



EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO



Instituto Tecnológico de Chihuahua II
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

**SISTEMA INTELIGENTE PARA LA IDENTIFICACIÓN DE
FALLAS EN TABLILLAS ELECTRÓNICAS**

TESIS

PARA OBTENER EL GRADO DE

MAESTRO EN SISTEMAS COMPUTACIONALES

PRESENTA

RUBÉN ARTURO DOMÍNGUEZ ALTAMIRANO

DIRECTOR DE TESIS

M.C. ARTURO LEGARDA SÁENZ

CODIRECTOR DE TESIS

DR. HERNÁN DE LA GARZA GUTIÉRREZ

CHIHUAHUA, CHIHUAHUA, MAYO 2022

Dictamen

Chihuahua, Chih., 23 de mayo 2022

M.C. MARÍA ELENEA MARTÍNEZ CASTELLANOS
COORDINADORA DE POSGRADO E INVESTIGACIÓN
PRESENTE

Por medio de este conducto el comité tutorial revisor de la tesis para obtención de grado de Maestro en Sistemas Computacionales, que lleva por nombre "SISTEMA INTELIGENTE PARA LA IDENTIFICACIÓN DE FALLAS EN TABLILLAS ELECTRÓNICAS", que presenta el C. RUBÉN ARTURO DOMÍNGUEZ ALTAMIRANO, hace de su conocimiento que después de ser revisado ha dictaminado la APROBACIÓN de la misma.

Sin otro particular de momento, queda de Usted.

Atentamente
La Comisión de Revisión de Tesis.



M.C. ARTURO LEGARDA SÁENZ
Director de tesis



DR. HERNÁN DE LA GARZA GUTIÉRREZ
Co-Director



M.I.S.C. JESUS ARTURO ALVARADO
GRANADINO
Revisor



DR. ALBERTO CAMACHO RIOS
Revisor

Agradecimientos

A mi esposa Daniela Paizan que me acompaña en este y otros proyectos.

A mis hijos Octavio, Leonardo y Sebastián con los que hemos vivido y viviremos los mejores momentos.

A mis padres Enedelia Altamirano y Oscar Domínguez por todo el apoyo que me han brindado toda mi vida.

A mis hermanos que me han apoyado durante toda mi educación.

Al Consejo Nacional de Ciencia y Tecnología, por el apoyo económico para la realización de mis estudios de posgrado.

A todos, Gracias.

Rubén Arturo Domínguez Altamirano

Resumen

En este trabajo de investigación se realiza el diseño y evaluación de un sistema inteligente (SI) para la identificación de posibles defectos en muestras físicas que son capturadas a imágenes por un equipo de inspección automática (AOI) modelo FX940 UV. Para lograrlo, se utilizan técnicas de inteligencia artificial que buscan aprovechar y ampliar el beneficio de un equipo AOI en una planta de manufactura, con el diseño y creación de su propio dataset ajustado al modelo que produce la planta de manufactura, en este caso el área de conformal coating. Además, se utiliza una técnica de transferencia de aprendizaje (Transfer learning) que aprovecha el entrenamiento previo de una red neuronal Mask R-CNN que tiene una velocidad más rápida de entrenamiento y pruebas. Esta investigación muestra que aún es posible obtener mayores beneficios de los equipos que se tienen actualmente en la industria y obtener mayor eficiencia en los procesos; así como que los actuales y futuros equipos que se encuentran disponibles en el mercado pueden mejorar y aprovechar las nuevas tecnologías.

Abstract

In this research work, the design and evaluation of an intelligent system (IS) for the identification of possible defects in physical samples that are captured in images by an automatic inspection equipment (AOI) model FX940 UV is carried out. To achieve this, artificial intelligence techniques are used to take advantage and extend the benefit of an AOI equipment in a manufacturing plant, with the design and creation of its own dataset adjusted to the model produced by the manufacturing plant, in this case the conformal coating area. In addition, a transfer learning technique is used that takes advantage of the previous training of a Mask R-CNN neural network that has a faster training and testing speed. This research shows that it is still possible to obtain greater benefits from the equipment currently available in the industry and obtain greater efficiency in the processes; as well as that the current and future equipment available in the market can improve and take advantage of new technologies.

Contenido

I. INTRODUCCIÓN	1
1.1 Introducción	1
1.2 Definición del problema.....	2
1.3 Alcances y limitaciones.....	4
1.4 Justificación.....	4
1.5 Objetivos	4
1.6 Hipótesis.....	5
II. ESTADO DEL ARTE.....	6
III. MARCO TEÓRICO	8
IV. DESARROLLO	18
4.1 Análisis.....	18
4.2 Diseño	19
4.2.1 Recolección de imágenes con defectos para entrenamiento y validación.....	20
4.2.2 Identificación de los defectos en <i>VGG</i>	20
4.2.3 Entrenamiento del modelo	21
4.2.4 Validación del modelo.	26
4.2.5 Preprocesamiento de los datos	27
4.2.6 Ejecutar el modelo en imágenes y hacer predicciones	39
4.3 Implementación.....	40
4.4 Pruebas	43
V. RESULTADOS Y DISCUSION.....	44
5.1 Muestras seleccionadas para la experimentación.....	44
5.2 Resultados de las pruebas en distintos estudios.	45
5.2.1 Detección con Keras para CNN	45
5.2.2 Detección usando matrices de transformación.....	46
5.2.3 Detección usando distintas Redes Neuronales	48
5.2.4 Detección final con <i>mask CNN</i>	50
VI. CONCLUSIONES	52
VII. BIBLIOGRAFIA.....	54

Índice de Figuras

Figura 1.1 Reporte Fallas de producción.	2
Figura 1.2 Reporte Escapes a Finales.....	3
Figura 3.1 Fases de la metodología RUP.	9
Figura 3.2 Cámaras AOI e imagen obtenida.	11
Figura 3.3 Neurona biológica.....	12
Figura 3.4 Tipos de neuronas.	13
Figura 3.5 Perceptrón.	13
Figura 3.6 Feed forward networks.	13
Figura 3.7 Convolutional Neural Networks.	14
Figura 3.8 Ejemplo de segmentación de Mask R-CNN.	14
Figura 4.1 Imagen de una tablilla completa con conformal.	18
Figura 4.2 Imagen con etiquetas y su extracción en VGG Image Annotator.....	19
Figura 4.3 Diagrama de flujo para crear el modelo.....	20
Figura 4.4 Identificación de clases.....	21
Figura 4.5 Imágenes con archivo de anotaciones (RPN).	21
Figura 4.6 Barra de progreso con valor 1.....	23
Figura 4.7 Progreso en épocas con valor 2.....	23
Figura 4.8 Configuraciones de entrenamiento.	24
Figura 4.9 Carpetas de entrenamiento.....	24
Figura 4.10 Resultado de carpetas de entrenamiento.	25
Figura 4.11 Clases y archivos en Dataset.....	25
Figura 4.12 Inicio del entrenamiento.	26
Figura 4.13 Resultados del entrenamiento.	26
Figura 4.14 Resultados de validación del modelo.....	27
Figura 4.15 Código para cargar el modelo.....	27
Figura 4.16 Imágenes y su máscara.	28
Figura 4.17 Imagen con máscara.....	28
Figura 4.18 Valores de los cuadros delimitadores.	29
Figura 4.19 Cuadros delimitadores en defectos.	29
Figura 4.20 Imagen redimensionada.	30
Figura 4.21 Imagen y sus mascararas con redimensionamiento.	31
Figura 4.22 Imagen con redimensionamiento y máscara.	32
Figura 4.23 Imagen con máscara aumentada.	32
Figura 4.24 Implementación de anclas.....	33
Figura 4.25 Anclas de una celda.	34
Figura 4.26 Generador de datos de anclas.....	34
Figura 4.27 Código de evaluación de anclas.....	35
Figura 4.28 Anclas positivas.	35
Figura 4.29 Anclas negativas.	36
Figura 4.30 Anclas neutrales.....	36
Figura 4.31 ROIs o RPNs.....	37
Figura 4.32 Código detección de ROIs, mascararas y cajas.....	38
Figura 4.33 Detección de defectos y fondos en imágenes muestra.....	38
Figura 4.34 Resultado de la predicción.....	39
Figura 4.35 Zoom del resultado de la predicción.....	39
Figura 4.36 Ventana de instalación de Anaconda.	40
Figura 4.37 Consola de Anaconda.	40

Figura 4.38 Comando de instalación de librerías.....	41
Figura 4.39 Carpeta de pruebas.....	41
Figura 4.40 Archivos y carpetas de la prueba.	41
Figura 4.41 Carpeta de imágenes para evaluación.....	42
Figura 4.42 Imagen de resultado de la evaluación.....	42
Figura 4.43 Aplicación para la evaluación del prototipo.	43
Figura 4.44 Evaluación de pruebas en primeras muestras.	43
Figura 5.1 Imagen completa de una inspección por componentes.....	44
Figura 5.2 Imagen completa de una inspección por conformal.....	45
Figura 5.3 Precisión del entrenamiento y la validación	45
Figura 5.4 Pérdida de entrenamiento y validación	46
Figura 5.5 Resultados de la predicción.	46
Figura 5.6 Alineación de imágenes.	47
Figura 5.7 Identificación y separación de plantillas.....	47
Figura 5.8 Detección de diferencias entre tarjeta muestra y tarjeta evaluada.	48
Figura 5.9 Resultados del entrenamiento y validación.....	48
Figura 5.10 Resultados del entrenamiento y validación.....	49
Figura 5.11 Ampliación y rotación de imágenes.....	49
Figura 5.12 Resultados del entrenamiento y validación.....	50
Figura 5.13 Rendimiento de las distintas redes.....	50
Figura 5.14 Predicción de muestras finales.....	51
Figura 6.1 Muestra inicial.	52
Figura 6.2 Muestra final con conformal.....	53

I. INTRODUCCIÓN

1.1 Introducción

El diseño y desarrollo de sistemas que resuelvan y/o apoyen a la solución de los problemas de manera inteligente, mediante la toma de decisiones se les llama Sistemas Inteligentes (SI), siendo estos completamente distintos de los Sistemas Automáticos (SA), con los que suelen confundirlos. Los Sistema Automáticos automatizan una tarea, mas no resuelven un problema que se les presente durante la ejecución de sus actividades y por esto no se consideran inteligentes. Para dotar a un SA de las características de inteligencia, la inteligencia artificial (IA) tiene varias técnicas y campos de aplicación como aprendizaje automático (Machine Learning), lógica difusa (FuzzyLogic), redes neuronales artificiales (Artificial Neural Networks), sistemas expertos (Expert Systems), redes Bayesianas (Bayesian Networks), algoritmos genéticos (GeneticAlgorithms), redes semánticas (Semantic Networks), procesamiento del lenguaje natural (Natural Language Processing), minería de datos (Data Mining), visión artificial, entre otras más (D'Aquila, 2005).

Por su parte los SI que analizan imágenes suelen utilizar algunas técnicas de las anteriores como Visión Artificial y Redes Neuronales, en donde la Visión Artificial consiste en métodos para adquirir, procesar, analizar y comprender imágenes; mientras que las Redes Neuronales tratan de imitar a las redes neuronales biológicas del ser humano, se constituyen por conexiones entre neuronas artificiales (Haykin, 1994, Hertz, 1991).

El desarrollo se realiza en una empresa manufacturera de tablillas electrónicas donde realizan la fabricación de controles de iluminación, balastos y algunos productos del ámbito de la iluminación, algunos de éstos son los interruptores que van en casas, edificios y muestran los niveles de atenuación en la iluminación, según el modelo y especificaciones del cliente, así como controladores de persianas. La investigación, da respuesta a la problemática de los falsos rechazos en tablillas electrónicas de producción de SMT (*Surface Mount Technology*).

El desarrollo de este SI presentado en este trabajo, intenta complementar los equipos

I. INTRODUCCIÓN

automáticos que están actualmente en la empresa y que se conocen como sistemas de Inspección Óptica Automatizada (AOI).

Básicamente estos equipos de inspección revisan el material para determinar si se presenta una “Falla” que consiste en la ausencia de componentes, están desplazados o es un “Falso” que representa una alerta, pero la tablilla si es funcional. El SI reduce el porcentaje de escapes a menos del 10% y el tiempo de inspección, además de que aprende de sus errores de detección. Esto representa un apoyo al personal de inspección reduciendo el tiempo invertido en las revisiones.

1.2 Definición del problema

Actualmente el departamento de calidad realiza la medición de la eficiencia en inspección, detección y falsos en tablillas de producción de manera diaria, semanal y mensual, presentando problemas en algunos casos en que se escapan fallas reales y no son detectadas por el equipo o las personas de inspección no lo detectan. Las detecciones erróneas representan incrementos en el ciclo de producción así como la ocupación del personal en labores de inspección, además del incremento en gastos monetarios cuando hay que reparar una falla que se escapó a la siguiente área de ensamble como se muestra en la figura 1.1 y figura 1.2.

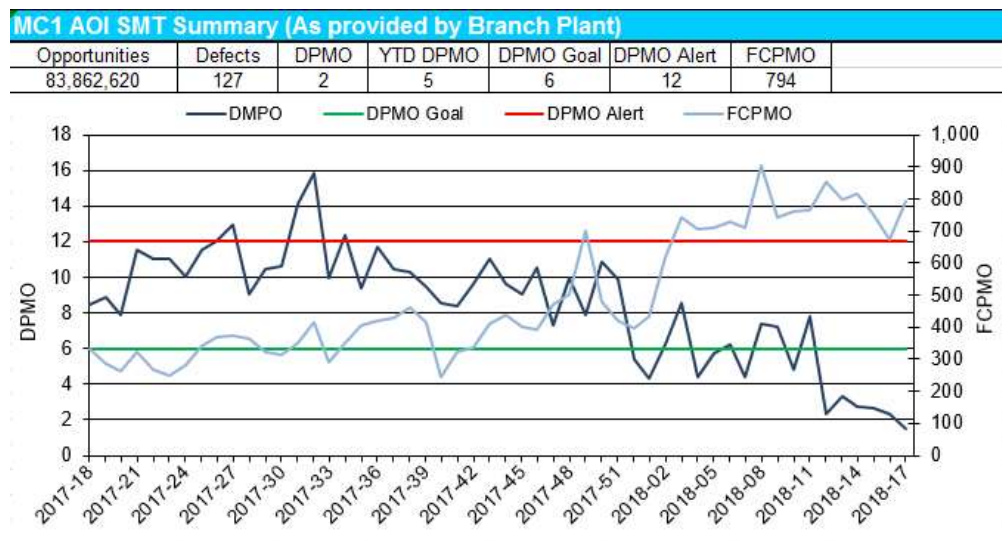


Figura 1.1 Reporte Fallas de producción.

I. INTRODUCCIÓN

En el primer reporte (ver figura 1.1), se encuentran los datos de los defectos por millón en SMT (DPMO), mostrando la tendencia de años anteriores, metas y alertas.

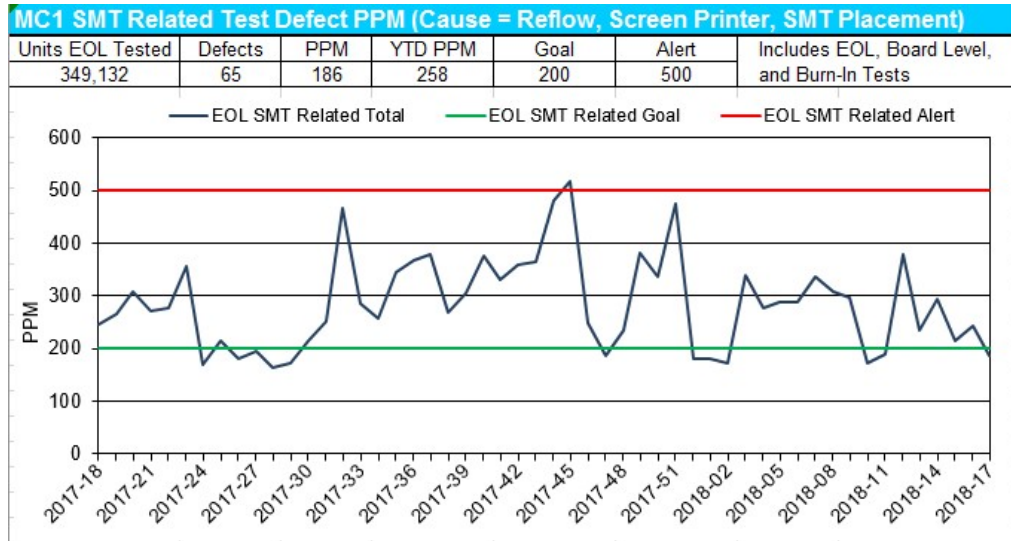


Figura 1.2 Reporte Escapes a Finales.

Mientras que el segundo reporte (ver figura 1.2), indica los escapes de defectos que han ocurrido desde SMT a la siguiente área de producción (Finales), en él están las detecciones por equipos de pruebas que realiza finales, y el métrico de piezas por millón que tuvieron falla atribuible a SMT (PPM). Por lo que las preguntas de investigación que se plantean son:

¿Cómo mejorar el proceso de detección de fallas y de falsos en las tablillas electrónicas?

¿Cómo disminuir el tiempo de inspección que se aplican a las tablillas a través de AOI?

1.3 Alcances y limitaciones

1.3.1. Alcances

- Este sistema abarcará el análisis de las tablillas a través de imágenes capturadas por la máquina.
- El sistema identificará los componentes previamente entrenados, presentes en las tablillas.
- Identificará las fallas en los componentes.

1.3.2. Limitaciones

- Aplicará para un solo modelo de tablilla.
- El tiempo de desarrollo del sistema está limitado a dos años.

1.4 Justificación

- Existen sistemas automatizados como el AOI, pero requieren de personal para la revisión de los componentes internos de la tablilla. El Sistema Inteligente después de un tiempo de entrenamiento automáticamente identificará las fallas.
- Este sistema puede fácilmente ser implementado para otros modelos de tablillas, ya que el proceso para ser programado consiste en entrenar el sistema con el nuevo modelo de tablillas y realizar ajustes en los análisis.

1.5 Objetivos

1.5.1. Objetivo General

Desarrollar un sistema que inspeccione tablillas electrónicas automáticamente para determinar si es un fallo real o un falso rechazo utilizando las técnicas como toma de decisiones, visión computacional y redes neuronales para mejorar los tiempos de inspección e incrementar los niveles de producción.

1.5.2. Objetivos Específicos

- Agilizar la entrega de información de posibles defectos en tablillas.
- Reducir el porcentaje de escapes al área de finales en <100 PPM a diferencia del método actual de solo AOI con un resultado >200 PPM.
- Reducir los tiempos de inspección y espera en 4 segundos por tablilla electrónica.

1.6 Hipótesis

Con el análisis de las imágenes obtenidas, el sistema deberá ser capaz de tomar decisiones rápidas y con mayor precisión para disminuir los escapes de las fallas y falsos rechazos en las tablillas electrónicas.

Variables

- Identificación de defectos.
- Escapes de falsos reales.
- Falsos rechazos.

II. ESTADO DEL ARTE

Actualmente se utilizan equipos de AOI para detectar fallas en tablillas de producción en los cuales una persona tiene que ajustar valores para detectar posibles fallas, pero no son 100% fiables ya que tienen que ser ajustados, por ejemplo en un caso pueden ajustarlo por alguna circunstancia y cuando vuelven a producir el mismo producto después hay que ajustarlo de nuevo, estos ajustes se basan en filtros de luz que se observan en las imágenes, en escalas de gris o blanco y negro, cuando el equipo detecta una posible falla, muestra en pantalla un mensaje, aquí es cuando las personas inspectoras de calidad llegan a revisar si fue en realidad un defecto real o solo un falso y el producto estaba bien.

Inicialmente los primeros pioneros en el área de la IA, fueron Alan Turing un matemático que dirigió el proyecto de la descryptación de información encriptada de los nazis, que consistía en la famosa maquina Enigma. Aunque Turing no propuso el nombre de IA, si elaboró en 1950 un criterio de gran utilidad que caracteriza a cualquier maquina inteligente, y que comprueba que la maquina está preparada para responder a preguntas, de manera que un humano no sepa que es una máquina, a esta prueba se le conoce como la Prueba Turing (D'Aquila, 2005).

No fue hasta 1956 donde un grupo de investigadores se reunieron en Dartmouth College en un congreso de 1 mes, durante este se dieron cuenta que las computadoras progresaban de manera rápida y que de algún modo superarían la inteligencia humana, donde tiempo después inventaron el término de IA.

En 2006 G. Acciani, G. Brunetti y G. Fornarelli desarrollaron un proceso para identificar defectos en circuitos integrados (IC) mediante el uso de un sistema de red neuronal múltiple colocados en el área de SMT, el sistema analiza imágenes de las soldaduras en el IC, extrayendo las áreas de interés, clasificando en 5 niveles la calidad de la soldadura con respecto a la pasta de soldar.

II. ESTADO DEL ARTE

Un ejemplo en México de aplicación de redes neuronales y visión artificial es el robot humanoide MEX-ONE, el objetivo de este robot es que aprenda y adquiera habilidades para desenvolverse exitosamente en un determinado entorno, mide 1.05 metros y pesa 15 kilogramos. Además de que el humanoide es capaz de reaccionar y aprender de manera autónoma mediante un cerebro artificial conformado por redes sensoriales (sensores que pueden monitorizar parámetros del entorno) y microprocesadores (circuitos integrados centrales más complejos de las computadoras) de última generación, que gasta poca energía, construido con una inversión de 100 mil dólares (Cyd.conacyt.gob.mx, 2018).

En 2013 Mihály J., Ákos B., László J., Richárd G. and Tibor T., realizaron un estudio acerca de las AOI disponibles en el mercado y sus posibilidades de detección y posibles mejoras en su desempeño, describiendo algunos aspectos importantes como los siguientes:

La inspección óptica automática (AOI) o la inspección visual automatizada (AVI) es un proceso de control. Evalúa la calidad de los productos fabricados con la ayuda de información visual. No sólo es la productividad, pero también se espera una alta calidad. Los requisitos de calidad de los dispositivos electrónicos ya han sido estandarizados, por ejemplo, las normas IPC, ANSI-JSTD. Hoy en día, la capacidad de las modernas máquinas de fabricación llega a 6σ como es habitual y 5σ para los más estrictos. Aun así, los procesos de fabricación se mantienen bajo supervisión constante. Todavía hay ocasiones en las que incluso un ensamblaje moderno no crea dispositivos completamente operativos (Mihály J, 2013).

Debido a sus capacidades y propiedades, la mayoría de los sistemas AOI se utilizan como sistemas en línea. La principal ventaja de estos sistemas es su capacidad para detectar fallos antes y no sólo cuando el producto ha sido montado. Los sistemas AOI se pueden utilizar inspeccionar la calidad en cada etapa del proceso de fabricación del dispositivo electrónico. Por consiguiente, existen ventajas financieras reales al utilizar estos sistemas, ya que cuanto antes se detecte un fallo, menor será la probabilidad de que se fabriquen dispositivos de recogida de basuras. Debido a reducción de la escala de los componentes y aumento de la densidad, la inspección óptica sólo es posible ahora con la ayuda de la visión artificial en lugar de la inspección manual.

III. MARCO TEÓRICO

En este capítulo se presenta la exposición organizada de los elementos teóricos generales o particulares, así como la explicación de los conceptos básicos en que se apoya el proyecto con el objeto de comprender las relaciones y aspectos fundamentales de los fenómenos y procesos de la realidad.

3.1 Ingeniería de software

La Ingeniería del Software es una disciplina o área de la Informática o Ciencias de la Computación, que ofrece métodos y técnicas para desarrollar y mantener software de calidad que resuelven problemas de todo tipo. Hoy día es cada vez más frecuente la consideración de la Ingeniería del Software como una nueva área de la ingeniería, y el ingeniero del software comienza a ser una profesión implantada en el mundo laboral internacional, con derechos, deberes y responsabilidades que cumplir, junto a una, ya, reconocida consideración social en el mundo empresarial y, por suerte, para esas personas con brillante futuro (Pressman, Roger S, 2005).

3.2 Metodologías de desarrollo de software

Se entiende como Desarrollo Ágil de Software a un paradigma de Desarrollo de Software basado en procesos ágiles. Los procesos ágiles, conocidos anteriormente como metodologías livianas, intentan evitar los tortuosos y burocráticos caminos de las metodologías tradicionales enfocándose en la gente y los resultados. El software desarrollado en una unidad de tiempo es llamado una iteración, la cual debe durar de una a cuatro semanas. Cada iteración del ciclo de vida incluye: planificación, análisis de requerimientos, diseño, codificación, revisión y documentación (Pressman, Roger S, 2005).

3.3 Metodología RUP

El Proceso Unificado fue desarrollado por Philippe Kruchten, Ivar Jacobson como el proceso complementario al UML. El RUP es un armazón de proceso y como tal puede acomodar una

III. MARCO TEÓRICO

gran variedad de procesos. El RUP puede usarse en un estilo muy tradicional de cascada o de una manera ágil. Como resultado se puede usar el RUP como un proceso ágil o como un proceso pesado - todo depende de cómo lo adapte a su ambiente. Se divide en 4 fases bien conocidas llamadas Concepción, Elaboración, Construcción y Transición (ver figura 3.1).

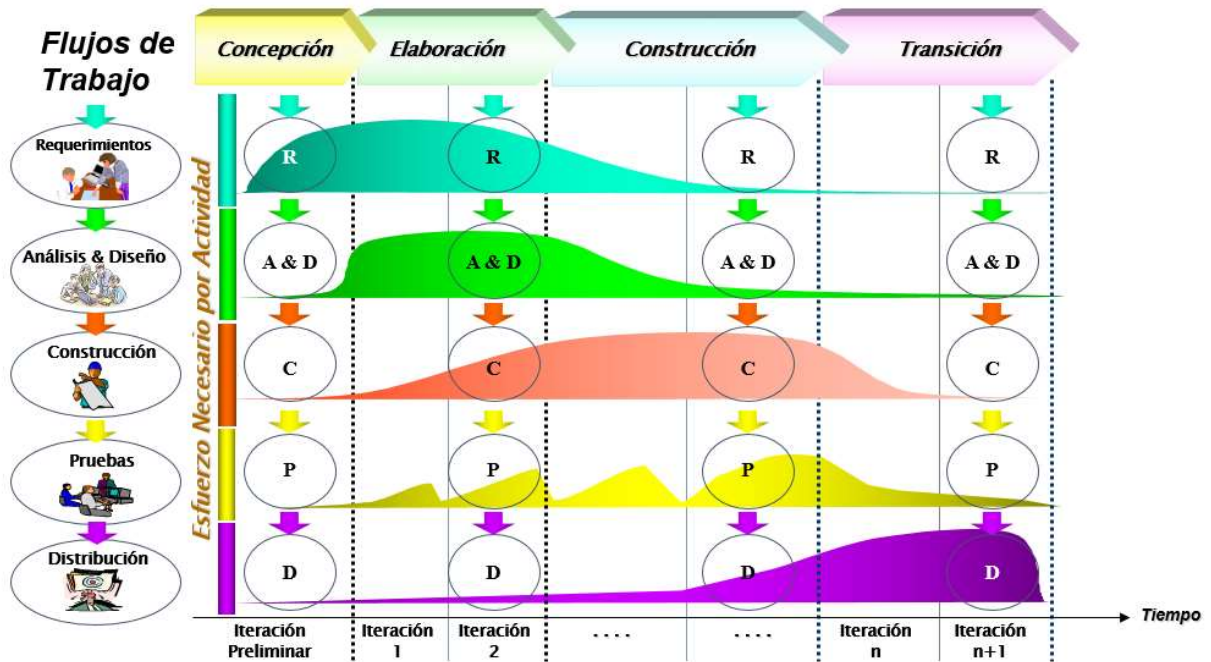


Figura 3.1 Fases de la metodología RUP.

Esas fases se dividen en iteraciones, cada una de las cuales produce una pieza de software demostrable. La duración de cada iteración puede extenderse desde dos semanas hasta seis meses. Entre cada etapa se encuentran ciertos Hitos que son muy útiles para continuar con el proyecto.

Requerimientos, son indispensables para el desarrollo del proyecto los requisitos iniciales que se necesitarán, estos deben ser consultados y cotejados en una junta en la cual se especificarán todas las necesidades. Una vez que se obtienen estos datos se desarrolla el tiempo estimado planificando las próximas etapas.

Análisis. En esta etapa se debe entender y comprender de forma detallada cuál es la

III. MARCO TEÓRICO

problemática para resolver, verificando el entorno en el cual se encuentra dicho problema, de tal manera que se obtenga la información necesaria y suficiente para afrontar su respectiva solución. Esta etapa es conocida como la del qué se va a solucionar. Esta fase comprendería desde la posible obtención de unos objetivos o requisitos iniciales para determinar la viabilidad del sistema y escrutar las distintas alternativas de solución, pasando por la elaboración del catálogo de requisitos, hasta la realización de casos de uso, prototipos de pantallas e informes, como una primera especificación del plan de pruebas.

Diseño. El diseño del software se enfoca en cuatro atributos distintos del programa: la estructura de los datos, la arquitectura del software, el detalle procedimental y la caracterización de la interfaz. El proceso de diseño traduce los requisitos en una representación del software con la calidad requerida antes de que comience la codificación, representa los requerimientos en una forma que permita la codificación del producto (además de una evaluación de la calidad previa a la etapa de codificación). Al igual que los requerimientos, el diseño es documentado y se convierte en parte del producto de software.

Codificación o Construcción. El diseño debe traducirse en una forma legible para la máquina. La etapa de codificación realiza esta tarea. Si el diseño se realiza de una manera detallada la codificación puede realizarse mecánicamente. Es la creación tanto del programa lógico como la construcción de las interfaces necesarias para el funcionamiento del sistema. En esta fase se realiza la construcción del sistema de información y las pruebas relacionadas con dicho proceso, como son las unitarias, integración y de sistema, así como otras actividades propias de las etapas finales de un desarrollo como es la realización de la carga inicial de datos (si bien en muchos casos se deja esto para cuando el producto está en producción) y/o la construcción del procedimiento de migración.

Pruebas. Una vez que se ha generado el código comienza la prueba del programa. La prueba se centra en la lógica interna del software, y en las funciones externas, realizando pruebas que aseguren que la entrada definida produce los resultados que realmente se requieren, se solucionan errores de “comportamiento” del software y se asegura que las entradas definidas

producen resultados reales que coinciden con los requerimientos especificados.

Implementación o Distribución. Es la fase en donde el usuario final ejecuta el sistema, para ello el o los programadores ya realizaron exhaustivas pruebas para comprobar que el sistema no falle (Pressman, Roger S, 2005).

3.4 AOI (Automated Optical Inspection)

Es una máquina que mediante el proceso de utilizar la tecnología de varias cámaras y la computadora para inspeccionar visualmente una tablilla electrónica para revisar la correcta colocación de los componentes y la soldadura (ver Figura 3.2).

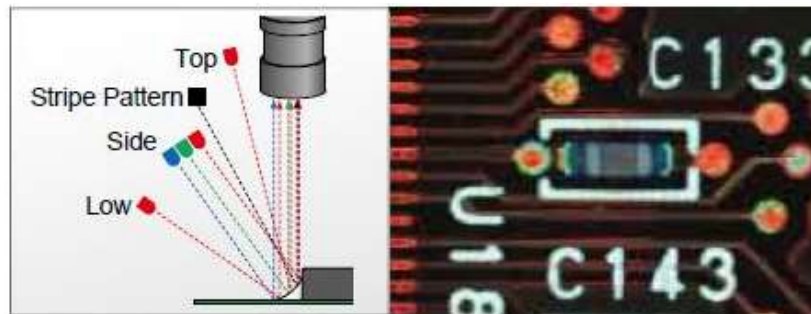


Figura 4.2 Cámaras AOI e imagen obtenida.

3.5 Medibles

Dentro del área de SMT es importante contar con medibles para determinar la eficiencia que se tiene y tomar las medidas necesarias en caso de no cumplir con los objetivos establecidos. En este departamento se tienen los siguientes medibles:

Yield Mide el porcentaje de tablillas que pasan por el proceso de producción sin ser scrap, ni con reparaciones, no requieren volver a pasar por algún proceso, estas ya cumplen con las pruebas necesarias.

DPMO (Defects Per Million Opportunities o Defectos por Millón de Oportunidades)

Se trata de una medición universal de la efectividad de un proceso, que tiene en cuenta el número total de defectos, así como el número total de oportunidades de defectos y, a continuación, calcula matemáticamente una clasificación para el rendimiento del proceso.

False Call (Llamadas en falso o falsos rechazos). Esta es una categoría utilizada por un

operador, cuando la máquina AOI identifica lo que cree que podría ser un defecto, pero que, al revisarlo, resulta aceptable. Esta operadora clasificará esta situación como una "llamada falsa". **FCPMO (False Calls Per Million Opportunities o Falsas Llamadas por Millón de Oportunidades)**. Esta es una medida de la efectividad del proceso AOI, que toma en cuenta el número total de llamadas falsas, así como el número total de oportunidades de defectos, y luego calcula matemáticamente una calificación de llamada falsa para el desempeño del proceso.

3.6 Redes Neuronales

Son modelos que tratan de simular las redes neuronales biológicas de los animales superiores, en particular, el hombre. Están constituidas por un alto grado de conexionismo de neuronas artificiales, de acuerdo con una cierta topología o arquitectura (Haykin, 1994, Hertz, 1991).

En la figura 3.3 se muestran los aspectos importantes a destacar en una neurona biológica. El soma conforma el cuerpo de la neurona. Las dendritas, los filetes finos, constituyen las entradas. El axón es el filete grueso de salida. Finalmente, las sinapsis, conexiones entre el axón de una neurona con las dendritas de otras, tienen distinta intensidad, y constituyen la memoria del sistema.

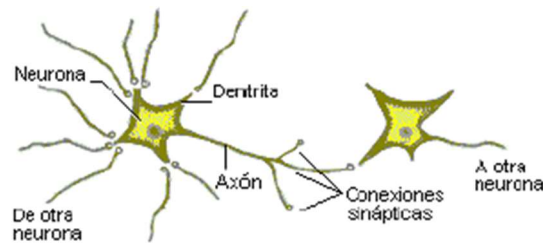


Figura 5.3 Neurona biológica.

Las propiedades de una red neuronal dependen del modelo de neurona y de la arquitectura utilizada, el concepto de **arquitectura** referida a redes neuronales hace mención no solo al número de capas neuronales o al número de neuronas en cada una de ellas, sino a la conexión entre neuronas o capas, al tipo de neuronas presentes e incluso a la forma en la que son entrenadas, a continuación se mencionan las más básicas (los nombres se dan en inglés, tal y como son frecuentemente mencionadas). El tipo de neurona en las figuras viene dado por la

siguiente figura 3.4:

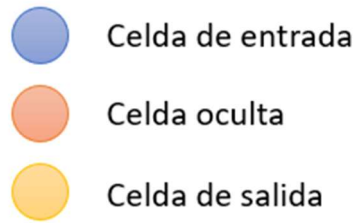


Figura 6.4 Tipos de neuronas.

Perceptrón

El perceptrón (ver Figura 3.5) es la arquitectura más sencilla posible: Consta de dos celdas de entrada y una de salida. Los datos de entrada llegan a las celdas de entrada, pasan a la celda de salida, se les aplica la media ponderada y la función de activación de la celda, y se devuelve el valor resultante.

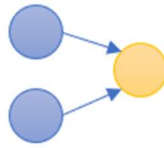


Figura 7.5 Perceptrón.

Feed forward networks

Este tipo de redes son una extensión del perceptrón. Constan de varias capas de neuronas, "fully connected" (es decir, una celda está conectada con todas las celdas de la siguiente capa ver Figura 3.6), hay al menos una capa oculta, la información circula de izquierda a derecha ("feed forward") y el entrenamiento de la red se suele realizar mediante back-propagation.

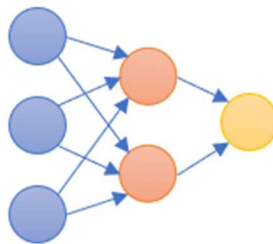


Figura 8.6 Feed forward networks.

3.7 Visión Artificial

Según la Asociación de Imágenes Automatizadas (AIA), la visión artificial incluye todas las aplicaciones industriales y no industriales donde una combinación de hardware y software brindan una guía operativa a dispositivos en la ejecución de sus funciones en base a la captura y el procesamiento de imágenes. Los sistemas de visión artificial se basan en sensores digitales protegidos dentro de cámaras industriales con ópticas especializadas en adquirir imágenes, para que el hardware y el software puedan procesar, analizar y medir diferentes características para tomar decisiones (Cognex.com, 2018).

3.8 VGG image annotator

VGG Image Annotator es un software de anotación manual simple y autónoma para imagen, audio y video. VIA se ejecuta en un navegador web y no requiere ninguna instalación o configuración. El software VIA completo cabe en una sola página HTML autónoma de tamaño inferior a 400 Kilobyte que se ejecuta como una aplicación fuera de línea en la mayoría de los navegadores web modernos.

VIA es un proyecto de código abierto basado únicamente en HTML, JavaScript y CSS (sin dependencia de bibliotecas externas). VIA se desarrolla en el Grupo de Geometría Visual (VGG) y se publica bajo la licencia de la cláusula BSD-2 que permite que sea útil tanto para proyectos académicos como para aplicaciones comerciales. (Abhishek Dutta and Andrew Zisserman. 2019)

3.9 Python

Python es un lenguaje de programación interpretado, interactivo y orientado a objetos. Incorpora módulos, excepciones, tipificación dinámica, tipos de datos dinámicos de muy alto nivel y clases. Admite múltiples paradigmas de programación más allá de la programación orientada a objetos, como la programación procedimental y funcional. Python combina una notable potencia con una sintaxis muy clara. Tiene interfaces para muchas llamadas y bibliotecas de sistemas, así

como para varios sistemas de ventanas, y es extensible en C o C++. También es utilizable como lenguaje de extensión para aplicaciones que necesitan una interfaz programable. Por último, Python es portátil: funciona en muchas variantes de Unix, incluyendo Linux y MacOS, y en Windows. (Python Software Foundation, 2020)

3.10 Machine learning

El aprendizaje automático es la ciencia de hacer que los ordenadores actúen sin ser programados explícitamente. En la última década, el aprendizaje automático nos ha dado coches autoconductores, reconocimiento de voz práctico, búsqueda efectiva en la web y una comprensión mucho mejor del genoma humano. El aprendizaje automático está tan extendido hoy en día que probablemente lo uses docenas de veces al día sin saberlo. Muchos investigadores también piensan que es la mejor manera de avanzar hacia la IA a nivel humano. (coursera.org, 2020)

3.11 Keras

Es un API diseñado para los seres humanos, no para las máquinas. Keras sigue las mejores prácticas para reducir la carga cognitiva: ofrece APIs consistentes y simples, minimiza el número de acciones de usuario requeridas para casos de uso común, y proporciona mensajes de error claros y procesables. También cuenta con una amplia documentación y guías para el desarrollador. (Keras.io, 2020)

3.12 TensorFlow

Es una interfaz para expresar algoritmos de aprendizaje automático y una implementación para ejecutar dichos algoritmos. Un cálculo expresado con TensorFlow se puede ejecutar con poco o ningún cambio en una amplia variedad de sistemas heterogéneos, que van desde dispositivos móviles como teléfonos y tabletas hasta sistemas distribuidos a gran escala de cientos de máquinas y miles de dispositivos computacionales como tarjetas GPU. . El sistema es flexible y se puede utilizar para expresar una amplia variedad de algoritmos, incluidos algoritmos de entrenamiento e inferencia para modelos de redes neuronales profundas, y se ha utilizado para

III. MARCO TEÓRICO

realizar investigaciones y para implementar sistemas de aprendizaje automático en producción en más de una docena de áreas de informática y otros campos, incluido el reconocimiento de voz, la visión por computadora, la robótica, la recuperación de información, el procesamiento del lenguaje natural, la extracción de información geográfica y el descubrimiento computacional de fármacos. La API de TensorFlow y una implementación de referencia se lanzaron como un paquete de código abierto bajo la licencia de Apache 2.0 en noviembre de 2015. (TensorFlow.org, 2020)

IV. DESARROLLO

Se utilizó una metodología RUP para el proyecto, ya que se puede agilizar el método cascada, utilizando sus iteraciones como ventaja para identificar posibles fallos o falta de análisis durante el proceso de desarrollo, apoyado por un desarrollo en Python aprovechando las facilidades que tiene este lenguaje para la inteligencia artificial y tratamiento e identificación de datos, imágenes.

4.1 Análisis

Antes de iniciar con la elaboración de la aplicación, lo primero es analizar el área donde se utilizará. En el momento que se logró el acceso a los recursos de SMT, lo primero que se hizo fue tener pláticas con el Ing. Alejandro Ramírez, el encargado de SMT. Una vez que se tuvo la información necesaria, se les ofreció un diseño de las tablas, tratando de que el proceso se haga mucho más rápido y que los técnicos no vean este proceso de una manera complicada. Se detectaron algunos casos en los que es difícil identificar si es una llamada es falsa o real, por lo que el sistema podría tener una pequeña variación a la hora de encontrarse con esos casos. En la Figura 4.1 se muestra una imagen de una tarjeta en tamaño completo.

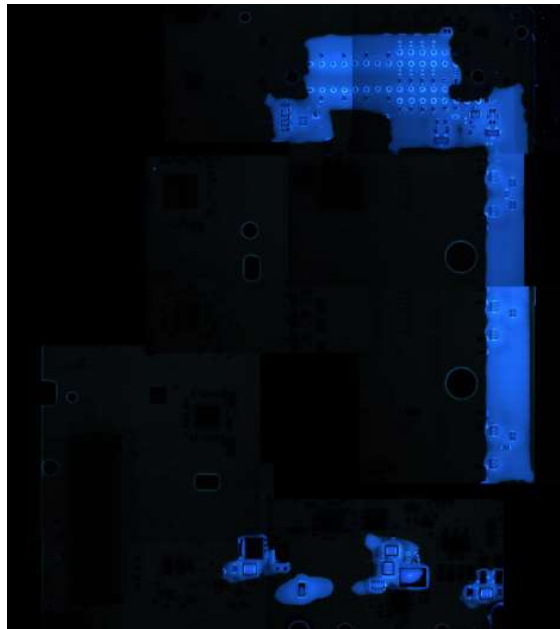


Figura 11.1 Imagen de una tablilla completa con *conformal*.

IV. DESARROLLO

Se utilizó una red MASK R-CNN que devuelve los cuadros de delimitación del objeto, lo que es de beneficio para el proyecto, ya que se puede empezar de no tener un *dataset* para hacer cualquier implementación y comenzar con la creación de un dataset personalizado para que pueda ser retroalimentado durante el avance del proyecto, para lograr todo lo anterior se utilizó la herramienta *VGG Image Annotator*, la que permite indicar y categorizar las imágenes desde un defecto hasta lo aceptable, así esta herramienta le transfiere la información de coordenadas donde el sistema puede iniciar a buscar y crear su propio criterio para crear el dataset, como se ve en la Figura 4.2. Inicialmente se utilizaron 22 muestras para verificar si este método sería útil para el proyecto, al verificar que se tenía una eficiencia del 65% con estas muestras entrenadas de manera no restrictiva lo que permitía algún porcentaje de error, se decidió incrementar la cantidad de muestras a 40, ya que no se podían obtener demasiadas con defectos, debido a que no es normal que existan tantos defectos en el proceso, y también ser más restrictivo al indicar los defectos en las imágenes, todas las muestras y equipo AOI fueron proporcionados por la empresa ya que estos estaban en sus instalaciones por uso diario en sus actividades.



Figura 12.2 Imagen con etiquetas y su extracción en *VGG Image Annotator*.

4.2 Diseño

El proyecto a desarrollar busca automatizar algunos procesos actuales, minimizando el tiempo que tarda en llevarse a cabo de forma manual la inspección. Terminado el análisis se realizaron investigaciones y diseños de experimentos para encontrar la mejor forma de obtener mejores resultados (ver figura 4.3).

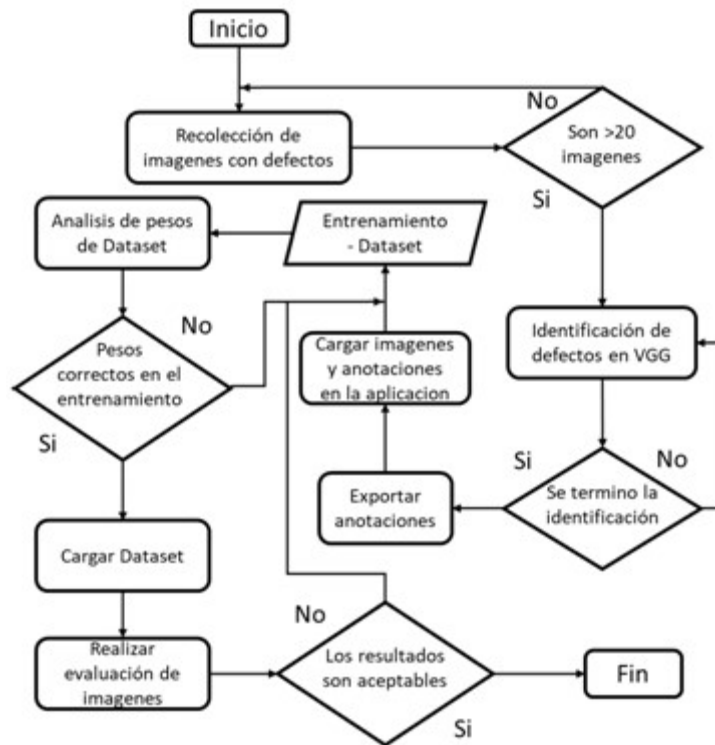


Figura 13.3 Diagrama de flujo para crear el modelo.

4.2.1 Recolección de imágenes con defectos para entrenamiento y validación.

Se obtuvieron la mayor cantidad de imágenes con defectos, ya que se requería revisar las imágenes una por una de la producción e identificar si era bueno o mala y segregar las imágenes para el proyecto, inicialmente se recolectaron 22 para validar si era viable esta opción de crear el dataset, después se incrementó la cantidad a 40 y 10 para validación.

4.2.2 Identificación de los defectos en VGG.

Al realizar una primera fase de prueba se obtuvo un resultado del 65% de efectividad, por lo que se decidió incrementar las muestras y ser más restrictivo en la identificación de los defectos. El primer paso es la importación de las imágenes a VGG, después hay que crear una clase llamada “name” e identificar los defectos mediante puntos para que sea más restrictivo y llamándolos “damage” en la clasificación, como se puede apreciar en la Figura 4.4. Se realizó este procedimiento con las 40 imágenes y cada una varía en la cantidad de defectos presentes, desde 1 hasta 6.

IV. DESARROLLO

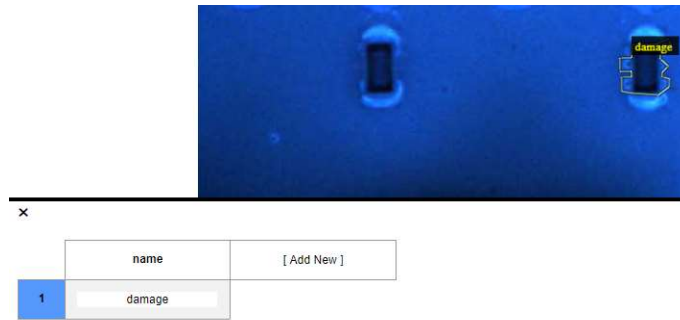


Figura 14.4 Identificación de clases.

Posteriormente se guardaron todas las anotaciones en un archivo de tipo JSON, junto a las imágenes clasificadas (ver Figura 4.5), para iniciar con el entrenamiento del modelo.



Figura 15.5 Imágenes con archivo de anotaciones (RPN).

4.2.3 Entrenamiento del modelo

Primeramente, se utilizó una red ya existente de *Mask R-CNN* en Python 3, Keras y TensorFlow. El modelo genera cajas delimitadoras y máscaras de segmentación para cada instancia de un objeto en la imagen. Está basado en la Red de Pirámide de Características (FPN) y un *backbone* de ResNet101. Para entrenar al modelo, usamos el modelo entrenado COCO como punto de control para realizar la transferencia de aprendizaje.

```
BATCH_SIZE = 128
```

```
NB_EPOCH = 2
```

```
NB_CLASSES = 1
```

```
VERBOSE = 1
```

```
VALIDATION_SPLIT = 0.2
```

IV. DESARROLLO

OPTIM = RMSprop()

Con información de la página oficial de Keras (Keras, 2019) explica los siguientes puntos:

- **batch_size**: integer. Number of samples per gradient update.
- **nb_epoch**: integer, the number of epochs to train the model.
- **verbose**: 0 for no logging to stdout, 1 for progress bar logging, 2 for one log line per epoch.
- **validation_split**: float ($0. < x < 1$). Fraction of the data to use as held-out validation data.

Pero más detalle, serían de acuerdo a los datos anteriores se explica el uso de las variables, BATCH_SIZE afecta significativamente al aprendizaje. Lo que sucede cuando se pone un lote en la red es que se promedian los gradientes. El concepto es que, si el tamaño del lote es lo suficientemente grande, esto proporcionará una estimación lo suficientemente estable de lo que sería el gradiente del conjunto de datos completo. Tomando muestras de su conjunto de datos, se estima el gradiente mientras se reduce el costo computacional significativamente.

Cuanto más bajo vayas, menos precisa será la estimación, sin embargo, en algunos casos estos ruidosos gradientes pueden ayudar a escapar de los mínimos locales. Cuando es demasiado bajo, las ponderaciones de su red pueden simplemente saltar si sus datos son ruidosos y puede que no puedan aprender o converjan muy lentamente, impactando así negativamente en el tiempo total de cálculo, otra ventaja de la agrupación es que para el cálculo de la GPU, las GPU son muy buenas para paralelizar los cálculos que ocurren en las redes neuronales si parte del cálculo es el mismo (por ejemplo, la multiplicación repetida de la matriz sobre la misma matriz de peso de su red). Esto significa que un tamaño de lote de 16 tomará menos del doble de la cantidad de un tamaño de lote de 8, en el caso de que necesite tamaños de lote más grandes pero que no quepan en su GPU.

Para VERBOSE 0, 1 o 2 sólo dices cómo quieres "ver" el progreso del entrenamiento para cada época.

- verbose=0 no te mostrará nada (silencioso).

IV. DESARROLLO

- `verbose=1` te mostrará una barra de progreso animada como la Figura 4.6:

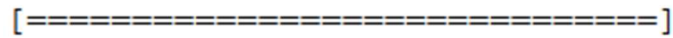


Figura 16.6 Barra de progreso con valor 1.

- `verbose=2` sólo mencionará el número de la época como la Figura 4.7:

Epoch 1/10

Figura 17.7 Progreso en épocas con valor 2.

Para `VALIDATION_SPLIT`, es un valor flotante entre 0 y 1. Una fracción de los datos de entrenamiento que se utilizará como datos de validación. El modelo separará esta fracción de los datos de entrenamiento, no entrenará en ella, y evaluará la pérdida y cualquier métrica del modelo sobre estos datos al final de cada época. Los datos de validación se seleccionan a partir de las últimas muestras en los datos `x - y` proporcionados, antes de la barajadura.

Para compilar un modelo Keras, se debe colocar un optimizador que en este caso es el algoritmo RMSprop, que consiste en:

- Mantener una media móvil (descontada) del cuadrado de los gradientes.
- Dividir el gradiente por la raíz de este promedio.

Esta implementación de RMSprop utiliza el impulso simple, no el impulso de Nesterov, la versión centrada mantiene adicionalmente un promedio móvil de los gradientes, y utiliza ese promedio para estimar la varianza. Nuestras configuraciones para el entrenamiento serán las de la figura 4.8:

IV. DESARROLLO

```
Configurations:
BACKBONE                resnet101
BACKBONE_STRIDES        [4, 8, 16, 32, 64]
BATCH_SIZE              1
BBOX_STD_DEV            [0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE None
DETECTION_MAX_INSTANCES 100
DETECTION_MIN_CONFIDENCE 0.9
DETECTION_NMS_THRESHOLD 0.3
FPN_CLASSIF_FC_LAYERS_SIZE 1024
GPU_COUNT              1
GRADIENT_CLIP_NORM      5.0
IMAGES_PER_GPU          1
IMAGE_MAX_DIM           1024
IMAGE_META_SIZE         14
IMAGE_MIN_DIM           800
IMAGE_MIN_SCALE         0
IMAGE_RESIZE_MODE       square
IMAGE_SHAPE             [1024 1024 3]
LEARNING_RATE           0.9
LEARNING_RATE           0.001
LOSS_WEIGHTS            {'rpn_class_loss': 1.0, 'rpn_bbox_loss': 1.0, 'mrcnn_class_loss': 1.0, 'mrcnn_bbox_loss': 1.0, 'mrcnn_mask_loss': 1.0}
MASK_POOL_SIZE          14
MASK_SHAPE              [28, 28]
MAX_GT_INSTANCES        100
MEAN_PIXEL              [123.7 116.8 103.9]
MINI_MASK_SHAPE         (56, 56)
NAME                    damage
NUM_CLASSES             2
POOL_SIZE               7
POST_NMS_ROIS_INFERENCE 1000
POST_NMS_ROIS_TRAINING  2000
ROI_POSITIVE_RATIO      0.33
RPN_ANCHOR_RATIOS       [0.5, 1, 2]
RPN_ANCHOR_SCALES       (32, 64, 128, 256, 512)
RPN_ANCHOR_STRIDE       1
RPN_BBOX_STD_DEV        [0.1 0.1 0.2 0.2]
RPN_NMS_THRESHOLD        0.7
RPN_TRAIN_ANCHORS_PER_IMAGE 256
STEPS_PER_EPOCH         100
TOP_DOWN_PYRAMID_SIZE   256
TRAIN_BN                False
TRAIN_ROIS_PER_IMAGE     200
USE_MINI_MASK           True
USE_RPN_ROIS            True
VALIDATION_STEPS        50
WEIGHT_DECAY            0.0001
```

Figura 18.8 Configuraciones de entrenamiento.

Se configuran las rutas para cargar las imágenes de entrenamiento, como previamente definimos (ver Figura 4.9).

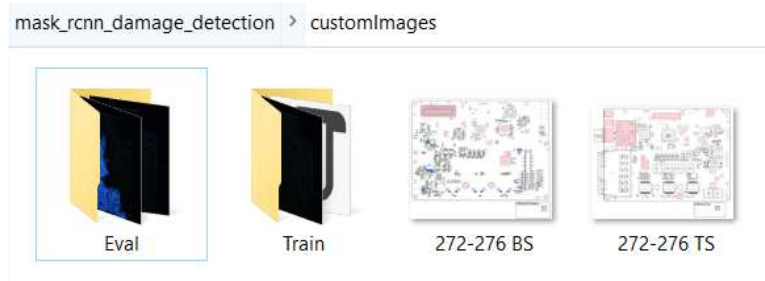


Figura 19.9 Carpetas de entrenamiento.

Se cargaron los datos de la carpeta de entrenamiento y se obtuvo la siguiente información de la Figura 4.10, como se observa son 2 clases una de BG o *background* y nuestra clase de *damage* y las 40 imágenes para este paso de entrenamiento.

IV. DESARROLLO

```
# Load dataset
# Get the dataset from the releases page
# https://github.com/matterport/Mask_RCNN/releases
dataset = custom.CustomDataset()
dataset.load_custom(CUSTOM_DIR, "train")

# Must call before using the dataset
dataset.prepare()

print("Image Count: {}".format(len(dataset.image_ids)))
print("Class Count: {}".format(dataset.num_classes))
for i, info in enumerate(dataset.class_info):
    print("{:3}. {:50}".format(i, info['name']))
```

Image Count: 40
Class Count: 2
0. BG
1. damage

Figura 20.10 Resultado de carpetas de entrenamiento.

Se cargó el *dataset* de las imágenes de validación también, donde están las imágenes que estamos usando para evaluación y aprovechamos el modelo COCO, como se ve en la Figura 4.11.

```
#Load dataset
#Get the dataset from the releases page
#https://github.com/matterport/Mask_RCNN/releases
dataset_val = custom.CustomDataset()
dataset_val.load_custom(CUSTOM_DIR, "val")

#Must call before using the dataset
dataset_val.prepare()

print("Image Count: {}".format(len(dataset_val.image_ids)))
print("Class Count: {}".format(dataset_val.num_classes))
for i, info in enumerate(dataset_val.class_info):
    print("{:3}. {:50}".format(i, info['name']))

image_ids = np.random.choice(dataset_train.image_ids, 4)
for image_id in image_ids:
    image = dataset_train.load_image(image_id)
    mask, class_ids = dataset_train.load_mask(image_id)
    visualize.display_top_masks(image, mask, class_ids, dataset_train.class_names)
    print(image_id)
    print(dataset_train.source_class_ids)
    #print(image_path)
    print(dataset_train.image_info[image_id]["source"])
    print(dataset_train.source_class_ids[dataset_train.image_info[image_id]["source"]])

print(MODEL_DIR)

model = modellib.MaskRCNN(mode="training", config=config,
model_dir=MODEL_DIR)

init_with = "coco" # imagenet, coco, or last

if init_with == "imagenet":
    model.load_weights(model.get_imagenet_weights(), by_name=True)
elif init_with == "coco":
    # Load weights trained on MS COCO, but skip layers that
    # are different due to the different number of classes
    # See README for instructions to download the COCO weights
    model.load_weights(COCO_MODEL_PATH, by_name=True,
    exclude=["mrcnn_class_logits", "mrcnn_bbox_fc",
    "mrcnn_bbox", "mrcnn_mask"])
elif init_with == "last":
    # Load the last model you trained and continue training
    model.load_weights(model.find_last(), by_name=True)

Image Count: 10
Class Count: 2
0. BG
1. damage
```

Figura 21.11 Clases y archivos en *Dataset*.

IV. DESARROLLO

Así que se inició con el entrenamiento (ver Figura 4.12) y se creó nuestro modelo “mask_rcnn_damage.h5”

```
#Train the head branches
#Passing layers="heads" freezes all layers except the head layers.
#You can also pass a regular expression to select which layers to train by name pattern.
model.train(dataset_train, dataset_val,
            learning_rate=config.LEARNING_RATE,
            epochs=1,
            layers='heads')

model_path = os.path.join(ROOT_DIR, "mask_rcnn_damage.h5")
model.keras_model.save_weights(model_path)

Starting at epoch 0. LR=0.001

Checkpoint Path: C:\Users\RubeDomingue\Desktop\TESIS\Deep-Learning-master\mask_rcnn_damage_detection\mask_rcnn_damage_detection
\logs\damage20191109T1134\mask_rcnn_damage_{epoch:04d}.h5
Selecting layers to train
fpn_c5p5          (Conv2D)
fpn_c4p4          (Conv2D)
fpn_c3p3          (Conv2D)
fpn_c2p2          (Conv2D)
fpn_p5            (Conv2D)
fpn_p2            (Conv2D)
fpn_p3            (Conv2D)
fpn_p4            (Conv2D)
In model:  rpn_model
           rpn_conv_shared      (Conv2D)
           rpn_class_raw        (Conv2D)
           rpn_bbox_pred        (Conv2D)
mrcnn_mask_conv1 (TimeDistributed)
mrcnn_mask_bn1   (TimeDistributed)
mrcnn_mask_conv2 (TimeDistributed)
mrcnn_mask_bn2   (TimeDistributed)
mrcnn_class_conv1 (TimeDistributed)
mrcnn_class_bn1  (TimeDistributed)
mrcnn_mask_conv3 (TimeDistributed)
mrcnn_mask_bn3   (TimeDistributed)
mrcnn_class_conv2 (TimeDistributed)
mrcnn_class_bn2  (TimeDistributed)
mrcnn_mask_conv4 (TimeDistributed)
mrcnn_mask_bn4   (TimeDistributed)
mrcnn_bbox_fc    (TimeDistributed)
mrcnn_mask_deconv (TimeDistributed)
mrcnn_class_logits (TimeDistributed)
mrcnn_mask       (TimeDistributed)
```

Figura 22.12 Inicio del entrenamiento.

Después de terminar el entrenamiento de una época, se obtienen valores de resultado (ver Figura 4.13).

```
Epoch 1/1
100/100 [=====] - 2651s 27s/step - loss: 1.3565 - rpn_class_loss: 0.0300 - rpn_bbox_loss: 0.2892 -
mrcnn_class_loss: 0.0732 - mrcnn_bbox_loss: 0.5032 - mrcnn_mask_loss: 0.4610 - val_loss: 1.3652 - val_rpn_class_loss: 0.0265
- val_rpn_bbox_loss: 0.2214 - val_mrcnn_class_loss: 0.0772 - val_mrcnn_bbox_loss: 0.6442 - val_mrcnn_mask_loss: 0.3959
```

Figura 23.13 Resultados del entrenamiento.

4.2.4 Validación del modelo.

Después de obtener nuestro modelo, se utilizó el notebook *Inspect Custom Weights*, para realizar un control de *sanity check* (cordura) para ver si los pesos y sesgos están bien distribuidos. Una muestra de salida se ve en la Figura 4.14.

IV. DESARROLLO

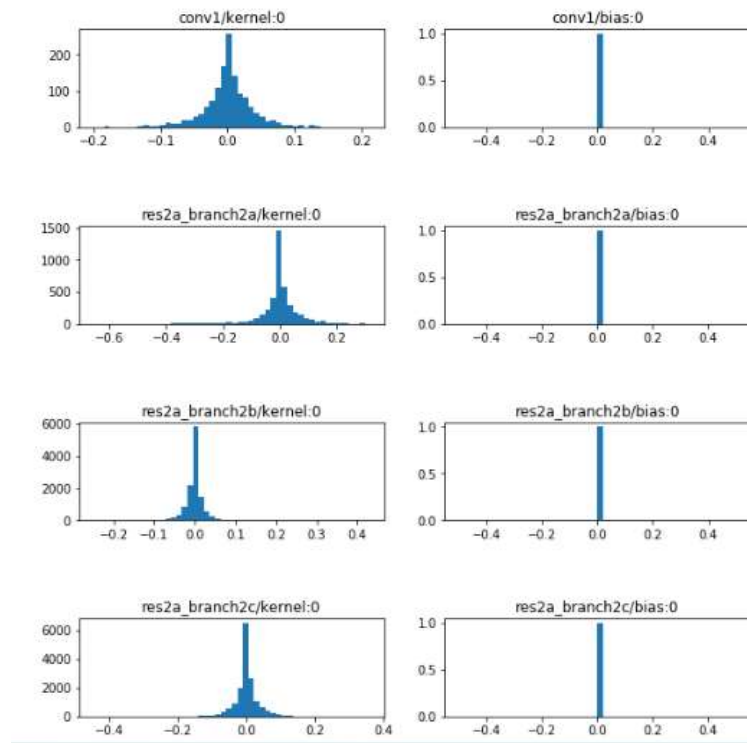


Figura 24.14 Resultados de validación del modelo.

Para esto se cargó el modelo con el siguiente código de la Figura 4.15.

Load Model ¶

```
# Create model in inference mode
with tf.device(DEVICE):
    model = modellib.MaskRCNN(mode="inference", model_dir=MODEL_DIR,
                              config=config)

# Load weights
#weights_path = 'mask_rcnn_damage_0010.h5'
weights_path = 'mask_rcnn_damage.h5'

# Load weights
print("Loading weights ", weights_path)
model.load_weights(weights_path, by_name=True)
```

Figura 25.15 Código para cargar el modelo.

4.2.5 Preprocesamiento de los datos

El notebook “*inspect_custom_data*”, muestra los diferentes pasos de pre procesamiento para preparar los datos de entrenamiento. Se cargan 5 imágenes aleatorias y muestra la imagen y su máscara ver Figura 4.16.

IV. DESARROLLO

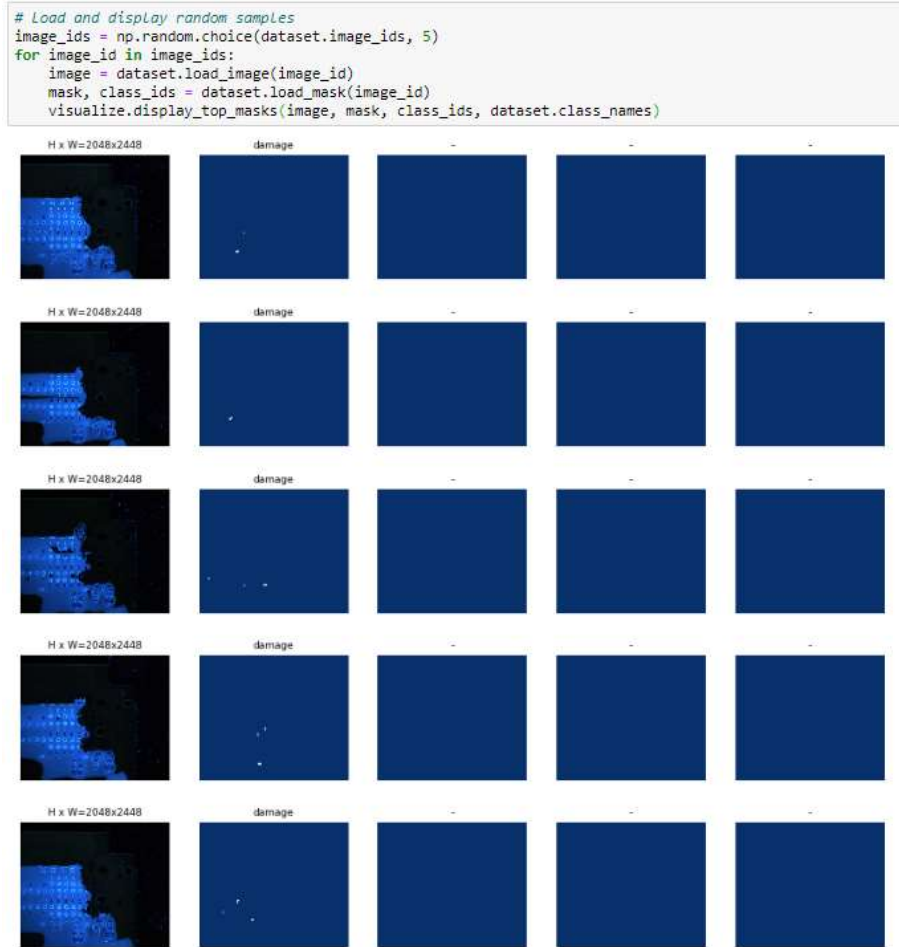


Figura 26.16 Imágenes y su máscara.

Mediante las áreas que se seleccionaron en los cuadros delimitadores previamente en *VGG*, el sistema en lugar de utilizar las coordenadas de los cuadros delimitadores proporcionados por los conjuntos de datos de la fuente que se creó, los calcula basándose en las máscaras (ver Figura 4.17).



Figura 27.17 Imagen con máscara.

Esto facilita el manejo de los cuadros en forma consistente, sin importar los datos de origen (ver Figura 4.18), y también permite cambiar de tamaño, rotación o recortar imágenes basándose en

IV. DESARROLLO

las máscaras de actualización (ver Figura 4.19).

```
# Load random image and mask.
image_id = random.choice(dataset.image_ids)
image = dataset.load_image(image_id)
mask, class_ids = dataset.load_mask(image_id)
# Compute Bounding box
bbox = utils.extract_bboxes(mask)

# Display image and additional stats
print("image_id ", image_id, dataset.image_reference(image_id))
log("image", image)
log("mask", mask)
log("class_ids", class_ids)
log("bbox", bbox)
# Display image and instances
visualize.display_instances(image, bbox, mask, class_ids, dataset.class_names)
```

image_id	8	C:\Users\RubeDomingue\Desktop\TESIS\mask_rcnn_damage_detection\customImages\train\9.jpg
image	shape: (2048, 2448, 3)	min: 0.00000 max: 255.00000 uint8
mask	shape: (2048, 2448, 5)	min: 0.00000 max: 1.00000 bool
class_ids	shape: (5,)	min: 1.00000 max: 1.00000 int32
bbox	shape: (5, 4)	min: 362.00000 max: 1801.00000 int32

Figura 28.18 Valores de los cuadros delimitadores.

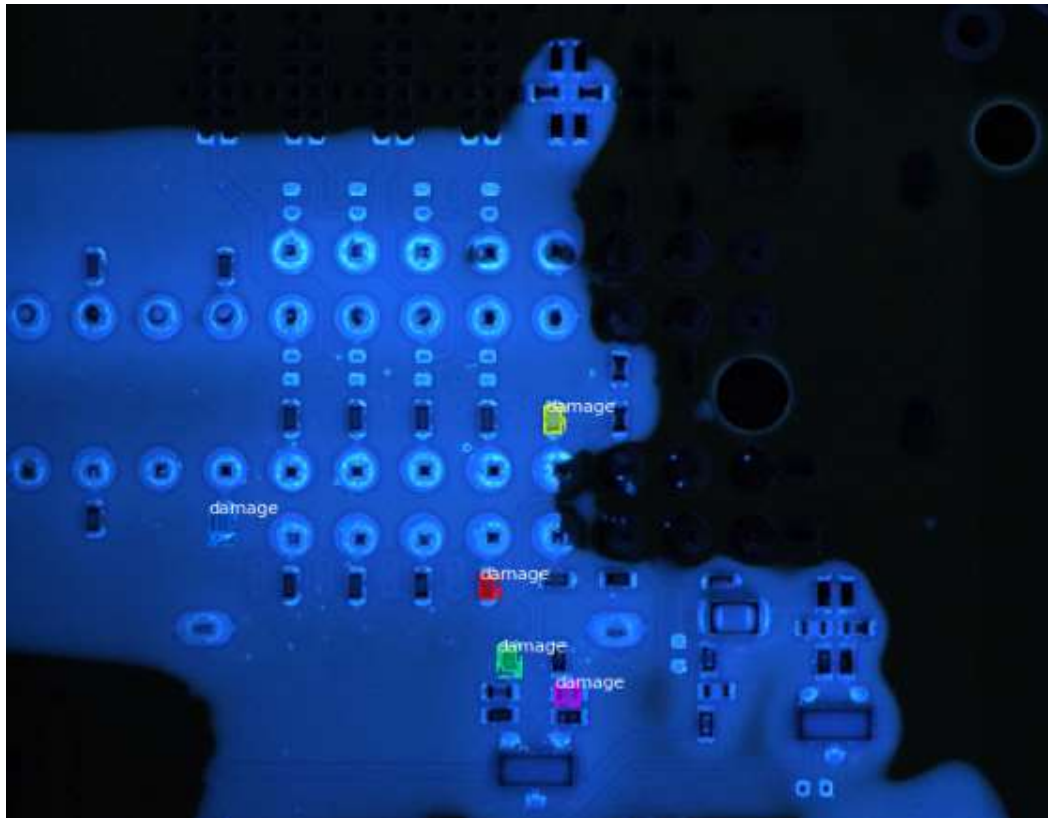


Figura 29.19 Cuadros delimitadores en defectos.

Para soportar múltiples imágenes por lote, las imágenes son redimensionadas a (1024x1024).

IV. DESARROLLO

Sin embargo, se conserva la relación de aspecto. Si una imagen no es igual en todos sus lados, se añaden ceros de relleno en los extremos de la imagen (ver Figura 4.20)

```
image_id: 36 C:\Users\RubeDomingue\Desktop\TESIS\mask_rcnn_damage_detection\customImages\train\37.jpg
Original shape: (2048, 2448, 3)
image      shape: (1024, 1024, 3)    min: 0.00000 max: 254.00000 uint8
mask      shape: (1024, 1024, 3) min: 0.00000 max: 1.00000 bool
class_ids shape: (3,)          min: 1.00000 max: 1.00000 int32
bbox      shape: (3, 4)        min: 256.00000 max: 818.00000 int32
```

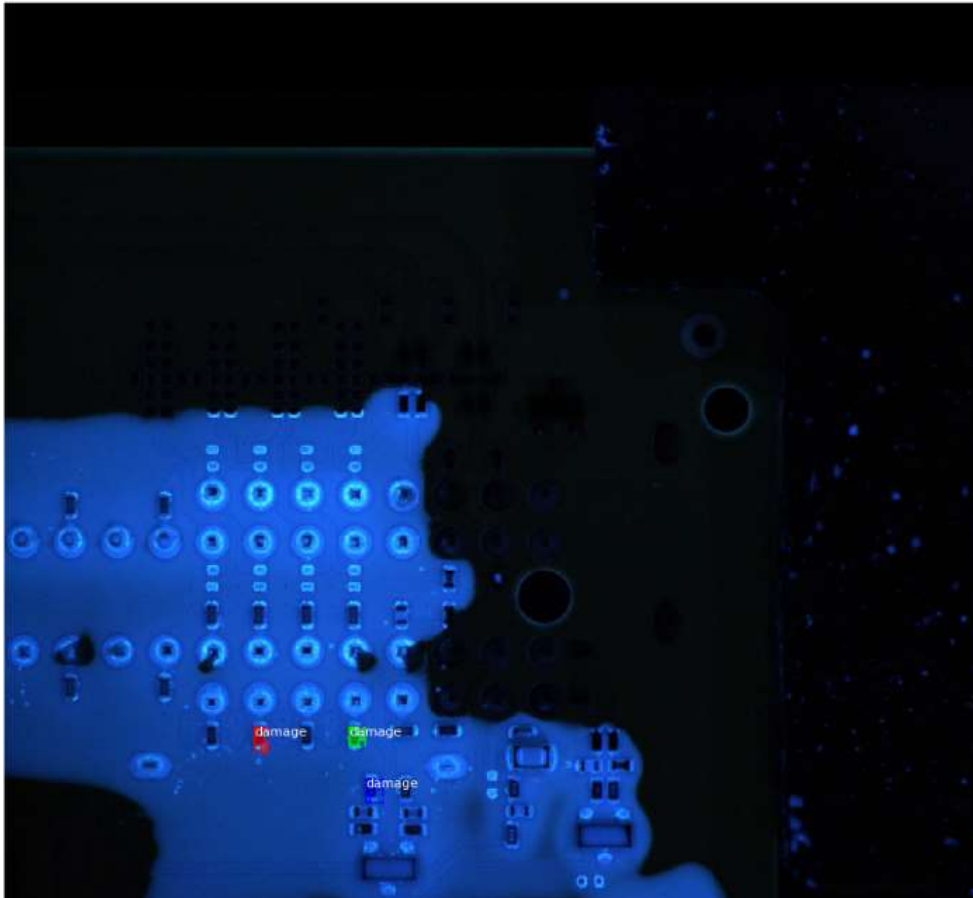


Figura 30.20 Imagen redimensionada.

Se crean máscaras pequeñas, estas máscaras de instancia pueden agrandarse al entrenar con imágenes de mejor resolución. Por ejemplo, si se utiliza una imagen de 1024x1024, la máscara de instancia utilizara 1MB de memoria (Numpy utiliza bytes en los valores de tipo booleano). Si una imagen utiliza 50 instancias entonces son 50MB para estas máscaras, para optimizar el entrenamiento, se optimizan estas máscaras:

- Almacenando los píxeles que están dentro del cuadro delimitador del objeto, en vez de utilizar una sola máscara de imagen completa. En general los objetos son de menor

IV. DESARROLLO

tamaño comparados con la imagen, así que se ahorra espacio por no almacenar demasiados ceros alrededor de los objetos (ver Figura 4.21)

```
image           shape: (1024, 1024, 3)   min:  0.00000  max: 254.00000  uint8
image_meta      shape: (14,)             min:  0.00000  max: 2448.00000 float64
class_ids       shape: (2,)           min:  1.00000  max:  1.00000  int32
bbox            shape: (2, 4)       min: 208.00000  max: 758.00000  int32
mask            shape: (1024, 1024, 2) min:  0.00000  max:  1.00000  bool
```

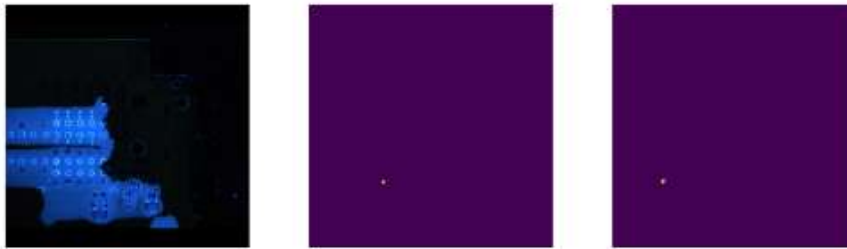


Figura 31.21 Imagen y sus máscaras con redimensionamiento.

- Redimensionamos la máscara para hacerla más pequeña (por ejemplo, 42x42). Para objetos más grandes que el tamaño definido se pierde un poco de exactitud. La mayoría de los objetos tienen anotaciones que no son muy exactas para iniciar, por lo cual la pérdida es insignificante para las intenciones prácticas. El tamaño de la *mini_máscara* se puede establecer en la configuración.

Para visualizar el efecto del redimensionamiento de la máscara, y para verificar la corrección del código, se visualizan algunos ejemplos en la siguiente Figura 4.22.

IV. DESARROLLO

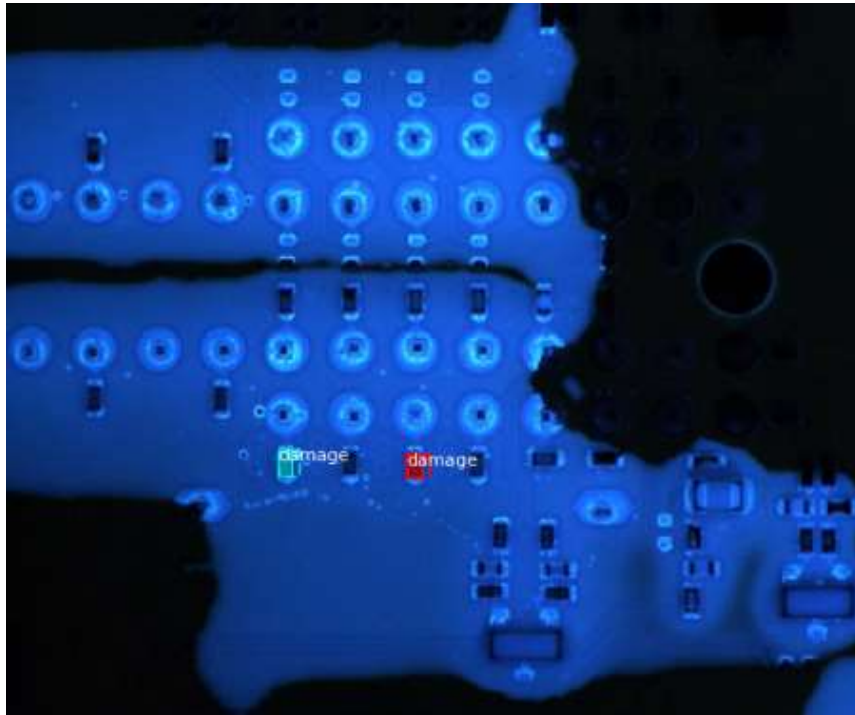


Figura 32.22 Imagen con redimensionamiento y máscara.

Se aumenta y redimensiona la máscara (ver Figura 4.23)

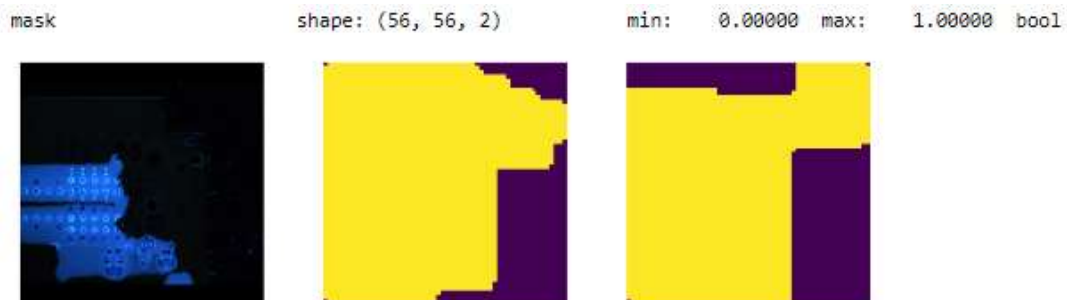


Figura 33.23 Imagen con máscara aumentada.

Se inicia con el proceso de las anclas (*anchors*), donde el orden de las anclas es importante estas deben estar en el mismo orden en las fases de entrenamiento y de predicción. Debe coincidir este orden para la ejecución de la capa de convolución, de una red FPN, las anclas deben estar ordenadas de forma que sea sencillo hacer coincidir las anclas con el resultado de las capas de convolución que predicen las puntuaciones y los desplazamientos de las anclas, se debe ordenar primero por nivel de pirámide. Todas las anclas del primer nivel, luego todas las del segundo y

IV. DESARROLLO

así sucesivamente. Esto hace más fácil separar las anclas por nivel, en cada nivel, se ordenan las anclas como en la secuencia de procesamiento del mapa de características.

Normalmente, la capa de convolución procesa un mapa de características comenzando en la parte superior izquierda, moviéndose fila por fila a la derecha, por cada celda en el mapa de características, se selecciona cualquier orden de clasificación para las anclas de diferentes proporciones. Aquí se coincide con el orden de las proporciones pasadas a la función.

“*Anchor Stride*”: La arquitectura *FPN*, los mapas de características son de alta resolución en las primeras capas. Por ejemplo, si una imagen de entrada es de 1024x1024, el mapa de características de la primera capa es de 256x256, lo que crea alrededor de 200K de anclas (256x256). Estas anclas son 32x32 píxeles y su superposición a los píxeles de la imagen es de 4 píxeles, lo que genera mucho solapamiento. Se puede reducir esta carga significativamente si se generan anclas para cada una de las otras celdas del mapa de características. Un *stride* de 2 reducirá el número de anclas en 4, por ejemplo. En esta implementación se usó un *stride* de 2 anclas (ver Figura 4.24).

```
# Generate Anchors
backbone_shapes = modellib.compute_backbone_shapes(config, config.IMAGE_SHAPE)
anchors = utils.generate_pyramid_anchors(config.RPN_ANCHOR_SCALES,
                                         config.RPN_ANCHOR_RATIOS,
                                         backbone_shapes,
                                         config.BACKBONE_STRIDES,
                                         config.RPN_ANCHOR_STRIDE)

# Print summary of anchors
num_levels = len(backbone_shapes)
anchors_per_cell = len(config.RPN_ANCHOR_RATIOS)
print("Count: ", anchors.shape[0])
print("Scales: ", config.RPN_ANCHOR_SCALES)
print("ratios: ", config.RPN_ANCHOR_RATIOS)
print("Anchors per Cell: ", anchors_per_cell)
print("Levels: ", num_levels)
anchors_per_level = []
for l in range(num_levels):
    num_cells = backbone_shapes[l][0] * backbone_shapes[l][1]
    anchors_per_level.append(anchors_per_cell * num_cells // config.RPN_ANCHOR_STRIDE**2)
    print("Anchors in Level {}: {}".format(l, anchors_per_level[l]))

Count: 261888
Scales: (32, 64, 128, 256, 512)
ratios: [0.5, 1, 2]
Anchors per Cell: 3
Levels: 5
Anchors in Level 0: 196608
Anchors in Level 1: 49152
Anchors in Level 2: 12288
Anchors in Level 3: 3072
Anchors in Level 4: 768
```

Figura 34.24 Implementación de anclas.

IV. DESARROLLO

A continuación en la Figura 4.25, se visualizan las anclas de la celda en el centro del mapa de características por cada nivel en específico.

```
Level 0. Anchors: 196608 Feature map Shape: [256 256]
Level 1. Anchors: 49152 Feature map Shape: [128 128]
Level 2. Anchors: 12288 Feature map Shape: [64 64]
Level 3. Anchors: 3072 Feature map Shape: [32 32]
Level 4. Anchors: 768 Feature map Shape: [16 16]
```

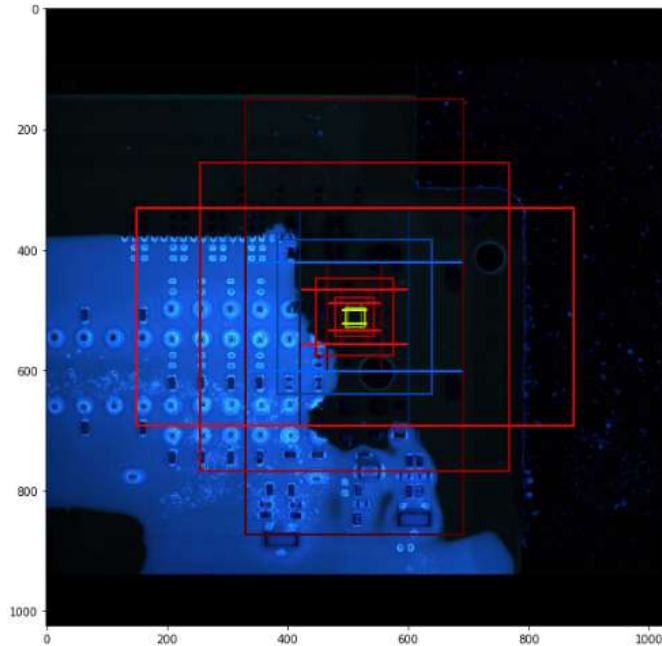


Figura 35.25 Anclas de una celda.

Se inicia con el Generador de datos con el código de la Figura 4.26

```
# Get Next Image
if random_rois:
    [normalized_images, image_meta, rpn_match, rpn_bbox, gt_class_ids, gt_boxes, gt_masks, rpn_rois, rois], \
    [mrcnn_class_ids, mrcnn_bbox, mrcnn_mask] = next(g)

    log("rois", rois)
    log("mrcnn_class_ids", mrcnn_class_ids)
    log("mrcnn_bbox", mrcnn_bbox)
    log("mrcnn_mask", mrcnn_mask)
else:
    [normalized_images, image_meta, rpn_match, rpn_bbox, gt_boxes, gt_masks], _ = next(g)

log("gt_class_ids", gt_class_ids)
log("gt_boxes", gt_boxes)
log("gt_masks", gt_masks)
log("rpn_match", rpn_match, )
log("rpn_bbox", rpn_bbox)
image_id = modellib.parse_image_meta(image_meta)["image_id"][0]
print("image_id: ", image_id, dataset.image_reference(image_id))

# Remove the last dim in mrcnn_class_ids. It's only added
# to satisfy Keras restriction on target shape.
mrcnn_class_ids = mrcnn_class_ids[:, :0]

rois shape: (4, 200, 4) min: 6.00000 max: 1020.00000 int32
mrcnn_class_ids shape: (4, 200, 1) min: 0.00000 max: 1.00000 int32
mrcnn_bbox shape: (4, 200, 2, 4) min: -3.17994 max: 3.14304 float32
mrcnn_mask shape: (4, 200, 28, 28, 2) min: 0.00000 max: 1.00000 float32
gt_class_ids shape: (4, 100) min: 0.00000 max: 1.00000 int32
gt_boxes shape: (4, 100, 4) min: 0.00000 max: 814.00000 int32
gt_masks shape: (4, 56, 56, 100) min: 0.00000 max: 1.00000 bool
rpn_match shape: (4, 261888, 1) min: -1.00000 max: 1.00000 int32
rpn_bbox shape: (4, 256, 4) min: -4.89548 max: 2.87262 float64
image_id: 1 C:\Users\RubeDomingue\Desktop\TESIS\mask_rcnn_damage_detection\customImages\train\2.jpg
```

Figura 36.26 Generador de datos de anclas.

IV. DESARROLLO

Se realizó la evaluación de las anclas con el código en la Figura 4.27, tanto positivas (ver Figura 4.28) como negativas (ver Figura 4.29) y neutrales (ver Figura 4.30),

```
b = 0

# Restore original image (reverse normalization)
sample_image = modellib.unmold_image(normalized_images[b], config)

# Compute anchor shifts.
indices = np.where(rpn_match[b] == 1)[0]
refined_anchors = utils.apply_box_deltas(anchors[indices], rpn_bbox[b, :len(indices)] * config.RPN_BBOX_STD_DEV)
log("anchors", anchors)
log("refined_anchors", refined_anchors)

# Get list of positive anchors
positive_anchor_ids = np.where(rpn_match[b] == 1)[0]
print("Positive anchors: {}".format(len(positive_anchor_ids)))
negative_anchor_ids = np.where(rpn_match[b] == -1)[0]
print("Negative anchors: {}".format(len(negative_anchor_ids)))
neutral_anchor_ids = np.where(rpn_match[b] == 0)[0]
print("Neutral anchors: {}".format(len(neutral_anchor_ids)))

# ROI breakdown by class
for c, n in zip(dataset.class_names, np.bincount(mrcnn_class_ids[b].flatten())):
    if n:
        print("{}:23: {}".format(c[:20], n))

# Show positive anchors
fig, ax = plt.subplots(1, figsize=(16, 16))
visualize.draw_boxes(sample_image, boxes=anchors[positive_anchor_ids],
                    refined_boxes=refined_anchors, ax=ax)

anchors          shape: (261888, 4)          min: -362.03867  max: 1322.03867  float64
refined_anchors  shape: (5, 4)          min: 206.00000   max: 810.00000   float32
Positive anchors: 5
Negative anchors: 251
Neutral anchors: 261632
BG               : 177
damage           : 23
```

Figura 37.27 Código de evaluación de anclas.

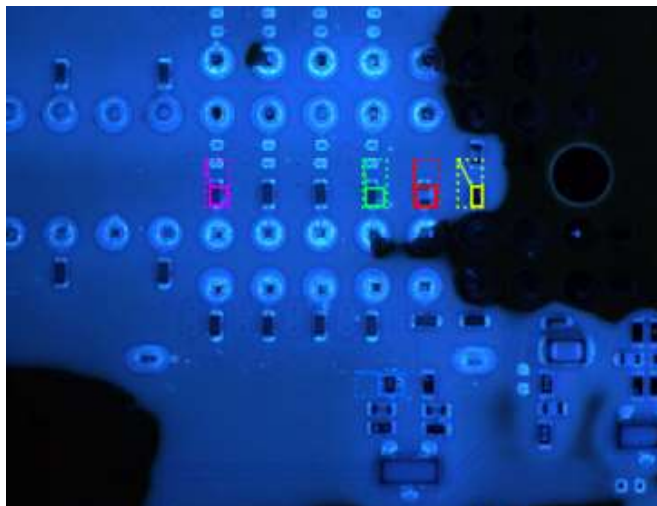


Figura 38.28 Anclas positivas.

IV. DESARROLLO

```
# Show negative anchors  
visualize.draw_boxes(sample_image, boxes=anchors[negative_anchor_ids])
```

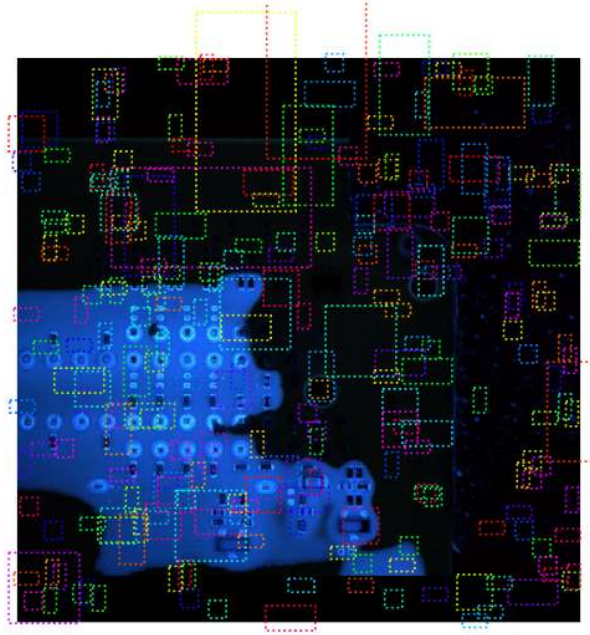


Figura 39.29 Anclas negativas.

```
# Show neutral anchors. They don't contribute to training.  
visualize.draw_boxes(sample_image, boxes=anchors[np.random.choice(neutral_anchor_ids, 100)])
```

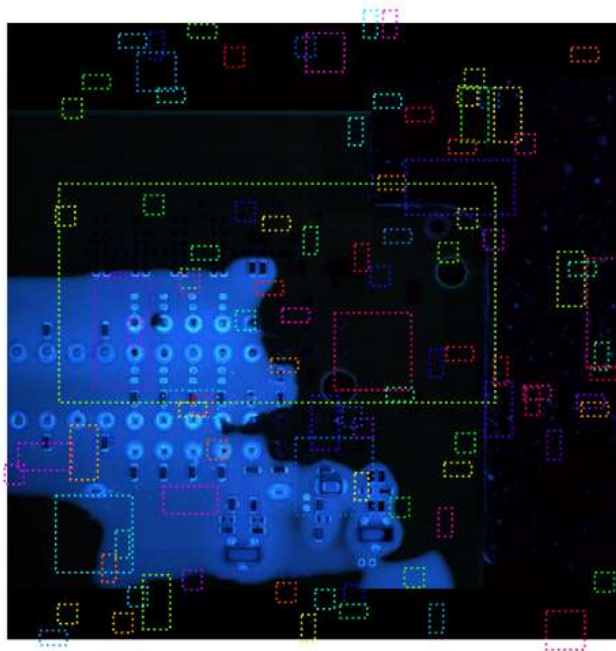


Figura 40.30 Anclas neutrales.

IV. DESARROLLO

Posteriormente se inició con las *ROIs* o *RPNs* con el código de la Figura 4.31 y se muestran algunos *ROI* aleatorios.

```
if random_rois:
    # Class aware bboxes
    bbox_specific = mrcnn_bbox[b, np.arange(mrcnn_bbox.shape[1]), mrcnn_class_ids[b], :]

    # Refined ROIs
    refined_rois = utils.apply_box_deltas(rois[b].astype(np.float32), bbox_specific[:, :4] * config.BBOX_STD_DEV)

    # Class aware masks
    mask_specific = mrcnn_mask[b, np.arange(mrcnn_mask.shape[1]), :, :, mrcnn_class_ids[b]]

    visualize.draw_rois(sample_image, rois[b], refined_rois, mask_specific, mrcnn_class_ids[b], dataset.class_names)

    # Any repeated ROIs?
    rows = np.ascontiguousarray(rois[b]).view(np.dtype((np.void, rois.dtype.itemsize * rois.shape[-1])))
    _, idx = np.unique(rows, return_index=True)
    print("Unique ROIs: {} out of {}".format(len(idx), rois.shape[1]))
```

Positive ROIs: 23
Negative ROIs: 177
Positive Ratio: 0.12
Unique ROIs: 200 out of 200

Showing 10 random ROIs out of 200

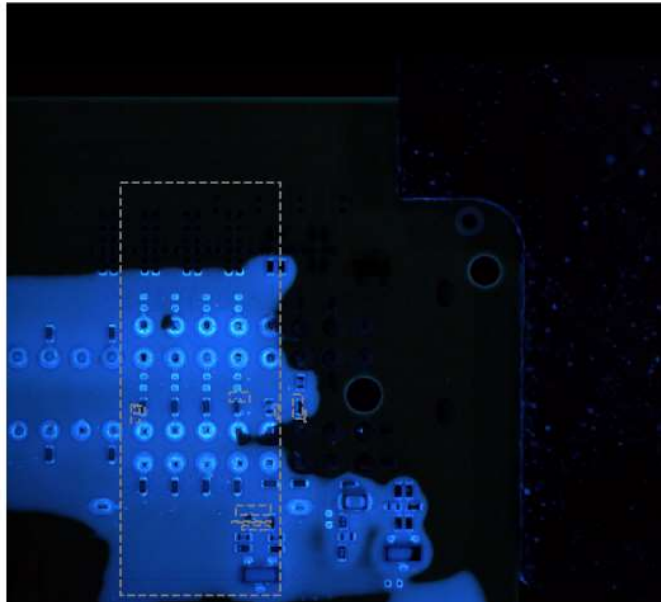


Figura 41.31 *ROIs* o *RPNs*.

Para continuar con la ejecución de proyecto, se utilizó el código de la Figura 4.32 y muestra *ROIs*, máscaras y cajas de detección, fondo de imagen de todas las imágenes como se observa en la Figura 4.33.

IV. DESARROLLO

```
if random_rois:
    # Display ROIs and corresponding masks and bounding boxes
    ids = random.sample(range(rois.shape[1]), 8)

    images = []
    titles = []
    for i in ids:
        image = visualize.draw_box(sample_image.copy(), rois[b,i,:4].astype(np.int32), [255, 0, 0])
        image = visualize.draw_box(image, refined_rois[i].astype(np.int64), [0, 255, 0])
        images.append(image)
        titles.append("ROI {}".format(i))
        images.append(mask_specific[i] * 255)
        titles.append(dataset.class_names[mrcnn_class_ids[b,i]][:20])

    display_images(images, titles, cols=4, cmap="Blues", interpolation="none")
```

Figura 42.32 Código detección de ROIs, máscaras y cajas.

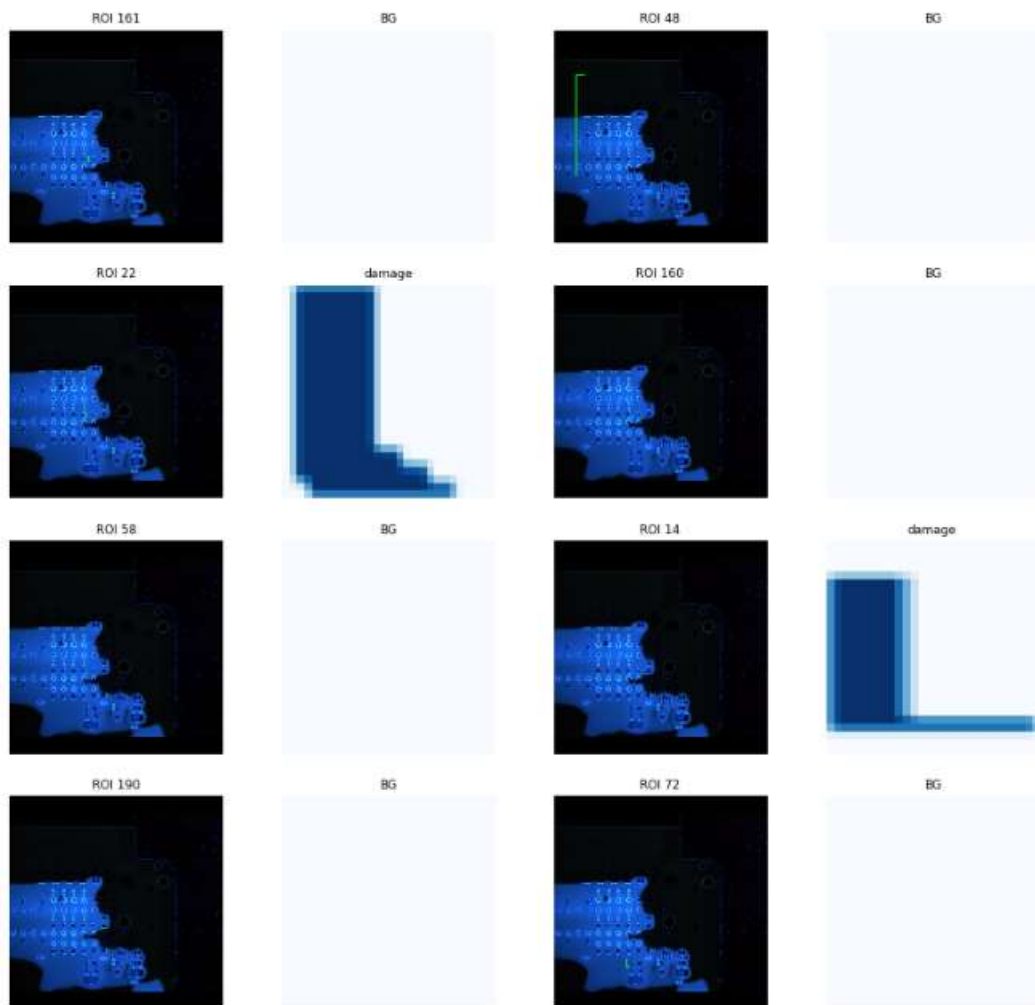


Figura 43.33 Detección de defectos y fondos en imágenes muestra.

Todas las explicaciones anteriores muestran cómo es que funciona el código para realizar la

detección de los defectos y sus respectivos procesos para lograrlo.

4.2.6 Ejecutar el modelo en imágenes y hacer predicciones

Para realizar la predicción se utilizó el notebook “*inspect_custom_model*” para ejecutar el modelo en las imágenes del conjunto de pruebas/evaluación y ver las predicciones del modelo. Como se muestra el resultado en la Figura 4.34 y una imagen ampliada con zoom en la Figura 4.35:

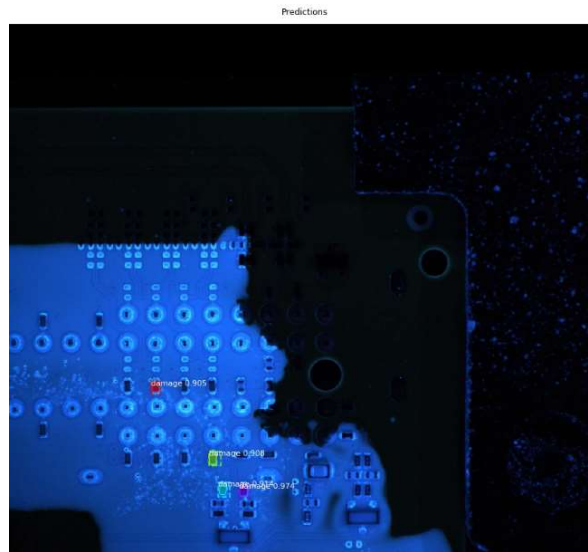


Figura 44.34 Resultado de la predicción.

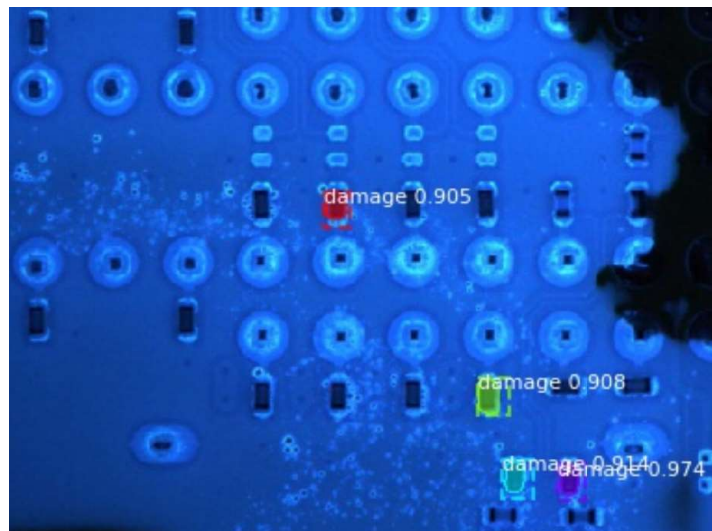


Figura 45.35 Zoom del resultado de la predicción.

Claramente se puede ver que puede identificar los defectos, pero aún tiene algunas oportunidades.

4.3 Implementación

Después de obtener y definir las imágenes para el entrenamiento y creación del modelo “mask_rcnn_damage.h5”, se realizó la implementación del prototipo en un computador personal en donde se desarrollan los programas de los equipos de inspección *xray*, aquí se instaló el software Anaconda (ver Figura 4.36)

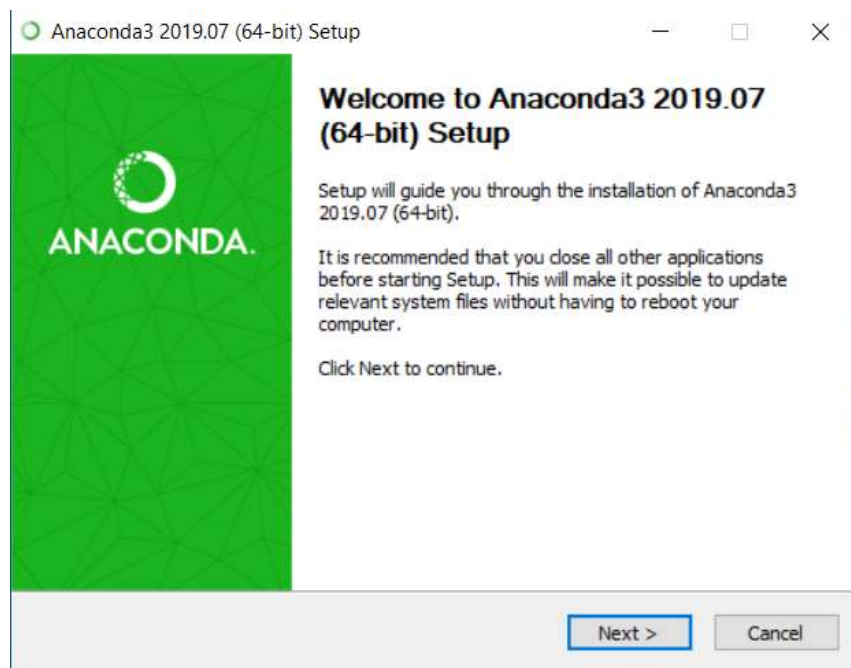


Figura 46.36 Ventana de instalación de Anaconda.

Al finalizar la instalación se inició la consola de Anaconda Prompt imagen (ver Figura 4.37), para instalar las librerías necesarias para ejecutar el notebook y realizar la evaluación de las imágenes.



Figura 47.37 Consola de Anaconda.

IV. DESARROLLO

Se instalaron las librerías de Keras, TensorFlow y OpenCV (ver Figura 4.38), exactamente estas versiones específicas, ya que versiones más antiguas o nuevas mostraban problemas de compatibilidad y/o compilaciones a la hora de ejecutar pruebas.

```
(base) C:\Users\RubeDomingue>pip install Keras==2.2.5  
(base) C:\Users\RubeDomingue>pip install tensorflow==1.13.1  
(base) C:\Users\RubeDomingue>pip install opencv-python
```

Figura 48.38 Comando de instalación de librerías.

Se creó una nueva carpeta en el escritorio llamada “Testing” (ver Figura 4.39), esta computadora tiene un usuario genérico, por lo que cualquier persona puede usar el prototipo.

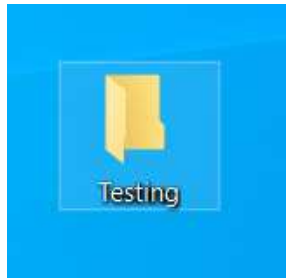


Figura 49.39 Carpeta de pruebas.

Dentro de la carpeta se colocan los siguientes archivos y carpetas (ver Figura 4.40), para garantizar el funcionamiento del prototipo.

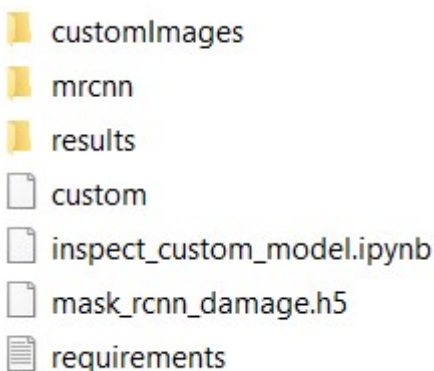


Figura 50.40 Archivos y carpetas de la prueba.

Cada uno de los archivos y carpetas dentro de la carpeta funciona para lo siguiente:

IV. DESARROLLO

- *customImages*, es donde se colocarán las imágenes a evaluar, todo esto dentro de la carpeta “val” (ver Figura 4.41).

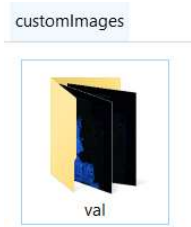


Figura 51.41 Carpeta de imágenes para evaluación.

- *mrcnn*, contiene algunas utilerías.
- *Results*, será la carpeta donde se guardará la imagen final con la identificación de los defectos (ver Figura 4.42).

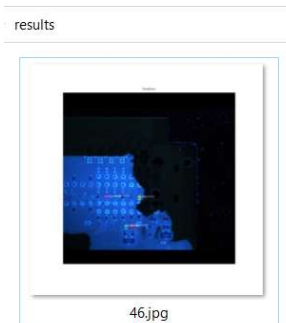


Figura 52.42 Imagen de resultado de la evaluación.

- *Custom.py*, es donde se encuentra nuestra configuración para el prototipo.
- *Inspect_custom_model.ipynb*, es nuestro notebook que realiza la evaluación de las imágenes.
- *Mask_rcnn_damage.h5*, es nuestro dataset.
- *Requirements.txt*, es donde se mencionan las instrucciones para ejecutar el prototipo.

Posteriormente se ejecuta la aplicación de Jupyter Notebook (ver Figura 4.43), para iniciar la aplicación de prueba de imágenes.



Figura 53.43 Aplicación para la evaluación del prototipo.

4.4 Pruebas

Durante la etapa de pruebas se observó que el *SI* logro una detección oportuna de algunos tipos de defecto similares a los entrenados durante las etapas previas, logrando un porcentaje de semejanza mayor al 90% (0.9) como se ve en la Figura 4.44.

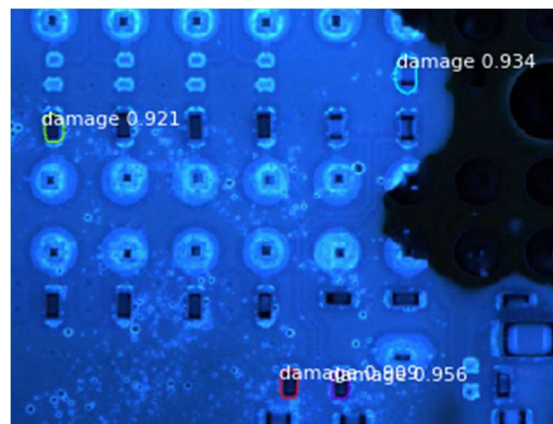


Figura 54.44 Evaluación de pruebas en primeras muestras.

Como se observó en las pruebas, en este tipo de proceso las fallas son muy diferentes entre cada muestra, por lo que al cambiar de muestra los valores de detección eran cambiantes desde un 90% hasta un 98% de similitud, además que es difícil conseguir tarjetas con defectos reales o fuera del margen.

V. RESULTADOS Y DISCUSION

Derivado de las muestras iniciales y de cómo se afectaron las distintas pruebas que se realizaron, para encontrar una mejor solución al problema, se obtuvieron los siguientes resultados.

5.1 Muestras seleccionadas para la experimentación

Inicialmente el proyecto se iba a centrar en imágenes a color con múltiples tipos de componentes, como se muestra en la Figura 5.1, pero esta tenía una gran complejidad y requería de mayor tiempo para la realización de los prototipos y pruebas, así que se eligió la opción de utilizar las imágenes de *conformal*, como se ve en la Figura 5.2, que tienen una menor complejidad, ya que la función es buscar la presencia o ausencia del material.

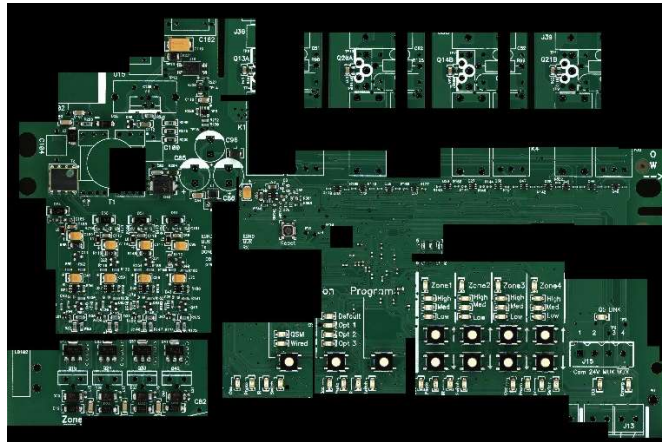


Figura 1.1 Imagen completa de una inspección por componentes.

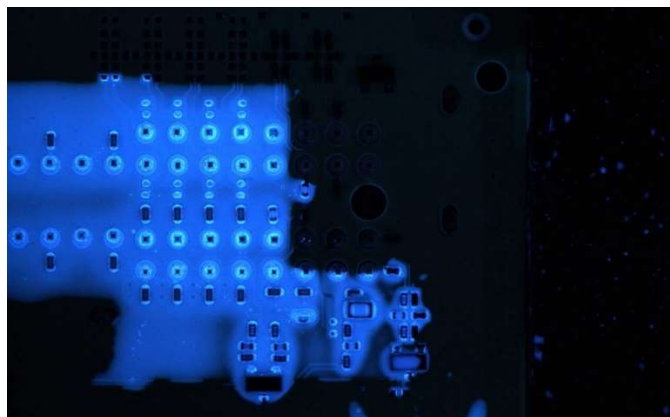


Figura 2.2 Imagen completa de una inspección por *conformal*.

A las primeras muestras se les aplicaron distintas pruebas, intentando encontrar la mejor opción para detectar los defectos en las tarjetas electrónicas.

5.2 Resultados de las pruebas en distintos estudios.

Se realizaron múltiples pruebas con las primeras muestras, identificando las ventajas o desventajas de utilizar cada tipo de prueba. A continuación, se mencionan las distintas pruebas utilizadas.

5.2.1 Detección con Keras para CNN

Se utilizó un prototipo de una red CNN, para crear el *dataset* y realizar las distintas operaciones, entrenamiento, validación y pruebas.

Inicialmente se entrenó obteniendo los siguientes resultados (ver Figura 5.3 y Figura 5.4)

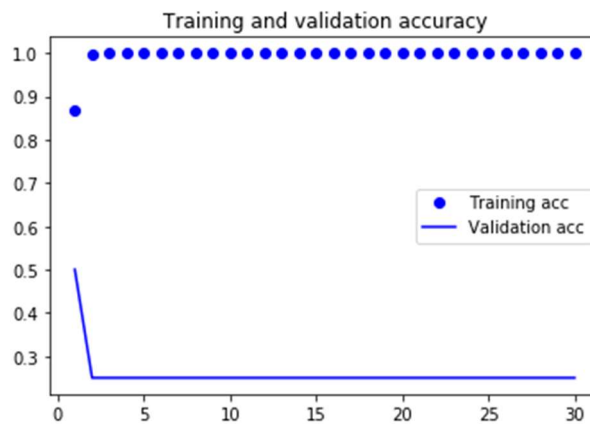


Figura 3.3 Precisión del entrenamiento y la validación

V. RESULTADOS Y DISCUSION

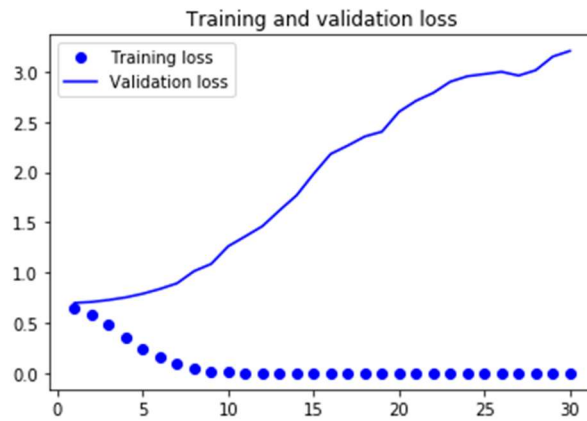


Figura 4.4 Pérdida de entrenamiento y validación

Después de esto se realizó una predicción, pero se obtuvieron predicciones muy equivocadas a las esperadas (ver Figura 5.5), logrando menos de un 5% de efectividad.

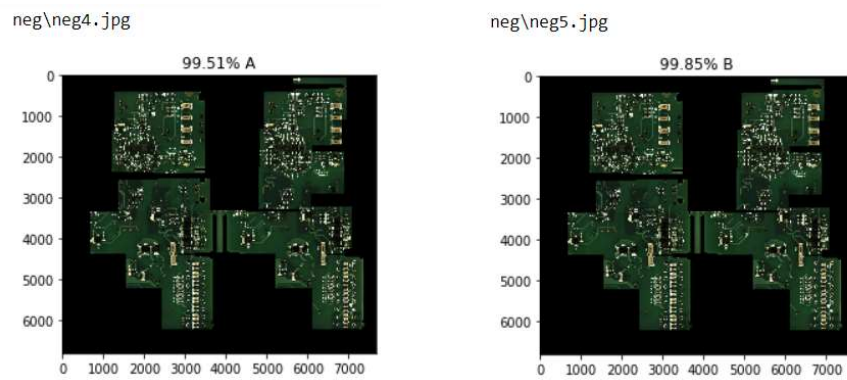


Figura 5.5 Resultados de la predicción.

5.2.2 Detección usando matrices de transformación

En este tipo de prueba se utilizaba una imagen como muestra y la comparaba con otra, realizando múltiples validaciones, como identificación de puntos iguales y alineación (ver Figura 5.6), con el fin de detectar diferencias de la segunda imagen comparada con la imagen muestra.

V. RESULTADOS Y DISCUSION

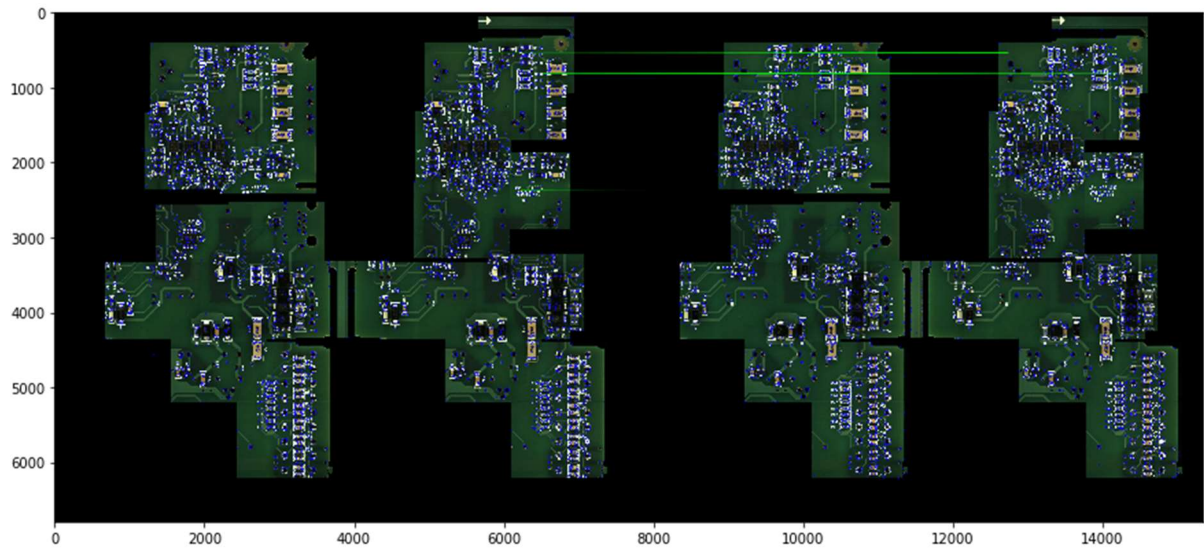


Figura 6.6 Alineación de imágenes.

Mediante el análisis de algunos histogramas detectaba las distintas partes de la imagen como fondo de la imagen, *pads* de soldadura, pistas eléctricas y hoyos en las tarjetas (ver Figura 5.7).

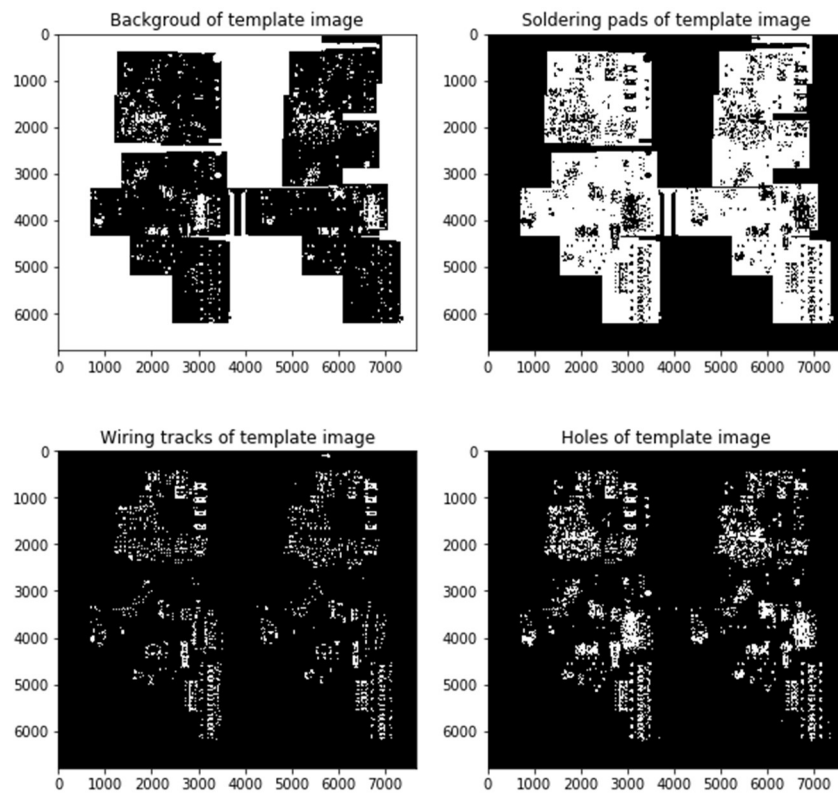


Figura 7.7 Identificación y separación de plantillas.

Después de utilizar múltiples operaciones, se lograba detectar algunas diferencias en las tarjetas, pero esta opción no era muy restrictiva, lo que generaba demasiados falsos (ver Figura 5.8).

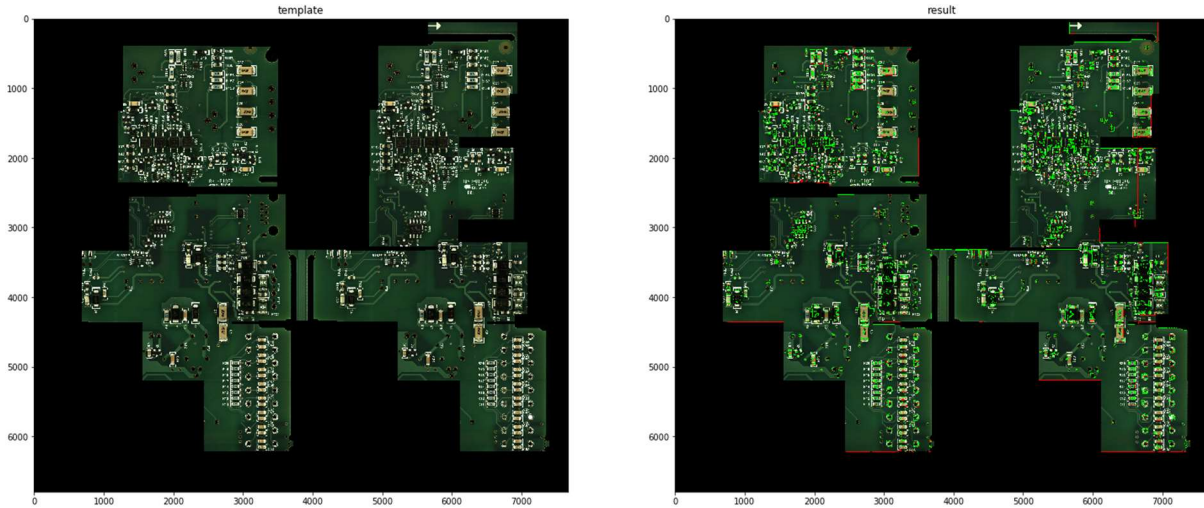


Figura 8.8 Detección de diferencias entre tarjeta muestra y tarjeta evaluada.

5.2.3 Detección usando distintas Redes Neuronales

Para esta prueba se utilizaron 3 distintas redes, el primer modelo de detección consistía en construir y entrenar una red neuronal convolucional (*CNN*) básica desde cero, obteniendo los resultados de la Figura 5.9.

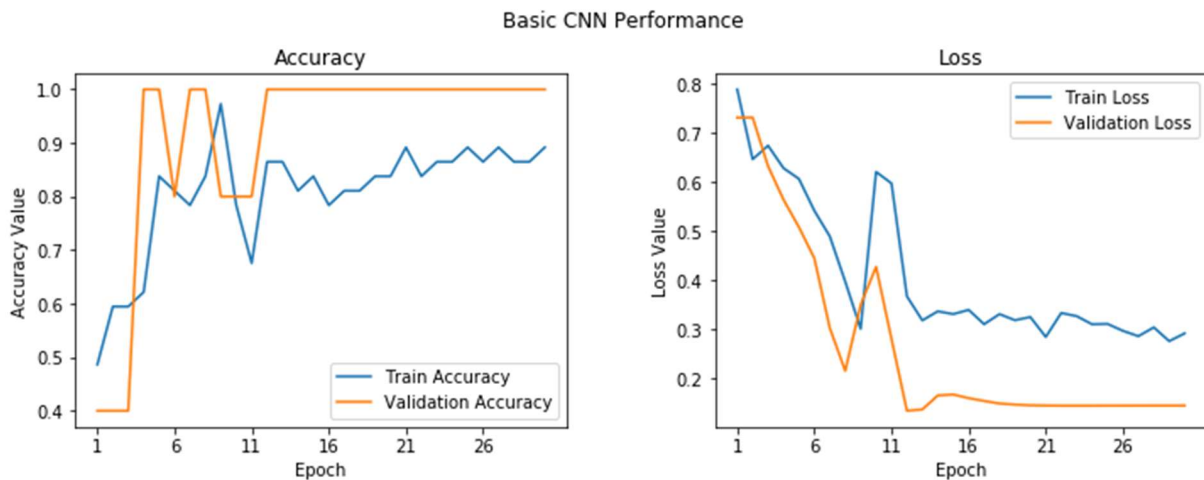


Figura 9.9 Resultados del entrenamiento y validación.

Para este modelo se guardó el *dataset* como *basic_cnn.h5* con un peso de 247MB.

V. RESULTADOS Y DISCUSION

El siguiente modelo se aprovechó *TensorFlow* para cargar el modelo VGG-19, y congelar los bloques de convolución añadiendo nuestras propias capas densas para realizar la tarea de clasificación. Se obtuvieron los siguientes resultados mostrados en la Figura 5.10.

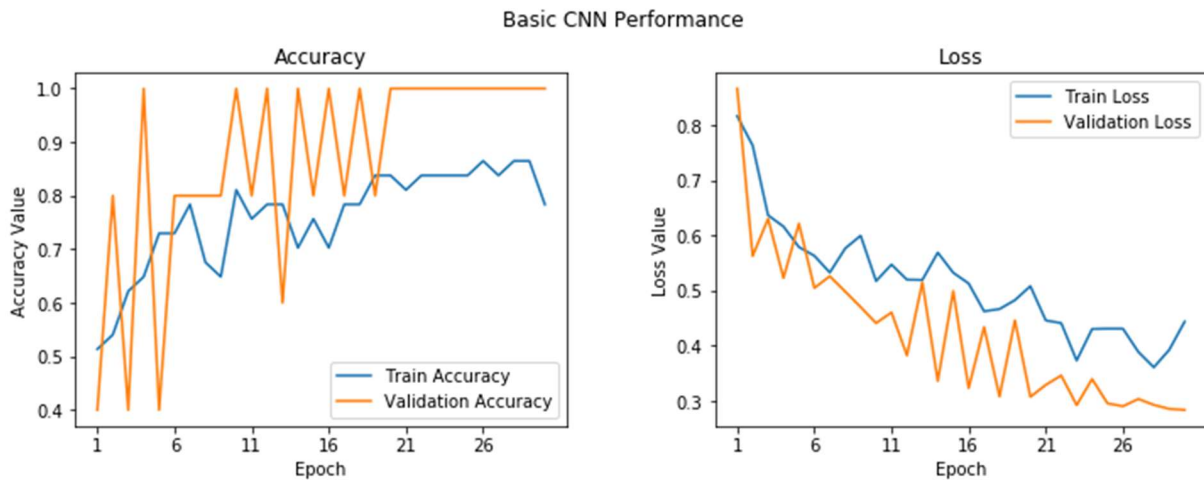


Figura 10.10 Resultados del entrenamiento y validación.

Para este modelo se guardó el *dataset* como *vgg_frozen.h5* con un peso de 110MB.

El último modelo utilizaba parte de la red anterior *VGG*, asegurándonos de que los dos últimos bloques del modelo *VGG-19* eran entrenables, aplicando una ampliación a las distintas imágenes (ver Figura 5.11).

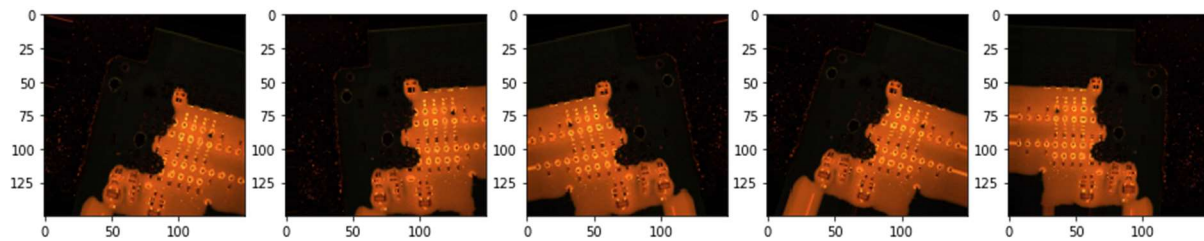


Figura 11.11 Ampliación y rotación de imágenes.

Después de aplicar el proceso de ampliar, se entrenó la red y se obtuvieron los resultados mostrados en la Figura 5.12.

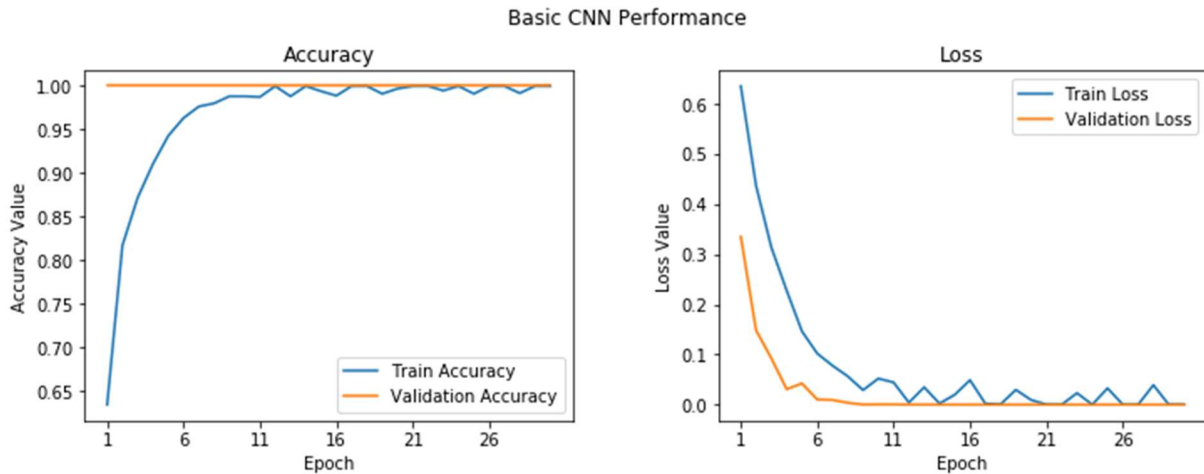


Figura 12.12 Resultados del entrenamiento y validación.

Para este modelo se guardó el *dataset* como *vgg_finetuned.h5* con un peso de 178MB.

Después de crear todos los *datasets*, se evaluaron los tres modelos diferentes que acabamos de construir en la fase de entrenamiento haciendo predicciones con ellos en los datos de nuestro conjunto de datos de prueba. Por lo que el último paso fue comprobar el rendimiento de cada modelo con las métricas de clasificación pertinentes (ver Figura 5.13).

Model Performance metrics:		

Accuracy: 0.6842	Accuracy: 0.7368	Accuracy: 1.0
Precision: 0.6815	Precision: 0.735	Precision: 1.0
Recall: 0.6842	Recall: 0.7368	Recall: 1.0
F1 Score: 0.6748	F1 Score: 0.7338	F1 Score: 1.0
'Basic CNN'	'VGG-19 Frozen'	'VGG-19 Fine-tuned'

Figura 13.13 Rendimiento de las distintas redes.

5.2.4 Detección final con *mask CNN*

Después de ver todas las posibles opciones que se podían usar, se comprobó que algunas ayudaban al desarrollo del proyecto, pero otras no eran aplicables a éste, por lo que después de indagar en todas las opciones se seleccionó la detección con *mask CNN* creando un *dataset*, lo que quiere decir que estábamos creando nuestro *dataset* desde 0, identificando nosotros donde estaban los defectos para que la red supiera donde buscar, lo que fue de beneficio ya que era más restrictivo comparado con las demás tipos de pruebas que se utilizaron, logrando una mejor detección de los defectos superior al 85%.

V. RESULTADOS Y DISCUSION

La primera fase de esta prueba se basó en pocas imágenes cerca de 20 para ver si ésta era capaz de lograr una mejor detección comparada con todas las pruebas anteriores, lo que se validó logrando una detección superior al 50% de las muestras en la predicción. Para mejorar el porcentaje de detección se decidió incrementar la cantidad de muestras a 80, logrando una mejor detección, pero aun con oportunidades de mejora por el tipo de identificación que se realizó durante el ajuste del *dataset*. En la figura 5.14 se muestra un ejemplo de los resultados obtenidos.

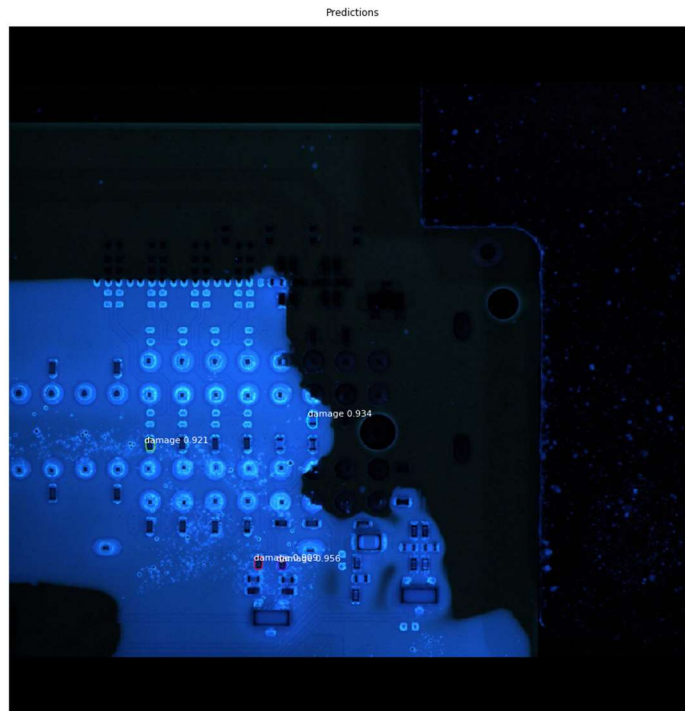


Figura 14.14 Predicción de muestras finales.

Durante las validaciones del entrenamiento se presentaron demoras, algunas de cerca de 10 horas las cuales consumían demasiada memoria del equipo, lo que hizo que en 3 ocasiones se abortara el entrenamiento.

VI. CONCLUSIONES

Durante el transcurso de esta investigación, se observaron muchas variables para el proyecto, las cuales fueron contenidas o descartadas, ya que era un universo muy grande de opciones disponibles que afectaban al desarrollo del proyecto. Como se observó durante las etapas de este proyecto, se pueden realizar mejoras en el proceso de creación de los *dataset* que son benéficas para el futuro del proyecto, logrando mejor detección de defectos en la mayoría de las imágenes que se proporcionan.

Inicialmente el proyecto estaba basado en la detección de defectos en las tarjetas electrónicas pero detectando presencia, similitud de los componentes y considerando un modelo de tarjeta de baja complejidad y con pocos componentes como se ve en la figura 6.1, pero derivado del cambio de equipo, se empezó a trabajar en la detección de defectos en las tarjetas electrónicas pero en aplicación de *conformal* que es un tipo de plástico que recubre las tarjetas para protegerlos de polvo, humedad (ver Figura 6.2).

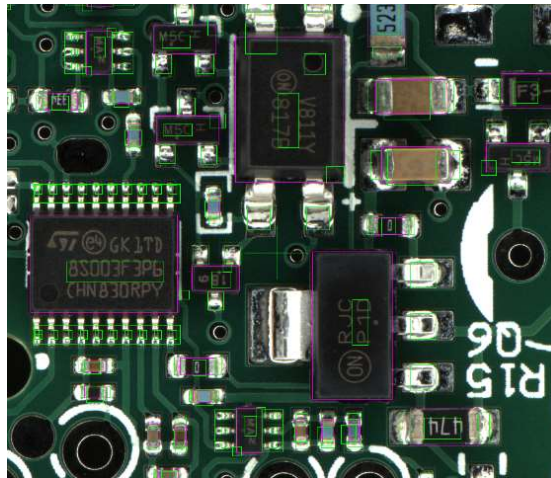


Figura 15.1 Muestra inicial.

VI. CONCLUSIONES

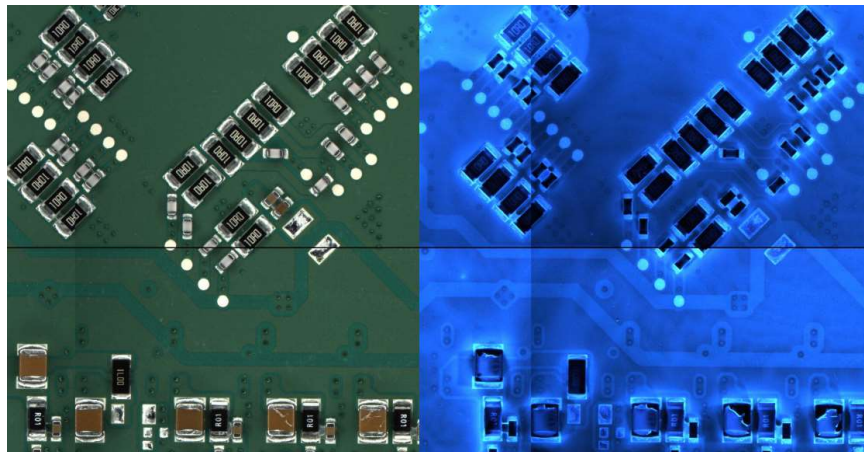


Figura 16.2 Muestra final con *conformal*

En lo que respecta al proyecto, inicialmente no se conocía alguna forma de realizar los *datasets* o los que estaban disponibles en la red no eran adecuados o no existían para este tipo de actividad, por lo que se debió desarrollar un propio *dataset* y aprender sobre la marcha como hacerlo de la mejor manera.

Se logró que el SI identificara defectos al menos en un 85% en las tablas electrónicas con *conformal*, este proyecto es muy extenso ya que se requiere mayor tiempo para lograr un sistema con un mejor desempeño y que pueda reemplazar en la mayoría de las ocasiones a una persona en esta actividad. El *dataset* no fue tan extenso en tamaño, aproximadamente 50MB, pero se debe considerar que en la industria es muy difícil conseguir tantas tarjetas con defectos para hacer un extenso *dataset*, todo esto incurre en un gasto monetario extra y material desechado, por lo que la empresa solo proporciono el poco material que tenía defectos y sobre ese material, se trabajó el proyecto, para expandir el universo de detección y mejorar, se debe sacrificar material y es algo que se evita en cualquier industria.

Las empresas actualmente destinan múltiples recursos para ejecutar sus funciones, pero muchas veces, no se aprovechan los recursos con los que ya cuentan, el propósito de este proyecto fue demostrar que se tienen los equipos, pero estos podrían aprovecharse de una mejor manera o complementarse y lograr un mejor desempeño de sus actividades, creando una innovación en sus procesos y reducir una carga para el personal y solo ocupar su tiempo en las tarjetas que el equipo esta casi 100% seguro que son defecto y no una variación del proceso.

VII. BIBLIOGRAFIA

- Abhishek Dutta and Andrew Zisserman. 2019. The VIA Annotation Software for Images, Audio and Video. In Proceedings of the 27th ACM International Conference on Multimedia (MM '19), October 21–25, 2019, Nice, France. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/3343031.3350535>.
- Cognex. Qué es la visión artificial (2018). Recuperado 30 mayo, 2018, de <https://www.cognex.com/es-ar/what-is/machine-vision/what-is-machine-vision>
- Conacyt. Revista Ciencia y Desarrollo. (2018). Recuperado 15 mayo, 2018, de <http://www.cyd.conacyt.gob.mx/Multimedia/multimedia-Inteligencia-Artificial-4.html>
- Coursera Inc (s.f.). Recuperado 24 noviembre, 2020, de <https://www.coursera.org/learn/machine-learning>
- D'Aquila, Raimundo O. (2005). Acerca de la INTELIGENCIA...de los SISTEMAS INTELIGENTES. ANALES TOMO I Año 2005. Recuperado 5 junio, 2018, de http://www.acadning.org.ar/indice_anales_2005.htm
- G. Acciani, G. Brunetti and G. Fornarelli (2006). A Multiple Neural Network System to Classify Solder Joints on Integrated Circuits. International Journal of Computational Intelligence Research, 2(4), 337-348.
- Haykin S. (1994). Neural Networks, A Comprehensive Foundation. Macmillan College Pub. Comp.
- Hertz J., Krogh A. y Palmer R. G. (1991), Introduction to the Theory of Neural Computation. AddisonWesley. Pub. Comp.
- InteractiveChaos (s.f.). Recuperado el 26 noviembre, 2020, de <https://www.interactivechaos.com/manual/tutorial-de-machine-learning/arquitectura-de-redes-neuronales> 516
- Janczki, M., Becker, A., Jakab, L., Grf, R., & Takcs, T. (2013). Automatic Optical Inspection of Soldering. Materials Science - Advanced Topics. <https://doi.org/10.5772/51699>
- Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick (2017). Mask R-CNN.

VII. BIBLIOGRAFIA

<https://arxiv.org/abs/1703.06870>

- Keras. (s.f.). Recuperado 24 noviembre, 2020, de <https://keras.io/>
- Khan Academy. Algorithms (s.f.). Recuperado 5 junio, 2018, de <https://www.khanacademy.org/computing/computer-science/algorithms>
- Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org
- Microsoft. Conceptos básicos sobre bases de datos. (s.f.). Recuperado 5 junio, 2018, de <https://support.office.com/es-es/article/conceptos-b%C3%A1sicos-sobre-bases-de-datos-a849ac16-07c7-4a31-9948-3c8c94a7c204>
- Pressman, Roger S. (2005), Ingeniería del Software. McGrawhill, España.
- Python Software Foundation. (s.f.). Recuperado 24 noviembre, 2020, de <https://docs.python.org/3/faq/general.html#what-is-python>