Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

Máster en Ingeniería Industrial

MÁSTER EN INGENIERÍA INDUSTRIAL

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

TRABAJO FIN DE MÁSTER

Detección de defectología asociada al recubrimiento por *laser cladding* mediante visión artificial

Autor: D. Pablo F. Santander de Soto
Tutor: D. Esteban Cañibano Álvarez

Valladolid, mayo, 2021

Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

Máster en Ingeniería Industrial

MÁSTER EN INGENIERÍA INDUSTRIAL

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

TRABAJO FIN DE MÁSTER

Detección de defectología asociada al recubrimiento por *laser cladding* mediante visión artificial

Autor: D. Pablo F. Santander de Soto
Tutor: D. Esteban Cañibano Álvarez

Valladolid, mayo, 2021

RESUMEN

Disponer de un sistema de control de calidad para la detección de defectos es imprescindible en cualquier aplicación industrial para corregir a tiempo cualquier desviación detectada. Se ha identificado que los ensayos no destructivos, tales como la inspección visual o por líquidos penetrantes, son una herramienta muy útil para obtener un control de la producción, pero suponen un proceso completamente manual y que requiere de una gran experiencia por parte del operario.

En el presente documento se evalúa la posibilidad de desarrollar un sistema de detección de defectos asociados al recubrimiento de piezas por *laser cladding* mediante visión artificial y *Deep Learning*. A través de este sistema, se lograría automatizar la detección de defectos con la consiguiente mejora en tiempo y coste. En primer lugar, se realiza un estudio de los temas relevantes del proyecto: el proceso de recubrimiento por *laser cladding*, las distintas técnicas de ensayos no destructivos y los campos de visión artificial y *Deep Learning*. Posteriormente, se proponen y desarrollan dos vías distintas para solventar esta problemática: realizar la detección y evaluación de los defectos a través de la previa aplicación del ensayo por líquidos penetrantes, lo que permite obtener el volumen de los defectos o prescindir del ensayo y aplicar directamente el algoritmo en las piezas a inspeccionar, obteniendo así sus dimensiones. Se plantean distintas alternativas para cada una de las propuestas, evaluando los resultados más relevantes.

Palabras clave: *Laser cladding*, líquidos penetrantes, defectos, visión artificial, *Deep Learning*

ABSTRACT

Having a quality control system for defect detection is essential in any industrial application to correct any detected deviations in a timely manner. Non-destructive testing, such as visual or dye penetrant inspection, has been identified as a very useful tool for production control, but it is a completely manual process and requires a great deal of experience on the part of the operator.

This paper evaluates the possibility of developing a system for detecting defects associated with the coating of parts by laser cladding using computer vision and Deep Learning. Through this system, the detection of defects could be automated with the consequent improvement in time and cost. Firstly, a study is carried out on the relevant topics of the project: the laser cladding process, the different non-destructive testing techniques and the fields of computer vision and Deep Learning. Subsequently, two different ways to solve this problem are proposed and developed: to carry out the detection and evaluation of the defects through the prior application of the dye penetrant test, which allows the volume of the defects to be obtained or to avoid the test and apply the algorithm directly to the parts to be inspected, thus obtaining their dimensions. Different alternatives are presented for each of the proposals, evaluating the most relevant results.

Keywords: Laser cladding, dye penetrant, defects, computer vision, Deep Learning

AGRADECIMIENTOS

A Esteban Cañibano Álvarez por permitirme colaborar de nuevo con la Fundación Cidaut para la realización de este TFM pese a las circunstancias.

A Arturo González Arnáiz por dedicarme su tiempo durante estos meses y ayudarme con la realización de este proyecto.

A mis compañeros del máster con los que tan buenos momentos he pasado durante estos dos años, incluso a pesar de la pandemia.

A mis amigos, con los que disfruto cada momento y siempre estarán ahí.

A mis padres y a mi hermana, por apoyarme siempre en mis decisiones y ayudarme a ser quien soy.

Gracias.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1: INTRODUCCIÓN

En este capítulo se sentarán las bases sobre las que se sustenta el trabajo desarrollado en el presente proyecto de fin de máster, poniéndolo en situación y contexto. Además, se plantearán una serie de objetivos que ayudarán a determinar lo que se busca una vez terminado el trabajo. Asimismo, se incluye una breve descripción de los capítulos de los que consta el documento.

1.1 CONTEXTO

Actualmente la industria del automóvil busca desarrollar vehículos que garanticen la sostenibilidad futura reduciendo el consumo de combustibles y, por tanto, disminuyendo la emisión de gases invernadero a la atmósfera.

La optimización de la estructura del vehículo, utilizando cada vez materiales más ligeros y técnicas de fabricación avanzadas, busca lograr estos objetivos. En los últimos años, la técnica de estampación en caliente ha crecido en gran volumen gracias a que permite obtener componentes de chapa metálica con una resistencia mecánica muy superior a los obtenidos por procesos de estampación convencionales [1].

La estampación en caliente es un proceso dual basado en la conformación de chapas de acero y el tratamiento térmico por templado de las mismas. El resultado de la estampación supone la obtención de una chapa cuya tensión última a tracción puede llegar a doblar a la de los aceros de alta resistencia habituales. Esto permite la reducción de los espesores, logrando reducir el peso de las piezas.

Para realizar el proceso de estampación se requieren unos útiles llamados troqueles de estampación, que suelen venir asociados a una gran complejidad como consecuencia de los requisitos impuestos por el diseño y fabricación. Estos troqueles deben hacer frente a unos altos requerimientos térmicos, dimensionales y mecánicos durante el proceso de conformado, por lo que la selección del material del troquel es vital.

Todo útil se ve desgastado debido al uso, y está altamente relacionado con el proceso de fabricación. Mientras que algunas herramientas son fácilmente sustituibles o reparables, los troqueles son unos útiles de muy alto coste y con un complejo diseño, por lo que prolongar su vida útil lo máximo posible se ha identificado como un factor crítico para garantizar la rentabilidad derivada de su fabricación.

Existen multitud de técnicas que permiten aumentar la dureza superficial de los troqueles, sin modificar las características del material. La tecnología de modificación superficial supone una alternativa muy interesante que permite mejorar las propiedades superficiales de las piezas sin alterar sus funciones estructurales. Dentro de esta tecnología destaca el recubrimiento por ***laser cladding***, que permite obtener revestimientos metálicos que logran hacer frente a los requerimientos del troquel.

Estos procesos no están libres de la aparición de defectos, por lo que es necesario mantener un control de calidad que permita evaluar e identificar dichos defectos. Una de las técnicas más extendidas es el uso de **ensayos no destructivos**, que permiten detectar y evaluar los defectos sin producir daños que afecten a la funcionalidad de la pieza.

Estos ensayos permiten detectar discontinuidades superficiales o internas, según sea el método escogido para su aplicación. Durante la realización del proceso de recubrimiento, los defectos más críticos se generan en la superficie, por lo que es vital disponer del método de inspección adecuado para su evaluación. Un método muy destacado para la

detección de defectos superficiales es el **ensayo por líquidos penetrantes**, aplicable a multitud de tipos de superficie, que permite obtener resultados muy ilustrativos gracias a las indicaciones proporcionadas. Aunque se utilicen técnicas específicas para realizar la evaluación de la calidad, la **inspección visual** se mantiene presente en cualquiera de ellas, siendo el método más sencillo de realizar.

Para interpretar correctamente las indicaciones proporcionadas, es necesario disponer de personal altamente cualificado debido al gran conocimiento requerido acerca de los posibles defectos encontrados. Mediante el uso de tecnologías más avanzadas como la **visión artificial** y el ***Deep Learning***, se puede facilitar el trabajo realizado en estas inspecciones manuales a través de la utilización de sistemas de cámaras y algoritmos de detección de defectos, que permiten desarrollar la automatización de los sistemas de inspección.

1.2 MOTIVACIÓN

La **Fundación Cidaut** dentro de una de sus líneas de investigación, apuesta por el desarrollo de soluciones alternativas e innovadoras, tanto de nuevos materiales, como del diseño de productos y procesos. Esto abre numerosas puertas que buscan adquirir el conocimiento tanto de distintos procesos de fabricación, como de técnicas para evaluar la calidad de los nuevos desarrollos.

Enmarcándose en este pensamiento, este TFM surge del problema que plantea la detección de defectos en piezas con recubrimiento por *laser cladding*. Las técnicas actuales pasan por aplicar algún tipo de ensayo no destructivo, como la realización de una inspección mediante líquidos penetrantes además de una inspección visual a cada una de las piezas, lo que en la mayoría de los casos requiere una gran experiencia por parte del operario.

A través del presente proyecto, se propone la posibilidad de desarrollar un algoritmo de visión artificial que permita de detectar estos defectos, con el objetivo de facilitar la inspección de las piezas.

1.3 OBJETIVOS

El principal objetivo de este proyecto es evaluar la posibilidad de aplicar las tecnologías de visión artificial y *Deep Learning* en un problema tan específico como es la detección de defectos, a través del desarrollo de un algoritmo diseñado para tal fin.

Además, el carácter investigativo del proyecto permite generar una base de conocimiento en este campo, y dotar de una poderosa herramienta que permitirá abordar otra serie de problemas directamente relacionados.

Para alcanzar este objetivo principal, se han planteado una serie de hitos secundarios, que permitirán avanzar en el desarrollo del proyecto:

- Determinar la relación entre los defectos y el proceso de recubrimiento por *laser cladding*.
- Estudiar la metodología de inspección mediante ensayos no destructivos, en especial el ensayo por líquidos penetrantes.
- Entender los conceptos clave de los campos de la visión artificial y el *Deep Learning*, que sientan las bases del desarrollo de este proyecto.

- Desarrollar y valorar las posibles alternativas de diseño de los algoritmos de detección de defectos.
- Evaluar el rendimiento de los algoritmos propuestos, analizando sus posibles limitaciones.

1.4 DESCRIPCIÓN DEL PROYECTO

Globalmente, el presente documento se ha estructurado en siete capítulos tal y como se muestra a continuación.

En este primer apartado, el CAPÍTULO 1: INTRODUCCIÓN, se ha contextualizado el marco en el que se encuadra el proyecto y se han planteado los principales objetivos a cumplimentar en los capítulos posteriores.

En el CAPÍTULO 2: ESTADO DE LA TÉCNICA se realiza un análisis de los campos que intervienen en la detección de defectos:

- En primer lugar, se realiza una breve introducción a los tratamientos superficiales, prestando mayor atención a la tecnología de recubrimiento por *laser cladding*, puesto que es la utilizada en las piezas de este proyecto.
- Posteriormente, se describe la metodología de inspección mediante ensayos no destructivos, especialmente la inspección visual y el ensayo por líquidos penetrantes.
- Por último, se desarrollan los temas de visión artificial y *Deep Learning*, describiendo ambos campos y distintas aplicaciones de detección de defectos abordadas mediante este tipo de tecnología.

En el CAPÍTULO 3: INTRODUCCIÓN AL DESARROLLO TÉCNICO se introduce la parte más técnica del proyecto, describiendo los recursos empleados y desarrollando brevemente la programación con Python y OpenCV.

El CAPÍTULO 4: DETECCIÓN DE DEFECTOS CON ENSAYO PREVIO recoge la primera alternativa seguida en el desarrollo del proyecto, que hace referencia a la utilización del ensayo de líquidos penetrantes de forma previa a la aplicación del algoritmo de visión artificial.

En el CAPÍTULO 5: DETECCIÓN DE DEFECTOS EN BRUTO se propone y evalúa la posibilidad de aplicar los algoritmos de visión artificial prescindiendo de la realización del ensayo por líquidos penetrantes.

El CAPÍTULO 6: APLICACIÓN DE DEEP LEARNING EN LA DETECCIÓN sirve como base de la aplicación del campo del *Deep Learning* en la problemática de la detección de defectos, evaluando el diseño de un clasificador como alternativa al algoritmo presentado en el capítulo 4.

Por último, en el CAPÍTULO 7: CONCLUSIONES Y LÍNEAS FUTURAS se desarrollan las conclusiones a las que se ha llegado durante la realización del proyecto y se plantean posibles líneas de mejora.

CAPÍTULO 2: ESTADO DE LA TÉCNICA

Este capítulo busca profundizar sobre los aspectos más técnicos de las áreas y tecnologías más determinantes para el desarrollo de este proyecto. Además de realizar la descripción del recubrimiento por *laser cladding*, se procede a la introducción de los ensayos no destructivos, haciendo particular hincapié sobre el método de inspección visual y al ensayo por líquidos penetrantes. En los últimos apartados se introduce al lector los campos de la visión artificial y el *Deep Learning*, estrechamente relacionados con el proyecto, junto con aplicaciones de este tipo de tecnología en la detección de defectos.

2.1 LASER CLADDING

El término **ingeniería de superficies** engloba una gran variedad de tecnologías, en ocasiones difíciles de clasificar, pero que se pueden agrupar en dos tecnologías diferentes:

- Tratamientos superficiales: incluyen técnicas que modifican las propiedades de la superficie de la pieza.
- Recubrimientos superficiales: hacen referencia a aquellas tecnologías en las que un material se deposita sobre la superficie para mejorar sus propiedades.

En la Figura 1 se recoge un esquema de la gran variedad de tecnologías englobadas en el contexto de la ingeniería de superficies. Se ha mantenido la imagen original debido a que muchas técnicas son más conocidas por sus términos en inglés.

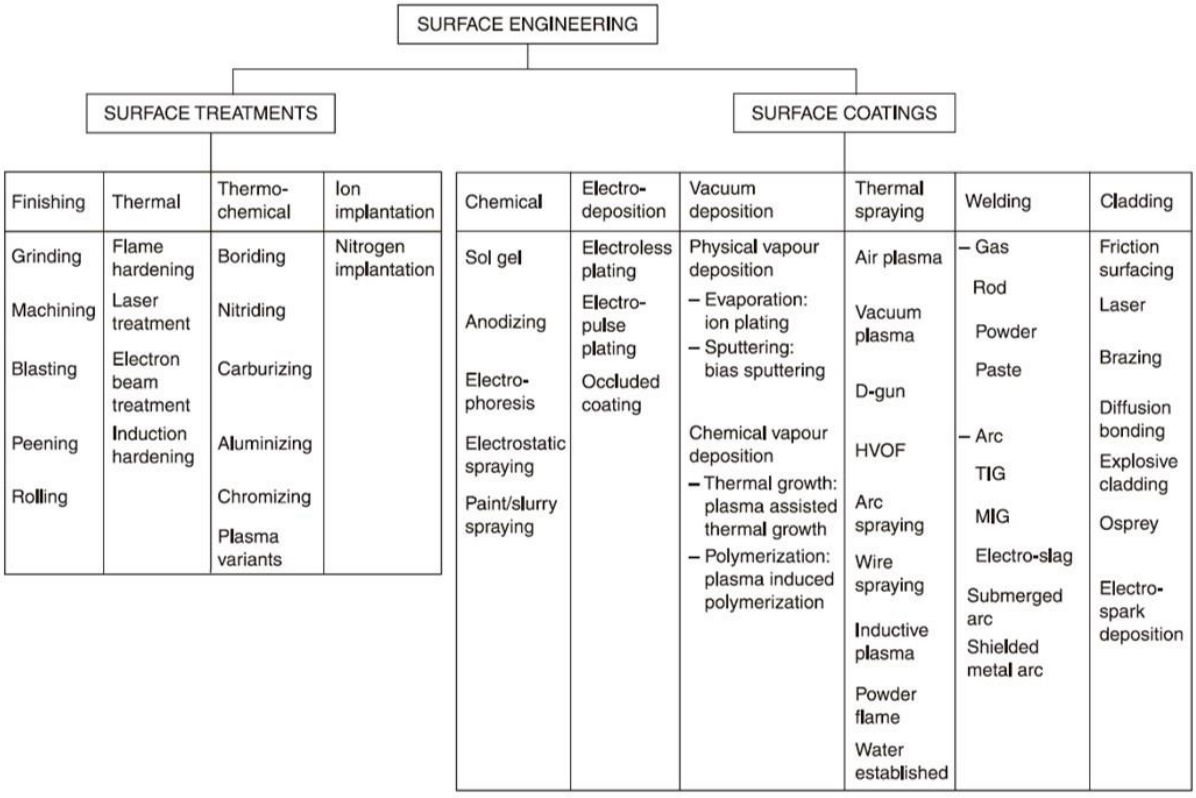


Figura 1. Clasificación de las tecnologías de ingeniería de superficies (Fuente: [2])

La aplicación de estas tecnologías permite prolongar la vida útil de las piezas, mejorando las propiedades superficiales de las mismas, pero sin modificar las características del material base. Esto permite obtener piezas con requerimientos térmicos y de resistencia muy específicos.

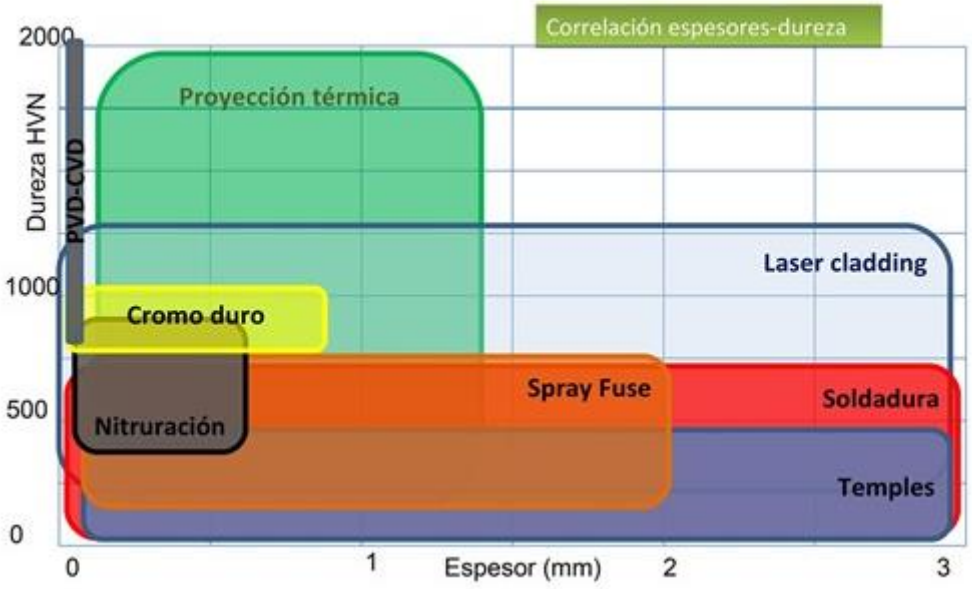


Figura 2. Rangos de espesor y durezas más habituales según el método superficial aplicado

Gracias a las mejoras en las piezas proporcionadas por estas técnicas, multitud de sectores industriales se ven beneficiados mediante su aplicación: automoción, mecánico, aeronáutico, construcción, químico, electrónico, alimentación, etc. [3]. En cuanto al caso específico de aplicación en troqueles de estampación destacan las siguientes técnicas: **tratamientos térmicos** (*termal treatments*), **nitruración y deposición física de vapor** (*nitriding and physical vapour deposition*), **proyección térmica** (*thermal spraying*) y **revestimiento por láser** (*laser cladding*).

En la Figura 2 se realiza una comparativa entre distintas técnicas utilizadas a nivel industrial para aumentar la dureza superficial de las piezas. Se puede observar que el *laser cladding* se presenta como la mejor alternativa en cuanto a la relación espesor-dureza alcanzada, permitiendo obtener valores muy altos de dureza en gran variedad de espesores dentro del rango establecido.

2.1.1 Definición del proceso

El **revestimiento por láser**, más conocido como ***laser cladding***, es una técnica de aporte de material que consiste en la fusión directa, mediante un haz láser, de un polvo de origen metálico o cerámico, inyectado sobre la superficie de la pieza a tratar. A través de esta técnica, se consiguen recubrimientos de propiedades prácticamente idénticas o incluso mejoradas respecto al material base de la superficie. Un esquema del proceso seguido se puede observar en la Figura 3.

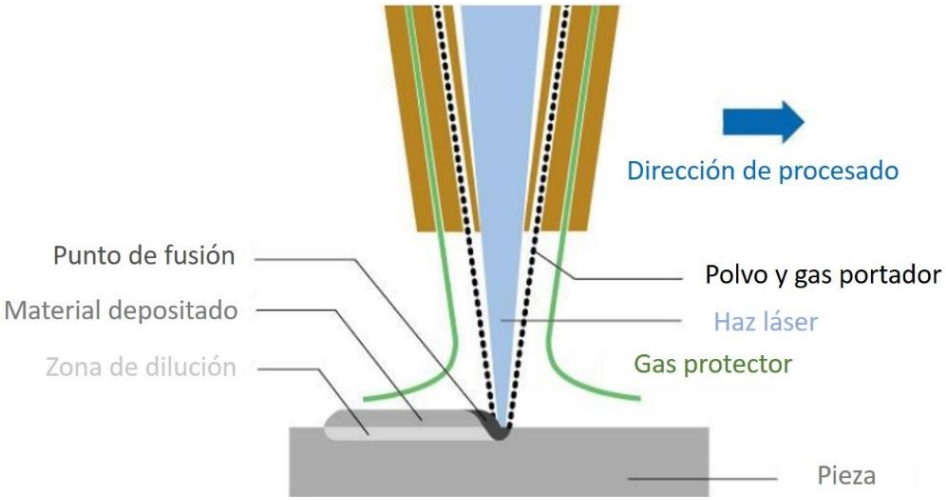


Figura 3. Esquema del proceso de laser cladding

Sus aplicaciones en la industria son múltiples, teniendo especial importancia en el sector de la automoción y aeronáutico [4] y [5]:

- Protección: es posible reforzar los componentes logrando protección frente a la corrosión y al desgaste, incluso logrando que el recubrimiento actúe como una barrera térmica.
- Restauración y reparación de piezas: se pueden recuperar partes desgastadas de manera más precisa que con otros procesos (TIG, MIG, etc.).
- Fabricación aditiva de piezas: gracias a la deposición del material es posible fabricar componentes funcionales y estructuras 3D mediante esta técnica.

En comparación con otros tipos de procesos de la misma familia, el *laser cladding* permite un mayor control, sin embargo, el equipo requerido para su aplicación encarece la tecnología. Esta técnica permite obtener una unión metalúrgica entre el material base y el material de aporte, creando una superficie homogénea. En la Figura 4 se puede observar un proceso real de aplicación de esta técnica.

Figura 4. Proceso de laser cladding (Fuente: [2])

Gracias a los grandes avances en los sistemas de automatización, la aplicación del *laser cladding* ha cobrado gran importancia, colocándose como una perfecta alternativa a métodos tradicionales de endurecimiento superficial, como puede ser la nitruración o la electrodeposición [6]. Esto ha provocado que este proceso se identifique como una opción

que se debe tener en cuenta a la hora de generar recubrimientos en la industria matricera, como puede ser en la fabricación de troqueles de estampación.

Ejemplos de aplicación del proceso en restauración y reparación de piezas se pueden encontrar en multitud de estudios. Por su parte, Łukasz & Marta [7] analizan el acabado de las superficies después de realizar un proceso de restauración mediante esta técnica, evaluando la rugosidad superficial obtenida después de realizar varios procesos de regeneración de la superficie, combinando la técnica de *laser cladding* con procesos de mecanizado.

Durante el proceso de aporte de material se deben controlar una serie de factores y parámetros que afectan al resultado final del recubrimiento, cobrando gran importancia la elección de las trayectorias adecuadas con el objetivo de reducir el posible efecto térmico en las características de la pieza. También es importante definir de forma correcta la distancia entre el cabezal y la superficie a tratar, controlando así la energía aportada por el haz láser.

La elección del material de aporte depende de las propiedades finales que se busca obtener con el recubrimiento, tales como la resistencia al desgaste o a la corrosión. El polvo metálico utilizado debe de ser de alta calidad formado por partículas sólidas esféricas que permitan obtener un flujo constante en la alimentación. En última instancia, los parámetros del proceso (distancia del cabezal, velocidad de avance, potencia del láser, etc.) determinarán la tasa de aporte de material.

Existen multitud de estudios que señalan la importancia de una correcta selección de los parámetros para minimizar la aparición de defectos. Por ejemplo, Rombouts et al. [8] analizan el acabado superficial de las piezas obtenidas a través de LMD (*laser metal deposition*). La Figura 5 recoge un resumen de los resultados obtenidos por los autores, analizando la rugosidad superficial tras realizar la mejora de la superficie mediante *laser cladding* con distintos parámetros del proceso. Durante el estudio también se evalúa el efecto del ángulo durante el aporte de material.

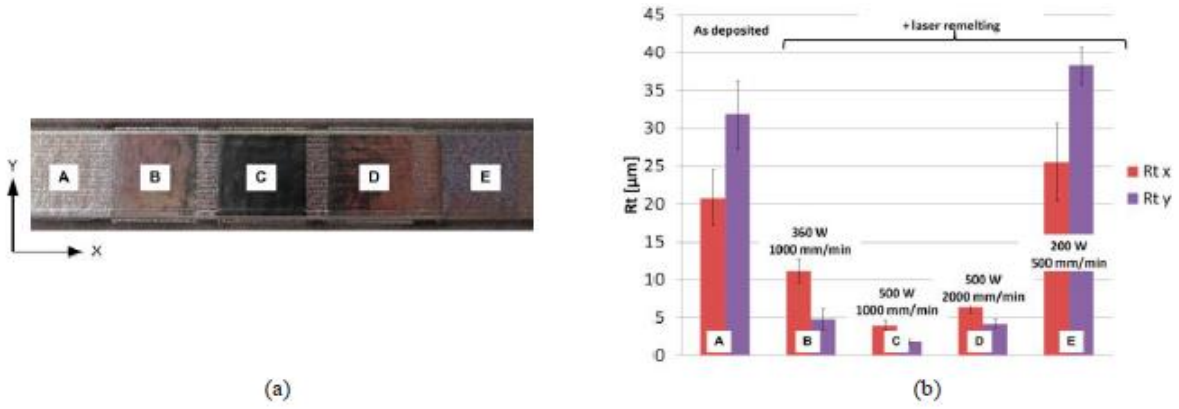


Figura 5. Efecto de las velocidades de avance y potencia del láser en el ensayo (Fuente: [8])

Candel et al. [9] caracterizan distintos cordones de 30 mm de longitud variando las condiciones de potencia, velocidad de avance y caudal de polvo. El estudio realizado consiste principalmente en la realización de un análisis microestructural de los cordones mediante el corte transversal de las muestras y su posterior caracterización. En la Figura 6 se observa cómo afecta la reducción de la potencia del haz y la velocidad de avance en la eliminación de la aparición de grietas y poros para el caso estudiado.

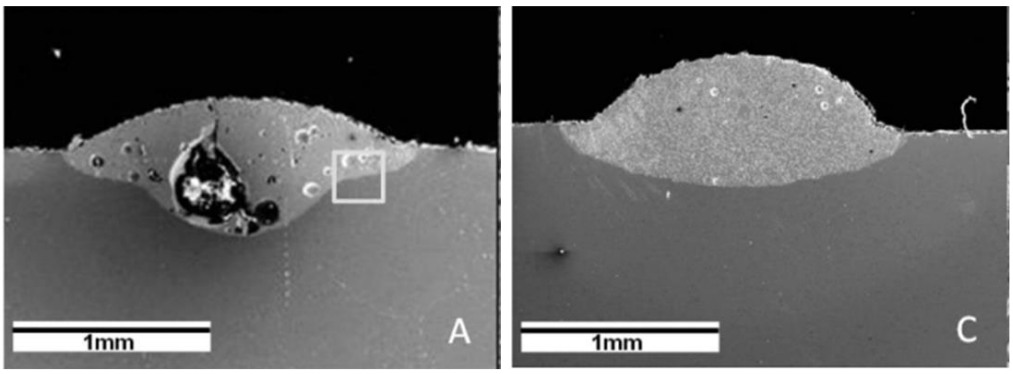


Figura 6. Efecto producido por la reducción de la potencia P y la velocidad de avance V. P = 700 W, V = 500 mm/min (izquierda), P = 600 W, V = 400 mm/min (derecha) (Fuente: [9])

2.1.2 Defectos asociados al proceso

Debido a las características intrínsecas del proceso y a la intervención de múltiples parámetros que deben ser controlados durante su aplicación, las superficies generadas suelen presentar determinados problemas que deben ser tenidos en consideración. Por ejemplo, las elevadas temperaturas que alcanza el sustrato pueden provocar la aparición de porosidad en la superficie. Mientras que los parámetros relacionados con la configuración del haz láser pueden afectar además a la aparición de grietas [10].

En el estudio realizado por Liu et al. [5], en el que demuestran la posibilidad de utilizar el proceso de *laser cladding* para reparar componentes aeronáuticos, mejorando su resistencia frente al desgaste y a la corrosión, se observó algunos de estos defectos, como la aparición de grietas en el revestimiento generado (Figura 7).

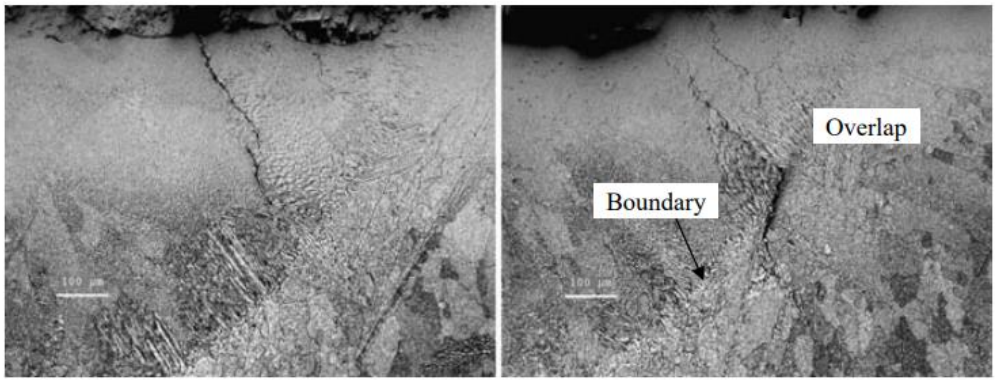


Figura 7. Ejemplos de grietas en el revestimiento (Fuente: [5])

Haldar & Saha [11] realizan un análisis exhaustivo respecto a los problemas derivados de esta tecnología, proponiendo algunas soluciones para tratar de eliminarlos durante el proceso. Los defectos más habituales suelen estar muy vinculados a las distintas capas de material generadas por el recubrimiento. Un esquema de este hecho se ilustra en la Figura 8.

Aunque podrían estudiarse de forma más exhaustiva, principalmente los defectos asociados al recubrimiento se pueden dividir en tres categorías:

- Problemas de unión: una inadecuada fusión del sustrato debido a un aporte incorrecto de energía puede provocar una incorrecta adherencia con el material base.

- Porosidad: los poros se forman debido al atrapamiento de burbujas de gas cuando el material solidifica. Este problema es muy común en cualquier proceso que implique aporte de material fundido.
- Fisuras y grietas: la rápida solidificación del sustrato y, los elevados gradientes de temperaturas existentes entre cordones, puede provocar la aparición de tensiones internas residuales, lo que se traduce en la aparición de grietas. Generalmente este es uno de los mayores problemas del proceso.

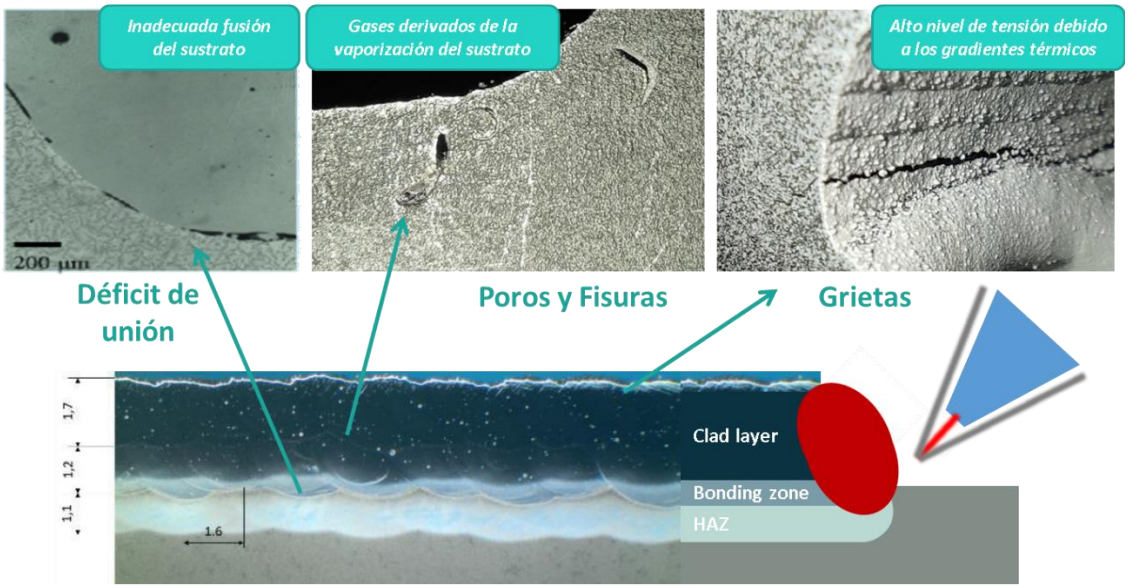


Figura 8. Tipología usual de defectos en los recubrimientos por laser cladding

Una excesiva acumulación de defectos en la superficie (o en capas cercanas a ella), puede provocar la pérdida de funcionalidad de las piezas. En el caso concreto de este proyecto, los troqueles pueden deteriorarse rápidamente durante el proceso de estampación debido al efecto de los distintos esfuerzos a los que son sometidos. Esto genera, como ya se ha mencionado anteriormente, la necesidad de mantener un exhaustivo proceso de inspección que permita detectar estos defectos.

2.2 ENSAYOS NO DESTRUCTIVOS

La necesidad de examinar distintos materiales y piezas de forma que no se altere ni limite su posterior uso manteniendo su forma original inalterada se remonta a la antigüedad, desde la detección de alimentos en mal estado por su olor en la prehistoria hasta la aparición de los ensayos por rayos X y ultrasonidos entre los siglos XIX y XX. Durante las últimas décadas se han ido desarrollando métodos cada vez más sofisticados lo que ha requerido una regulación de la disciplina.

La correcta realización y aplicación de estos ensayos, además de la adecuada interpretación de los resultados, requiere unos conocimientos técnicos muy específicos y una alta experiencia en el campo, por lo que la cualificación del personal que los realiza es decisiva. Actualmente existe un amplio marco normativo y la posibilidad de realizar cursos formativos de certificación que permiten acreditar a los profesionales.

España cuenta con la *"Asociación Española de Ensayos No Destructivos"* que cuenta con una amplia gama de recursos para la formación y certificación de profesionales en este ámbito. En uno de sus documentos, "*Introducción a los END"* [12], se presenta una visión global de los métodos más utilizados en la industria.

2.2.1 Definición

¿Qué se entiende por ensayos no destructivos? Consisten en la utilización de técnicas no invasivas realizadas a un objeto, con el fin de determinar su integridad, analizando la posible presencia de discontinuidades que afecten a la utilizad del objeto. Es decir, consiste en examinar algo sin destruirlo ni dañarlo en ningún punto de la prueba. Comúnmente se les denomina por las siglas END.

En general es posible aplicar estos ensayos en cualquier punto del proceso productivo y a cualquier componente.

Existen tres amplias áreas que agrupan la variedad de aplicaciones que pueden tener estos ensayos:

1) Defectología: permiten detectar y caracterizar heterogeneidades, discontinuidades, etc. Ello permite ahorrar costes y tomar las medidas oportunas para corregir la aparición de dichos defectos.
2) Caracterización de los materiales: permiten, en ciertos casos, evaluar las propiedades de los materiales, como pueden ser características químicas, estructurales o mecánicas.
3) Metrología: es posible garantizar que los equipos están completos y dimensionados de forma correcta a través del control dimensional de las piezas.

A diferencia de los ensayos destructivos, en los que la inspección se realiza por muestreo, los ensayos no destructivos permiten el ensayo de toda la producción, asegurando la calidad y fiabilidad del producto. Su utilización permite, entre otras cosas:

- Asegurar la integridad y fiabilidad de los productos
- Prevenir accidentes
- Garantizar la satisfacción del cliente
- Ayudar en el diseño del producto
- Controlar los procesos de fabricación y disminuir costos
- Mantener un nivel de calidad

Gracias a su gran variedad de usos y aplicaciones, estos ensayos se utilizan en multitud de sectores distintos: aeronáutico, automoción, construcción, ferroviario, metalúrgico, naval, plantas industriales, etc., siendo una parte clave en determinados procesos dada la necesidad de asegurar la integridad de elementos, por ejemplo, de aviones, trenes o puentes. En la Figura 9 se pueden observar algunos ejemplos reales de aplicación de este tipo de ensayos.

La correcta realización de estos ensayos es crítica. En un estudio realizado por McEvily [13] se recogen determinados casos en los que una incorrecta selección o realización del ensayo llegaron a provocar graves accidentes, debido a que no fueron capaces de detectar los defectos asociados.

Figura 9. Ejemplos de aplicación de ensayos no destructivos

2.2.2 Etapas

De forma general, pueden concretarse cuatro etapas básicas de inspección de un material o elemento:

1) Elección del método y las técnicas adecuadas al ensayo: a la hora de seleccionar el ensayo es importante tener en cuenta la naturaleza del material, su estado estructural y la forma del producto a ensayar, así como conocer previamente el tipo de imperfecciones que se desean detectar, puesto que cada método posee una serie de limitaciones.
2) Obtención de las indicaciones: los ensayos siguen procedimientos indirectos para la detección de discontinuidades a través de la relación con determinadas propiedades del producto. Según sea el método elegido, la visualización de una misma discontinuidad se visualizará de forma muy distinta.
3) Interpretación de las indicaciones: a raíz de los resultados obtenidos del ensayo es importante interpretarlos y sacar las conclusiones oportunas. En esta etapa la experiencia previa juega un papel fundamental.
4) Evaluación: una vez realizada la interpretación, será preciso tomar las decisiones acerca de la aceptación o rechazo de las discontinuidades encontradas según los criterios establecidos previamente.

2.2.3 Clasificación de los ensayos no destructivos

Todos los ensayos no destructivos se basan en la determinación o variación de una propiedad física del material del objeto a analizar. Comúnmente es posible realizar una clasificación en base al alcance de cada uno de los métodos, es decir, la capacidad de detección que poseen:

a. Métodos superficiales: detectan discontinuidades abiertas a la superficie.
b. Métodos subsuperficiales: detectan discontinuidades cercanas a la superficie pero que no se encuentran abiertas a la misma.
c. Métodos volumétricos: detectan discontinuidades en cualquier lugar del volumen de la pieza.

En la tabla siguiente se recogen los métodos más convencionales relacionados con el fenómeno físico que utilizan.

Tabla 1. Clasificación de los métodos de ensayos no destructivos

	MÉTODO	FENÓMENO FÍSICO
Métodos superficiales	Inspección visual	Luz visible
	Líquidos penetrantes	Capilaridad
Métodos subsuperficiales	Partículas magnéticas	Campos magnéticos
	Corrientes inducidas	Corriente eléctrica de Foucault
	Termografía infrarroja	Radiación infrarroja
Métodos volumétricos	Ensayo de fugas	Presión
	Emisión acústica	Ondas mecánicas elásticas
	Radiografía industrial	Radiaciones ionizantes
	Ultrasonidos	Ondas mecánicas elásticas

Dadas las características de los troqueles recubiertos por *laser cladding* (elevada rugosidad superficial, defectos superficiales, geometría irregular…), los ensayos más apropiados para determinar la defectología asociada al proceso son los correspondientes a los métodos superficiales: inspección visual e inspección por líquidos penetrantes.

Puesto que son los que tienen más relevancia dentro del proyecto, en los apartados siguientes se describen estos dos métodos.

2.2.4 Inspección visual

Pese a que la **inspección visual** es un método ampliamente utilizado tanto dentro como fuera de la industria para todo tipo de tareas, no debe ser menospreciado ya que, con los instrumentos ópticos adecuados, como lupas o endoscopios, posibilita la identificación y reporte de defectos de forma sencilla.

Siempre está presente en cualquiera de los otros métodos debido a que se inspecciona el elemento antes de ensayar, permitiendo rechazar de forma directa aquellos indudablemente defectuosos y reconocer imperfecciones que no son significativas, además de realizar el análisis de la información obtenida de los ensayos de forma visual.

Para realizar una inspección visual es importante tener una gran experiencia y estar familiarizado con la pieza, puesto que hay que saber qué es lo que se busca durante la inspección. Sus características más comunes se recogen en la Tabla 2.

Tabla 2. Ventajas e inconvenientes del ensayo de inspección visual

VENTAJAS	INCONVENIENTES
• Muy accesible • Relativamente sencillo • Aplicable a cualquier material • Permite la localización precisa de discontinuidades	• Solo detecta discontinuidades superficiales • Requiere operadores bien entrenados • No permite determinar la profundidad de la discontinuidad

Estos métodos de inspección están en continua mejora gracias al uso de tecnologías como la visión artificial o el *Machine Learning*, que permiten diseñar procesos más avanzados de inspección fácilmente automatizables. La utilización de sistemas de captura de imágenes, que posteriormente son tratadas por algoritmos diseñados para el reconocimiento de defectos, supone el desarrollo lógico de los sistemas manuales, facilitando el trabajo realizado por los inspectores.

En posteriores apartados se discutirá más en profundidad este tipo de tecnología junto con aplicaciones reales de su utilización.

2.2.5 Inspección por líquidos penetrantes

La **inspección** o el **ensayo de líquidos penetrantes** [14] es un método superficial basado en la utilización de un líquido tintado, ya sea coloreado o fluorescente, cuya capilaridad le permite penetrar en las discontinuidades abiertas a la superficie. Una vez limpiado el exceso de líquido de la superficie quedará solamente el que haya en las discontinuidades y con la ayuda de un agente revelador se exponen las zonas en las que existen dichas discontinuidades.

Gracias a su gran versatilidad es ampliamente empleado en todo tipo de industrias, sin apenas restricciones de tipos de materiales o de forma de las piezas. La única condición es que la superficie no sea porosa. Las principales características de este ensayo vienen recogidas en la Tabla 3.

Conviene destacar los diferentes objetivos que puede tener realizar una inspección por líquidos penetrantes según sea la industria o la aplicación. Por ejemplo, en la industria de la construcción es usual analizar el estado del hormigón para detectar posibles grietas que hayan aparecido durante su servicio [15] obteniendo así un certificado acerca del estado del material.

Para el caso de la industria de la automoción lo más común es obtener un informe de defectos encontrados en las piezas. Desde hace años se han buscado métodos para automatizar estas inspecciones en automoción [16].

Tabla 3. Ventajas e inconvenientes del ensayo de líquidos penetrantes

VENTAJAS	INCONVENIENTES
• Rápido • Fácil aplicación • Muy sensible • Muy portátil • Aplicable a casi cualquier material	• Solo detecta discontinuidades superficiales • Existe riesgo de contaminación • No permite determinar la profundidad de la discontinuidad • No se puede utilizar en materiales porosos

Dependiendo de la sensibilidad que se requiera en el ensayo, existen múltiples técnicas de inspección. En la Tabla 4 se recogen los distintos productos que se pueden utilizar en el ensayo, distinguiendo así las técnicas existentes.

Tabla 4. Clasificación de los productos usados en el ensayo de líquidos penetrantes según ISO 3452-1 (Fuente: [12])

Penetrante		Eliminador del exceso de penetrante		Revelador	
Tipo	**Denominación**	**Método**	**Denominación**	**Forma**	**Denominación**
I	Penetrante fluorescente	A	Lavable con agua	a	Seco
II	Contraste de color	B	Post-emulsificable, lipofílico	b	Soluble en agua
III	Penetrante Mixto (penetrante fluorescente y coloreado)	C	Eliminable con disolvente – Clase 1, halogenado – Clase 2, no halogenado – Clase 3, aplicaciones especiales	c	Suspensión en agua
		D	Post- emulsificante, hidrofílico	d	Base-disolvente (no-acuoso para tipo I)
		E	Eliminable con agua y disolvente	e	Base-disolvente (no-acuoso para tipos II y III)
				f	Aplicaciones especiales
En casos específicos, es necesario usar un producto de ensayo penetrante que responda a ciertos requisitos en lo que concierne a inflamabilidad, contenido de azufre, halógenos, sodio y otros contaminantes, véase la Norma ISO 3452-2.					

Según sea la técnica elegida, será necesario utilizar equipos auxiliares para la realización del ensayo. Por ejemplo, el ensayo con penetrantes coloreados eliminables con disolvente se necesita verificar la luz blanca disponible mediante luxómetros. En el caso de utilizar penetrantes fluorescentes, que disponen de mayor sensibilidad, se requerirá de lámparas de luz ultravioleta.

En general es un método sencillo y que no requiere de equipos costosos, pero debe de hacerse con sumo cuidado para obtener los mejores resultados. El ensayo puede dividirse en varias etapas, ilustradas en la Figura 10:

a) Preparación y limpieza de la superficie a ensayar
b) Secado de la superficie
c) Aplicación del líquido penetrante
d) Eliminación del exceso de penetrante en la superficie
e) Aplicación del relevador
f) Observación de defectos

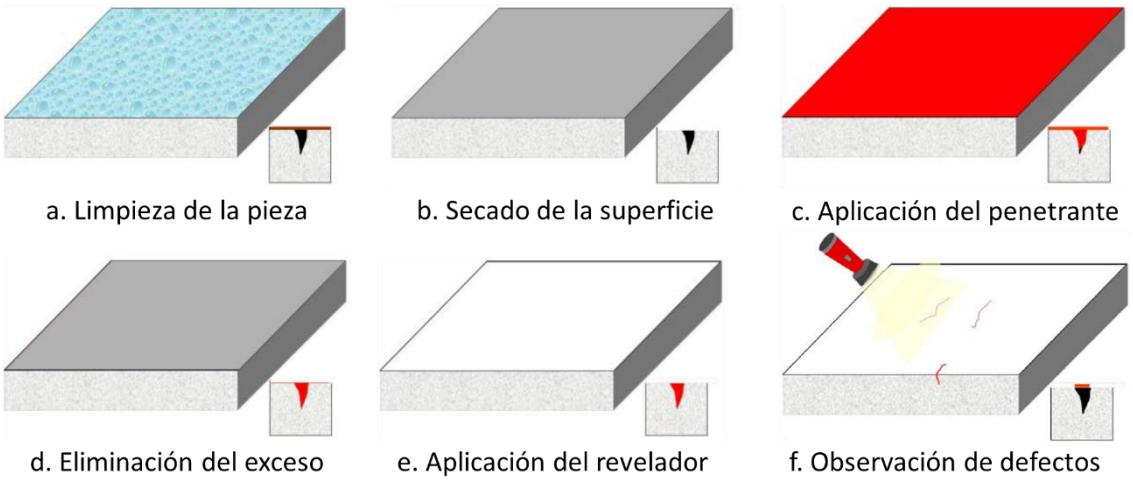


Figura 10. Etapas del ensayo de líquidos penetrantes (Fuente: [12])

2.2.5.1 Preparación y limpieza de la superficie

Se debe eliminar cualquier posible contaminante de la superficie de la pieza antes de comenzar con en el ensayo. Por ejemplo, la presencia de restos de aceites, pinturas o barnices pueden impedir que el penetrante no alcance las discontinuidades. Por tanto, la elección de la técnica de limpieza dependerá del contaminante a eliminar:

- Limpieza con detergentes: tienen que ser no corrosivos para no dañar la superficie. Generalmente se emplea para la limpieza de pequeñas piezas sumergidas en tanques.
- Limpieza con disolvente: puede aplicarse directamente con un trapo o sumergiendo la pieza en tanques.
- Limpieza alcalina: similar a los detergentes, pero seleccionados para eliminar varios tipos de suciedad, siendo capaces incluso de eliminar óxidos. Es muy importante seguir las instrucciones del fabricante.
- Limpieza con vapor desengrasante: eficaz para la eliminación de aceites, grasas y otros contaminantes orgánicos.
- Limpieza por ultrasonidos: se aplica junto a la limpieza alcalina o con disolvente en baño para reducir el tiempo de limpieza.
- Limpieza con decapantes: eliminan la pintura de la superficie.
- Limpieza por medios mecánicos: empleo de cepillos de alambre, chorro de arena, granallado, rascado u otros métodos. Se utilizan en condiciones muy específicas puesto que pueden llegar a manipular la superficie ocultando determinadas discontinuidades.
- Limpieza por ataque ácido: elimina metales que ocultan las discontinuidades e impiden el paso del líquido penetrante, volviéndolas a abrir a la superficie.

2.2.5.2 Secado de la superficie

Una vez limpiada la superficie es importante que se eliminen todos los restos de detergentes, disolventes o decapantes utilizados. Además, se debe secar completamente la pieza para evitar que la presencia de cualquier tipo de líquido impida la correcta aplicación del penetrante.

Existen multitud de opciones para secar las piezas, desde la utilización de hornos de secado o dejándolas secar a temperatura ambiente en condiciones controladas.

2.2.5.3 Aplicación del líquido penetrante

La aplicación del líquido penetrante se puede realizar de varias formas, según sea el tamaño y el número de piezas que se deben inspeccionar, desde pulverización con aerosoles o pistolas hasta inmersión de las piezas en tanques.

Generalmente, la elección del tipo de penetrante se realizará en función de la forma de visualización de discontinuidades deseada. Como se mencionó en la Tabla 4, existen tres tipos de líquidos penetrantes:

1) Penetrantes coloreados: contienen determinados pigmentos que los hacen visibles a luz natural o luz artificial blanca.

2) Penetrantes fluorescentes: se necesita utilizar luz negra o ultravioleta para su visualización.
3) Penetrantes fluorescentes-coloreados o mixtos: mezclan los dos.

Tan importante es la elección del líquido a utilizar, como el tiempo que se debe dejar para que actúe y penetre en las discontinuidades, conocido como tiempo de penetración. Este tiempo se establece según el material de la pieza a ensayar, el tipo de discontinuidad y otras recomendaciones del fabricante, estando comprendido entre cinco minutos y una hora.

Un buen penetrante debe poseer una serie de características generales para que el ensayo se pueda realizar en unas condiciones aceptables. Estas características se pueden lograr a través de la mezcla de diversas sustancias puesto que no existe una sustancia única que las reúna todas:

- Capaz de introducirse en grietas muy finas.
- Capaz de permanecer en el interior de grietas muy gruesas.
- No volátil.
- Soluble.
- Capaz de permanecer en estado fluido para salir de las grietas con facilidad al aplicar el revelador.
- No ser corrosivo ni que ataque al material.
- No ser tóxico ni inflamable.
- Resistente a la pérdida de color o a la fluorescencia.
- Ser económicamente rentable.

El ensayo se basa principalmente en una serie de propiedades físicas de los líquidos, que se recogen principalmente en dos conceptos:

1) Mojabilidad o poder humectante: mide la capacidad de un líquido de mojar una determinada superficie. Se caracteriza a través del ángulo de contacto o de mojado α. Este ángulo se determina a través de la diferencia entre las fuerzas de cohesión (que mantienen unidas las moléculas del líquido) y las fuerzas de adherencia (que hacen referencia a la interfase de dos sustancias diferentes, líquido-sólido o líquido-aire).

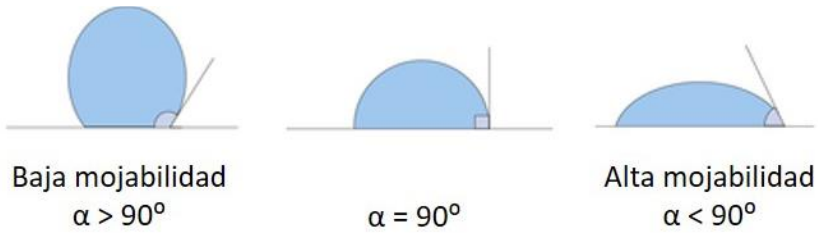


Figura 11. Niveles de mojabilidad

En el caso de los líquidos penetrantes interesa que sean capaces de mojar la superficie del material y de fluir sobre ella formando una película continua y uniforme.

2) Capilaridad o poder de penetración: mide el comportamiento de un determinado líquido al penetrar zonas estrechas. Se caracteriza mediante un tubo muy fino (tubo capilar) que se introduce en el líquido, pudiéndose dar tres situaciones distintas en función del ángulo α.

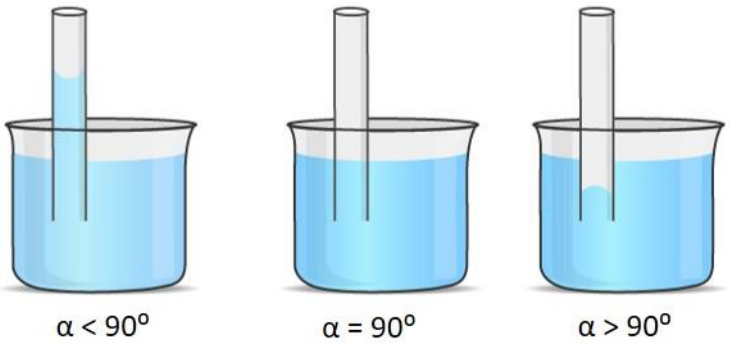


Figura 12. Niveles de capilaridad

Interesa que el líquido posea buena capilaridad otorgándole un alto poder de penetración, que le permita introducirse en las discontinuidades que no son visibles a simple vista. La viscosidad por su parte no afecta directamente en la capacidad de penetración, pero afecta a la velocidad con la que el líquido penetra en las discontinuidades [17].

2.2.5.4 Eliminación del exceso de penetrante

Aplicado el penetrante y transcurrido el tiempo requerido, se debe retirar todo el exceso para evitar falsas indicaciones durante la inspección. Es importante realizar la eliminación con cuidado para no extraer el líquido introducido en las discontinuidades.

Existen varios métodos según sea el tipo de líquido penetrante utilizado. En la Tabla 4 se recogen los cinco métodos existentes, siendo los principales:

1) Lavables con agua o auto-emulsionantes
2) Post-emulsionantes: se requiere aplicar previamente un emulsionante, siendo la mezcla del líquido y el emulsionante la que se puede eliminar con agua.
3) Eliminables con disolvente: suele existir uno específico para cada líquido penetrante particular, según lo indique el fabricante.

2.2.5.5 Aplicación del revelador

Estando la superficie preparada, es necesario aplicar el revelador para hacer el defecto visible al ojo humano. El revelador se encarga de extraer el líquido de la discontinuidad, aumentando la visibilidad de la misma gracias al color opuesto al del penetrante.

Para el caso de los reveladores existen varias formas disponibles, mostradas en la Tabla 4, destacando:

1) Secos: polvos muy finos. El yeso o el talco fueron los primeros relevadores, y aún hoy se siguen utilizando.
2) Acuosos: partículas solubles o no solubles en agua formando una suspensión acuosa.
3) No acuosos: polvos en suspensión en un disolvente orgánico volátil.

Durante la aplicación es importante obtener una capa muy fina y uniforme sobre toda la superficie. Se puede realizar espolvoreando el revelador, por inmersión o por pulverización, según sea el revelador utilizado.

Al igual que en otras etapas, la selección del método de aplicación del revelador dependerá del número y tamaño de las piezas, del acabo superficial, etc.

2.2.5.6 Observación de defectos

La última etapa del ensayo de líquidos penetrantes consiste en la inspección para la observación de las indicaciones obtenidas. En esta etapa entra el juego el tipo de iluminación necesaria según sea el tipo de líquido penetrante usado durante el ensayo.

La experiencia y conocimiento del operador es vital en esta etapa, la interpretación de las indicaciones mostradas se realiza bajo su criterio y debe establecer la causa que originó el defecto y, si se da el caso, diferenciar aquellas indicaciones que puedan resultar falsas. En relación con este apartado existen tres tipos de indicaciones:

- Falsas: no causadas por una discontinuidad.
- No relevantes: causadas por una discontinuidad de diseño, pero no por un defecto.
- Relevantes: causadas por un defecto.

En caso de encontrarse con indicaciones relevantes, se debe decidir si se aceptan o se rechazan, juzgando en función de la gravedad si deben ser reparadas o si se debe desechar la pieza por completo.

Como se ha mencionado anteriormente, en esta etapa sería posible introducir la utilización de un algoritmo basado en visión artificial, que permitiría identificar automáticamente las indicaciones presentadas por el ensayo. Además, utilizando el algoritmo adecuado, sería posible medir dichas indicaciones y obtener las dimensiones de los defectos asociados automáticamente.

2.3 VISIÓN ARTIFICIAL Y DEEP LEARNING

La inspección visual supone la metodología más utilizada para la detección de defectos debido a que todos los demás métodos requieren de ella. En ocasiones, supone un trabajo completamente manual y que requiere de una gran experiencia por parte del operador, tanto en el conocimiento de la pieza a inspeccionar como de la posible defectología asociada. Mediante técnicas más avanzadas como la visión artificial, es posible facilitar las inspecciones, siendo posible automatizar todo el proceso.

La utilización de los complejos algoritmos utilizados en este campo permite la fácil automatización de procesos de inspección de calidad basados en la visión artificial. Además, en la actualidad, el amplio desarrollo de las redes neuronales convolucionales permite diseñar algoritmos cada vez más potentes que son capaces de clasificar los distintos tipos de defectos detectados. Como se verá al final de este mismo apartado, la aplicación de estas tecnologías es muy amplia, desde el análisis de la calidad de las soldaduras [18] a través de procesos de visión artificial, hasta la clasificación de tornillos defectuosos [19] mediante el uso de redes neuronales.

Adicionalmente, sería posible combinar esta tecnología con métodos de inspección más tradicionales como el ensayo por líquidos penetrantes, lo cual permitiría facilitar la identificación de los defectos y su posterior evaluación para la toma de decisiones, siendo en esta problemática uno de los puntos donde se enfoca el presente proyecto.

2.3.1 Visión artificial

La finalidad de la **visión artificial** es extraer una determinada información de un entorno físico (y tridimensional) a partir de imágenes bidimensionales de dicho entorno. Esta información se obtiene a través del análisis de distintas propiedades, desde la forma y el tamaño de los objetos, hasta el color y el brillo de la escena. La literatura existente respecto a este campo es muy amplia y puesto que realizar un desarrollo exhaustivo no es el objetivo del presente documento, se invita al lector a acudir a la bibliografía especializada en caso de querer profundizar en la materia.

Como bibliografía básica en castellano es posible destacar "*Visión por computador*" [20] en el que se introduce al lector los conceptos clave para sentar unas bases en el campo de la visión artificial. En "*Técnicas y algoritmos básicos de visión artificial*" [21] se desarrolla una amplia selección de técnicas y algoritmos de gran efectividad, surgida de la experiencia de los autores.

Actualmente la visión artificial (o visión por computador) supone una tecnología en continuo desarrollo, teniendo aplicaciones en multitud de campos. Algunos ejemplos por destacar son [22] y [23]:

- Inspección industrial y control de calidad: desde verificación del etiquetado de productos hasta inspección de soldaduras y clasificación de piezas.
- Vigilancia, identificación biométrica y reconocimiento facial.
- Control de tráfico a través del reconocimiento de matrículas.
- Guiado de robots industriales y vehículos autónomos (AGVs).
- Multitud de aplicaciones en la industria militar, medicina, alimentación…

Algunas aplicaciones concretas de aplicación de la visión artificial en inspección industrial focalizada en la problemática de la detección de defectos, área en el que se enfoca el presente proyecto, se discuten al final del apartado.

En todo proceso de visión artificial es posible identificar cuatro etapas (Figura 13):

1. Adquisición o captura: cuyo objetivo es la toma de imágenes digitales a través de un sensor o cámara.
2. Pre-procesado: consiste en la aplicación de determinadas operaciones a la imagen con el objetivo de facilitar las etapas siguientes.
3. Segmentación: se encarga de la división de los elementos principales de una imagen.
4. Clasificación: el objetivo de la última etapa es distinguir los elementos anteriores a través del análisis de características diferenciadoras.

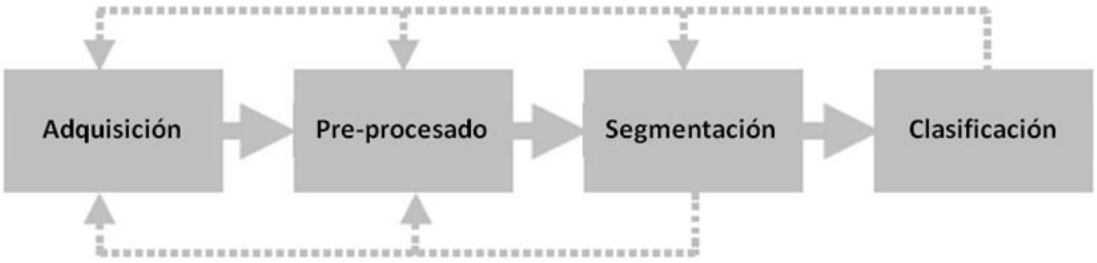


Figura 13. Etapas comunes en un proceso de visión artificial (Fuente: [20])

2.3.1.1 Adquisición de la imagen

La primera etapa de un proceso de visión artificial es la **adquisición** de la imagen a analizar. En esta etapa, el objetivo primordial es la obtención de dicha imagen de la forma más adecuada posible, facilitando así posteriores etapas. La elección de la cámara, el tipo de sensor o la iluminación de la escena, son elementos clave para una óptima adquisición.

Dentro de esta etapa se distinguen dos fases distintas. En primer lugar, se **captura** la escena mediante una cámara, elegida en función de las necesidades de la aplicación. En este punto entra en juego la correcta iluminación y enfoque de la escena, lo que facilitará los procesos siguientes. En segundo lugar, se **digitaliza**, transformando la información continua en una imagen digital discreta.

El resultado de esta etapa es una imagen bidimensional I_D formada por una matriz de *NxM* píxeles. Esto es lo que se conoce como **resolución espacial** y cuanto menor sea, menor será el tamaño en píxeles de la imagen, pero mayor será la pérdida de información debido al ruido en la misma. Esta matriz se puede representar de la manera siguiente:

$$I_D(x,y) = \begin{pmatrix} I_D(0,0) & \cdots & I_D(0,M-1) \\ \vdots & \ddots & \vdots \\ I_D(N-1,0) & \cdots & I_D(N-1,M-1) \end{pmatrix} \quad (1)$$

Cada píxel recoge la información sobre una determinada región, esta información se corresponde con el **nivel de intensidad** de la imagen en dicho punto. Para una imagen binaria (blanco y negro) solamente existen dos niveles de intensidad, para una imagen en niveles de gris se tienen hasta 256 niveles, desde el color negro (0) hasta el color blanco (255), por último, para una imagen a color se tienen 256 niveles por cada uno de los tres colores primarios, normalmente RGB (*Red Green Blue*).

En la Figura 14 se puede observar el efecto producido por la elección de distintos niveles de intensidad y distinto número de píxeles en la imagen de Lena, una imagen muy común en el campo del procesado de imágenes

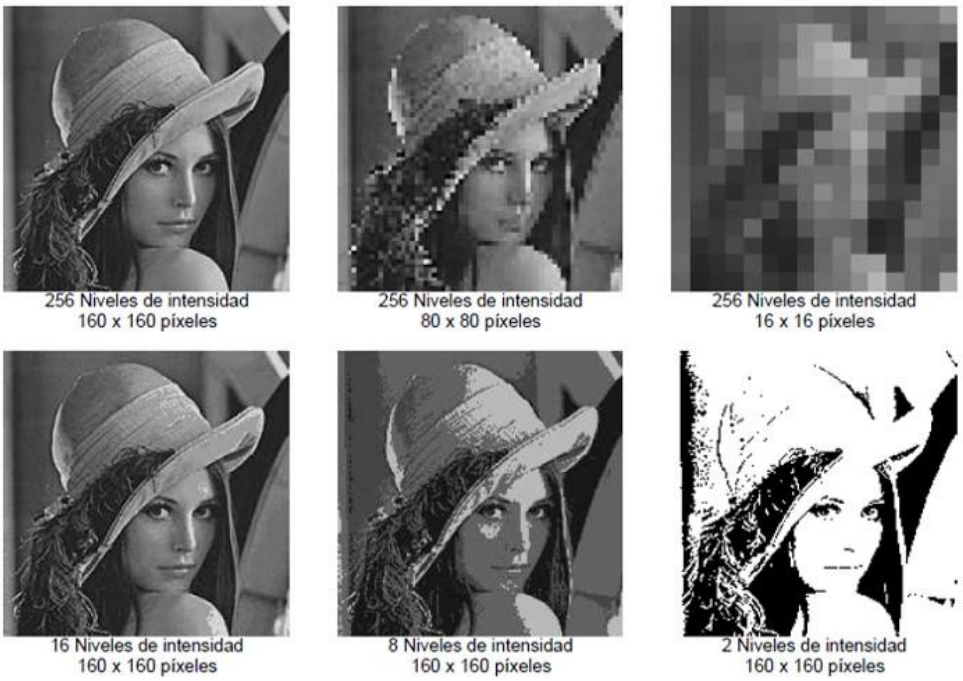


Figura 14. Efecto de la elección del n.º de niveles de intensidad y del n.º de píxeles (Fuente: [20])

2.3.1.2 Pre-procesado

Todas las imágenes muestran defectos una vez digitalizadas. Estos defectos pueden producirse por múltiples razones y suelen manifestarse en la imagen en forma de ruido, falta o exceso de contraste o brillo, etc. A través de la etapa de **pre-procesado** se pretende reparar dichos desperfectos con el objetivo de mejorar la imagen y facilitar el resto de las etapas.

El mecanismo por el cual se realizan los cambios en la imagen se conoce como filtro. De forma simplificada, es posible entender el funcionamiento de un filtro como la transformación de una señal de entrada (E) en una señal de salida (S), mediante la aplicación de una función de transferencia (H), cuyo esquema se puede ver representado Figura 15.

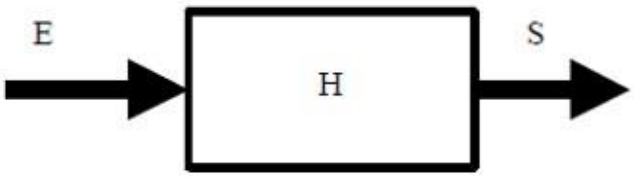


Figura 15. Esquema básico de un filtro (Fuente: [20])

Estos filtros se corresponden a diferentes algoritmos que se encargan de modificar la imagen a través de la realización de distintas operaciones de mejora. Existe una gran variedad de operaciones, desde las más básicas hasta las más complejas. A continuación, se describen brevemente algunas de estas operaciones:

1. Operaciones básicas: se realizan de manera discreta sobre cada píxel. Se suelen dividir en **operaciones aritmeticológicas** (suma, resta, multiplicación, división, conjunción, disyunción, negación, etc.) y **operaciones geométricas** (traslación, escalado, rotación, espejo, etc.). En la Figura 16 se representan algunas operaciones aritmeticológicas.

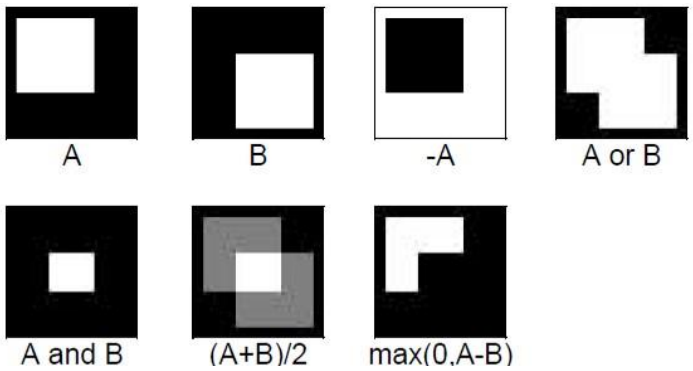


Figura 16. Ejemplos de operaciones aritmeticológicas (Fuente: [20])

2. Operaciones sobre el histograma: el histograma de una imagen es una representación estadística en forma de diagrama de barras de los niveles de intensidad de dicha imagen. Es posible aclarar los niveles claros y oscurecer los oscuros (o, al contrario) a través de la modificación del histograma de la imagen.

   Un ejemplo se encuentra en la Figura 17, donde se puede observar una reducción del contraste en la imagen superior, y un aumento del contraste en la imagen inferior.

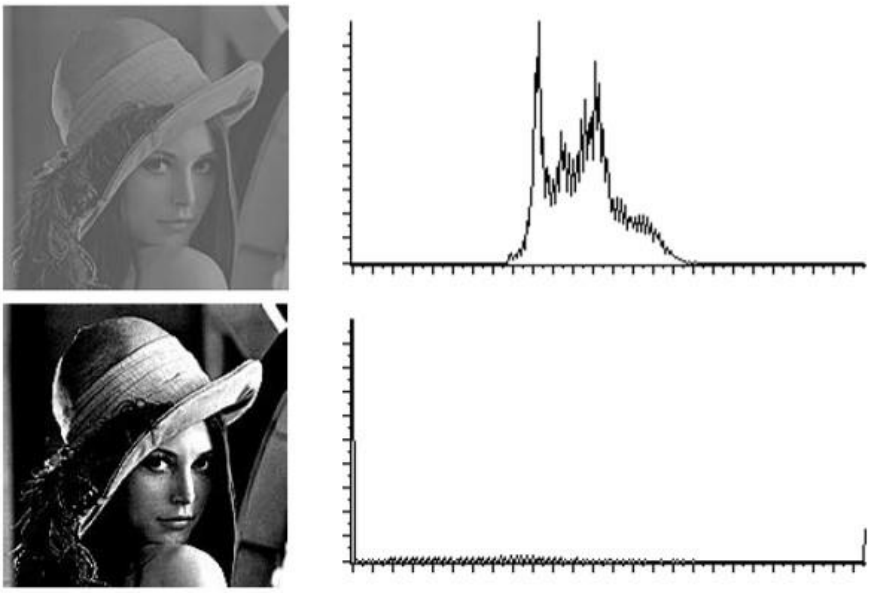


Figura 17. Ejemplos de modificación del histograma (Fuente: [20])

3. Operaciones en el dominio de la frecuencia: estas técnicas se basan en la aplicación de la Transformada Discreta de Fourier (DFT) y de la Transformada Discreta Inversa de Fourier (IDFT). Conociendo la frecuencia de aparición de ciertos elementos en una imagen es posible apreciar y modificar elementos como el ruido, bordes, texturas, etc.
4. Filtrado espacial: es posible trabajar directamente en el dominio del espacio a través de la operación de convolución. La convolución consiste en recorrer una imagen píxel a píxel realizando una determinada operación aritmética con un número de píxeles vecinos. Comúnmente a estos píxeles se les denomina "**ventana**" o "***kernel***", la cual podrá tener diferente tamaño (3x3, 5x5, etc.). La representación de un filtro se realiza a través de una matriz de convolución también de tamaño 3x3, 5x5, etc.

   Es posible formar una gran variedad de filtros mediante la selección de distintos valores para las celdas de la matriz. Algunos ejemplos son el filtro de la mediana o el filtro de **Gauss**, que suavizan la imagen, o los filtros de **Sobel** o **Prewitt**, que permiten acentuar los bordes. En la Figura 18 se ilustran dos ejemplos acompañados de las matrices de convolución correspondientes.

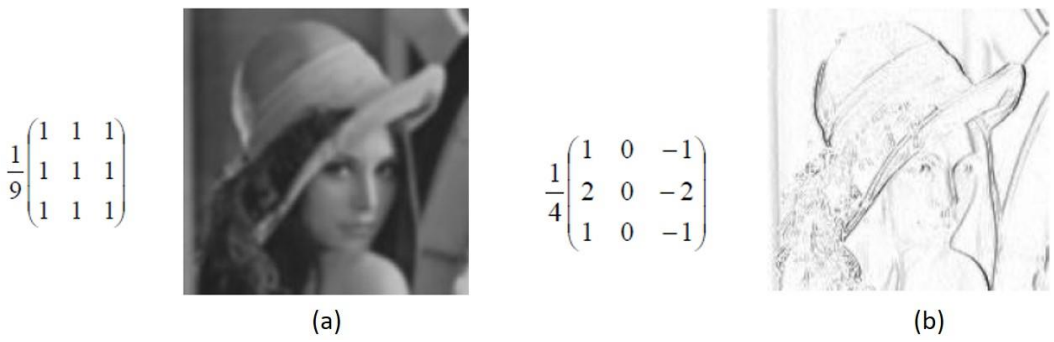


Figura 18. Ejemplos de suavizado (a) y acentuado de bordes (b) (Fuente: [20])

5. Operaciones morfológicas: este tipo de transformaciones cambian la forma y estructura de los elementos de una imagen, permitiendo separar elementos unos de otros, obtener contornos, etc. Para ello es necesario conocer previamente el elemento estructurante o máscara con el cual se realizará la operación. Las más conocidas son la **dilatación**, la **erosión**, la **apertura** y el **cierre**. El efecto de algunas de estas operaciones se ilustra en la Figura 19.

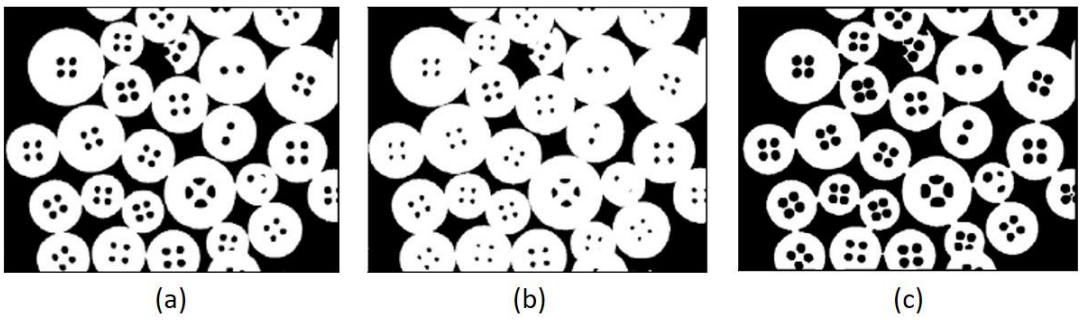


Figura 19. Efecto en una imagen (a) de las operaciones de dilatación (b) y erosión (c) (Fuente: [20])

2.3.1.3 Segmentación

Una vez realizadas aquellas operaciones necesarias para mejorar la imagen, se procede a realizar la **segmentación**. Este proceso consiste en la división de la imagen en distintas regiones respecto a una o más características comunes (brillo, color, etc.) facilitando así el posterior análisis y reconocimiento de los elementos.

Existe una gran variedad de técnicas en este aspecto, pudiéndose agrupar en tres grandes grupos:

1. Técnicas basadas en umbralizado: la umbralización consiste en transformar una imagen cualquiera en una imagen binaria (en blanco y negro), separando así el fondo de los elementos de interés. Las técnicas más comunes son la **umbralización fija**, en la que se elige un valor fijo de nivel de intensidad para marcar el umbral de separación, y la **umbralización adaptativa**, en la que el valor del umbral varía en distintas regiones de la imagen. La Figura 20 muestra la diferencia de estos dos tipos de umbralizado.

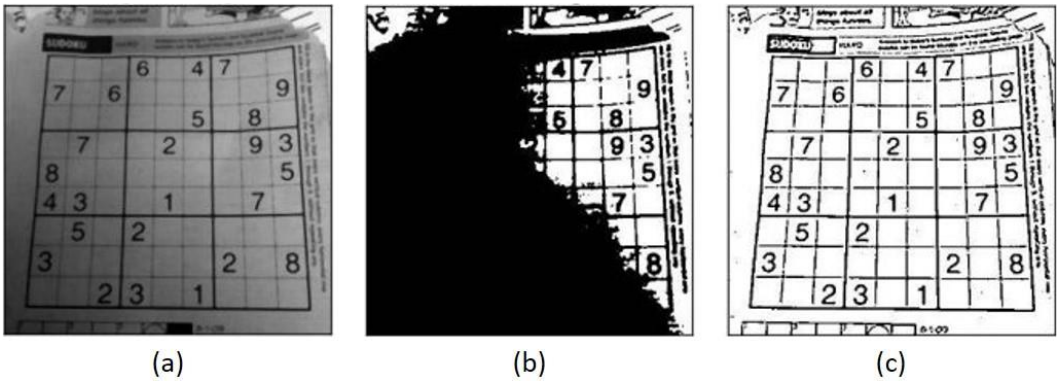


Figura 20. Aplicación en una imagen (a) de una umbralización fija (b) y una adaptativa (c)

2. Técnicas basadas en la detección de bordes: este grupo engloba un amplio número de técnicas que utilizan la información proporcionada por las fronteras de los elementos de una imagen. Estas fronteras marcan puntos donde se producen discontinuidades como cambio en el color, en los niveles de gris, en la textura, etc.

   Para acentuar las fronteras existen multitud de detectores de bordes basados en filtros como puede ser el de **Sobel**, **Prewitt**, **Canny**... En la Figura 21 se muestra el efecto de dos de estos detectores.

Figura 21. Distintos algoritmos de detección de bordes (Fuente: [20])

3. Técnicas basadas en el crecimiento de regiones: estas técnicas se basan en determinar zonas de la imagen a través de la similitud de sus píxeles, uniendo o dividendo la imagen en regiones comunes.
4. Otras técnicas: dada la gran cantidad de técnicas existentes, no es posible englobar todas en los distintos grupos. En imágenes a color es posible aprovechar la información aportada por el color para lograr segmentar elementos. También existen técnicas basadas en la extracción y el análisis de texturas.

La representación de los elementos segmentados se puede describir de distintas formas utilizando lo que se conoce como **descriptores** que permiten obtener representaciones únicas e independientes de la posición, orientación o tamaño del elemento.

Por ejemplo, para el caso de representación de fronteras (o bordes) se puede realizar mediante métodos como el código cadena, la signatura, o una aproximación poligonal. Algunos ejemplos de parámetros utilizados como descriptores son: la longitud del contorno, el diámetro de la frontera o la curvatura.

En cuanto a la representación de regiones, los métodos más comunes son el código de longitud variable, los *quadtrees* y las proyecciones. Existen multitud de descriptores utilizados: área, centro de gravedad, perímetros, momentos, ejes, etc.

2.3.1.4 Clasificación

La última etapa del proceso se corresponde con la **clasificación** de los elementos anteriormente segmentados. La distinción entre elementos se realiza dentro de un conjunto conocido previamente llamado **universo de trabajo**, que se encontrará divido en un repertorio de **clases** correspondientes con cada tipo de elemento.

Cada elemento que se quiera clasificar vendrá definido por unas **características discriminantes** o **rasgos**, cuyos valores concretos formarán un **patrón** que servirá para representar al elemento. Posteriormente este patrón se comparará con el conocimiento que se tiene de las clases existentes, determinando un grado de semejanza entre ambas

partes. Aquella clase con la que se tenga mayor grado de semejanza será a la que pertenezca el patrón. Este proceso se ilustra de manera simplificada en la Figura 22.

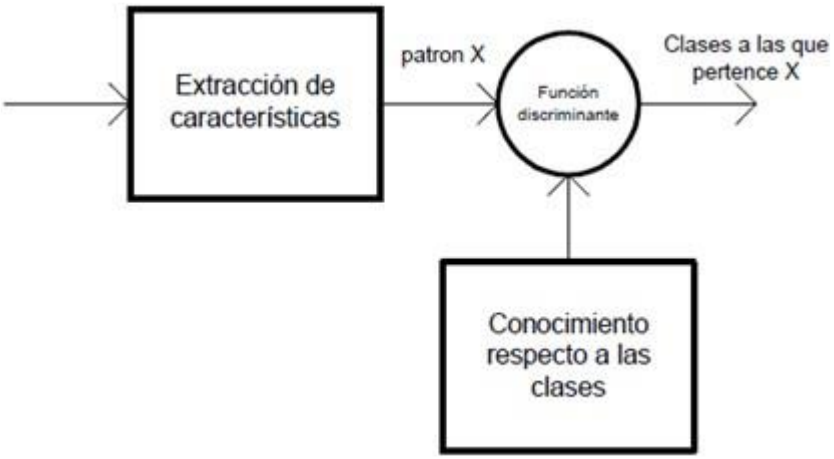


Figura 22. Esquema simplificado de un clasificador (Fuente: [20])

La comparación entre clases y patrón se realiza a través a una **función discriminante** calculada a través de un conjunto de patrones similares a los que se desea clasificar. Para ello se utilizan generalmente dos subconjuntos independientes entre sí que serán representativos del universo de trabajo. El primero se conoce como **conjunto de aprendizaje** o **entrenamiento**, y se utilizará para dotar al sistema del conocimiento de las clases, el segundo se conoce como **conjunto de prueba**, y su función es la de probar el sistema una vez construido. En ocasiones si se disponen de suficientes muestras, se puede formar un tercer conjunto llamado **conjunto de validación** que se utilizará durante la construcción del sistema.

Los porcentajes asociados a cada conjunto pueden ser muy variados. En ocasiones se utiliza un 75% para el de entrenamiento y un 25% para el de prueba. En otros textos recomiendan un 60% para el conjunto de aprendizaje, un 30% para el de prueba y un 10% para el de validación [20].

Existe una gran variedad de algoritmos, que pueden clasificarse atendiendo a distintas formas, siendo una de las más comunes según sea su aprendizaje:

- Supervisados: un maestro proporciona las etiquetas de las clases del conjunto de entrenamiento. Como ejemplo de clasificador supervisado se tiene SVM (*Support Vector Machines*) o AdaBoost (*Adaptive Boosting*).
- No supervisados: el conjunto de entrenamiento no dispone de las etiquetas de clase y el sistema será el que determine las clases. Los más conocidos son HMM (*Hidden Markov Models*), k-medias (*k-means*) o SOM (*Self-Organizing Maps*).
- Semi-supervisados: una mezcla de los dos anteriores, una parte del conjunto de entrenamiento estará etiquetado. Destacan el *Co-Training* y el *Re-Weighting*.
- Reforzados: el aprendizaje por refuerzo busca entrenar al sistema basándose en la experiencia y no en un conjunto de entrenamiento. Para este caso, uno de los más conocidos es *Q-Learning.*

2.3.2 Deep Learning

El ***Deep Learning*** es un subcampo dentro del ***Machine Learning***, que a la vez es un subcampo de la **inteligencia artificial**. De forma gráfica, la diferencia entre los tres campos se ilustra en la Figura 23.

El objetivo de la inteligencia artificial es el desarrollo de algoritmos que permitan facilitar al ser humano la realización de determinadas tareas repetitivas pero que suponen una importante carga computacional para un ordenador.

El análisis de imágenes es un ejemplo de una de estas tareas. Una persona corriente no tiene gran dificultad en interpretar una imagen, mientras que un ordenador, como se ha visto en el apartado anterior, necesita realizar una serie de operaciones que pueden llegar a ser muy complejas para llegar al mismo resultado.

Dentro del amplio campo de la inteligencia artificial, la parte más especializada en el desarrollo de técnicas para lograr que los ordenadores aprendan es el *Machine Learning*. Si se profundiza aún más en este campo se llega a lo que se conoce como *Deep Learning*. Su representación más conocida son las **Redes Neuronales Artificiales**, en inglés *Artificial Neural Networks* (ANNs), y comúnmente ambos términos suelen hacer referencia al mismo concepto.

Puesto que el objetivo del documento no es desarrollar de forma exhaustiva este campo, se vuelve a invitar al lector a acudir a bibliografía especializada en la materia. Por ejemplo, en "*Deep Learning for Computer Vision with Python*" [24] se realiza un amplio desarrollo del *Deep Learning* aplicado al desarrollo de aplicaciones de visión artificial.

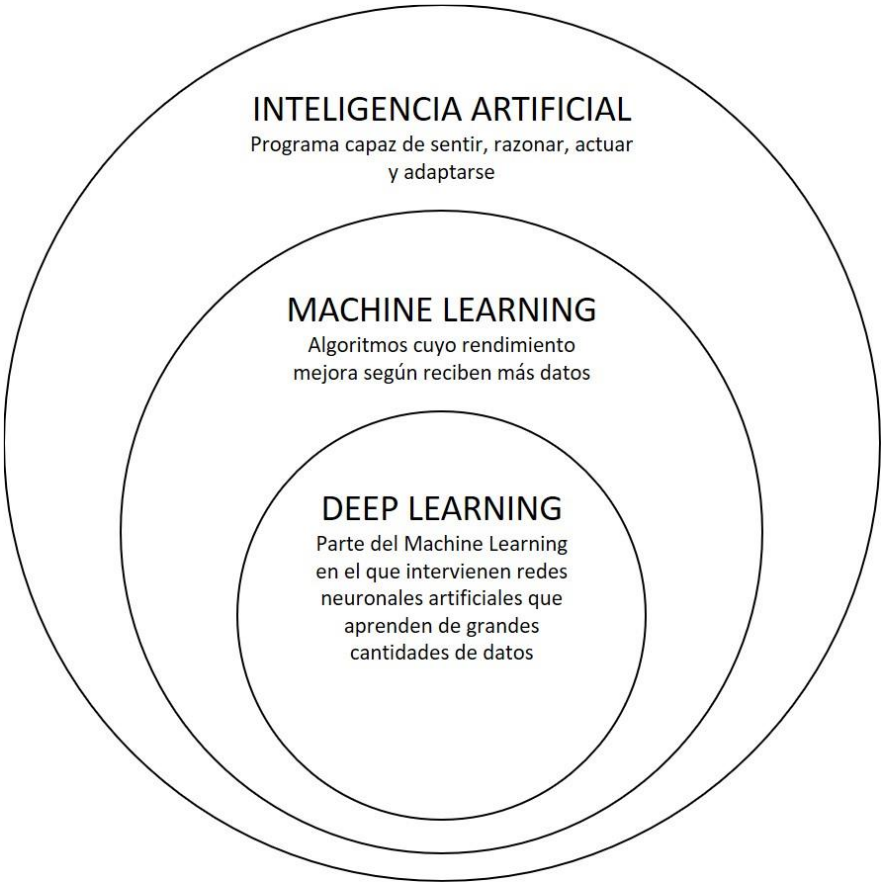


Figura 23. Diagrama de Venn para ilustrar los campos de la inteligencia artificial

2.3.2.1 Redes neuronales artificiales

El modelo de una **red neuronal artificial** se inspira en el cerebro humano, intentando imitar el funcionamiento de las neuronas. Estas redes consisten en una serie de unidades de procesamiento llamadas **neuronas**, que se encuentran interconectadas entre sí. Estas neuronas recogen una determinada señal de entrada y la procesan, generando una señal de salida.

Las redes se dividen en **capas** y cada capa tendrá un número determinado de neuronas. Existen tres tipos de capas: la **capa de entrada**, cuyas neuronas dependen directamente de la información del exterior; las **capas intermedias** u **ocultas**, que procesan y propagan la información desde la entrada hasta la salida; y la **capa de salida**, que definen la respuesta de la red ante una entrada determinada. En la Figura 24 se ilustra un esquema básico de una red neuronal artificial.

Cada neurona tiene tantas conexiones de salida como neuronas tenga la capa siguiente. Estas conexiones entre neuronas tienen asociado un valor real variable denominado **peso**, que determina la influencia de una neurona en el resultado de la siguiente.

La representación de una neurona se puede realizar mediante una ecuación matemática denominada **función de activación**. Esta función se encarga de procesar todas las entradas que llegan a la neurona, definiendo su estado de activación y generando una salida.

Por tanto, la salida de la red neuronal solamente depende de los valores de entrada, de las conexiones existentes entre las neuronas y de los pesos asociados a dichas conexiones.

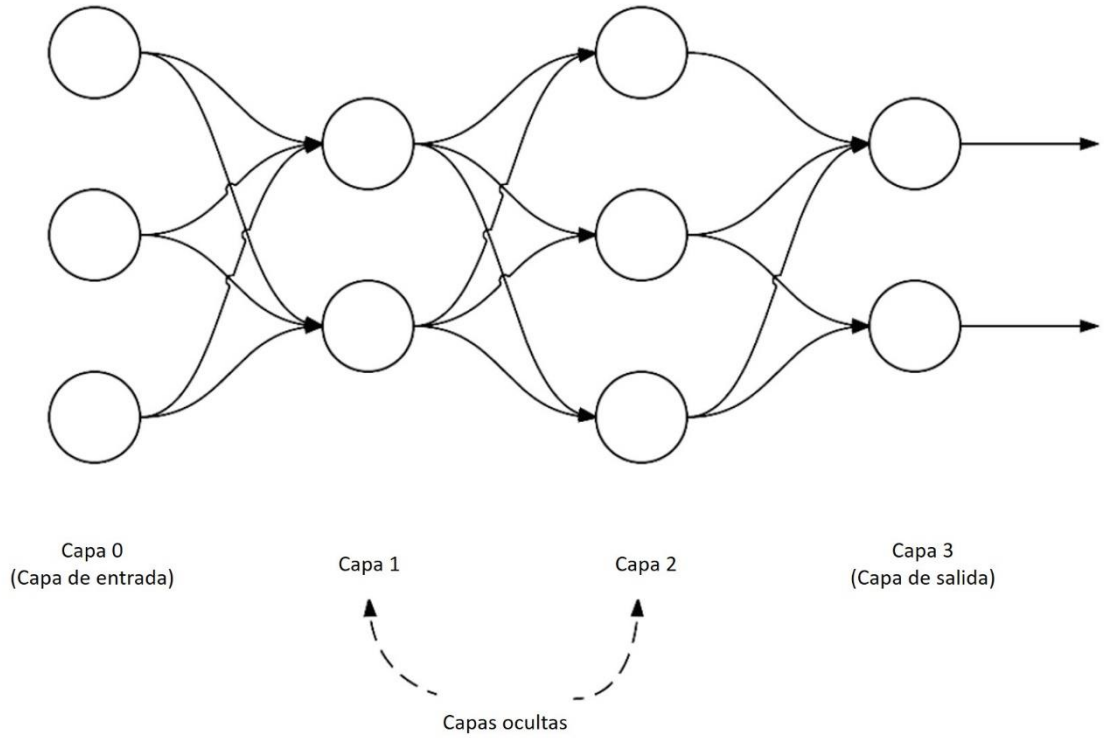


Figura 24. Esquema de una red neuronal artificial

2.3.2.2 Aprendizaje de una red neuronal

El proceso cuyo objetivo es el cálculo de los pesos que deben tener las conexiones de una red neuronal es lo que se conoce como **entrenamiento** o **aprendizaje**. En redes muy simples, se podría pensar en buscar estos valores de forma manual, pero es un proceso altamente complejo para redes normales. Por esta razón, la comunidad científica ha ido diseñando distintos tipos de algoritmos para lograr este objetivo.

De forma matemática, este proceso consiste en el cálculo de los valores de una serie de incógnitas (los pesos de las conexiones), para que unas determinadas funciones (la ecuación matemática que representa cada neurona) obtengan unos valores (la salida buscada) a partir de unos parámetros (las entradas de la red).

Es común no alcanzar el valor exacto buscado, por lo que el problema pasa por buscar el mínimo error cometido entre la salida deseada y la obtenida. Para ello, durante el proceso, se van actualizado los pesos de cada una de las conexiones de forma iterativa buscando ese mínimo error. El procedimiento más común se conoce como ***backpropagation*** (retropropagación), que consiste en la modificación de los pesos desde atrás hacia delante.

La idea del aprendizaje es similar a la etapa de clasificación de un proceso de visión artificial, siendo necesaria la existencia de un **conjunto de entrenamiento** que utilizará el algoritmo que define la red para ir variando esos pesos.

El objetivo final del aprendizaje es entrenar una red que sea capaz de **generalizar**, es decir, que sea capaz de obtener resultados adecuados a partir de una entrada nueva que no esté incluida en el conjunto de entrenamiento y, por tanto, la red no conozca.

Para conseguirlo es común extraer un segundo conjunto, para saber en qué momento detener el entrenamiento. Existe un punto de generalización óptimo (Figura 25), si el entrenamiento se detiene antes de llegar a dicho punto la red todavía es capaz de mejorar, pero si se pasa ese punto, se da el caso de **sobreentrenamiento** de la red, provocando que no sea capaz de generalizar.

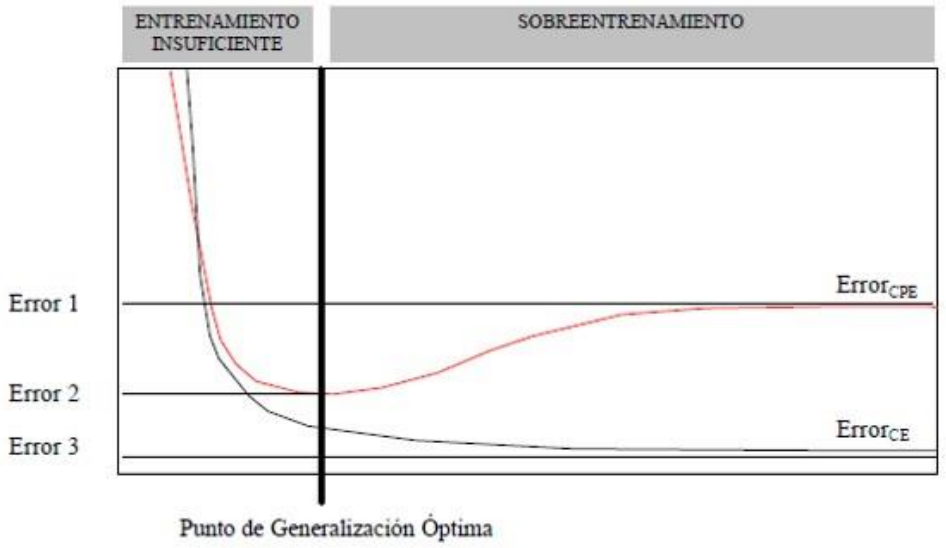


Figura 25. Generalización óptima de una red neuronal (Fuente: [20])

Para evaluar la red una vez entrenada es necesario disponer de un tercer conjunto de datos (o imágenes) que la red no haya utilizado durante el entrenamiento, con el objetivo de comprobar que la red es capaz de generalizar.

En un proceso de visión artificial tradicional, el conjunto de entrenamiento se compone de multitud de imágenes que el algoritmo utiliza para extraer una serie de características que podrá usar como patrón para diferenciar entre distintos elementos y clases. Esta extracción puede entenderse como un proceso "manual", puesto que se realiza durante el desarrollo del algoritmo.

Una de las mayores ventajas del uso de una red neuronal en este campo es evitar esta extracción "manual", centrándose en entrenar a la red para que aprenda esos patrones que le permitirán distinguir entre elementos. La Figura 26 representa una comparativa entre estos dos procesos.

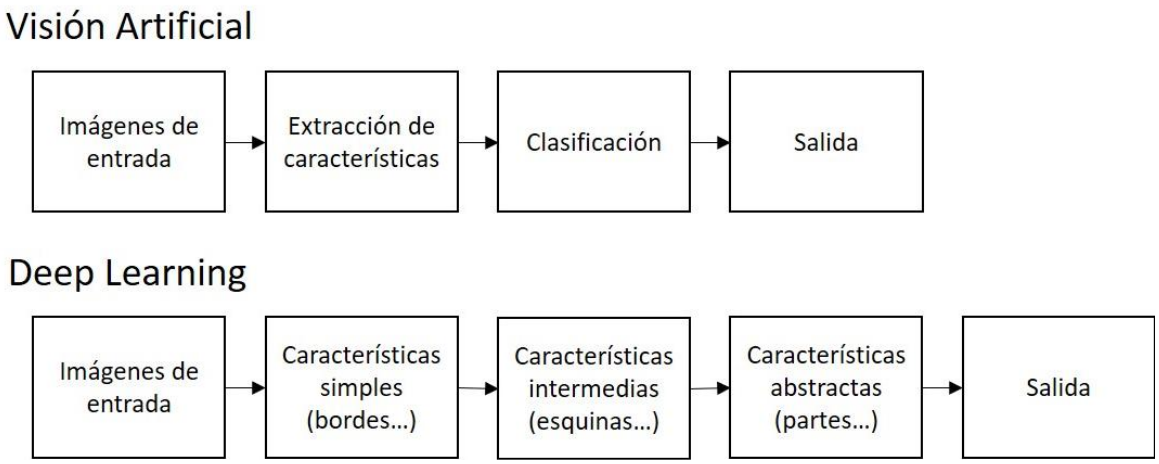


Figura 26. Comparativa entre un proceso tradicional de visión artificial y Deep Learning

2.3.2.3 Estructura de una neurona artificial

El modelo más sencillo de una red neuronal es el del **perceptrón**. Este modelo corresponde al uso de una sola neurona artificial en el que su estado de activación z es función del resultado de sumar todas las entradas x_i de la neurona multiplicadas por sus pesos w_i correspondientes, sumando una variable que depende únicamente de la neurona llamada sesgo o *bias, b*. Su representación matemática es:

$$z = b + \sum_i w_i x_i \tag{2}$$

Esta suma se transforma a través de la **función de activación σ** o **φ**, que determinará si la neurona se "dispara" o no. Para el caso del perceptrón esta función de activación es la función escalón. La salida de la neurona se corresponderá con:

$$y = \sigma(z) \tag{3}$$

Si se compone una red formada por múltiples perceptrones consecutivos se formaría lo que se conoce como **perceptrón multicapa**, generando una red más robusta y compleja. Este modelo se puede desarrollar mediante la utilización de otro tipo de funciones de activación. La Figura 27 representa la estructura simple de una neurona con *m* entradas.

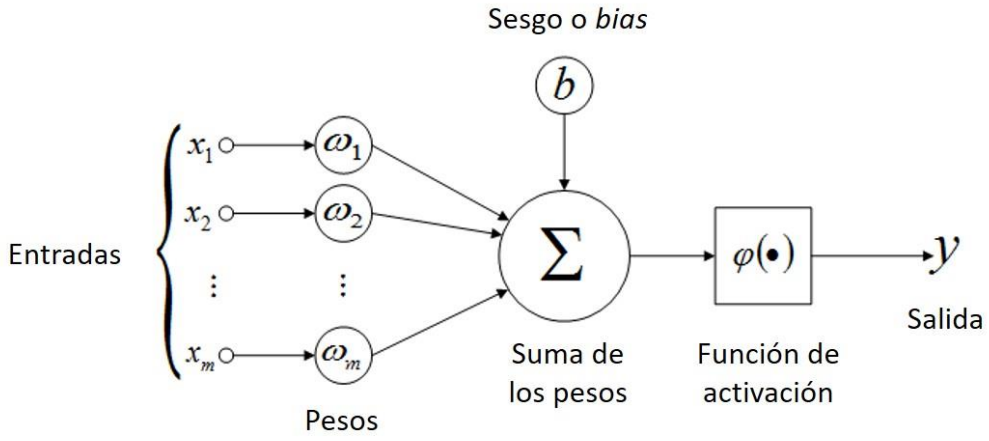


Figura 27. Estructura de una neurona artificial

Existen multitud de funciones de activación [24], [25] y [26], siendo algunas de las más comunes: la función escalón, la función sigmoide, la función tangente hiperbólica, la función ReLU, la función Leaky ReLU, la función ELU o la función *Softmax*.

- Función escalón: también conocida como función umbral. Es la función más simple de todas, utilizada por el modelo del perceptrón.

$$\sigma(z) = \begin{cases} 0 & si\ z < 0 \\ 1 & si\ z \geq 0 \end{cases} \tag{4}$$

- Función sigmoide: durante años ha sido una de las más utilizadas en el desarrollo de redes neuronales, pero ha perdido interés debido al aumento de complejidad de las redes.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{5}$$

- Función tangente hiperbólica: caso parecido a la sigmoide. Fue utilizada durante muchos años, pero ya ha perdido interés.

$$\sigma(z) = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{6}$$

- Función ReLU (*Rectified Linear Unit*): actualmente es la función más utilizada debido a su alta eficiencia computacional. Puede ocasionar problemas con valores negativos.

$$\sigma(z) = \max(0, z) \quad \text{ó} \quad \sigma(z) = \begin{cases} 0 & si\ z < 0 \\ z & si\ z \geq 0 \end{cases} \tag{7}$$

- Función Leaky ReLU: variante de la función ReLU que busca solucionar el problema dicha función.

$$\sigma(z) = \begin{cases} \alpha z & si\ z < 0 \\ z & si\ z \geq 0 \end{cases} \tag{8}$$

- Función ELU (*Exponential Linear Unit*): alternativa a la ReLU que suele obtener mejores resultados.

$$\sigma(z) = \begin{cases} \alpha(e^z - 1) & si\ z < 0 \\ z & si\ z \geq 0 \end{cases} \quad (9)$$

- Función *Softmax*: utilizada en clasificación multiclase. Suele utilizarse en las últimas capas de la red.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad (10)$$

Además de estas funciones, existen muchas otras posibilidades, pero no se ha querido entrar en gran detalle ni profundidad. En la Figura 28 se encuentran representadas seis de estas funciones.

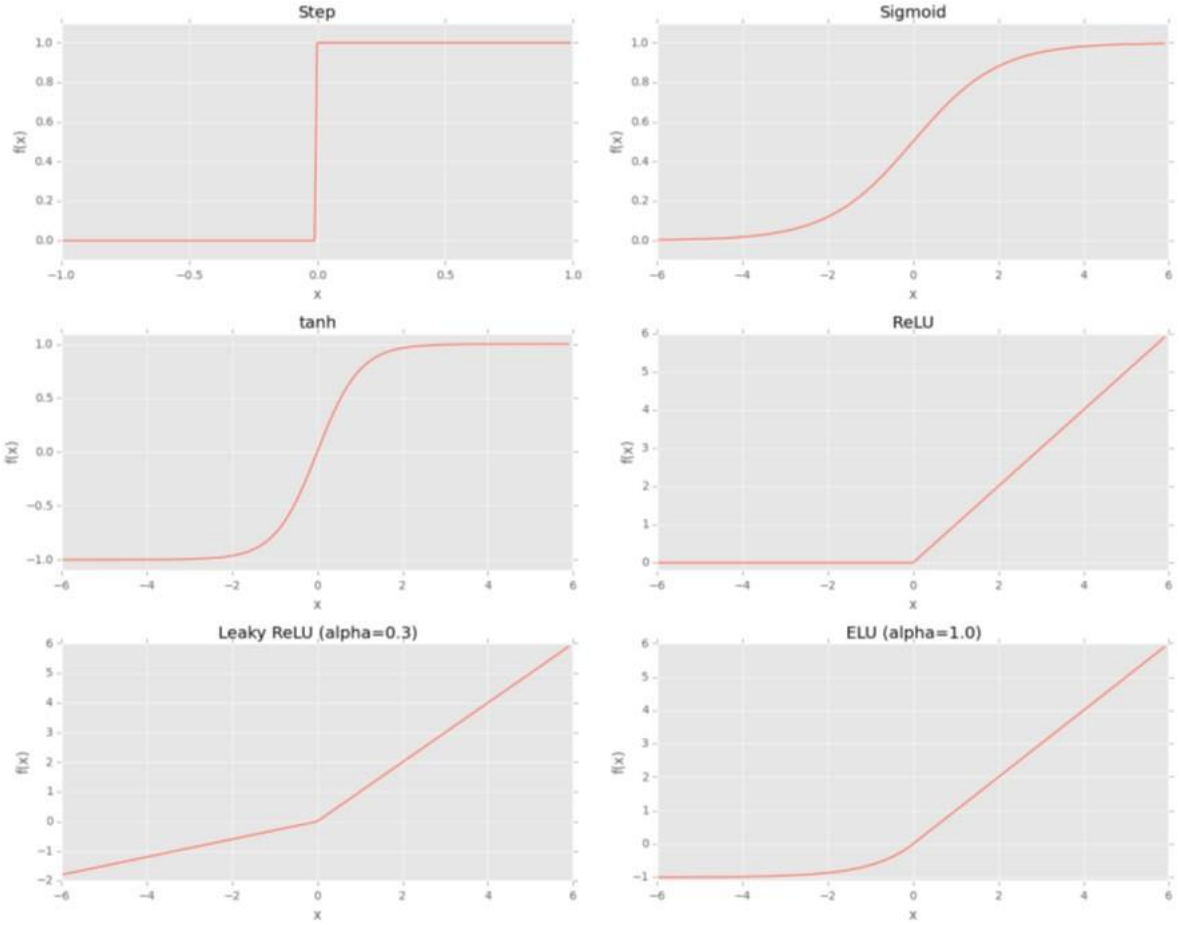


Figura 28. Representación gráfica de algunas funciones de activación (Fuente: [24])

La elección de utilizar una u otra función depende mucho del tipo de red neuronal a desarrollar. En la literatura especializada se puede encontrar que la mejor elección es utilizar la función ReLU, ya que suele obtener mejores resultados que las otras. También se recomienda utilizar la función *Softmax* en la capa de salida de la red, si se dispone de un problema de clasificación.

2.3.2.4 Redes neuronales convolucionales

Existe un tipo de red neuronal de gran importancia en el *Deep Learning*, en especial en el campo del reconocimiento de imágenes, las **redes neuronales convolucionales**. El desarrollo y explicación de este tipo de redes puede llegar a ser largo y complejo, por lo que en este apartado se pretende dar una idea general de los conceptos más generales respecto a este tema.

En una red neuronal tradicional, cada neurona de una capa está conectada con todas las neuronas de la capa siguiente, esto es lo que se conoce como **capa densa** o **completamente conectada** (*fully-connected layer*). Para una red neuronal convolucional, este tipo solamente se utiliza en la última (o últimas) capa(s) de la red, utilizando en su lugar una **capa convolucional**.

Tras cada capa convolucional se aplica una función de activación, típicamente la ReLU, hasta llegar a las capas finales de la red, que suelen ser una o dos capas densas.

Cada capa en una red neuronal convolucional se encarga de aprender una serie de filtros (cientos o incluso miles), combinando los resultados y pasando la información a la siguiente capa. Por ejemplo, para el caso del reconocimiento de imágenes, la red seguirá un proceso similar a:

- Detectar bordes de los elementos de la imagen.
- Utilizar esos bordes para detectar formas concretas.
- Utilizar esas formas para detectar características de alto nivel y reconocer elementos concretos.

En las operaciones de pre-procesado del apartado anterior, ya se habló acerca de la operación de convolución, pero a continuación se entrará algo más en detalle para explicar el funcionamiento de esta operación.

Matemáticamente una **convolución** consiste en multiplicar elemento a elemento dos matrices de la misma dimensión y sumar los elementos. Para el caso específico de trabajar con imágenes, la operación consiste en recorrer la imagen de izquierda a derecha y de arriba abajo, con una ventana o *kernel*, realizando la convolución y obteniendo una nueva matriz.

Por ejemplo, para el ejemplo de la Figura 29, la operación de convolución para la primera ventana de píxeles sería:

$$O_{11} = \begin{bmatrix} 3 & 0 & 1 \\ 1 & 5 & 8 \\ 2 & 7 & 2 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} =$$

$$= (3 * (-1) + 0 * 0 + 1 * 1) + (1 * (-1) + 5 * 0 + 8 * 1) + (2 * (-1) + 7 * 0 + 2 * 1)$$

$$O_{11} = 5$$

Esta misma operación se realiza de forma iterativa recorriendo toda la imagen obteniendo como resultado una nueva imagen.

Como ya se comentó, las distintas operaciones de suavizado, realzado, etc. consisten en la realización de operaciones convolucionales con *kernels* específicos conocidos como filtros. En visión artificial, estos filtros se aplican de forma "manual" y tienen valores específicos para obtener resultados determinados.

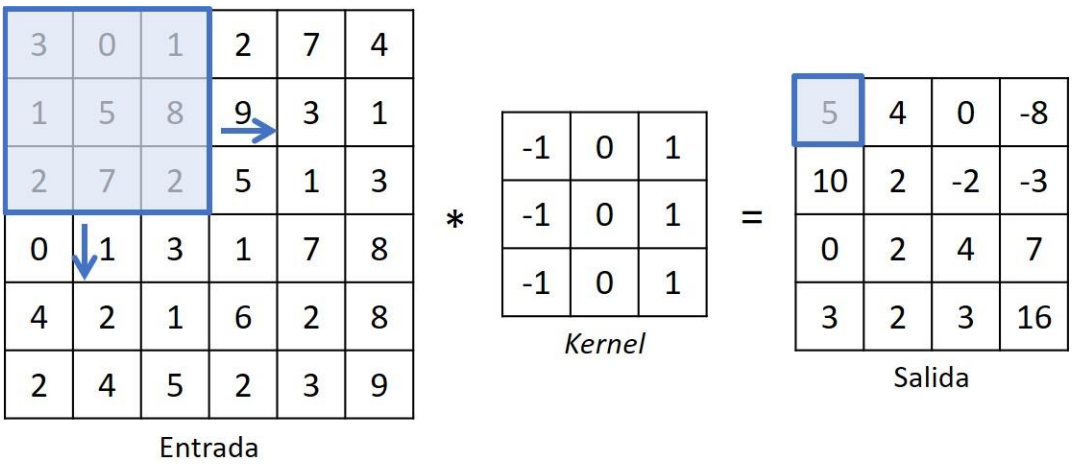


Figura 29. Ejemplo de operación de convolución

Ya se ha mencionado que las redes neuronales convolucionales son capaces de aprender esta clase de filtros de forma automática. En las primeras capas estos filtros son de bajo nivel y se utilizan para construir nuevos bloques que permiten analizar características de alto nivel en las capas más profundas de la red.

Tras la realización de las operaciones de convolución (y las de activación), la salida de cada capa se compone de múltiples características, por lo que es necesario reducir el tamaño de salida. Para ello, se utiliza una **capa de muestreo** o **agrupación** (*subsampling* o *max-pooling*) que se encarga de tomar las muestras más representativas de la capa anterior, reduciendo el coste computacional descartando aquellas características que tienen menos importancia dentro de la red.

De esta manera se alternan las capas convolucionales y de agrupación hasta llegar a las últimas capas de la red, que se corresponden con capas tradicionales. Por último, estas capas se conectan con el clasificador *Sotfmax* que tendrá tantas salidas como clases se quieran conocer. En esquema de este tipo de redes se ilustra en la Figura 30.

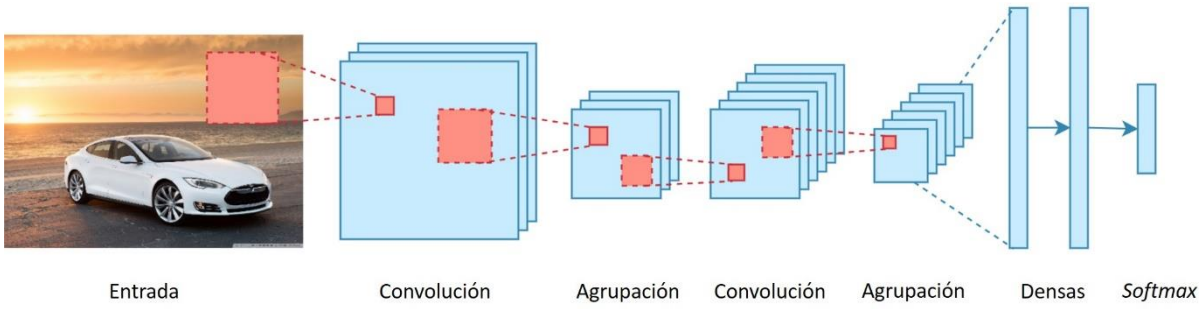


Figura 30. Esquema de una red neuronal convolucional

A través del uso de este tipo de redes, dotamos a los algoritmos de visión artificial de la capacidad de elegir de forma autónoma los filtros que debe ir aplicando en cada nivel para obtener las características más relevantes de unas imágenes de partida (asociadas a una serie de clases) y posteriormente, utilizar esos algoritmos para reconocer los elementos aprendidos y distinguir entre las distintas clases.

2.3.3 Aplicación en inspección industrial

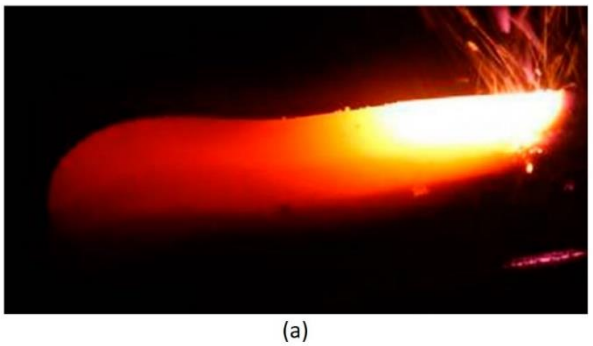
La detección de la defectología asociada a distintos procesos industriales a través de técnicas avanzadas como el uso de visión artificial o redes neuronales no es un problema nuevo y los esfuerzos realizados en este ámbito han estado presentes durante años.

Es posible encontrar en la literatura especializada multitud de aplicaciones distintas basadas en el uso de estos sistemas, tanto para la inspección en tiempo real de los procesos como para la detección y clasificación de diferentes tipos de defectos aparecidos a posteriori.

2.3.3.1 Control de procesos

En determinados procesos, la aplicación de sistemas de visión artificial permite controlar los parámetros clave en su ejecución. Con ello es posible obtener determinados datos que permiten optimizar el proceso en tiempo real.

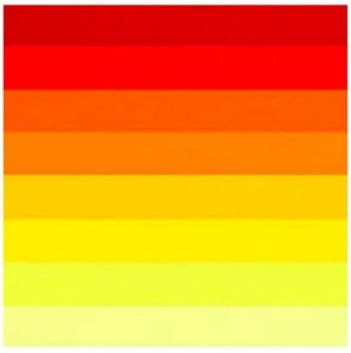
Por ejemplo, Barua et al. [27] proponen un sistema para la detección de los defectos producidos durante el proceso LMD (*laser metal deposition*). El sistema propuesto se encarga de analizar el hilo de metal según se va depositando en la superficie de la pieza, a través de la evaluación de la temperatura de cada píxel de la imagen. A través de distintas tonalidades de color en base RGB (*Red Green Blue*) se relacionan con distintos valores de la temperatura del hilo en Celsius, en la Figura 31 se muestra la relación seguida por los autores para obtener los gradientes de temperatura. Mediante estos datos es posible alertar en el caso de detectar una desviación, lo cual permite implementar una metodología de análisis continúo denominada control “on-line”, corrigiendo el proceso al instante, y por ende mejorando la calidad de la producción al evitar la aparición de grietas o poros.



(a)

R	G	B	Temperature (Celsius)
2	1	251	650
0	2	254	700
0	145	254	750
0	223	255	800
0	255	255	850
15	255	244	900
212	255	252	950
167	254	250	1,000

(b)



(c)

Figura 31. Gradiente de color del hilo depositado (a) y relación entre el gradiente de color RGB y su temperatura asociada (b) y (c) (Fuente: [27])

2.3.3.2 Análisis del estado de soldaduras

A través de la aplicación de operaciones de pre-procesado y segmentación como las vistas en apartados anteriores (2.3.1.2 y 2.3.1.3) es posible analizar el estado de los cordones de soldadura.

Chu & Wang [18] proponen un sistema que permite examinar la calidad de las soldaduras mediante algoritmos de procesamiento de imágenes. El algoritmo propuesto se encarga de extraer la región de interés relacionada con el cordón de soldadura y obtener los puntos más importantes de este, analizando sus dimensiones y posibles defectos.

La imagen de partida se suaviza mediante el filtro de la mediana (Figura 32c) para reducir el ruido de la imagen y se realiza una umbralización (Figura 32d) para extraer el fondo. Posteriormente, se procesa la imagen mediante la operación de cierre (Figura 32e) para lograr que la línea sea continua y eliminando las zonas de mayor área se logra separar la línea de interés (Figura 32f). Se extrae la región de interés de la imagen, para reducir el tiempo y la carga computacional (Figura 32g). El siguiente paso consiste en obtener la línea media del contorno a analizar (Figura 32h y Figura 32i) para al final extraer los puntos característicos del cordón (Figura 32j) a través de un algoritmo basado en áreas.

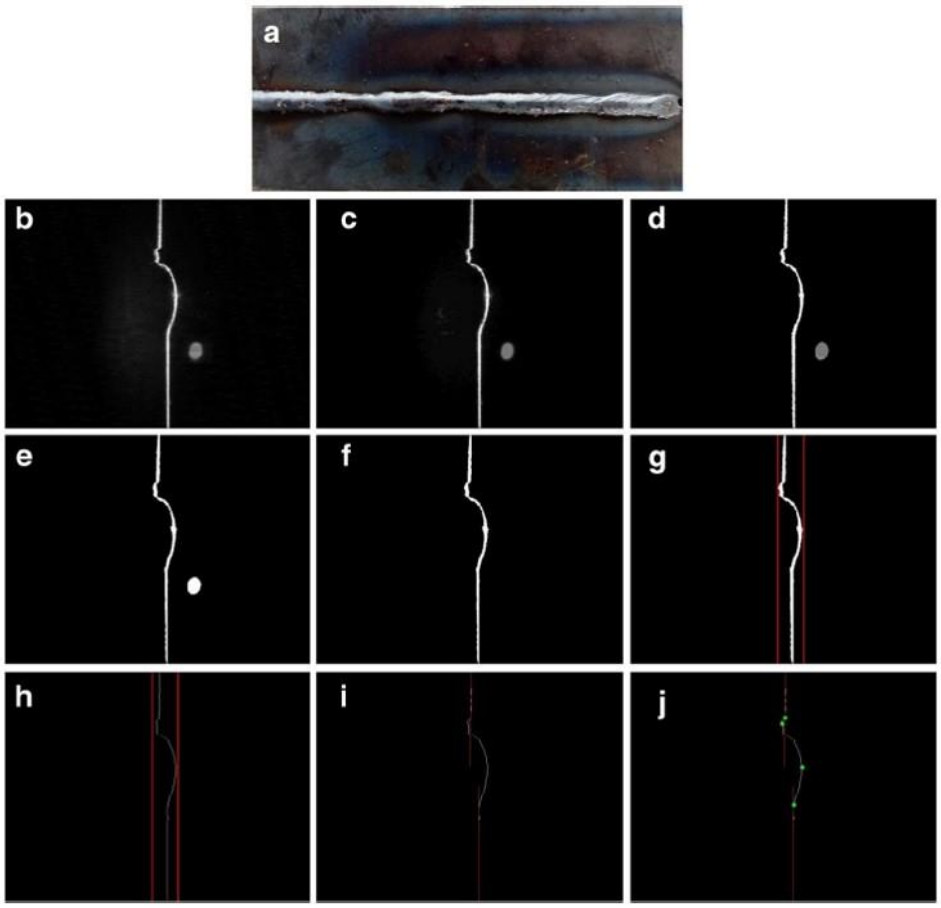


Figura 32. Procesamiento interno de las imágenes para el análisis de cordones de soldadura (Fuente: [18])

Como se puede apreciar, a través de la obtención de estos puntos es posible determinar las dimensiones del cordón de soldadura. Estas dimensiones se pueden comparar con los valores de referencia, lo que permite juzgar en última instancia la calidad del cordón.

Otro método de detección de defectos en soldaduras es el propuesto por Sun et al. [28]. Proponen un algoritmo para identificar y clasificar defectos de soldadura en recipientes metálicos, a través de la categorización previa de dichos defectos y la posterior extracción de las características relevantes del cordón de soldadura. La Figura 33 muestra el proceso seguido.

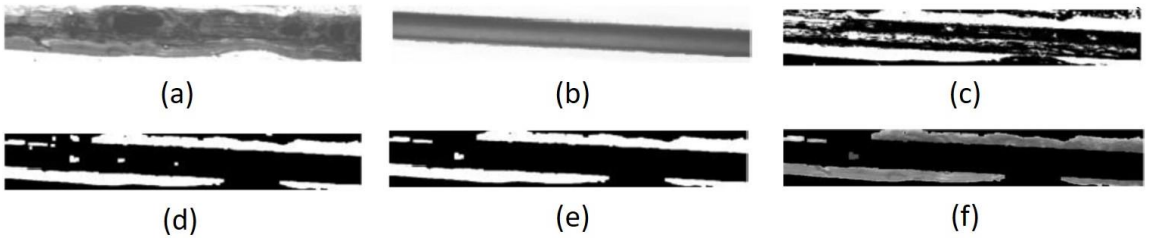


Figura 33. Extracción de los elementos característicos de la soldadura (Fuente: [28])

A diferencia del anterior, este sistema permite clasificar los defectos encontrados en los cordones de soldadura. Una vez descartados los pseudefectos y las falsas detecciones, a través de la medición del brillo de las imágenes el algoritmo es capaz de distinguir entre los distintos defectos asociados.

2.3.3.3 Inspección de raíles

Otra aplicación muy interesante de la visión artificial es la inspección de raíles in situ a través de sistemas portátiles como el propuesto por Min et al. [29].

Utilizan un procesado morfológico y el código cadena para obtener las características de los defectos evitando realizar un pre-procesado excesivo de la imagen. La propuesta consiste en un sistema portátil y rápido, capaz de realizar el análisis en tiempo real durante una inspección de campo.

En primer lugar, el algoritmo detecta las regiones de interés de la imagen para poder analizar únicamente las zonas correspondientes a los raíles. El procesado seguido consiste en la aplicación del filtro de la mediana para reducir el ruido de la imagen y posteriormente utilizan el algoritmo de Canny junto con operaciones morfológicas para lograr segmentar la imagen.

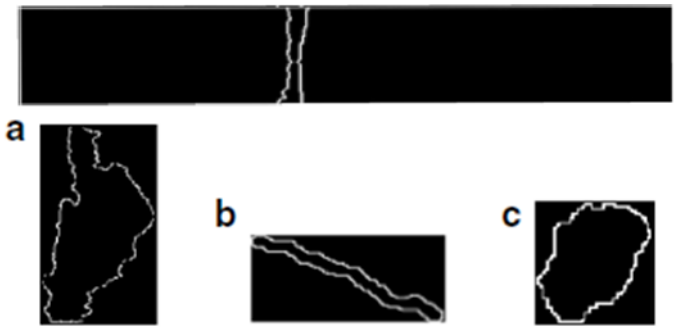


Figura 34. Ejemplos de extracción de defectos a través del procesado (Fuente: [29])

2.3.3.4 *Análisis y clasificación de tornillería*

La inspección industrial a través de visión artificial y el *Deep Learning* permite la fácil localización y clasificación de ciertos elementos como pueden ser tornillos u otros elementos de unión.

El sistema propuesto por Song et al. [19] se basa en el uso de redes neuronales convolucionales para la detección de varios tipos de micro-defectos en cabezas de tornillos. El método no requiere obtener las características de los tornillos, puesto que la red está entrenada para distinguir entre tornillos con defectos y tornillos libres de defectos. Además, se realiza una comparativa con técnicas tradicionales, como el uso de patrones de comparación, demostrando la gran robustez y superioridad del uso de una red convolucional.

En la Figura 35 se muestra el esquema de la red neuronal utilizada por los autores, cuya arquitectura está basada en la red LeNet-5, utilizada originalmente para el reconocimiento de caracteres.

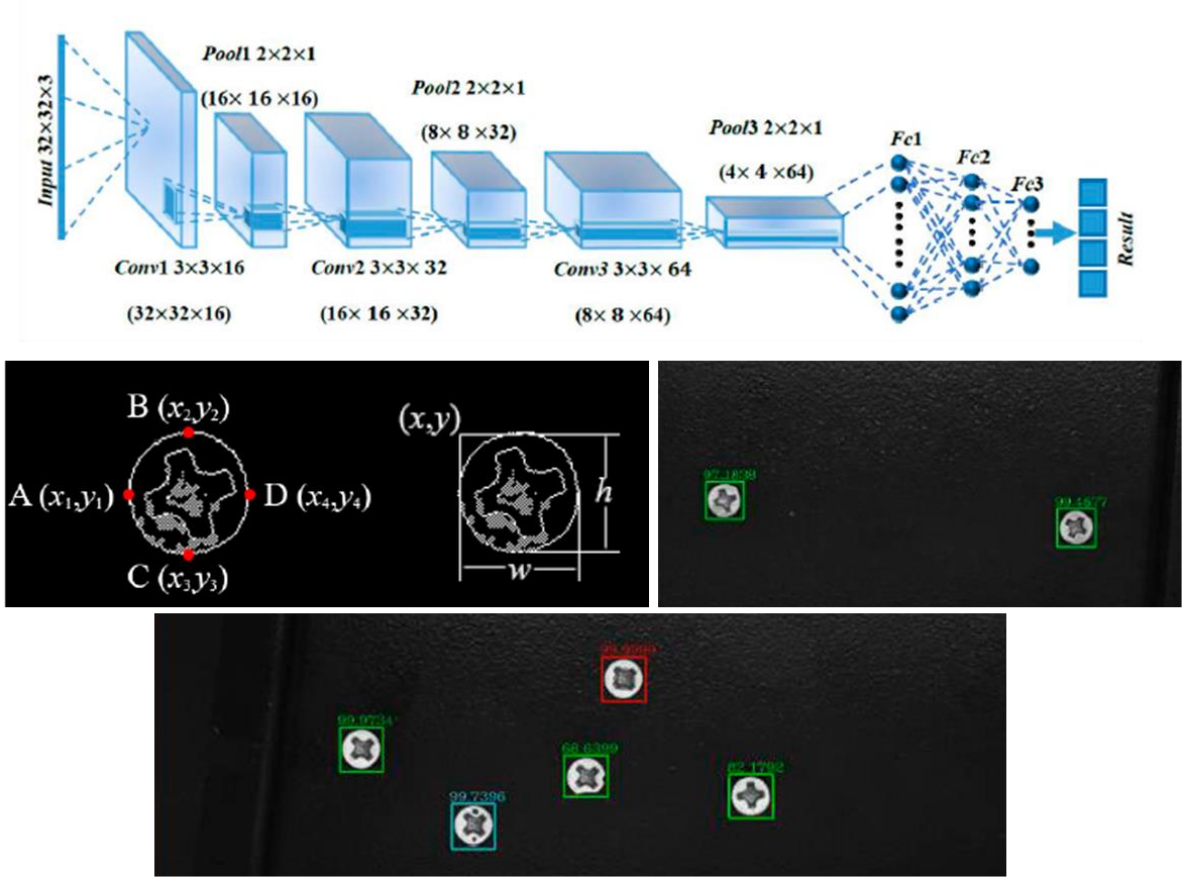


Figura 35. Red neuronal propuesta por los autores y resultados del algoritmo (Fuente: [19])

Como se puede observar en la figura, a través de la detección de los contornos de los tornillos, es posible extraer determinados puntos característicos que permiten tanto obtener las dimensiones de estos como su localización. El objetivo final del algoritmo es facilitar tanto la localización como la clasificación de los tornillos con distintos tipos de defectos.

2.3.3.5 Detección automática de defectos

La posibilidad de desarrollar un sistema automático de detección de defectos en piezas ensayadas a través de líquidos penetrantes resulta altamente interesante. La inspección visual directa de estos ensayos se ve altamente influenciada por el factor humano llegando a producir resultados muy variables.

Shipway et al. [30] estudian distintos métodos para la detección automática de defectos en piezas analizadas mediante el ensayo de líquidos penetrantes fluorescentes, uno de los ensayos más utilizados para la detección de grietas y otros defectos en componentes aeroespaciales. En el estudio se busca evaluar el rendimiento del algoritmo *Random Forest*, basado en árboles de decisión que se ha mostrado como una alternativa muy prometedora en la detección automática de defectos en líquidos penetrantes fluorescentes. La metodología seguida consiste en la modificación del algoritmo para analizar si se obtienen mejoras en el rendimiento y el coste computacional asociado

2.3.3.6 Detección y clasificación de defectos

Una de las aplicaciones más interesantes consiste en desarrollar un sistema capaz de detectar y clasificar defectos a través del uso de redes neuronales. Existen multitud de estudios que buscan encontrar el mejor método según sea la aplicación buscada.

Una de las propuestas más novedosas es la de Jiang et al. [31], quien define y entrena un algoritmo para realizar el análisis de ejes defectuosos. La metodología propuesta pasa por el desarrollo de un algoritmo basado en una red neuronal convolucional para realizar la detección y clasificación de los defectos encontrados en los ejes.

En la Figura 36 se ilustra el esquema del sistema desarrollado por los autores. En primer lugar, se etiquetan las imágenes con los defectos buscados que se utilizarán para el entrenamiento. Después, la red entrenada se encarga de localizar y clasificar los defectos detectados en los ejes.

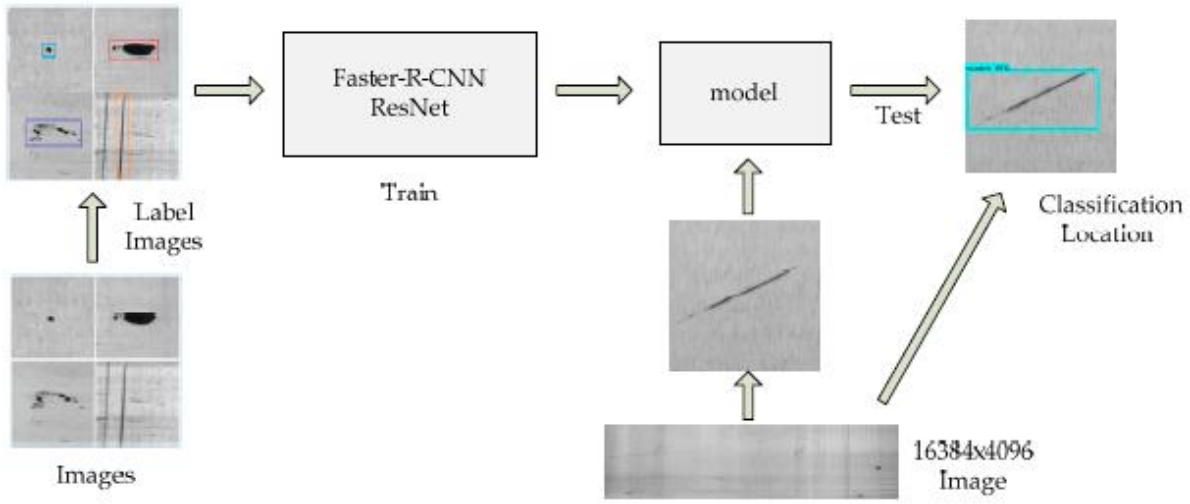


Figura 36. Esquema del proceso seguido (Fuente: [31])

Principalmente los autores destacan la aparición de cuatro tipos de defectos: huecos y hoyos, abrasiones, raspaduras y grietas. En la Figura 37 se muestran resultados de la aplicación de la red neuronal planteada.

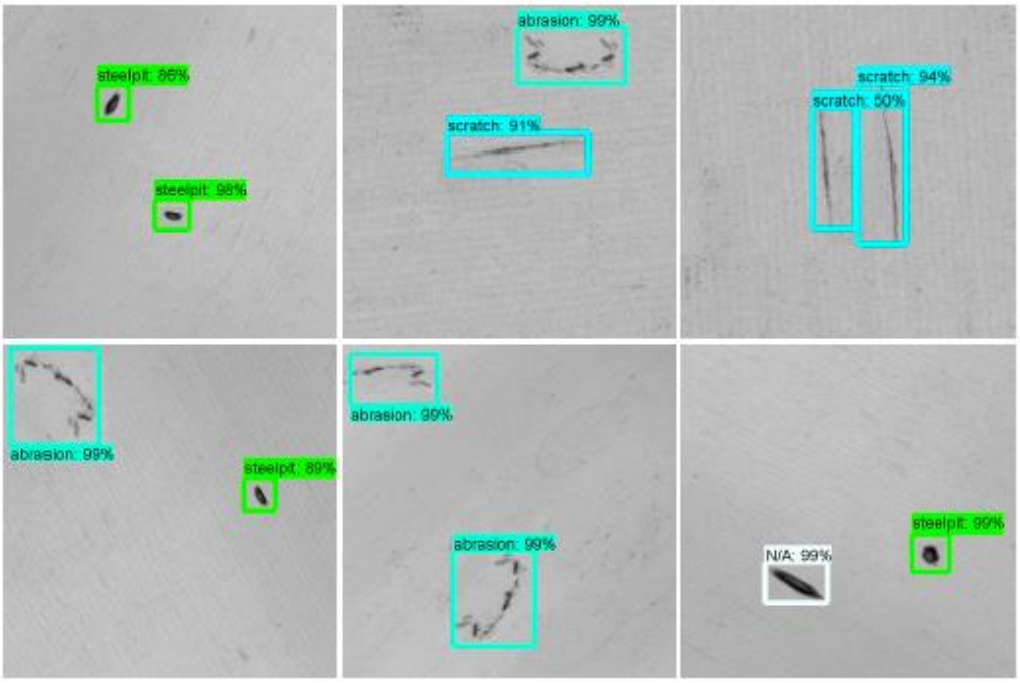


Figura 37. Ejemplos de resultados de detección y clasificación (Fuente: [31])

Un detalle interesante planteado por los autores es la utilización de un utillaje para la adquisición de imágenes basado en un escáner lineal y eje de rotación para los ejes para la obtención de las imágenes de los defectos, debido a la dificultad de obtener una buena captura de superficies brillantes y pulidas.

2.3.3.7 Conclusiones

A través de esta revisión de la bibliografía se han llegado a varias conclusiones respecto a la posibles alternativas a seguir:

- De forma general existen dos maneras de abordar el problema de detección de defectos: a través del uso de un sistema tradicional de visión artificial utilizando operaciones de procesado y segmentación, o mediante el uso de *Deep Learning* utilizando redes neuronales convolucionales.
- La detección se realiza generalmente en la pieza sin haber realizado un ensayo no destructivo previo, por lo que es interesante analizar la combinación de las dos técnicas. A través de dicha combinación, sería posible obtener y analizar la información de tal manera que mediante el uso de una única técnica no sería posible.
- Los métodos de obtención de la imagen son variados, dependiendo de cada una de las aplicaciones.
- Algunas de las propuestas son específicas para la aplicación analizada mientras que otras son extrapolables a otros campos.

CAPÍTULO 3: INTRODUCCIÓN AL DESARROLLO TÉCNICO

Antes de comenzar con el desarrollo técnico del proyecto, es conveniente conocer los recursos con los que se va a trabajar. Para ello, en este capítulo se describe el software utilizado para la ejecución del proyecto y se presentan las especificaciones del sistema utilizado. Además, se realiza una introducción general a la programación en el lenguaje escogido y se presenta la metodología empleada para la evaluación y presentación de resultados.

3.1 METOLODOGÍA DE DESARROLLO

El desarrollo del proyecto se ha divido en cuatro grandes bloques muy relacionados entre sí que permiten ir avanzando en distintas fases de desarrollo:

- La **primera fase** supone la familiarización con el entorno de trabajo. Ya se había trabajado antes con el software escogido y por tanto el objetivo final de esta fase no es más que realizar una introducción a la programación con Python y OpenCV.
- La **segunda fase** recoge el grueso más técnico del proyecto. En esta parte se desarrolla el algoritmo de detección de defectos que se utilizará de forma posterior a la inspección por líquidos penetrantes, evaluando los resultados obtenidos.
- La **tercera fase** es una continuación directa de la anterior. Mediante el algoritmo desarrollado se pretende evaluar la posibilidad de prescindir del ensayo por líquidos penetrantes para realizar la detección de defectos.
- La **cuarta fase** supone la aplicación de una alternativa al algoritmo basado en visión artificial. En este punto se evalúa la posibilidad de aplicar el *Deep Learning* a la detección defectos.

3.2 RECURSOS UTILIZADOS

En cuanto a software, existen multitud de formas de abordar el desarrollo de un sistema de visión artificial. Por ejemplo, la empresa MathWorks® posee los software de MATLAB® y Simulink® que incluyen numerosas herramientas para el desarrollo de aplicaciones de visión artificial y el procesamiento de imágenes.

También existen potentes programas diseñados específicamente para realizar sistemas automáticos de inspección de calidad a nivel industrial.

Una opción muy común en el desarrollo de aplicaciones basadas en visión artificial y *Deep Learning* es la utilización de la librería OpenCV en combinación con un lenguaje de programación como C++, Java o Python.

Una de las mayores ventajas del lenguaje Python es su facilidad de uso y programación. La combinación de este lenguaje con la gran variedad de funcionalidades que proporciona la librería OpenCV permite desarrollar herramientas muy avanzadas.

Debido a las grandes ventajas aportadas por ambas herramientas, se decidió trabajar con esta combinación del lenguaje Python con la librería OpenCV para realizar el desarrollo del proyecto.

3.2.1 Software

Como se ha mencionado, existen múltiples lenguajes de programación que permiten el desarrollo de aplicaciones de visión artificial. Uno de los más conocidos y utilizados, gracias en gran medida a su carácter gratuito y multiplataforma, es **Python** [32].

Python surge a principios de los años 90 como pasatiempo del ingeniero holandés Guido Van Rossum cuyo objetivo fue el diseño de un lenguaje sencillo de utilizar y aprender, pero manteniendo su potencia. Es un lenguaje en continua evolución con multitud de características que lo hacen ideal para trabajar con grandes volúmenes de datos.

La extensa comunidad ha permitido aumentar las capacidades del lenguaje a través de la incorporación de librerías que añaden nuevas funciones. En los campos de *Machine Learning* y *Deep Learning*, algunas de las más utilizadas son matplotlib, NumPy, SciPy, TensorFlow o Keras.

La instalación de **Python** en un ordenador personal puede resultar complicado, sobre todo a la hora de seleccionar los paquetes que se desean implementar de forma manual. Existen métodos alternativos siendo uno de los más conocidos la utilización de la suite de **Anaconda** [33], que incorpora multitud de librerías preinstaladas, formando una completa biblioteca orientada al procesamiento de grandes volúmenes de datos y a la programación científica.

Por último, el desarrollo de aplicaciones de visión artificial se ve facilitado mediante la utilización de **OpenCV** [34]. **OpenCV** (*Open Source Computer Vision Library*) consiste en una biblioteca de código abierto multiplataforma que incluye multitud de algoritmos de visión artificial. Incluye distintos módulos que recogen distintas funcionalidades de procesamiento de imágenes, detección de objetos, calibrado de cámaras, etc. Fue desarrollada por Intel, bajo la licencia BSD (*Berkeley Software Distribution*), lo que permite su libre uso tanto con fines comerciales como de investigación. Multitud de compañías hacen uso de ella, como, por ejemplo: Google, Microsoft, Intel, IBM o Sony, entre otras.

3.2.2 Hardware

El desarrollo del proyecto se realizó en un ordenador personal de sobremesa con las siguientes especificaciones:

Tabla 5. Principales especificaciones del ordenador utilizado

Sistema operativo	Windows 10 Pro de 64 bits	
Placa base	Gigabyte GA-B85M-D2V	
Procesador	Intel® Core™ i7-4790 3.60 GHz	
Tarjeta gráfica	NVIDIA GeForce® GTX 1060 6 GB	
Memoria RAM	16 GB DDR3	
Almacenamiento	HDD: 1 TB	SSD: 500 GB + 250 GB

3.2.3 Entorno de programación

Previamente al desarrollo de la herramienta de detección de defectos mediante visión artificial sobre las imágenes captadas en los ensayos por líquidos penetrantes fue necesario familiarizarse con el entorno de programación. Se utilizará Spyder como entorno de trabajo, al que es posible acceder desde el navegador principal de Anaconda.

Una vez dentro de Spyder encontramos una interfaz limpia y sencilla en la que es posible distinguir cuatro secciones (Figura 38):

1. Barra de tareas: menú de acceso a todas las herramientas del IDE
2. Editor: ventana de edición de código fuente
3. Explorador: ventana que permite explorar variables, archivos, gráficos y mostrar ayuda
4. Terminal: permite la introducción de comandos a través de la consola y de la visualización del historial de ejecución

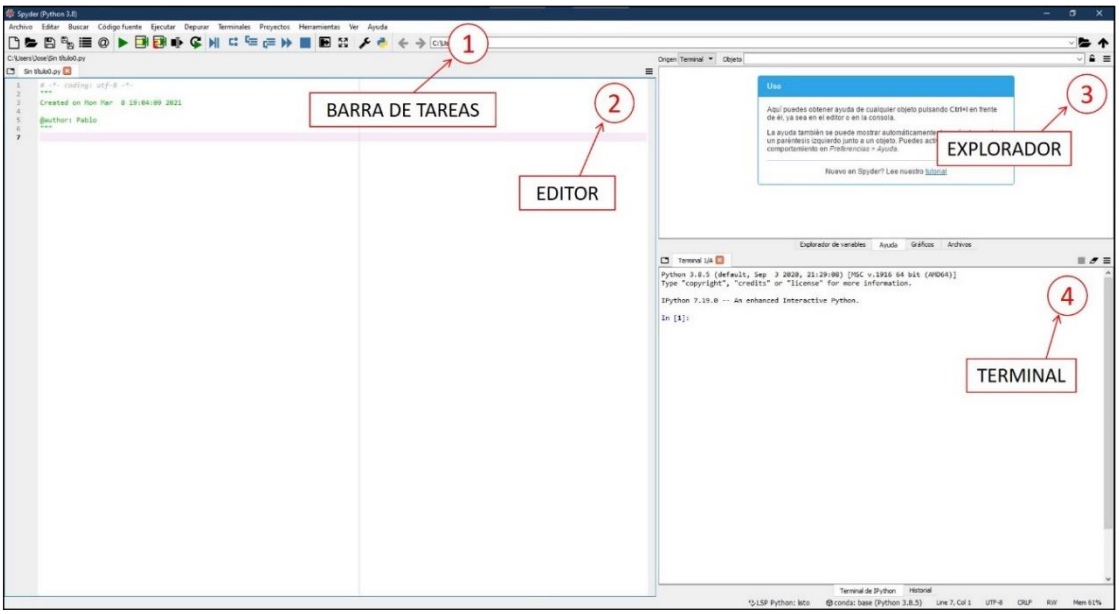


Figura 38. Pantalla principal de Spyder

3.3 PROGRAMACIÓN CON OPENCV

Como se ha comentado, la combinación de OpenCV junto con Python supone una poderosa herramienta para el desarrollo de aplicaciones de visión artificial. Dada la gran cantidad de funciones que permite la librería, muy relacionadas con los conceptos teóricos mencionados en el capítulo anterior, es conveniente comprobar su funcionamiento.

Para entender los aspectos técnicos de estas funciones y la programación en este lenguaje, en internet es posible encontrar gran variedad de recursos en forma de manuales, guías o tutoriales para todo tipo de usuarios. Por ejemplo, OpenCV dispone de sus propios manuales con ejemplos y descripciones teóricas [35]. La web "*GeeksforGeeks*" [36] dispone de una gran variedad de ejemplos de aplicación de multitud de funciones, al igual que en el blog "*Tutor de Programación*" [37]. Por otra parte, el doctor Adrian Rosebrock dispone de una gran cantidad de documentación gratuita en el campo de la visión artificial en su blog [38], destacando una guía para comenzar con el desarrollo de aplicaciones basadas en OpenCV [39].

El objetivo del documento no es realizar una guía exhaustiva acerca de la programación en este lenguaje. Aun así, con el propósito de mostrar las posibilidades que incorpora la librería, sin llegar a entrar en mucho detalle, a continuación, se muestran ejemplos de aplicación de algunas de las funciones disponibles, que incluyen la capacidad de abrir imágenes, modificarlas de distintas maneras o procesarlas.

3.3.1 Operaciones básicas

3.3.1.1 Leer, guardar y mostrar imágenes

En las primeras líneas del código siempre será necesario importar los paquetes necesarios que incluyen todas las funciones que se utilizarán en adelante. Para el caso de la librería de OpenCV, se debe importar el paquete `cv2.`

La primera función que se debe conocer es aquella que nos permite cargar imágenes en el programa. La función `imread()` se encarga de cargar una imagen desde una ruta seleccionada. También es posible realizar cualquier modificación, como cambiar la extensión del archivo, y guardar el resultado a través de la función `imwrite()`. Por último, la función `imshow()` es la encargada de visualizar la imagen en una ventana flotante.

Mediante la función `waitKey()` se impide a la ventana cerrarse hasta que se presione una tecla o pase un tiempo determinado por el programador. La función `destroyAllWindows()` cierra todas las ventanas flotantes abiertas.

```
import cv2

# Cargamos la imagen en memoria
image = cv2.imread("prueba_defectos.jpg")
# Guardamos la imagen con el formato PNG
cv2.imwrite("prueba_defectos.png", image)
# Mostramos la imagen en pantalla
cv2.imshow("Imagen original", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figura 39. Fragmento de código correspondiente a la apertura, guardado y visualización de una imagen

El resultado de ejecución de este código (Figura 39) nos muestra una imagen por pantalla y guarda en la misma ruta la imagen con otra extensión. En la Figura 40 se muestra la imagen ejemplo con la que se trabajará a partir de este momento.

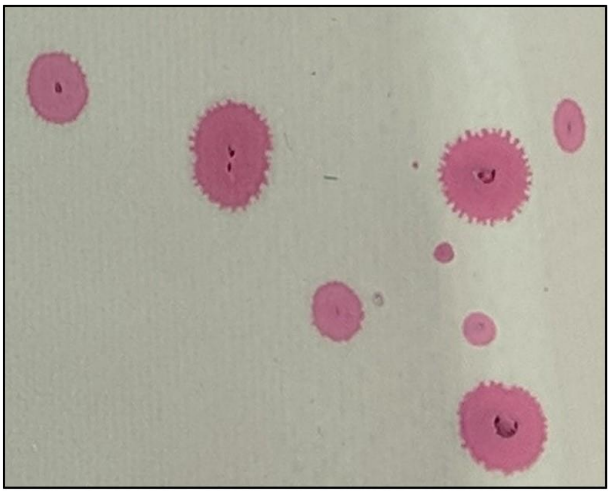


Figura 40. Imagen original mostrada

3.3.1.2 Propiedades de una imagen

Una vez cargada la imagen, es posible obtener sus dimensiones a través de la función `shape`. Esta función devuelve un vector de tres componentes: alto, ancho y fondo. El fondo hace referencia al número de canales, que para el caso de imágenes a color es 3.

También es posible realizar diferentes transformaciones a la imagen, como redimensionarla o girarla. Para el caso del redimensionado se puede hacer uso de la función `resize()`, a la que se debe introducir los nuevos valores de alto y ancho de la imagen, o un factor de escala para cada dimensión. Igualmente es necesario indicar el tipo de interpolación que utilizará la función para realizar la operación.

Otra operación interesante es la posibilidad de acceder a una región de interés concreta de la imagen. Para acceder a esta región concreta debemos indicar sus coordenadas en *x* e *y* de la manera siguiente: [*y0:y1, x0:x1*].

```
import cv2

# Cargamos la imagen en memoria
image = cv2.imread("prueba_defectos.jpg")
# Obtenemos las dimensiones de la imagen
(h, w, d) = image.shape
print("ancho={}, alto={}, fondo={}".format(w, h, d))
# Redimensionamos la imagen al 70%
red = cv2.resize(image, None, fx=0.70, fy=0.70,
                 interpolation=cv2.INTER_CUBIC)
cv2.imshow("Imagen 70%", red)
# Mostramos una región concreta de la imagen
roi = image[100:300, 200:400]
cv2.imshow("ROI",roi)
# Mostramos la imagen original
cv2.imshow("Imagen original", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figura 41. Fragmento de código correspondiente al escalado y extracción de una región

En la Figura 42 se muestra la comparativa de las tres imágenes resultantes de la ejecución del código anterior (Figura 41).

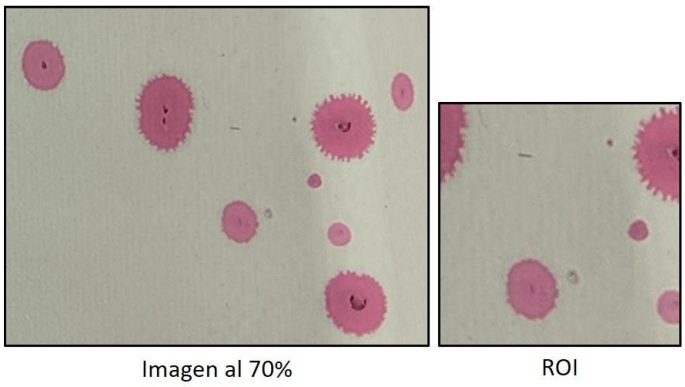


Figura 42. Escalado y región de interés de una imagen

3.3.2 Procesado de imágenes

3.3.2.1 Niveles de gris

Por cuestiones de programación de la librería, OpenCV trabaja con imágenes en el formato BGR (*Blue Green Red*) por lo que en ocasiones es necesario cambiar su representación al formato estándar RGB (*Red Green Blue*) o lo que suele ser más común, convertir la imagen a escala de grises. La función `cvtColor()` permite realizar estos cambios de formato y a través del parámetro `COLOR_BGR2GRAY`, se pasa de una imagen BGR a una imagen en niveles de gris.

```
import cv2

# Cargamos la imagen en memoria
image = cv2.imread("prueba_defectos.jpg")
# Convertimos a escala de grises
gris = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow("Imagen en niveles de gris", gris)
# Mostramos la imagen original
cv2.imshow("Imagen original", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figura 43. Fragmento de código correspondiente a la conversión a escala de grises

El resultado del código adjunto (Figura 43) permite visualizar una imagen por pantalla y su conversión a escala de grises, como se puede ver en la Figura 44

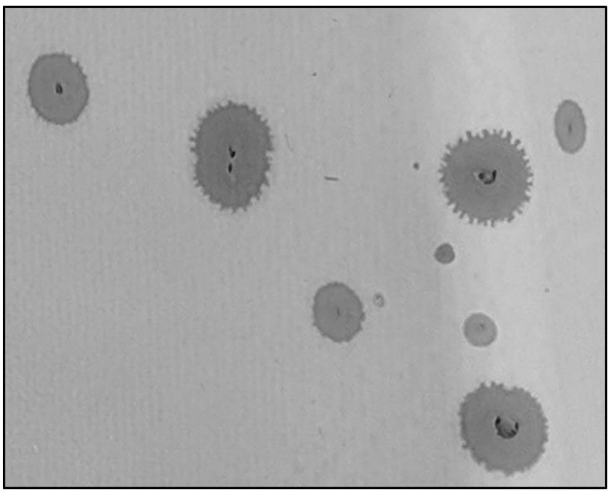


Figura 44. Conversión a niveles de gris

3.3.2.2 Filtros

En ocasiones es necesario aplicar filtros de forma previa al análisis de las imágenes con el objetivo de reducir el ruido o simplemente suavizarla. Aunque es posible programar filtros personalizados, OpenCV dispone de varias funciones que se encargan de realizar estas operaciones. Dos de los filtros más habituales son el filtro de Gauss y el filtro de la mediana.

La función `GaussianBlur()` utiliza una ventana, máscara o *kernel* de tamaño *NxN* píxeles (p.ej. 5x5, 7x7, 9x9...), mientras que el parámetro de la función `medianBlur()` es *N*.

```
import cv2

# Cargamos la imagen en memoria
image = cv2.imread("prueba_defectos.jpg")
# Convertimos a escala de grises
gris = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow("Imagen en niveles de gris", gris)
# Filtro de Gauss
gauss = cv2.GaussianBlur(gris, (7,7), 0)
cv2.imshow("Imagen con filtro Gauss", gauss)
# Filtro de la mediana
mediana = cv2.medianBlur(gris, 7)
cv2.imshow("Imagen con filtro de la mediana", mediana)
# Mostramos la imagen original
cv2.imshow("Imagen original", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figura 45. Fragmento de código correspondiente a la aplicación de filtros

La ejecución de este código (Figura 45) no solo nos muestra una imagen y su conversión a niveles de gris por pantalla, sino también el resultado de aplicación de los filtros mencionados. En la Figura 46 se puede observar la diferencia entre los filtros.

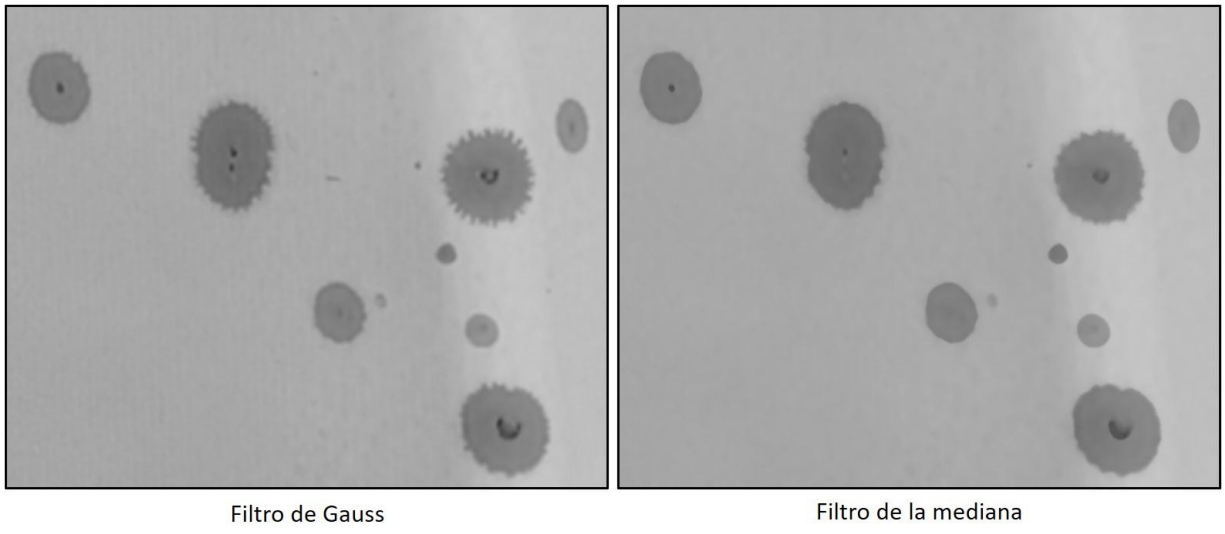


Figura 46. Aplicación de filtros de suavizado

3.3.2.3 Obtención de histogramas

OpenCV también nos permite obtener el histograma de una imagen a través de la función `calcHist()` y utilizando las herramientas gráficas de la librería de *matplotlib*.

```
import cv2
from matplotlib import pyplot as plt

# Cargamos la imagen en memoria
image = cv2.imread("prueba_defectos.jpg")
cv2.imshow("Imagen original", image)
# Convertimos a escala de grises
gris = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow("Imagen en niveles de gris", gris)
# Obtenemos el histograma de la imagen
hist = cv2.calcHist([gris], [0], None, [256], [0, 256])
# Graficamos el histograma
plt.plot(hist, color="gray")
plt.xlabel("Intensidad de iluminación")
plt.ylabel("Cantidad de píxeles")
plt.show()
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figura 47. Fragmento de código correspondiente a la obtención de histogramas

Una vez ejecutado el código (Figura 47) se visualiza tanto la imagen original como su conversión a niveles de gris (Figura 44) y además se grafica el histograma mostrado en la Figura 48. Como se puede apreciar, el histograma nos indica que la mayoría de los píxeles posee un nivel de intensidad de aproximadamente 170. Esto es lógico puesto que, en la imagen en niveles de gris, la mayor parte de los píxeles se corresponden con el fondo.

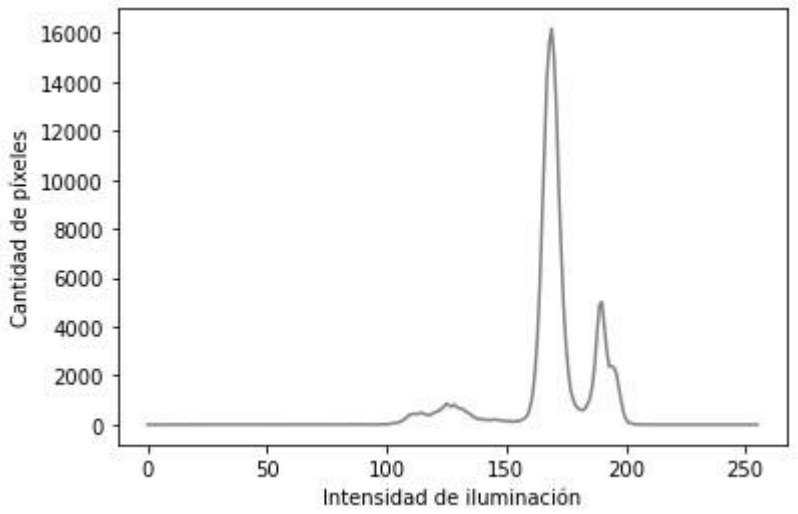


Figura 48. Histograma de la imagen en niveles de gris

También es posible obtener el histograma de imágenes en color. En este caso se obtendría una representación de la frecuencia con la que aparecen los distintos niveles de color para cada uno de los canales RGB.

3.3.2.4 Ecualización del histograma

Es posible realizar una ecualización del histograma de una imagen en niveles de gris a través de la función `equalizeHist()`.

```
import cv2

# Cargamos la imagen en memoria
image = cv2.imread("prueba_defectos.jpg")
# Convertimos a escala de grises
gris = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow("Imagen en niveles de gris", gris)
# Ecualización del histograma
eq = cv2. equalizeHist(gris)
cv2.imshow("Imagen con histograma ecualizado", eq)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figura 49. Fragmento de código correspondiente a la ecualización del histograma

Este código (Figura 49) permite obtener y visualizar el ecualizado del histograma de una imagen en niveles de gris, como se ve en la Figura 50. Además, siguiendo el ejemplo anterior se podría visualizar la gráfica del histograma ecualizado.

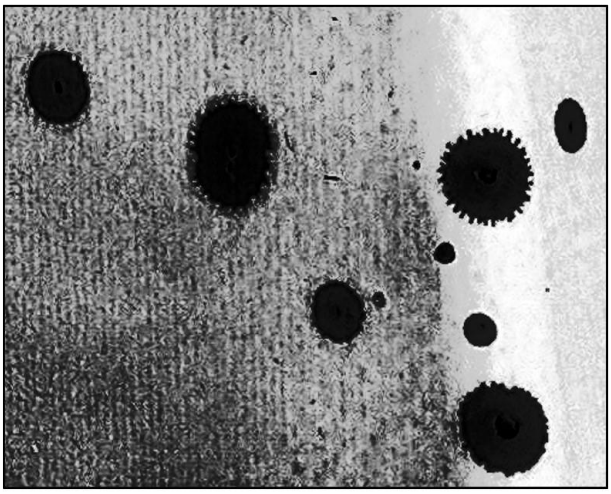


Imagen con el histograma ecualizado

Figura 50. Efecto del ecualizado del histograma

3.3.2.5 Umbralización

Como ya se ha comentado anteriormente. Una técnica muy común de segmentación es la umbralización. En OpenCV es posible aplicar dos de los métodos más comunes que son la umbralización fija y la umbralización adaptable.

La función `threshold()` permite realizar una umbralización fija definiendo un umbral entre un valor mínimo y un valor máximo. Se pueden aplicar cinco tipos distintos según sea el resultado al que se desea llegar, siendo el más común el umbral binario (`THRESH_BINARY`) o el umbral binario invertido (`THRESH_BINARY_INV`).

Por otro lado, la función `adaptiveThreshold()` aplica un umbral variable a la imagen. En ocasiones es difícil seleccionar el valor para el umbral fijo y se puede perder detalle de

determinadas áreas de la imagen, la aplicación de un umbral variable puede solucionar este problema.

```
import cv2

# Cargamos la imagen en memoria
image = cv2.imread("prueba_defectos.jpg")
# Convertimos a escala de grises
gris = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow("Imagen en niveles de gris", gris)
# Filtro de Gauss
gauss = cv2.GaussianBlur(gris, (7,7), 0)
cv2.imshow("Imagen con filtro Gauss", gauss)
# Umbral fijo
_, umb = cv2.threshold(gauss, 150, 255, cv2.THRESH_BINARY)
cv2.imshow("Imagen con umbral fijo", umb)
# Umbral adaptable
umb_ad = cv2.adaptiveThreshold(gauss, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                               cv2.THRESH_BINARY, 11, 2)
cv2.imshow("Imagen con umbral adaptable", umb_ad)
# Mostramos la imagen original
cv2.imshow("Imagen original", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figura 51. Fragmento de código correspondiente a la umbralización

El resultado de los dos tipos de umbralización implementados en el código anterior (Figura 51) se muestra en la Figura 52. Como se puede apreciar, la diferencia es notable.

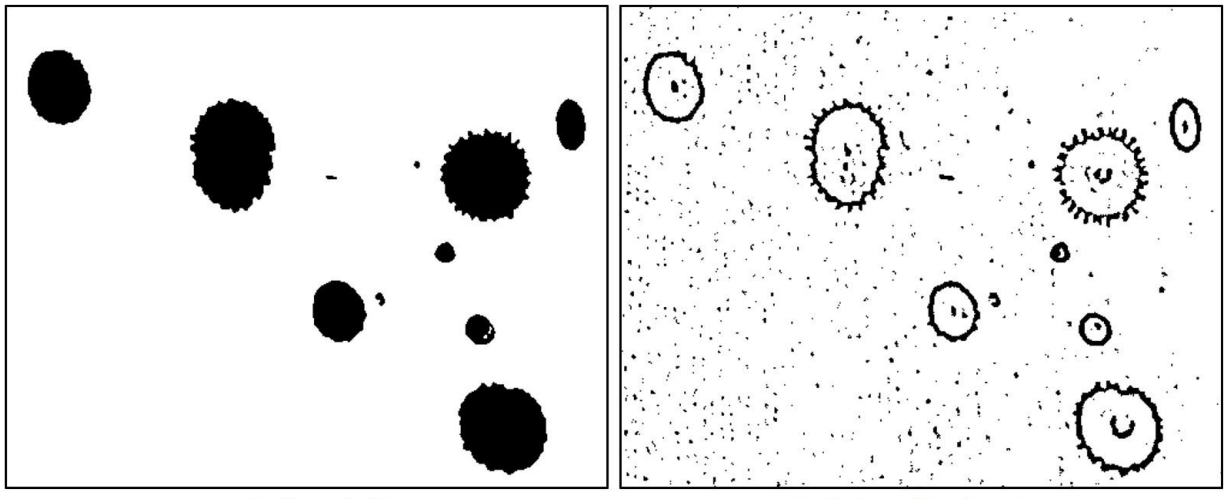


Figura 52. Efecto de distintos tipos de umbralización

Dependiendo de la aplicación deseada, será conveniente utilizar un tipo de umbralización u otro. En ocasiones la utilización de un umbral fijo permite discriminar más fácilmente ciertas zonas de la imagen cuya importancia sea menor.

3.3.2.6 Operaciones morfológicas

También es posible realizar transformaciones de la forma de los elementos de una imagen a través de la aplicación de una máscara o *kernel*, como se vio en el capítulo anterior. La erosión y la dilatación son las dos básicas, mientras que la apertura y el cierre son variantes de las primeras.

Para aplicar estas operaciones se suele partir de imágenes binarias, por lo que de forma previa se puede realizar un umbralizado para obtener una imagen de estas características. Como se desea que los elementos sean "blancos", el método de umbralización es el umbral binario invertido.

En primer lugar, es necesario declarar el *kernel* que se utilizará, siendo en este caso una matriz unidad 7x7. La función `erode()` realiza una erosión, y la función `dilate()` una dilatación. Se puede observar que también es posible indicar el número de iteraciones que se realizarán, esto hace referencia al número de pasadas que realizará el *kernel*.

La función `morphologyEx()` nos permite realizar operaciones más complejas como la apertura (`MORPH_OPEN`), que consiste en una erosión seguida de una dilatación o el cierre (`MORPH_CLOSE`), que es la operación contraria.

```
import cv2
import numpy as np

# Cargamos la imagen en memoria
image = cv2.imread("prueba_defectos.jpg")
# Convertimos a escala de grises
gris = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow("Imagen en niveles de gris", gris)
# Filtro de Gauss
gauss = cv2.GaussianBlur(gris, (7,7), 0)
cv2.imshow("Imagen con filtro Gauss", gauss)
# Umbral fijo
_, umb = cv2.threshold(gauss, 150, 255, cv2.THRESH_BINARY_INV)
cv2.imshow("Imagen con umbral fijo", umb)
# Realizamos una erosión
kernel = np.ones((7,7),np.uint8)
erosion = cv2.erode(umb, kernel,iterations = 1)
cv2.imshow("Imagen con erosion", erosion)
# Realizamos una dilatación
dilation = cv2.dilate(umb, kernel,iterations = 1)
cv2.imshow("Imagen con dilatacion", dilation)
# Realizamos una apertura
opening = cv2.morphologyEx(umb, cv2.MORPH_OPEN, kernel)
cv2.imshow("Imagen con apertura", opening)
# Realizamos un cierre
closing = cv2.morphologyEx(umb, cv2.MORPH_CLOSE, kernel)
cv2.imshow("Imagen con cierre", closing)
# Mostramos la imagen original
cv2.imshow("Imagen original", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figura 53. Fragmento de código correspondiente a las operaciones morfológicas

Mediante la ejecución del código (Figura 53) se puede observar el resultado de la aplicación de cada una de estas operaciones.

Las diferencias entre las operaciones de erosión y de dilatación se observan en la Figura 54. La erosión reduce los bordes de los elementos en blanco, por lo que puede ser útil

para eliminar cierto ruido en la imagen, mientras que la dilatación los amplía, lo que permite unir bordes.

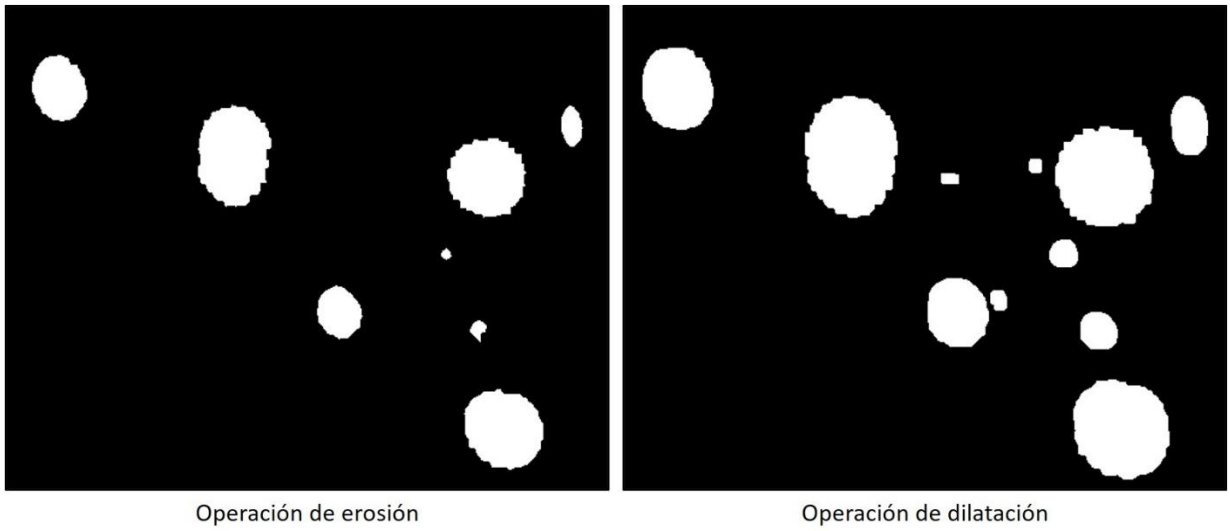


Figura 54. Efecto de las operaciones de erosión y dilatación

Por otro lado, el resultado de las operaciones de cierre y apertura se muestra en la Figura 55. En el caso de la apertura, es útil para reducir el ruido sin modificar excesivamente el borde de los elementos, la apertura se suele utilizar para cerrar pequeños agujeros que pueden darse en el interior de los elementos.

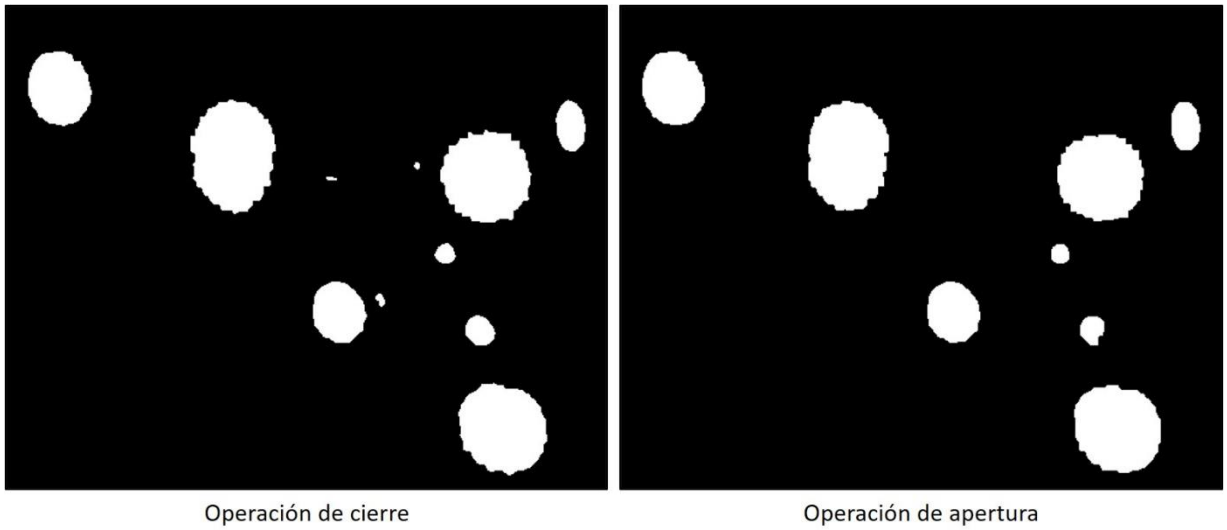


Figura 55. Efecto de las operaciones de cierre y apertura

3.3.3 Extracción y detección de contornos

3.3.3.1 Detector de bordes de Canny

Un algoritmo muy popular de detección de contornos es el de Canny. El algoritmo consta de varios pasos, pero OpenCV los agrupa todos en la función de `Canny()`. Internamente el algoritmo aplica una serie de filtros para reducir el ruido de la imagen y eliminar aquellos elementos que no considera que son bordes.

```
import cv2

# Cargamos la imagen en memoria
image = cv2.imread("prueba_defectos.jpg")
# Convertimos a escala de grises
gris = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow("Imagen en niveles de gris", gris)
# Filtro de Gauss
gauss = cv2.GaussianBlur(gris, (7,7), 0)
cv2.imshow("Imagen con filtro Gauss", gauss)
# Detectamos bordes por Canny
canny = cv2.Canny(gauss, 75, 100)
cv2.imshow("Bordes de Canny", canny)
# Mostramos la imagen original
cv2.imshow("Imagen original", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figura 56. Fragmento de código correspondiente al detector de bordes de Canny

En la Figura 57 se observa el resultado de aplicación del algoritmo (Figura 56) a una imagen suavizada mediante un filtro de Gauss. Este algoritmo se puede combinar con las operaciones vistas anteriormente para obtener los resultados deseados.

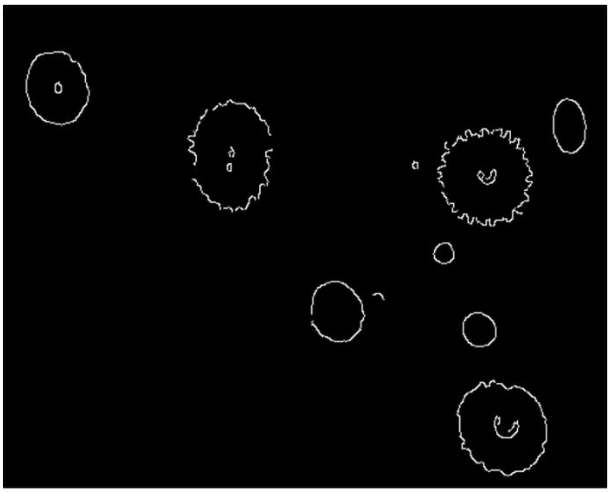


Figura 57. Detección de bordes mediante el algoritmo de Canny

3.3.3.2 Detección de contornos

También es posible buscar contornos en la imagen a través de la función `findContours()`. Es necesario partir de una imagen binaria, lo que se consigue a través de un umbralizado. En ocasiones la selección del valor del umbral no es obvia y a través de `THRESH_TRIANGLE`, OpenCV es capaz de elegir el umbral óptimo.

Para obtener los contornos es necesario indicar a la función dos parámetros: la jerarquía seguida para descartar contornos y el método usado para simplificar el contorno obtenido. Las elecciones más comunes son `RETR_TREE` o `RETR_CCOMP` para el primero y `CHAIN_APPROX_SIMPLE` para el segundo.

A la hora de visualizar los contornos en la imagen existen varias alternativas. La primera es dibujar el propio contorno en la imagen. Puesto que en ocasiones se detectan contornos

no deseados, ya sea porque son elementos demasiado grandes o muy pequeños, se puede discretizar la visualización indicando un rango de valores aceptables respecto al área del contorno, obtenida a través de la función `contourArea()`. Por ello esta operación es común verla incluida en un bucle *for* como se muestra en el ejemplo.

Por último, `drawContours()` es la función encargada de dibujar los contornos. Es posible indicar el tipo, grosor y color de la línea que representará los contornos.

```
import cv2

# Cargamos la imagen en memoria
image = cv2.imread("prueba_defectos.jpg")
# Convertimos a escala de grises
gris = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow("Imagen en niveles de gris", gris)
# Filtro de Gauss
gauss = cv2.GaussianBlur(gris, (7,7), 0)
cv2.imshow("Imagen con filtro Gauss", gauss)
# Umbral fijo
t, umb = cv2.threshold(gauss, 0, 255,
                       cv2.THRESH_BINARY_INV | cv2.THRESH_TRIANGLE)
cv2.imshow("Imagen con umbral fijo", umb)
# Buscamos contornos en la imagen
contours, _ = cv2.findContours(umb, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)
# Dibujamos los contornos
for c in contours:
    area = cv2.contourArea(c)
    if area > 100 and area < 100000:
        cv2.drawContours(image, [c], 0, (0, 255, 0), 2, cv2.LINE_AA)
# Mostramos la imagen original con contornos
cv2.imshow("Contornos en la imagen", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figura 58. Fragmento de código correspondiente a la detección y representación de contornos

El resultado de ejecución del código (Figura 58) se puede ver reflejado en la Figura 59. Gracias al haber indicado un rango de valores para el área, se han obtenido los contornos más representativos de la imagen.

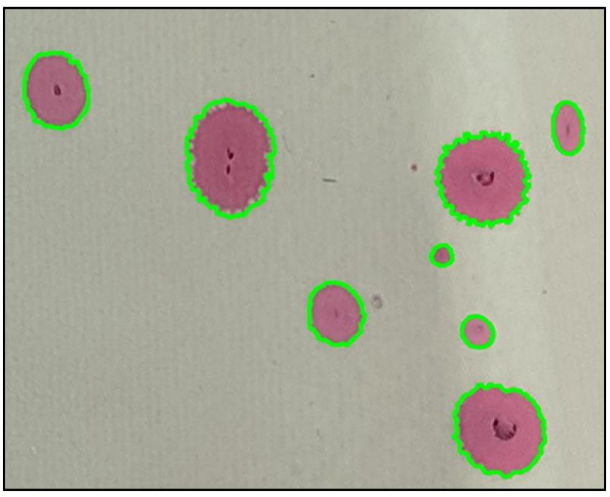


Figura 59. Detección de contornos y representación en la imagen

Una representación alternativa consiste en buscar el mínimo rectángulo que contenga el contorno. Para ello se utiliza la función `boundingRect()` que obtiene las medidas del rectángulo que posteriormente se dibuja a través de `rectangle()` de forma similar al ejemplo anterior.

```
import cv2

# Cargamos la imagen en memoria
image = cv2.imread("prueba_defectos.jpg")
# Convertimos a escala de grises
gris = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow("Imagen en niveles de gris", gris)
# Filtro de Gauss
gauss = cv2.GaussianBlur(gris, (7,7), 0)
cv2.imshow("Imagen con filtro Gauss", gauss)
# Umbral fijo
t, umb = cv2.threshold(gauss, 0, 255,
                       cv2.THRESH_BINARY_INV | cv2.THRESH_TRIANGLE)
cv2.imshow("Imagen con umbral fijo", umb)
# Buscamos contornos en la imagen
contours, _ = cv2.findContours(umb, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)
# Dibujamos los contornos mediante rectángulos
for c in contours:
    area = cv2.contourArea(c)
    if area > 100 and area < 100000:
        (x, y, w, h) = cv2.boundingRect(c)
        cv2.rectangle(image, (x, y), (x + w, y + h),
                      (0, 255, 0), 2, cv2.LINE_AA)
# Mostramos la imagen original con contornos
cv2.imshow("Contornos en la imagen", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figura 60. Fragmento de código correspondiente a la detección y representación mediante rectángulos

Como se puede observar en la Figura 61, se han encontrado los mismos elementos que para el caso anterior pero su representación es distinta, simplemente modificando unas pocas líneas del código (Figura 60).

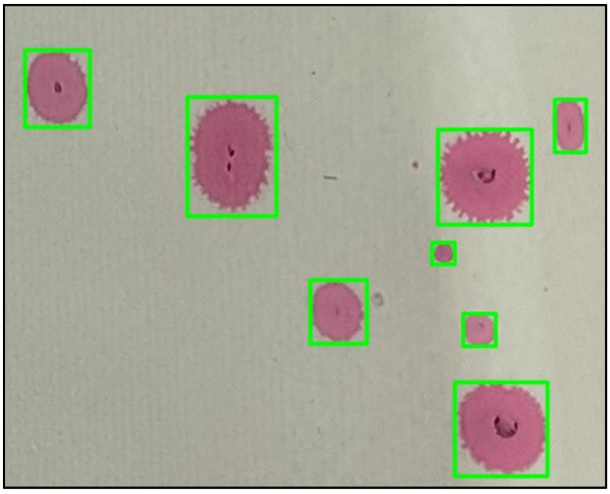


Figura 61. Detección de contornos y representación mediante rectángulos

Otra opción muy similar es la de obtener la representación del contorno mediante la mínima circunferencia que contenga a cada contorno. Para obtener el centro y el radio de

cada circunferencia se utiliza la función `minEnclosingCircle()` y para dibujarla la función `circle()`.

```
import cv2

# Cargamos la imagen en memoria
image = cv2.imread("prueba_defectos.jpg")
# Convertimos a escala de grises
gris = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2.imshow("Imagen en niveles de gris", gris)
# Filtro de Gauss
gauss = cv2.GaussianBlur(gris, (7,7), 0)
cv2.imshow("Imagen con filtro Gauss", gauss)
# Umbral fijo
t, umb = cv2.threshold(gauss, 0, 255,
                       cv2.THRESH_BINARY_INV | cv2.THRESH_TRIANGLE)
cv2.imshow("Imagen con umbral fijo", umb)
# Buscamos contornos en la imagen
contours, _ = cv2.findContours(umb, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)
# Dibujamos los contornos mediante círculos
for c in contours:
    area = cv2.contourArea(c)
    if area > 100 and area < 100000:
        (x, y), radius = cv2.minEnclosingCircle(c)
        center = (int(x), int(y))
        radius = int(radius)
        cv2.circle(image, center, radius, (0, 255, 0), 2)
# Mostramos la imagen original con contornos
cv2.imshow("Contornos en la imagen", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figura 62. Fragmento de código correspondiente a la detección y representación mediante circunferencias

A través de la ejecución del nuevo código (Figura 62), se puede apreciar como el resultado sigue siendo el mismo que en los casos anteriores, pero en la Figura 63 la detección se representa mediante circunferencias.

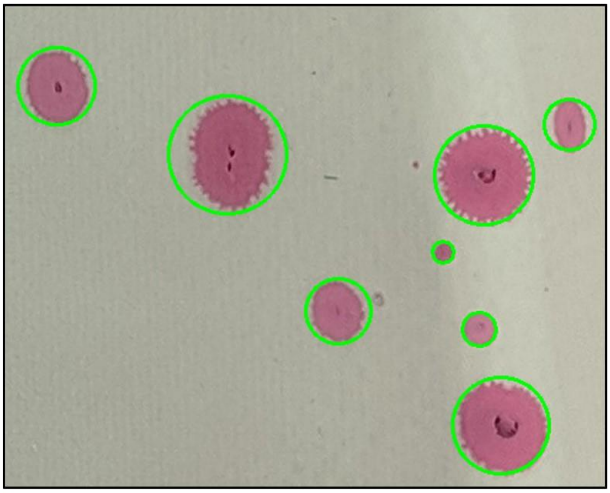


Figura 63. Detección de contornos y representación mediante circunferencias

3.4 PRESENTACIÓN DE RESULTADOS

A la hora de obtener resultados en aplicaciones de visión artificial y detección de objetos en generales, existen una serie de conceptos que permiten evaluar el desempeño de los algoritmos.

Las muestras o imágenes utilizadas para realizar la evaluación contendrán un conjunto de imágenes positivas (*true examples*) y un conjunto de imágenes negativas (*false examples*). Estas imágenes podrán ser clasificadas como positivas (*classified positives*) o negativas *(classified negatives*), obteniendo cuatro tipos de resultados:

- *True positives* o **verdaderos positivos**: positivas clasificadas como positivas.
- *False positives* o **falsos positivos**: negativas clasificadas como positivas.
- *False negatives* o **falsos negativos**: positivas clasificadas como negativas.
- *True negatives* o **verdaderos negativos**: negativas clasificadas como negativas.

Tabla 6. Tabla de clasificación de resultados

	Classified positives	***Classified negatives***
True examples	TP = *True positives*	FN = *False negatives*
False examples	FP = *False positives*	TN = *True negatives*

De esta clasificación se pueden obtener los siguientes parámetros, que hacen referencia al total de muestras:

- **Total de muestras positivas** (NP): suma de verdaderos positivos y falsos negativos.

$$NP = TP + FN \tag{11}$$

- **Total de muestras negativas** (NN): suma de falsos positivos y verdaderos negativos.

$$NN = FP + TN \tag{12}$$

- **Total de clasificados positivos** (CP): suma de verdaderos positivos y falsos positivos.

$$CP = TP + FP \tag{13}$$

- **Total de clasificados negativos** (CN): suma de falsos negativos y verdaderos negativos.

$$CN = FN + TN \tag{14}$$

Estos parámetros se pueden relacionar entre sí, obteniendo una serie de coeficientes que permiten evaluar los algoritmos:

- *True positive rate* o *detection rate*: **Tasa de detección** o **de verdaderos positivos**. Indica el número de detecciones positivas dentro de un conjunto de muestras positivas. También se denomina **sensibilidad** o **exhaustividad**.

$$\rho_{TP} = \frac{TP}{NP} \tag{15}$$

- *True negative rate* o *reject rate*: **Tasa de rechazo** o **de verdaderos negativos**. Indica el número de detecciones negativas dentro de un conjunto de muestras negativas. También se denomina **especificidad**.

$$\rho_{TN} = \frac{TN}{NN} \tag{16}$$

- *False positive rate*, *false alarm rate* o *type I error rate*: **Tasa de falsos positivos** o **de falsa alarma**. Indica el número de detecciones positivas dentro de un conjunto de muestras negativas.

$$\rho_{FP} = 1 - \rho_{TN} \tag{17}$$

- *False negative rate*, *miss rate* o *type II error rate*: **Tasa de falsos negativos** o **de pérdida**. Indica el número de detecciones negativas dentro de un conjunto de muestras positivas.

$$\rho_{FN} = 1 - \rho_{TP} \tag{18}$$

- *Ratio of negative examples to positive examples*: Permite relacionar el número de muestras negativas con el de muestras positivas.

$$r = \frac{NN}{NP} \tag{19}$$

- *Precision*: **Precisión**. Indica el número de verdaderos positivas dentro de los todos los clasificados positivos.

$$P = \frac{TP}{CP} \tag{20}$$

En aquellas ocasiones que interese, es posible realizar una representación gráfica de la relación existente entre estos ratios:

- Curva ROC (*Receiver Operating Characteristic Curve*): relaciona la sensibilidad (*sensitivity*) con la especificidad (*specifity*). Lo más normal es representar la tasa de detección frente a la tasa de falsos positivos.
- Curva *Precision-Recall*: relaciona la precisión (*precision*) con la exhaustividad (*recall*).
- Curva DET (*Detection Error Trafe-off Curve*): relaciona la tasa de falsos negativos (*miss rate*) frente a la tasa de falsos positivos (*false alarm rate*). Suele realizarse una representación logarítmica.

CAPÍTULO 4: DETECCIÓN DE DEFECTOS CON ENSAYO PREVIO

La primera parte del desarrollo consiste en valorar la posibilidad de realizar una evaluación de las piezas inspeccionadas mediante el ensayo de líquidos penetrantes a través de algoritmos de visión artificial. Estos algoritmos permiten facilitar la identificación y contabilización de los defectos de forma automática.

4.1 REALIZACIÓN DEL ENSAYO

Se recuerda que este método consiste en la aplicación de un líquido llamado penetrante sobre la superficie de la pieza. Las características de este líquido le permiten penetrar en los defectos superficiales, y tras el tiempo de espera necesario y eliminación del exceso, la aplicación del revelador se encarga de absorber el líquido y de señalizar dichos defectos.

4.1.1 Productos necesarios

Además de otros materiales como trapos y papel absorbente, el proceso de inspección requiere fundamentalmente de tres elementos:

- Penetrante: líquido con las propiedades adecuadas para penetrar en los defectos de las piezas.
- Revelador: producto en polvo que reacciona con el líquido y se encarga de producir la marca que señaliza el defecto.
- Eliminador: líquido disolvente que permite limpiar la pieza y eliminar el exceso de penetrante.

Los productos que se utilizarán se corresponden a una de las líneas de la marca PFINDER CHEMIE. En la tabla adjunta se muestran los productos utilizados:

Penetrante **PFINDER 800** (aerosol de 500 ml)
Revelador **PFINDER 871** (aerosol de 500 ml)
Eliminador **PFINDER 890** (aerosol de 500 ml)

PFINDER CHEMIE

Figura 64. Aerosoles de PFINDER

4.1.2 Proceso de inspección

Lo primero que se debe realizar antes de la inspección es la **preparación** y **limpieza** de la pieza. Es muy importante que la superficie de la pieza se encuentre completamente libre de cualquier tipo de contaminante (óxidos, pinturas, grasas, etc.) que pueda impedir la correcta penetración del líquido en las discontinuidades.

En el caso concreto de los troqueles mecanizados en el que la superficie exhibe un elevado número de poros abiertos a la superficie, es preferible la utilización de procedimientos por vía química en lugar de mecánicos para evitar dañar la superficie de la pieza. En la Figura 65 se observa una de las piezas a analizar preparada y limpia para realizar la inspección.

Figura 65. Preparación y limpieza de la pieza

Para la **aplicación del penetrante** se dispone del líquido en formato aerosol. Su uso es muy sencillo, siendo necesario agitar el bote y rociar el penetrante sobre la superficie de la pieza, obteniendo una película fina y uniforme. Para que el penetrante actúe correctamente es necesario dejarlo actuar, esperando un tiempo que suele rondar entre los 5 y los 15 minutos, dependiendo tanto del penetrante utilizado como del material y los defectos que se desean inspeccionar. La Figura 66 muestra el resultado de esta etapa.

Figura 66. Aplicación del penetrante

Una vez esperado el tiempo de actuación, es necesario eliminar el exceso de líquido que permanece sobre la superficie de la pieza. Es importante la correcta realización de esta etapa, puesto que es importante no extraer el líquido que ha penetrado en las discontinuidades.

Para realizar la **eliminación del exceso** es aconsejable retirar la mayor parte del penetrante con un trapo o papel absorbente seco y después hacer uso del eliminador aplicándolo sobre un trapo y nunca directamente sobre la pieza para retirar el líquido restante.

Una vez lista la pieza, es posible proceder a la **aplicación del revelador**, que permite sacar a la superficie la indicación de los defectos. Para ejecutar correctamente esta etapa, es importante realizar trayectorias continuas y paralelas manteniendo una distancia nunca inferior a 20-30 cm para lograr una capa uniforme y homogénea. Igual que para la aplicación del revelador, es necesario esperar un tiempo (entre 5 y 15 minutos) para que actúe. El resultado se ilustra en la Figura 67.

Figura 67. Aplicación del revelador e inspección de la pieza

Según el revelador va absorbiendo el penetrante, el área de las superficies mostradas aumenta de forma directamente proporcional al volumen de líquido contenido en el defecto. De esta forma es posible realizar la **inspección de la pieza**, determinando el tamaño de los defectos. En esta etapa es importante disponer de los medios y conocimientos necesarios para juzgar la gravedad de los defectos y diferenciar entre las discontinuidades consideradas defectos y las que no.

Una vez realizado todo el proceso de inspección es conveniente volver a utilizar el eliminador para eliminar los restos de penetrante y revelador, ayudándose en caso necesario de trapos o de papel absorbente.

Una de las desventajas del ensayo de líquidos penetrantes supone el riesgo de contaminación de la pieza tanto por parte del penetrante como del revelador, siendo además bastante complejo realizar su eliminación completa. Por esta razón, el desarrollar una metodología que permita prescindir de este ensayo para realizar la detección de defectos sería muy apropiado para el caso de los troqueles de estampación y supondría una alternativa muy interesante como se podrá ver en el capítulo siguiente.

4.2 ALGORITMO DE DETECCIÓN DE CONTORNOS

Como ya se ha mencionado, el primer problema por resolver es la detección de defectos en piezas una vez realizado el ensayo de líquidos penetrantes. El resultado del ensayo resalta los defectos en rosa, proporcionando una indicación de su tamaño y profundidad.

Algunos fragmentos de las imágenes a tratar, correspondientes a las piezas tras realizar el ensayo de líquidos penetrantes se pueden observar en la Figura 68.

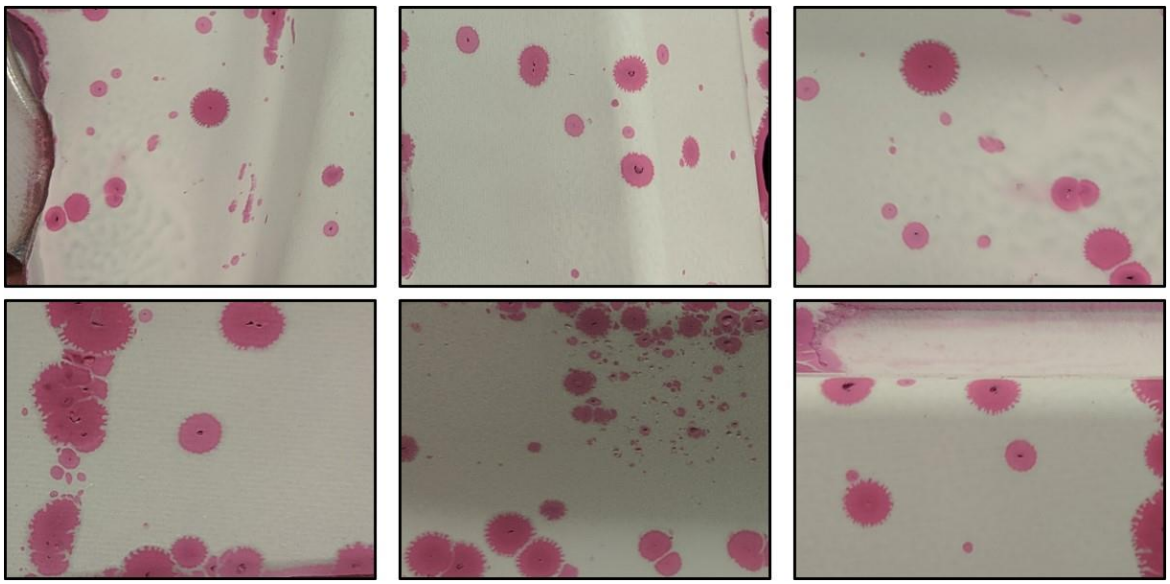


Figura 68. Ejemplos de fragmentos de imágenes a tratar

Una forma de abordar dicho problema es la programación del código únicamente utilizando algoritmos de visión artificial como los mostrados anteriormente, siendo el objetivo final del algoritmo separar los elementos en rosa (correspondientes a los defectos) del fondo blanco de la pieza.

4.2.1 Desarrollo del algoritmo

Para el desarrollo se ha trabajado diseñando una sencilla interfaz para facilitar el acceso a las distintas imágenes. Además, esta interfaz incorpora comandos para realizar algunas de las operaciones vistas en el apartado anterior. Puesto que su aplicación es meramente visual y no aporta un desarrollo técnico excesivo, no se explicará el código correspondiente a dicha interfaz, quedando reflejado en su propio anexo.

A continuación, se procede a describir las funciones que intervienen en el algoritmo de detección de contornos.

En primer lugar, se dispone de la función `detectarManual()`, cuyo código se recoge en la Figura 69. Básicamente esta función se encarga de solicitar al usuario la apertura de una imagen concreta, redimensionarla y posteriormente llamar a la función que se encargará de la detección de bordes. Es posible trabajar con una gran variedad de extensiones, lo que le aporta mayor flexibilidad al código y facilita el trabajo al usuario, evitando tener que realizar una conversión previa.

```python
# FUNCIÓN DETECCIÓN EN UNA SOLA IMAGEN
def detectarManual(self):
    global ruta_imagen, image, img

    # Especificamos los tipos de archivos correspondientes a imagenes
    ruta_imagen = filedialog.askopenfilename(title="Abrir imagen", filetypes = [
        ("Archivos de mapa de bits (*.bmp, *.dib)","*.bmp *.dib"),
        ("JPEG (*.jpg, *.jpeg, *.jpe, *.jfif)","*.jpg *.jpeg *.jpe *.jfif"),
        ("PNG (*.png)","*.png"),
        ("Todos los archivos","*.bmp *.dib *.jpg *.jpeg *.jpe *.jfif *.png")])
    # Si la ruta es correcta, leemos la imagen
    if len(ruta_imagen) > 0:
        # Leemos y redimensionamos la imagen
        img = cv2.imread(ruta_imagen)
        img = imutils.resize(img, height=min(800, img.shape[0]))
        # Llamamos a la función
        self.FuncionDetectar()
        cv2.waitKey(0)
        cv2.destroyAllWindows()
    # Aviso en caso de ruta incorrecta
    else:
        tk.messagebox.showerror("Error","Ninguna imagen seleccionada.")
```

Figura 69. Fragmento de código correspondiente a la función detectarManual()

El grueso de esta parte del algoritmo se encuentra en la función `FuncionDetectar()`. Esta función se encarga de realizar el pre-procesado, la segmentación y la clasificación de la imagen. El código correspondiente se encuentra más adelante en la Figura 77. Para su desarrollo se han seguido etapas similares a las propuestas por Chu & Wang [18], Sun et al. [28] o Song et al. [19].

En cuanto al **pre-procesado**, es importante en primer lugar realizar la conversión de la imagen a niveles de gris, puesto que es mucho más sencillo trabajar con ese tipo de imágenes en visión artificial. También se debe reducir el ruido de la imagen mediante filtros.

Uno de los más usados en este tipo de aplicaciones es el filtro de Gauss, que consiste en aplicar la función de Gauss en forma de matriz *NxM* a un determinado número de píxeles. Los valores *N* y *M* de la matriz deberán ser números impares. Aunque no es necesario que sean del mismo valor, suele ser lo más común.

En la Figura 70 se muestran ejemplos de estas matrices gaussianas que actuarán como *kernel* del filtro. Un suavizado excesivo puede llegar a provocar pérdida de información en la imagen o incluso puede aumentar el tiempo de procesado sin afectar a los resultados obtenidos. Seleccionar un tamaño de *kernel* mayor que 11x11 no suele ser lo recomendado, por lo que en este caso la elección de un *kernel* de 7x7 permite obtener buenos resultados.

$$\frac{1}{16}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix} \qquad \frac{1}{273}\begin{bmatrix}1&4&7&4&1\\4&16&26&16&4\\7&26&41&26&7\\4&16&26&16&4\\1&4&7&4&1\end{bmatrix} \qquad \frac{1}{1003}\begin{bmatrix}0&0&1&2&1&0&0\\0&3&13&22&13&3&0\\1&13&59&97&59&13&1\\2&22&97&159&97&22&2\\1&13&59&97&59&13&1\\0&3&13&22&13&3&0\\0&0&1&2&1&0&0\end{bmatrix}$$

Figura 70. Ejemplos de filtros de Gauus 3x3, 5x5 y 7x7

Los efectos producidos por la aplicación de distintas matrices gaussianas a la imagen de partida se observan en la Figura 71. Para tamaños de matriz bajos la diferencia no es muy notable, pero el efecto empieza a aumentar según aumentados el tamaño.

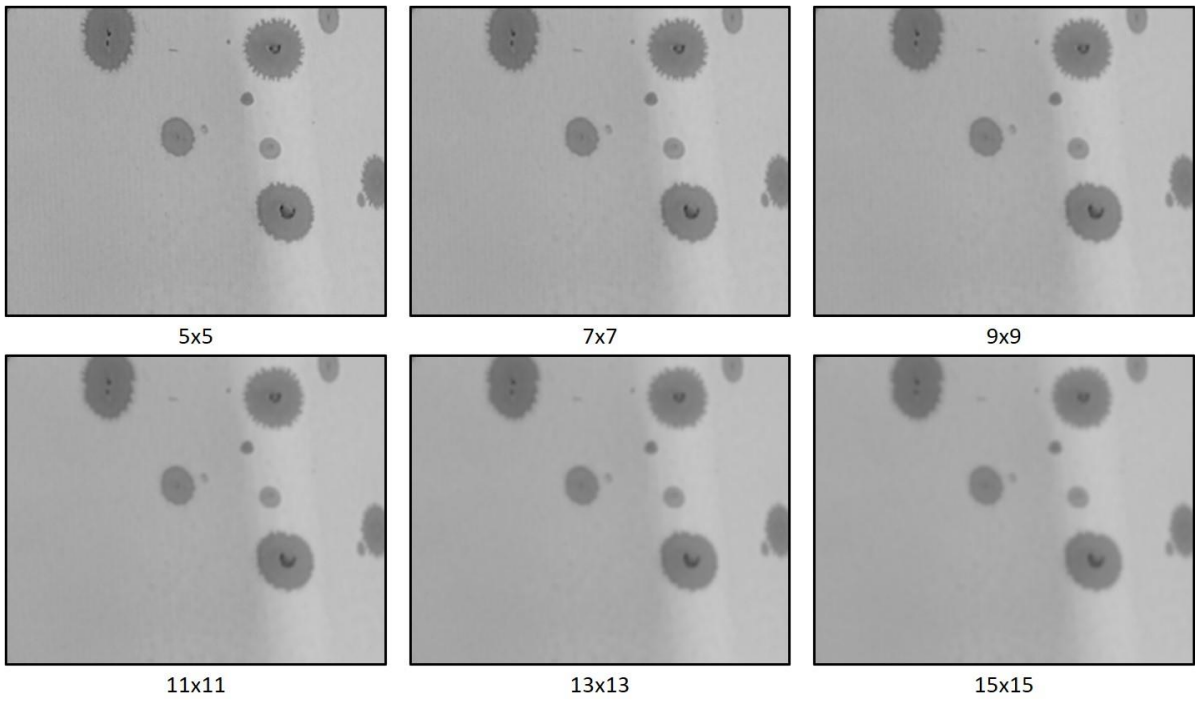


Figura 71. Efecto de distintos tamaños de matrices gaussianas

Para la etapa de **segmentación** se ha utilizado una umbralización fija, puesto que, en este caso la utilización de un umbral adaptativo no permitiría distinguir los elementos deseados del fondo. En la Figura 72 se aprecia la notable diferencia entre distintos valores del umbral.

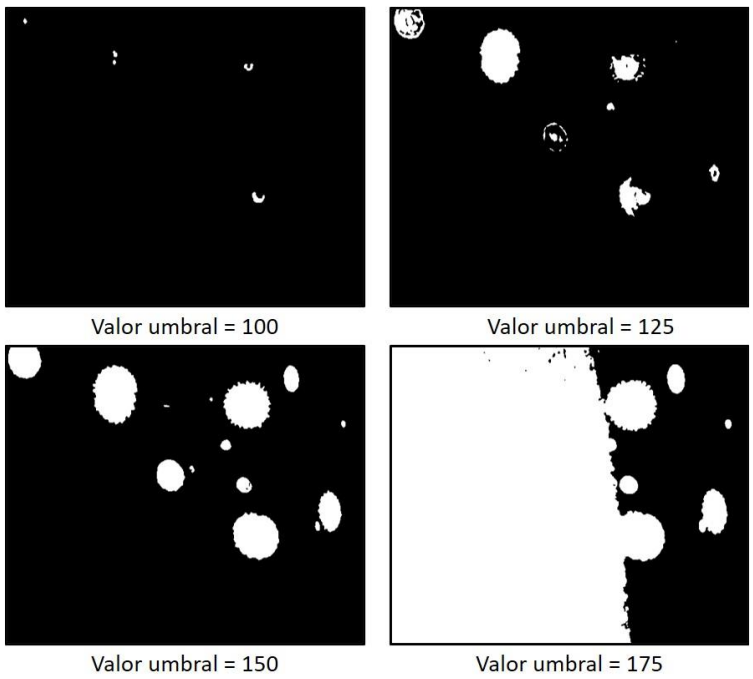


Figura 72. Efecto de la elección de distintos umbrales

Debido a la dificultad de seleccionar un valor de umbral adecuado y que sea apto para todo tipo de imágenes, existe la posibilidad de que la propia función seleccione el valor óptimo de dicho umbral. OpenCV proporciona dos métodos para la selección de este valor a través de los parámetros `THRESH_TRIANGLE` o `THRESH_OTSU`. Los diferentes resultados se observan en la Figura 73. En este caso, a través del método del triángulo se obtienen mejores resultados.

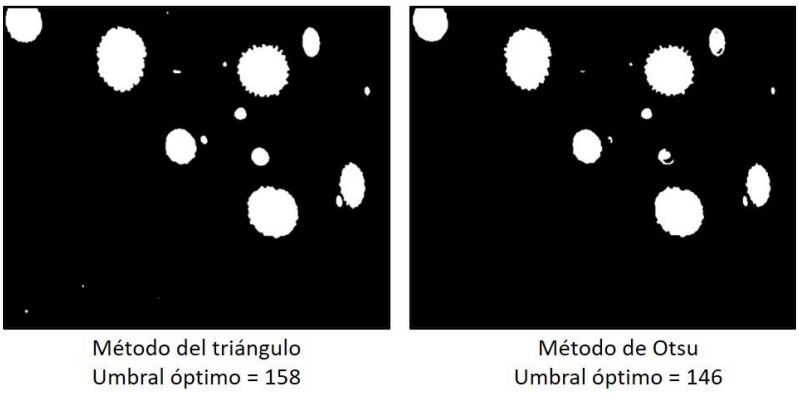


Figura 73. Umbral óptimo según el método escogido

Ya se ha comentado anteriormente la posibilidad de realizar distintas operaciones morfológicas para acentuar los elementos más significativos de una imagen, unir bordes o reducir el ruido. El resultado de cada una de estas operaciones una vez realizada la umbralización anterior se puede observar en la Figura 74.

En este caso se ha elegido finalmente realizar una operación de apertura mediante un *kernel* unidad de tamaño 5x5, lo que permite eliminar algo de ruido restante y acentuar algunos elementos sin modificar de forma excesiva su silueta.

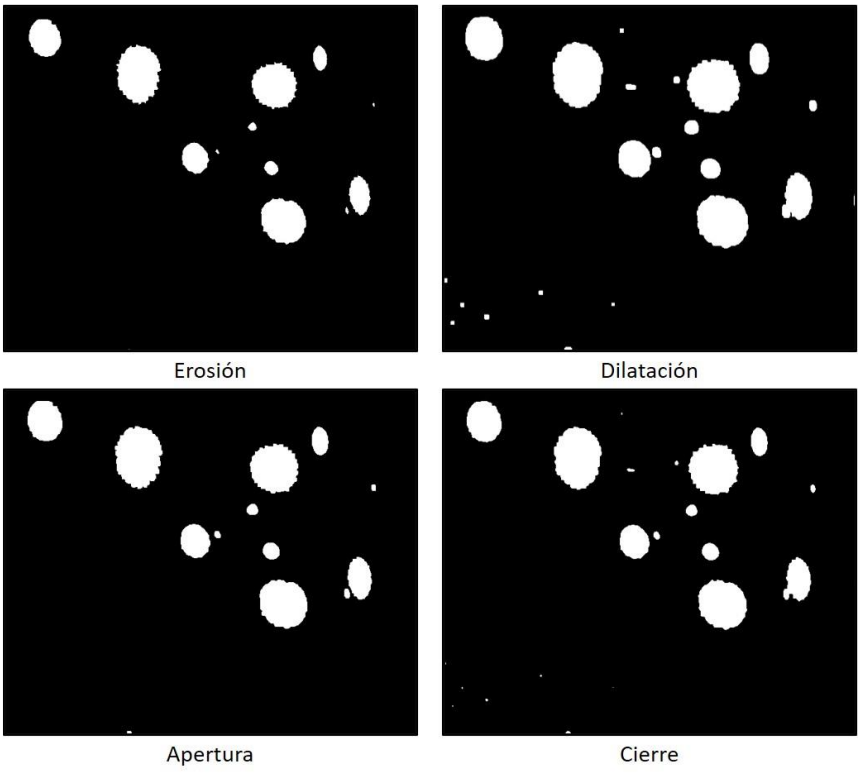


Figura 74. Efecto de las operaciones morfológicas

El último paso de esta etapa es la detección de contornos, que ya se explicó en el apartado anterior. En este caso, para su representación es más interesante utilizar la circunferencia circunscrita de cada elemento, puesto que permite obtener las coordenadas del centro de cada defecto y una medida aproximada de su tamaño a través del radio de la circunferencia

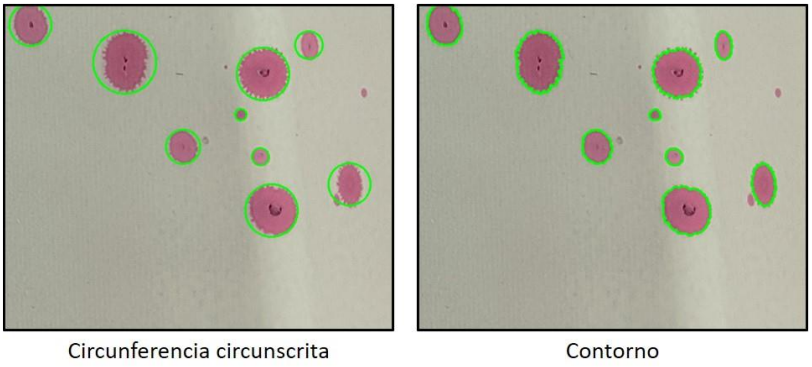


Figura 75. Diferentes formas de representación del elemento detectado

La etapa de **clasificación** en este caso es bastante básica. En primer lugar, se descartan aquellos elementos cuyo área sea demasiado pequeña o grande, puesto que se conoce el tamaño medio de los defectos. También se discrimina aquellos círculos cuyo radio sea superior a un valor determinado, evitando así la visualización de determinados contornos que no supongan ningún elemento concreto.

En la Figura 76 se tiene la referencia seguida para el cálculo del límite de tamaño de los elementos. La imagen de partida tiene una resolución espacial de 800 píxeles de ancho y 600 de alto, por lo que los elementos más grandes tienen un radio de unos 50 píxeles. Conviene destacar que conociendo la distancia a la que se captura la imagen y la resolución de esta, es posible relacionar los píxeles con las medidas reales.

Este tamaño de elemento no tiene por qué ser el mismo en todas las imágenes, por lo que se ha fijado el valor máximo de radio en 100 píxeles, descartando todos los elementos cuyo radio sea superior.

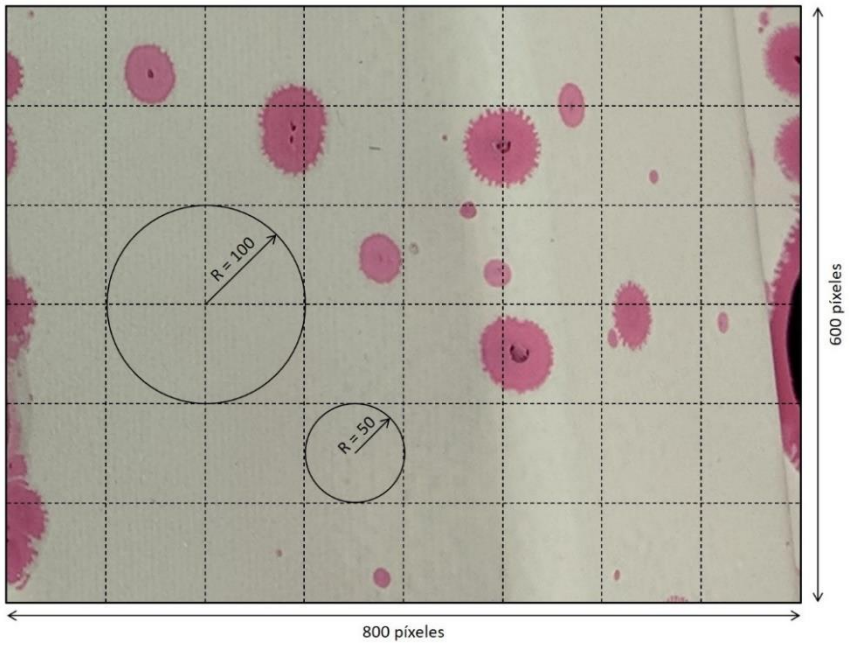


Figura 76. Aproximación del tamaño de los elementos

Cada círculo que cumple los requisitos para ser considerado como defecto, aumenta el valor del contador en una unidad. El número final de posibles detecciones aparecerá en la imagen final de detección. Además, se ha dotado al algoritmo la capacidad de imprimir por pantalla las dimensiones de los defectos encontrados.

Una vez elegidas las operaciones que realizará el algoritmo, el código resultante es el mostrado:

```
# APLICACIÓN DE FILTROS Y POSTERIOR DETECCIÓN DE CONTORNOS
def FuncionDetectar(self):

    # Conversión de la imagen a niveles de gris
    gris = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Aplicación de un filtro de Gauss para reducir el ruido
    gauss = cv2.GaussianBlur(gris, (7,7), 0)
    # Umbralización para obtener una imagen binaria
    _, umb = cv2.threshold(gauss, 0, 255, cv2.THRESH_BINARY_INV |
                           cv2.THRESH_TRIANGLE)
    # Operación de dilatación
    kernel = np.ones((5,5), np.uint8)
    opn = cv2.morphologyEx(umb, cv2.MORPH_OPEN, kernel)
    # Búsqueda de contornos en la imagen
    contours, _ = cv2.findContours(opn, cv2.RETR_CCOMP,
                                   cv2.CHAIN_APPROX_SIMPLE)
    # Inicializamos las detecciones a 0
    dets=0
    # Dibujo de círculos en aquellos lugares donde se han detectado contornos
    for c in contours:
        area = cv2.contourArea(c)
        if area > 100 and area < 100000:
            (x,y),radius = cv2.minEnclosingCircle(c)
            center = (int(x),int(y))
            radius = int(radius)
            if radius < 100:
                cv2.circle(img, center, radius, (0,255,0), 2)
                dets = dets + 1
                dim = "Defecto {}: (x, y)={} , \
                    R={}, area={}".format(dets,center,radius,area)
                print(dim)
    # Mostramos el número de detecciones en la imagen
    text1 = "Posibles defectos detectados: {}".format(dets)
    cv2.putText(img, text1, (20, img.shape[0]-10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6,(255,255,255),1)
    # Mostramos la imagen
    cv2.imshow("Detecciones", img)
```

Figura 77. Fragmento de código correspondiente a la función FuncionDetectar()

Todo el proceso interno de pre-procesado y segmentación seguido por el algoritmo se observa en la Figura 78. El resultado de estas etapas dependerá mucho de la calidad de la imagen original, y dada la gran variabilidad de las estas, en ocasiones no siempre será posible obtener los mejores resultados.

Comparando la Figura 78a y la Figura 78b, se puede observar fácilmente el efecto del filtrado mencionado anteriormente, estando la imagen más suavizada que en la original, permitiendo reducir la aparición de ruido, que es el objetivo principal de esta etapa.

Realizada la umbralización (Figura 78c) es posible distinguir algunos pequeños puntos en la imagen que podrían ocasionar detecciones erróneas por lo que mediante la operación de apertura (Figura 78d) se consiguen eliminar las más pequeñas.

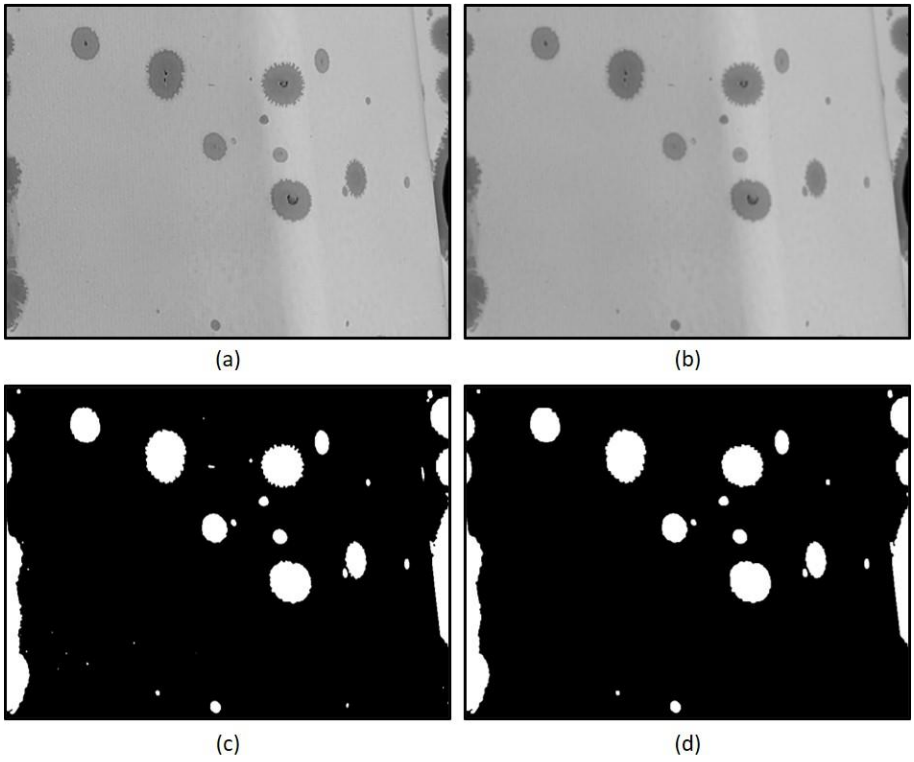


Figura 78. Conversión a niveles de gris (a), filtro de Gauss (b), umbralización fija (c) y apertura (d)

El resultado final de aplicación del algoritmo se puede observar en la Figura 79. Se han representado las circunferencias en verde para facilitar su visualización en la imagen. Como ya se ha mencionado, internamente se lleva la cuenta de los círculos dibujados que representan a cada defecto, y en la esquina inferior izquierda aparece el número de posibles defectos detectados, que en este caso son 15 detecciones.

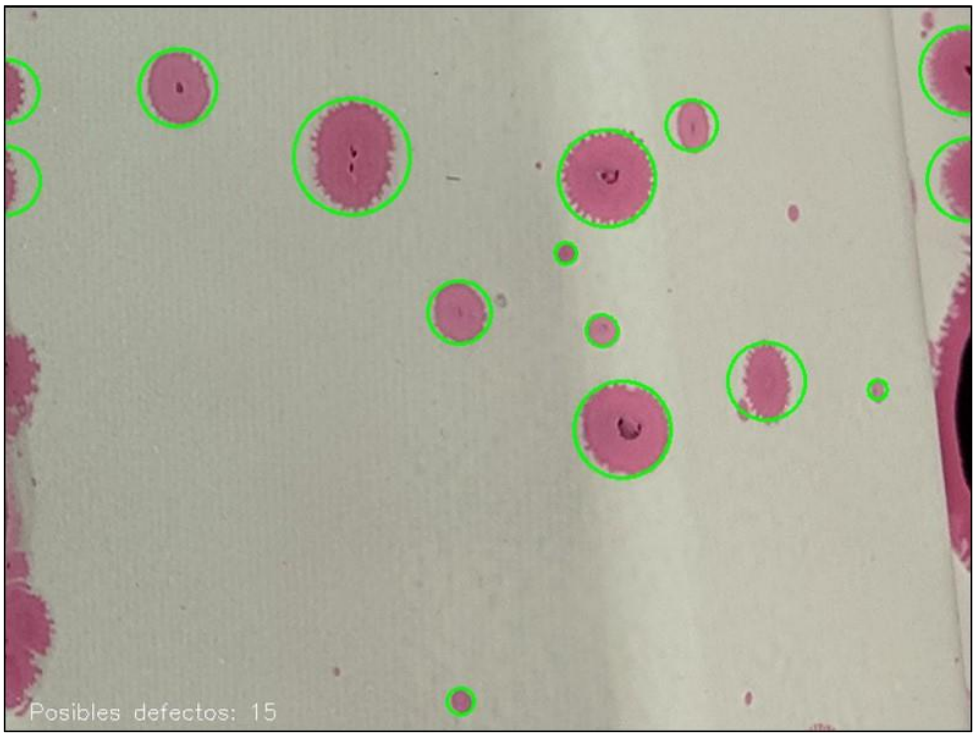


Figura 79. Resultado de la detección de contornos

4.2.2 Análisis de los resultados

Las imágenes de partida son imágenes de alta resolución de 4000x3000 píxeles, por lo que para su análisis cada imagen se dividió en una serie de fragmentos de 800x600 píxeles. Estas imágenes son las utilizadas para realizar el desarrollo expuesto en el apartado anterior. Un ejemplo de esta división se encuentra ilustrado en la Figura 80.

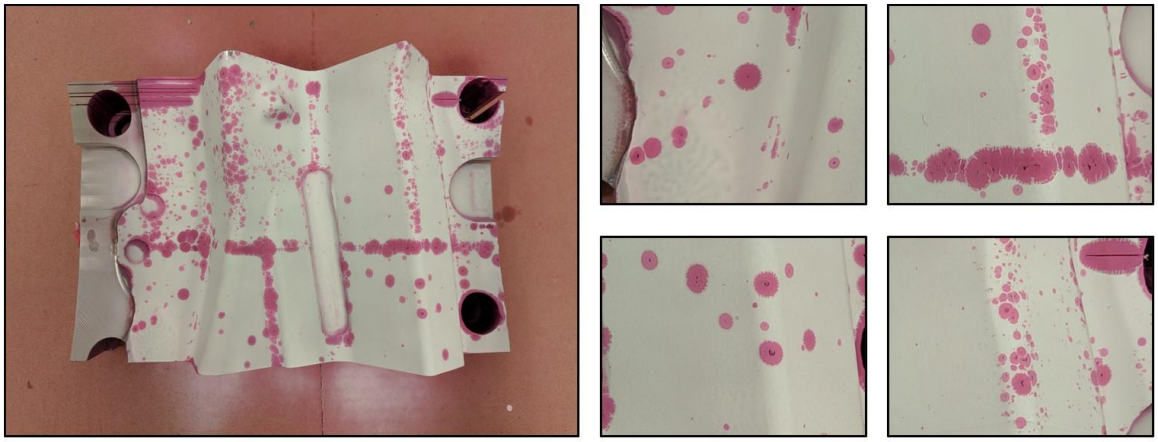


Figura 80. Imagen de partida y ejemplo de los fragmentos de esta

En la imagen de partida de la Figura 81, se puede apreciar una distribución desigual de defectos en función del cuadrante, estos resultados se corresponden con diferentes configuraciones del láser durante el recubrimiento. En algunas partes de dicha imagen, el análisis puede llegar a ser confusa para el algoritmo a consecuencia de las singularidades formadas por la continuidad entre una serie de defectos. La utilización del algoritmo en estas zonas no tendría sentido práctico debido a que los resultados serán claramente erróneos, como se podrá comprobar más adelante.

El correcto funcionamiento de este algoritmo depende en gran medida de la calidad de la imagen, siendo importante una correcta iluminación, ausencia de sombras, nitidez, etc. Con el fin de obtener unos resultados apropiados, para la evaluación de su desempeño se han seleccionado aquellas imágenes en las que la calidad es óptima.

En la Figura 81 se muestran cuatro ejemplos en los que el algoritmo ha conseguido detectar defectos con una exactitud bastante aceptable. Los resultados de esta primera prueba se recogen en la Tabla 7.

Tabla 7. Resultados de detección de la primera prueba

	Posibles	Detección	Correctas	*Precision*	*Recall*	*Miss rate*
Defectos	97	65	62	95%	64%	36%

De forma manual, en total se han detectado 97 posibles defectos que el algoritmo debería ser capaz de detectar. Atendiendo al valor de la **precisión** (*precision*), se deduce que los resultados son positivos puesto que solamente tres detecciones han sido incorrectas. Los resultados para la **exhaustividad** (*recall*) y la **tasa de pérdida** (*miss rate*) por su parte no son los esperados, puesto que el algoritmo no ha sido capaz de detectar el 36% de los defectos hallados manualmente.

Para este tipo de imágenes el algoritmo puede obtener buenos resultados en el caso de que los defectos tengan patrones distinguibles y puedan ser fácilmente separables del fondo.

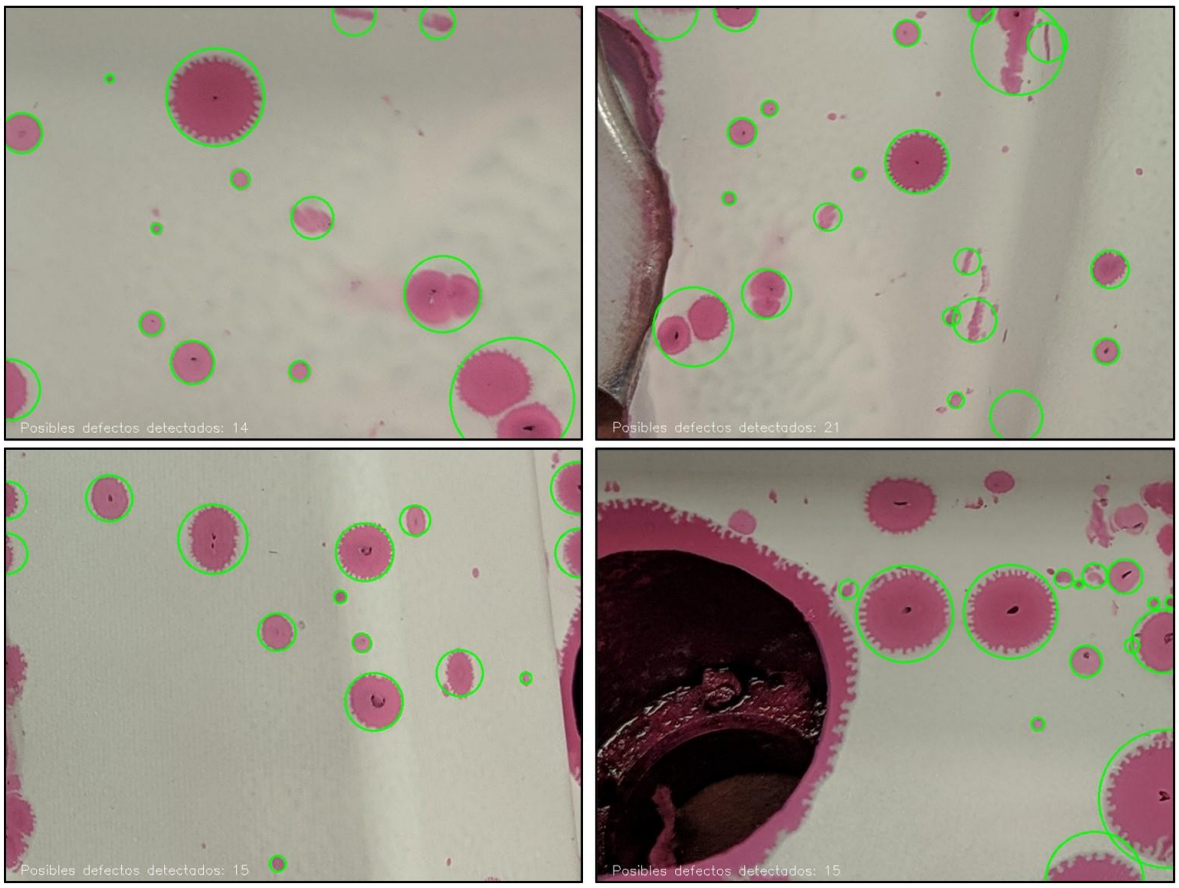


Figura 81. Primera prueba de detección

Como se ha mencionado, al ser imágenes con bastante buena iluminación y nitidez, el posible ruido generado no es muy alto, lo que provoca que los elementos puedan ser distinguidos fácilmente, resultando en una buena detección. En ocasiones pueden existir sombras y brillos que provocan errores durante la etapa de segmentación, generando malas detecciones.

En las imágenes disponibles existen agrupaciones de defectos en determinadas zonas que provocan el solapamiento de los líquidos penetrantes. En la Figura 82 se ilustran este tipo de imágenes. De forma manual, se han encontrado grandes dificultades para contar los defectos visibles en ambas imágenes, aproximadamente se han llegado a obtener unos 160 defectos. Esta segunda prueba, cuyos resultado se encuentran en la Tabla 8, muestra las grandes limitaciones de este algoritmo en determinadas situaciones.

Tabla 8. Resultados de detección de la segunda prueba

	Posibles	Detección	Correctas	*Precision*	*Recall*	*Miss rate*
Defectos	160	98	90	92%	56%	44%

Aunque el 92% de las detecciones realizadas corresponden en realidad con defectos, solamente se ha detectado el 56% de los defectos de las imágenes. Aunque a simple vista para el ojo humano algunos de estos defectos puedan a llegar a ser evidentes, las transformaciones realizadas por el algoritmo cambian totalmente la imagen.

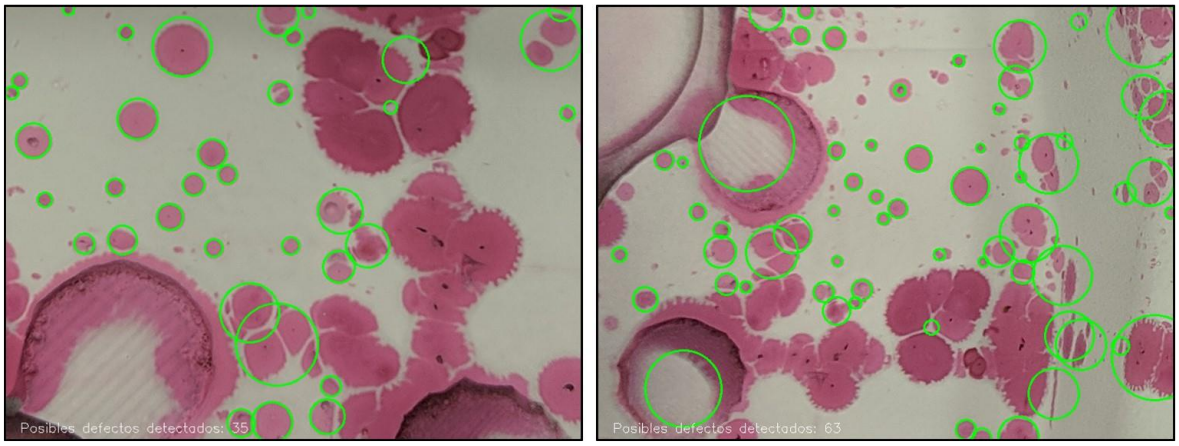


Figura 82. Segunda prueba de detección

En una tercera prueba (Figura 83) se observa todavía más las dificultades del algoritmo para obtener buenas detecciones en imágenes saturadas de defectos. En este caso se hace casi imposible contar de forma manual el número de defectos en las imágenes, por lo que no se presentan los resultados numéricos al respecto.

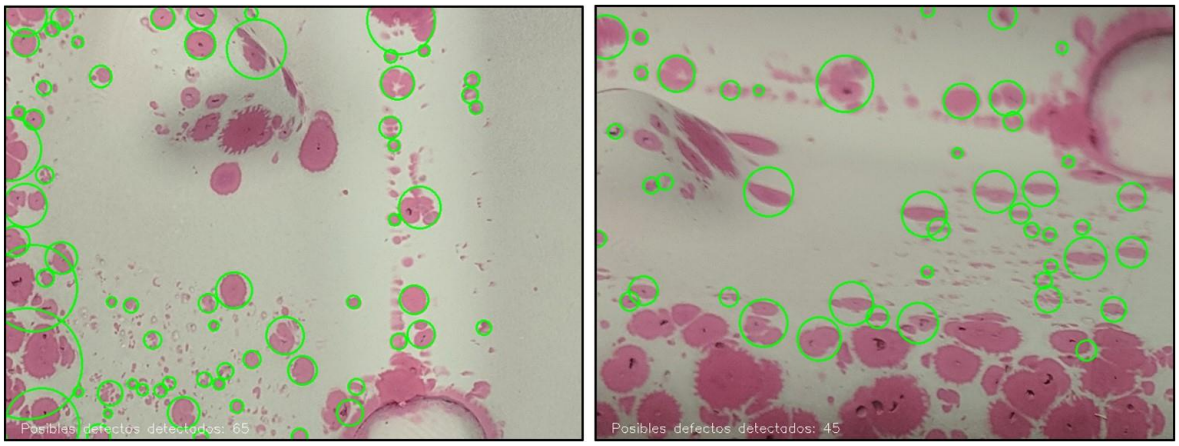


Figura 83. Tercera prueba de detección

A la vista de estos resultados, es evidente la importancia que tiene trabajar con imágenes de buena calidad puesto que los procesos realizados por el algoritmo se ven altamente influidos por las características de la imagen.

La utilización de este tipo de algoritmos puede dar buenos resultados a la hora de contar indicaciones bien diferenciadas en las piezas. En el momento que se forman agrupaciones de defectos, se hace casi imposible diferenciar correctamente unos de otros.

Cabe mencionar también la dificultad intrínseca debido a la forma de la pieza, cuya superficie es altamente irregular y requeriría que las imágenes fueran obtenidas desde distintos ángulos, según sea la superficie para analizar.

En piezas más simples y con una menor frecuencia de aparición de defectos, este algoritmo podría llegar a obtener buenos resultados.

4.3 CONCLUSIONES DEL CAPÍTULO

Esta posibilidad analizada implica tener que realizar de forma obligatoria el ensayo de líquidos penetrantes a cada una de las piezas a evaluar, con el riesgo de contaminación asociado. Además, realizar el ensayo de forma manual provoca que los tiempos de inspección sigan siendo largos.

Como se ha mostrado, en ocasiones, las indicaciones proporcionadas por este ensayo forman agrupaciones que son difíciles de identificar mediante algoritmos de visión artificial generando grandes problemas en los resultados de detección.

Aunque se ha demostrado la posibilidad de aplicar estos algoritmos en determinados casos, no suponen la mejor solución para automatizar un sistema de inspección de defectos, debido a que en todo momento debe de haber un operario responsable y con el conocimiento adecuado para tomar las decisiones oportunas.

El camino lógico sería realizar la detección directamente en la pieza a inspeccionar, utilizando únicamente métodos de inspección visual mejorados a través de la aplicación de algoritmos de visión artificial.

CAPÍTULO 5: DETECCIÓN DE DEFECTOS EN BRUTO

La aplicación de algoritmos de visión artificial para la detección de defectos sobre imágenes recogidas durante el ensayo de líquidos penetrantes puede ser muy interesante dado que se permite identificar y medir el área de las marcas del penetrante (parámetro que puede asociarse de forma proporcional con la gravedad de los defectos). Sin embargo, para los troqueles de estampación es vital que el tamaño de los defectos superficiales no supere valores umbrales que pudieran comprometer la calidad de las piezas conformadas. Se ha detectado que los algoritmos desarrollados hasta este punto no han sido capaces de determinar el tamaño del defecto de forma independiente a las marcas del penetrante. En este contexto, en el presente capítulo se evaluará la posibilidad de contabilizar los defectos directamente en la pieza, sin la necesidad de realizar el ensayo de líquidos penetrantes, que puede llegar a contaminar la pieza. Para ello, se propone una modificación del algoritmo presentado en el capítulo anterior.

5.1 DETECCIÓN EN PIEZAS SIN TRATAR

5.1.1 Desarrollo del algoritmo de detección

A diferencia de la adquisición de las imágenes del capítulo anterior la superficie mecanizada presenta un menor contraste de los defectos que ocasiona que no sean fácilmente distinguibles a simple vista, asimismo la naturaleza del material dificulta las operaciones posteriores a causa de los reflejos y las sombras.

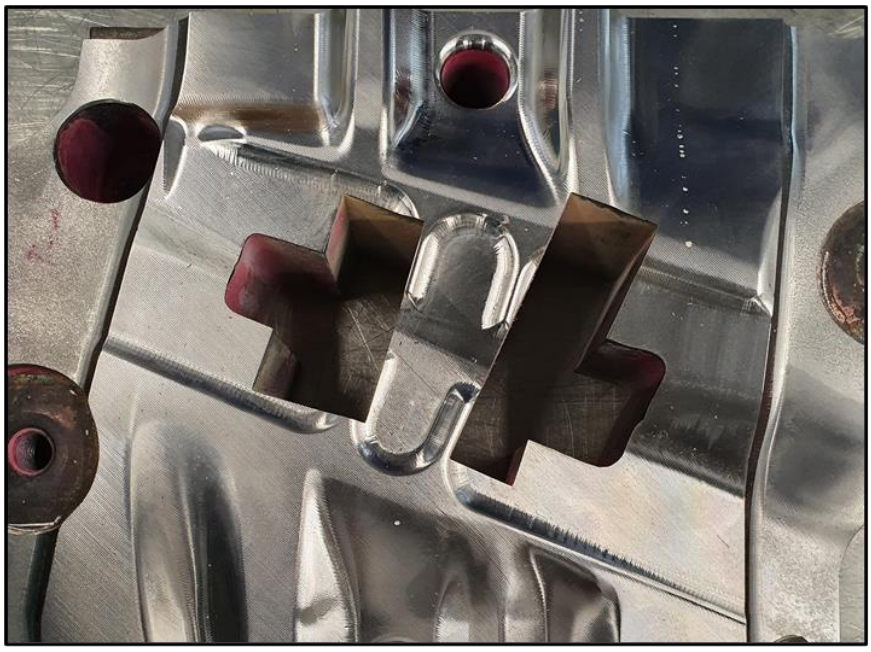


Figura 84. Ejemplo de pieza sin ensayar

En Figura 85 se recogen distintos fragmentos de las imágenes que se utilizarán en este apartado. Se han escogido fragmentos bastante distintos entre sí en cuanto a la iluminación y la textura de las imágenes, para comprobar el funcionamiento del algoritmo desarrollado. Cabe mencionar que es fácilmente apreciable la gran diferencia con las utilizadas anteriormente.

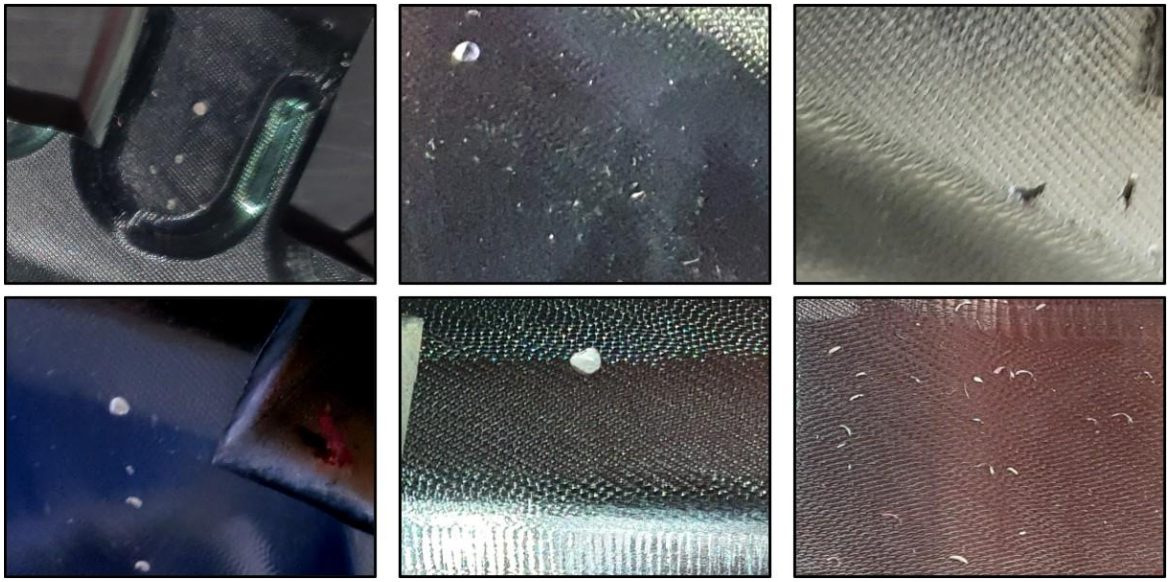


Figura 85. Fragmentos de imágenes en bruto

El algoritmo base se corresponde con el código desarrollado anteriormente, por lo que no se mostrarán los fragmentos correspondientes a cada parte como sí se hizo en el apartado anterior. El código completo se encuentra disponible en el anexo A.

La etapa de **pre-procesado** es equivalente a la utilizada anteriormente, convirtiendo en primer lugar la imagen a niveles de gris y aplicando posteriormente un filtro de Gauss con un *kernel* 7x7 para reducir el ruido de la imagen. Cabe recordar que la aplicación de un suavizado excesivo provoca la pérdida de información de la imagen. En la Figura 86 se puede observar el efecto de la aplicación de este filtro a las imágenes mostradas anteriormente.

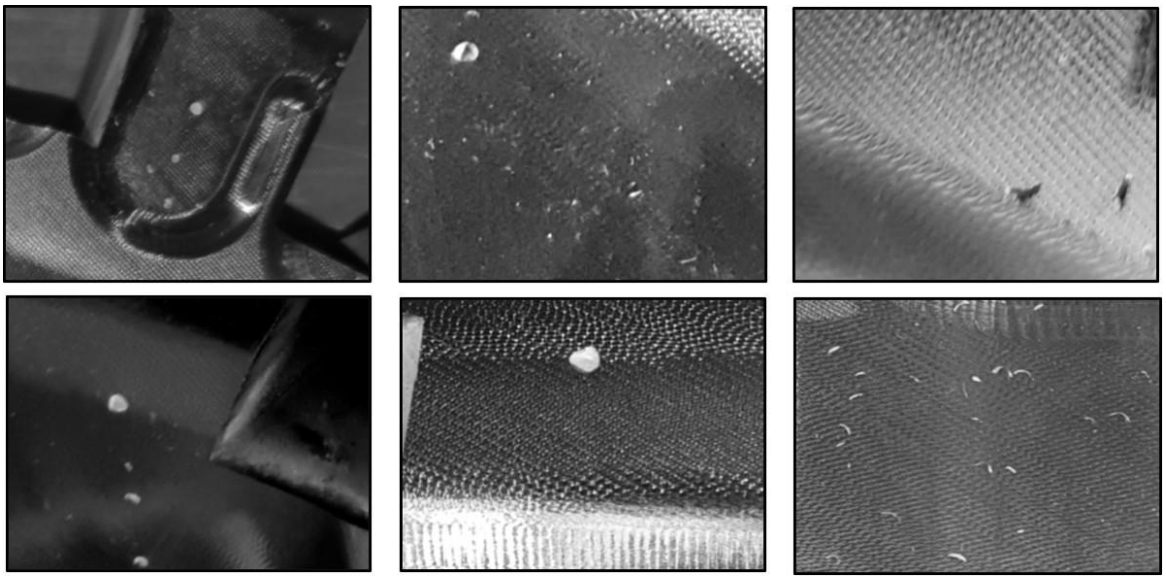


Figura 86. Aplicación del filtro de Gauss 7x7

Se han detectado diversas complicaciones durante la fase de **segmentación,** derivadas de la gran dificultad que supone tratar imágenes tan desiguales mediante la misma operación. En el algoritmo anterior se vio la dificultad que supone la selección del umbral adecuado para separar los elementos de una imagen, para este caso esta dificultad se ve ampliada. Aun así, debido a que la superficie de las piezas no es uniforme y limpia, la aplicación de un detector de bordes de Canny no sería posible y la imagen no sería segmentada de forma correcta. Por ello, se mantuvo la aplicación del método del triángulo para la obtención de la umbralización para comprobar los resultados obtenidos.

En la Figura 87 se recogen los resultados de aplicación del método del umbral óptimo para las imágenes tratadas. Se puede observar que, a diferencia de las imágenes del apartado anterior, el ruido persiste, dificultado la tarea de diferenciación de los elementos más importantes de la imagen.

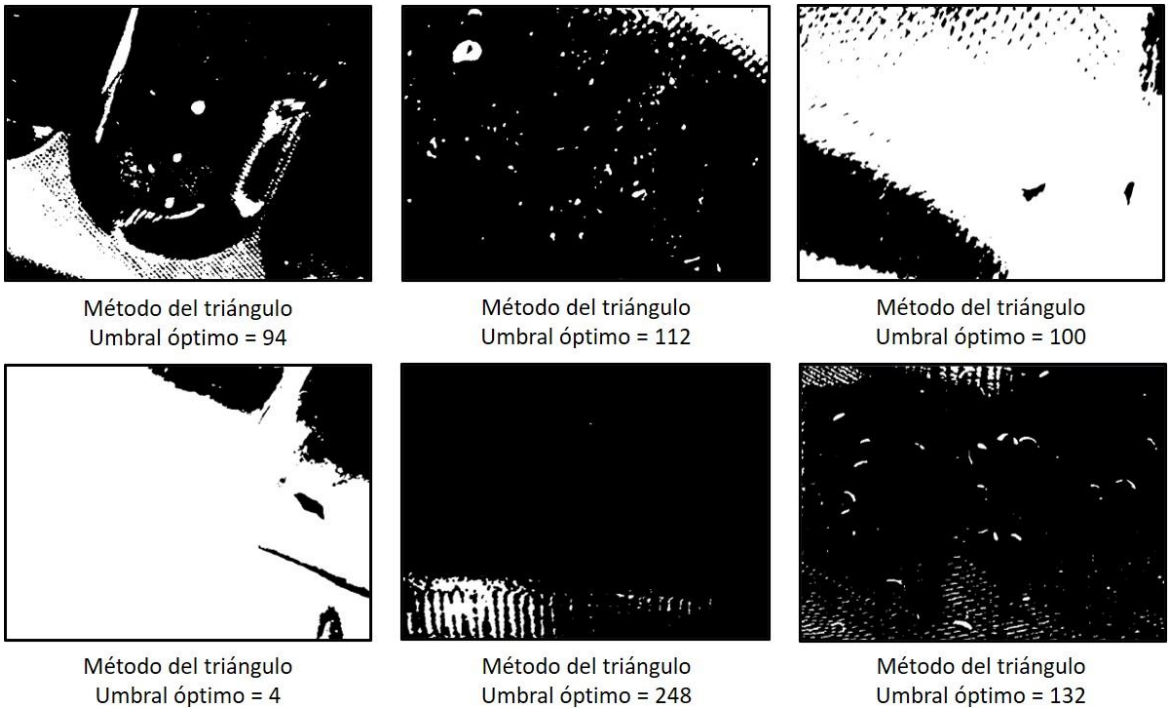


Figura 87. Efecto de la aplicación del umbral óptimo

Con el objetivo de reducir el ruido todavía presente, se realiza la operación de apertura, cuyo resultado se puede observar en la Figura 88. El tamaño de *kernel* es el mismo que el utilizado anteriormente, pero se puede observar que la eliminación del ruido en este caso no es tan buena. Aumentando el tamaño del *kernel* utilizado se podría conseguir eliminar gran parte del ruido. El mayor problema es que al aplicar la operación con un *kernel* tan grande se podría perder parte de la información de los defectos más pequeños.

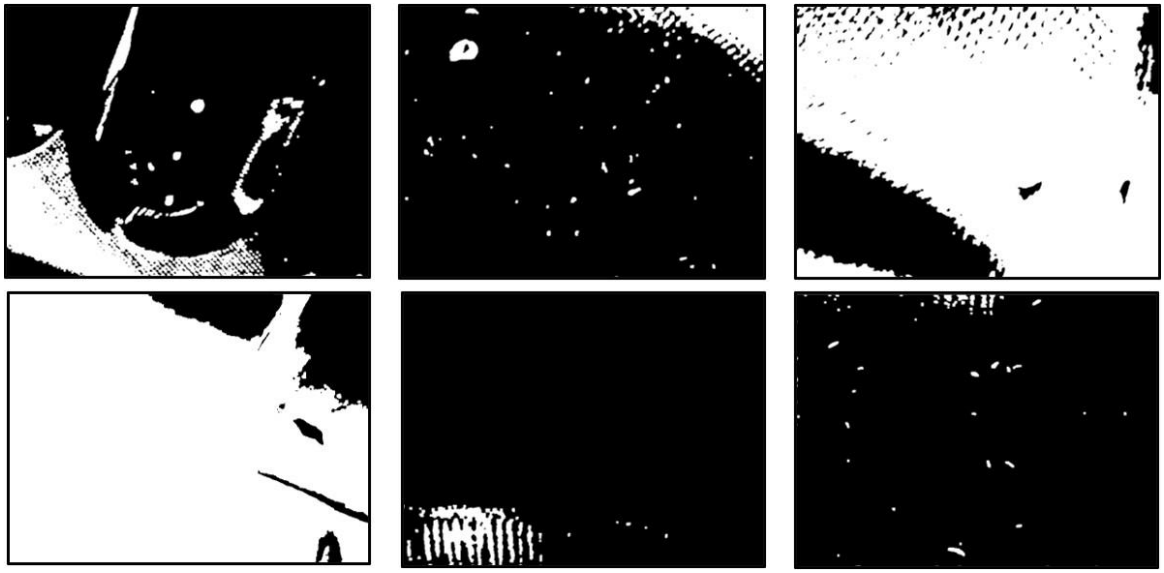


Figura 88. Efecto de la aplicación de la operación de apertura 5x5

Llegado a este punto se comenzó a observar las dificultades presentadas al trabajar con imágenes brillantes. El funcionamiento de la detección de contornos es la misma que la presentada en el algoritmo anterior.

Para realizar la **clasificación** es difícil seleccionar un criterio adecuado para descartar aquellas áreas más pequeñas o grandes debido a la gran diferencia entre las formas de los defectos. Variando los valores para el área máxima y la mínima se consiguen los resultados de la Figura 89, que como se puede observar son completamente inadecuados, produciendo una gran cantidad de detecciones falsas.

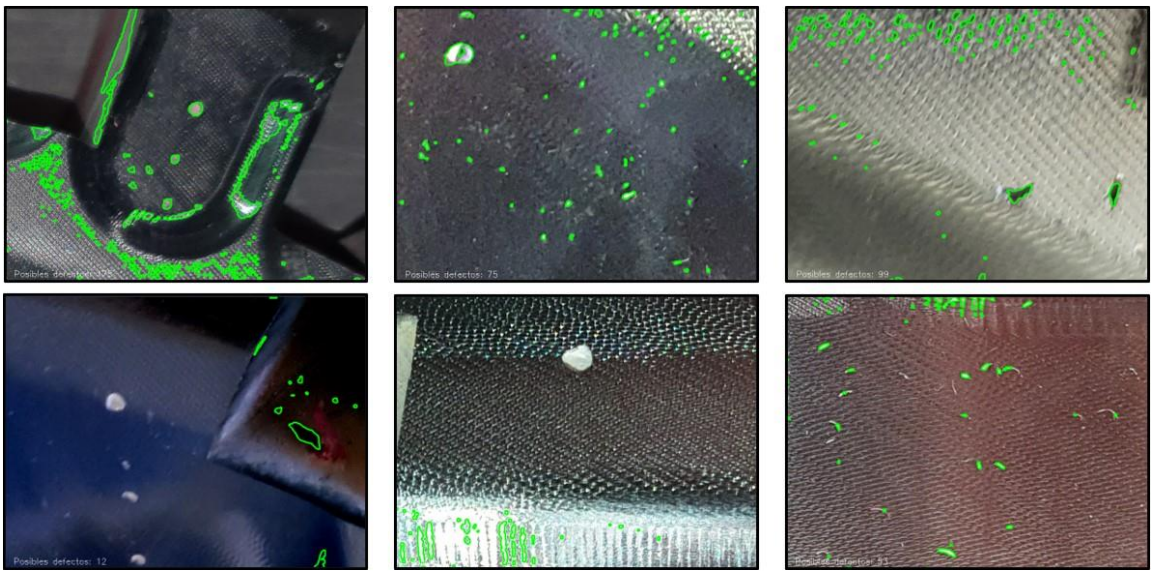


Figura 89. Resultados de la detección

5.1.2 Análisis de resultados

La evaluación del algoritmo a través de la medición de parámetros como la precisión o la exhaustividad hacen evidente el deficiente desempeño del algoritmo. En las imágenes utilizadas para las pruebas se han observado de forma manual alrededor de unos 67 posibles defectos que el algoritmo debería ser capaz de detectar. Al ejecutarlo, la suma total de las detecciones en las imágenes asciende a 447, siendo solamente 40 detecciones correspondientes a los defectos mencionados, como se puede ver en la Tabla 9. Por ello, la **precisión** del algoritmo es muy baja (menor del 10%), haciendo que sea inviable su utilización. La **exhaustividad,** aunque sea mayor del 50%, no cumple tampoco con las expectativas esperadas de un algoritmo de detección.

Tabla 9. Resultados de detección de las piezas sin tratar

	Posibles	Detección	Correctas	*Precision*	*Recall*	*Miss rate*
Defectos	67	447	40	9%	60%	40%

Para que este algoritmo pueda ser utilizado en un caso real, el primer obstáculo que es necesario solventar es el problema persistente de la presencia de brillos y sombras en las imágenes, que provoca grandes dificultades en la etapa de segmentación. Los umbrales óptimos seleccionados por el método del triángulo distan mucho de ser los adecuados para la correcta diferenciación de los elementos en algunas de las imágenes utilizadas, llegando incluso a provocar la desaparición de los elementos más significativos.

Es muy difícil seleccionar una sola secuencia de operaciones común para todas las imágenes, debido a las características tan distintas que poseen. Por ejemplo, si se seleccionara un valor único de umbral para la segmentación, se perdería gran parte de la información en ciertas imágenes.

En el caso de utilización de las imágenes tratadas previamente mediante el ensayo de líquidos penetrantes, se lograba obtener un contraste entre los elementos característicos y el fondo. Esto no es posible conseguirlo a través de las piezas en bruto, por lo que sería necesario cambiar las condiciones ambientales de captura de la imagen, a través de cambios en la iluminación.

Son varias las evidencias [27], [18], [29] o [31] que demuestran que la adquisición de las imágenes está comprometida con la calidad de estas (brillos, sombras, etc.) y por ellos es necesario adecuar el entorno de adquisición buscando unas condiciones específicas según sea el tipo de aplicación buscada.

En concreto, una de las alternativas sería la propuesta por Jiang et al. [31] que, como ya se mencionó anteriormente, solucionaba el problema mediante la captura de las imágenes a través de un escáner en vez de una cámara convencional.

5.2 DETECCIÓN EN PIEZAS TRATADAS

5.2.1 Realización del tratamiento

Como se ha observado en el apartado anterior, la mayor dificultad para el tratamiento de imágenes obtenidas de este tipo de piezas es hacer frente a los brillos y sombras de la superficie.

En la Figura 90 (izquierda) se puede observar un ejemplo de una de las piezas sin realizar el ensayo de líquidos penetrantes. Es fácilmente apreciable la existencia de estos brillos, por lo que su eliminación de forma previa a la aplicación del algoritmo puede llegar a ser una opción muy válida para reducir las limitaciones expuestas anteriormente.

Figura 90. Efecto del tratamiento de reducción de brillo

La compleja naturaleza de los troqueles de estampación, cuyas superficies forman geometrías muy irregulares formando complejos patrones en 3D provocan que la elección del proceso de captura no sea tan evidente. Esta geometría irregular favorece la aparición de brillos y sombras en la pieza, y no siempre es posible solventar esta problemática a través de unas condiciones de iluminación específicas. Por ello, debido a la imposibilidad de modificar el proceso de captura de las imágenes o de seleccionar unas condiciones de iluminación óptimas, con el objetivo de reducir la aparición de brillos y disponer de una superficie mate, se decidió impregnar a las piezas mediante el aerosol en polvo utilizado como revelador en el ensayo de líquidos penetrantes.

A través de la aplicación del aerosol, se puede conseguir una capa uniforme y continua de polvo blanco. Esta fina capa permite reducir los brillos de la imagen, lo que se espera que facilite la aplicación del algoritmo de detección. El resultado de aplicación de este aerosol se puede observar en la Figura 90 (derecha).

5.2.2 Desarrollo del algoritmo de detección

Igual que en ocasiones anteriores, las imágenes disponibles son muy grandes por lo que para evaluar la capacidad del algoritmo en el procesamiento se han seleccionado fragmentos concretos de las mismas. Algunos ejemplos de estos fragmentos se recogen en la Figura 91, donde se puede observar cómo gracias a la aplicación del aerosol, se ha incrementado el contraste entre los defectos y el fondo y se ha disminuido la influencia de las singularidades geométricas de la pieza.

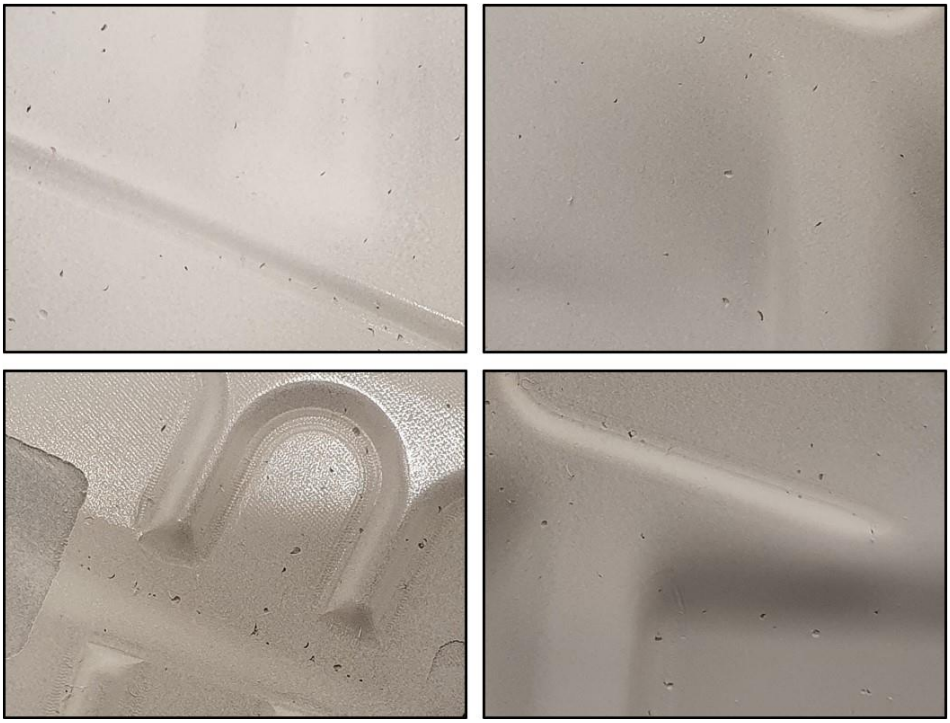


Figura 91. Fragmentos de imágenes sin brillo

Como en el caso anterior, se parte del algoritmo base desarrollado previamente con el objetivo de realizar aquellas modificaciones necesarias para adaptar su funcionamiento al nuevo problema planteado y únicamente se expondrá el proceso seguido para el tratamiento de las imágenes, dejando el código completo disponible en el anexo B.

Con el objetivo de que la modificación del algoritmo suponga un aumento de su robustez, el desarrollo se realiza partiendo de una pieza distinta a la utilizada en el capítulo anterior, cuya geometría y condiciones del proceso de *laser cladding* son distintas, mostrando nuevas condiciones de aparición de defectos.

La primera diferencia con el algoritmo utilizado previamente la encontramos en la etapa de **pre-procesado**. Sigue siendo necesario convertir la imagen a niveles de gris para posteriormente realizar un suavizado, y se sigue manteniendo el uso del filtro de Gauss para este fin. La diferencia reside en el tamaño del *kernel* utilizado, siendo en este caso uno de tamaño 5x5. Se ha escogido un tamaño menor debido a que en este caso, los defectos aparecidos en las imágenes tratadas son muy pequeños y como ya se ha expuesto anteriormente, un suavizado excesivo podría provocar pérdida de información y muchos de esos defectos desaparecerían en etapas posteriores. El efecto de aplicación de este filtrado se observa en la Figura 92.

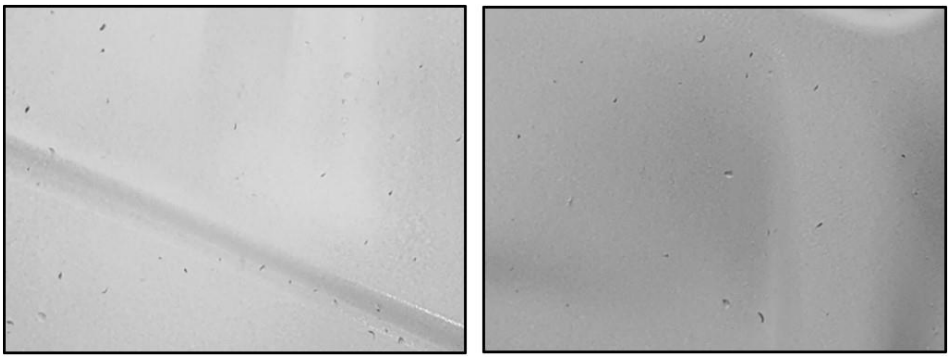


Figura 92. Aplicación del filtro de Gauss 5x5

Una primera prueba de la etapa de **segmentación** consiste en la aplicación (como en los casos anteriores) de una umbralización fija. Como siempre, la selección del valor del umbral es una tarea muy complicada cuando se quieren tratar distintas imágenes, e incluso utilizando un método de selección como el método del triángulo (Figura 93) no se obtienen los resultados adecuados. La alternativa escogida es la utilización del detector de bordes de Canny. Para su aplicación se fijan el valor mínimo de umbral en 75 y el valor máximo en 100 que, al ser unos valores medios, generalmente dan buenos resultados en la diferenciación de elementos. Además, para unir los bordes extraídos y generar elementos más diferenciables, se realiza una operación de cierre con un *kernel* 7x7.

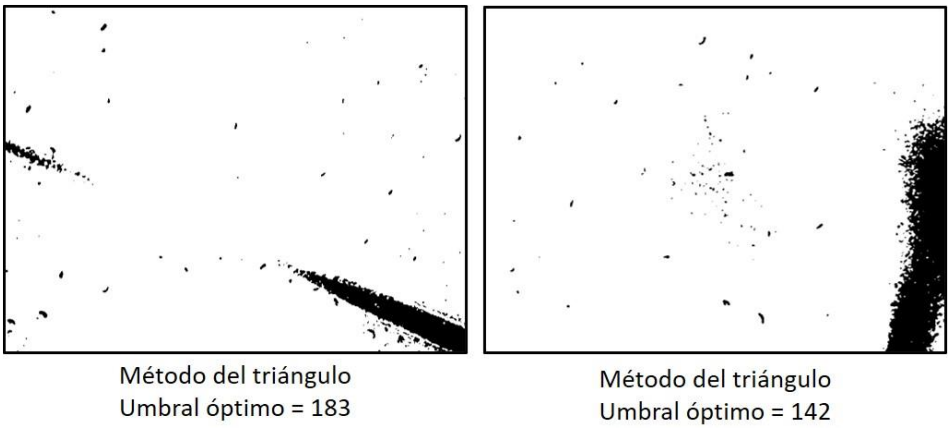


Figura 93. Aplicación del umbral óptimo

En la Figura 94 se observa el resultado de aplicación del algoritmo de Canny antes mencionado y su posterior corrección mediante la operación de cierre.

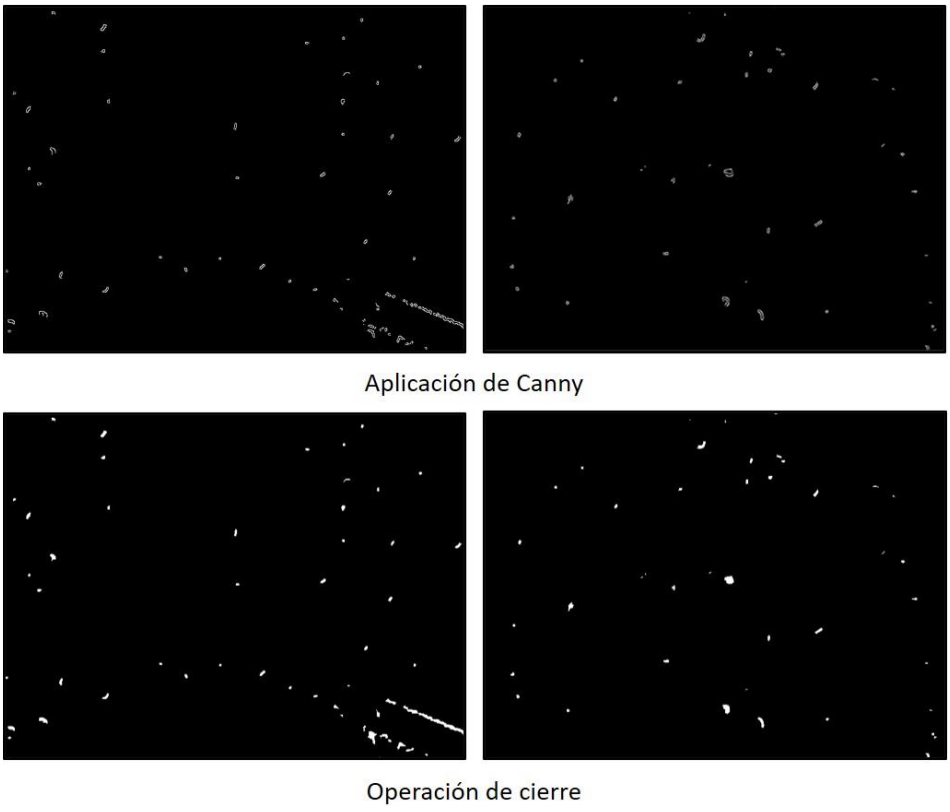


Figura 94. Aplicación del detector de Canny y la operación de cierre

Para la **clasificación** se sigue un método similar al del algoritmo original, estableciendo unos valores mínimo y máximo correspondientemente para el área calculada de cada una de las detecciones y así descartar aquellos elementos que por su menor o mayor tamaño no son considerados en principio defectos. Debido a las diferentes formas de los defectos se ha decidido emplear la representación del contorno directamente, en vez de realizar la circunferencia representativa, puesto que no todos los elementos se ajustan bien a esta representación.

Con la modificación del código se ha aprovechado para introducir algunas mejoras que pueden ser perfectamente implementadas en el código original. En primer lugar, se ha numerado cada uno de los defectos aparecidos en la imagen.

Pese a no representar las circunferencias circunscritas de cada elemento, los datos de la localización del centro de la circunferencia y de su radio sí que son útiles para caracterizar los defectos. Estos datos, junto con las dimensiones del área, permiten obtener un listado con las características representativas de cada defecto.

Por último, al igual que en los otros casos, se muestra el número de posibles defectos encontrados en la imagen analizada. El resultado de aplicación del algoritmo se encuentra representado en la Figura 95.

Al lado de cada detección se muestra el área del defecto. En ocasiones, este valor se muestra fuera de la ventana correspondiente a la imagen o se solapa con el resto de las detecciones debido a que la posición del texto se fija respecto al centro del defecto. Por ello, además de mostrarse en cada una de las imágenes, toda esta información también aparece en la ventana de comandos una vez ejecutado el código.

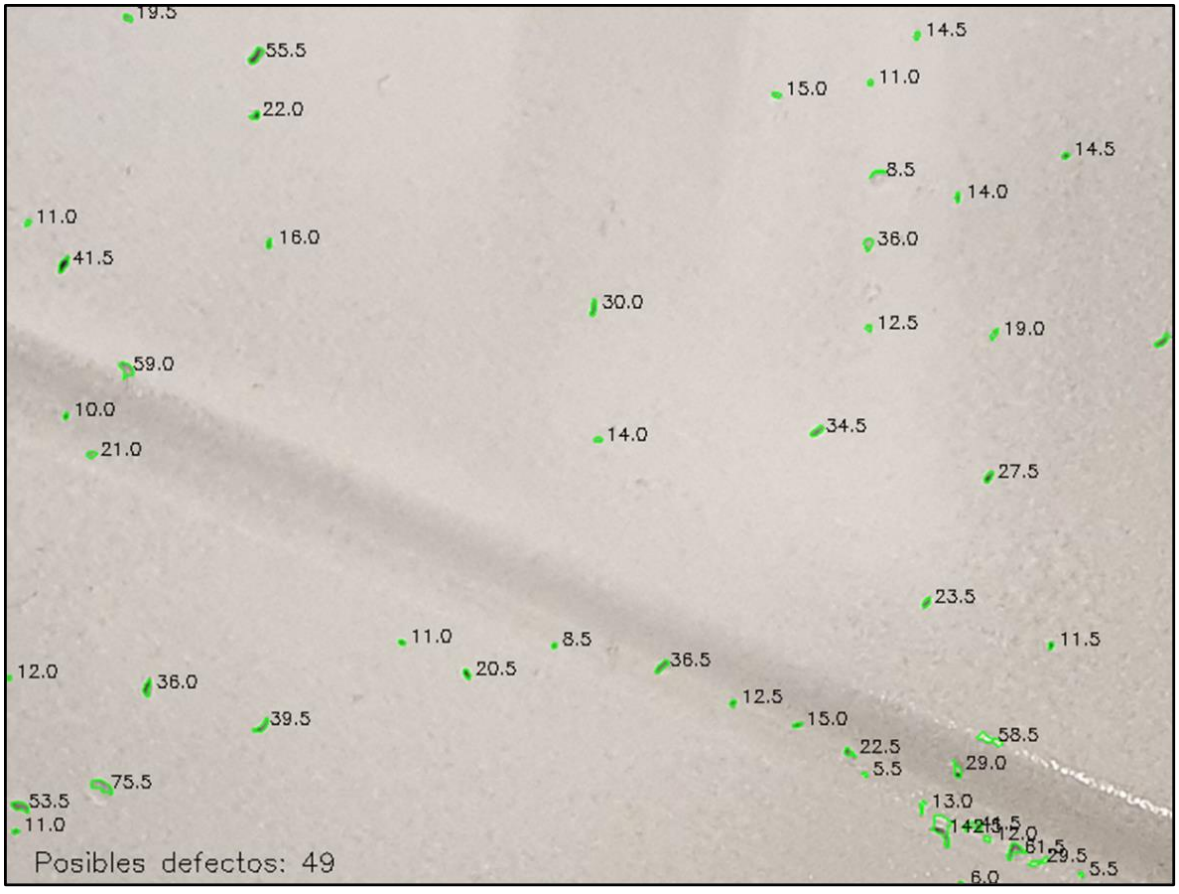


Figura 95. Ejemplo de resultado de la detección

5.2.3 Análisis de resultados

Como en el caso anterior, al prescindir del ensayo por líquidos penetrantes la dificultad para contar el número de defectos a simple vista es mayor. Aun así, para obtener un valor numérico que ayude a justificar el proceso seguido y analizar el rendimiento, se ha realizado una aproximación de los posibles defectos detectados manualmente en cada una de las imágenes. En la Figura 96 se muestran los resultados más representativos de la ejecución del algoritmo, que corresponden con tres sectores distintos de la pieza analizada.

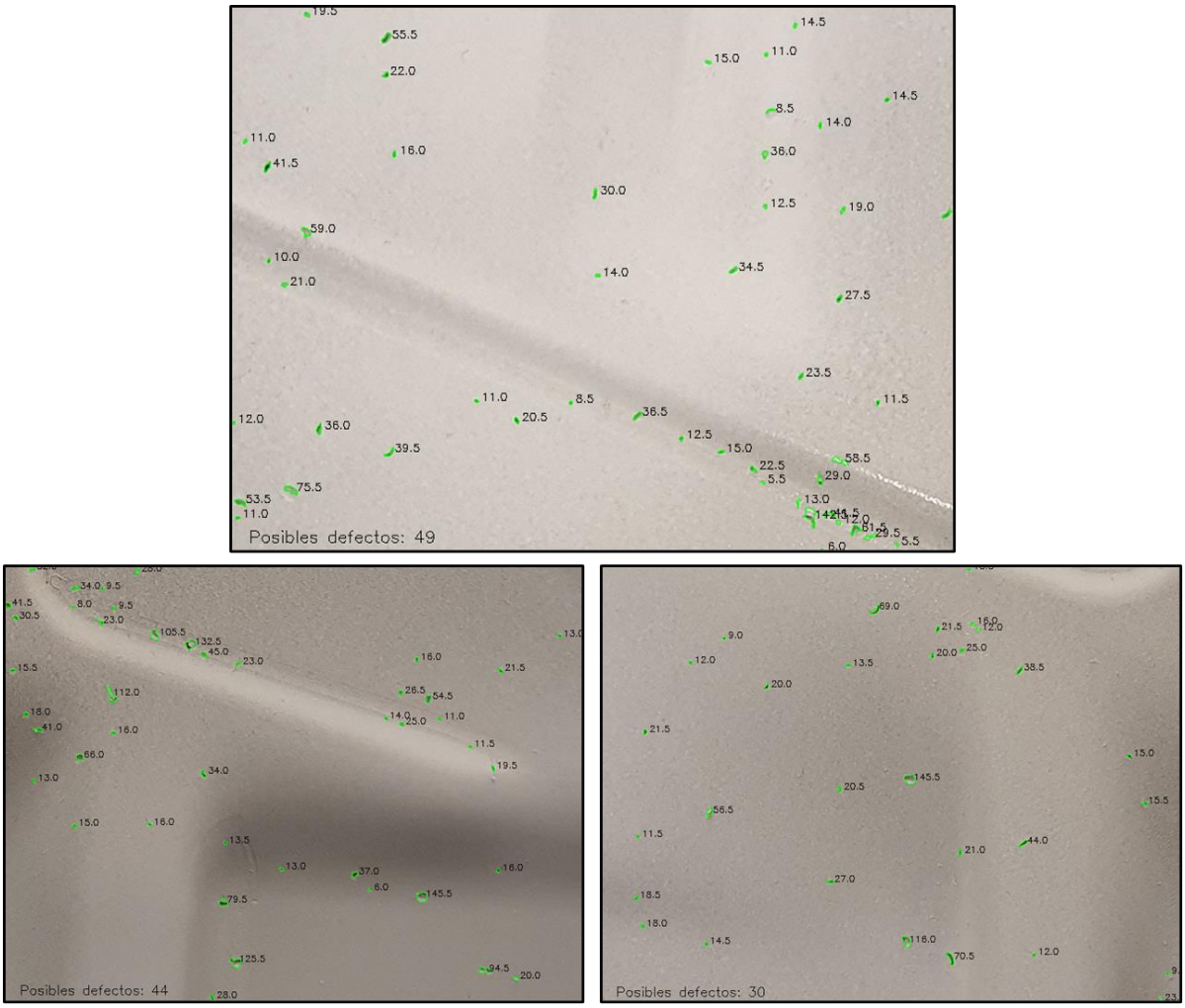


Figura 96. Resultados de la ejecución del algortimo

En el conjunto de las tres imágenes analizadas se han contabilizado aproximadamente 118 defectos. El algoritmo por su parte muestra un total de 123 detecciones entre las cuales se encuentran los 118 defectos registrados. Estos buenos resultados se traducen en altos valores tanto para la **precisión** como para la **exhaustividad**, demostrando que el tratamiento aplicado mejora en gran medida el rendimiento del algoritmo.

Tabla 10. Resultados de detección en la pieza

	Posibles	Detección	Correctas	*Precision*	*Recall*	*Miss rate*
Defectos	118	123	118	96%	100%	0%

Al haber reducido en gran medida el brillo de las imágenes, se ha logrado realizar la detección y contabilización de los defectos prescindiendo del ensayo por líquidos

penetrantes. Además, con las modificaciones realizadas en el algoritmo se ha conseguido añadir la posibilidad de medir directamente el área de los defectos.

Gracias al tratamiento realizado y a la aplicación del algoritmo de visión artificial propuesto, se ha definido una metodología de inspección que abre la posibilidad de desarrollar un sistema automático de detección y evaluación de los defectos. A través de este sistema, la inspección mediante el ensayo por líquidos penetrantes no sería necesaria, lo que supondría un ahorro en los tiempos de inspección.

Cabe mencionar que el área de los defectos en la imagen podría relacionarse de forma directa con su tamaño real, estableciendo una relación entre el número de píxeles de la imagen y la distancia a la que se ha realizado la captura.

A través de la medición de su área, es posible realizar un histograma como los mostrados en la Figura 97 y Figura 98. Mediante este histograma es posible analizar la frecuencia de aparición de los defectos según unos intervalos de tamaño. Por ejemplo, es fácilmente observable que los defectos cuyo área está comprendida entre 10 y 20 píxeles cuadrados son más habituales.

La obtención de este histograma permite evaluar de forma mucho más rápida y sencilla el estado de las piezas. Con el objetivo de distinguir entre piezas válidas y no válidas según el tamaño de los defectos encontrados, es posible marcar unos límites de tamaño y frecuencia que permitan facilitar la toma de decisiones a través de un simple vistazo del histograma.

Además, podría llegar a ser interesante dotar al algoritmo de la capacidad de obtener este histograma de forma automática para cada una de las piezas. En este caso, el histograma se ha obtenido de forma manual.

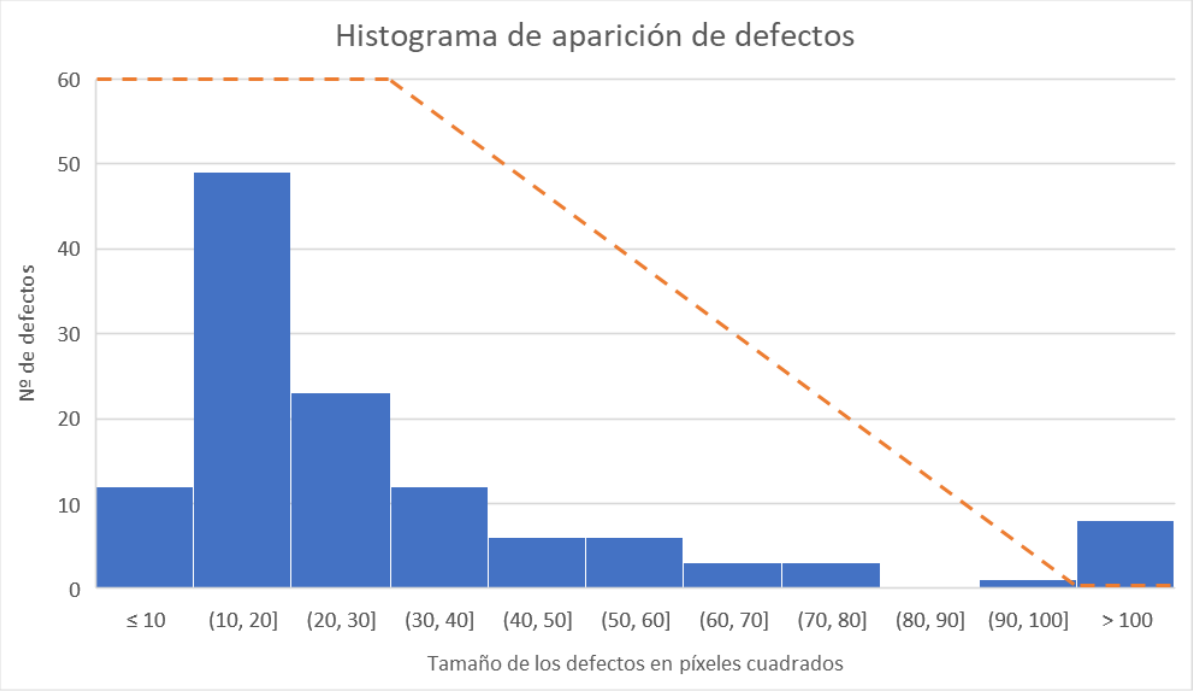


Figura 97. Histograma de aparición de defectos

En la Figura 98 se recogen los sectores analizados de la pieza junto con sus histogramas de aparición de defectos. Combinando los resultados obtenidos de estos sectores se llega al histograma mostrado en la figura anterior.

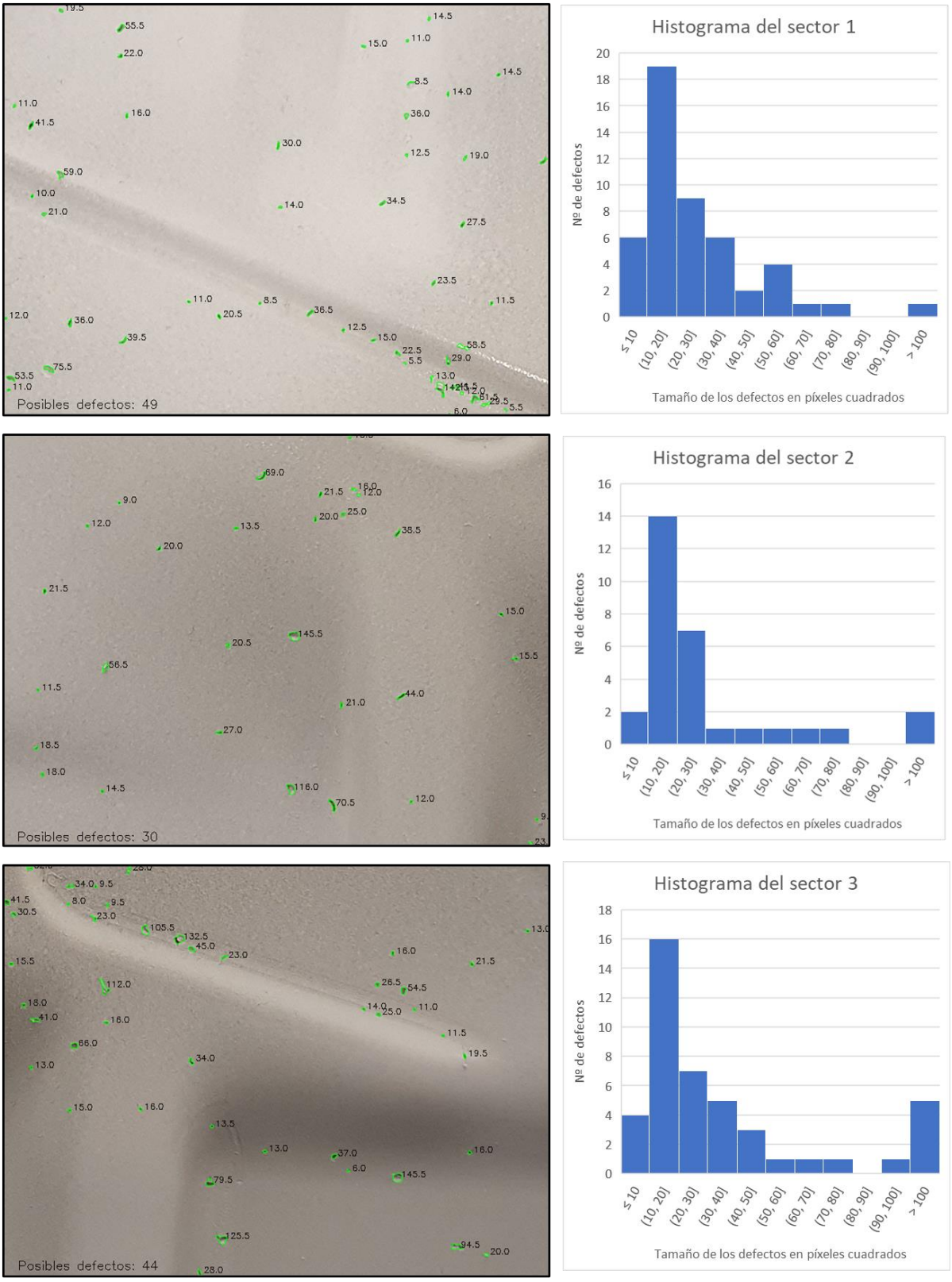


Figura 98. Sectores de la pieza analizada junto con sus histogramas

La inspección de una pieza a nivel industrial a través de este algoritmo supone en primer lugar la división de la pieza en zonas de interés que deberán ser evaluadas de forma individual. En el ejemplo mostrado, estas zonas se corresponden con las imágenes mostradas en la Figura 96.

Una vez seleccionadas estas zonas, se aplica el algoritmo obteniendo los resultados mostrados anteriormente. De cada imagen, correspondiente a una zona concreta de la pieza, se obtiene el número de defectos, su localización y su tamaño. Toda esta información se recoge y se almacena, formando la información de la defectología asociada a la pieza.

A través de la medición del área de cada uno de los defectos, se construiría un histograma similar a los mostrados en las figura que permite observar la frecuencia de aparición de los defectos de una forma sencilla. Como se ha mencionado, es rápidamente observable qué tamaño de defectos son los que tienen una frecuencia mayor.

La elección de los límites de tolerancia debe ser fijada por el inspector de calidad encargado de la tarea. En este caso (Figura 97), los defectos cuyo tamaño está comprendido entre 0 y 30 son perfectamente admisibles puesto que no tienen influencia negativa. Los defectos de tamaño comprendido entre 30 y 100 pueden tener cierta influencia y se establece un límite lineal. Los defectos cuyo tamaño supere esos valores no son admisibles. Toda esta información debe ser proporcionada por el departamento de calidad responsable de la inspección y el límite mostrado en este desarrollo solamente es un ejemplo.

En resumen, la delimitación fijada por esta línea permite realizar la toma de decisiones siguiendo un único criterio: si las barras del histograma superar el límite de la línea, la pieza debe ser descartada puesto que la frecuencia de aparición de los defectos es superior a la tolerancia marcada.

Analizando el caso de la pieza en cuestión, tomando como referencia la línea utilizada se observa que la pieza debe ser descartada puesto que los defectos más grandes superan la tolerancia fijada por el inspector de calidad.

Cabe mencionar, que para la inspección a nivel industrial conviene tener una relación entre el tamaño de los defectos en píxeles y su tamaño real. En el caso de la pieza analizada, los defectos más comunes son de alrededor de 1,5 mm, que es un tamaño que se considera admisible.

5.3 CONCLUSIONES DEL CAPÍTULO

El desarrollo del algoritmo de detección y su posterior modificación ha permitido obtener una herramienta válida para realizar la detección de defectos en piezas sin ensayo previo. A través de esta herramienta es posible analizar el estado de las piezas de forma rápida, permitiendo evaluar la calidad de estas y descartar aquellas defectuosas.

Esta posibilidad se muestra como la mejor alternativa para desarrollar un sistema automático y completo de detección de defectos. La aplicación de este algoritmo para realizar la evaluación supone una serie de ventajas:

- Permite contabilizar los defectos de forma adecuada sin cometer errores reseñables.
- La medición de las dimensiones de los defectos permite realizar la construcción de un histograma que facilite la evaluación de la pieza.
- El tratamiento propuesto es fácilmente automatizable.

Las limitaciones de aplicación de los algoritmos de visión artificial siguen estando muy relacionadas con las condiciones de adquisición de la imagen por lo que, a la hora de diseñar un sistema real de inspección, se debe hacer un estudio exhaustivo de las condiciones de iluminación.

CAPÍTULO 6: APLICACIÓN DE DEEP LEARNING EN LA DETECCIÓN

Llegados a este punto, se ha mostrado la posibilidad de utilizar los algoritmos de visión artificial para realizar la detección de los defectos, su contabilización e incluso la obtención de sus dimensiones. Estos algoritmos únicamente están basados en la búsqueda de contornos a través de la diferencia de niveles de intensidad de la imagen. Esto provoca que en ocasiones se pierda información debido a la posible similitud entre píxeles adyacentes o que, a la hora de realizar las distintas operaciones, aparezcan posibles falsos contornos. En este capítulo, se evalúa la posibilidad de aplicar algoritmos más avanzados a través del uso de redes neuronales.

6.1 ALGORITMO DE CLASIFICACIÓN DE IMÁGENES

Como se ha comentado, los algoritmos anteriores suponen una aplicación muy general de la visión artificial. A través del *Deep Learning*, se podría llegar a entrenar una compleja red neuronal que fuese capaz de localizar y clasificar los distintos defectos. Este procedimiento más avanzado se puede conseguir a través de un clasificador de imágenes o de un detector de objetos.

En primer lugar, es importante distinguir entre estos dos conceptos:

- Clasificador de imágenes (*image classifier*): un clasificador se encarga de categorizar una imagen según las clases consideradas por la red neuronal.
- Detector de objetos (*object detector*): localiza elementos en una imagen y categoriza cada elemento según las clases consideradas.

Generalmente, la detección de objetos es una tarea mucho más compleja que la clasificación, puesto que interviene un mayor número de procesos, por ello para evaluar la posibilidad de entrenar una red neuronal para resolver la problemática de la detección de defectos, se va a desarrollar un clasificador simple que permita clasificar imágenes que contengan indicaciones proporcionadas por el ensayo de líquidos penetrantes.

6.1.1 Preparación de la base de datos

Ya sea el desarrollo de un clasificador o de un detector, el primer paso es la preparación de una base de datos lo suficientemente amplia para que el entrenamiento de la red neuronal sea el correcto.

Para este caso en concreto, solamente existe un tipo de elemento a detectar, correspondiente a las posibles indicaciones encontradas en la imagen. Por ello es necesario generar dos conjuntos distintos, imágenes que contengan estas indicaciones e imágenes que no.

De forma manual se han obtenido recortes de las imágenes de partida para formar los conjuntos. En total se dispone de 300 imágenes positivas (correspondientes a defectos) y 300 imágenes negativas (correspondientes a todo aquello no considerado defecto).

En la Figura 99 se muestran unos subconjuntos de las imágenes positivas y de las imágenes negativas recopiladas en la base de datos.

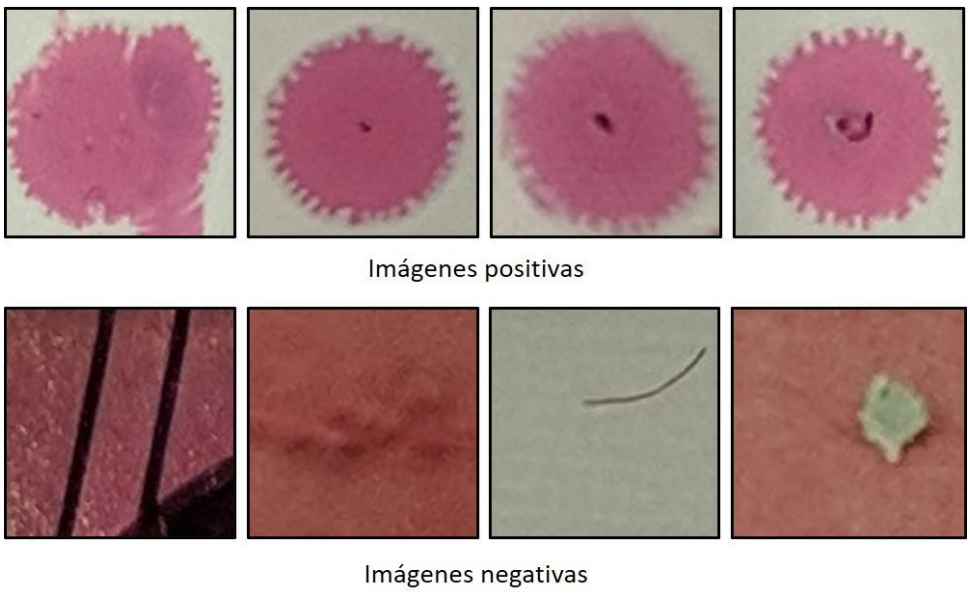


Figura 99. Ejemplos de imágenes positivas y negativas

6.1.2 Creación de la red neuronal

Existen múltiples redes neuronales convolucionales que pueden utilizarse para la clasificación de objetos. El entrenamiento de estas redes puede llegar a tener un coste computacional muy alto y requiere tiempo y equipos muy potentes.

Ya se ha mencionado previamente el completo material que proporciona el doctor Adrian Rosebrock de forma gratuita en forma de guías y tutoriales. Adicionalmente en [24] se encuentra recogida una gran cantidad de información apropiada para comenzar con el desarrollo de redes neuronales, proponiendo varias soluciones para resolver distintos casos de clasificación de imágenes a través de OpenCV.

Una arquitectura sencilla y perfecta para empezar a trabajar es LeNet [40], pensada en sus orígenes para la clasificación de dígitos escritos a mano. Anteriormente ya se ha visto su posible aplicación en detección de defectos en tornillos por Song et al. [19]. La arquitectura de la red (Figura 100) consiste en dos conjuntos de capas de convolución, activación y agrupación, seguidas de una capa densa y una activación, y por último otra capa densa y un clasificador *softmax*.

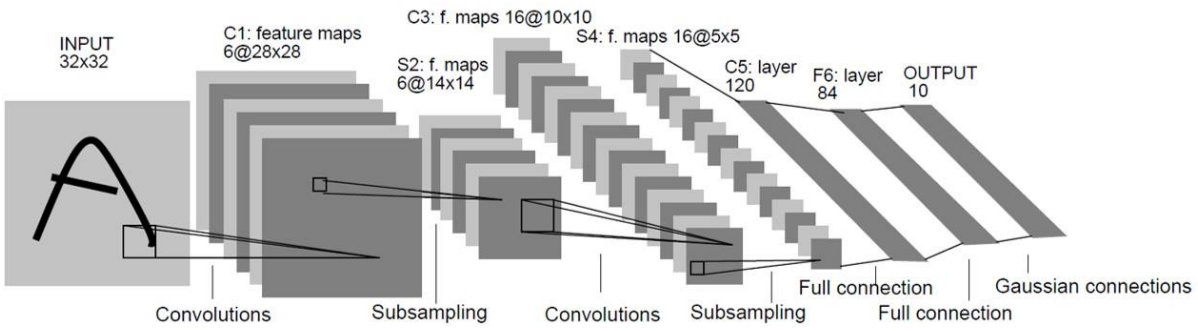


Figura 100. Arquitectura de la red LeNet (Fuente: [40])

Como se ha mencionado anteriormente, la red se puede dividir en cuatro conjunto de capas. En el primer conjunto se tiene una capa convolucional que se encargará de

aprender 20 filtros de tamaño 5x5. Posteriormente se aplica la función de activación ReLU y luego se realiza un *max-pooling* de 2x2 en *x* e *y*.

El segundo conjunto de capas se compone igual que el anterior. Esta vez la red aprenderá 50 filtros convolucionales. Es muy común aumentar el número de filtros según se añaden capas cada vez más profundas.

En el tercer conjunto de capas transformamos la salida anterior en un vector plano lo que permite aplicar ahora una capa densa (*dense* o *fully-connected*) que contiene 500 nodos, y posteriormente aplicar de nuevo la función de activación ReLU.

La última capa densa contiene tantos nodos como salidas se desean obtener, que en este caso serán dos, correspondientes a las dos clases (defecto y no defecto). Posteriormente se aplica el clasificador *softmax* que se encarga de dar una probabilidad de pertenencia para cada clase.

```
# PRIMER CONJUNTO DE CAPAS CONV => RELU => POOL
model.add(Conv2D(20, (5, 5), padding="same",
    input_shape=inputShape))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# SEGUNDO CONJUNTO DE CAPAS CONV => RELU => POOL
model.add(Conv2D(50, (5, 5), padding="same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# CONJUNTO DE CAPAS FC => RELU
model.add(Flatten())
model.add(Dense(500))
model.add(Activation("relu"))

# CLASIFICADOR SOFTMAX
model.add(Dense(classes))
model.add(Activation("softmax"))
```

Figura 101. Fragmento de código correspondiente a las capas neuronales

6.1.3 Entrenamiento de la red neuronal

Una vez creada la red neuronal el siguiente paso es entrenarla con la base de datos creada para tal efecto. Para realizar el entrenamiento es preciso disponer del código que se encargará de esa tarea. El código propuesto permite entrenar la red a partir de una base de datos proporcionada por el usuario, obteniendo una gráfica que relaciona la pérdida y la precisión durante el entrenamiento.

Puesto que el objetivo del documento no es explicar cada línea de código de forma exhaustiva, se hará hincapié en aquellas partes más relevantes para su comprensión general (Figura 102 y Figura 104).

Para empezar el entrenamiento de cualquier red neuronal es necesario inicializar los valores para los siguientes parámetros:

- *Epochs*: hace referencia al número de veces que la red trabaja toda el conjunto de datos. Valores muy altos provocan un entrenamiento lento y en ocasiones puede resultar en un sobreentrenamiento de la red. Un valor de partida puede ser 25.
- *Learning rate*: hace referencia al tamaño del escalón en cada iteración para encontrar el mínimo de la función de pérdida. Valores típicos son 0.1, 0.01 o 0.001.

- *Batch size*: hace referencia al tamaño de los lotes que utiliza la red para el entrenamiento. Un valor de referencia puede ser 32.

El primer proceso que realiza el algoritmo es la lectura de cada una de las imágenes mezcladas de forma aleatoria y actualizando la lista de etiquetas interna del algoritmo para discretizar entre las distintas clases.

```
# INICIALIZAMOS LOS DATOS Y LAS ETIQUETAS
print("[INFO] CARGANDO IMAGENES...")
data = []
labels = []

# LEEMOS LAS RUTAS DE LAS IMAGENES Y LAS MEZCLAMOS ALEATORIAMENTE
imagePaths = sorted(list(paths.list_images(args["dataset"])))
random.seed(42)
random.shuffle(imagePaths)

# BUCLE PARA LAS IMAGENES DE ENTRADA
for imagePath in imagePaths:
    # LEEMOS LA IMAGEN, PREPROCESAMOS Y LA ALMACENAMOS EN LOS DATOS
    image = cv2.imread(imagePath)
    image = cv2.resize(image, (64, 64))
    image = img_to_array(image)
    data.append(image)

    # EXTRAEMOS LA CLASE Y ACTUALIZAMOS LA LISTA DE ETIQUETAS
    label = imagePath.split(os.path.sep)[-2]
    label = 1 if label == "Defect" else 0
    labels.append(label)
```

Figura 102. Fragmento de código correspondiente al entrenamiento de la red (1)

Posteriormente se generan los conjuntos de entrenamiento y de prueba. Para ello se dividen los datos de partida en dos grupos: un 75% de los datos corresponden al conjunto de entrenamiento y el 25% restante al conjunto de prueba. Además, para aumentar el tamaño de los conjuntos se generan nuevas imágenes a partir de los datos de partida, modificando parámetros como el ángulo de rotación, el zoom de la imagen, la altura, el ancho, etc. En la Figura 103 se ilustra esta operación.

Figura 103. Ejemplo de incremento de los datos

El siguiente paso es compilar el modelo utilizando la red creada anteriormente. El algoritmo de optimización utilizado es el *Adam Optimizer* (*Adaptive Moment Estimation*), muy utilizado en *Deep Learning*. Al tener un problema de clasificación de dos clases, la función de pérdida utilizada es la entropía cruzada binaria (*binary cross-entropy*).

Una vez inicializado el modelo, las siguiente líneas de código corresponden al entrenamiento de la red utilizando los conjuntos definidos anteriormente, sumando los nuevos datos generados.

Por último, obtenemos el archivo que recoge el modelo de red entrenado y preparado para utilizar. Tras el entrenamiento también se genera una gráfica que muestra la evolución de la precisión y de la pérdida durante el proceso de aprendizaje.

```
# INICIALIZAMOS EL MODELO
print("[INFO] COMPILANDO MODELO...")
model = LeNet.build(width=64, height=64, depth=3, classes=2)
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
    metrics=["accuracy"])

# ENTRENAMOS LA RED
print("[INFO] ENTRENANDO RED...")
H = model.fit(x=aug.flow(trainX, trainY, batch_size=BS),
    validation_data=(testX, testY), steps_per_epoch=len(trainX) // BS,
    epochs=EPOCHS, verbose=1)

# GUARDAMOS EL MODELO
print("[INFO] GENERANDO RED...")
model.save(args["model"], save_format="h5")
```

Figura 104. Fragmento de código correspondiente al entrenamiento de la red (2)

El entrenamiento se realiza a través de la ventana de comandos de Python. En la línea de comandos se debe indicar la ruta de la carpeta que contiene las imágenes de entrenamiento y el nombre que se le dará al modelo entrenado. En este caso:

```
python train_LeNet.py --dataset defectos --model defectos_LeNet.model
```

El proceso de entrenamiento puede llevar un tiempo. Modificando los parámetros iniciales se pueden conseguir otros resultados. En ocasiones, la selección de estos parámetros puede ser una tarea difícil sin llegar a obtener los resultados esperados.

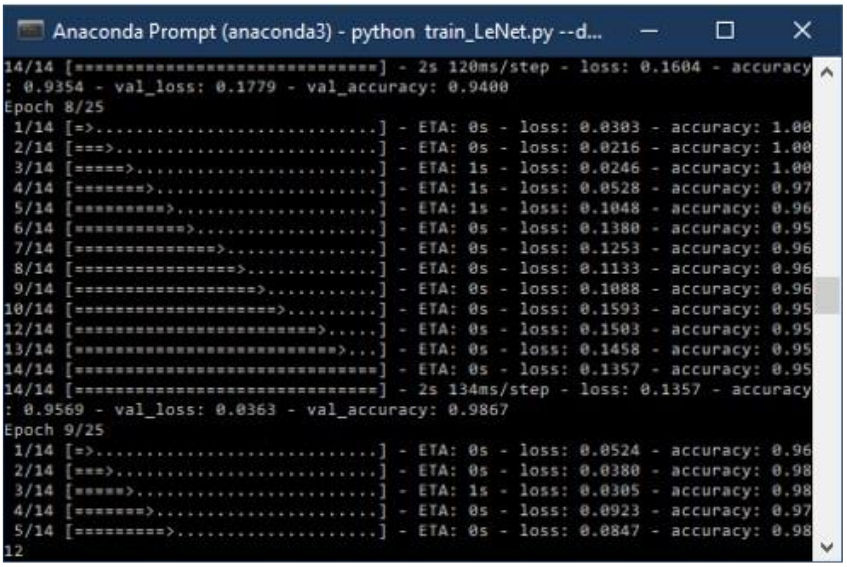


Figura 105. Entrenamiento de la red neuronal

Tras realizar el entrenamiento de la red para 25 *epochs* se alcanza una precisión del 99,33% con unas pérdidas casi nulas, como se puede observar en la Figura 106. Esto probablemente se deba a la gran similitud que tienen todas las imágenes de las indicaciones entre sí, haciendo que el entrenamiento sea sencillo.

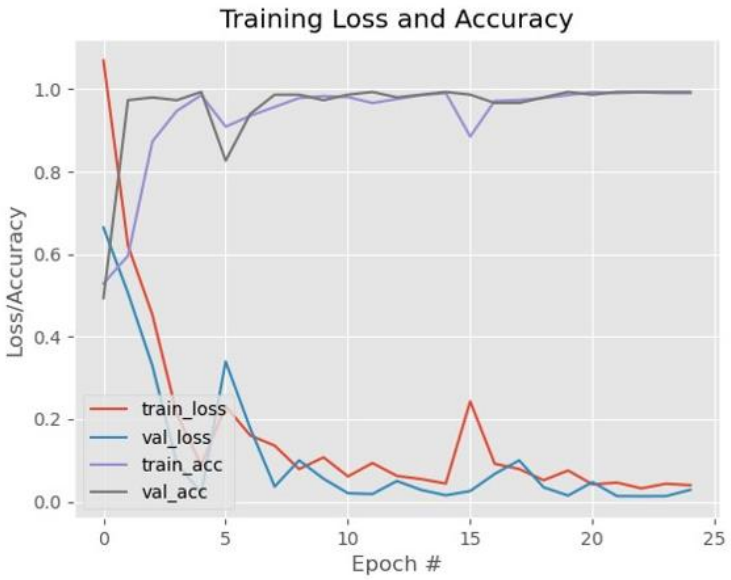


Figura 106. Evolución de la pérdida y la precisión durante el entrenamiento

6.1.4 Clasificación de imágenes a través de la red neuronal

El último paso es evaluar la red neuronal entrenada presentándola imágenes que no haya utilizado durante el entrenamiento, para ello es necesario crear un nuevo subconjunto de datos con dichas imágenes.

En este caso, el código (Figura 107) se encargará de leer una imagen que posteriormente clasificará. Para ello se realiza un ajuste de la imagen convirtiéndola en un vector y añadiendo una dimensión extra, necesaria para trabajar con redes neuronales convolucionales.

Posteriormente se carga el modelo de la red neuronal, que se utiliza para predecir a qué clase pertenece la imagen y etiquetarla en consecuencia. La predicción devuelve un porcentaje de seguridad asociado a cada clase (defecto o no defecto), este valor es el que se toma como referencia para decidir a qué clase pertenece la imagen analizada.

```
# CARGAMOS LA RED NEURONAL
print("[INFO] CARGANDO RED...")
model = load_model(args["model"])

# CLASIFICAMOS LA IMAGEN
(notDefect, Defect) = model.predict(image)[0]

# ETIQUETAMOS LA IMAGEN
label = "Defecto" if Defect > notDefect else "No defecto"
proba = Defect if Defect > notDefect else notDefect
label = "{}: {:.2f}%".format(label, proba * 100)
```

Figura 107. Fragmento de código correspondiente a la clasificación

Para realizar la clasificación se debe introducir el modelo a utilizar y la imagen que se desea clasificar. En este caso las líneas de comandos tienen el siguiente aspecto:

```
python test_network.py --model defectos_LeNet.model --image images/pos_test0001.png
python test_network.py --model defectos_LeNet.model --image images/neg_test0001.png
```

El resultado de ejecución de estas líneas de comando se recoge en la Figura 108. Se puede apreciar que los resultados de clasificación son muy buenos pese a la sencillez de la red neuronal utilizada.

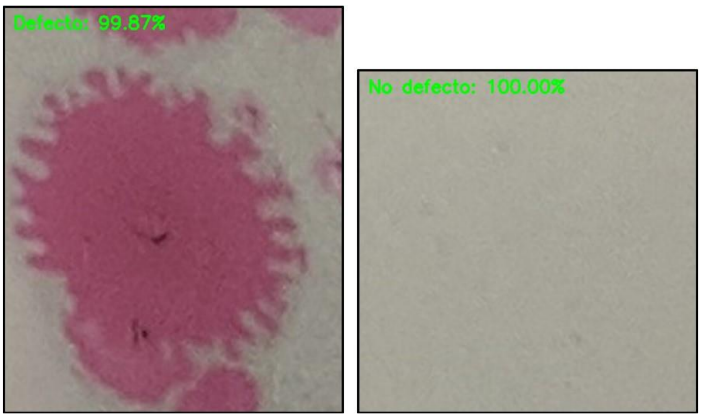


Figura 108. Ejemplo de clasificación

6.1.5 Análisis de resultados

6.1.5.1 Primer entrenamiento

Primero, se va a evaluar el clasificador entrenado con el conjunto de 300 imágenes positivas y 300 imágenes negativas aumentadas. Lo característico de este conjunto es que todas las imágenes se han "estandarizado" para que tengan un ancho de 500 pixeles cada una.

Ya se mencionó que para la evaluación del algoritmo de clasificación de imágenes fue necesario realizar un subconjunto de datos con imágenes que el algoritmo no haya utilizado durante el entrenamiento. En este caso, el nuevo conjunto dispone de 70 imágenes positivas y 70 imágenes negativas.

Debido a la naturaleza del problema, las imágenes serán muy similares a las utilizadas originalmente, pero nunca iguales para no realizar una evaluación del algoritmo errónea. En la Figura 109 se recogen algunas de las imágenes utilizadas durante la evaluación.

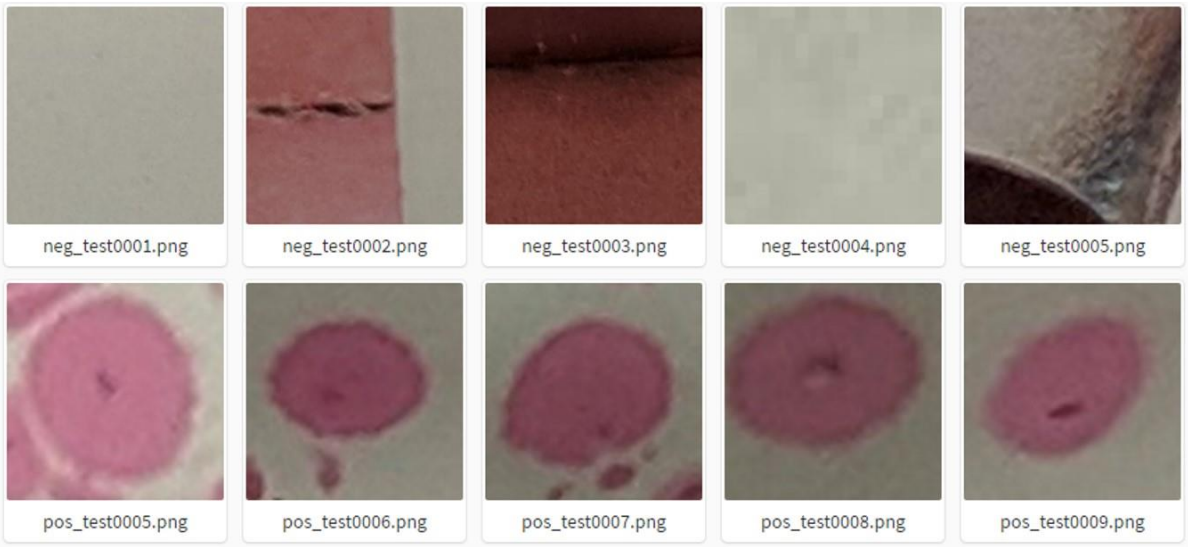


Figura 109. Imágenes del conjunto de datos usado en la primera evaluación

Con el objetivo de facilitar la evaluación del algoritmo, se ha realizado una modificación al código presentado que permite realizar la clasificación automática en un directorio de imágenes. El código completo se encuentra en el anexo al final del documento. En este caso, el comando para ejecutar el código es el siguiente:

```
python test_network_auto.py --model defectos_LeNet.model --path positivas
```

```
python test_network_auto.py --model defectos_LeNet.model --path negativas
```

Una vez ejecutado el código en los dos subconjuntos de imágenes se obtienen los resultados de clasificación mostrados en la Tabla 11. A partir de estos resultados es fácilmente deducible que en la mayor parte de los casos el clasificador actúa correctamente.

Tabla 11. Resultados de clasificación (primer entrenamiento)

	Positivas	Negativas	Total
Defectos	67	3	70
No defectos	2	68	70
Total	69	71	140

Como ya se mencionó anteriormente, utilizando estos resultados es posible obtener una serie de valores (Tabla 12) que permiten evaluar el clasificador, y en caso necesario, permiten comparar su rendimiento con otros algoritmos.

Tabla 12. Rendimiento de clasificación (primer entrenamiento)

Precision	*Recall*	*Specificity*	*Miss rate*	*False alarm*
97%	96%	97%	4%	3%

Se ha conseguido alcanzar un 97% de **precisión** (*precision*). Esto indica que casi todas las imágenes clasificadas como positivas, realmente lo son. De las 69 imágenes que el clasificador ha etiquetado como positivas, solamente 2 no lo eran realmente.

En **exhaustividad** (*recall*) se ha alcanzado un 96%. Este valor también nos indica un buen rendimiento del clasificador. Se disponía de un conjunto inicial de 70 imágenes positivas, de las cuales 67 han sido etiquetadas correctamente.

La **especificidad** (*specificity*) también tiene un valor alto, un 97%. Su significado es muy similar al anterior, pero para las detecciones negativas. En este caso, del total de 70 imágenes negativas, solamente 2 han sido etiquetadas como positivas.

Por definición, la **tasa de pérdida** (*miss rate*) representa lo contrario a la exhaustividad. Del conjunto inicial de 70 imágenes positivas, solamente 3 han sido etiquetadas incorrectamente.

Por último, la **tasa de falsa alarma** (*false alarm*) está estrechamente relacionada con la especificidad, resultando que solamente 2 imágenes han sido etiquetadas como positivas, del total de 70 imágenes negativas.

A continuación, se muestras algunos de los resultados obtenidos durante la clasificación, divididos en cuatro categorías.

En la Figura 110 se recogen algunos de las clasificaciones positivas correctas. En general, los porcentajes de seguridad muy altos, superando el 90% en la mayor parte de los casos.

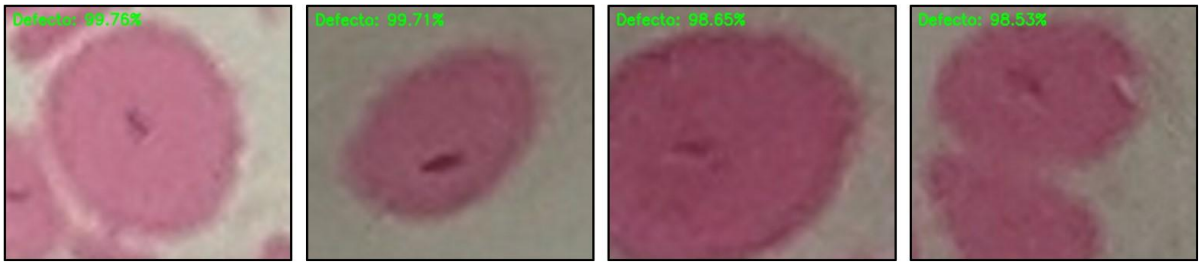


Figura 110. Ejemplo de clasificación positiva de imágenes positivas

Durante la evaluación se detectaron tres clasificaciones incorrectas, mostradas en la Figura 111. Es probable que esto se deba a la resolución de las imágenes, puesto que durante el entrenamiento del algoritmo se utilizaron imágenes aumentadas respecto a su tamaño original.

Se puede observar que la imagen de la izquierda de la Figura 111 posee mejor calidad que el resto de las imágenes, puesto que su tamaño original es mayor.

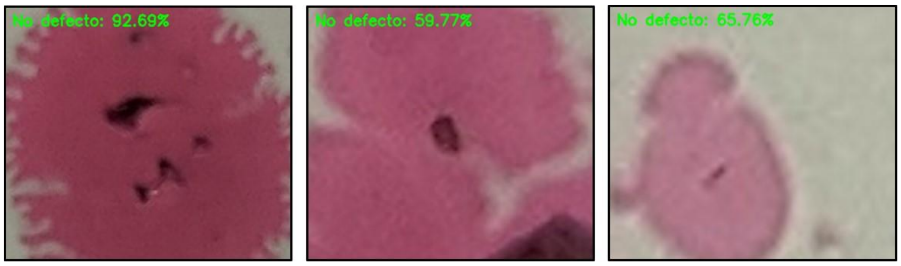


Figura 111. Clasificación negativa de imágenes positivas

En la mayoría de los casos de la clasificación de imágenes negativas se consiguió un porcentaje de seguridad del 100%, algunas de las clasificaciones se pueden observar en la Figura 112. En este conjunto se introdujeron imágenes que podrían llegar a ser consideradas defectos por el algoritmo, debido a su similitud de colores o forma, pero en la mayor parte de los casos la clasificación fue correcta.

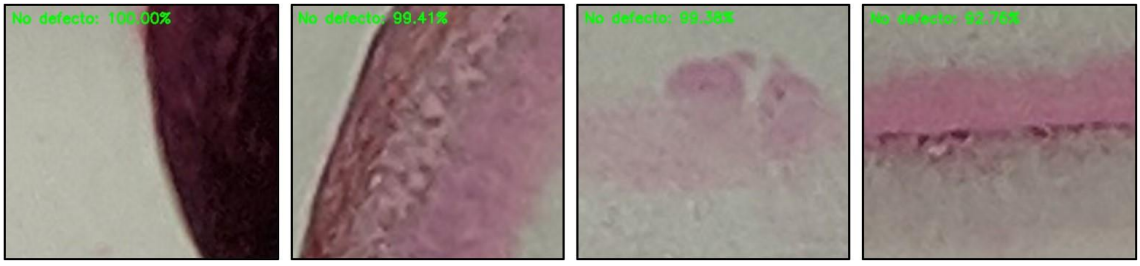


Figura 112. Ejemplo de clasificación negativa de imágenes negativas

La Figura 113 muestra las dos clasificaciones incorrectas para este grupo de imágenes. La primera imagen (izquierda) debido a la calidad de esta podría ser considerada como un defecto a simple vista, puesto que se trata de un recorte de una imagen más grande.

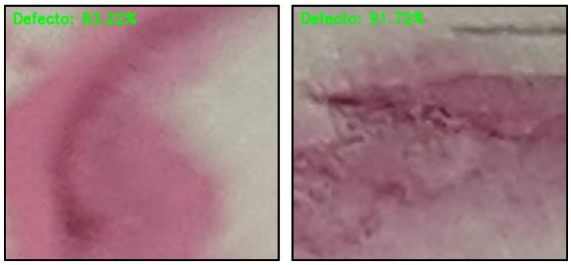


Figura 113. Clasificación positiva de imágenes positivas

6.1.5.2 Segundo entrenamiento

A la vista de los resultados anteriores, se ha decidido modificar el entrenamiento de la red utilizando los fragmentos de imágenes en su tamaño original. Además, en este nuevo entrenamiento se ha reducido el tamaño del conjunto de entrenamiento a 225 imágenes para cada tipo, dejando 75 imágenes escogidas aleatoriamente para formar un nuevo conjunto de evaluación.

En la Figura 114 se muestra el resultado de los dos entrenamientos de forma comparativa. A primera vista parece que el segundo entrenamiento es más efectivo que el primero, por lo que estandarizar las imágenes no ha dado los resultados esperados.

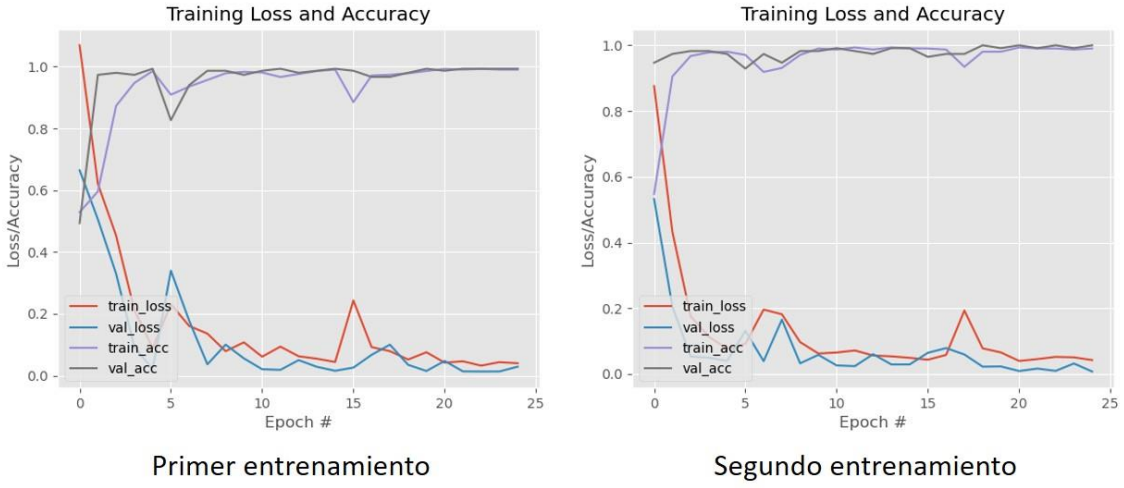


Figura 114. Comparativa entre los entrenamientos

La Tabla 13 muestra los resultados obtenidos utilizando el nuevo conjunto de evaluación que dispone de 75 imágenes distintas a las originales para cada caso.

Tabla 13. Resultados de clasificación (segundo entrenamiento) con el segundo conjunto de evaluación

	Positivas	Negativas	Total
Defectos	75	0	75
No defectos	1	74	75
Total	76	74	150

El rendimiento del clasificador se puede observar en la Tabla 14. Se aprecia que los resultados son mejores en comparación con el primer entrenamiento. En el conjunto total de la prueba, solo se ha realizado una detección incorrecta que se corresponde con la clasificación de una imagen negativa como positiva.

Tabla 14. Rendimiento de clasificación (segundo entrenamiento) con el segundo conjunto de evaluación

Precision	*Recall*	*Specificity*	*Miss rate*	*False alarm*
99%	100%	99%	0%	1%

Para poder comparar los resultados con los obtenidos en el primer caso, ahora se utiliza el mismo conjunto de evaluación utilizado anteriormente, obteniendo la Tabla 15.

Tabla 15. Resultados de clasificación (segundo entrenamiento) con el primer conjunto de evaluación

	Positivas	Negativas	Total
Defectos	69	1	70
No defectos	2	68	70
Total	71	69	140

A raíz de estos resultados (Tabla 16), se observa que con este nuevo entrenamiento se ha reducido en gran parte el error producido durante la clasificación de imágenes positivas. En este caso solamente una de las imágenes ha sido clasificada incorrectamente.

Las dos imágenes negativas que se habían clasificado antes incorrectamente siguen siendo clasificadas como positivas.

Tabla 16. Rendimiento de clasificación (segundo entrenamiento) con el primer conjunto de evaluación

Precision	*Recall*	*Specificity*	*Miss rate*	*False alarm*
97%	99%	97%	1%	3%

En la Figura 115 se muestra el error de clasificación para la imagen positiva antes mencionada.

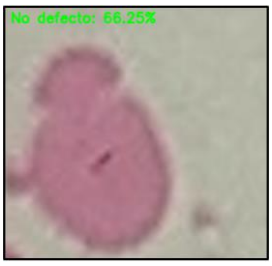


Figura 115. Clasificación negativa de imágenes positivas (segundo entrenamiento)

Las dos clasificaciones erróneas para el conjunto de imágenes negativas se vuelven a mostrar en la Figura 116.

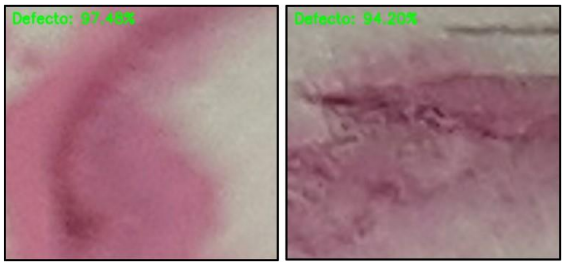


Figura 116. Clasificación positiva de imágenes positivas (segundo entrenamiento)

6.2 CONCLUSIONES DEL CAPÍTULO

El desarrollo realizado en esta capítulo supone una base para el posible desarrollo de un algoritmo basado en *Deep Learning* mucho más complejo y sofisticado.

A raíz de los resultados obtenidos, se observa que, pese a la utilización de una red neuronal convolucional muy simple, la utilización de algoritmos basados en este campo resulta muy prometedora. Aunque la comparación con aquellos resultados obtenidos en el capítulo 4 no tiene realmente un sentido práctico debido a las grandes diferencias entre ambos algoritmos, se puede observar que se ha conseguido mejorar sustancialmente la precisión de la clasificación.

Para el caso de aplicación de este algoritmo de forma complementaria al ensayo de líquidos penetrantes, no se tiene la capacidad de clasificar los defectos puesto que las indicaciones proporcionadas son muy similares entre sí.

En el caso de la aplicación del Deep Learning en la detección de defectos en piezas sin ensayar, sería posible diseñar una red neuronal mucho más profunda y compleja partiendo del desarrollo propuesto, capaz de clasificar varios tipos de defectos, siendo necesaria una base de datos compleja que contenga de forma diferenciada dichos defectos.

Además, el desarrollo lógico en este caso sería diseñar un algoritmo basado en la detección de objetos, un algoritmo mucho más complejo que además de clasificar los defectos en las distintas clases sería capaz de localizarlos en la pieza. Este algoritmo requiere también una base de datos completa de los defectos, con numerosas imágenes en las que estén etiquetados y localizados los posibles defectos susceptibles de aparecer.

Como se ha mencionado, una de las mayores limitaciones durante el desarrollo de este algoritmo ha sido no disponer de una base de datos completamente robusta. Además, su obtención ha supuesto un trabajo muy tedioso por lo que la aplicación de este algoritmo en la detección de defectos en las piezas sin ensayar se ha tenido que descartar por falta de recursos.

CAPÍTULO 7: CONCLUSIONES Y LÍNEAS FUTURAS

El desarrollo del proyecto y el posterior análisis de los resultados del apartado anterior, permiten llegar a una serie de conclusiones que se presentan en este capítulo. Además, se proponen una serie de líneas futuras tomando como base lo desarrollado en este documento.

7.1 CONCLUSIONES

De manera principal, se ha logrado alcanzar el objetivo principal del proyecto, mostrando la posibilidad de aplicar tanto la visión artificial como el *Deep Learning* en la problemática presentada. Con ello, se sientan unas bases de conocimiento que permiten seguir avanzando en el desarrollo de multitud de aplicaciones similares.

7.1.1 Conclusiones generales

En relación con los objetivos planteados en el primer capítulo del documento, se han llegado a las siguientes conclusiones:

La técnica de ***laser cladding*** supone una buena solución para la obtención de recubrimientos, pero requiere tener gran cuidado y control durante su aplicación, para permitir la correcta solidificación del metal y evitar dentro de lo posible la aparición de grietas y poros.

Los **ensayos no destructivos** se muestran como una importante herramienta de prevención de accidentes y mantenimiento de la calidad y son imprescindibles en multitud de sectores. Su mayor desventaja es el tiempo requerido para la inspección y la alta experiencia previa que se debe poseer para realizar el ensayo correctamente. La **inspección visual** continúa estando presente en cualquiera de los ensayos y se ve mejorada con la aplicación de tecnologías más avanzadas como la visión artificial. La aplicación de **líquidos penetrantes** permite observar fácilmente los defectos superficiales de las piezas, pero en ocasiones puede llegar a ser confuso si existen zonas con muchos defectos concentrados.

La **visión artificial** es un campo en continua mejora, cuyos algoritmos son cada vez más complejos y preciosos. En cuanto al ***Deep Learning***, se muestra como el camino a seguir en el desarrollo de aplicaciones de reconocimiento y clasificación de objetos. La aplicación de esta tecnología ha permitido mejorar los resultados de forma notable y deja la puerta abierta al desarrollo de aplicaciones más complejas y que permitan resolver multitud de problemas.

En el documento, se ha mostrado el desarrollo de **tres alternativas** distintas, con sus propias ventajas y limitaciones.

- El algoritmo propuesto para realizar la detección de las indicaciones proporcionadas por el ensayo de líquidos penetrantes se muestra como una solución válida para imágenes de buena calidad y que presenten indicaciones de forma clara, permitiendo aproximar las dimensiones de los defectos localizados.

- La detección de defectos directamente sobre las piezas se ha mostrado como una alternativa completamente válida, pero que requiere de reducir al máximo las excitaciones externas de las imágenes obtenidas, ya sea modificando el proceso de captura o realizando un tratamiento previo a las piezas, para obtener llevar a cabo la ejecución. Además, la aplicación de este algoritmo permite diseñar un proceso de evaluación a través del análisis del histograma de aparición de defectos.
- El algoritmo de clasificación se ha presentado como el camino a seguir, sentando unas bases para el desarrollo de un posible detector, que permita localizar y clasificador las indicaciones.

7.1.2 Conclusiones particulares

A raíz de todo el desarrollo seguido, de los resultados obtenidos y de las conclusiones planteadas, es posible destacar unas conclusiones más particulares:

- La adquisición de la imagen es un punto fundamental en el proceso de inspección de defectos. Es muy importante disponer de buenas condiciones de iluminación, evitando la aparición de brillos y sombras que puedan dificultar el proceso de tratamiento posterior de las imágenes. En el proyecto no se ha desarrollado directamente esta parte, pero sería conveniente pararse en ella para futuros desarrollos.
- La utilización del algoritmo de detección de defectos es un buen punto de partida para realizar la inspección de piezas, pero está muy limitado a la buena visibilidad de las indicaciones o los defectos en las imágenes. Un factor muy limitante es que no permite su distinción.
- La aplicación de los algoritmos de detección directamente en las piezas requiere de un tratamiento previo para la eliminación de brillos. La solución propuesta tras la aplicación del aerosol se muestra como una alternativa válida, pero sería conveniente desarrollar este aspecto en más profundidad, analizando otros posibles métodos para reducir el brillo o modificando el proceso de captura de las imágenes.
- Los algoritmos de clasificación de muestran como una buena alternativa y como base de un algoritmo más complejo. Las redes neuronales convolucionales han demostrado ser una solución con un elevado potencial para el desarrollo de algoritmos de detección y clasificación. Mediante la utilización de una red más compleja y una base de datos completa se podría dotar al algoritmo desarrollado de la capacidad de localizar los defectos, consiguiendo una herramienta muy robusta en este ámbito.
- Por último, es vital disponer de una completa base de datos con todos los defectos susceptibles de aparecer en la pieza, puesto que es el punto de partida para realizar el correcto entrenamiento del algoritmo. La disponibilidad de esta base de datos permitiría desarrollar el algoritmo de clasificación para el caso de la detección de defectos en bruto.

7.2 LÍNEAS FUTURAS

El presente proyecto sienta las bases para el desarrollo de un sistema automático de detección de defectología, centrándose en unas posibles líneas de mejora, algunas muy relacionadas con las conclusiones anteriores:

1) Diseño e implantación de un sistema automático de detección

Sería altamente interesante realizar un prototipo del sistema de detección de defectos a través del diseño de una estructura dispuesta con una cámara, sensores, CPU y demás sistemas necesarios. Este dispositivo permitiría realizar pruebas en tiempo real, permitiendo escoger la mejor posición de adquisición de la imagen, la iluminación de la escena y otros factores que no es posible controlar con la simple toma de imágenes. Además, sería posible realizar ajustes en el algoritmo utilizado de forma directa, analizando los resultados obtenidos.

2) Creación de una base de datos completa

Una de las mayores limitaciones de este trabajo ha sido no disponer de suficientes imágenes que permitiesen distinguir cada tipo de defecto. Disponer de una base de datos completa con muestras de cada defecto es de suma importancia puesto que permitiría desarrollar algoritmos más completos.

3) Desarrollo de un algoritmo basado en la detección de objetos

Ya se ha mencionado que existe un tercer tipo de algoritmo, muy relacionado con el algoritmo de clasificación. A través de la base de datos mencionada, sería posible entrenar una red neuronal más compleja y combinar la clasificación de defectos con un algoritmo capaz de localizarlos en la pieza. Esta opción supondría un algoritmo más complejo y que se beneficiaría del enfoque mencionado a continuación.

4) Detección y clasificación de defectos en bruto

El objetivo final de esta línea de trabajo sería realizar no solo la detección sino también la clasificación de los defectos sin la necesidad de realizar previamente el ensayo por líquidos penetrantes. Ya se ha evaluado la posibilidad de realizar la detección a través de algoritmos de visión artificial, pero sería altamente interesante dotar a los algoritmos de la capacidad de clasificar los distintos defectos de las piezas. Para llegar a desarrollar esta alternativa, es importante disponer de la base de datos mencionada.

5) Mejorar los algoritmos planteados

Se ha mostrado cómo a través de la posterior utilización de un histograma de aparición de defectos es posible evaluar de forma rápida y sencilla el estado de las piezas. Este histograma en principio se obtiene de forma manual con la información proporcionada por el algoritmo. Dotar al algoritmo la capacidad de decidir de forma autónoma el estado de las piezas, no sólo mostrando los defectos encontrados, sino también evaluando la gravedad de estos, supondría una valiosa mejora.

REFERENCIAS BIBLIOGRÁFICAS

[1] M. Merklein y J. Lechler, «Investigation of the thermo-mechanical properties of hot stamping steels,» *Journal of Materials Processing Technology,* vol. 177, nº 1-3, pp. 452-455, 2006.

[2] L. Quintino, «Overview of coating technologies,» de *Surface Modification by Solid State Processing*, Lisbon, Portugal, Woodhead Publishing, 2014, pp. 1-24.

[3] Red de Ingeniería de Superficies y Capas Delgadas (INGESNET), «La ingeniería de superficies,» [En línea]. Available: http://www.ingesnet.org/view/view_paginas.php?menu=111&id=62&lang=es. [Último acceso: Marzo 2021].

[4] I. Etxeberria, G. Alberdi y I. Vicario, «Láser "cladding": novedosa técnica de aporte de material,» 2006. [En línea]. Available: https://www.interempresas.net/MetalMecanica/Articulos/14238-Laser-cladding-novedosa-tecnica-de-aporte-de-material.html. [Último acceso: Abril 2021].

[5] Q. Liu, M. Janardhana, B. Hinton, M. Brandt y K. Sharp, «Laser cladding as a potential repair technology for damaged aircraft components,» *International Journal of Structural Integrity,* vol. 2, nº 3, pp. 314-331, 2011.

[6] I. Tabernero Campos, A. Calleja Ochoa, E. Ukar Arrien y L. N. López de Lacalle Marcaide, «Laser cladding en 5 ejes continuos para la fabricación de piezas de alto valor añadido,» XIX Congreso Nacional de Ingeniería Mecánica.

[7] N. Łukasz y W. Marta, «Finishing Surface After Regeneration with Laser Cladding,» *Procedia Engineering,* vol. 192, pp. 1012-1015, 2017.

[8] M. Rombouts, G. Maes, W, Hendrix, E. Delarbre y F. Motmans, «Surface Finish after Laser Metal Deposition,» *Physics Procedia,* vol. 41, pp. 810-814, 2013.

[9] J. J. Candel, V. Amigó, J. Sampedro y V. Bonache, «Evaluación de las transformaciones estructurales en recubrimientos de WC10Ni depositados por laser cladding sobre acero para herramienta EN 12379,» *Revista de metalurgia,* vol. 47, nº 4, pp. 355-364, 2011.

[10] S. D. Sun, R. Mohammed, M. Brandt, G. Clark, Q. Liu y M. Janardhana, «Analysis of defects in laser cladding of high strength steel for aerospace application,» de *Incorporating Sustainable Practice in Mechanics and Structures of Materials*, 2010.

[11] B. Haldar y P. Saha, «Identifying defects and problems in laser cladding and suggestions of some remedies for the same,» *Materials Today: Proceedings,* vol. 5, nº 5, part 2, pp. 13090-13101, 2018.

[12] Asociación Española de Ensayos no Destructivos, «Introducción a los END,» AEND, 2016.

[13] A. J. McEvily, «Failures in inspection procedures: case studies,» *Engineering Failure Analysis,* vol. 11, nº 2, pp. 167-176, 2004.

[14] R. S. Lledó, «Apuntes asignatura Tecnología de Materiales de la UMA,» 2013-2014. [En línea]. Available: https://www.raquelserrano.com/apuntes-2/cuarto/tecnologia-de-materiales/. [Último acceso: Febrero 2021].

[15] M. I. Hamakareem, «Liquid Penetrant Test on Concrete: Purpose, Procedure, and Applications,» [En línea]. Available: https://theconstructor.org/practical-guide/liquid-penetrant-test-concrete/9542/. [Último acceso: Abril 2021].

[16] K. Abend, «Fully Automated Dye-Penetrant Inspection of Automotive Parts,» *SAE Technical Paper 980739,* p. 8, 1998.

[17] resnick_halliday, «Líquidos Penetrantes,» [En línea]. Available: https://www.monografias.com/trabajos31/liquidos-penetrantes/liquidos-penetrantes.shtml. [Último acceso: Marzo 2021].

[18] H.-H. Chu y Z.-Y. Wang, «A vision-based system for post-welding quality measurement and defect detection,» *Int J Adv Manuf Technol,* vol. 86, p. 3007–3014, 2016.

[19] L. Song, X. Li, Y. Yang, X. Zhu, Q. Guo y H. Yang, «Detection of Micro-Defects on Metal Screw Surfaces Based on Deep Convolutional Neural Networks,» *Sensors,* vol. 18, nº 3709, 2018.

[20] J. F. Vélez Serrano, A. B. Moreno Díaz, C. Á. Sánchez y J. L. Esteban Sánchez-Marín, Visión por computador, S.L. - DYKINSON, 2003.

[21] A. González Marcos, F. J. Martínez de Pisón Ascacíbar, A. V. Pernía Espinoza, F. Alba Elías, M. Castejón Limas, J. Ordieres Meré y E. Vergara González, Técnicas y algoritmos básicos de visión artificial, Universidad de La Rioja, 2006.

[22] F. Gómez Rodríguez y M. J. Domínguez Morales, «Fundamentos de la Visión Artificial,» Departamento de Arquitectura y Tecnología de Computadores, Universidad de Sevilla.

[23] L. Salgado, «Visión Artificial: Fundamentos y Aplicaciones,» E.T.S. Ingenieros de Telecomunicación, Universidad Politécnica de Madrid, 2007.

[24] A. Rosebrock, Deep Learning for Computer Vision with Python, PyImageSearch, 2017.

[25] S. Sharma, «Activation Functions in Neural Networks,» Towards Data Science, 6 Septiembre 2017. [En línea]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. [Último acceso: Abril 2021].

[26] S. R. Rath, «Activation Functions in Neural Networks,» Debugger Cafe, 1 Abril 2019. [En línea]. Available: https://debuggercafe.com/activation-functions-in-neural-networks/. [Último acceso: Abril 2021].

[27] S. Barua, F. Liou, J. Newkirk y T. Sparks, «Vision-based defect detection in laser metal deposition process,» *Rapid Prototyping Journal,* vol. 20, nº 1, pp. 77-85, 2014.

[28] J. Sun, C. Li, X.-J. Wu, V. Palade y W. Fang, «An Effective Method of Weld Defect Detection and Classification Based on Machine Vision,» *IEEE Transactions on Industrial Informatics,* vol. 15, nº 12, pp. 6322-6333, 2019.

[29] Y. Min, B. Xiao, J. Dang, B. Yue y T. Cheng, «Real time detection system for rail surface defects based on machine vision,» *EURASIP Journal on Image and Video Processing,* vol. 2018, nº 3, 2018.

[30] N. J. Shipway, P. Huthwaite, M. J. S. Lowe y T. J. Barden, «Performance Based Modifications of Random Forest to Perform Automated Defect Detection for Fluorescent Penetrant Inspection,» *Journal of Nondestructive Evaluation,* vol. 38, nº 37, 2019.

[31] Q. Jiang, D. Tan, Y. Li, S. Ji, C. Cai y Q. Zheng, «Object Detection and Classification of Metal Polishing Shaft Surface Defects Based on Convolutional Neural Network Deep Learning,» *Applied Sciences,* vol. 10, nº 87, 2020.

[32] «Sitio web de Python,» [En línea]. Available: https://www.python.org/. [Último acceso: Febrero 2019].

[33] «Sitio web de Anaconda,» [En línea]. Available: https://www.anaconda.com/. [Último acceso: Febrero 2019].

[34] «Sitio web de OpenCV,» [En línea]. Available: https://opencv.org/. [Último acceso: Febrero 2019].

[35] OpenCV, «OpenCV-Python Tutorials,» [En línea]. Available: https://docs.opencv.org/master/d6/d00/tutorial_py_root.html. [Último acceso: Marzo 2021].

[36] GeeksforGeeks, «OpenCV Python Tutorial,» 2021. [En línea]. Available: https://www.geeksforgeeks.org/opencv-python-tutorial/. [Último acceso: Marzo 2021].

[37] TrProgramación, «Blog: Tutor de Programación,» 2017. [En línea]. Available: http://acodigo.blogspot.com/p/tutorial-opencv.html. [Último acceso: Marzo 2021].

[38] A. Rosebrock, «PyImageSearch: Computer Vision, Deep Learning, and OpenCV,» PyImageSearch, 2021. [En línea]. Available: https://www.pyimagesearch.com/. [Último acceso: Marzo 2021].

[39] A. Rosebrock, «OpenCV Tutorial: A Guide to Learn OpenCV,» 19 Julio 2018. [En línea]. Available: https://www.pyimagesearch.com/2018/07/19/opencv-tutorial-a-guide-to-learn-opencv/. [Último acceso: Marzo 2021].

[40] Y. LeCun, L. Bottou, Y. Bengio y P. Haffner, «Gradient-based learning applied to document recognition,» *Proceedings of the IEEE,* vol. 86, nº 11, pp. 2278-2324, 1998.

ANEXO A: CÓDIGO DEL ALGORITMO DE DETECCIÓN DE DEFECTOS

DETECTOR DE CONTORNOS CON ENSAYO PREVIO

ARCHIVO *defectos_v0.8.py*

```
# PAQUETES NECESARIOS
import cv2
import time
import imutils
import numpy as np
import tkinter as tk
from tkinter import filedialog
from imutils import paths

# VARIABLES INICIALES
version = "0.8"
hora = time.strftime("%H:%M:%S")
fecha = time.strftime("%d/%m/%y")

img_new = img_actual = img_gris = None



# EL CODIGO IMPLEMENTA UNA INTERFAZ GRAFICA PARA FACILITAR EL MANEJO
# DEL PROGRAMA

class detectorDefectos:

    def __init__(self):

        # CONFIGURACION DE LA VENTANA PRINCIPAL DE LA APLICACION
        self.ventanaPrincipal = tk.Tk()

        self.ventanaPrincipal.title("D&P {}".format(version))
        self.ventanaPrincipal.resizable(False, False)
        self.ventanaPrincipal.geometry("800x400")
        self.ventanaPrincipal.iconbitmap('./icono_det.ico')

        self.Frame1 = tk.LabelFrame(self.ventanaPrincipal)
        self.Frame1.pack(side=tk.LEFT, fill="both", expand="yes",
                         padx=10, pady=10)

        self.Frame2 = tk.LabelFrame(self.ventanaPrincipal)
        self.Frame2.pack(side=tk.RIGHT, fill="both", expand="yes",
                         padx=10, pady=10)

        self.LabelInfo = tk.Label(self.Frame2,
                                  text="Ninguna imagen seleccionada")
        self.LabelInfo.pack(side=tk.TOP)

        self.Label1 = tk.Label(self.Frame2, text="Versión {}\n{} {}"
                               .format(version, fecha, hora))
        self.Label1.pack(side=tk.BOTTOM)

        # CONFIGURACION DEL MENU DE LA INTERFAZ
        barraMenu = tk.Menu(self.ventanaPrincipal)
        self.ventanaPrincipal.config(menu=barraMenu, width=300, height=300)
```

```
        menuArchivo = tk.Menu(barraMenu, tearoff=0)
        menuArchivo.add_command(label="Abrir imagen...",
                                command=self.abrirImagen)
        menuArchivo.add_separator()
        menuArchivo.add_command(label="Cerrar todo",
                                command=self.cerrarImg)
        menuArchivo.add_separator()
        menuArchivo.add_command(label="Salir",
                                command=self.salirAplicacion)

        menuFiltrado = tk.Menu(barraMenu, tearoff=0)
        menuFiltrado.add_command(label="Cambiar a escala de grises",
                                 command=self.filtrarGris)
        menuFiltrado.add_command(label="Filtro de Gauss",
                                 command=self.filtrarGauss)
        menuFiltrado.add_command(label="Filtro de la mediana",
                                 command=self.filtrarMediana)

        menuSegmentacion = tk.Menu(barraMenu, tearoff=0)
        menuSegmentacion.add_command(label="Umbralización fija",
                                     command=self.segUmbral)
        menuSegmentacion.add_command(label="Umbralización adaptativa",
                                     command=self.segUmbralAd)
        menuSegmentacion.add_command(label="Ecualización del histograma",
                                     command=self.segEqual)

        memuDetectar = tk.Menu(barraMenu, tearoff=0)
        memuDetectar.add_command(label="Detector de contornos",
                                 command=self.detectarManual)
        memuDetectar.add_command(label="Detector de contornos (directorio)",
                                 command=self.detectarAuto)

        menuAyuda = tk.Menu(barraMenu, tearoff=0)
        menuAyuda.add_command(label="Acerca de...",
                              command=self.infoAdicional)

        barraMenu.add_cascade(label="Archivo", menu=menuArchivo)
        barraMenu.add_cascade(label="Filtrado", menu=menuFiltrado)
        barraMenu.add_cascade(label="Segmentación", menu=menuSegmentacion)
        barraMenu.add_cascade(label="Detección", menu=memuDetectar)
        barraMenu.add_cascade(label="Ayuda", menu=menuAyuda)

        self.ventanaPrincipal.mainloop()

    # FUNCION INTERFAZ ABRIR IMAGEN
    def abrirImagen(self):
        global ruta_imagen, img_ruta, img_actual

        #Especificamos los tipos de archivos correspondientes a imagenes
        ruta_imagen = filedialog.askopenfilename(title="Abrir imagen",
                                                 filetypes = [
            ("Archivos de mapa de bits (*.bmp, *.dib)","*.bmp *.dib"),
            ("JPEG (*.jpg, *.jpeg, *.jpe, *.jfif)","*.jpg *.jpeg *.jpe *.jfif"),
            ("PNG (*.png)","*.png"),
            ("Todos los archivos de imagen",
             "*.bmp *.dib *.jpg *.jpeg *.jpe *.jfif *.png")])
```

```
        if len(ruta_imagen) > 0:

            # Abrimos la imagen de la ruta seleccionada
            img_new = cv2.imread(ruta_imagen)
            img_new_red = imutils.resize(img_new, height=800)
            img_actual = img_new_red
            self.LabelInfo.configure(text="{}".format(ruta_imagen))
            cv2.imshow("Imagen original", img_actual)
            img_new = img_new_red = None


    # FUNCION NIVELES DE GRIS
    def filtrarGris(self):
        global img_actual, img_gris

        if img_actual is None:
            tk.messagebox.showerror("Error","Ninguna imagen seleccionada.")

        else:
            img_gris = cv2.cvtColor(img_actual, cv2.COLOR_BGR2GRAY)
            img_actual = img_gris
            cv2.imshow("Escala de grises", img_actual)
            img_gris = None

    #FUNCION FILTRO DE GAUSS
    def filtrarGauss(self):
        global img_actual

        if img_actual is None:
            tk.messagebox.showerror("Error","Ninguna imagen seleccionada.")

        else:
            img_gauss = cv2.GaussianBlur(img_actual, (7,7), 0)
            img_actual = img_gauss
            cv2.imshow("Filtro de Gauss", img_actual)
            img_gauss = None

    #FUNCION FILTRO DE GAUSS
    def filtrarMediana(self):
        global img_actual

        if img_actual is None:
            tk.messagebox.showerror("Error","Ninguna imagen seleccionada.")

        else:
            img_med = cv2.medianBlur(img_actual, 7)
            img_actual = img_med
            cv2.imshow("Filtro de la mediana", img_actual)
            img_med = None

    #FUNCION UMBRALIZACION FIJA
    def segUmbral(self):
        global img_actual

        if img_actual is None:
            tk.messagebox.showerror("Error","Ninguna imagen seleccionada.")

        else:
            _, img_umb = cv2.threshold(img_actual, 150, 255, cv2.THRESH_BINARY)
            img_actual = img_umb
            cv2.imshow("Umbral fijo", img_actual)
            img_umb = None
```

```
#FUNCION UMBRALIZACION ADAPTATIVA
def segUmbralAd(self):
    global img_actual
    if img_actual is None:
        tk.messagebox.showerror("Error","Ninguna imagen seleccionada.")
    else:
        img_umb_ad = cv2.adaptiveThreshold(img_actual, 255,
                                           cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                           cv2.THRESH_BINARY, 11, 2)
        img_actual = img_umb_ad
        cv2.imshow("Umbral adaptativo", img_actual)
        img_umb_ad = None

#FUNCION ECUALIZACION
def segEqual(self):
    global img_actual
    if img_actual is None:
        tk.messagebox.showerror("Error","Ninguna imagen seleccionada.")
    else:
        img_actual = cv2.cvtColor(img_actual, cv2.COLOR_BGR2GRAY)
        img_eq = cv2.equalizeHist(img_actual)
        img_actual = img_eq
        cv2.imshow("Histograma ecualizado", img_actual)
        img_eq = None

# APLICACIÓN DE FILTROS Y POSTERIOR DETECCIÓN DE CONTORNOS
def FuncionDetectar(self):
    # Conversión de la imagen a niveles de gris
    gris = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Aplicación de un filtro de Gauss para reducir el ruido
    gauss = cv2.GaussianBlur(gris, (7,7), 0)
    # Umbralización para obtener una imagen binaria
    _, umb = cv2.threshold(gauss, 0, 255, cv2.THRESH_BINARY_INV |
                           cv2.THRESH_TRIANGLE)
    # Operación de dilatación
    kernel = np.ones((5,5), np.uint8)
    opn = cv2.morphologyEx(umb, cv2.MORPH_OPEN, kernel)
    # Búsqueda de contornos en la imagen
    contours, _ = cv2.findContours(opn, cv2.RETR_CCOMP,
                                   cv2.CHAIN_APPROX_SIMPLE)
    # Inicializamos las detecciones a 0
    dets=0
    # Dibujo de círculos en aquellos lugares donde se han detectado contornos
    for c in contours:
        area = cv2.contourArea(c)
        if area > 100 and area < 100000:
            (x,y),radius = cv2.minEnclosingCircle(c)
            center = (int(x),int(y))
            radius = int(radius)
            if radius < 100:
                cv2.circle(img, center, radius, (0,255,0), 2)
                dets = dets + 1
                dim = "Defecto {}: (x, y)={} , \
                    R={}, area={}".format(dets,center,radius,area)
                print(dim)
    # Mostramos el número de detecciones en la imagen
    text1 = "Posibles defectos detectados: {}".format(dets)
    cv2.putText(img, text1, (20, img.shape[0]-10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6,(255,255,255),1)
    # Mostramos la imagen
    cv2.imshow("Detecciones", img)
```

```
# FUNCIÓN DETECCIÓN EN UNA SOLA IMAGEN
def detectarManual(self):
    global ruta_imagen, img

    # Especificamos los tipos de archivos correspondientes a imagenes
    ruta_imagen = filedialog.askopenfilename(title="Abrir imagen",
                                             filetypes = [
        ("Archivos de mapa de bits (*.bmp, *.dib)","*.bmp *.dib"),
        ("JPEG (*.jpg, *.jpeg, *.jpe, *.jfif)","*.jpg *.jpeg *.jpe *.jfif"),
        ("PNG (*.png)","*.png"),
        ("Todos los archivos",
         "*.bmp *.dib *.jpg *.jpeg *.jpe *.jfif *.png")])
    # Si la ruta es correcta, leemos la imagen
    if len(ruta_imagen) > 0:
        # Leemos y redimensionamos la imagen
        img = cv2.imread(ruta_imagen)
        img = imutils.resize(img, height=min(800, img.shape[0]))
        # Llamamos a la función
        self.FuncionDetectar()
        cv2.waitKey(0)
        cv2.destroyAllWindows()
    # Aviso en caso de ruta incorrecta
    else:
        tk.messagebox.showerror("Error","Ninguna imagen seleccionada.")

#FUNCION INTERFAZ DETECCION EN UN DIRECTORIO
def detectarAuto(self):
    global directorio, img
    i=int(0)
    # Especificamos la ruta de las imagenes
    directorio = filedialog.askdirectory(title="Seleccione el directorio")
    # Bucle para todas las imagenes
    for imagen in paths.list_images(directorio):
        img = cv2.imread(imagen)
        img = imutils.resize(img, height=min(800, img.shape[0]))
        # Llamamos a la función
        self.FuncionDetectar()
        cv2.waitKey(0)
        cv2.destroyAllWindows()
        i+=1

# FUNCIÓN INTERFAZ ACERCA DE...
def infoAdicional(self):
    tk.messagebox.showinfo("Acerca de...",
                           "Versión {}\n\nPrograma demo que permite: \
                               \n- Visualización de imágenes \
                               \n- Aplicación de filtros \
                               \n- Detección de contornos \
                               \n\nRealizado con Python 3.8.5 y OpenCV 4.4.0"
                               .format(version))

# FUNCIÓN INTERFAZ SALIR DEL PROGRAMA
def salirAplicacion(self):
    valor = tk.messagebox.askquestion("Salir",
                                      "¿Desea salir de la aplicación?")
    if valor=="yes":
        cv2.destroyAllWindows()
        self.ventanaPrincipal.destroy()
```

```
    # FUNCIÓN CERRAR TODO
    def cerrarImg(self):
        valor = tk.messagebox.askquestion("Cerrar todo",
                                          "¿Desea cerrar las imágenes abiertas?")
        if valor=="yes":
            cv2.destroyAllWindows()

detector = detectorDefectos()
cv2.destroyAllWindows()
```

PIEZAS SIN TRATAR

ARCHIVO *seleccion_operaciones_SINT.py*

```
import cv2
import imutils
import numpy as np

# CARGAMOS LA IMAGEN
img = cv2.imread("defectos_10.jpg")

# REALIZAMOS EL PRE-PROCESADO AJUSTANDO EL TAMAÑO Y EL COLOR
img = imutils.resize(img, height=min(800, img.shape[0]))
cv2.imshow("Original", img)
gris = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# REALIZAMOS UN SUAVIZADO MEDIANTE UN FILTRO DE GAUSS 7X7
gris = cv2.GaussianBlur(gris, (7,7), 0)
cv2.imshow("Gauss", gris)

# CALCULAMOS EL UMBRAL OPTIMO
t, umb = cv2.threshold(gris, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_TRIANGLE)
cv2.imshow("Imagen con umbral fijo optimo", umb)
print(t)

# REALIZAMOS UNA APERTURA 5X5
kernel = np.ones((5,5),np.uint8)
opening = cv2.morphologyEx(umb, cv2.MORPH_OPEN, kernel)
cv2.imshow("Imagen con umbral fijo abierta", opening)

# OBTENEMOS LOS CONTORNOS DE LA IMAGEN
contours, _ = cv2.findContours(opening, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)
det = 0
for c in contours:
    area = cv2.contourArea(c)
    if area > 10 and area < 5000:
        cv2.drawContours(img, [c], 0, (0, 255, 0), 2, cv2.LINE_AA)
        det = det + 1

# MOSTRAMOS LAS DETECCIONES
text= "Posibles defectos: {}".format(det)
cv2.putText(img, text, (20, img.shape[0]-10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.6,(255,255,255),1)
cv2.imshow("Detecciones", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

PIEZAS TRATADAS

ARCHIVO *seleccion_operaciones_SIN.py*

```
import cv2
import imutils
import numpy as np

# CARGAMOS LA IMAGEN
img = cv2.imread("defectosSIN_04.jpg")

# REALIZAMOS EL PRE-PROCESADO AJUSTANDO EL TAMAÑO Y EL COLOR
img = imutils.resize(img, height=min(800, img.shape[0]))
cv2.imshow("Original", img)
gris = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# REALIZAMOS UN SUAVIZADO MEDIANTE UN FILTRO DE GAUSS 5X5
gris = cv2.GaussianBlur(gris, (5,5), 0)
cv2.imshow("Gauss", gris)

# DETECTAMOS BORDES MEDIANTE CANNY
canny = cv2.Canny(gris, 75, 100)
cv2.imshow("Imagen con bordes de Canny (Gauss)", canny)

# REALIZAMOS UNA APERTURA 7X7
kernel = np.ones((7,7),np.uint8)
closing = cv2.morphologyEx(canny, cv2.MORPH_CLOSE, kernel)
cv2.imshow("Imagen cerrada", closing)

# OBTENEMOS LOS CONTORNOS DE LA IMAGEN Y DIMENSIONES DE LOS DEFECTOS
contours, _ = cv2.findContours(closing, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMPLE)
det = 0
for c in contours:
    area = cv2.contourArea(c)
    if area > 5 and area < 150:
        (x,y),radius = cv2.minEnclosingCircle(c)
        center = (int(x),int(y))
        radius = int(radius)
        cv2.drawContours(img, [c], 0, (0, 255, 0), 1, cv2.LINE_AA)
        det = det + 1
        areadef = "{}".format(area)
        cv2.putText(img, areadef, (int(x)+6, int(y)),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.4,(0,0,0),1)
        dim = "Defecto {}: (x, y)={} , R={}, area={}".format(det,center,radius,area)
        print(dim)

# MOSTRAMOS LAS DETECCIONES
text= "Posibles defectos: {}".format(det)
cv2.putText(img, text, (20, img.shape[0]-10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.6,(0,0,0),1)
cv2.imshow("Detecciones", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

ANEXO B: CÓDIGO DEL ALGORITMO CLASIFICADOR

CLASIFICADOR DE IMÁGENES

ARCHIVO *lenet.py*

```
# IMPORTAMOS LOS PAQUETES NECESARIOS
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras import backend as K

# DEFINIMOS LA RED NEURONAL
class LeNet:
    @staticmethod
    def build(width, height, depth, classes):
        # INICIALIZAMOS EL MODELO
        model = Sequential()
        inputShape = (height, width, depth)

        if K.image_data_format() == "channels_first":
            inputShape = (depth, height, width)

        # PRIMER CONJUNTO DE CAPAS CONV => RELU => POOL
        model.add(Conv2D(20, (5, 5), padding="same",
            input_shape=inputShape))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

        # SEGUNDO CONJUNTO DE CAPAS CONV => RELU => POOL
        model.add(Conv2D(50, (5, 5), padding="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

        # CONJUNTO DE CAPAS FC => RELU
        model.add(Flatten())
        model.add(Dense(500))
        model.add(Activation("relu"))

        # CLASIFICADOR SOFTMAX
        model.add(Dense(classes))
        model.add(Activation("softmax"))

        # DEVOLVEMOS EL MODELO CREADO
        return model
```

ARCHIVO *train_LeNet.py*

```
# IMPORTAMOS PAQUETES NECESARIOS
import matplotlib
matplotlib.use("Agg")
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.utils import to_categorical
from network.lenet import LeNet
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import random
import cv2
import os

os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"

# OBTENEMOS LOS PARAMETROS NECESARIOS DE LA CONSOLA DE COMANDOS
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
  help="ruta de entrada al dataset")
ap.add_argument("-m", "--model", required=True,
  help="ruta de salida al modelo")
ap.add_argument("-p", "--plot", type=str, default="plot_lenet.png",
  help="ruta de salida a la grafica perdida/precision")
args = vars(ap.parse_args())

# INICIALIZAMOS LOS VALORES PARA EPOCHS, EL LEARNING RATE Y EL BATCH SIZE
EPOCHS = 25
INIT_LR = 1e-3
BS = 32

# INICIALIZAMOS LOS DATOS Y LAS ETIQUETAS
print("[INFO] CARGANDO IMAGENES...")
data = []
labels = []

# LEEMOS LAS RUTAS DE LAS IMAGENES Y LAS MEZCLAMOS ALEATORIAMENTE
imagePaths = sorted(list(paths.list_images(args["dataset"])))
random.seed(42)
random.shuffle(imagePaths)

# BUCLE PARA LAS IMAGENES DE ENTRADA
for imagePath in imagePaths:
    # LEEMOS LA IMAGEN, PREPROCESAMOS Y LA ALMACENAMOS EN LOS DATOS
    image = cv2.imread(imagePath)
    image = cv2.resize(image, (64, 64))
    image = img_to_array(image)
    data.append(image)

    # EXTRAEMOS LA CLASE Y ACTUALIZAMOS LA LISTA DE ETIQUETAS
    label = imagePath.split(os.path.sep)[-2]
    label = 1 if label == "Defect" else 0
    labels.append(label)
```

```
# CAMBIAMOS LA ESCALA DE INTENSIDAD DE [0, 1]
data = np.array(data, dtype="float") / 255.0
labels = np.array(labels)

# GENERAMOS LOS CONJUNTOS DE ENTRENAMIENTO 75% Y TEST 25%
(trainX, testX, trainY, testY) = train_test_split(data,
    labels, test_size=0.25, random_state=42)

# CONVERTIMOS LAS ETIQUETAS A VECTORES
trainY = to_categorical(trainY, num_classes=2)
testY = to_categorical(testY, num_classes=2)

# INCREMENTAMOS LOS DATOS DE PARTIDA GENERANDO NUEVAS IMAGENES
aug = ImageDataGenerator(rotation_range=30, width_shift_range=0.1,
    height_shift_range=0.1, shear_range=0.2, zoom_range=0.2,
    horizontal_flip=True, fill_mode="nearest")

# INICIALIZAMOS EL MODELO
print("[INFO] COMPILANDO MODELO...")
model = LeNet.build(width=64, height=64, depth=3, classes=2)
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
    metrics=["accuracy"])

# ENTRENAMOS LA RED
print("[INFO] ENTRENANDO RED...")
H = model.fit(x=aug.flow(trainX, trainY, batch_size=BS),
    validation_data=(testX, testY), steps_per_epoch=len(trainX) // BS,
    epochs=EPOCHS, verbose=1)

# GUARDAMOS EL MODELO
print("[INFO] GENERANDO RED...")
model.save(args["model"], save_format="h5")

# GRAFICA PERDIDA/PRECISION
plt.style.use("ggplot")
plt.figure()
N = EPOCHS
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig(args["plot"])
```

ARCHIVO *test_network.py*

```
# IMPORTAMOS PAQUETES NECESARIOS
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
import numpy as np
import argparse
import imutils
import cv2
import os

os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"

# OBTENEMOS LOS PARAMETROS NECESARIOS DE LA CONSOLA DE COMANDOS
ap = argparse.ArgumentParser()
ap.add_argument("-m", "--model", required=True,
    help="ruta al modelo entrenado")
ap.add_argument("-i", "--image", required=True,
    help="ruta a la imagen de entrada")
args = vars(ap.parse_args())

# CARGAMOS LA IMAGEN
image = cv2.imread(args["image"])
orig = image.copy()

# PREPROCESAMOS LA IMAGEN PARA LA CLASIFICACION
image = cv2.resize(image, (64, 64))
image = image.astype("float") / 255.0
image = img_to_array(image)
image = np.expand_dims(image, axis=0)

# CARGAMOS LA RED NEURONAL
print("[INFO] CARGANDO RED...")
model = load_model(args["model"])

# CLASIFICAMOS LA IMAGEN
(notDefect, Defect) = model.predict(image)[0]

# ETIQUETAMOS LA IMAGEN
label = "Defecto" if Defect > notDefect else "No defecto"
proba = Defect if Defect > notDefect else notDefect
label = "{}: {:.2f}%".format(label, proba * 100)

# DIBUJAMOS LA ETIQUETA EN LA IMAGEN DE SALIDA
output = imutils.resize(orig, width=400)
cv2.putText(output, label, (10, 25),  cv2.FONT_HERSHEY_SIMPLEX,
    0.7, (0, 255, 0), 2)

# MOSTRAMOS LA IMAGEN DE SALIDA
cv2.imshow("Clasificacion", output)
cv2.waitKey(0)
```

ARCHIVO *test_network_auto*.py

```
# IMPORTAMOS PAQUETES NECESARIOS
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils import paths
import numpy as np
import argparse
import imutils
import cv2
import os

os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"

# OBTENEMOS LOS PARAMETROS NECESARIOS DE LA CONSOLA DE COMANDOS
ap = argparse.ArgumentParser()
ap.add_argument("-m", "--model", required=True,
    help="ruta al modelo entrenado")
ap.add_argument("-p", "--path", required=True,
    help="ruta del directorio")
args = vars(ap.parse_args())

# CARGAMOS LA RED NEURONAL
print("[INFO] CARGANDO RED...")
model = load_model(args["model"])

neg=pos=0
i=int(0)
for imagen in paths.list_images(args["path"]):

    # CARGAMOS LA IMAGEN
    image = cv2.imread(imagen)
    orig = image.copy()

    # PREPROCESAMOS LA IMAGEN PARA LA CLASIFICACION
    image = cv2.resize(image, (64, 64))
    image = image.astype("float") / 255.0
    image = img_to_array(image)
    image = np.expand_dims(image, axis=0)

    # CLASIFICAMOS LA IMAGEN
    (notDefect, Defect) = model.predict(image)[0]

    # ETIQUETAMOS LA IMAGEN
    label = "Defecto" if Defect > notDefect else "No defecto"
    proba = Defect if Defect > notDefect else notDefect
    label = "{}: {:.2f}%".format(label, proba * 100)

    # DIBUJAMOS LA ETIQUETA EN LA IMAGEN DE SALIDA
    output = imutils.resize(orig, width=400)
    cv2.putText(output, label, (10, 25),  cv2.FONT_HERSHEY_SIMPLEX,
        0.7, (0, 255, 0), 2)

    # CONTAMOS EL NUMERO DE CLASIFICACIONES
    if Defect > notDefect: pos=pos+1
    else: neg=neg+1
    total=pos+neg
    print("[INFO] Total:", total, "|| Positivas:",pos, "|| Negativas:",neg)

    # MOSTRAMOS LA IMAGEN DE SALIDA
    cv2.imshow("Clasificacion", output)
    cv2.waitKey(0)
    i+=1
```