ISSN 1688-2806

Universidad de la República
Facultad de Ingeniería

Visión Artificial de Bajo Mantenimiento para Inspección Automática de PCBAs

Tesis presentada a la Facultad de Ingeniería de la Universidad de la República por

Federico Nin

en cumplimiento parcial de los requerimientos para la obtención del título de Magister en Ingeniería Eléctrica.

Directores de Tesis

Dr. Ing. Federico Lecumberry ..... Universidad de la República
Dr. Ing. Javier Preciozzi ..... Universidad de la República

Tribunal

Dr. Ing. Álvaro Gómez ..... Universidad de la República
Dr. Ing. Nicolás Pérez ..... Universidad de la República
Dr. Ing. Leonardo Steinfeld ..... Universidad de la República

Director Académico

Dr. Ing. Federico Lecumberry ..... Universidad de la República

Montevideo
miércoles 22 octubre, 2025

Visión Artificial de Bajo Mantenimiento para Inspección Automática de PCBAs,
Federico Nin.

ISSN 1688-2806

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 95 páginas.
Compilada el miércoles 22 octubre, 2025.
http://iie.fing.edu.uy/

Agradecimientos

Quisiéramos expresar nuestro más sincero agradecimiento al equipo de Integer Montevideo por su invaluable colaboración, dedicando su tiempo, recursos, habilidades y conocimiento para el desarrollo de este trabajo. Su apoyo ha sido fundamental para alcanzar los objetivos planteados.

Asimismo, extendemos nuestra gratitud a nuestros tutores, cuyo asesoramiento y guía han sido esenciales para la correcta ejecución de este proyecto. Sus consejos y aportes han enriquecido significativamente nuestra investigación.

Finalmente, queremos agradecer a ClusterUY [37] (https://cluster.uy) por proporcionarnos los recursos computacionales necesarios.

A todos ustedes, muchas gracias.

Esta página ha sido intencionalmente dejada en blanco.

Resumen

Español

La industria de producción de electrónica es esencial en la actualidad y se apoya en herramientas automatizadas para mantener altos niveles de calidad a costos reducidos. A pesar de los recientes avances en el área del aprendizaje automático y la visión artificial, las soluciones habituales de *Automated Optical Inspection* (AOI) en *Printed Circuit Board Assembly*s (PCBAs) siguen siendo basadas fuertemente en técnicas clásicas, como la segmentación manual, que si bien tienen buen rendimiento, conllevan elevados costos de mantenimiento. En este trabajo, proponemos aplicar algoritmos modernos de aprendizaje automático combinados con información de *Computer Assisted Design* (CAD) de los PCBAs para generar soluciones de AOI con buen desempeño y bajo mantenimiento.

Adaptamos los métodos de aprendizaje por transferencia y redes siamesas al problema de detección de fallas en PCBAs, sobre conjuntos de datos de componentes electrónicos, tanto públicos como originales de este trabajo. Los resultados muestran que los algoritmos desarrollados requieren menos mantenimiento que los métodos tradicionales y son suficientemente efectivos para detectar errores de componentes incorrectos, faltantes o en exceso, aunque presentan limitaciones en la detección de errores de posicionamiento o soldadura.

Mostramos que es posible usar visión artificial con aprendizaje automático para abordar el problema de manera flexible como un apoyo para el inspector humano, dando una mejora sustancial en el tiempo de inspección por placa. Sin embargo, hallamos que la carencia de conjuntos de datos públicos y de buena calidad para componentes electrónicos y fallas en PCBAs es el principal impedimento para el desarrollo de un sistema general robusto de AOI.

Existe un camino prometedor para el avance del AOI incorporando nuevas tecnologías de aprendizaje automático, con un enfoque en el bajo mantenimiento que eventualmente mejorará la eficiencia de los sistemas de control de calidad en la industria electrónica.

English

The electronics manufacturing industry is essential in the modern society and relies on automated tools to maintain high levels of quality at reduced costs. Despite recent advances in machine learning and computer vision, common AOI solutions for PCBAs are still heavily based on classical techniques, such as manual

segmentation, which demand a high maintenance costs for their good performance. In this work, we propose applying modern machine learning algorithms combined with CAD information of PCBAs to generate low-maintenance AOI solutions with good performance.

We adapt transfer learning methods and Siamese networks to the problem of fault detection in PCBAs, using datasets of electronic components, both public and specifically created for this study. The results show that the developed algorithms require less maintenance than traditional methods and are sufficiently effective in detecting incorrect, missing, or excess components, although they have limitations in detecting positioning or soldering errors.

We demonstrate that it is possible to use computer vision with machine learning to address the problem flexibly, supporting the human inspector and substantially improving inspection time per board. However, we found that the lack of public and high-quality datasets for electronic components and PCBAs failures is the main impediment to developing a robust general AOI solution.

There is a promising path for the advancement of AOI by incorporating new machine learning technologies, focusing on low maintenance, which will eventually improve the efficiency of quality control systems in the electronics industry.

Tabla de contenidos

Tabla de contenidos

Glosario

Tanto para el área de aprendizaje automático como para la industria de manufactura de circuitos electrónicos es muy habitual, y en ocasiones inevitable, usar nombres en inglés para denominar técnicas, procesos, máquinas, componentes, objetos, etc. Además de incluir este glosario para las abreviaciones más comunes, intentamos utilizar las palabras equivalentes en castellano cuando es posible. Cuando no lo es, o resulta inconveniente, usamos las palabras en inglés con tipografía *cursiva*.

AOI *Automated Optical Inspection*

PCB *Printed Circuit Board*

PCBA *Printed Circuit Board Assembly*

LED *Light-Emitting Diode*

DSLR *Digital Single-Lens Reflex*

SMD *Surface-Mount Device*

BGA *Ball Grid Array*

IC *Integrated Circuit*

CAD *Computer Assisted Design*

BOM *Bill of Materials*

RGB *Red, Green, Blue*

ILSVRC *ImageNet Large Scale Visual Recognition Challenge*

ROC *Receiver Operating Characteristic*

RMSE *Root Mean Square Error*

FPR *False Positive Rate*

TFT-LCD *Thin Film Transistor - Liquid Crystal Display*

CSV *Comma Separeted Values*

NXCORR *Normalized Cross-Correlation*

PCA *Principal Component Analysis*

GA *Genetic Algorithm*

GLCM *Grey-Level Co-occurrence Matrix*

SURF *Speeded-Up Robust Features*

HOG *Histogram of Oriented Gradients*

SVM *Support Vector Machine*

CNN *Convolutional Neural Network*

kNN *k Nearest Neighbor*

MLP *Multi-Layer Perceptron*

OCR *Optical Character Recognition*

Capítulo 1

Introducción

Este trabajo se enfoca en el estudio de técnicas de visión artificial para detectar fallas en la producción de circuitos electrónicos. El presente capítulo es una introducción al contexto de la producción industrial de los mismos, abordando problemas comunes y revisando el estado del arte académico y comercial. Proporcionamos al lector una comprensión del dominio del problema, así como herramientas para entender la motivación y el objetivo de este trabajo.

1.1. Producción Industrial de Circuitos Electrónicos

La producción de circuitos electrónicos a nivel industrial es un proceso relativamente estandarizado y secuencial. Suele comenzar con una placa sin componentes llamada *Printed Circuit Board* (PCB) y terminar en un producto cerrado para un usuario final.

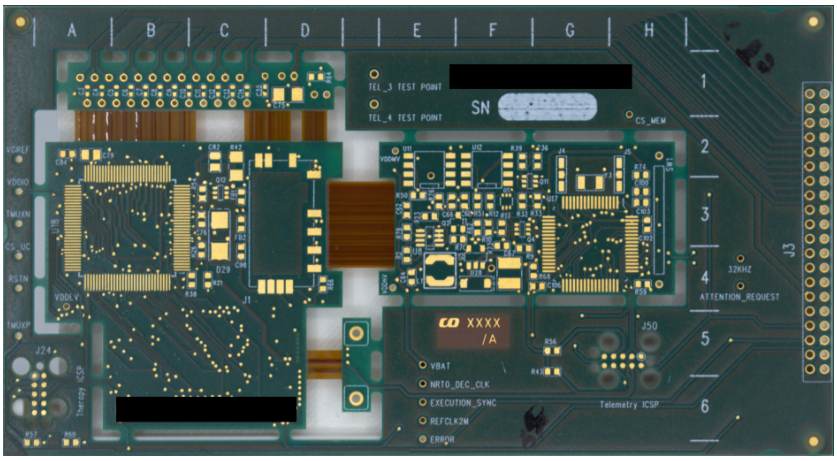
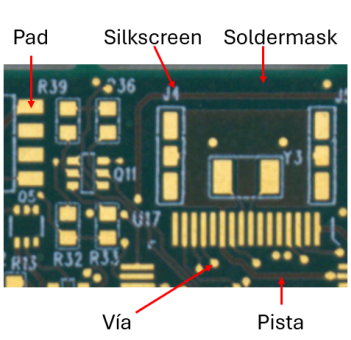


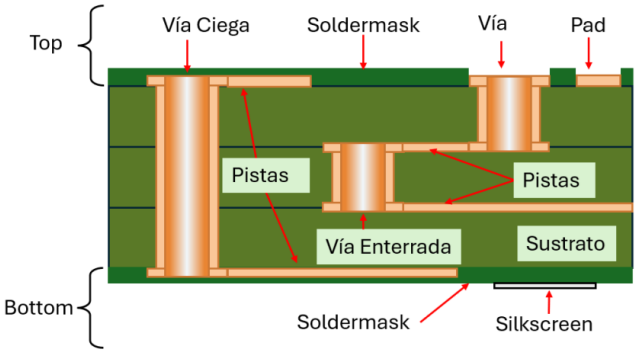
Figura 1.1: Ejemplo de un PCB donde los componentes aun no están soldados. Zonas enmascaradas en negro por confidencialidad.

El PCB (figura 1.1) es típicamente un apilado de capas de fibra de vidrio y cobre. Las distintas capas pueden interconectarse por "vías" (*vias* en inglés),

mientras que dentro de una misma capa las conexiones se realizan por "pistas" (*traces* en inglés). Las capas de los extremos visibles, conocidas como *top* y *bottom* tienen cobre expuesto sólo en ciertos lugares definidos por el diseñador, que son principalmente los *pads* donde se soldarán los *pins* de los componentes. En diseños sencillos, se montan componentes de un único lado del PCB, pero es posible hacerlo de ambos lados para disminuir la superficie total. El típico color verde de los PCBs lo da la máscara de soldadura (*solder mask* en inglés), que es el material que enmascara al cobre para solo exponer los lugares deseados. Los textos blancos son la serigrafía (*silkscreen* en ingés) y proporcionan ayudas para el ensamblaje o *debugging*. En la imagen 1.2 pueden verse un ejemplo real y un esquema mostrando la estructura y los elementos mencionados.



(a) Sección de un PCB destacando *pads*, *silkscreen*, *soldermask*, vías y pistas. El cobre expuesto está enchapado en oro.



(b) Corte vertical del PCB mostrando el apilado de sustrato de fibra de vidrio y cobre con sus interconexiones internas. Notar que el *soldermask* solo expone algunas partes deseadas del cobre y enmascara la mayor parte.

Figura 1.2: Estructura de un PCB

El primer paso en la producción del circuito electrónico es cubrir el cobre expuesto con estaño, lo que se conoce como el empastado. Para esto pueden utilizarse máquinas automáticas que, con la ayuda de un *stencil* (una máscara complementaria al *solder mask*), aplican una pasta de estaño en los lugares deseados. El estaño va acompañado de *flux*, una sustancia que disuelve los posibles óxidos presentes en los metales y favorece el flujo del metal fundido para que pueda cubrir toda la superficie.

Luego, se posicionan los componentes de montaje superficial (SMD, figura 1.3) con un robot conocido como *pick-and-place*. Este robot recoge con una bomba de aire los componentes y los deposita en las coordenadas y orientaciones especificadas (figura 1.4). Este proceso recibe el nombre de "populado".

Con los componentes ya ubicados sobre las placas, estas pasan por una cinta transportadora dentro de un horno de convección que funde el estaño, fijando los componentes en su lugar. Este proceso es conocido como *reflow soldering*. Los pasos se ejecutan desde el empastado hasta el horno para cada lado de la placa que tenga componentes (*top* y/o *bottom*). En este punto, al conjunto de placa y componentes soldados se le llama *Printed Circuit Board Assembly* (PCBA), como

(a) Resistencias SMD (b) Condensadores SMD (c) Circuito integrado SMD

Figura 1.3: Ejemplos de componentes SMD

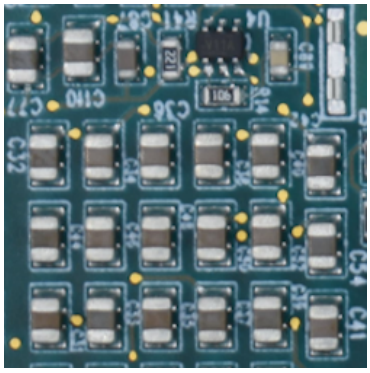


Figura 1.4: Componentes SMD soldados sobre el PCB

se ve en la figura 1.5.

Inspeccionar el PCBA es una parte crucial del control de calidad porque permite detectar errores en una etapa temprana, lo que evita arreglos o descartes costosos, o incluso fallas en campo. Es habitual usar máquinas de inspección óptica automática (AOI) que adquieren fotografías de los PCBAs y detectan autónomamente fallas en los componentes, su posicionamiento o las soldaduras. Algunos tipos de encapsulados tienen conexiones que quedan ocultas a la visión, por lo que pueden hacerse inspecciones ópticas con Rayos X, además de en el espectro visible.

El proceso sigue con el montaje de los componentes *through-hole*, es decir, cuyos *pins* atraviesan la placa y no se apoyan sobre un *pad*. Esto puede comprender soldaduras directas de cables, componentes discretos de gran tamaño, etc. Si bien la soldadura y el montaje pueden ser automatizados, muchas veces es un procedimiento manual.

Una vez que todos los componentes están soldados, puede hacerse un lavado para quitar restos de *flux* y otros contaminantes y se procede a una verificación funcional del circuito. Esto suele comprender ensayos y medidas que verifican que el comportamiento del circuito es acorde a su especificación. Finalmente, se ensambla el aparato y se empaca para su distribución.

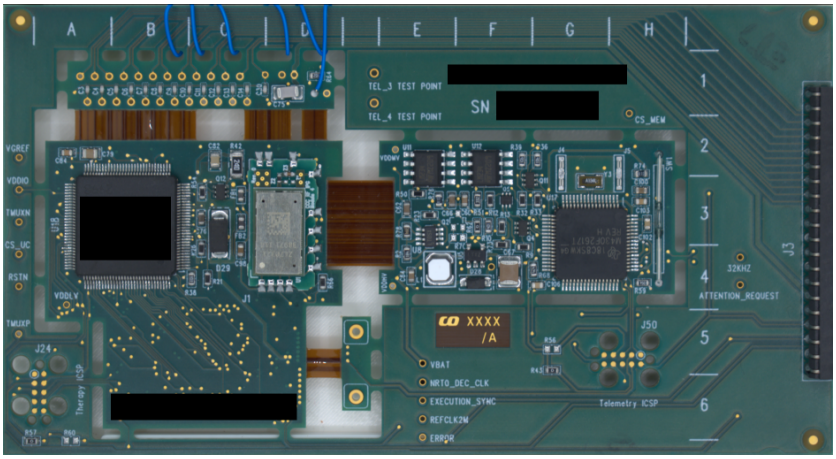


Figura 1.5: PCBA completamente ensamblado. Componentes SMD y *through-hole* soldados. Zonas enmascaradas en negro por confidencialidad.

1.2. El Rol de la AOI en la Producción

Si bien es esperable que un proceso automático de empastado, montaje y soldadura logre buenos resultados y supere a un proceso manual en exactitud y velocidad, no está exento de errores. Se pueden encontrar componentes incorrectos, ausentes, sobrantes, en cortocircuito, sin soldar o con soldaduras de mala calidad. Estas fallas pueden provocarse por diversos motivos, como por ejemplo un error humano al cargar los componentes en la *pick-and-place* o un error del robot al depositar el componente. Por otro lado, durante la soldadura, la variabilidad de las posiciones de los componentes junto con las masas de cobre presentes en el PCB pueden afectar la transferencia del calor e impedir que el componente se suelde en la posición correcta, induciendo fallas en la soldadura (ver figura 1.6). Las fallas, los criterios y tolerancias de aceptación son definidos en la normativa IPC [1].

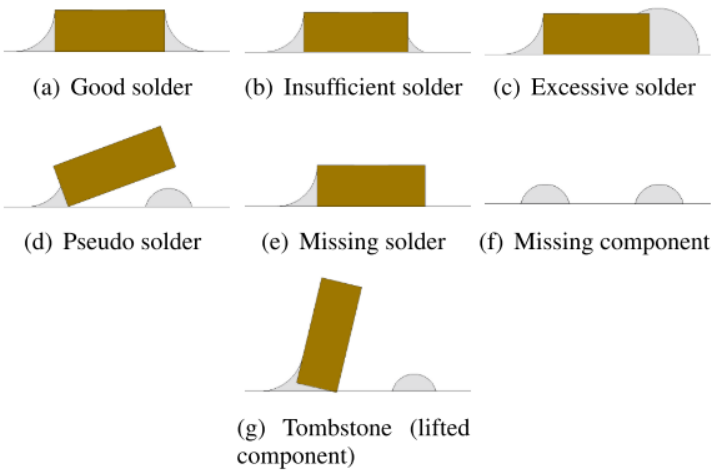


Figura 1.6: Defectos comunes de soldadura, tomado de [2]

Cualquiera de estos errores puede resultar en comportamientos impredecibles en el circuito. Estos van desde fallas graves en la línea de producción, como un cortocircuito que destruye la placa al energizarla, hasta problemas más sutiles que podrían manifestarse durante una verificación funcional, como un condensador o una resistencia con un valor incorrecto que afecte ligeramente la frecuencia de un oscilador o cause un consumo inesperado. Además, estos errores podrían desencadenar problemas críticos en el campo, como un fusible inapropiado que no proteja eficazmente al circuito. La evaluación del costo de cada tipo de falla y sus consecuencias, como el descarte completo, la reparación en la línea de producción o reclamaciones de clientes, depende del dominio de trabajo y de la criticidad del dispositivo producido. Sin embargo, en todos los casos suele ser deseable mantener altos estándares de producción a costos bajos.

La inspección visual, tanto automática como manual, es una buena herramienta para detectar algunos de estos errores en etapas tempranas. Para productos de volúmenes muy altos, automatizar este paso se vuelve extremadamente valioso, por lo que suele ser el estándar en la industria contar con un paso de AOI tras la soldadura. Esto permite funcionar a velocidades muchísimo más altas que las que un humano puede alcanzar y evita el cansancio de los operarios, que induce al error por falta de atención. Si bien se pueden alcanzar rendimientos muy buenos, como todo sistema de visión artificial, la AOI viene con sus propios problemas asociados. Existe una variabilidad enorme en el aspecto visual de los componentes, como por ejemplo en la forma y tamaño del encapsulado o de los *pins*, en los *markings*, en los colores, etc. Estas diferencias ocurren por diversos motivos, como tolerancias de fabricación o diferencias entre proveedores. En consecuencia, es difícil construir un sistema de AOI robusto y versátil, por lo que no es raro contar con un humano que supervise el resultado de la AOI para descartar falsos positivos.

Para productos de volúmenes pequeños o que cambian rápidamente por estar en fases de diseño, el costo de mantenimiento de un sistema de AOI debido a la variabilidad de los componentes se vuelve considerable. Reconfigurar el sistema para ingresar componentes alternativos o cambiar tolerancias en la inspección con toda seguridad va a detener la producción por un tiempo, por lo que si ocurre habitualmente, deja de ser una solución rentable. El balance entre la automatización y la mantenibilidad del sistema de AOI es una de las motivaciones centrales de este trabajo, por lo que volveremos sobre este tema en las secciones siguientes.

1.3. Estado del Arte

1.3.1. Estado del Arte Comercial

Las soluciones comerciales de AOI para PCBAs son usualmente sistemas cerrados e integrados (desde la adquisición hasta la inspección), de alta complejidad y costo, diseñados para trabajar con volúmenes grandes de producción, a alta velocidad y con buenos desempeños. Algunos ejemplos pueden verse en la figuras 1.7a, 1.7b y 1.7c.

Estos sistemas pueden incorporar arreglos de luces de colores a distintas alturas

(a) Saki 3Di-MS3 (b) Orbotech Discovery II

(c) Mirtec MV-3L (d) AgnosPCB

Figura 1.7: Sistemas comerciales de AOI

y orientaciones, a veces múltiples cámaras o cámaras móviles, una plataforma móvil para el PCB y ópticas de alta calidad, como lentes telecéntricos. Están diseñadas para detectar una amplia gama de fallas, incluyendo componentes incorrectos, mala posición y errores de soldadura. Dada su complejidad, los costos de estas máquinas fácilmente sobrepasan varias decenas de miles de dólares estadounidenses.

El principio de funcionamiento de estos sistemas requiere un extenso conocimiento *a priori* del problema: la máquina debe tener un mapa del PCBA, conocer el tipo de encapsulado presente en cada coordenada y saber la estructura interna del encapsulado para identificar la ubicación de pines, marcas, soldaduras y polaridad, entre otros. Además, debe tener información sobre las tolerancias de traslación y otros límites establecidos por la normativa IPC. La tarea de dar de alta esta información en la máquina, si bien suele ser asistida por *software*, recae sobre un operario humano, quien debe ingresar manualmente coordenadas y tolerancias para cada uno de los cientos de componentes. Como consecuencia, estas máquinas tienden a ser poco robustas frente a cambios y exigen una inversión de tiempo de puesta en marcha y mantenimiento elevado. El costo es justificable solo cuando los volúmenes son suficientemente grandes.

Sin embargo, más recientemente, han comenzado a aparecer otros sistemas que

reconocen estas dificultades y la necesidad de una parte de la industria de un tener una solución de AOI más flexible y de menor mantenimiento. Este es el caso de AgnosPCB, una empresa española que lanzó su producto a finales de 2020 (figura 1.7d). Si bien venden un sistema de adquisición diseñado por ellos, el *hardware* es esencialmente abierto, opcional y de baja complejidad: consiste en una caja blanca con luces LED con difusores, una cámara DSLR *off-the-shelf* sobre un brazo de altura regulable y una computadora embebida de poca potencia. La innovación del sistema radica en el *software* que hace la inspección, que trabaja con el paradigma de *golden sample*. La idea es tener una imagen de referencia de un PCBA sin fallas (el *golden sample* o *golden board*) y luego compararla con la imagen del PCBA a inspeccionar. Según AgnosPCB, esta comparación se hace con una red neuronal que fue entrenada en una base de datos propietaria de fallas etiquetadas de PCBAs. El *software* de inspección luego resalta las diferencias significativas encontradas.

Este sistema tiene ventajas muy deseables para fabricantes de pequeña y mediana escala, o de alta variedad (*high-mix*), pues permite automatizar la inspección visual con una inversión baja de capital y tiempo. La única entrada al sistema es la imagen del *golden board*, así que el mantenimiento debería ser tan sencillo como adquirir nuevamente una sola imagen de una placa buena. En cambio, un sistema tradicional llevaría mucho más tiempo de configuración inicial, especificando cada componente con sus tolerancias y potencialmente más tiempo de mantenimiento para tener en cuenta variaciones en los componentes como *markings* o colores.

1.3.2. Estado del Arte Académico en AOI

La visión artificial como mecanismo de control de calidad se ha utilizado industrialmente por décadas, por lo que es un tema con mucho contenido publicado. Sin embargo, hasta donde sabemos, no existen revisiones bibliográficas de AOI para PCBAs específicamente. El lector podrá consultar en [43] y [2] revisiones del estado del arte de la detección de defectos por visión artificial en la industria general y en la electrónica en particular, respectivamente, donde se cubren temas relacionados a los PCBAs, entre otros.

El AOI para PCBAs es un área fuertemente impulsada por la industria, por lo que los contenidos tienden a ser más bien aplicaciones prácticas, enfocadas a un problema muy concreto y, frecuentemente, sobre bases de datos privadas e inaccesibles al público. Esto impide reproducir los resultados publicados y comparar desempeños o mejoras. A su vez, pone una barrera de entrada difícil de superar para personas ajenas a la industria, puesto que es muy costoso o directamente imposible crear una base de datos de buena calidad sin tener acceso a una línea de producción.

Por otro lado, hallamos en estas revisiones las dos grandes familias de soluciones del aprendizaje automático: los métodos tradicionales basados en extracción de características, donde las características se seleccionan y procesan manualmente antes del aprendizaje y los métodos de aprendizaje profundo, donde las características se aprenden automáticamente a partir de los datos crudos. Si bien el advenimiento del aprendizaje profundo en la última década ha abierto la puerta

a muchísimas alternativas y mejoras a las técnicas tradicionales, estas aún abundan y siguen siendo sujeto activo de estudio, por lo que es valioso incluirlas en el análisis.

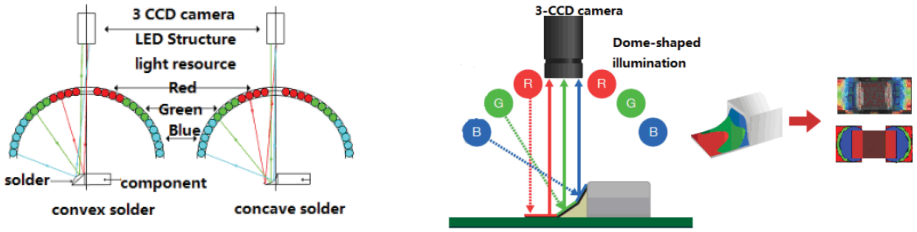


Figura 1.8: Adquisición propuesta por [52] para clasificar soldaduras. A la izquierda, se ilustra un corte vertical de un domo de iluminación RGB, con anillos de colores distintos según la altura. Las soldaduras convexas y cóncavas generan un patrón de reflexión distinto. A la derecha, se muestra el mismo sistema y el patrón de colores resultante sobre la soldadura que se utiliza para su clasificación.

Desde el punto de vista de la adquisición, se encuentran varias soluciones: arreglos con una cámara única [50] [52], cámaras múltiples o estereoscópicas [47], así como plataformas móviles para trasladar las unidades inspeccionadas [32]. Hay varios ejemplos de sistemas de iluminación especiales (oblicuos y con superficies reflectivas [49] o luz polarizada o filtros de polarización [53]) para detectar fallas en texturas. También existen soluciones con luz estructurada para hacer reconstrucción 3D de las soldaduras con distintos anillos de LEDs de colores a alturas y ángulos variados [52] [51] [50], como puede verse en la figura 1.8. Lo más habitual suele ser uno o varios anillos de LEDs coaxiales [50].

En relación con los algoritmos de clasificación y detección de errores, coexisten soluciones clásicas de visión artificial con enfoques más modernos. Es frecuente encontrar el uso de *template matching* como en [26] y [50], donde se usa NXCORR para hallar regiones específicas de la imagen a inspeccionar usando otra imagen de referencia.

Encontramos una gran diversidad de técnicas para la extracción de características. PCA [19] es ampliamente utilizado, como por ejemplo en [57] en la detección de fallos de soldadura láser. GA [44] se usa en [20] para inspección de soldadura de arco y en [46] para clasificación de soldaduras de componentes SMD. GLCM [15] es usado en [42] para detección de fallas en telas y en [56] para detección de fallas en pantallas TFT-LCD. El detector de esquinas de Harris [16] es utilizado en [3] para detectar la posición de componentes SMD en un PCBA. Otsu *thresholding* [38] puede hallarse en [54] para detección de fallas en superficies altamente reflectivas de metal y en [40] y [39] para detección de fallas en LEDs. SURF [5] es usado en [58] y [17] para registración y detección de fallos en PCBAs. HOG [11] es utilizado en [14] para detección de fallos en caños. *Wavelet Transform* [36] se puede encontrar en el ya mencionado [54], en [55] para detección de fallos en telas y en [8] para detección de fallos de componentes en PCBAs.

Entre los clasificadores utilizados se encuentra con mucha frecuencia SVM [10], como por ejemplo en [30] para detección de fallos en obleas de silicio, así como en

los ya mencionados [54], [20], [14], [46] y [57]. kNN es utilizado en el ya mencionado [55]. *Decision Trees* pueden hallarse en [22] y [18] para inspección de contactos y de soldadura respectivamente. Así mismo, MLP es utilizado en [7] para detección de fallos en BGA. En trabajos más recientes encontramos el uso generalizado de CNNs [29], como en [21] para detección de fallos en semiconductores, en [23] para detección de fallas en texturas y en [24] y [31] para detección de fallos en componentes electrónicos. En particular, [21] y [23] utilizan la técnica de aprendizaje por transferencia (*transfer learning*) para reducir el conjunto de datos de entrenamiento requerido, lo que ayuda a compensar la falta de bases de datos públicas. [23] será un trabajo de referencia en secciones posteriores donde intentaremos adaptarlo al dominio de componentes electrónicos.

En [50] se presenta un sistema de inspección clásico de soldadura que requiere al usuario segmentar a mano las regiones de interés (*markings*, *pins* y *pads*) en una imagen de referencia. Luego usa *template matching* para encontrar la posición de cada región en la imagen a inspeccionar y toma umbrales programados manualmente acordes a la normativa IPC [1]. Si bien tiene buen desempeño, requiere mucha información *a priori* y es muy costoso de crear y mantener. Es muy similar a lo que realizan algunos sistemas comerciales de AOI que mostramos en la sección anterior.

Por otro lado, [35] y [31] presentan sistemas más modernos basados en aprendizaje profundo, pero pensados esencialmente como clasificadores de componentes. [31] hace énfasis en la dificultad del problema de clasificación debido a la gran variedad de clases de componentes existentes, alta variabilidad intra-clase y la falta de bases de datos públicas. De hecho, el problema es tan complejo que solo alcanzan un buen desempeño porque lo restringen a un dominio muy acotado de limitada utilidad práctica: [31] clasifica en dos clases “resistencia” y “condensador”, mientras que [35] clasifica en 14 clases habituales en PCBAs, pero que están lejos de poder cubrir la cantidad de clases posibles en la realidad. Estos sistemas son muy limitados pues la clasificación del componente sólo resuelve algunos tipo de fallas (componente incorrecto o faltante) y es, en la práctica, imposible entrenar un modelo con suficientes clases para que sea útil puesto que existe una variedad gigantesca de componentes con encapsulados distintos.

1.4. Objetivo y Alcance

El objetivo del presente trabajo es profundizar en el conocimiento de los sistemas de AOI para PCBAs y desarrollar y evaluar distintos métodos de detección de fallas, usando visión artificial clásico y técnicas modernas de aprendizaje profundo. Definimos los siguientes tipos posibles de fallas en un PCBA:

(E.1) Falta de componente: debería haber un componente y no lo hay.

(E.2) Exceso de componente: no debería haber un componente y sí lo hay.

(E.3) Tipo de componente incorrecto: el componente presente no es el que debería haber.

(E.4) Error de posicionamiento: el componente está trasladado o rotado de manera que no está apoyado correctamente sobre los *pads*.

(E.5) Error de polaridad: el componente hace buen contacto con todos los *pads*, pero su orientación no es la correcta.

(E.6) Error de soldadura: la soldadura es defectuosa, de mala calidad o inexistente.

Este trabajo se centrará principalmente sobre las fallas (E.1), (E.2) y (E.3), mientras que intentaremos evaluar la viabilidad de algunas soluciones para (E.4) y (E.5).

La detección de fallas de soldadura (E.6) es de una naturaleza muy distinta a los demás; requiere técnicas completamente diferentes y notablemente más complejas, por lo que queda fuera del alcance de este trabajo.

Notamos que al usar técnicas de visión artificial, especialmente las basadas en aprendizaje automático, no es prácticamente viable demostrar cumplimiento con la normativa IPC de soldaduras, por lo que el sistema debe ser visto como un apoyo para el inspector humano. Incluso así, esto puede redundar en una mejora sustancial en el tiempo de inspección por placa y en la detección de fallas.

Capítulo 2

Arquitectura

En este capítulo describimos las técnicas de adquisición y el *pipeline* de procesamiento previo a los algoritmos de detección de fallas, que son el foco central de este trabajo. Un sistema típico de AOI tiene una arquitectura general dividida en tres grandes fases: la adquisición, el procesamiento y la clasificación [4] [43], como puede verse en la figura 2.1.

Figura 2.1: Arquitectura típica de AOI

Para este trabajo, proponemos una arquitectura del mismo tipo, basada en el paradigma de un PCBA de referencia, o *golden board*, como puede verse en la figura 2.2. Los distintos bloques se detallan en las siguientes secciones.

El sistema tiene dos modos de funcionamiento principales: **configuración** e **inspección**. El modo de **configuración** es aquel en el que se da de alta un nuevo modelo de PCBA a inspeccionar. Para ello, las entradas necesarias son una imagen de un PCBA de referencia, los archivos de diseño Gerber y el *Bill of Materials* (BOM). Estos archivos son salidas habituales del proceso de diseño que describiremos más adelante. Una vez configurado, el sistema funciona en modo **inspección**, que es el más habitual. En este caso, la única entrada es la imagen de *test*, que se adquiere desde una cámara o desde archivo. Tras realizar un *histogram matching* para compensar posibles diferencias en el nivel de iluminación entre la imagen de *test* y la de referencia, ambas reciben el mismo procesamiento: registración y segmentación. Luego se aplica una clasificación comparativa. Finalmente, se

genera un reporte al usuario con las fallas detectadas. Será el usuario quien actúe y tome las decisiones finales.

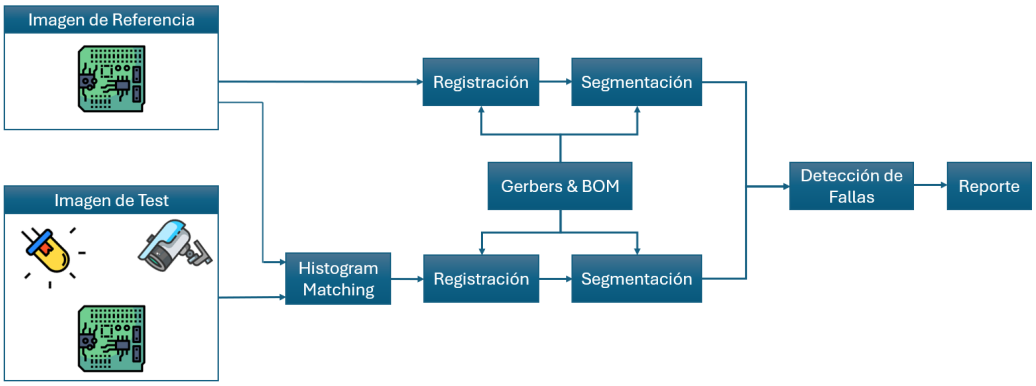


Figura 2.2: Arquitectura propuesta para AOI[1].

En secciones anteriores hicimos énfasis en que algunos sistemas de AOI utilizan tanta información *a priori* que socava su utilidad práctica. Por ejemplo, como vimos en 1.3.2, en [50] se pide al usuario que segmente manualmente los encapsulados de los componentes, lo que es extremadamente costoso en tiempo. Sin embargo, la tarea de diseño de PCBAs tiene como producto varios insumos que podemos aprovechar en el sistema de AOI sin incurrir en un costo extra, pues ya se utilizan para otros propósitos y no requieren intervención alguna del usuario. Algunos de ellos son los Gerbers y el BOM.

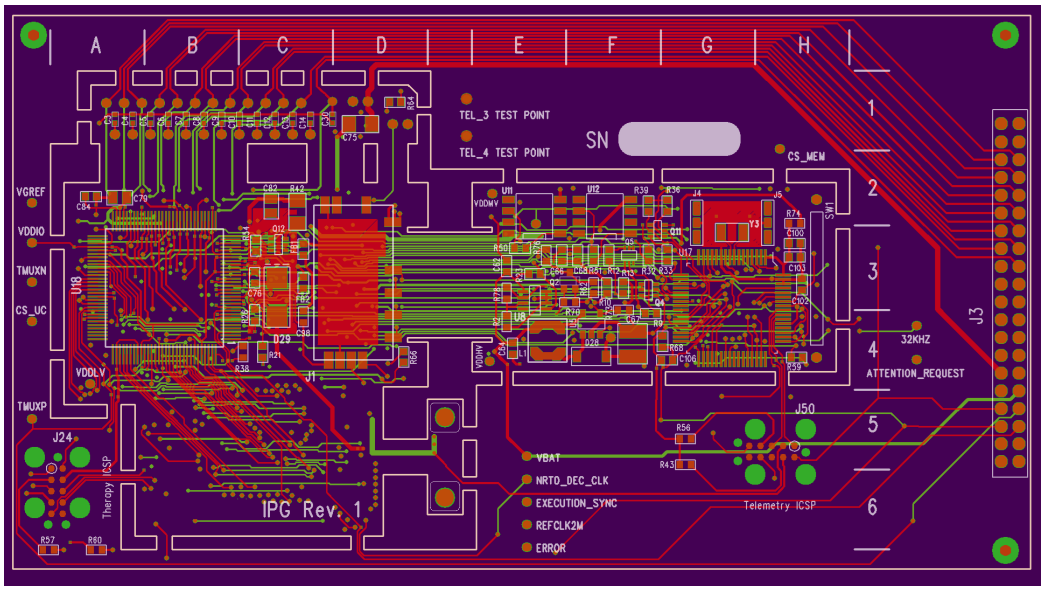


Figura 2.3: Renderización de archivos Gerber mostrando distintos elementos: *pads*, vías, pistas, *silkscreen*, etc. Se muestra solo la capa *top* y una capa intermedia de ruteo.

[1]Íconos tomados de flaticon.com

Los archivos Gerber son el estándar de la industria para la especificación del *layout* de un PCB (ver figura 2.3). Son archivos de texto que detallan las ubicaciones de pistas, *pads*, *silkscreen* y demás datos esenciales para la fabricación del PCB. Internamente, se representan mediante polígonos con lados y vértices, careciendo de *metadata* como el nombre o tipo de componente. Generalmente, se generan varios archivos que se diferencian por nombre o extensión, uno para cada función. Cualquier *software* de CAD capaz de crear esquemáticos y *layouts* de circuitos puede generar los Gerbers.

A su vez, es común crear un BOM. Estos son típicamente archivos de texto tipo CSV o con columnas de tamaño fijo que listan cada componente junto con información sobre su tipo, encapsulado, ubicación, orientación y cantidad. Esto facilita la gestión de compras de componentes y la configuración de la *pick-and-place*.

Al combinar ambos conjuntos de datos, se logra proporcionar al *software* de AOI una gran cantidad de información sobre el diseño sin agregar un requerimiento complejo o costoso al usuario, por lo que entendemos que los beneficios superan con creces a las desventajas.

2.1. Adquisición

Para la adquisición diseñamos, con la ayuda del departamento de Diseño Mecánico de Integer Montevideo, un sistema tipo caja blanca que puede verse en la figura 2.4. Consiste en una caja de acrílico recubierta internamente con papel blanco mate y una tira LED de 12V como fuente de iluminación. En la parte superior, cuenta con una abertura para la adquisición de la imagen y en la parte inferior un soporte intercambiable, fabricado en impresora 3D, con la forma exacta del PCBA a inspeccionar. Además, la cámara se encuentra montada sobre un brazo fijado a la mesa. Este diseño permite mantener una distancia y orientación fija entre el PCBA y la cámara. También provee un nivel de iluminación consistente que minimiza las sombras y da un buen contraste.

En una instancia final de diseño de un sistema de AOI, es decir, en el despliegue y uso en la línea de producción real, el sistema debería emplear una cámara de AOI profesional equipada con una lente telecéntrica, la cual elimina el paralaje. Este tipo de sistema es preferible porque ofrece una integración de software más sencilla y una longitud focal fija, lo que simplifica enormemente la calibración y el uso. Sin embargo, son sistemas de costos elevados, especialmente cuando los sensores son de alta resolución, además de que carecen de la flexibilidad necesaria para experimentar con distintos parámetros, como la longitud focal. Por estos motivos, es usual trabajar con equipos ajustables en las fases de prototipado. Para este trabajo utilizamos una cámara DSLR *off-the-shelf* Nikon D3500 que ya estaba disponible y es controlable por software. Este equipo tiene un sensor de 6000×4000 pixeles, que a la distancia de trabajo y longitud focal seleccionada de 55mm, da una resolución de aproximadamente 130 px/mm, por lo que el componente más pequeño (encapsulado 0603) tendrá una resolución de 80×40 pixeles. Una cámara de AOI profesional con resolución similar implicaría costos muy elevados.

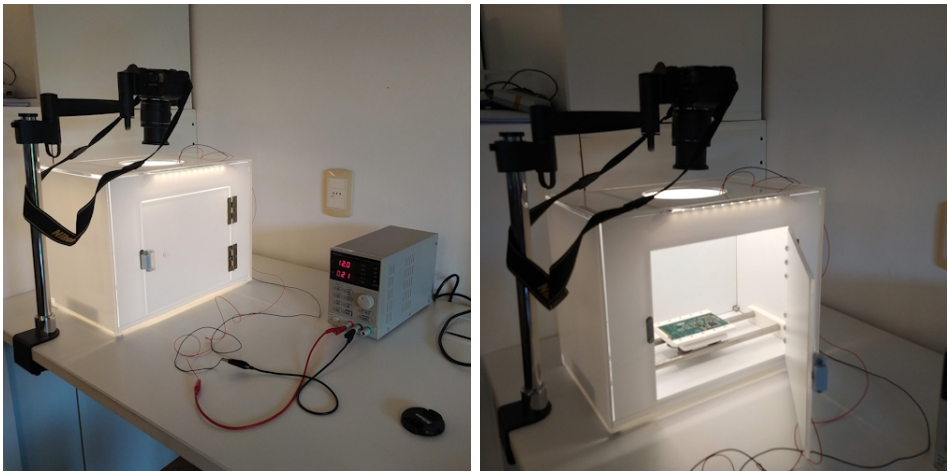


Figura 2.4: Sistema de adquisición utilizado en DS-PCBA y DS-ENCAPSULADOS.

La desventaja más evidente que hallamos de trabajar con este equipo es que la longitud focal variable, si bien nos da flexibilidad para experimentar, complica significativamente la calibración, ya que cualquier ajuste en la lente puede descalibrarla. A su vez, la escasa profundidad de campo a distancias cortas con lentes no especializadas dificulta aún más la tarea de calibración, ya que al cambiar la orientación del patrón de calibración este queda rápidamente desenfocado. Debido a estas complicaciones, los *datasets* presentados en este trabajo están tomados sin calibración de la cámara. Aún así, entendemos que el sistema de adquisición es suficientemente bueno para la tarea a resolver, como se verá más adelante.

2.2. Registración

Esta sección detalla la implementación del bloque de registración que aparece en la figura 2.2. Esto consiste en llevar la imagen de entrada a un sistema de referencia conocido, que nos permita ubicar los componentes por coordenadas. En el caso más general, una cámara ideal se puede modelar con una transformación proyectiva. Por lo tanto, el objetivo de la registración es, partiendo de una imagen con una placa en una posición desconocida, hallar la transformación que la lleve a una imagen donde las coordenadas de los componentes sean fijas y conocidas. Para ello implementamos el procesamiento que se muestra en la figura 2.5. En la imagen, las flechas simples indican que el paso se ejecuta una vez, mientras que las flechas gruesas indican que los pasos se aplican a un conjunto de datos iterativamente.

Debido a las condiciones controladas y favorables del entorno, como la iluminación y escala controladas, junto con la información disponible en los Gerbers y el BOM, las máquinas de AOI y *pick-and-place* suelen resolver los problemas de registración y segmentación utilizando técnicas clásicas de visión artificial, ya que estas son más simples y ofrecen buenos resultados en este contexto. En particular, se usan *fiducials*, que son marcadores circulares ubicados deliberadamente en el PCB para que el sistema de visión artificial pueda encontrar la placa en la imagen.

Como la disponibilidad de este recurso está prácticamente garantizada (de la misma manera que con los Gerbers y el BOM) y sabemos que es suficiente, elegimos el mismo enfoque clásico de visión artificial en este trabajo para estos bloques.

La secuencia de pasos busca la forma global de la placa haciendo Otsu *thresholding* y ajustando con un rectángulo rotado. Luego, para cada esquina del rectángulo, se basa en el Gerber y el BOM para determinar si hay un *fiducial* o no en su proximidad. Si lo hay, busca su coordenada haciendo una transformada de Hough para detectar círculos. Si no lo hay, busca la esquina de la placa usando el detector de esquinas de Harris. Una vez encontrados estos cuatro puntos en la imagen, podemos calcular una transformación proyectiva que los mapea a las coordenadas de nuestro sistema de referencia, extraídas del Gerber. Finalmente, aplicamos esta transformación a la imagen RGB completa.

Si bien tres puntos deberían ser suficientes para modelar traslación, rotación y escala con una transformación afín, hallamos que hay errores asociados a la perspectiva debido a que el eje de la cámara no es perfectamente perpendicular al plano del PCB. Esto surge tanto de que el montaje de la cámara como el del PCB tienen pequeñas desviaciones de la horizontal. Incluir un cuarto punto para corregir deformaciones de perspectiva mejora considerablemente la registración en algunos casos. Para los PCBAs de este trabajo utilizaremos tres *fiducials* más una esquina de la placa para obtener un total de cuatro puntos. En la sección A.1 pueden encontrarse más detalles y ejemplos de la secuencia sobre una placa real.

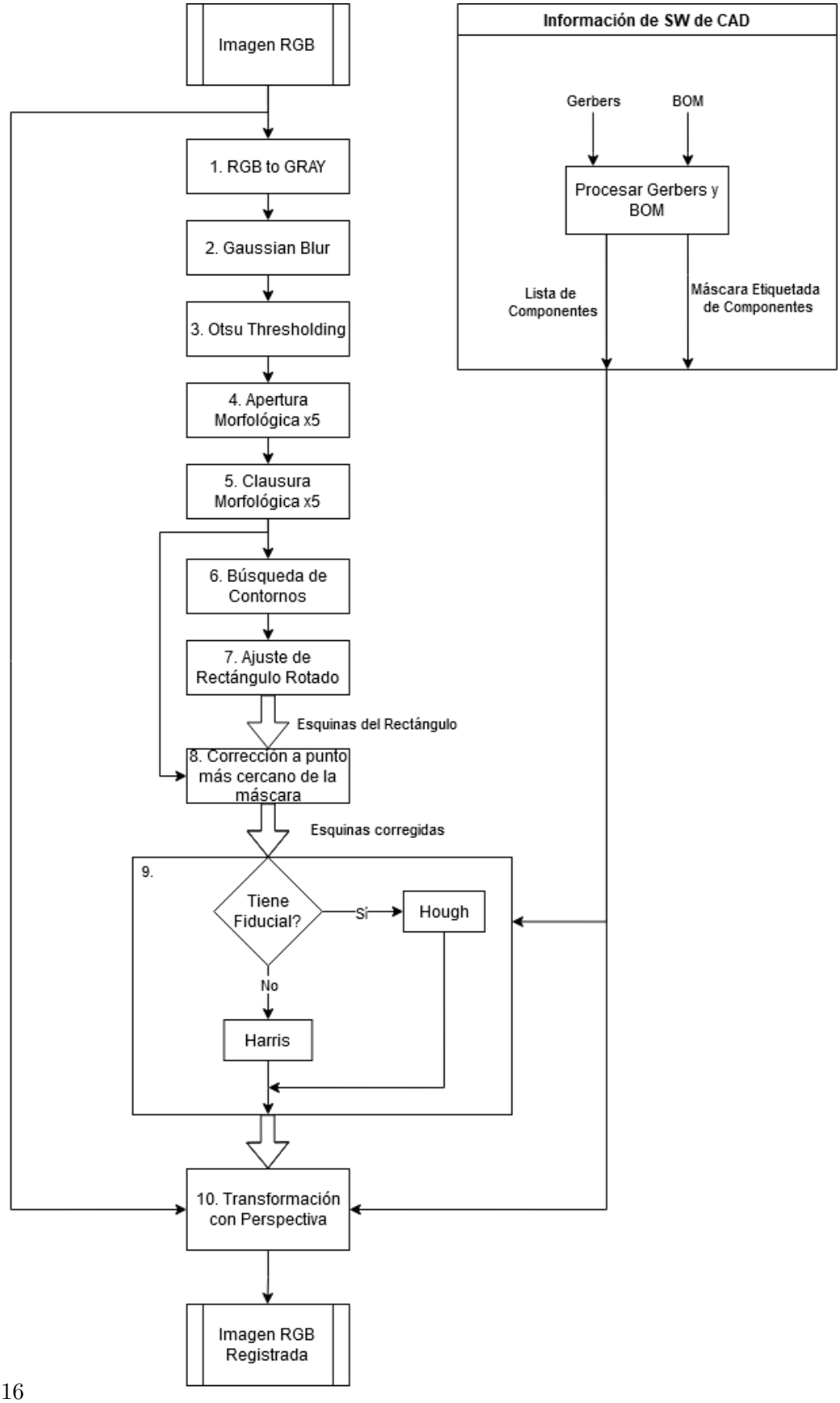


Figura 2.5: *Pipeline* de registración

2.3. Segmentación

Esta sección detalla la implementación del bloque de segmentación que aparece en la figura 2.2. Esto consiste en asignar una etiqueta a cada pixel que identifique a qué componente pertenece. A su vez, resulta conveniente tener el mapeo inverso, es decir, dado un componente poder obtener los pixeles que le corresponden. Para ello, el archivo Gerber y el BOM son herramientas muy valiosas.

El Gerber es una colección de polígonos que codifica la forma y ubicación de los componentes. Sin embargo, no facilita una manera de distinguir un polígono de otro o de saber a qué componente corresponde. Para inferirlo, correlacionamos las posiciones declaradas en el BOM con los centros de masa de los contornos hallados en el Gerber y etiquetamos la máscara, asumiendo que no hay componentes superpuestos. El resultado puede verse en la figura 2.6.

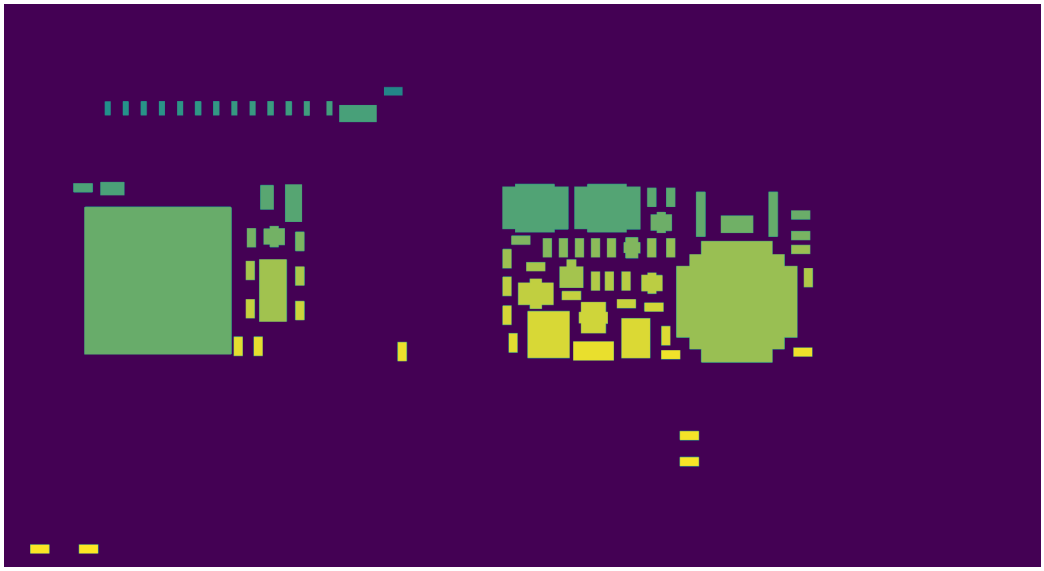


Figura 2.6: Máscara etiquetada del Gerber. La región donde se encuentra cada componente está identificada por una etiqueta distinta.

Además, podemos asociar a cada etiqueta el resto de los atributos del BOM: posición, orientación, nombre, tipo de componente, encapsulado y valor.

Con la imagen registrada y la máscara etiquetada, la tarea de segmentación de componentes queda resuelta: dado un componente cualquiera, podemos usar su etiqueta para hallar su máscara y obtener su imagen. Además, contamos con información del esquemático, como el tipo de componente y el encapsulado, que serán de mucha utilidad para la clasificación. La figura 2.7 muestra el resultado de este proceso.

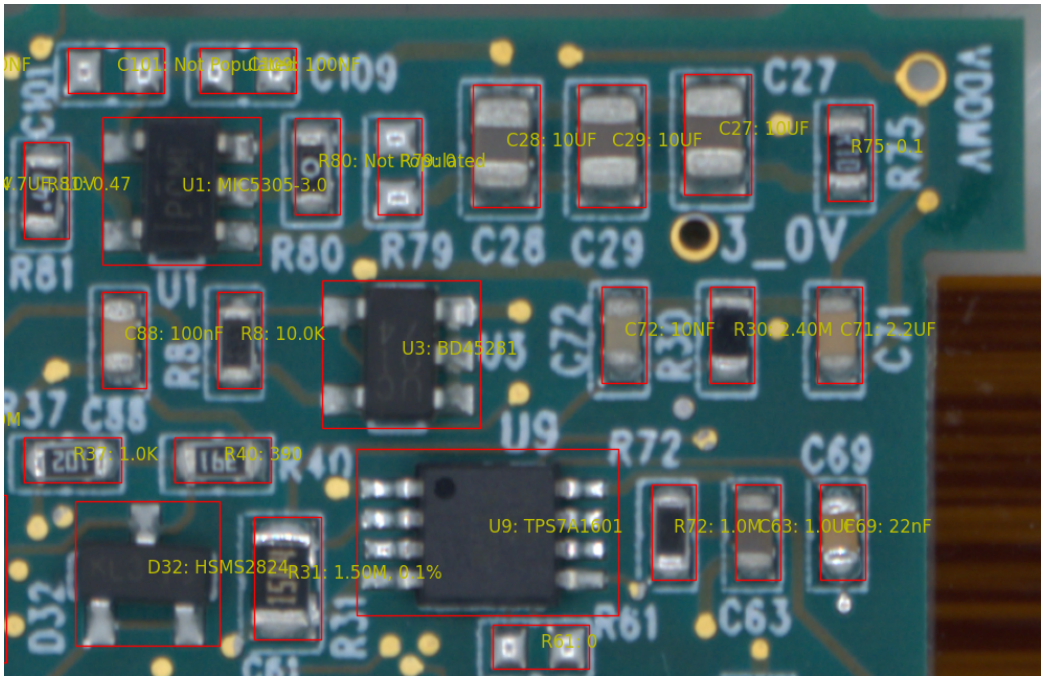


Figura 2.7: Detalle del resultado de la segmentación de componentes para una porción del PCBA. Pueden verse las ubicaciones tomadas del Gerber en rojo y textos en amarillo con el nombre del componente y su valor, provenientes del BOM.

2.4. Detección de Fallas

Esta sección trata brevemente de las posibles arquitecturas del bloque de clasificación que aparece en la figura 2.2. El resto de este trabajo se centra sobre la evaluación de distintas técnicas para esta implementación, por lo que no daremos una discusión en profundidad aún. En particular, consideramos relevante resaltar la diferencia de estrategia entre un clasificador global y uno que trabaje a nivel de componente.

Muchos algoritmos de detección de anomalías realizan la tarea de clasificación computando sobre la imagen completa y marcan automáticamente las regiones detectadas (por ejemplo, como un mapa de calor transparente sobre la imagen, que representa una probabilidad de fallo). Esto puede presentar varias limitaciones. En primer lugar, el tiempo de procesamiento de una imagen de alta resolución puede hacerse bastante extenso o inviable para algunos modelos de aprendizaje profundo. Además, entrenar un algoritmo de aprendizaje profundo de esta manera condiciona las anomalías a la estadística de un PCBA en particular, lo que da un modelo poco flexible y que hay que re-entrenar por cada tipo de placa a inspeccionar.

En cambio, centrarse en la clasificación a nivel de componente ofrece ventajas significativas. No solo permite un modelo más flexible y reutilizable entre diferentes PCBAs, sino que también mejora considerablemente el tiempo de procesamiento al evitar inspeccionar partes no relevantes de la placa. Además, abre la posibilidad de utilizar *datasets* públicos, pues los componentes son bastante universales. Por

ello, optamos por este camino para el diseño.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 3

Métodos y Materiales

En este capítulo damos una descripción de las bases de datos que utilizaremos en capítulos posteriores. Hay algunas de acceso libre así como otras que desarrollamos para este trabajo. Por otro lado, presentamos las métricas de evaluación que utilizaremos para medir el desempeño en los capítulos siguientes.

3.1. *Datasets* Públicos

Existen pocas bases de datos públicas de componentes electrónicos. Las más populares son FICS-PCB [33], PCB-Metal [34] y PCB DSLR [41]. Son bastante similares entre sí en sus condiciones de adquisición y contenido, pues son imágenes adquiridas con cámaras DSLR *off-the-shelf* en condiciones de laboratorio, con anotaciones y máscaras para ubicar el PCBA y los componentes.

3.1.1. PCB-DSLR

PCB-DSLR tiene, en total, 9.313 instancias de componentes. Sin embargo, no incluye a los componentes más pequeños como resistencias o condensadores por limitaciones de resolución; solo se anotan los integrados. Además, son placas adquiridas en una planta de reciclado, por lo que algunas muestran contaminación y suciedad, que no refleja lo que se ve en una línea de producción.

3.1.2. PCB-Metal

PCB-Metal tiene, en total, 12.231 instancias de componentes. Incluye anotaciones para más tipos de componentes: condensadores, integrados, resistencias e inductores.

3.1.3. FICS-PCB

De las opciones disponibles, FICS-PCB resulta ser el más atractivo por la cantidad de imágenes y su resolución superior, por lo que será el *dataset* público que usaremos en este trabajo.

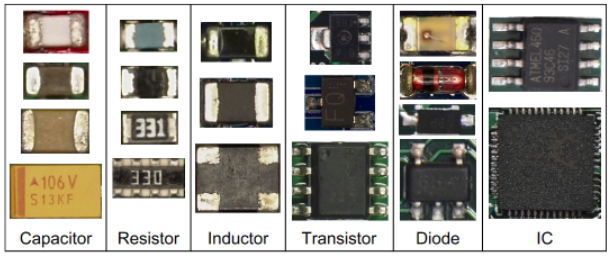


Figura 3.1: Componentes de FICS-PCB, tomado de [33], agrupados por sus clases.

Componente	Cantidad original	Cantidad *Train*	Cantidad *Test*
Capacitors	3453	10920	723
Resistors	3348	10604	697
ICs	542	1708	115
Diodes	195	656	31
Inductors	182	588	35
Transistors	165	552	27
Transducers*	5	-	-
Fuses*	6	-	-
Transformers*	1	-	-

Tabla 3.1: Clases y cantidades en FICS-PCB (* son descartadas). Particionado en *Train* y *Test* aleatoriamente, con aumentación con rotaciones en *Train*.

FICS-PCB tiene, en total, 77.347 instancias de componentes distribuidos en 9 categorías (ver figura 3.1). Además, las anotaciones son más ricas que para los casos anteriores, pues incluyen a los *markings*. Sin embargo, gran parte del *dataset* es redundante porque se toman imágenes con varias cámaras: DSLR y microscopio en varios aumentos. En la tabla 3.1 pueden verse las categorías y las cantidades de imágenes distintas que hay para cada una. Algunos componentes tienen tan pocas instancias que no vale la pena incluirlos, puesto que es poco probable que una red logre aprender sin más datos y estaríamos introduciendo un desbalance demasiado grande al *dataset*. Por otro lado, como vimos en la sección 2, esperamos que el sistema de AOI funcione con una imagen completa de un PCBA tomada con una cámara DSLR, por lo que las imágenes de microscopio no son representativas de lo que se vería en el sistema real, tanto por la iluminación como por la resolución. Por estos motivos, para este trabajo usamos solo las imágenes tomadas con DSLR de las categorías con instancias mayores a 100 componentes, es decir, descartamos las siguientes clases: *fuses*, *transformers* y *transducers*. En total, hay unas 7.885 imágenes distintas. Separamos aleatoriamente los componentes en conjuntos de *train* y *test* con una razón de 80 % a 20 % y del conjunto de *train* reservamos un 20 % aleatoriamente para validación. Realizamos aumentación del conjunto de *train* aplicando rotaciones de 90°, 180° y 270°

Es claro que el conjunto es bastante desbalanceado, pero la resolución de las

imágenes es relativamente buena y la cantidad es suficiente para poder entrenar algunos algoritmos de aprendizaje profundo.

3.2. *Datasets* Propuestos

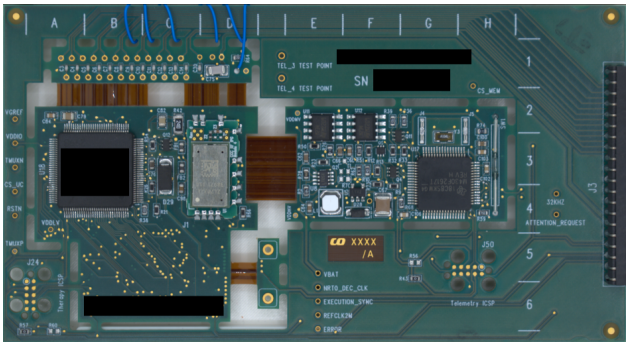
En el transcurso de este trabajo hallamos que los *datasets* públicos no eran suficientes para algunas de las aplicaciones que queríamos investigar, por lo que generamos nuevos a partir de PCBAs provistos por Integer Montevideo. Hay dos tipos de *datasets* presentados en esta sección: en primer lugar, uno de PCBAs con errores etiquetados y, en segundo lugar, de componentes etiquetados por encapsulado. Si bien los PCBAs utilizados para generar ambos tipos de *datasets* son los mismos, la información disponible en cada uno es diferente y permiten abordar distintos problemas.

3.2.1. *Dataset* de PCBAs con Fallas Etiquetadas: DS-PCBA

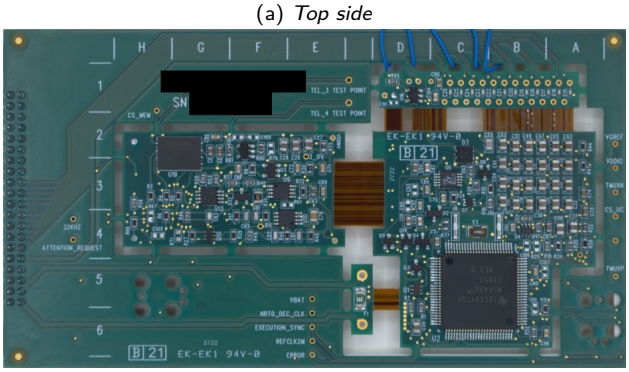
Para evaluar algoritmos de detección de fallas en un escenario real, construimos una base de datos de PCBAs con fallas tomadas de la línea de producción que llamaremos DS-PCBA. La mayoría de las fallas no lo son realmente, sino que son componentes alternativos válidos. A los efectos de una inspección visual, ya que se ven completamente diferentes, el sistema debería considerarlos como componentes distintos, por lo que es equivalente a un error de "componente incorrecto"(E.3), es decir, que se encuentra un componente distinto al esperado[1].

Todas las placas son del mismo modelo de PCBA, que llamaremos modelo A, para distinguirlo de otro modelo que mencionaremos más adelante. Fueron inspeccionados por inspectores humanos y etiquetadas manualmente. Hay un total de 49 placas, con dos lados cada una que suman 235 componentes. En total, hay 85 fallas presentes, 79 del tipo "componente incorrecto" (E.3), 5 del tipo "error de polaridad" (E.5) y 1 del tipo "error de posicionamiento" (E.4). La adquisición está descripta en más detalle en la sección 2.1. En la figura 3.2 pueden verse ejemplos de un PCBA de la base de datos y en la figura 3.3 puede verse un ejemplo de un error de populado.

[1]La existencia de componentes alternativos válidos puede ser resuelta en más alto nivel con varios PCBAs de "referencias" posibles, no es necesario que el modelo conozca esta información.



(a) *Top side*



(b) *Bottom side*

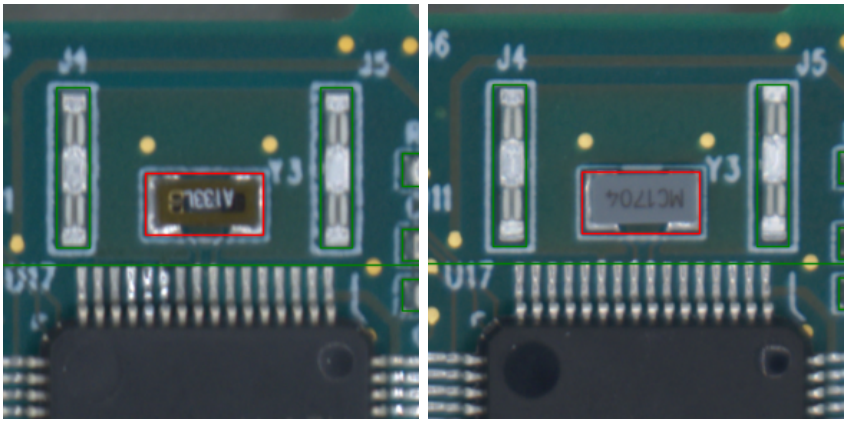
Figura 3.2: Ejemplo de PCBA de la base de datos. Zonas enmascaradas en negro por confidencialidad.

3.2.2. *Dataset* de Componentes Etiquetados por Encapsulado: DS-ENCAPSULADOS

Observamos que las clases presentes en FICS-PCB (sección 3.1.3) son demasiado generales para algunas aplicaciones. En particular, la variación intra-clase de clases como ICs es tan grande que dificulta una tarea de inspección donde estamos intentando detectar, entre otros problemas, un componente incorrecto. Por ejemplo, en la figura 3.1 pueden verse ICs, diodos y transistores completamente diferentes en su encapsulado que, en una inspección visual, deberían ser marcados como componentes distintos.

Para una tarea de comparación visual, la etiqueta que necesitamos debe agrupar componentes por similitud visual, no funcional, porque es lo que podemos esperar que el sistema logre predecir razonablemente. Esto se aproxima bastante bien usando el encapsulado del componente. Etiquetar encapsulados manualmente en FICS-PCB sería demasiado trabajoso, por lo que decidimos tomar el *dataset* de PCBAs con fallas de la sección anterior y etiquetar cada componente con su encapsulado automáticamente, tomando la información del esquemático como fue descripto en la sección 2.3.

Contamos con dos modelos distintos de PCBA para generar este *dataset*, que llamaremos modelo A y modelo B. El modelo A es el que ya fue utilizado en DS-



(a) Referencia (b) *Test*

Figura 3.3: Ejemplo de error de componente (E.3) en Y3.

PCBA. Para cada modelo de PCBA, generaremos una partición de este *dataset*, con el objetivo de poder evaluar el desempeño de los algoritmos estudiados al cambiar de dominio.

Modelo A

Tomamos los PCBAs de DS-PCBA y excluimos los componentes etiquetados como fallas y algunos que, por su ubicación en el PCB, presentan mucha distorsión en la imagen o no tienen buena registración. Además, particionamos los componentes en dos conjuntos que llamaremos *background* y *evaluation*, siguiendo la notación presente en [27] y [25]. El objetivo es excluir completamente algunas clases del proceso de entrenamiento y *test* para poder medir el rendimiento del sistema en clases desconocidas. El conjunto de componentes *background* se particiona de la manera tradicional en *train*, *validation* y *test* para entrenar y medir el rendimiento en instancias nuevas de clases conocidas. Luego, se prueba el modelo en el conjunto de *evaluation*, donde hay clases completamente nuevas para el sistema.

En las tablas 3.2 y 3.3 se encuentran las clases particionadas en *background* y *evaluation* respectivamente, con la cantidad de imágenes de cada una. En la figura 3.4 pueden verse algunos ejemplos. Hay una variedad grande tanto de resistencias, condensadores, inductores, diodos e integrados. A su vez, se incluyen componentes "no populados", es decir, *pads* sin componentes, lo que permite modelar las fallas de tipo (E.1) y (E.2) de la misma manera que (E.3). Las resistencias no están separadas por valor ni orientadas en ninguna dirección particular, de la misma manera que los ICs no están separados por su función o tipo específico. Es decir, solo tenemos en cuenta el encapsulado e ignoramos el componente y sus *markings*[2]. En total, hay 8812 imágenes distribuidas en 25 clases en *background* y 413 en 4

[2]Hay una excepción a esto y es la clase *R_0603_no_marking*. Las resitencias de encapsulado 0603 sin *marking* van en una clase aparte.

Clase	Cantidad	Clase	Cantidad
10MSOP	47	R_0603_NO_MARKING	1566
8MSOP	147	R_0805	49
8SOP150	95	R_1206	49
BSS138DW	245	SOD-123	49
C_0603	1957	SOT-323	196
C_0805	1470	SOT23	147
C_1206	49	SOT23-5	195
C_1210	49	SOT23-8	49
DFN3030-4	49	TQFP100	49
DO-214AC	49	TQFP120	49
LPS3314	49	TQFP64	49
NOT_POPULATED	539	WSON8X6MM	49
R_0603	1571	Total	8812

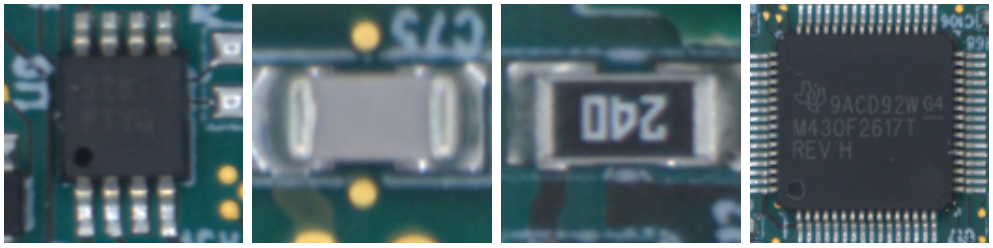
Tabla 3.2: Clases y cantidades en *dataset* de encapsulados, modelo A, partición *background*

Clase	Cantidad
32KHZ_SMD	70
CLIP_SHIELDING	196
F0603	98
F1206	49
Total	413

Tabla 3.3: Clases y cantidades en *dataset* de encapsulados, modelo A, partición *evaluation*

clases en *evaluation*. Notamos que para la clase 32KHZ_SMD hay 30 fallas de tipo "componente incorrecto" (E.3) en el DS-PCBA, por lo que si entrenamos un algoritmo con la partición de *background* de DS-ENCAPSULADOS tendremos fallas para detectar en clases desconocidas durante el entrenamiento.

En las secciones que describen el trabajo con este *dataset* empleamos distintas particiones de *train*, *validation* y *test*, así como técnicas de aumentación y subclases. El detalle de cada una puede encontrarse en las secciones relevantes.



(a) IC 8MSOP (b) Capacitor 1206 (c) Resistencia 1206 (d) IC TQFP64

Figura 3.4: Ejemplos de encapsulados

Modelo B

De manera análoga a la sección anterior, construimos un *dataset* de componentes pero tomando otro modelo de PCBA que llamaremos modelo B. Este PCBA comparte algunas clases de componentes con el modelo A, mientras que otras son completamente distintas. Si bien la cantidad de instancias es mucho menor, será de utilidad para estimar la pérdida de rendimiento en un cambio de dominio. Tomamos la partición de *background* y *evaluation* de manera tal que las etiquetas presentes en *background* para el modelo B estén incluidas en el conjunto *background* del modelo A, es decir, que las clases comunes entre A y B quedan en *background*.

Background		*Evaluation*	
Clase	Cantidad	Clase	Cantidad
BSS138DW	182	32KHZ_SMD	14
C_0603	322	BSS84DW	14
C_0805	28	DCK	28
C_1206	126	DCK__SC70-6_	28
C_1210	42	DT1608C	14
R_0603	490	F1206	14
R_0603_no_marking	168	R_ARRAY_4_0804	84
SOT23	139	SOT-563	42
SOT23-5	42		
TQFP64	28		
Total	1567	Total	238

Tabla 3.4: Clases y cantidades en *dataset* de encapsulados, modelo B

3.3. Métricas de Evaluación

En esta sección describiremos las métricas utilizadas para evaluar el desempeño de los algoritmos de clasificación en los siguientes capítulos.

Para los problemas de clasificación binarios, usaremos la notación habitual de VP (Verdadero Positivo), VN(Verdadero Negativo), FP (Falso Positivo) y FN (Flaso Negativo). La semántica de "positivo" y "negativo", es decir, cómo se relacionan con los eventos a detectar o con los valores de la etiqueta binaria 0 o 1, es generalmente arbitrario y suele asignarse en cada problema acorde al contexto. En este trabajo usaremos la convención de detección de anomalías, donde la clase "positiva" es la de una falla, mientras que la clase "negativa" es la ausencia de falla.

Por otro lado, para los problemas de clasificación multiclase, resulta útil generalizar los conceptos de VP_i, VN_i, FP_i y FN_i a nivel de cada clase i. En estos casos, tomaremos la clase "positiva" como la clase objetivo que estemos estudiando en ese momento (i) y la clase "negativa" como la unión de todas las demás. De esta manera podemos aplicar algunas métricas pensadas para problemas binarios a problemas multiclase.

3.3.1. *Accuracy*

La exactitud, o *accuracy*, es la proporción de clasificaciones correctas. En un problema de clasificación binaria, esto se reduce a

$$A = \frac{VP + VN}{VP + VN + FP + FN}$$

Para un problema multiclase, podemos definir el *accuracy* de la clase i

$$A_i = \frac{VP_i + VN_i}{VP_i + VN_i + FP_i + FN_i}$$

Así como el *accuracy* global

$$A = \frac{\sum_i (VP_i + VN_i)}{\sum_i (VP_i + VN_i + FP_i + FN_i)}$$

El *accuracy* global es una medida de desempeño general sencilla, aunque es muy incompleta. Es particularmente susceptible a conjuntos de datos desbalanceados, donde puede indicar buenos resultados si la clase más prevalente tiene un buen desempeño, por más que en las clases menos frecuentes el sistema funcione muy mal. Para estos casos suelen ser más informativas las métricas de *precision* y *recall* por clase.

3.3.2. *Precision*

La precisión es la proporción de verdaderos positivos en el total de los clasificados como positivos, es decir,

$$P_i = \frac{VP_i}{VP_i + FP_i}$$

En un problema de detección de anomalías binario, la precisión es la proporción de fallas verdaderas entre todas las fallas detectadas. En un problema de

clasificación multiclase, la precisión asociada a una clase objetivo es la proporción de elementos que verdaderamente pertenecían a la clase objetivo entre todos los que fueron clasificados como de la clase objetivo.

3.3.3. *Recall*

La exhaustividad, o *recall*, es la proporción de verdaderos positivos en el total de los elementos que realmente son positivos, es decir,

$$R_i = \frac{VP_i}{VP_i + FN_i}$$

En un problema de detección de anomalías binario, el *recall* es la proporción de fallas verdaderas que fueron detectadas. En un problema de clasificación multiclase, el *recall* asociado a una clase objetivo es la proporción de elementos de la clase objetivo que fueron clasificados como tales.

3.3.4. FPR

Para un problema de clasificación binario, el *False Positive Rate* (FPR) es la proporción de elementos que fueron clasificados como "positivos" entre todos los que eran "negativos"

$$FPR = \frac{FP}{FP + TN}$$

En el contexto de detección de anomalías, el FPR es la proporción de elementos que el sistema detecta como fallas, cuando no lo son. Esta métrica es importante para estimar la cantidad de tiempo de inspección de un PCBA, pues el inspector humano debe estudiar cada falso positivo para desestimarlo.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 4

Detección de Fallas

En este capítulo presentamos distintos algoritmos y estudiamos sus rendimientos al aplicarlos a la detección de fallas en PCBAs. Todos ellos trabajan a nivel de componente, de manera que puedan ser incluidos en la arquitectura presentada en el capítulo 2. No solo nos centramos en su desempeño sino que también valoramos su costo de entrenamiento y mantenimiento.

En la sección 4.1, tomamos como referencia principal a [23], donde se plantea el uso de *transfer learning* para detectar anomalías en texturas con *datasets* de pocas imágenes, lo que puede ayudar a subsanar la falta de bases de datos públicas. A partir de ellos, desarrollamos un modelo de clasificación de componentes electrónicos entrenado por *transfer learning* y comparamos su desempeño en un *dataset* público así como en uno original de este trabajo.

En la sección 4.2, tomamos como referencia principal a [25], donde se estudia el uso de redes siamesas para hacer *one-shot learning*, es decir, clasificación en clases no conocidas durante el entrenamiento y teniendo una única instancia de cada una, disponible durante la inferencia. Dado que la aplicación de dicho trabajo es para clasificación de caracteres, formulamos un problema análogo para componentes electrónicos y modificamos la arquitectura para poder estudiar el desempeño en la detección de fallas de componentes electrónicos con un *dataset* original.

4.1. Clasificador de Componentes con *Transfer Learning*

La técnica de *transfer learning* consiste en aplicar el conocimiento aprendido para una tarea a otra tarea similar. Es de gran utilidad cuando existen buenos modelos ya entrenados para resolver un problema particular y deseamos obtener un nuevo modelo que intenta resolver otro problema que comparte algunas características con el original, pero no es posible construir un *dataset* suficientemente grande. Los modelos modernos de clasificación de imágenes pueden superar los cientos de millones de parámetros (VGG16 tiene aproximadamente 140 millones [45]) y entrenar con un conjunto reducido de datos lleva rápidamente al *overfitting*. Al aplicar *transfer learning* podemos aprovechar el conocimiento común a ambos problemas y relajar los requerimientos de datos de entrenamiento.

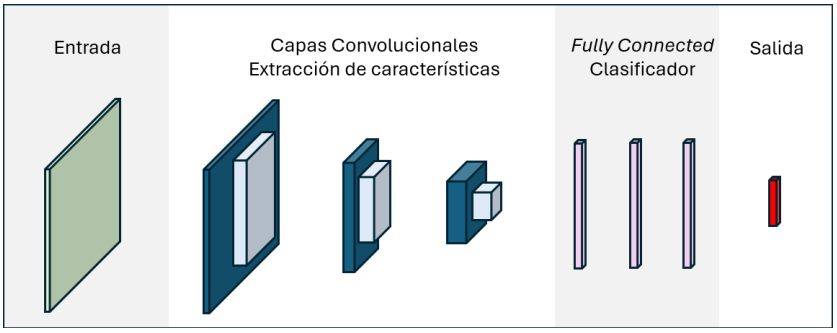


Figura 4.1: Ejemplo de arquitectura de red convolucional

Una red convolucional típica tiene una arquitectura como se muestra en la figura 4.1 [23] [24] [45]. Consiste en una secuencia de filtros convolucionales seguidos por otra sección de capas *fully connected*. Estas redes tienden a aprender *features* de manera jerárquica [28], pues las capas iniciales realizan operaciones de convolución sobre la imagen de entrada, detectando patrones locales como bordes y formas geométricas elementales. A medida que las capas se profundizan, tienden a detectar conceptos más complejos y las características se vuelven más abstractas. Las capas de *max-pooling* favorecen este proceso al bajar progresivamente la resolución de la imagen, agregando invarianza ante traslación. Finalmente, las capas *fully connected* combinan las características de manera global para obtener la clasificación final. Un ejemplo de esto puede verse en la figura 4.2.

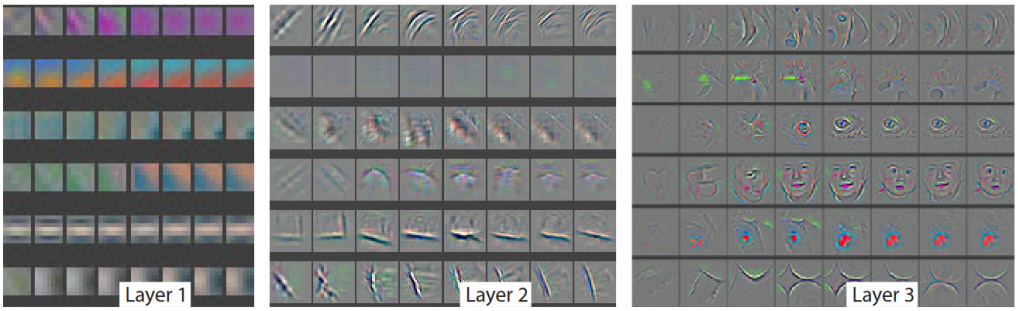


Figura 4.2: Activaciones jerárquicas de capas de una red neuronal. En la capa 1 vemos gradientes y bordes. En la capa 2 vemos estructuras más complejas compuestas de varios bordes, aunque aun no se corresponden a objetos reconocibles. Finalmente en la capa 3 ya logramos reconocer estructuras como caras u ojos. Extraído de [59].

Si bien no hay una frontera bien definida, podemos entender que las capas convolucionales realizan una tarea principalmente de extracción de características y las capas *fully connected* de clasificación. Observando la jerarquía de las capas, es esperable que las de más bajo nivel sean aplicables a otros problemas similares, ya que las formas elementales pueden estar presentes en cualquier *dataset*. El *transfer learning* nos permite **reentrenar las capas más profundas** de clasificación y **reutilizar las capas iniciales** de extracción de características.

La manera más sencilla de implementar el *transfer learning* es tomando un modelo pre-entrenado y conservar la extracción de características, mientras que se descarta el clasificador y se re-entrena en el dominio deseado. Esto se implementa manipulando la inicialización y actualización de los pesos de cada capa durante el entrenamiento. Las capas convolucionales se inicializan con los valores del modelo anterior y quedan congeladas. Por otro lado, las capas *fully connected* se inicializan con pesos aleatorios y se entrenan con los datos nuevos. Esto puede disminuir significativamente la cantidad de parámetros a entrenar y el tiempo de entrenamiento, así como mejorar el rendimiento. Si los dominios son suficientemente similares para que la extracción de características sea útil, es posible que de buenos resultados.

Una vez entrenado el clasificador nuevo, también es posible descongelar las capas convolucionales y continuar el entrenamiento, ajustando los parámetros del modelo completo. Dado que partimos de un punto mas cercano al óptimo local, esta segunda fase puede verse como un *fine tuning* de los parámetros de la red convolucional y, como tal, puede hacerse con un *learning rate* menor.

La metodología del *transfer learning* cobra especial relevancia en el problema clásico de clasificación de imágenes, que ha experimentado profundos avances en la última década. Competencias como ImageNet establecieron un estándar de referencia contra el que medir el rendimiento de distintos algoritmos, donde se demostraron los varios beneficios de las redes convolucionales. Entre ellas, VGG16 [45] logró un gran desempeño y se ubicó como uno de los mejores modelos. Si bien algunas redes han obtenido menores tasas de error, VGG se ha mantenido popular, entre otras cosas, por ser liberada al público.

Figura 4.3: Ejemplos de imágenes de ImageNet

El conjunto de datos de ImageNet para el *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) es un *dataset* de 1,4 millones de imágenes, divididas en 1.000 clases (ver figura 4.3). Son imágenes obtenidas de la web y etiquetadas manualmente, con contenidos variados como objetos cotidianos, personas, animales, etc. Esto no es adecuado para nuestras necesidades, pues el dominio de nuestro problema es mucho más concreto y está completamente ausente de este *dataset*. Sin embargo, FICS-PCB consta de 7.885 imágenes, por lo que la cantidad de imágenes por clase es, en promedio, del mismo orden que en ImageNet (aunque FICS-PCB es más desbalanceado). Esto sugiere que si logramos aprovechar la detección de características general de VGG16 aplicando *transfer learning*, podríamos entrenar un buen clasificador.

En el trabajo original que presenta a VGG [45] se reconoce la viabilidad de

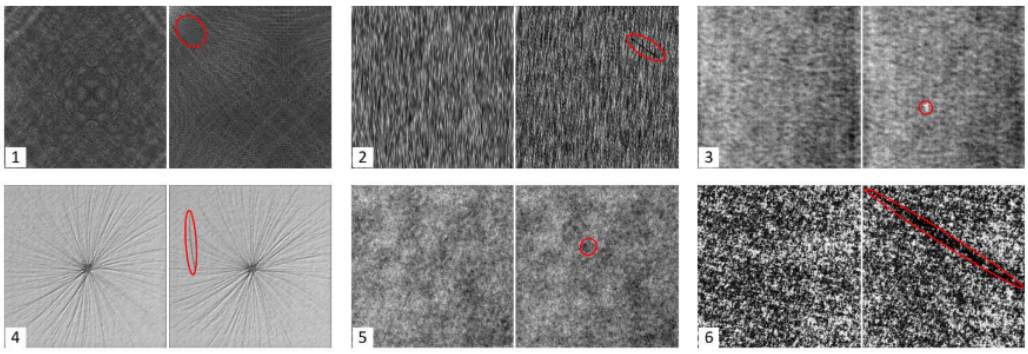


Figura 4.4: Ejemplos de texturas con y sin fallas del *dataset* DAGM. Extraído de [23].

utilizar esta red pre-entrenada para la extracción de características, prescindiendo de las capas *fully connected* y empleando un clasificador SVM, sin utilizar *fine tuning*. Por otro lado, autores como [23] han aplicado efectivamente la técnica de *transfer learning* con VGG16, logrando mejoras de rendimiento en la clasificación. En [23] se entrena un clasificador basado en VGG16 con *transfer learning* para detectar la textura y la presencia de anomalías en el *dataset* de la competición de DAGM de 2007 (ver figura 4.4). Este consta de 10 clases de textura, cada una con 2.000 imágenes sin falla y 30 con fallas, repartido 50 %-50 % entre *train* y *test*. En las secciones siguientes modificamos la arquitectura de VGG16 y exploramos el resultado de entrenar este modelo con *transfer learning* en nuestro *dataset* de componentes electrónicos para implementar un clasificador de componentes, siguiendo el trabajo de [23] como referencia.

4.1.1. Arquitectura Propuesta del Clasificador

Basamos la arquitectura global de la red utilizada en la de VGG16, modificando la última capa *fully connected*, debido a la diferencia de la cantidad total de clases. Entrenaremos el clasificador sobre dos *datasets*: FICS-PCB y DS-ENCAPSULADOS, que tienen 6 y 25 clases respectivamente. Para cada caso, el tamaño de la última capa es ajustado para ser igual al número de clases.

En la figura 4.5 vemos una representación gráfica de la arquitectura de la red resultante, para el caso de FICS-PCB. Los volúmenes representan un tensor de datos en cada punto de la red, mientras que su color codifica cuál fue la última operación que se le aplicó, es decir, de qué bloque es salida. El cambio de arquitectura solo afecta a las dimensiones de las capas *fully connected* del bloque 6 y del bloque de salida. En total, la red tiene aproximadamente 134 millones de parámetros, de los cuales 119 corresponden a las capas *fully connected* y 15 a las capas convolucionales.

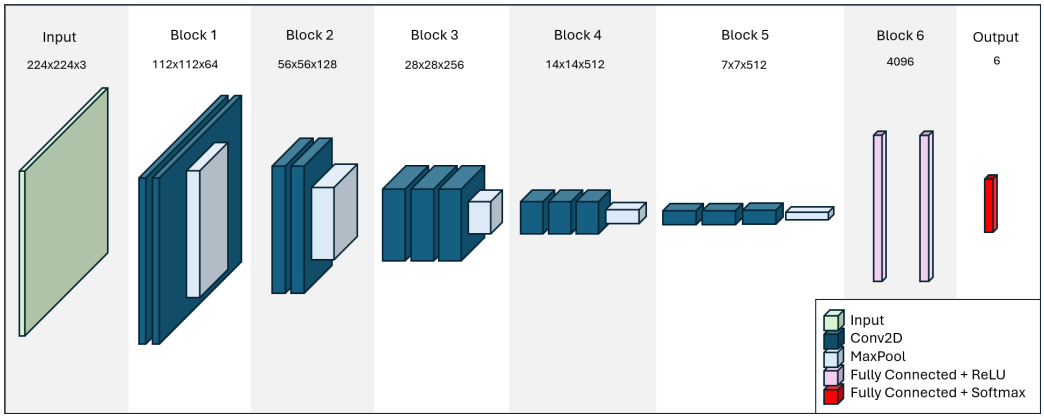


Figura 4.5: Arquitectura del clasificador basado en VGG16

4.1.2. Entrenamiento

El entrenamiento se divide en dos fases secuenciales: una primera instancia que conserva la extracción de características mientras entrena la clasificación desde cero y luego una segunda instancia de *fine tuning* que entrena la red completa.

Durante la primera fase descartamos completamente los pesos de los bloques 6 y de salida al tiempo que conservamos los pesos de los bloques 1 a 5. Usamos parámetros similares a los de [23] y con los que logramos reproducir sus resultados. Estos son: *batch size* de 32, *validation split* de 20 % sobre los datos de *train* del *dataset*, optimizador ADAM con *learning rate* de 10^{-3}, entropía cruzada como función de pérdida y *early stopping* de 5 épocas monitoreando la pérdida en validación, con un máximo de 20 épocas. En esta fase los pesos de los bloques 1 a 5 no se actualizan.

Para la fase de *fine-tuning* los parámetros de entrenamiento son idénticos, con la excepción del *learning rate*, que cambia a uno más pequeño de 10^{-5}, con el fin de prevenir el *overfitting*. En esta fase todos los pesos de la red se actualizan.

4.1.3. *Datasets*

Entrenaremos la red sobre dos conjuntos de datos distintos que fueron presentados en 3: FICS-PCB y DS-ENCAPSULADOS.

FICS-PCB

Para entrenar sobre FICS-PCB separamos aleatoriamente los componentes en conjuntos de *train* y *test* con una razón de 80 % a 20 % y del conjunto de *train* reservamos un 20 % aleatoriamente para validación. Realizamos aumentación del conjunto de *train* aplicando rotaciones de 90°, 180° y 270°. No aplicamos simetrías pues los *markings* en espejo serían incorrectos. Así mismo, entendemos que no sería beneficioso incorporar otros ángulos, ya que no son frecuentes en PCBAs. Además, llevamos todas las imágenes a una resolución de 224×224 pixeles, dado

que es la resolución de entrada de VGG16. Escalamos la dimensión más grande y hacemos *zero-padding* si es necesario para mantener el *aspect ratio*.

DS-ENCAPSULADOS

Para este *dataset*, intentamos que los parámetros no se desvíen del caso de FICS-PCB, pero debemos hacer algunas modificaciones. Trabajamos con la partición de *background* del modelo A del *dataset* DS-ENCAPSULADOS, re-escalando las imágenes a 224×224. Tomamos una partición aleatoria de *train* y *test* de 50 %-50 %. Esta proporción es distinta a FICS-PCB y se debe a que la cantidad de imágenes es mucho menor, por lo que tomar 80 %-20 % dejaría muy pocas instancias de *test* para las clases menos numerosas. Del conjunto de *train* reservamos un 20 % para validación, elegido aleatoriamente. Aumentamos el conjunto de *train* aplicando rotaciones de 90°, 180° y 270°. La cantidad final de imágenes puede verse en la siguiente tabla 4.1.

Clase	*Train*	*Test*	Clase	*Train*	*Test*
10MSOP	80	27	R_0603_NOMARKING	3016	812
8MSOP	296	73	R_0805	88	27
8SOP150	228	38	R_1206	104	23
BSS138DW	520	115	SOD-123	92	26
C_0603	3904	981	SOT-323	336	112
C_0805	2924	739	SOT23	300	72
C_1206	76	30	SOT23-5	388	98
C_1210	108	22	SOT23-8	96	25
DFN3030-4	92	26	TQFP100	76	30
DO-214AC	108	22	TQFP120	92	26
LPS3314	100	24	TQFP64	76	30
NOTPOPULATED	1092	266	WSON8X6MM	80	29
R_0603	3052	808			

Tabla 4.1: Partición de *dataset* de encapsulados con aumentación.

4.1.4. Resultados

Aplicación a FICS-PCB

En esta sección mostramos los resultados de entrenar la red con *transfer learning* y desde cero. Realizamos 10 entrenamientos distintos para cada caso y reportamos los resultados promedio medidos sobre el conjunto de *test*, con la desviación estándar como incertidumbre. En cada experimento tomamos una nueva partición aleatoria del conjunto de *train* y validación.

En la tabla 4.2 puede verse el resultado global del entrenamiento, donde rápidamente se nota la mejora de rendimiento al usar *transfer learning*. Vale la pena notar que, de los 10 entrenamientos sin *transfer learning*, 7 no logran converger y clasifican todas las imágenes a la misma clase (resistencia o condensador, dependiendo el caso). Eso arroja un rendimiento de aproximadamente 0,40, dada la distribución del *dataset*. En la tabla reportamos el rendimiento restringido a los entrenamientos que sí convergen.

	Transfer	*No transfer*
Épocas	15 ± 3	20
Accuracy	$0{,}96 \pm 0{,}01$	$0{,}93 \pm 0{,}01$[1]

Tabla 4.2: Rendimiento global del clasificador sobre el conjunto de *test* de FICS-PCB

Si bien el rendimiento global parece bueno, el *dataset* es bastante desbalanceado, por lo que vale la pena mirar el rendimiento por clase. En la tabla 4.3 incluimos la precisión y *recall* por clase. Puede verse que para las clases de capacitores y resistencias, el rendimiento es muy bueno, mientras que para las demás, si bien hay distintos grados, en general es bastante inferior. Es claro que hay una correlación grande entre la cantidad de instancias y el rendimiento final, por lo que parece razonable asumir que el tamaño del *dataset* no fue suficiente y que, de contar con uno mayor podría mejorar el rendimiento considerablemente.

Clase	Precisión	*Recall*	*# Train*	*# Test*
Capacitors	$0{,}98 \pm 0{,}01$	$0{,}98 \pm 0{,}01$	10920	723
Resistors	$0{,}98 \pm 0{,}01$	$0{,}99 \pm 0{,}01$	10604	697
ICs	$0{,}92 \pm 0{,}02$	$0{,}93 \pm 0{,}02$	1708	115
Diodes	$0{,}73 \pm 0{,}12$	$0{,}75 \pm 0{,}10$	656	31
Inductors	$0{,}85 \pm 0{,}10$	$0{,}59 \pm 0{,}12$	588	35
Transistors	$0{,}61 \pm 0{,}10$	$0{,}65 \pm 0{,}06$	552	27
Weighted Avg	$0{,}96 \pm 0{,}01$	$0{,}96 \pm 0{,}01$		

Tabla 4.3: Rendimiento por clase

[1] Promediado sobre las 3 instancias que convergen

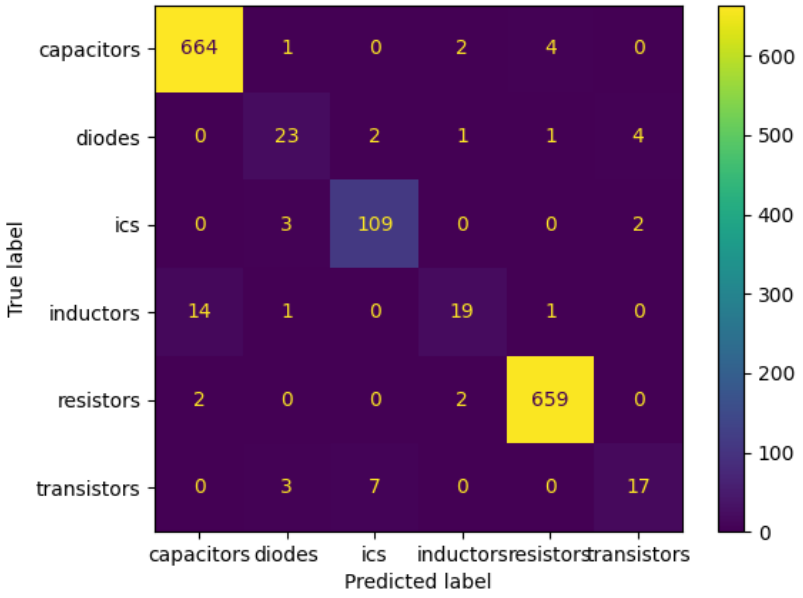
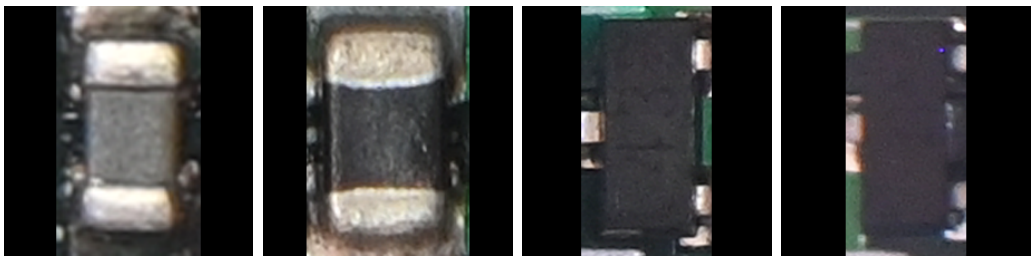


Figura 4.6: Matriz de confusión del clasificador

Por otro lado, al estudiar la matriz de confusión (figura 4.6) vemos que algunos errores de clasificación probablemente surjan de la poca separabilidad de las clases. En la figura 4.7 pueden verse ejemplos de algunos componentes que son virtualmente indistinguibles aun para un humano, debido a que usan el mismo encapsulado. Incluso si pudiéramos aumentar la cantidad de imágenes, hay un problema de fondo en el diseño del *dataset*.



(a) Capacitor (b) Inductor (c) Transistor (d) Diodo

Figura 4.7: Poca separabilidad de FICS-PCB. Incluso para un humano sería imposible distinguir (a) de (b) o (c) de (d), a pesar que llevan etiquetas distintas.

Aplicación a DS-ENCAPSULADOS

De la misma manera que con el clasificador anterior, realizamos 10 entrenamientos distintos y reportamos los resultados promedio medidos sobre el conjunto de test, con la desviación estándar como incertidumbre. En cada experimento tomamos una nueva partición aleatoria del conjunto de *train* y validación.

El clasificador entrena en 26 ± 6 épocas y obtiene un *accuracy* global de $0{,}999 \pm 0{,}001$. En la tabla 4.4 puede verse el rendimiento por clase. Es muy notoria la mejora comparando con el caso de FICS-PCB, incluso en las clases que tienen igual o menor cantidad de imágenes. Sin embargo, es importante recordar que, dado el tamaño limitado del *dataset*, podríamos estar sobreestimando o subestimando el rendimiento del clasificador en las clases con menos muestras, ya que el conjunto de *test* no es suficientemente robusto.

Clase	Precisión	*Recall*	#*Train*	#*Test*
C_0603	$1{,}000 \pm 0{,}000$	$0{,}999 \pm 0{,}001$	3904	981
R_0603_NO_MARKING	$1{,}000 \pm 0{,}000$	$1{,}000 \pm 0{,}001$	3016	812
R_0603	$0{,}999 \pm 0{,}001$	$1{,}000 \pm 0{,}001$	3052	808
C_0805	$0{,}999 \pm 0{,}001$	$1{,}000 \pm 0{,}000$	2924	739
NOT_POPULATED	$1{,}000 \pm 0{,}000$	$0{,}998 \pm 0{,}003$	1092	266
BSS138DW	$0{,}985 \pm 0{,}018$	$1{,}000 \pm 0{,}000$	520	115
SOT-323	$0{,}998 \pm 0{,}005$	$0{,}998 \pm 0{,}005$	336	112
SOT23-5	$1{,}000 \pm 0{,}000$	$0{,}981 \pm 0{,}024$	388	98
8MSOP	$1{,}000 \pm 0{,}000$	$1{,}000 \pm 0{,}000$	296	73
SOT23	$1{,}000 \pm 0{,}000$	$0{,}997 \pm 0{,}006$	300	72
8SOP150	$1{,}000 \pm 0{,}000$	$1{,}000 \pm 0{,}000$	228	38
C_1210	$1{,}000 \pm 0{,}000$	$1{,}000 \pm 0{,}000$	108	22
DO-214AC	$0{,}992 \pm 0{,}025$	$1{,}000 \pm 0{,}000$	108	22
R_1206	$0{,}988 \pm 0{,}035$	$0{,}991 \pm 0{,}017$	104	23
LPS3314	$1{,}000 \pm 0{,}000$	$1{,}000 \pm 0{,}000$	100	24
SOT23-8	$1{,}000 \pm 0{,}000$	$1{,}000 \pm 0{,}000$	96	25
DFN3030-4	$1{,}000 \pm 0{,}000$	$0{,}996 \pm 0{,}012$	92	26
SOD-123	$1{,}000 \pm 0{,}000$	$1{,}000 \pm 0{,}000$	92	26
TQFP120	$1{,}000 \pm 0{,}000$	$1{,}000 \pm 0{,}000$	92	26
R_0805	$0{,}996 \pm 0{,}011$	$0{,}993 \pm 0{,}022$	88	27
WSON8X6MM	$0{,}997 \pm 0{,}010$	$1{,}000 \pm 0{,}000$	80	29
10MSOP	$0{,}996 \pm 0{,}011$	$1{,}000 \pm 0{,}000$	80	27
C_1206	$0{,}997 \pm 0{,}010$	$1{,}000 \pm 0{,}000$	76	30
TQFP100	$1{,}000 \pm 0{,}000$	$1{,}000 \pm 0{,}000$	76	30
TQFP64	$1{,}000 \pm 0{,}000$	$1{,}000 \pm 0{,}000$	76	30

Tabla 4.4: Rendimiento del clasificador sobre el conjunto de *test* de encapsulados

4.1.5. Conclusión

La aplicación de *transfer learning* a FICS-PCB fue claramente beneficiosa y permitió entrenar un clasificador con un *dataset* reducido, que de otra manera no hubiera sido posible. Sin ella, el entrenamiento resultó muy inestable dado que no convergió en la mayor parte de las veces. Incluso en los casos que sí convergió, obtenemos un rendimiento mejor en una menor cantidad de épocas al aplicar *transfer learning*. Sin embargo, como mencionamos en la sección 1.3.2, este tipo de clasificador tiene una aplicación práctica bastante limitada, dado que las clases presentes en FICS-PCB son completamente insuficientes para cubrir la grandísima variedad de encapsulados y componentes que existen en el mundo real. Puede tener sentido como detector de componente incorrecto dentro de un sistema más complejo, aunque sería deseable una clase *not-populated* para que fuera realmente útil en esta aplicación. El principal beneficio de este clasificador es que, si contamos con información de diseño como *gerbers* y BOM, podemos tener una inspección general sin imagen de referencia, basada enteramente en *datasets* públicos.

Por otro lado, la aplicación a DS-ENCAPSULADOS arrojó resultados notoriamente superiores. La marcada mejora de rendimiento parece indicar que el cambio de paradigma en el *dataset* es un paso en la dirección correcta. El resultado es suficientemente bueno como para usar en un contexto real gracias a su gran desempeño y entrenamiento sencillo de pocas épocas, que requiere una relativamente baja cantidad de imágenes. No obstante, **el clasificador obtenido no es general pues está restringido a operar en el dominio de un modelo particular de PCBA** y, por lo tanto, no hay una manera sencilla de actualizar el clasificador sin reentrenar. Esto podría suceder si, por ejemplo, queremos considerar una nueva variante para un componente existente. Además, sigue siendo una solución limitada pues solo puede detectar algunos tipos de fallos (componente faltante (E.1), sobrante (E.2) o incorrecto (E.3)). En la siguiente sección analizamos soluciones que utilizan redes siamesas, las cuales intentan superar estas limitaciones adoptando un paradigma de clasificación diferente que permite tratar clases no vistas durante el entrenamiento.

4.2. Redes Siamesas

Los problemas de aprendizaje automático pueden dividirse en dos grandes categorías: *closed-set* y *open-set*. En un problema *closed-set* el conjunto de clases está completamente definido durante el entrenamiento y el modelo solo debe clasificar dentro de estas clases conocidas. Por otro lado, en un problema *open-set* el conjunto de clases no está completamente definido durante el entrenamiento y el modelo puede encontrarse con clases desconocidas durante la inferencia. En la sección anterior presentamos una solución de aprendizaje profundo tradicional, formulada como *closed-set*, donde las muestras de *train* y *test* pertenecen al mismo espacio de etiquetas. Si bien obtuvimos buenos resultados, su utilidad práctica se ve restringida.

Existen varios problemas, como por ejemplo la verificación facial, donde se han observado dificultades similares. En este caso, la tarea a resolver es determinar si un rostro pertenece a una persona. Para ello, es imposible implementar un clasificador de gran escala con una clase por cada usuario, dado que esto generaría un número excesivo de clases. Además, requeriría una cantidad considerable de imágenes para cada individuo y tendría que ser reentrenado constantemente para incorporar a nuevos usuarios. Otro ejemplo análogo es la verificación de firmas para la detección de fraudes, ya que es inviable una implementación que demande una clase para las firmas verdaderas y otra para las falsas para cada usuario.

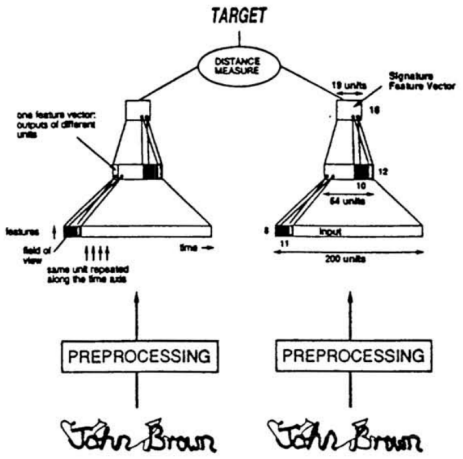


Figura 4.8: Arquitectura de la red siamesa (extraído de [6], 1993). Dos ramas paralelas procesan las entradas por separado y se computa una distancia como salida.

Resulta más conveniente, aunque más complejo de resolver, formular estos problemas como *open-set*, donde pueden aparecer clases en *test* que no estaban presentes durante el entrenamiento. En 1993 [6] introdujo una nueva arquitectura de redes neuronales para atacar el problema de la verificación de firmas, la red siamesa, que permite modelar el problema como *open-set* (figura 4.8). A diferencia de otras redes clásicas, este diseño acepta dos entradas independientes y su salida

es una medida de qué tan similares son ambas entradas. Esta innovación le permite cambiar un problema de clasificación tradicional por uno binario, donde la salida predice si las entradas son de la misma clase o no.

La formulación con redes siamesas es mucho más flexible, pues puede entrenarse con una base de datos cualquiera, tanto en el caso de verificación de firmas [12] como verificación facial [48], **sin necesidad de conocer datos de los usuarios finales durante el entrenamiento**. La red aprende a comparar las entradas, lo que es fácilmente generalizable sin reentrenar todo el modelo cuando aparecen clases nuevas. Asumiendo que la red tenga la capacidad y datos suficientes, bastaría tener una sola firma o rostro auténtico para poder comparar contra una instancia desconocida y poder verificarla.

Los modelos para problemas *open-set* se dividen en tres categorías, según cuántas instancias de la clase desconocida pueden usar durante la inferencia: *zero-shot*, *one-shot* y *few-shot learning*. En *zero-shot learning*, los modelos reconocen clases nunca vistas usando solo conocimiento previo, es decir, sin ninguna instancia. En *one-shot learning*, tienen una única instancia de la nueva clase. En *few-shot learning*, disponen de pocas instancias para aprender, aunque no hay un límite superior riguroso que las defina. Las redes siamesas naturalmente están diseñadas para operar en modo *one-shot*, aunque es generalizable a *few-shot*, por ejemplo clasificando varias veces y decidiendo por mayoría.

La detección de errores de populado en un PCBA tiene bastantes similitudes con los problemas que expusimos previamente. La diversidad de componentes electrónicos es tan grande que no es posible entrenar un modelo que resuelva un problema de clasificación general. Sin embargo, podemos contar con una imagen de referencia de un PCBA, de la misma forma que contamos con una imagen de referencia de un rostro o una firma, y compararla con la imagen a inspeccionar. Esto nos permite, en principio, entrenar un modelo general que clasifique usando *one-shot learning* y no necesite ser reentrenado para distintos PCBA.

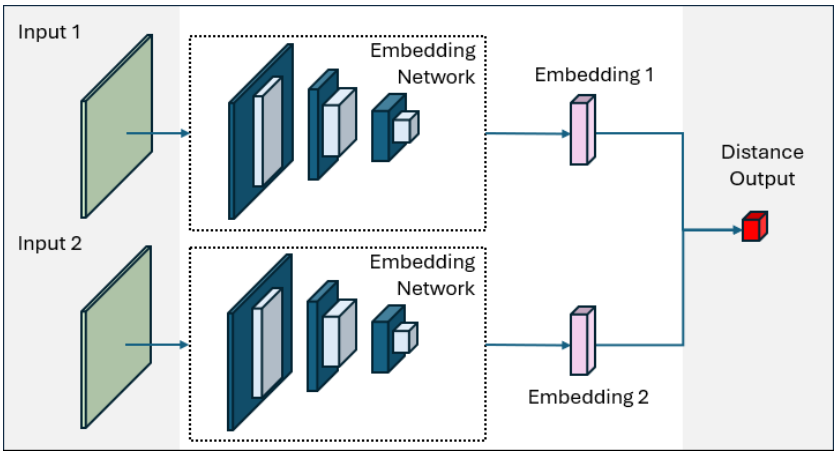


Figura 4.9: Arquitectura básica de red siamesa moderna. Similar a 4.8, pero con *embedding networks* convolucionales y sin *feature engineering*.

La arquitectura general se basa en dos instancias idénticas de la misma red,

llamada *embedding network* (figura 4.9). Estas procesan cada entrada en paralelo para obtener una representación de cada objeto en un espacio vectorial de más baja dimensión, conocida como *embedding*. Finalmente, la salida de la red es una distancia vectorial entre los *embeddings*. La idea detrás de esto es que el espacio sea tal que agrupe semánticamente los objetos de entrada, es decir, que las representaciones de objetos similares estén cerca entre sí, mientras que las representaciones de objetos diferentes estén más alejadas. Este agrupamiento facilita tareas como la clasificación y la detección de anomalías, ya que la proximidad en el espacio vectorial se correlaciona con la similitud en las características de los objetos. Por ejemplo, una tarea de clasificación *one-shot* puede implementarse comparando un elemento a clasificar contra cada representante de las clases posibles y eligiendo la que de la distancia menor.

La red presentada en [6] (figura 4.8) es antigua, pues tiene pocas capas y usa *feature engineering* para codificar las entradas. Más recientemente se han presentado avances muy significativos. En [12] se presenta una red siamesa convolucional profunda para la verificación de firmas, mientras que [25] y [48] presentan arquitecturas similares para resolver problemas de *one-shot learning* de clasificación de caracteres y verificación de rostros, respectivamente. Dado que han tenido buenos resultados con problemas similares, en esta sección estudiaremos la aplicación de una red siamesa al problema de detección de errores en PCBAs. Trabajamos con [25] como referencia principal, pues reporta resultados sobre Omniglot, una base de datos pública y de fácil acceso. Si bien es un *dataset* genérico de reconocimiento de caracteres, nos permite reproducir resultados y comparar rendimientos durante el desarrollo antes de adaptar el problema a componentes electrónicos. Los cambios y mejoras propuestas a la arquitectura y el entrenamiento de la red se detallan en las siguientes secciones.

4.2.1. Arquitectura Propuesta

La arquitectura de una red siamesa se compone de dos grandes piezas. Por un lado, la *embedding network*, que es la red encargada de generar las representaciones vectoriales de las entradas. Por otro lado, la métrica elegida en el espacio del *embedding*, la cual determina cómo se comparan estas representaciones para evaluar su similitud o disimilitud. A continuación describiremos en detalle el diseño de cada parte.

Embedding Network

En la figura 4.10 puede verse la *embedding network* que utilizamos. Es una red casi exclusivamente convolucional, compuesta de una secuencia de bloques como se detallan en la tabla 4.5. Utilizamos capas de *batch normalization* (BN), convolución bidimensional con núcleo $m \times m \times n$ (que anotamos como $\text{Conv}(m, n)$), ReLU y MaxPooling bidimensional 2×2.

La arquitectura se basa en [25], con algunas modificaciones propuestas por este trabajo. En primer lugar, cambiamos las dimensiones de la entrada de $(105, 105, 1)$ a $(128, 128, 3)$ para incluir la información de color y tener más resolución, pues de

Block1	*Block2*	*Block3*	Block4	Block5
BN	BN	BN	BN	Flatten
Conv(10,64)	Conv(7,128)	Conv(4,128)	Conv(4,256)	BN
ReLU	ReLU	ReLU	ReLU	FC(4096)
MaxPool	MaxPool	MaxPool	MaxPool	Sigmoid

Tabla 4.5: Detalle de los bloques de la *embedding network*

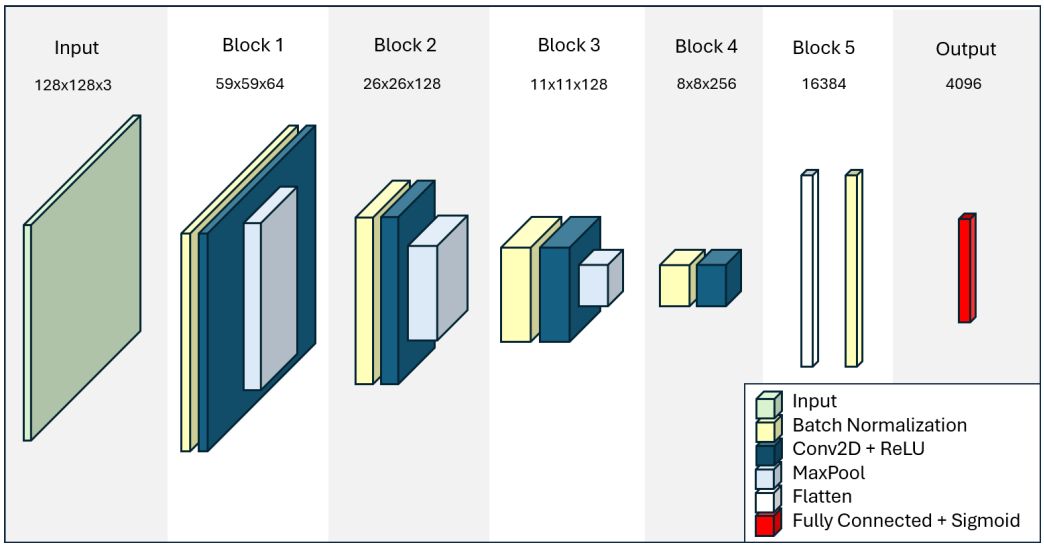


Figura 4.10: *Embedding network* para la red siamesa

otra manera no sería suficiente para capturar los detalles de menor escala. Esto implica que cambian los tamaños de los filtros convolucionales también. Además, agregamos capas de *batch normalization* antes de las capas *fully connected* pues observamos que facilita enormemente el entrenamiento. Incluso al intentar reproducir los resultados con Omniglot, suele ser difícil lograr la convergencia sin estas capas, pues el *accuracy* se estanca en 0,5 indefinidamente.

Esta red tiene aproximadamente 68 millones de parámetros, por lo que es considerablemente más sencilla que otras redes convolucionales como VGG16, que ya estudiamos en secciones anteriores. Contar con una red más potente podría dar mejores resultado, ya sea usando VGG16 o alguna variación de esta misma red con más capas o resolución de entrada, pero el costo de entrenamiento se vuelve prohibitivo para los recursos con los que contamos para este trabajo. Recordamos que la *embedding network* se aplica a cada entrada por separado, por lo que el cómputo del gradiente debe ejecutarse dos veces, incluso si los pesos son compartidos entre ambas ramas. En particular, al entrenar la red siamesa con VGG16 como *embedding network* hallamos que la VRAM de la mejor GPU disponible en el Cluster-UY [37] en ese momento, NVIDIA A100, no era suficiente.

Métricas en el Espacio del *Embedding*

Formularemos el problema más rigurosamente para poder discutir las distintas alternativas que hallamos en la literatura, así como la implementación de este trabajo. Sea un conjunto de vectores de entrenamiento $S \subset \mathbb{R}^{H \times W \times C}$, donde cada elemento pertenece a una clase cualquiera. Tomamos duplas de vectores de entrenamiento, $(x_i, y_i) \in S^2$, y creamos etiquetas $z_i \in \{0, 1\}$ que valen 0 si x_i e y_i pertenecen a la misma clase y 1 en caso contrario. La *embedding network* es entonces una función que cumple

$$f : S \rightarrowtail T \subset \mathbb{R}^d$$

Llamamos espacio del *embedding* a T, el codominio de f, que queda incluido en $\mathbb{R}^d$ para algún $d \ll HWC$.

Para completar la arquitectura de la red siamesa, es necesario definir cuál es la métrica del espacio del *embedding*, es decir, cómo vamos a medir distancia dentro del espacio T. Una primera opción, quizás la más intuitiva, es tomar una p-norma vectorial, de manera tal que la distancia entre dos vectores (x_i, y_i) sea

$$D(x_i, y_i) = ||f(x_i) - f(y_i)||_p$$

Este es el caso de trabajos como [9], [12] y [13], que usan normas L1 y L2. Otra manera, como se presenta en [6], es usar la distancia coseno.

Sin embargo, en trabajos más recientes como [48] y [25], los autores optan por fijar una cierta estructura para la distancia e incluir los coeficientes dentro de la optimización, por lo que la métrica se aprende parcialmente durante el entrenamiento. Esta es la implementación que seguimos en el presente trabajo. Una vez obtenidos los *embeddings* $f(x_i)$ y $f(y_i)$, se procesan con una capa de distancia L1 elemento a elemento para luego pasar por *batch normalization* y una red *fully connected* con activación sigmoide. Por lo tanto, la distancia tiene la forma

$$D(x_i, y_i) = \sigma\left(\sum_{k=1}^{d} \alpha_k \left|f(x_i)_k - f(y_i)_k\right| + \beta_k\right)$$

Donde los subíndices k denotan el k-ésimo elemento del vector y los coeficientes α_k y β_k son los que resultan de la composición del *batch normalization* y la capa *fully connected*, que son aprendidos durante el entrenamiento. Notamos que esta formulación cumple las propiedades de no negatividad y de simetría necesarias para una distancia, pero no cumple necesariamente que la distancia de un vector consigo mismo sea siempre nula, ni la desigualdad triangular. Sin embargo, es esperable que el entrenamiento converja a una solución donde, al menos, la distancia de un vector consigo mismo sea muy cercana a cero.

Red Completa

La red completa puede verse en la figura 4.11. Existen dos ramas paralelas, una correspondiente a cada entrada, las cuales son procesadas simultáneamente por la *embedding network*. Ambas ramas comparten los mismos pesos. Tras obtener el

embedding, empleamos una capa de distancia L1 punto a punto ponderada que une las ramas en una rama central. Posteriormente, aplicamos una capa de *batch normalization*, seguida de una *fully connected* con activación sigmoide. Esta última capa tiene una única salida que codifica la distancia entre las entradas. En total, la red cuenta aproximadamente con 68 millones de parámetros, que en su enorme mayoría pertenecen a la *embedding network*.

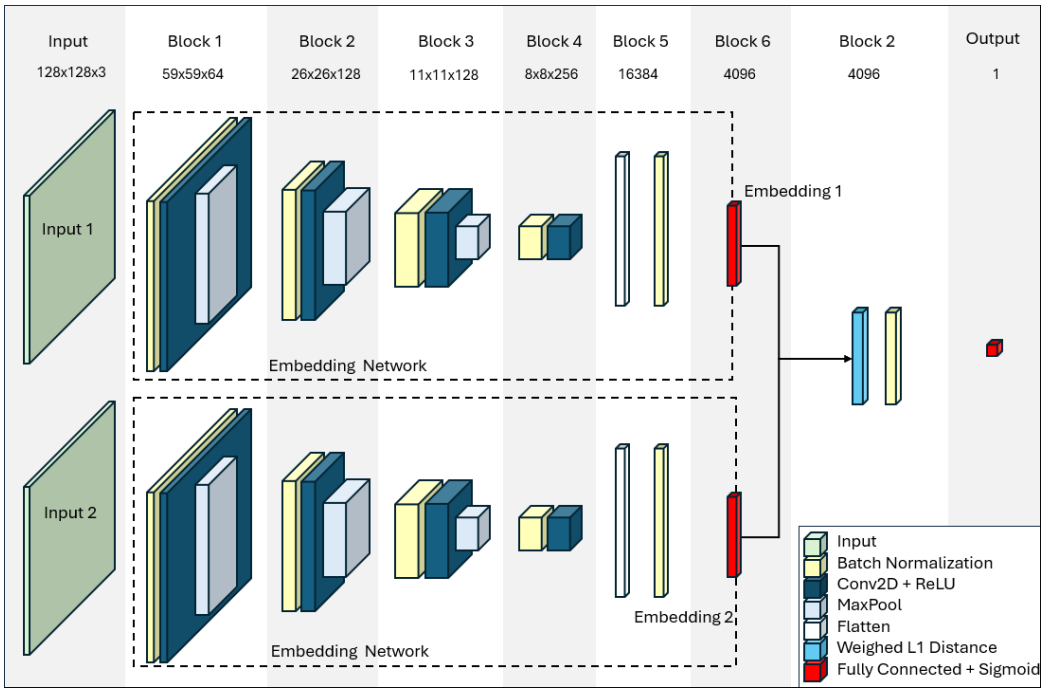


Figura 4.11: Red siamesa implementada

4.2.2. Entrenamiento

Función de Pérdida

Para entrenar la red nos basamos en el algoritmo descripto en [13], donde se presenta la *contrastive loss*. Esta es una función de pérdida diseñada para entrenar una red siamesa que computa distancia, usando duplas de datos. A continuación describimos brevemente la motivación y posibles ventajas; la sección A.2 presenta más detalles al respecto. La función de pérdida para una dupla se define, dado un margen $m > 0$, como

$$L(x_i, y_i) = \frac{1 - z_i}{2} D^2(x_i, y_i) + \frac{z_i}{2} \max{}^2\{0, m - D(x_i, y_i)\}$$

El objetivo de la función de pérdida en este problema es poder aprender el mapeo f que agrupa semánticamente a las entradas. El margen m establece una distancia a partir de la cual un par "malo" ($z_i = 1$) tiene pérdida nula, permitiendo al algoritmo de optimización priorizar los pares que aún no ha logrado clasificar

bien. Además, es un buen umbral para usar durante la inferencia. Si usáramos entropía cruzada deberíamos asumir 0,5 o elegir un punto de operación basados en la ROC en caso de tener un *dataset* desbalanceado.

Si bien nuestra red termina en una capa sigmoide, por lo que podríamos usar entropía cruzada, elegimos trabajar con la *contrastive loss* porque promete otorgar un poco más de estabilidad al entrenamiento y el margen nos da un grado de libertad más, en caso de que fuera necesario. De todas formas, la entropía cruzada puede usarse de manera satisfactoria, como muestran [25] y [48].

Dataset

Trabajaremos con el *dataset* DS-ENCAPSULADOS. Recordamos que, para el modelo A, la partición de *background* consiste de 8812 imágenes de componentes electrónicos etiquetados por su encapsulado, distribuidas en 25 clases de una manera desbalanceada, mientras que la partición de *evaluation* contiene 413 imágenes de 4 clases distintas. Llevamos todas las imágenes a una resolución de 128×128 pixeles, escalando la dimensión más grande.

En primer lugar separamos la partición de *background* en conjuntos de *train* y *test* en 80 % y 20 %, haciendo un sorteo ponderado para que todas las clases sean equiprobables.[2] Dentro del conjunto de *train*, usamos aumentación agregando rotaciones de 90°, 180° y 270°. Luego, dentro de cada conjunto y para cada clase, creamos tantas duplas de componentes de la misma clase como podamos ($z_i = 0$) y creamos la misma cantidad de duplas con componentes de otras clases ($z_i = 1$) aleatoriamente, con clases equiprobables. Dada el alto desbalance del *dataset*, limitamos la cantidad de duplas a 2.000 por cada clase (1.000 "buenas" y 1.000 "malas"), muestreando aleatoriamente de las generadas. El conjunto de *train* lo dividimos nuevamente en una razón 80 %-20 % aleatoriamente para obtener un conjunto de validación. Esto da un total de 67.938 duplas, de las cuales 40.000 son para el entrenamiento, 10.000 para validación y 17.938 para *test*.

Es importante notar que siempre es posible construir muchas más duplas "malas" que "buenas", por lo que implícitamente las estamos descartando. A su vez, para algunas clases descartamos una gran cantidad de duplas "buenas" explícitamente, debido al límite de duplas por clase. Esto nos permite tener un conjunto balanceado a la hora del entrenamiento, pero desaprovecha parte del *dataset*. Sin embargo, es posible regenerar las duplas del conjunto de entrenamiento al final de cada época, por ejemplo, con el objetivo de explotar al máximo todas las combinaciones posibles del *dataset*, sin introducir desbalances graves.

Esta manera de crear el *dataset*, en particular la aumentación con rotaciones, tiene como consecuencia que la red aprenda a distinguir tipos de componentes por encapsulado (componente faltante (E.1), sobrante (E.2) o incorrecto (E.3)), pero no su posición (E.4) u orientación (E.5). Sería deseable que la red pudiera distinguir

[2] En la sección 4.1.3 trabajamos con el mismo conjunto de datos pero separamos en una proporción de 50 %-50 %, para evitar que el conjunto de *test* fuera demasiado reducido. En este caso no es tan necesario porque construiremos parejas de imágenes, lo que da muchas combinaciones con pocas instancias.

al menos la orientación, para poder detectar errores de polaridad. Sin embargo, al crear clases independientes para cada rotación de 90°, 180° y 270°, observamos que el rendimiento de la red fue inferior al deseado para esta aplicación, lo que sugiere que carece de la capacidad suficiente. El modelo obtenido de este entrenamiento realizará una tarea similar a clasificación *open-set*, pero hay algunos tipos de fallas que quedan fuera de su alcance.

Optimización y Búsqueda de Hiperparámetros

En [25] los autores describen una búsqueda de hiperparámetros de una dimensionalidad muy alta. Dado que nuestra red es similar, nos basamos en ese trabajo pero con numerosas simplificaciones para reducir el tiempo de búsqueda.

Usamos un optimizador ADAM con *batch size* de 128, velocidad de aprendizaje η y regularización cuadrática para las capas convolucionales y *fully connected*, con peso λ. Implementamos un *learning schedule* de parámetro ρ donde se decrementa la velocidad de entrenamiento multiplicativamente en cada época, de manera que la velocidad de aprendizaje en la época i verifica $\eta_{i+1} = \rho\eta_i$. Entrenamos por un máximo de 25 épocas, monitoreando el *accuracy* de validación y aplicando *early stopping* con paciencia de 5 épocas. Como mencionamos en la sección anterior, regeneramos las duplas de entrenamiento al final de cada época.

Inicializamos los pesos de las capas convolucionales con una normal de media cero y desviación estándar 10^{-2} y los *biases* con media 0,5 e igual desviación. Los *biases* de las capas *fully-connected* se inicializan de la misma forma, pero los pesos se toman con una desviación de 2×10^{-1}.

Para la búsqueda de hiperpárametros utilizamos optimización Bayesiana con 128 entrenamientos. El espacio de hiperparámetros es el siguiente: $\eta \in \left[10^{-5}, 10^{-3}\right]$, $\lambda \in \left[10^{-4}, 10^{-1}\right]$ y $\rho \in \left[9 \times 10^{-1}, 1\right]$.

El margen para la *contrastive loss* lo excluimos de la búsqueda de hiperparámetros y lo fijamos en $m = 0{,}5$ pues hallamos que no hay una correlación fuerte entre su valor y el desempeño de la red al intentar reproducir los resultados de [25] con Omniglot.

4.2.3. Resultados

Hiperparámetros

La búsqueda de hiperparámetros arroja los mejores resultados para $\eta = 2{,}55 \times 10^{-4}$, $\lambda = 10^{-4}$ y $\rho = 0{,}9$. Tanto el peso de regularización como el factor del *learning rate decay* tomaron valores sobre el borde de su rango, por lo que es posible que mover el espacio de búsqueda arroje mejor rendimiento aun. Para este trabajo conservaremos estos resultados, pues la búsqueda es muy costosa y lleva varios días de ejecución en el Cluster-UY.

Desempeño Global

Con los hiperparámetros hallados, entrenamos la red 10 veces con particiones aleatorias del *dataset* y reportamos los resultados a continuación en la tabla 4.6. Al igual que en secciones previas, promediamos los valores de *accuracy* e incluimos la desviación estándar como incertidumbre. Puede verse un buen desempeño global, pero es conveniente analizar el rendimiento clase por clase para entender en detalle su comportamiento.

Subconjunto	*Accuracy*
Training	$0{,}982 \pm 0{,}004$
Validation	$0{,}980 \pm 0{,}004$
Test	$0{,}979 \pm 0{,}006$

Tabla 4.6: Desempeño de la red Siamesa.

Análisis por Clase

En la tabla 4.7 podemos ver el desempeño global y detallado. Para cada clase acumulamos todos las duplas de *test* que la involucren, tanto "buenas" como "malas", a través de los 10 entrenamientos con particiones aleatorias y computamos la precisión y el *recall*, considerando la "falla" como clase relevante. Observamos que la precisión es notoriamente buena, lo que significa que una predicción de fallo, es decir que una dupla tiene componentes distintos, es de muy alta confiabilidad.

Por otro lado, si bien muchas clases tienen un buen *recall*, algunas presentan un *recall* menor al deseado para esta aplicación, de entre 80 % y 90 %. En estos casos, hay errores que no están siendo detectados, es decir, duplas de componentes distintos que la red está clasificando como de la misma clase.

Para entender en más detalle aún los puntos débiles de la red, construimos una matriz de *accuracy*, que puede verse en la figura 4.12. Esta tiene un propósito bastante parecido al de una matriz de confusión, pero la información está organizada de una manera diferente. Asignamos a la entrada M_{ij} el *accuracy* obtenido al clasificar duplas que tienen elementos de la clase i y de la clase j. Por simetría de la red, $M_{ij} = M_{ji}$, pero la mostramos como una matriz triangular superior para no recargar tanto la imagen de información redundante.

Hallamos en la diagonal los elementos M_{ii}, que son todas las duplas "buenas". Estas tienen muy buena *accuracy*, lo que es consistente con la alta precisión global de la tabla 4.7. Luego, en el triángulo superior, tenemos todas las combinaciones posibles de duplas "malas", desagregadas por clases.

Observamos fácilmente confusiones en algunas clases problemáticas como entre condensadores C1206, C0603 y C0805 así como entre resistencias R1206 y R0603 e incluso integrados SOT-323 y SOT-23 (ver figura 4.13). Si bien esto es un poco inesperado, pues en la sección 4.1.4 el clasificador fue capaz de distinguirlas, es bastante razonable, ya que las clases son inherentemente poco separables. Es posible

Clase	Precisión	*Recall*	Soporte
Global	0,99	0,96	89690
10MSOP	1,00	0,96	4446
8MSOP	1,00	0,98	11731
8SOP150	1,00	1,00	7140
BSS138DW	1,00	0,98	23427
C_0603	1,00	0,91	23289
C_0805	1,00	0,90	23288
C_1206	1,00	0,82	4321
C_1210	1,00	0,94	4504
DFN3030-4	1,00	0,99	4435
DO-214AC	1,00	0,99	4433
LPS3314	0,99	0,99	4498
not_populated	1,00	0,99	23391
R_0603	1,00	0,98	23367
R_0603_no_marking	1,00	0,99	23254
R_0805	1,00	1,00	4485
R_1206	1,00	0,98	4426
SOD-123	1,00	0,99	4426
SOT23	1,00	0,91	11710
SOT23-5	1,00	0,96	18219
SOT23-8	1,00	1,00	4406
SOT-323	1,00	0,95	18141
TQFP100	1,00	0,99	4490
TQFP120	1,00	1,00	4362
TQFP64	1,00	1,00	4445
WSON8X6MM	1,00	0,97	4436

Tabla 4.7: Desempeño por clase

que esta diferencia se deba a una diversidad de motivos, como la menor capacidad de la red de *embedding* frente a VGG16, la menor resolución de las imágenes o la diferencia de arquitectura.

Por otro lado, vemos confusiones en clases más difíciles de explicar, como por ejemplo entre 10MSOP y 8MSOP, SOT23-5 y SOT23 o TQFP100 y DFN3030-4 (ver figura 4.14). En estos casos los componentes se ven muy diferentes, incluso con una cantidad de *pins* distinta. Si bien es complejo encontrar una razón por la que la red pueda equivocarse de esta manera mientras que en otros casos similares funcione correctamente (ver figura 4.15), es posible que se pueda mejorar el rendimiento forzando a la red a darle más peso a los *pins*, con un mecanismo de *attention*.

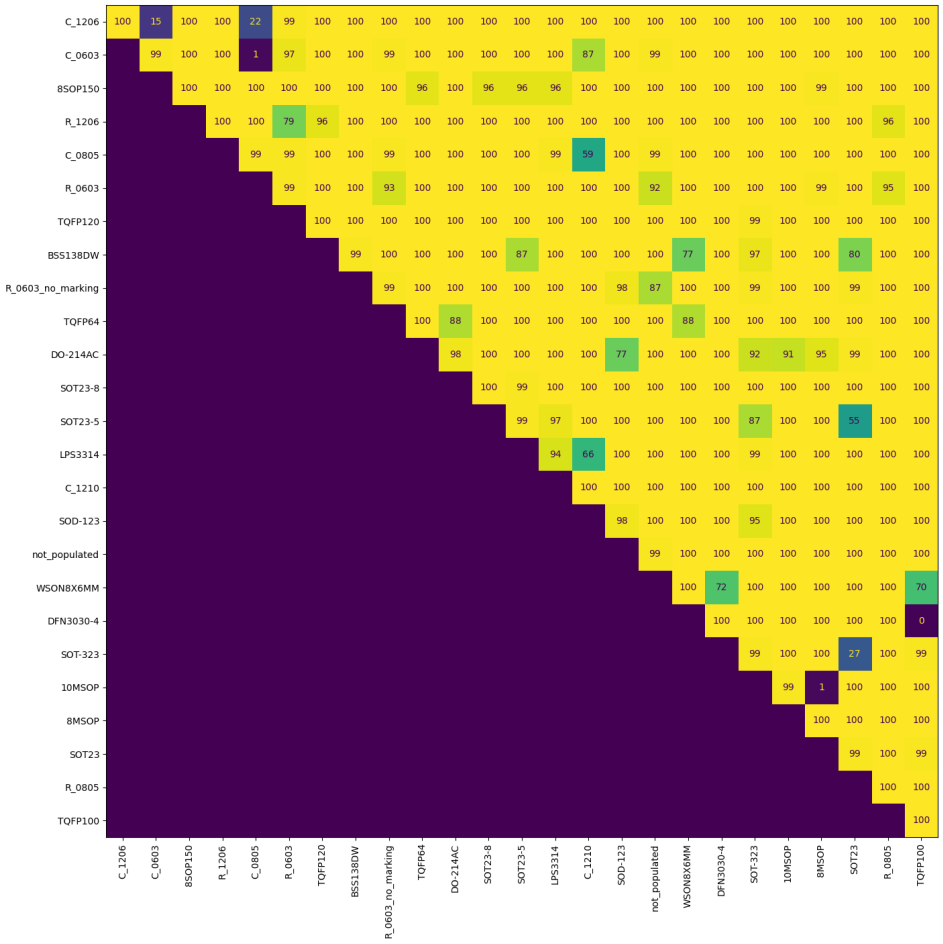
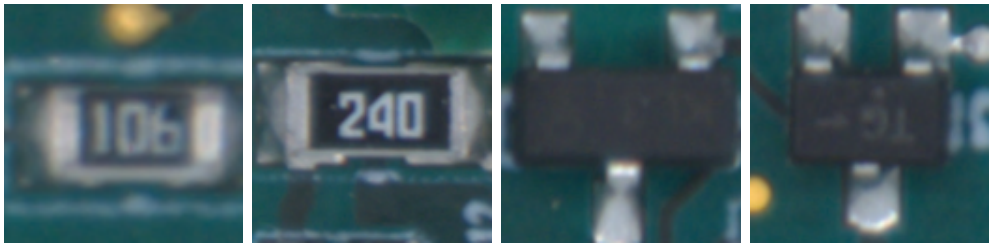


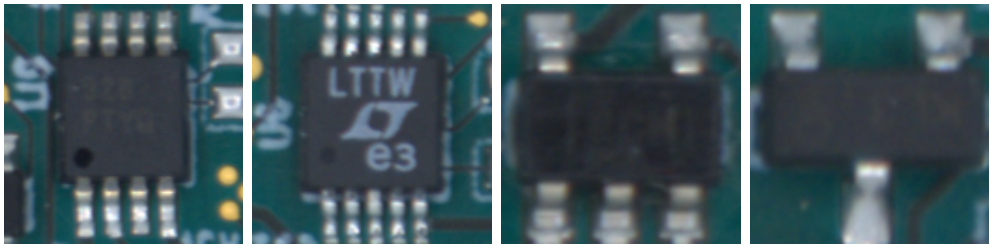
Figura 4.12: Matriz de *accuracy* en función de clases en la dupla

Es importante notar que no todas estas confusiones se darían en un escenario real ni son realmente relevantes, principalmente por un tema de escala. Durante el entrenamiento escalamos todos los componentes por su *bounding box* a 128×128 pixeles, pero, por ejemplo, una resistencia 1206 es del doble de tamaño que una 0603, por lo que no es una falla esperable en el campo, ya que ni siquiera sería posible soldarla en el mismo *footprint*.



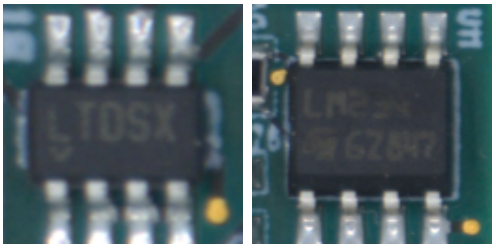
(a) Resistencia 0603 (b) Resistencia 1206 (c) SOT-23 (d) SOT-323

Figura 4.13: Confusiones entre clases similares. La red confunde (a) con (b) y (c) con (d), que son componentes muy similares. Presenta *accuracies* de 79 % y 27 % respectivamente, que son inferiores al promedio. Esto es un comportamiento esperado.



(a) 8MSOP (b) 10MSOP (c) SOT-235 (d) SOT-23

Figura 4.14: Clases separables con rendimiento menor al esperado. Los componentes de (a) y (b) son fácilmente distinguibles, al igual que (c) y (d), pero la red tiene más error que en otras clases aparentemente más difíciles. Presenta *accuracies* de 1 % y 55 %, que son inferiores al promedio. Este comportamiento es inesperado.



(a) SOT23-8 (b) 8SOP150

Figura 4.15: Clases poco separables con buen rendimiento. Los integrados de (a) y (b) son encapsulados distintos, pero visualmente muy similares. Sin embargo, la red logra distinguirlos con buen rendimiento (*accuracy* de 96 %).

Visualización del *Embedding*

Como mencionamos previamente, la arquitectura de redes siamesas se basa en calcular un *embedding* de más baja dimensión al espacio de entrada, de manera tal que la distancia en ese espacio tenga un valor semántico. Si bien el espacio del *embedding* sigue siendo de una dimensión demasiado alta para poder visualizarlo directamente (4096), podemos usar PCA para aplicar una reducción de dimensionalidad a 2D e intentar graficar los puntos obtenidos. En la figura 4.16 puede verse el resultado de esta operación para una muestra aleatoria de algunas de las clases presentes en el *dataset*.

Podemos observar que los componentes, incluso en dos dimensiones, son fácilmente separables y están agrupados de acuerdo a su clase. Además, es notable que dentro de la misma clase hay subgrupos que corresponden a su apariencia visual. Por ejemplo, en la clase C0603 se observan dos subgrupos, que se caracterizan por tener condensadores de color naranja y de color blanco. Cerca de estos últimos hallamos la clase C1202, que también tiene condensadores de color blanco.

De manera similar, se puede ver que las resistencias y los condensadores están próximos entre sí sobre la izquierda del gráfico, mientras que los integrados están agrupados entre ellos sobre la derecha. Finalmente, se observa que la clase 10MSOP tiene dos subconjuntos muy distintos, donde la diferencia parece ser la intensidad del *marking*.

En resumen, observamos el comportamiento esperado: la red mapea los vectores de entrada a un espacio donde se agrupan por similitud visual. Además, hallamos que este comportamiento puede verificarse en distintas escalas, tanto intra-clase como inter-clase.

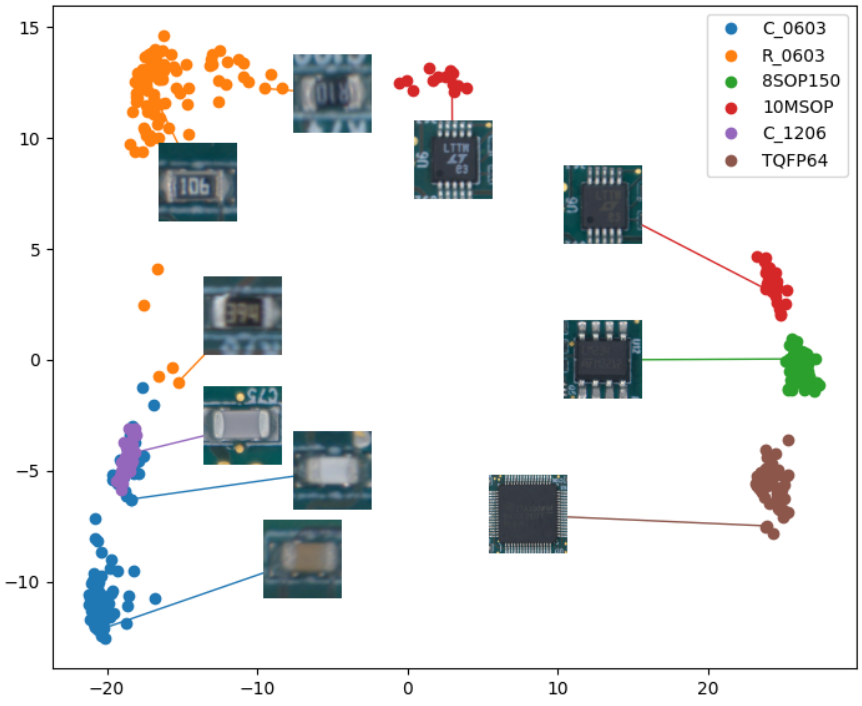


Figura 4.16: Visualización del espacio de *embedding* reducido a 2D por PCA.

Desempeño en modo *open-set*

La motivación principal detrás de la elección de la arquitectura de redes siamesas es su idoneidad para problemas *open-set*. Como tal, resulta interesante evaluar su desempeño al cambiar de *dataset*, en particular, sobre clases que no estuvieron presentes en el entrenamiento. Para ello, tomamos los conjuntos de *evaluation* tanto del modelo A como del modelo B de DS-PCBA y los unificamos para construir un único *dataset* con clases nuevas. Generamos pares con la misma lógica descripta previamente y tomamos una única partición de *test* para evaluar el rendimiento.

El desempeño global es un *accuracy* de 0,933, precisión de 0,918 y *recall* de 0,951. En la tabla 4.8 podemos ver el desempeño por clase y en la figura 4.17 la matriz de *accuracy* para cada tipo de dupla.

Notamos un desempeño muy bueno para algunas clases y un pequeño descenso en otras, comparando con el rendimiento general de *background*. Esto es esperable, como muestra [25]. Aún así, entendemos que el resultado final sigue siendo muy bueno para la aplicación deseada, especialmente teniendo en cuenta el tamaño del *dataset* de entrenamiento.

Clase	*Precision*	*Recall*	*Support*
32KHZ_SMD_A	0,99	0,99	2573
32KHZ_SMD_B	0,99	0,98	823
BSS84DW	1,00	0,92	879
CLIP_SHIELDING	1,00	0,97	2608
DCK	0,94	0,94	1460
DCK__SC70-6_	0,99	0,89	1365
DT1608C	0,99	0,97	842
F0603	0,98	0,93	2643
F1206	0,93	0,91	2572
R_ARRAY_4_0804	0,97	0,98	2610
SOT-563	0,83	0,96	2295

Tabla 4.8: Desempeño por clase en *evaluation*

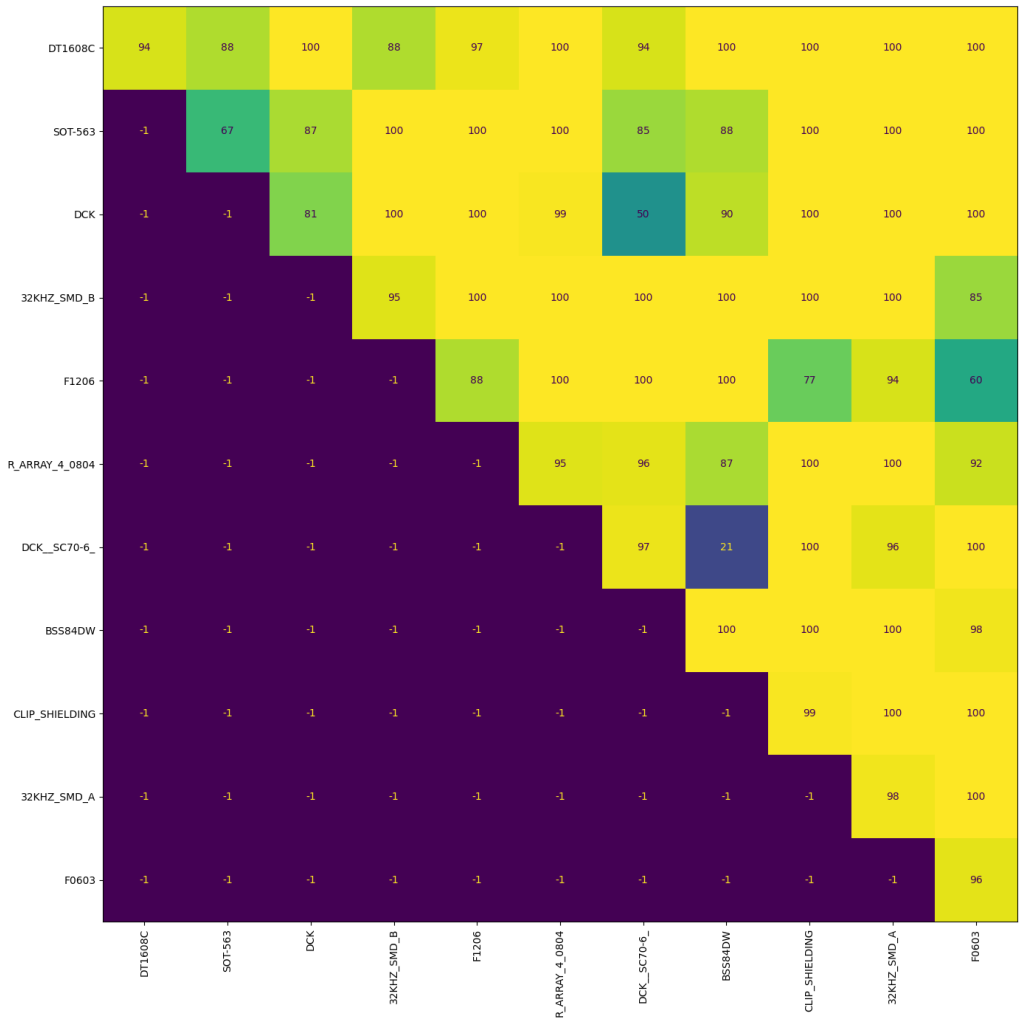


Figura 4.17: Matriz de *accuracy* en función de clases en la dupla para conjunto de *evaluation*

4.2.4. Conclusión

El análisis detallado del desempeño de la red revela una precisión global satisfactoria en la clasificación de componentes electrónicos. Aunque es destacable la alta fiabilidad en la detección de fallos, observamos desafíos en la distinción precisa entre algunas clases. Si bien entendemos que algunos de estos casos no reflejan un escenario real, hay otros que podrían potencialmente ser problemáticos, por lo que valdría la pena explorar hacer uso de los Gerbers para implementar la técnica de *attention*, enfatizando la relevancia de los *pins*.

Notamos que el entrenamiento de la red siamesa fue particularmente inestable y trabajoso, especialmente en comparación con el *transfer learning* de la sección anterior. Fue absolutamente necesaria una búsqueda de hiperparámetros y la introducción de capas de *batch normalization* para lograr la convergencia. A su vez, fue necesaria la aumentación con todas las rotaciones de 90° para obtener resultados aceptables, por lo que logramos un modelo que es capaz de detectar las fallas de componente faltante (E.1), sobrante (E.2) e incorrecto (E.3), pero no fallas de posición (E.4) u orientación (E.5). Entendemos que esto podría ser explicado por una falta de capacidad de la *embedding network*, por lo que sería interesante explorar una red más compleja, incluso ya inicializada con pesos preentrenados, combinado la red siamesa con la técnica de *transfer learning*.

A su vez, dada la inestabilidad observada y el hecho de que el margen de la *contrastive loss* no resultó ser relevante para el resultado, es posible que una *loss* de entropía cruzada tradicional obtenga resultados similares.

Destacamos principalmente que la red siamesa probó dar un desempeño más que aceptable al ser evaluada en régimen de *open-set*, ya que arrojó buenos resultados con clases de componentes que no estaban presentes durante el entrenamiento. Esto valida la elección de esta arquitectura y demuestra que es viable su aplicación a un problema de detección de fallos en PCBAs.

Capítulo 5

Evaluación del Sistema de AOI Completo

En el capítulo 4 estudiamos algoritmos destinados a detectar algunos tipos de errores de ensamblaje en PCBAs. Sin embargo, su desempeño no fue medido en un sistema punta a punta de inspección, si no que los evaluamos en las tareas de clasificación para la que fueron diseñados. En particular, ambos obtuvieron buen rendimiento en la detección de componentes incorrectos, pero no estamos considerando otras posibles fallas.

En este capítulo mostramos el resultado de realizar una inspección completa al *dataset* DS-PCBA usando el clasificador de red siamesa entrenado en la sección 4.2 y comparándolo con un clasificador sencillo basado en RMSE. Recordamos que dicho *dataset* tiene 85 fallas presentes, 79 del tipo "componente incorrecto" (E.3), 5 del tipo "error de polaridad" (E.5) y 1 del tipo "error de posicionamiento" (E.4)..

Dado el escaso tamaño del *dataset*, es posible que una gran proporción de los componentes sin fallas hayan formado parte del conjunto de entrenamiento. Además, la mayor parte de las fallas ocurren en clases de la partición de *background*. Esto significa que la medida de desempeño que obtengamos aquí no es buena, porque podría sufrir parcialmente de *overfitting*. De todas formas, entendemos que hay información útil para aprender al evaluar el sistema de una manera más realista. Además, refleja parcialmente cómo sería el uso normal pues, de un PCBA a otro, es esperable que algunos componentes efectivamente sí se repitan. Dicho esto, recordamos que como mencionamos en la sección 3.2.2, hay 30 fallas de "componente incorrecto" (E.3) en clases presentes en *evaluation*, por lo que la medida de desempeño que obtendremos es un punto medio entre un problema completamente *closed-set* y uno *open-set*, similar a un caso de uso real.

5.1. RMSE

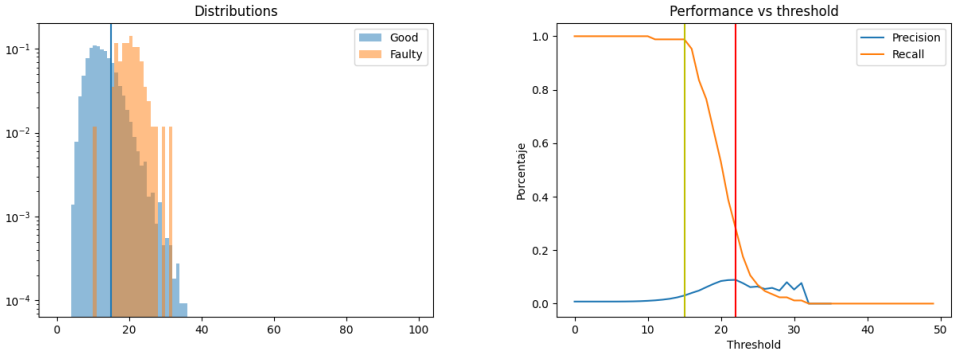
En primer lugar, estudiamos un clasificador sencillo basado en RMSE, con el objetivo de tener una línea base de comparación de desempeño. Tomamos una placa de referencia sin fallas e implementamos el diseño descripto en la sección

2. En este caso, la implementación del clasificador mostrado en la figura 2.2 es la siguiente

$$\text{RMSE}(x, y) = \sqrt{\frac{1}{HW}\sum_{i}^{H}\sum_{j}^{W} ||\text{rgb2lab}\,(x_{ij}) - \text{rgb2lab}\,(y_{ij})\,||^2}$$

Esto es una media cuadrática sobre la diferencia punto a punto de ambas imágenes. Tomamos los puntos en el espacio L*a*b*, dado que representa mejor las diferencias de color. El clasificador es simplemente una umbralización del RMSE. Para elegir los umbrales, estudiamos la distribución del RMSE para duplas con y sin fallas, como puede verse en la figura 5.1a. Notamos que claramente el *score* obtenido por el RMSE no logra separar las clases, por lo que el desempeño no puede ser muy bueno.

En la figura 5.1b podemos ver los resultados de precisión y *recall* en función del umbral de clasificación. Marcamos umbrales de advertencia en $\theta_{warn} = 15$ y de error en $\theta_{err} = 22$.



(a) Distribución de RMSE

(b) Precisión y *Recall* en función del umbral

Figura 5.1: Resultados para RMSE. Las clases no son separables y obtenemos un desempeño pobre.

En la tabla 5.1 mostramos los valores de las curvas para cada umbral así como el FPR. Como es evidente, el rendimiento es muy pobre y no sería posible utilizar este tipo de inspección en un contexto real debido a la enorme cantidad de falsos positivos.

Umbral	Precisión	*Recall*	FPR
Advertencia	0,03	0,99	0.25
Error	0,09	0,28	0.02

Tabla 5.1: Desempeño para distintos umbrales de clasificación.

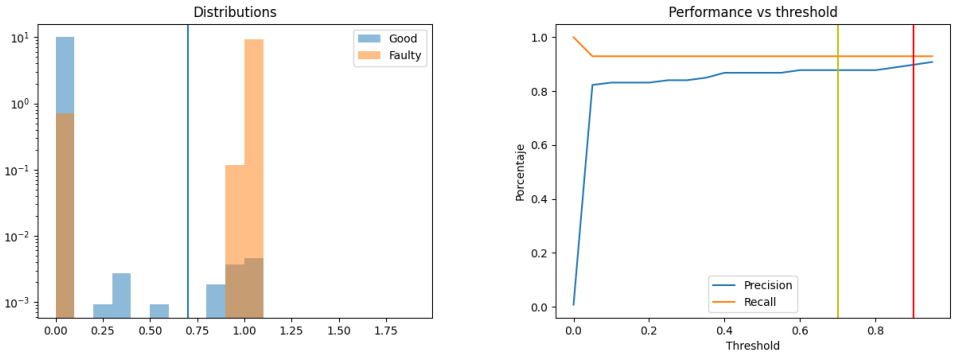
5.2. Red Siamesa

Repetimos el mismo procedimiento que para RMSE pero, en esta instancia, el *score* es la distancia computada por la red siamesa. Inspeccionar ambos lados de cada PCBA lleva aproximadamente 25 segundos de procesamiento, sin contar el tiempo de la intervención humana.

Desempeño General

En primer lugar, vemos en la figura 5.2a la distribución condicional del *score*. Esta es claramente bimodal para los componentes con fallas mientras que se comporta aceptablemente para componentes buenos. El comportamiento bimodal tiene una explicación sencilla y es la presencia de los componentes rotados. Como detallamos en la sección 4.2.2, el *dataset* de entrenamiento está aumentado con rotaciones de 90°, 180° y 270°. Es decir, por diseño la red no es sensible a las rotaciones de los componentes, por lo que esto es esperado.

Por otro lado, en la figura 5.2b vemos la curvas de precisión y *recall* en función del umbral. Tomamos umbrales de $\theta_{warn} = 0{,}7$ y de error en $\theta_{err} = 0{,}9$ para evaluar el rendimiento (ver tabla 5.2), aunque puede verse de la gráfica que es relativamente independiente.



(a) Distribución de *score*

(b) Precisión y *Recall* en función del umbral

Figura 5.2: Resultados para la red siamesa

Umbral	Precisión	*Recall*	FPR
Advertencia	0,88	0,93	0.0010
Error	0,90	0,93	0.0008

Tabla 5.2: Desempeño para distintos umbrales de clasificación.

Rotaciones

En la figura 5.3 podemos ver un ejemplo de una falla no detectada por la red siamesa, de tipo componente rotado (E.5). Existen algunas convenciones sobre cómo determinar la polaridad, que son necesarias en casos donde el componente tiene alguna simetría que lo convierte en invariante ante ciertas rotaciones, como puede verse en la imagen. Suelen utilizarse círculos o líneas en el *marking* o en el encapsulado que marcan el *pin* número 1 (ver figura 5.4), aunque no hay una convención establecida. Esta falta de consistencia entre fabricantes de ICs presenta un desafío considerable al momento de diseñar una solución general de bajo mantenimiento. En este caso particular, no hay ninguna indicación más allá del texto del *marking*, y el bajo contraste del dificulta la tarea aún más.

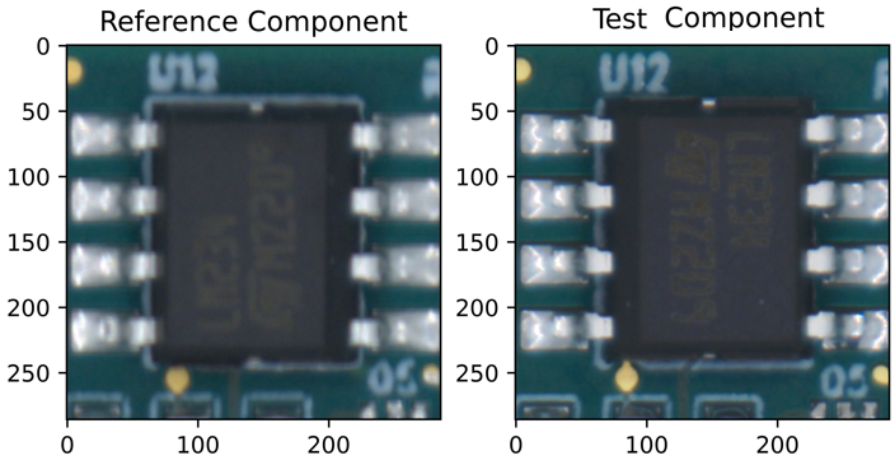


Figura 5.3: Clasificación incorrecta para error de tipo (E.5).

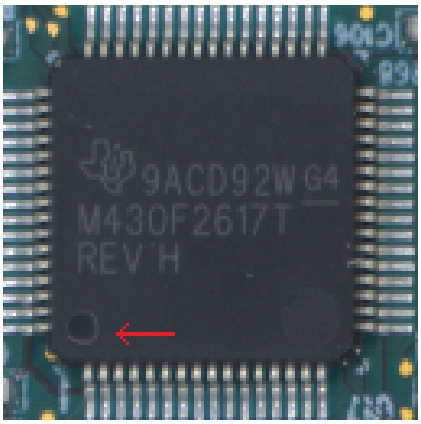
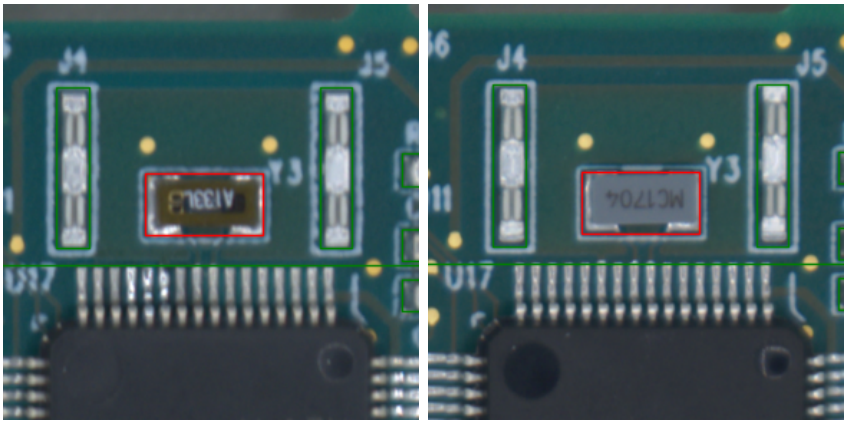


Figura 5.4: Ejemplo de identificación de polaridad, indicado con flecha roja.

Desempeño *open-set*

Notamos que, para la clase 32KHZ_SMD, que por ser parte del conjunto *evaluation* no estuvo presente durante el entrenamiento, se detectan el 100 % de los errores (ver Y3 en la figura 5.5), lo que parece prometer una buena generalización de la red, como habíamos observado en la sección 4.2.3. En la misma figura pueden verse componentes de clase CLIP_SHIELDING (J4 y J5), también excluida del entrenamiento, que son clasificados correctamente.



(a) Referencia (b) *Test*

Figura 5.5: Clasificaciones correctas para clases no presentes en *train*

5.3. Conclusión

Al utilizar la red siamesa como clasificador en el *dataset* de fallas de componentes observamos un rendimiento global muy bueno, de 90 % de precisión, 93 % de *recall* y 0.08 % de FPR, superando ampliamente al clasificador base de RMSE. Estas métricas implican que para una placa de 235 componentes (como es el modelo A de este *dataset*), es esperable detectar el 93 % de los fallas presentes y tener una media de menos de un falso positivo por placa.

Hallamos que la distribución del *score* es bimodal para las fallas, como era esperado, pues hay algunos tipos de fallas que no son detectables por diseño. Esto hace que el *recall* sea independiente del umbral y no tengamos manera de cambiar el comportamiento del clasificador sin re-entrenar. Sin embargo, los componentes buenos no presentan el mismo comportamiento, por lo que podemos mejorar considerablemente la precisión aumentando el umbral, sin compromiso aparente del *recall*.

Si bien los resultados son generalmente satisfactorios, notamos que esta solución no es capaz de detectar componentes rotados y que las medidas de desempeño son estimativos optimistas debido al posible *overfitting* dado el tamaño escaso del *dataset*.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 6

Conclusiones

En este trabajo, presentamos una arquitectura para un sistema AOI general basada en el paradigma de *golden board*, explotando información de diseño como Gerbers y BOMs. Propusimos la aplicación de algoritmos de aprendizaje automático novedosos en el problema de detección de fallas en PCBAs con el objetivo de disminuir el costo de mantenimiento del sistema de AOI. Desarrollamos y evaluamos estos métodos para adaptarlos a la aplicación propuesta, obteniendo buenos desempeños. Hallamos que los métodos desarrollados son entrenables con pocos datos y requieren menos mantenimiento que los métodos comúnmente discutidos en el estado del arte. Además, para evaluarlos, creamos conjuntos de datos originales de PCBAs con fallas y de componentes etiquetados por encapsulados.

En la implementación de *transfer learning*, los resultados obtenidos fueron muy buenos, con un entrenamiento relativamente sencillo. No obstante, este método requiere varias imágenes de cada PCBA y exige reentrenar el modelo cuando aparecen cambios en los componentes. La arquitectura de la red siamesa mostró ser mucho más flexible y resolver estas limitaciones, al costo de un entrenamiento significativamente más complejo y un rendimiento ligeramente inferior. Aún así, destacamos principalmente que el desempeño de la red siamesa fue más que aceptable incluso en la modalidad *open-set*, por lo que esta arquitectura es la mejor respuesta que hemos hallado a la problemática del mantenimiento del sistema de AOI planteado en la introducción de este trabajo.

Notamos que los algoritmos fueron diseñados para detectar los tipos de fallas presentes en el conjunto de datos. Aunque este enfoque es razonable, la limitada cantidad de fallas disponibles sugiere que los desempeños podrían estar sobrestimados. La carencia de conjuntos de datos públicos y de buena calidad para componentes electrónicos y fallas en PCBAs es el impedimento más grande para el desarrollo de un sistema general robusto de AOI. Una mayor variedad de componentes permitiría una mejor generalización para la red siamesa así como validar la eficacia de los algoritmos en diferentes escenarios y asegurar su robustez en aplicaciones reales. Por otro lado, condiciones de adquisición superiores en términos de resolución, distorsión óptica y múltiples fuentes de iluminación facilitarían la tarea de clasificación y registración en algunos de los componentes más pequeños y en la lectura de los *markings*.

En el comienzo de este trabajo clasificamos los distintos tipos de fallas más habituales en PCBAs (sección 1.4) y pusimos como objetivo desarrollar y evaluar el desempeño de distintos métodos en la detección de las mismas. Hallamos que los métodos desarrollados mostraron buen desempeño en fallas de tipo componente faltante (E.1), sobrante (E.2) e incorrecto (E.3) mientras que presentaron dificultades para las fallas de posicionamiento (E.4) y polaridad (E.5). Es posible que las fallas de posicionamiento sean abordadas exitosamente con técnicas estándar como OCR o NXCORR. Por otro lado, las fallas de polaridad presentan una complejidad considerable, especialmente en componentes auto-similares. Entendemos que aumentar la capacidad de la red de *embedding*, técnicas de *attention* y mejorar el contraste en la adquisición con luces en distintas orientaciones sería un buen camino a explorar para atacar estos problemas.

En el campo de la detección de fallas en PCBAs, la mayoría de los trabajos utilizan o bien bases de datos públicas limitadas, por lo que restringen el alcance del problema a algo de poca utilidad práctica, o bien bases de datos privadas de alta calidad, pero que son tan específicas que obtienen soluciones poco flexibles. Este trabajo se sitúa en un punto intermedio al emplear una base de datos privada, que, aunque acotada, nos permitió enfocarnos en restringir el problema a un nivel manejable pero con una potencial aplicación práctica general. En este sentido, los algoritmos evaluados demostraron ser efectivos, alcanzando desempeños aceptables para una implementación en el mundo real.

Apéndice A

Apéndice

A.1. Detalles y Ejemplo de Registración de PCB

En la sección 2.2 dimos una introducción a la arquitectura global de la registración y sus pasos. Aquí explicamos en más profundidad su funcionamiento y diseño, así como mostramos un ejemplo para facilitar la comprensión. Todo el procesamiento está hecho en Python usando principalmente implementaciones del paquete OpenCV.

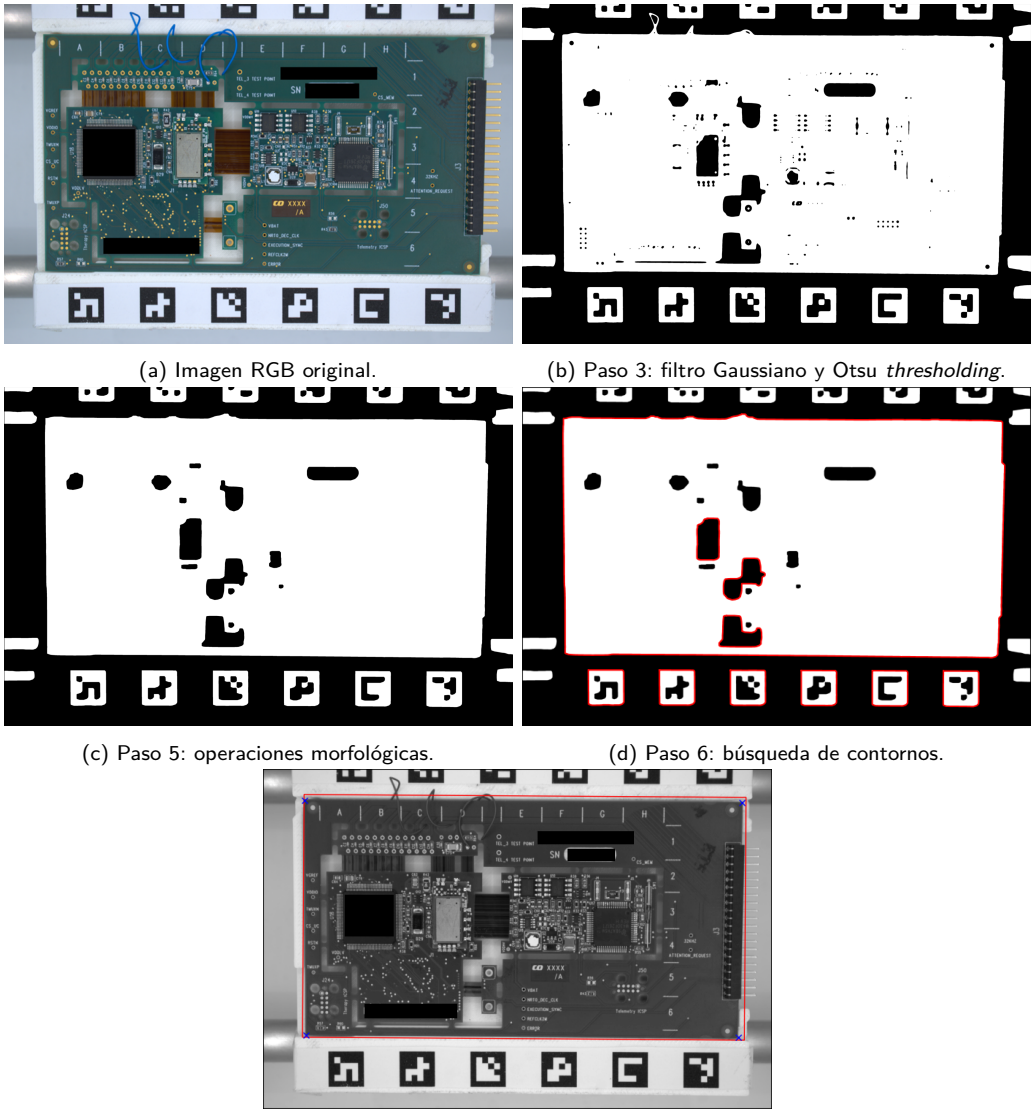
El proceso de registración comienza con una imagen RGB como la de A.1a, que está en una ubicación relativamente controlada pero desconocida[1]. Termina en una imagen como A.2e, donde la ubicación y tamaño son conocidos, por lo que podemos ubicar cada componente en la imagen a partir de sus posiciones obtenidas del Gerber y BOM.

Como mostramos en la figura 2.5, los primeros pasos consisten en convertir la imagen a escala de grises, aplicar un filtro pasa-bajos Gaussiano y Otsu *thresholding*. Con el *thresholding* estamos buscando ubicar a la placa dentro de la imagen. Dado que la placa es un objeto grande, aplicamos el filtrado para eliminar detalles de alta frecuencia que puedan entorpecer la detección. Sin embargo, esto no es suficiente, como puede verse en la imagen A.1b, pues algunos vestigios del cable quedan aun en la parte superior izquierda. Los próximos pasos de apertura y clausura morfológica (en ese orden) logran minimizar estos detalles lo suficiente para obtener una aproximación razonable de la placa original (figura A.1c).

Dado que el fondo es blanco y la placa es relativamente oscura, el *thresholding* logra etiquetar una gran proporción correctamente. Para descartar las partes del montaje de la placa que son etiquetadas erróneamente aplicamos una búsqueda de contornos, que marca los bordes continuos y cerrados dentro de la imagen. Tomando el de mayor área, encontramos el que corresponde al PCB (figura A.1d).

En este punto ya logramos segmentar la placa, pero necesitamos encontrar coordenadas de puntos conocidos para poder calcular una transformación de perspectiva. Para ello, primero buscamos las esquinas del PCB. Ajustamos el contorno

[1]Los marcadores ArUco (similares a QR) que se ven en la imagen no forman parte de este proceso, se ubicaron en el soporte para una prueba no relacionada de *stitching*.



(a) Imagen RGB original.

(b) Paso 3: filtro Gaussiano y Otsu *thresholding*.

(c) Paso 5: operaciones morfológicas.

(d) Paso 6: búsqueda de contornos.

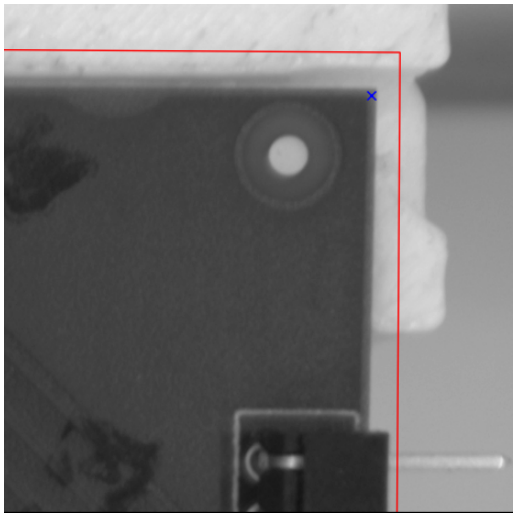
(e) Paso 8: ajuste de rectángulo y esquinas corregidas.

Figura A.1

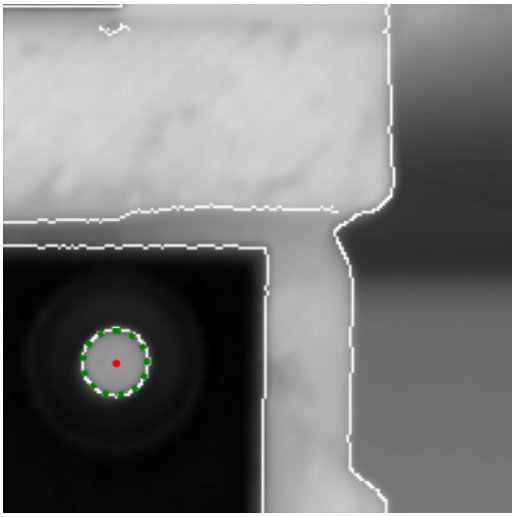
con un rectángulo rotado, como se ve en la figura A.1e y luego corregimos las esquinas predichas al punto más cercano del contorno (figura A.2a). A partir de las posiciones de las esquinas, buscamos en una ventana cercana a cada una los *fiducials* o, en su defecto, la esquina del PCB. Los *fiducials* son marcadores circulares, por lo que usamos Canny y una transformada de Hough para hallarlos, como se ve en A.2b. Por otro lado, la esquina del PCB la buscamos con un detector de esquinas de Harris, como se ve en A.2c y A.2d.

Finalmente, con las coordenadas de destino (tomadas del Gerber) de los cuatro puntos y las coordenadas de origen halladas en la imagen podemos calcular la matriz de la transformación proyectiva y obtener la imagen registrada como se ve

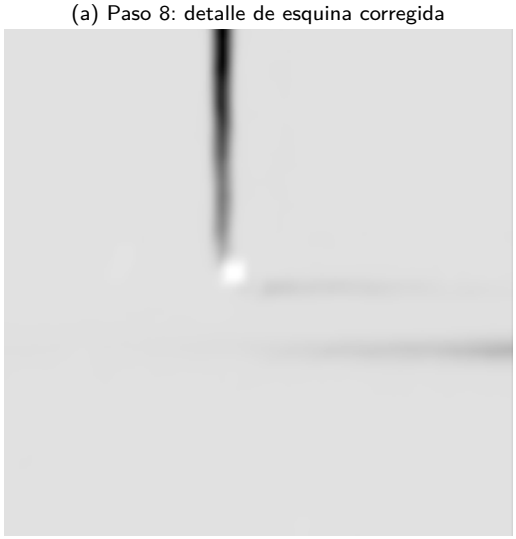
en A.2e.



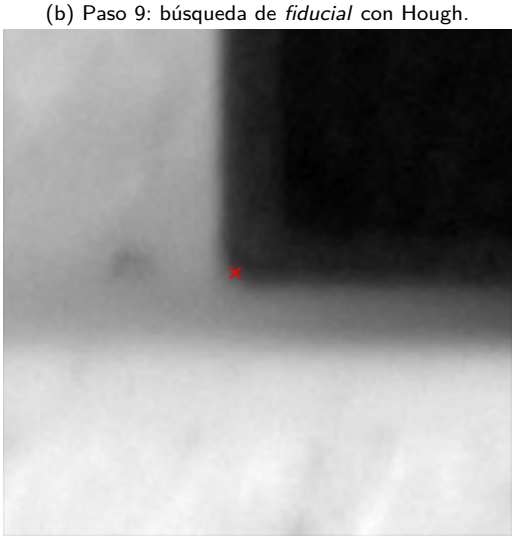
(a) Paso 8: detalle de esquina corregida



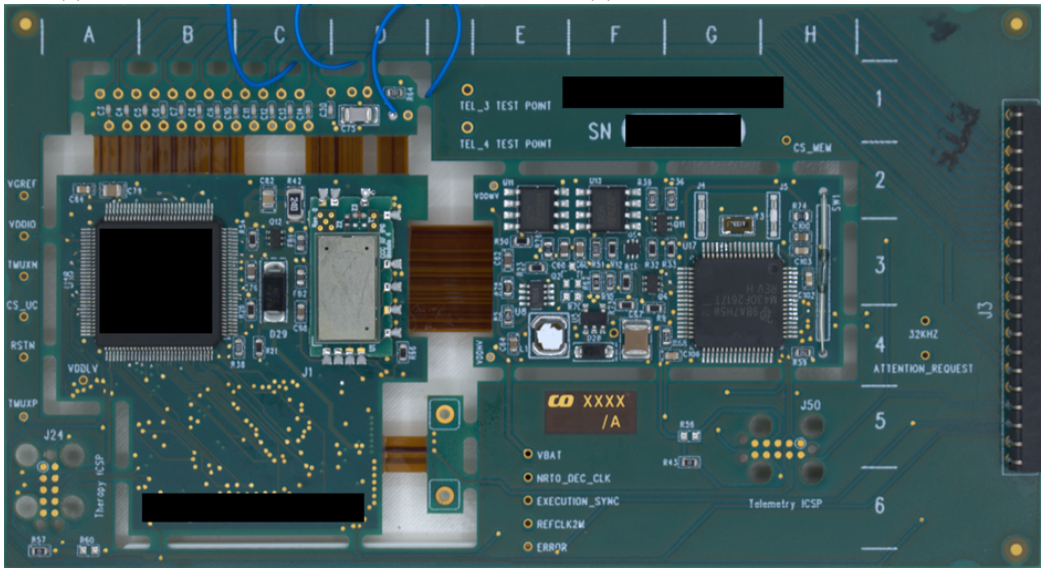
(b) Paso 9: búsqueda de *fiducial* con Hough.



(c) Paso 9: respuesta de detector de Harris.



(d) Paso 9: esquina detectada por Harris.



(e) Resultado: imagen registrada.

Figura A.2

A.2. *Contrastive Loss*

La función de pérdida para una dupla se define, dado un margen $m > 0$, como

$$L(x_i, y_i) = \frac{1 - z_i}{2} D^2(x_i, y_i) + \frac{z_i}{2} \text{máx}^{\,2}\{0, m - D(x_i, y_i)\}$$

Esta función es una suma de dos términos pero, dada la definición de z_i, solo uno va a ser no nulo a la vez. Para pares "buenos", es decir, donde ambas entradas pertenecen a la misma clase, $z_i = 0$, por lo que tenemos

$$L_{eq}(x_i, y_i) = \frac{D^2(x_i, y_i)}{2}$$

Como D es positiva, L_{eq} es monótona creciente con D y vale exactamente 0 para entradas idénticas. Por otro lado, para un par "malo", es decir, donde las entradas no pertenecen a la misma entrada, tenemos que $z_i = 1$ y, por lo tanto

$$L_{diff}(x_i, y_i) = \frac{1}{2} \text{máx}^{\,2}\{0, m - D(x_i, y_i)\}$$

Esto es una función partida que vale $\frac{1}{2}(m-D)^2$ si $D < m$ y 0 en caso contrario. Notar que, a diferencia del término L_{eq}, a partir de cierta distancia, la pérdida se considera nula independientemente del valor de D, anulando el gradiente. Además, es una función acotada y tiene como máximo $m^2/2$ en $D = 0$.

A modo de comparación, recordamos que la función de pérdida de entropía cruzada binaria es

$$L(x_i, y_i) = -z_i \log D(x_i, y_i) - (1 - z_i) \log [1 - D(x_i, y_i)]$$

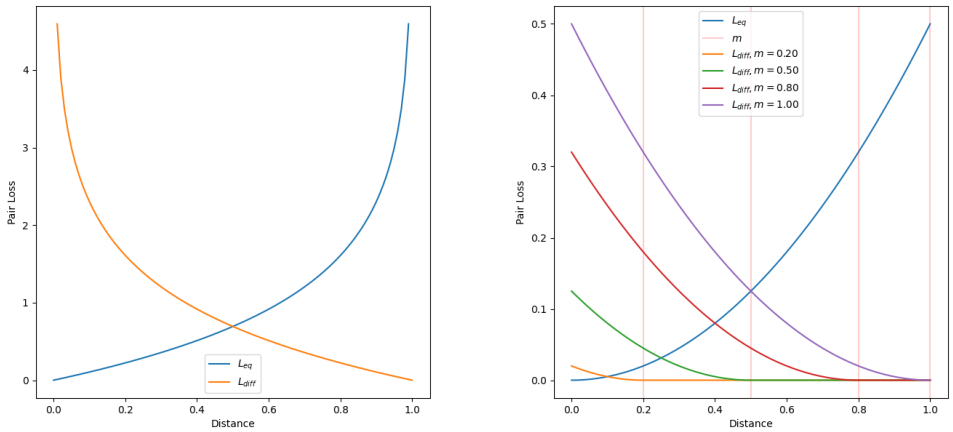
Y análogamente

$$\begin{aligned} L_{eq}(x_i, y_i) &= -\log [1 - D(x_i, y_i)] \\ L_{diff}(x_i, y_i) &= -\log D(x_i, y_i) \end{aligned}$$

En la figura A.3 podemos ver las gráficas de las funciones de pérdida en función de la distancia D entre los pares. Mostramos los términos L_{eq} y L_{diff} por separado, así como distintos casos de m para la *contrastive loss*.

A diferencia de la entropía cruzada, la *contrastive loss* es acotada en ambos extremos y no necesita que la distancia D esté normalizada al intervalo $[0, 1]$. Además, tiene un hiperparámetro m que permite introducir cierta asimetría entre los términos L_{eq} y L_{diff}, por lo que otorga un grado de libertad más que podría ser provechoso.

Como L_{diff} tiene gradiente nulo para $D > m$, los pares distintos ($z_i = 1$) que estén suficientemente lejos en el espacio del *embedding* dejan de afectar la optimización. Es posible que esto favorezca la estabilidad del entrenamiento y permita al optimizador concentrarse en pares que aún no ha logrado clasificar correctamente.



(a) Pérdida de entropía cruzada binaria.

(b) *Contrastive Loss*

Figura A.3

Referencias

[1] Requirements for soldered electrical and electronic assemblies, 2020. IPC J-STD-001 Revision H.

[2] M Abd Al Rahman and Alireza Mousavi. A review and analysis of automatic optical inspection and quality monitoring methods in electronics industry. *Ieee Access*, 8:183192–183271, 2020.

[3] Lifei Bai, Xianqiang Yang, and Huijun Gao. Corner point-based coarse–fine method for surface-mount component positioning. *IEEE Transactions on Industrial Informatics*, 14(3):877–886, 2017.

[4] Bruce G. Batchelor. *Machine Vision Handbook*. Springer, 2012.

[5] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*, pages 404–417. Springer, 2006.

[6] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a "siamese" time delay neural network. *Advances in neural information processing systems*, 6, 1993.

[7] Yih-Chih Chiou, Chern-Sheng Lin, and Bor-Cheng Chiou. The feature extraction and analysis of flaw detection and classification in bga gold-plating areas. *Expert Systems with Applications*, 35(4):1771–1779, 2008.

[8] Han-Jin Cho and Tae-Hyoung Park. Wavelet transform based image template matching for automatic component inspection. *The International Journal of Advanced Manufacturing Technology*, 50:1033–1039, 2010.

[9] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 539–546. IEEE, 2005.

[10] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

[11] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

[12] Sounak Dey, Anjan Dutta, J Ignacio Toledo, Suman K Ghosh, Josep Lladós, and Umapada Pal. Signet: Convolutional siamese network for writer independent offline signature verification. *arXiv preprint arXiv:1707.02131*, 2017.

[13] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[14] Mahmoud R Halfawy and Jantira Hengmeechai. Automated defect detection in sewer closed circuit television images using histograms of oriented gradients and support vector machine. *Automation in Construction*, 38:1–13, 2014.

[15] Robert M Haralick, Karthikeyan Shanmugam, and Its' Hak Dinstein. Textural features for image classification. *IEEE Transactions on systems, man, and cybernetics*, (6):610–621, 1973.

[16] Chris Harris, Mike Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[17] Abdel-Aziz IM Hassanin, Fathi E Abd El-Samie, and Ghada M El Banby. A real-time approach for automatic defect detection from pcbs based on surf features and morphological operations. *Multimedia Tools and Applications*, 78(24):34437–34457, 2019.

[18] Xie Hongwei, Zhang Xianmin, Kuang Yongcong, and Ouyang Gaofei. Solder joint inspection method for chip component using improved adaboost and decision tree. *IEEE Transactions on components, packaging and manufacturing technology*, 1(12):2018–2027, 2011.

[19] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[20] Yiming Huang, Di Wu, Zhifen Zhang, Huabin Chen, and Shanben Chen. Emd-based pulsed tig welding process porosity defect detection and defect diagnosis using ga-svm. *Journal of Materials Processing Technology*, 239:92–102, 2017.

[21] Kazunori Imoto, Tomohiro Nakai, Tsukasa Ike, Kosuke Haruki, and Yoshiyuki Sato. A cnn-based transfer learning method for defect classification in semiconductor manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, 32(4):455–459, 2019.

[22] BC Jiang and YM Wang CC Wang. Bootstrap sampling technique applied to the pcb golden fingers defect classification study. *International Journal of Production Research*, 39(10):2215–2230, 2001.

[23] Seunghyeon Kim, Wooyoung Kim, Yung-Kyun Noh, and Frank C Park. Transfer learning for automated optical inspection. In *2017 international joint conference on neural networks (IJCNN)*, pages 2517–2524. IEEE, 2017.

[24] Young-Gyu Kim and Tae-Hyoung Park. Smt assembly inspection using dual-stream convolutional networks and two solder regions. *Applied Sciences*, 10(13):4598, 2020.

[25] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille, 2015.

[26] Chung-Feng Jeffrey Kuo, Tz-ying Fang, Chi-Lung Lee, and Han-Cheng Wu. Automated optical inspection system for surface mount device light emitting diodes. *Journal of intelligent manufacturing*, 30:641–655, 2019.

[27] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[28] Yann LeCun. Learning invariant feature hierarchies. In *Computer Vision–ECCV 2012. Workshops and Demonstrations: Florence, Italy, October 7-13, 2012, Proceedings, Part I 12*, pages 496–505. Springer, 2012.

[29] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[30] Chung-Shou Liao, Tsung-Jung Hsieh, Yu-Syuan Huang, and Chen-Fu Chien. Similarity searching for defective wafer bin maps in semiconductor manufacturing. *IEEE Transactions on Automation Science and Engineering*, 11(3):953–960, 2013.

[31] Dae-ui Lim, Young-Gyu Kim, and Tae-Hyoung Park. Smd classification for automated optical inspection machine using convolution neural network. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 395–398. IEEE, 2019.

[32] Wu-Ja Lin and Sin-Sin Jhuo. A fast luminance inspector for backlight modules based on multiple kernel support vector regression. *IEEE Transactions on Components, Packaging; Manufacturing Technology*, 4(8):1391 – 1401, 2014.

[33] Hangwei Lu, Dhwani Mehta, Olivia Paradis, Navid Asadizanjani, Mark Tehranipoor, and Damon L Woodard. Fics-pcb: A multi-modal image dataset for automated printed circuit board visual inspection. *Cryptology ePrint Archive*, 2020.

[34] Gayathri Mahalingam, Kevin Marshall Gay, and Karl Ricanek. Pcb-metal: A pcb image dataset for advanced computer vision machine learning component analysis. In *2019 16th International Conference on Machine Vision Applications (MVA)*, pages 1–5. IEEE, 2019.

[35] Mukhil Azhagan Mallaiyan Sathiaseelan, Olivia P Paradis, Shayan Taheri, and Navid Asadizanjani. Why is deep learning challenging for printed circuit board (pcb) component recognition and how can we address it? *Cryptography*, 5(1):9, 2021.

[36] Stephane G Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE transactions on pattern analysis and machine intelligence*, 11(7):674–693, 1989.

[37] Sergio Nesmachnow and Santiago Iturriaga. Cluster-uy: collaborative scientific high performance computing in uruguay. In *Supercomputing: 10th International Conference on Supercomputing in Mexico, ISUM 2019, Monterrey, Mexico, March 25–29, 2019, Revised Selected Papers 10*, pages 188–202. Springer, 2019.

[38] Nobuyuki Otsu et al. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.

[39] Der-Baau Perng, Hsiao-Wei Liu, and Ching-Ching Chang. Automated smd led inspection using machine vision. *The International Journal of Advanced Manufacturing Technology*, 57:1065–1077, 2011.

[40] Der-Baau Perng, Hsiao-Wei Liu, and Ssu-Han Chen. A vision-based led defect auto-recognition system. *Nondestructive Testing and Evaluation*, 29(4):315–331, 2014.

[41] Christopher Pramerdorfer and Martin Kampel. A dataset for computer-vision-based pcb analysis. In *2015 14th IAPR International Conference on Machine Vision Applications (MVA)*, pages 378–381. IEEE, 2015.

[42] Jagdish Lal Raheja, Sunil Kumar, and Ankit Chaudhary. Fabric defect detection based on glcm and gabor filter: A comparison. *Optik*, 124(23):6469–6474, 2013.

[43] Zhonghe Ren, Fengzhou Fang, Ning Yan, and You Wu. State of the art in defect detection based on machine vision. *International Journal of Precision Engineering and Manufacturing-Green Technology*, 9(2):661–691, 2022.

[44] Jeffrey R Sampson. Adaptation in natural and artificial systems (john h. holland), 1976.

[45] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[46] Ji-Deok Song, Young-Gyu Kim, and Tae-Hyoung Park. Smt defect classification by feature extraction region optimization and machine learning. *The International Journal of Advanced Manufacturing Technology*, 101:1303–1313, 2019.

[47] Te-Hsiu Sun, Chun-Chieh Tseng, and Min-Sheng Chen. Electric contacts inspection using machine vision. *Image and Vision Computing*, 28(6):890–901, 2010.

[48] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[49] Gui Yun Tian, Rong Sheng Lu, and Duke Gledhill. Surface measurement using active vision and light scattering. *Optics and Lasers in Engineering*, 45(1):131–139, 2007.

[50] Pedro MA Vitoriano, Tito G Amaral, and Octávio Páscoa Dias. Automatic optical inspection for surface mounting devices with ipc-a-610d compliance. In *2011 International Conference on Power Engineering, Energy and Electrical Drives*, pages 1–7. IEEE, 2011.

[51] PM Vitoriano and Tito G Amaral. 3d solder joint reconstruction on smd based on 2d images. *SMT Magazine*, 31:82–93, 2016.

[52] Hao Wu, Xianmin Zhang, Hongwei Xie, Yongcong Kuang, and Gaofei Ouyang. Classification of solder joint using feature selection based on bayes and support vector machine. *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 3(3):516–522, 2013.

[53] Zhang Xue-wu, Ding Yan-qiong, Lv Yan-yun, Shi Ai-ye, and Liang Rui-yu. A vision inspection system for the surface defects of strongly reflected metal based on multi-class svm. *Expert Systems with Applications*, 38(5):5930 – 5939, 2011.

[54] Zhang Xue-Wu, Ding Yan-Qiong, Lv Yan-Yun, Shi Ai-Ye, and Liang Rui-Yu. A vision inspection system for the surface defects of strongly reflected metal based on multi-class svm. *Expert Systems with Applications*, 38(5):5930–5939, 2011.

[55] Kazım Yıldız, Ali Buldu, and Mustafa Demetgul. A thermal-based defect classification method in textile fabrics with k-nearest neighbor algorithm. *Journal of Industrial Textiles*, 45(5):780–795, 2016.

[56] Young-Geun Yoona, Seok-Lyong Leea, Chin-Wan Chungb, and Sang-Hee Kimc. An effective defect inspection system for polarized film images using image segmentation and template matching techniques. *Computers & Industrial Engineering*, 55:567–583, 2008.

[57] Deyong You, Xiangdong Gao, and Seiji Katayama. Wpd-pca-based laser welding process monitoring and defects diagnosis by using fnn and svm. *IEEE Transactions on Industrial Electronics*, 62(1):628–636, 2014.

[58] Eun Hye Yuk, Seung Hwan Park, Cheong-Sool Park, and Jun-Geol Baek. Feature-learning-based printed circuit board inspection via speeded-up robust features and random forest. *Applied Sciences*, 8(6):932, 2018.

[59] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 818–833. Springer, 2014.

Índice de tablas

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

Índice de figuras

Índice de figuras

Esta es la última página.
Compilado el miércoles 22 octubre, 2025.
http://iie.fing.edu.uy/