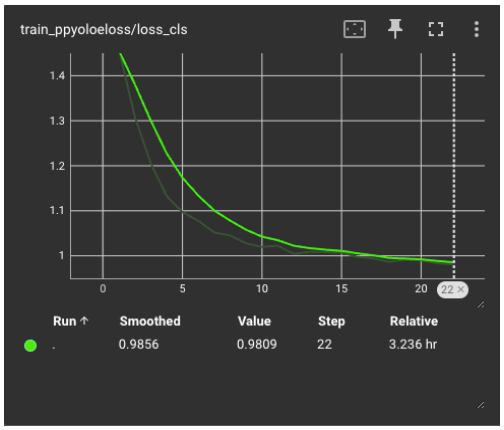
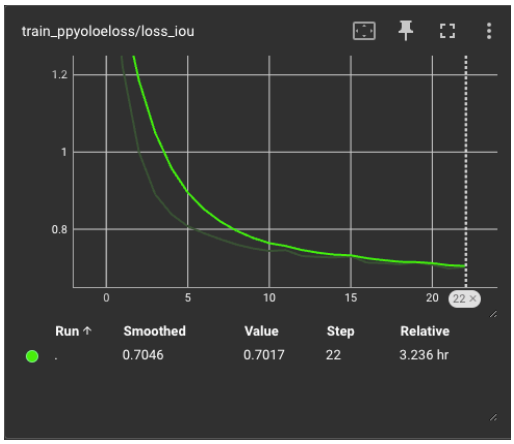
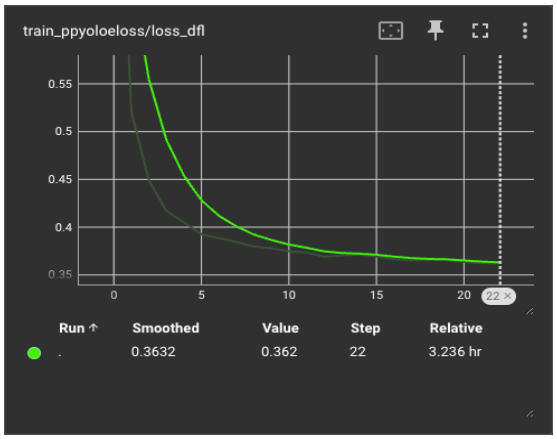
Evolución del Loss durante el entrenamiento	
	
	

Table 14Evolución de validaciones durante el entrenamiento

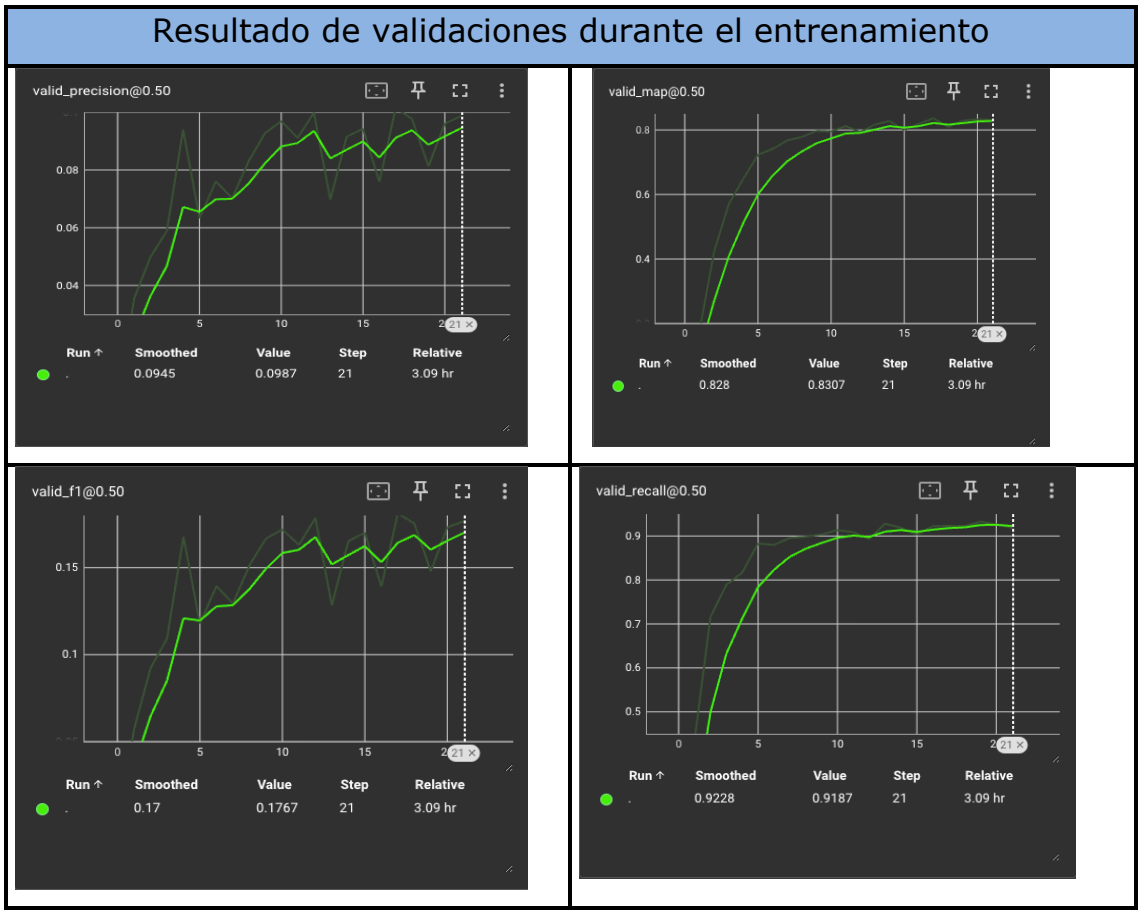


Table 15 YOLO_Nas – Escoger el mejor modelo

```
import os
avg_model = models.get(config.MODEL_NAME,
  num_classes=config.NUM_CLASSES,
  checkpoint_path=os.path.join(config.CHECKPOINT_DIR,
  config.EXPERIMENT_NAME, 'average_model.pth'))
```

3.2.2. Faster R-CNN

Instalar e importar las librerías y los módulos necesarios:

Table 16 Faster-RCNN – Descargar librerías:

```
!python -m pip install -e detectron2
```

Table 17 Faster-RCNN –Importar librerías y módulos:

```
# You may need to restart your runtime prior to this, to let your installation take
effect
# Some basic setup
# Setup detectron2 logger
import detectron2
from detectron2.utils.logger import setup_logger
setup_logger()

# import some common libraries
import matplotlib.pyplot as plt
import numpy as np
import cv2
from google.colab.patches import cv2_imshow

# import some common detectron2 utilities
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog, DatasetCatalog
```

Descargar el Dataset

Table 18 Faster-RCNN – Descargar el Dataset

```
import os
os.chdir("/content/drive/MyDrive")
!mkdir coco
os.chdir("/content/drive/MyDrive/coco")
!mkdir annotations
!wget https://app.roboflow.com/ds/azxalgtQkD?key=b2bwlrXwJA
!unzip azxalgtQkD?key=b2bwlrXwJA
```

Definir los datos de entrenamiento, validación y comprobación

Table 19 Faster-RCNN- Definir los datos de entrenamiento y validación

```
from detectron2.data import MetadataCatalog
from detectron2.data.datasets import register_coco_instances
#DatasetCatalog.clear()
##第一个参数为自己注册的数据名称，第二个参数无需管，第三个参数为数据的地址
register_coco_instances('self_coco_train', {},
                        '/content/drive/MyDrive/coco/annotations/instances_train2017.json',
                         '/content/drive/MyDrive/coco/train')
register_coco_instances('self_coco_val', {},
                         '/content/drive/MyDrive/coco/annotations/instances_val2017.json',
                        '/content/drive/MyDrive/coco/valid')
```

Table 20 YOLO_Nas – Descargar el Dataset

```
#查看一下自己的元数据
##运行后可以看到 thing_classes=["自己的数据类别"]
coco_val_metadata = MetadataCatalog.get("self_coco_val")
dataset_dicts = DatasetCatalog.get("self_coco_val")
coco_val_metadata
```

Definir los hiper parámetros e inicializar el entrenamiento

Table 21 Faster-RCNN- Definir los hiper parámetros

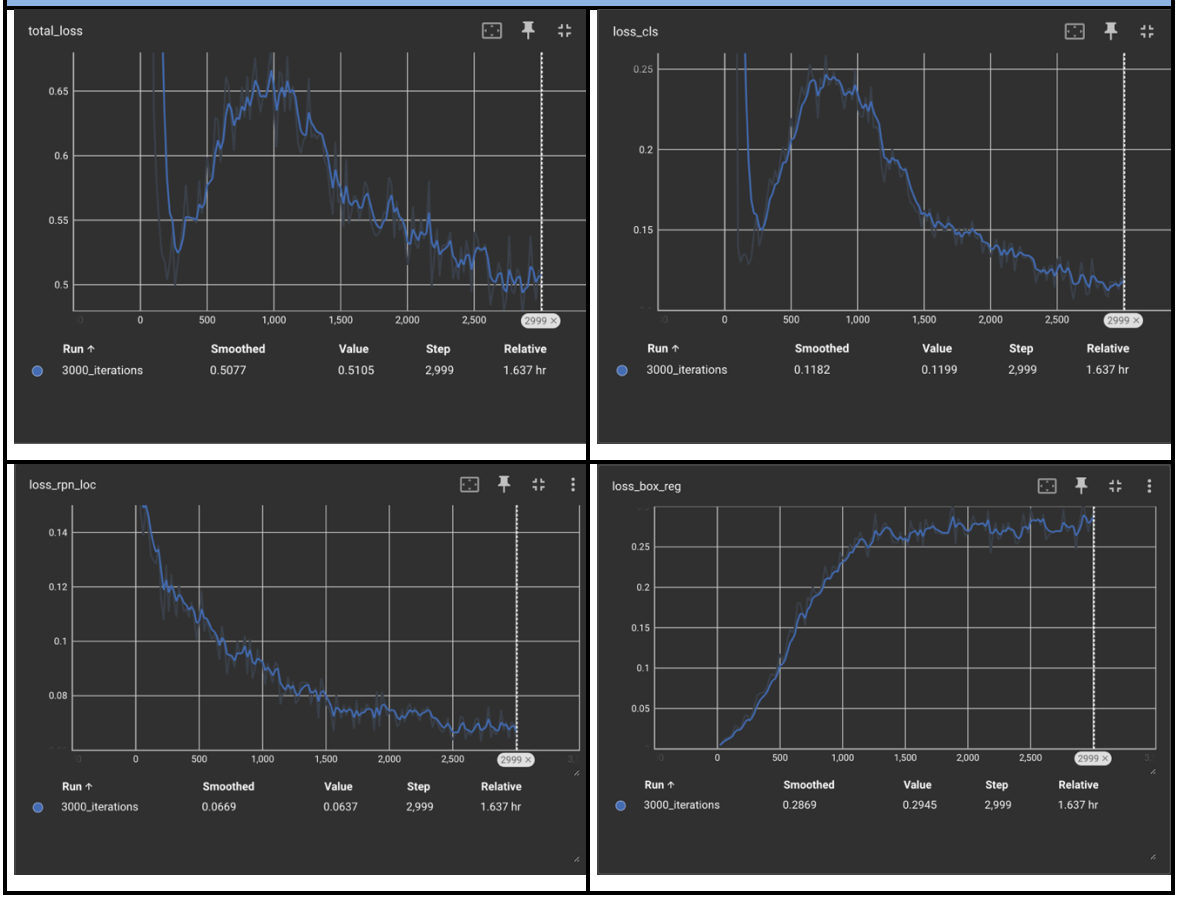
```
from detectron2.engine import DefaultTrainer
from detectron2.config import get_cfg
from detectron2 import model_zoo
import os

cfg = get_cfg()
cfg.merge_from_file(\
"/content/drive/MyDrive/detectron2/configs/COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml")
#配置的地方
cfg.DATASETS.TRAIN = ("self_coco_train",)  #设置好训练的数据集，也就是刚刚自己注册的数据集
cfg.DATASETS.TEST = ("self_coco_val", )
cfg.OUTPUT_DIR = ("/content/drive/MyDrive/coco/output")

cfg.DATALOADER.NUM_WORKERS = 4
#cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url(\
"COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml")  #让训练初试化从对应的配置开始，这里也可以先下载
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
cfg.SOLVER.IMS_PER_BATCH = 16  #Batchsize
cfg.SOLVER.BASE_LR = 5e-4  #单个 GPU 应该设置的学习率
cfg.SOLVER.MAX_ITER = 1500  # 迭代的次数   图片的数量/Batchsize 等于一个 epoch 迭代的次数
(yolo_nas_s = 3036701 iterations)
cfg.TEST.EVAL_PERIOD= 100   # 每迭代几次就进行一次评估，但结果未显现？？？表示很疑问
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 256   # default=512
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 6 + 1 # 自己数据集的类别
#cfg.MODEL.RETINANET.NUM_CLASSES = 7 + 1
#cfg.MODEL.SEM_SEG_HEAD.NUM_CLASSES = 7 + 1

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)  #传入 output 文件夹 存放训练好的权重等
trainer = DefaultTrainer(cfg)
#DefaultTrainer.test(evaluators='coco')
trainer.resume_or_load(resume=False)
trainer.train()
```

Visualizar las curvas de entrenamientos

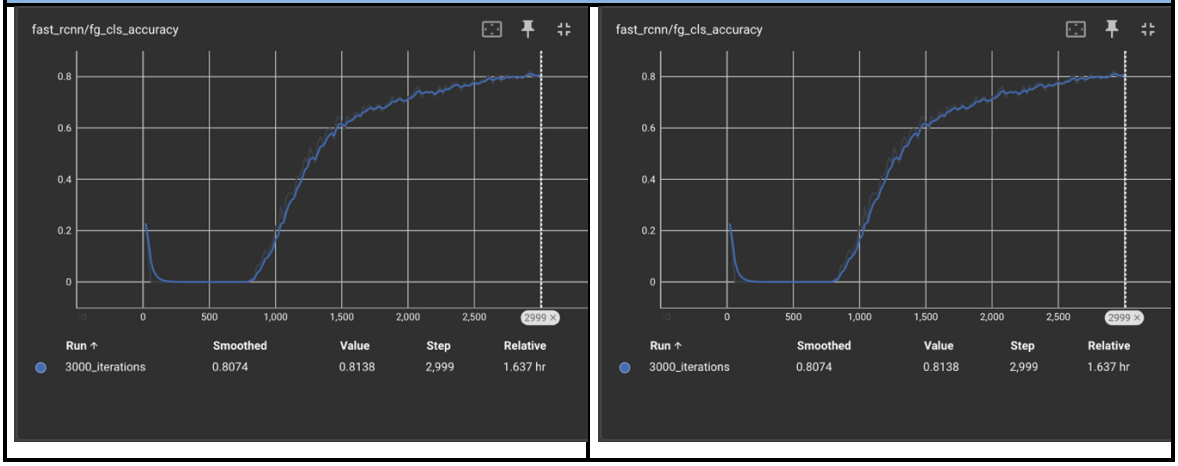
Table 22 Faster-RCNN- Visualizar las curvas de entrenamiento

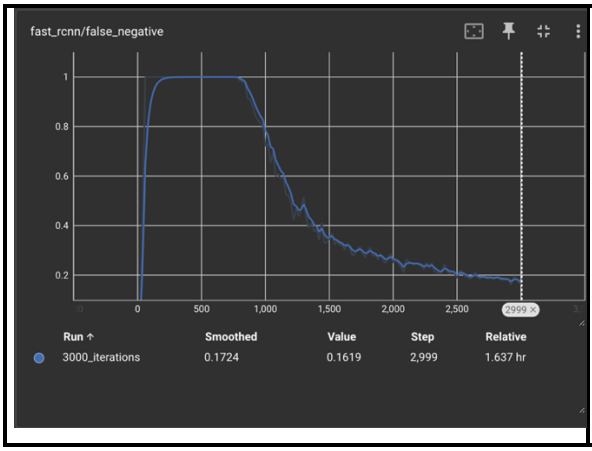
```
# Look at training curves in tensorboard:
%load_ext tensorboard
%tensorboard --logdir /content/drive/MyDrive/coco/output
#%reload_ext tensorboard
```

Evolución del Loss durante el entrenamiento



Resultado de validaciones durante el entrenamiento





Escoger el mejor modelo

Table 23 Faster-RCNN- Escoger el mejor modelo y cargar para predictor

```
import os
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
#我们进行训练的 权值文件存放处
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5  # 设置一个阈值
cfg.DATASETS.TEST = ("self_coco_val", )
predictor = DefaultPredictor(cfg)
```

3.3. VALIDACIÓN DEL MODELO

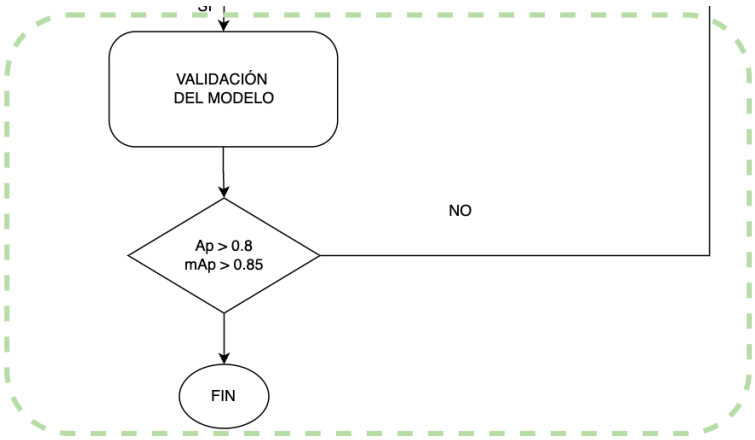


Ilustración 57 Validación del modelo

3.3.1. YOLO_NAS_S

Evaluación

Table 24 YOLO_Nas – Evaluación con el testloader

```
trainer.test(model=avg_model,
             test_loader=test_data,
             test_metrics_list=DetectionMetrics_050(score_thres=0.1,
                                                    top_k_predictions=300,
                                                    num_cls=config.NUM_CLASSES,
                                                    normalize_targets=True,
              post_prediction_callback=PPYoloEPostPredictionCallback(score_threshold=0.01,

nms_top_k=1000,

max_predictions=300,

nms_threshold=0.7)
                                                   ))
```

Visualización y predicción

Table 25 YOLO_Nas – Visualización y predicción de la imagen

```
from PIL import Image
import torch
from ultralytics import YOLO
# import supervision to visualize our results
import supervision as sv
# import cv2 to helo load our image
import cv2

ds= sv.DetectionDataset.from_yolo(
    images_directory_path=f"{config.DATA_DIR}/valid/images",
    annotations_directory_path=f"{config.DATA_DIR}/valid/labels",
    data_yaml_path=f"{config.DATA_DIR}/data.yaml",
    force_masks=False
)

IMAGE_NAME = list(ds.images.keys())[300]

image = ds.images[IMAGE_NAME]
annotations = ds.annotations[IMAGE_NAME]

box_annotator = sv.BoxAnnotator()
```

```
mask_annotator = sv.MaskAnnotator()

labels = [f"{ds.classes[class_id]}" for class_id in annotations.class_id]

annotated_image = mask_annotator.annotate(image.copy(), detections=annotations)
annotated_image = box_annotator.annotate(
    annotated_image, detections=annotations, labels=labels
)

sv.plot_image(image=annotated_image, size=(8, 8))
print(annotations)

prediction = avg_model.predict(image, conf=0.3, fuse_model=False).show()
```

3.3.2. Faster R-CNN

Evaluación

Table 26 Faster-RCNN- Evaluación

```
from detectron2.evaluation import COCOEvaluator, inference_on_dataset
from detectron2.data import build_detection_test_loader

evaluator = COCOEvaluator("self_coco_val", cfg, False,
output_dir="/content/drive/MyDrive/coco/output")
val_loader = build_detection_test_loader(cfg, "self_coco_val")
inference_on_dataset(trainer.model, val_loader, evaluator)
```

Visualización

Table 27 Faster-RCNN-Visualización

```
from detectron2.utils.visualizer import ColorMode
import random

for d in random.sample(dataset_dicts, 1):
    img = cv2.imread(d["file_name"])
    visualizer = Visualizer(img[:, :, ::-1], metadata=coco_val_metadata, scale=0.5)
    vis = visualizer.draw_dataset_dict(d)
    cv2_imshow(vis.get_image()[:, :, ::-1])
    im = cv2.imread(d["file_name"])
    outputs = predictor(im)
    v = Visualizer(im,
                   metadata=coco_val_metadata,
                   scale=0.8,
                   instance_mode =  ColorMode.IMAGE
        )
                   # remove the colors of unsegmented pixels
    print(outputs['instances'].pred_classes)
    print(outputs['instances'].pred_boxes)
    v = v.draw_instance_predictions(outputs["instances"].to("cpu"))
    cv2_imshow(v.get_image())
```

3.4. APLICACIÓN DEL MODELO

El objetivo final de este proyecto es implantar el modelo en aplicaciones reales, para ello, primero debemos establecer la escena ideal del uso del modelo de Deep Learning.

En nuestro caso queremos diseñar e implementar un sistema electrónico programable que sea capaz de trabajar en computadoras de bajo costo y de tamaño compacto como un Rasperry Pi, o en computadoras pequeñas especialmente diseñadas para aplicaciones de Machine Learning o Deep Learning como Jetson Nano.

Por lo que tenemos dos criterios básicos:

1. El modelo debería trabajar constantemente en una computadora como un servicio en segundo plano.
2. El modelo debería realizar predicciones a tiempo real.

De modo que, con los criterios establecidos, surgen dos preguntas a continuación:

1. ¿Dónde se almacenará mi modelo?
   - On-device (edge)
   - Cloud

Localización de la aplicación	Pros	contras
En el dispositivo	Es rápido	Limitados rendimiento de comutación
	Preservar la privacidad	Limitados espacios de almacenamiento
	No necesita conexiones al internet	Especificaciones especiales en las computadoras
En la Nube	No hay limites en el rendimiento de computación	Altos costes
	El modelo puede ser aplicado en cualquier sitio (API)	Predicciones más lentas
	Interconexión al ecosistema del nube	Problema de privacidad

Figure 3 Ventajas y desventajas de cada localización de aplicación

2. ¿Cómo funcionará mi modelo?
   - Online (tiempo real)
   - Offline (Periódicamente)

En nuestro caso nos interesa que el modelo de Deep Learning este almacenado en la nube, dado que nos interesa tener una alta potencia de computación y poder aplicarlo en cualquier entorno.

Las principales herramientas o fuentes de implantación del modelo son las siguientes:

Herramientas/Fuentes	Tipos de implantación
Google's ML Kit	On-device (Android y iOS)
Apple's Core ML y coremltools Python package	On-device (todos los dispositivos de apple)
AWS Sagemaker	Cloud
Google Cloud's Vertex AI	Cloud
Microsoft's Azure Machine Learning	Cloud
Hugging Face Space	Cloud
API con FastAPI	Cloud/self-hosted server
API con Torchservice	Cloud/self-hosted server
ONNX(Open Neural Network Exchange)	En general

Figure 4 Herramientas o fuentes de implantacion del modelo

Si has entrenado tu modelo previamente en la plataforma de data ciencia de Roboflow, en la misma página te ofrece numerosas opciones de implantación del modelo:

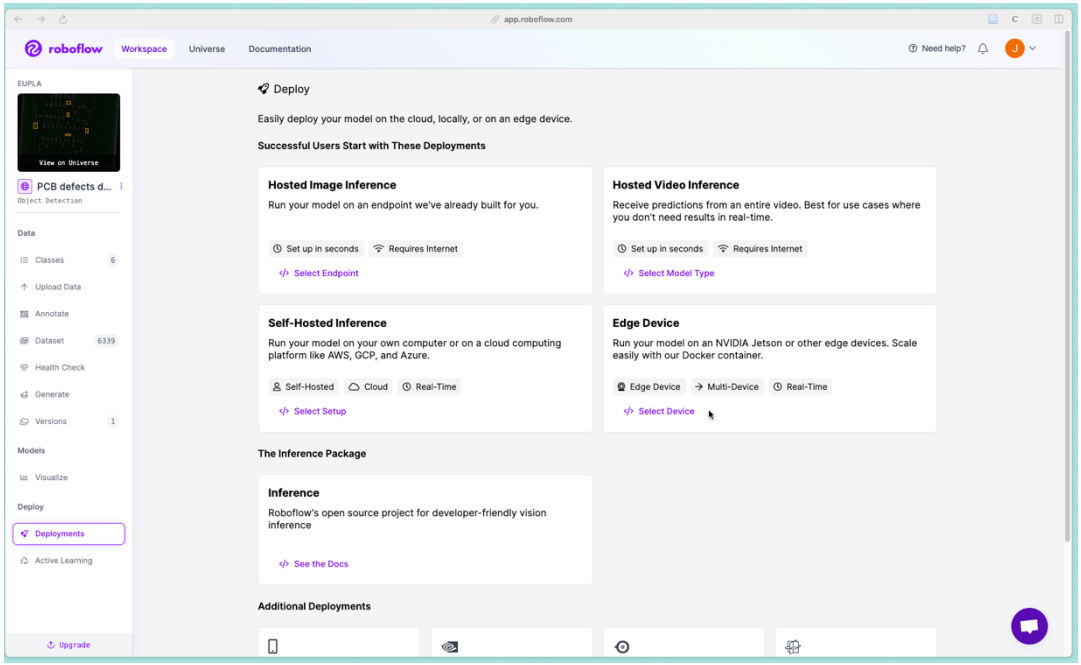


Figure 5Opciones de implantación en el ROBOFLOW

En nuestro caso, habíamos entrenado el modelo en el Roboflow y hemos obtenido buenos resultados:

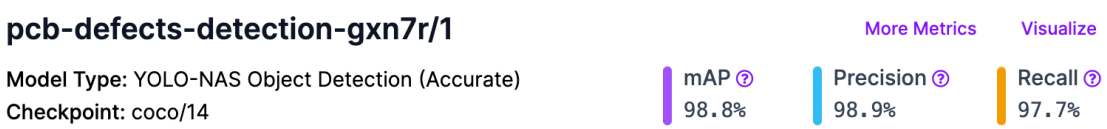


Ilustración 58 Roboflow-Resultado de entrenamiento

Por lo que se va a implantar este modelo para nuestra aplicación. Para ello acudimos al foro oficial del Roboflow (Roboflow, 2024):

3.4.1. Para Windows S.O.

Lo primero es instalar la librería inference en nuestro entorno del trabajo, abrimos el terminal y escribimos el siguiente código:

Table 28 Instalando la librería inference

```
pip install inference
```

Después debemos exportar nuestra clave de API del Roboflow al entorno del trabajo, así que, en el mismo terminal, introducimos lo siguiente:

Table 29 Exportando el código Api al entorno del trabajo

```
export ROBOFLOW_API_KEY="your-api-key"
```

Lo siguiente sería escribir el Script donde importamos las librerías necesarias y llamamos a nuestro modelo mediante API y la cámara web para realizar las predicciones a tiempo real.

Table 30 Script de Inferencia con camara

```
# Import the InferencePipeline object
from inference import InferencePipeline
# Import the built in render_boxes sink for visualizing results
from inference.core.interfaces.stream.sinks import render_boxes

# initialize a pipeline object
pipeline = InferencePipeline.init(
    model_id="pcb-defects-detection-gxn7r/1", # Roboflow model to use
    video_reference=0, # Path to video, device id (int, usually 0 for built in webcams),
or RTSP stream url
    on_prediction=render_boxes, # Function to run after each prediction
)
pipeline.start()
pipeline.join()
```

Finalmente, con la herramienta “auto-py-to-exe” convertimos el script de Python en un archivo ejecutable para el sistema operativo de Windows.

3.4.2. Para el Rasperry Pi 4

1. Instalar Docker

Table 31 Instalar Docker

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

2. Instalar el CLI Inference y empezar un contenedor.

Table 32 Instalar CLI Inference y empezar un contenedor

```
pip install inference-cli && inference server start
```

3. Ejecutar la inferencia por API.

Table 33 Ejecutar la inferencia

```
# for inference-cli<0.9.9
inference infer YOUR_IMAGE.jpg \
    --project-id pcb-defects-detection-gxn7r \
    --model-version 1 \
    --api-key YOUR-API-KEY

# for inference-cli>0.9.9
inference infer \
    -i YOUR_IMAGE.jpg \
    -m pcb-defects-detection-gxn7r/1 \
    --api-key YOUR-API-KEY
```

4. Para nuestro caso de aplicación necesitamos que el script funcione como un servicio en un segundo plano, por lo que vamos a crear un archivo RealTimeDetection.service.

Table 34 crear un archivo RealTimeDetection.service

```
[Unit]
Description=My service
After=network.target

[Service]
Restart=on-failure
RestartSec=5
ExecStart=/usr/bin/python3 -u /home/pi/RealTimeDetection_tfg.py
StandardOutput=inherit
StandardError=inherit
User=pi

[Install]
WantedBy=multi-user.target
```

Table 35 RealTimeDetection_tfg.py

```
# Import the InferencePipeline object
from inference import InferencePipeline
# Import the built in render_boxes sink for visualizing results
from inference.core.interfaces.stream.sinks import render_boxes

# initialize a pipeline object
pipeline = InferencePipeline.init(
    model_id="pcb-defects-detection-gxn7r/1", # Roboflow model to use
    video_reference=0, # Path to video, device id (int, usually 0 for built in webcams),
or RTSP stream url
```

```
    on_prediction=render_boxes, # Function to run after each prediction
)
pipeline.start()
pipeline.join()
```

5. Guardar RealTimeDetection.service en el directorio de /etc/systemd/system

Table 36 salvar RealTimeDetection.service

```
sudo cp realtime.service /etc/systemd/system/realtime.service
```

6. Commandos

7. Table 37 Comandos

```
#Ejecutar el servicio
sudo systemctl start realtime.service

#Detener el servicio
sudo systemctl stop realtime.service

#Configurar el modo daemon
sudo systemctl enable realtime.service

#Visualizar el estatus del servicio
systemctl status realtime.service

#Visualizar el proceso del servicio
ps -ef |grep realtime.service

#Salir del proceso el estatus del servicio
kill -9  567(Process_ID)
```

3.4.3. Para el Jetson Nano

1. Formatear y reinstalar el S.O. al dispositivo Jetson Nano

Table 38 Flashear el dispositivo

```
git clone https://github.com/jetsonhacks/jetsonUtilities.git
cd jetsonUtilities
python jetsonInfo.py
```

Asegurar que el dispositivo este flasheado con Jetpack 4.5, 4.6, o 5.1.

2. Ejecutar el contenedor Docker

Table 39 Ejecutar el contenedor Docker

```
sudo docker run --privileged --net=host --runtime=nvidia \
--mount source=roboflow,target=/tmp/cache -e NUM_WORKERS=1 \
roboflow/roboflow-inference-server-jetson-4.5.0:latest
```

3. Utilizar el servidor

Table 40 Utilizar el servidor

```
base64 YOUR_IMAGE.jpg | curl -d @- "http://localhost:9001/pcb-defects-detection-
gxn7r/1?api_key=jQ9nQwgqzpT7KBd1f9ON"
```

4. Para que el Script funcione en modo Daemon, vamos a crear primero un archivo start.bash

Table 41 Start.bash

```
#! /bin/bash
cd /home/zhlab/tensorRT_Pro-main/workspace
./pro yolo
```

5. Crear Start.service

Table 42 Start.service

```
[Unit]
Description=Run a Custom Script at Startup
After=default.target

[Service]
ExecStart=/home/zhlab/tensorRT_Pro-main/workspace/start.sh

[Install]
WantedBy=default.target
```

6. Agregar permisos

Table 43 Agregar permisos

```
sudo chmod 777 start.sh
sudo chmod 777 start.service
```

7. Guardar Start.service en el directorio /etc/systemd/system

Table 44 Salvar Start.service

```
sudo mv start.service /etc/systemd/system
```

8. Iniciar el servicio

Table 45 iniciar el servicio

```
systemctl daemon-reload
systemctl enable start.service        // iniciar el servicio
sudo reboot        // reiniciar todo el sistema
```

4. RESULTADOS

Durante el entrenamiento, nuestro propósito fue que ambos modelos llevaran a cabo el mismo número de épocas y que compararan los resultados de ambos. No obstante, el criterio de definición del ciclo de entrenamiento para cada modelo es distinto. Para YOLO_Nas se deben definir las épocas de entrenamiento, mientras que para Faster R-CNN se considera el número total de iteraciones.

Se sabe que:

$$Iteraciones_{época} = \frac{Conjunto\ de\ Datos}{Batchsize} \qquad \text{Ecuación 22}$$

Entonces, para un Batchsize de 16 imágenes y un conjunto de datos de 5093 imágenes, el número de iteraciones sería:

$$Iteraciones_{época} = \frac{5093}{16} \approx 318 \frac{iteraciones}{época} \qquad \text{Ecuación 23}$$

De esta forma, podríamos establecer el mismo número de épocas o iteraciones para el entrenamiento de ambos modelos.

Finalmente, estableciendo los mismos parámetros de entrenamiento y realizado los entrenamientos, los resultados de evaluación son los siguientes:

YOLO_Nas:

Table 46 Resultado de Validación YOLO_Nas

Épocas	10	35
Precisión@0.50	0.6642	0.9480
mAP@0.50	0.6599	0.9169
Recall@0.50	0.1605	0.8473
F1@0.50	0.1973	0.8914

- **Faster R-CNN:**

Table 47 Resultado de validación Faster R-CNN

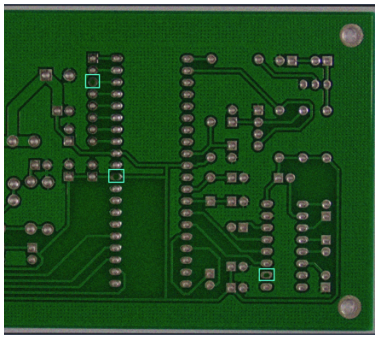
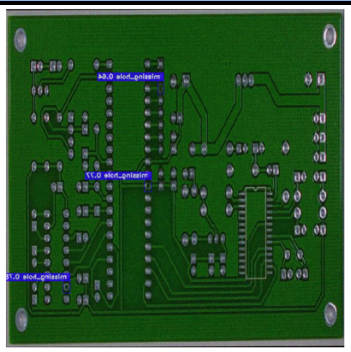
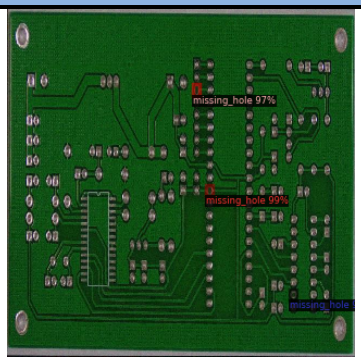
Iteraciones	3000	9000
Precisión@0.50:0.95	0.398	0.478
mAP@0.50	0.852	0.925
Recall@0.50:0.95	0.477	0.559
F1@0.50:0.95	0.433	0.515

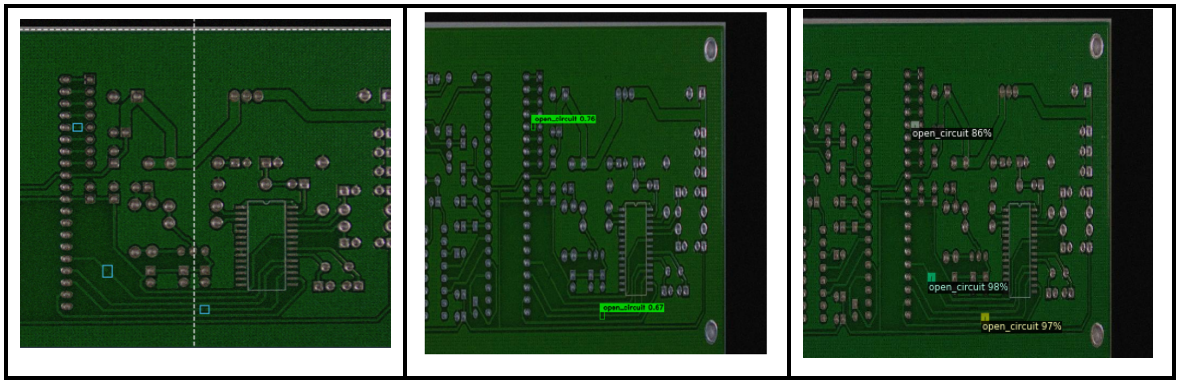
La precisión indica la capacidad del acierto del modelo en sus propias predicciones, mientras que la sensibilidad o Recall indica la capacidad del acierto en los valores de referencia.
En el caso de YOLO_Nas, su precisión indica que el modelo detecta placas defectuosas el 94.80% de sus predicciones y su Recall dice que esas predicciones sobre placas defectuosas un 84.73% son verdaderos.

Se aprecia que el mAP (precisión promedio) de ambos modelos son similar; a pesar de ello, el tiempo empleado para alcanzarlo es sumamente distinto. El entrenamiento para YOLO_Nas fue de 6 horas, mientras que el mismo número de épocas para Faster R-CNN le duraría aproximadamente 13 horas.

En cuanto a la inferencia de los modelos sobre las imágenes, observamos que las predicciones de Faster R-CNN son más precisas que las de YOLO_Nas.

Table 48 Predicciones

Imágenes de referencia	Predicciones de YOLO_Nas	Predicciones de Faster R-CNN
		

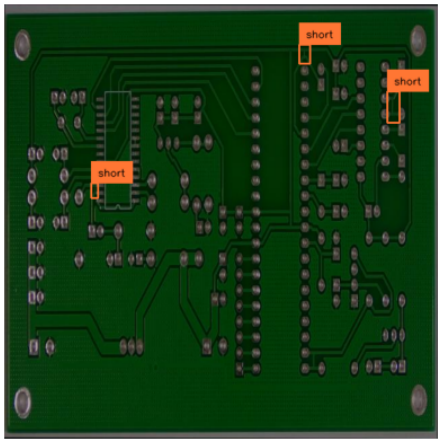
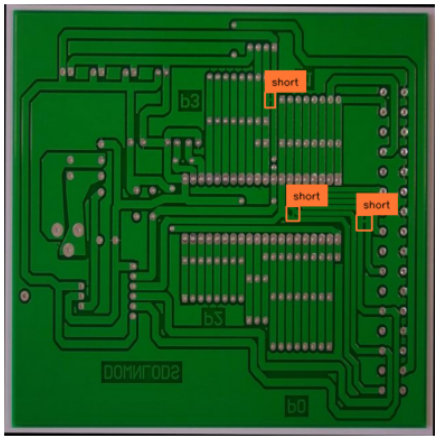


YOLO_Nas es capaz de realizar detección a tiempo real muy rápida debida a su arquitectura de una única etapa, donde predice los cuadros delimitadores y la clase del objeto; sin embargo, debido a su "simplicidad" es difícil de extraer las características para objetos pequeños, como los son en este caso.

Por el contrario, Faster R-CNN no es adecuado para aplicaciones de tiempo real, ya que es más lento debido a su arquitectura de dos etapas. Se requiere más potencia de computación para generar las regiones propuestas y clasificar los objetos, pero es más preciso y robusto a variaciones en las condiciones de iluminación y fondo.

Se observa en la Table 48 que a pesar de que las imágenes hayan reducido sus resoluciones, ambos modelos han sido capaces de detectar los defectos de fabricación. A pesar de que, en situaciones adversas de luminosidad y resolución de imagen, Faster R-CNN es más preciso. Esto también se observa en Table 49 y Table 50, sea el porcentaje de predicción de clases de objeto o sea el área de intersección entre cuadros delimitadores, el modelo de Faster R-CNN es mejor.

Table 49 Predicciones de YOLO_Nas

Predicciones de YOLO_Nas		
Imágenes de referencia		

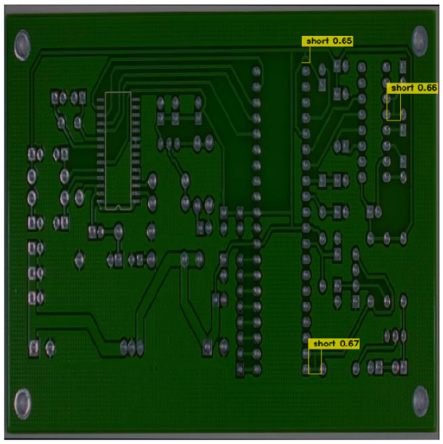
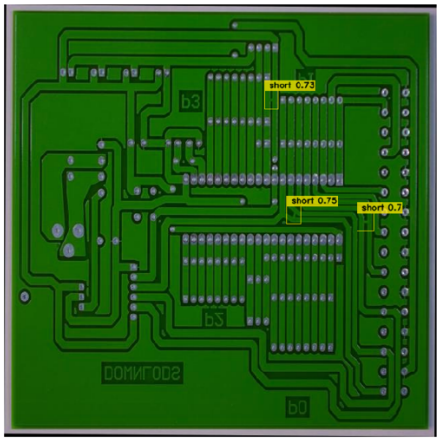
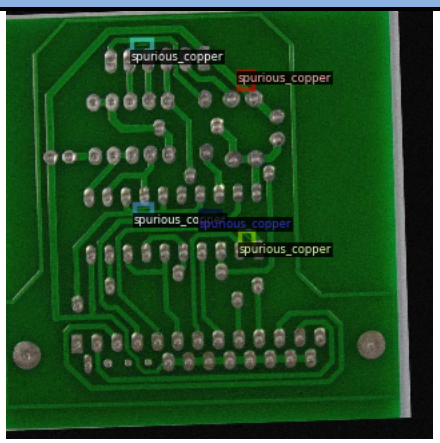
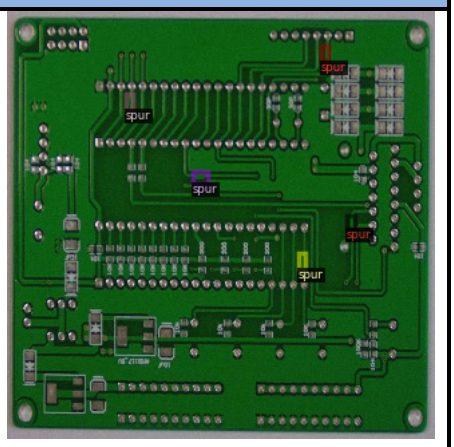
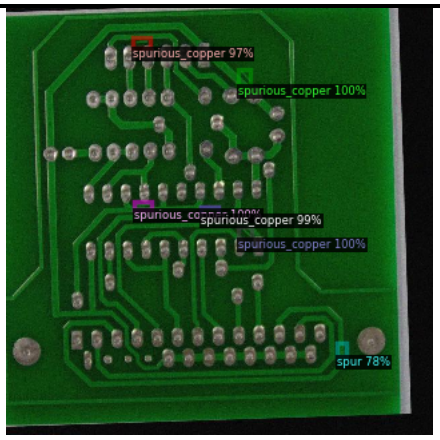
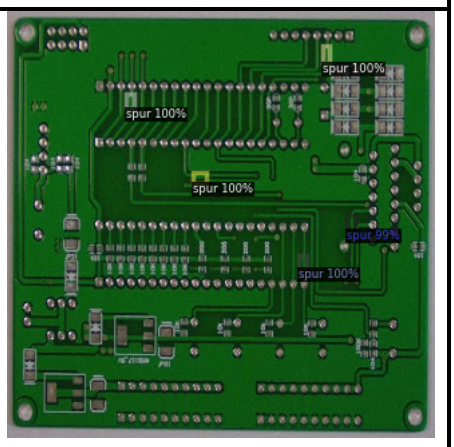
Imágenes predichas		

Table 50 Predicciones Faster R-CNN

Predicciones de Faster R-CNN		
Imágenes De referencia		
Imágenes predichas		

5. CONCLUSIONES

La etapa del control de calidad en el proceso de producción de placas impresas PCB es crucial para el proceso industrial. Dado que la producción es masiva y que la velocidad de producción es rápida, es necesario que el modelo de inspección de defectos se realice en tiempo real.

En primera instancia, se había planteado y estudiado la aplicación de redes neuronales profundas como VGGnet, Resnet o Mobilenet para la clasificación de imágenes. Se han llevado a cabo algunos estudios con el conjunto de datos de las placas impresas defectuosas y se ha comprobado que, con una red neuronal profunda de 16 capas como VGGnet, solo era capaz de clasificar la imagen en una categoría; pero no podía detectar los objetos pequeños que aparecían en la imagen. Asimismo, un problema era que solo se podrían introducir imágenes de 28x28 píxeles en su entrada, ya que, con las dimensiones más grandes, era incapaz de procesarlo.
Son redes capaces de identificar y clasificar las imágenes en diferentes categorías y también de localizar y delimitar los objetos dentro de una imagen. No obstante, para la aplicación de detección de objetos es necesario construir varias redes de VGGnet, cada una para una tarea específica. Sin embargo, no es viable la construcción de un modelo de detección de objetos por cuenta propia partiendo desde cero.

Por consiguiente, se procede a acudir a las fuentes, librerías y modelos libres para la identificación de objetos. Se seleccionaron modelos como YOLO_Nas y Faster R-CNN, ya que son los más utilizados para la detección de objetos y son fáciles de entrenar e implementar. Además, ofrecen redes Backbone de diferentes arquitecturas, con o sin pesos preentrenados.

Los resultados obtenidos, como se muestra Table 46 y Table 47, son muy semejantes; pero como se había mencionado anteriormente, la diferencia del tiempo de entrenamiento es sumamente significativa. Se observa que, para el mismo número de épocas, Faster R-CNN cuesta el doble del tiempo para el mismo número de épocas.

Además, se observa en la validación del modelo, que para una aplicación de tiempo real como el control de calidad en la producción masiva y rápida de placas impresas, es mejor el modelo de YOLO_Nas, ya que es capaz de detectar defectos a tiempo real; sin embargo, como se muestra en Table 48, la precisión del YOLO_Nas es inferior a la de Faster RCNN.

A pesar de que Faster R-CNN no es apto para la detección en tiempo real, la robustez del modelo frente a las variaciones de luminosidad y fondo hace que sea óptimo para controles de calidad que no requieren velocidad de detección.

El tiempo de montaje de un circuito impreso depende del número de componentes y el tiempo de colocación de cada componente.

Para (*NeoDen K1830 Automatic SMD Pick And Place Machine*, 2021), el tiempo estándar de montaje para el componente Chip es de 0.3s a 0.6s. En consecuencia, para el montaje de 100 componentes se requeriría entre 40s-50s, es decir, tendríamos 40s-50s para realizar la inspección de la placa.

Para un conjunto de datos estándar de Pascal VOC con GPU, YOLO_Nas es capaz de alcanzar una detección de 20-30 FPS (Fotograma por segundo); mientras que Faster R-CNN llega a una velocidad de 5-10 FPS.
El tiempo mínimo que necesita YOLO_Nas para inspeccionar:

$$\frac{1s}{20\frac{frame}{s}} = 0.05s = 50ms$$

El tiempo mínimo para Faster R-CNN es:

$$\frac{1s}{5\frac{frame}{s}} = 0.2s = 200ms$$

En los casos más adversos, para el modelo Faster R-CNN con un lapso de detección de 200 ms, se requeriría al menos una única instalación de un componente para un ciclo estándar de montaje. No obstante, para el ciclo de montaje máximo (30ms-60ms), se necesitaría en este caso como mínimo la instalación de 7 componentes para poder realizar la inspección.

Pues en el caso del Faster R-CNN, en el trabajo de investigación de (Ren et al., 2016b), donde utiliza la combinación del modelo Faster R-CNN y RPN (Red de propuesta de regiones), consiguieron un tiempo de detección de 196ms con un nivel de confianza de 0.6. En nuestro caso, para una imagen de 640x640 píxeles, el tiempo de revisión es de 52.2 ms. La velocidad de inspección varía en función de la cantidad de defectos en la placa, las regiones propuestas necesarias, el tamaño del conjunto de datos, la red neuronal Backbone, etc.

6. OBJETIVOS DE DESARROLLO SOSTENIBLE

Los objetivos de este Trabajo Fin de Grado están alineados con los siguientes Objetivos de Desarrollo Sostenible (ODS) y metas de la Agenda 2030:

- Objetivo 4 - Garantizar una educación inclusiva y equitativa de calidad y promover oportunidades de aprendizaje permanente para todos
- Meta 4.4 De aquí a 2030, aumentar considerablemente el número de jóvenes y adultos que tienen las competencias necesarias, en particular técnicas y profesionales, para acceder al empleo, el trabajo decente y el emprendimiento

- Objetivo 8 - Promover el crecimiento económico sostenido, inclusivo y sostenible, el empleo pleno y productivo y el trabajo decente para todos
- Meta 8.2 Lograr niveles más elevados de productividad económica mediante la diversificación, la modernización tecnológica y la innovación, entre otras cosas centrándose en los sectores con gran valor añadido y un uso intensivo de la mano de obra

- Objetivo 12 - Garantizar modalidades de consumo y producción sostenibles
- Meta 12.5 De aquí a 2030, reducir considerablemente la generación de desechos mediante actividades de prevención, reducción, reciclado y reutilización

- Objetivo 13 – Adoptar medidas urgentes para combatir el cambio climático y sus efectos

- Meta 13.3 Mejorar la educación, la sensibilización y la capacidad humana e institucional respecto de la mitigación del cambio climático, la adaptación a él, la reducción de sus efectos y la alerta temprana

7. BIBLIOGRAFÍA

Adibhatla, V. A., Chih, H.-C., Hsu, C.-C., Cheng, J., Abbod, M. F., Shieh, J.-S., Adibhatla, V. A., Chih, H.-C., Hsu, C.-C., Cheng, J., Abbod, M. F., & Shieh, J.-S. (2021). Applying deep learning to defect detection in printed circuit boards via a newest model of you-only-look-once. *Mathematical Biosciences and Engineering*, *18*(4), 4411-4428. https://doi.org/10.3934/mbe.2021223

Aguilar Margalejo, E. (2023). *Desarrollo de modelos de prediccio´n para la mejora de servicios de transporte colectivo de autobuses urbanos mediante t´ecnicas de Machine Learning*. UNIVERSIDAD DE ZARAGOZA.

Bhandari, A. (2020, abril 17). Understanding & Interpreting Confusion Matrix in Machine Learning (Updated 2024). *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/

Bravo Jordana, J. (2020). *Descripción, fabricación y montaje de una PCB*. http://dspace.uib.es/xmlui/handle/11201/151210

Chauhan, A. P. S., & Bhardwaj, S. C. (2011). *Detection of Bare PCB Defects by Image Subtraction Method using Machine Vision*.

Coombs, C. F., & Holden, H. T. (2016). *Printed Circuits Handbook, Seventh Edition* (7th edition). McGraw-Hill Education.

Dai, L. (2022). *Open Lab on Human Robot Interation*. https://robotics.pkusz.edu.cn/resources/dataset/

Daniel. (2021, diciembre 16). Convolutional Neural Network: Definición y funcionamiento. *Formación en ciencia de datos | DataScientest.com*. https://datascientest.com/es/convolutional-neural-network-es

Ding, R., Dai, L., Li, G., & Liu, H. (2019). TDD-net: A tiny defect detection network for printed circuit boards. *CAAI Transactions on Intelligence Technology*, *4*(2), 110-116. https://doi.org/10.1049/trit.2019.0019

Duan, X. (2018). *深入浅出Python机器学习*. 清华大学出版社. http://www.tup.com.cn，http://www.wqbook.com

Glenn J. (2023, noviembre 12). *YOLO-NAS Ultralytics*. https://docs.ultralytics.com/zh/models/yolo-nas

Hu, B., & Wang, J. (2020). Detection of PCB Surface Defects With Improved Faster-RCNN and Feature Pyramid Network. *IEEE Access*, *8*, 108335-108345. https://doi.org/10.1109/ACCESS.2020.3001349

Laverde, A. (2016, febrero 9). NORMAS PARA DISEÑO DE CIRCUITOS IMPRESOS PCB Y ELECTRÓNICA. *|Normas IPC|Certificación|ALDELTA*. https://www.aldeltatec.com/blog-diseno-con-normas-y-certificaciones/normas-pcb-y-electronica/

Laverde, A. (2020a, marzo 18). *Criterios de aceptacion IPC 610 ¿que son y para que sirven?* https://www.aldeltatec.com/blog-diseno-con-normas-y-certificaciones/criterios-de-aceptacion-ipc-610-que-son/

Laverde, A. (2020b, marzo 19). *IPC 600 para aceptación de PCB en la manufactura electrónica*. https://www.aldeltatec.com/blog-diseno-con-normas-y-certificaciones/ipc-600-espanol-e-ipc-610-para-manufactura-de-pcb/

li, voluntad. (2022, marzo 22). ¿Qué es una placa PCB?: Una guía completa para principiantes. *Tecnología: Su socio de servicios de fabricación electrónica de confianza*. https://www.mokotechnology.com/what-is-a-pcb-board/

Liao, X., Lv, S., Li, D., Luo, Y., Zhu, Z., & Jiang, C. (2021). YOLOv4-MN3 for PCB

Surface Defect Detection. *Applied Sciences*, *11*(24), 11701. https://doi.org/10.3390/app112411701

Limitada, E. I. (2024, enero 5). *Electrosoft Ingenieria Limitada*. https://www.pcb.electrosoft.cl/04-articulos-circuitos-impresos-desarrollo-sistemas/01-conceptos-circuitos-impresos/conceptos-circuitos-impresos-pcb.html

Ling, Q., & Isa, N. A. M. (2023). Printed Circuit Board Defect Detection Methods Based on Image Processing, Machine Learning and Deep Learning: A Survey. *IEEE Access*, *11*, 15921-15944. https://doi.org/10.1109/ACCESS.2023.3245093

Murphy, K. P. (2012). *Machine learning: A probabilistic perspective*. MIT Press.

Pacheco, J. A. B. (2023). Fundamentos de Placas de Circuito Impreso. *Vida Científica Boletín Científico de la Escuela Preparatoria No. 4*, *11*(22), 15-17. https://doi.org/10.29057/prepa4.v11i22.11013

Ren, S., He, K., Girshick, R., & Sun, J. (2016a). *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks* (arXiv:1506.01497). arXiv. https://doi.org/10.48550/arXiv.1506.01497

Ren, S., He, K., Girshick, R., & Sun, J. (2016b). *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks* (arXiv:1506.01497; Versión 3). arXiv. http://arxiv.org/abs/1506.01497

Roboflow. (2024, mayo 12). *Predict on a Video, Webcam or RTSP Stream—Roboflow Inference*. https://inference.roboflow.com/quickstart/run_model_on_rtsp_webcam/

Sanín Ramírez, D., & Nates Huertas, J. D. (2021). *Sistema de detección de defectos en placas electrónicas PCBs* (https://repositorio.uniajc.edu.co/handle/uniajc/1687) [Trabajo de grado - Pregrado, Institución Universitaria Antonio José Camacho]. https://repositorio.uniajc.edu.co/handle/uniajc/1687

Shanmugamani, R., & Moore, S. M. (2018). *Deep Learning for Computer Vision: Expert Techniques to Train Advanced Neural Networks Using TensorFlow and Keras*. Packt Publishing. https://books.google.es/books?id=dgdOswEACAAJ

SMTpick and place máquina. (2021, junio 8). https://www.neodenpnp.com/news/how-to-calculate-the-actual-smt-patch-speed-46601492.html

Tech, T. (2024, enero 31). *¿Qué es la Función de Activación?* ¿Qué es la Función de Activación? https://aiofthings.telefonicatech.com/recursos/datapedia/funcion-activacion

What is Computer Vision? | IBM. (2023, diciembre 19). https://www.ibm.com/es-es/topics/computer-vision

YEGE. (2020, noviembre 20). *目标检测（Object Detection）*. https://blog.csdn.net/yegeli/article/details/109861867

Zhang, H., Jiang, L., & Li, C. (2021). CS-ResNet: Cost-sensitive residual convolutional neural network for PCB cosmetic defect detection. *Expert Systems with Applications*, *185*, 115673. https://doi.org/10.1016/j.eswa.2021.115673

Zhang, Y., Xie, F., Huang, L., Shi, J., Yang, J., & Li, Z. (2021). A Lightweight One-Stage Defect Detection Network for Small Object Based on Dual Attention Mechanism and PAFPN. *Frontiers in Physics*, *9*. https://www.frontiersin.org/articles/10.3389/fphy.2021.708097

Zhou, Z. (2016). *Machine Learning (Chinese Edition)*. 清华大学出版社. http ://www. tup .com. cn, http ://www. wqbook. com

斋藤康毅. (2018). *深度学习入门: 基于Python的理论与实现 = Deep learning from Scratch* (Lu Y., Trad.; Di 1 ban). 人民邮电出版社.

Relación de documentos

(X) Memoria 101 páginas

La Almunia, a 04 de 06 de 2024

Firmado: Jiahe Qiu