Universidad Miguel Hernández de Elche

MÁSTER UNIVERSITARIO EN ROBÓTICA

“Reconocimiento de características de planchas de cartón ondulado mediante segmentación semántica”

Trabajo de Fin de Máster

2019/2020

Autor: Fouad Latrach
Tutor/es: Oscar Reinoso Garcia

Resumen

Uno de los procesos más importantes en el sector de envase y embalaje es la formación de cajas de cartón ondulado. Una vez las planchas de cartón han sido impresas y recortadas según el troquel adecuado, las máquinas plegadoras y formadoras intervienen en el proceso para acabar la caja dándole la forma esperada. Hasta el momento, las máquinas formadoras y plegadoras de cajas de cartón ondulado dependen en gran medida de la intervención del operario para la puesta en marcha y el ajuste de medidas y formatos. Dependiendo de cada modelo de caja, el operario tiene que instalar los mecanismos de formación adecuados manualmente y adaptar las medidas y los parámetros de la máquina antes de inicializar la producción. Estas operaciones de ajuste manual requieren un conocimiento específico y consumen un tiempo importante ya que se tienen que realizar varias veces en la misma jornada.

En este trabajo, trataremos un tema relacionado con la automatización de estas máquinas formadoras de cajas de cartón, para darles la autonomía a la hora de adaptarse a los requerimientos de producción. Se trata concretamente de desarrollar un sistema para el reconocimiento de las características de las planchas de cartón mediante la implementación de un algoritmo de inteligencia artificial que analiza la imagen tomada mediante un sistema de visión artificial, aplicándole una segmentación semántica para distinguir las diferentes características, pudiendo así, conocer el modelo de la caja en producción y localizar sus elementos para ajustar de manera automática y más optimizada los parámetros de la máquina.

Índice general

Índice de tablas

Índice de figuras

Índice de códigos

Capítulo 1

1. Introducción

1.1. Presentación

Probablemente no hay mejor manera de describir la inteligencia artificial (IA) que ateniéndose a las definiciones que los fundadores de esta disciplina establecieron. Como la descripción de Marvin Minsky, uno de los padres de las ciencias de la computación y cofundador del laboratorio de IA del MIT: «Es la ciencia de hacer que las máquinas hagan cosas que requerirían inteligencia si las hicieran las personas» [1].

La inteligencia artificial (IA), es la inteligencia llevada a cabo por máquinas. En ciencias de la computación, una máquina «inteligente» ideal es un agente flexible que percibe su entorno y lleva a cabo acciones que maximicen sus posibilidades de éxito en algún objetivo o tarea. El término inteligencia artificial se aplica cuando una máquina imita las funciones «cognitivas» que los humanos asocian con otras mentes humanas, como, por ejemplo: «percibir», «razonar», «aprender» y «resolver problemas» [2].

Por otra parte, la industria 4.0 está cambiando y evolucionando a nuevos conceptos, y sobre todo, se está adaptando a los nuevos entornos industriales y a las necesidades de los clientes. Este término fue utilizado por primera vez por el Gobierno alemán y describe una organización de los procesos de producción basada en la tecnología y en dispositivos que se comunican entre ellos de forma autónoma a lo largo de la cadena de valor [3]. Es un fenómeno que representa un cambio tan grande que también se denomina la cuarta revolución industrial.

En esta nueva etapa, los sensores, las máquinas, los componentes y los sistemas informáticos estarían conectados a lo largo de la cadena de valor, más allá de los límites de las empresas individuales. Estos sistemas conectados podrían interactuar entre ellos usando protocolos estándar basados en Internet

y analizar los datos para prever errores, configurarse ellos mismos y adaptarse a posibles cambios. Dicho de otro modo, las tecnologías digitales permiten la vinculación del mundo físico (dispositivos, materiales, productos, maquinaria e instalaciones) con el digital (sistemas). Esta conexión habilita que dispositivos y sistemas colaboren entre ellos y con otros sistemas para crear una industria inteligente, con producción descentralizada y que se adapta a los cambios en tiempo real. En este entorno, las barreras entre las personas y las máquinas se difuminan.

La inteligencia artificial supone un pilar muy importante dentro de la industria 4.0, ya que se trata de una de las tecnologías que posibilita la automatización de los procesos de fabricación, que es uno de los elementos clave de las factorías conectadas [4]. Las máquinas con IA representan un gran salto para los procesos industriales ya que ayudan a generar un mayor rendimiento en el área donde son usadas. Estas máquinas son capaces de recopilar datos procesarlos y aprender a tomar decisiones y adelantarse a posibles eventos antes de que estos pasen, por lo que hacen que el sistema tenga una gran capacidad de resolución de problemas. Además, son capaces de tomar decisiones que hasta ahora se adoptaban mediante monitorizaciones y correcciones manuales [3].

En la actualidad, una de las herramientas en que más se apoya la IA es la visión artificial. La aplicación de la visión artificial en la industria junto con la IA permite que los procesos de fabricación tengan mejores resultados de producción mediante la aplicación de controles de calidad y mayor agilidad en cada una de las fases.

Como otras industrias, el sector de envase y embalaje (E+E), se va adaptando constantemente a los requisitos del mercado, que ahora se caracteriza por la regulación, la escasez de profesionales y la transformación digital. Para convertir estos desafíos en oportunidades, las empresas innovadoras de este sector se tienen que enfrentar al reto de dejar de lado los procesos tradicionales y pasar a ciclos de producción más homogéneos, ligados directamente con la producción, y a la vez más flexibles, y confiar cada vez más en soluciones como la colaboración entre humanos y robots y el uso de la Inteligencia Artificial para optimizar la producción y el procesamiento intensivo de datos y poder adaptarse a una demanda cada vez más cambiante [5].

Las máquinas usadas en el proceso de creación de embalajes, a su vez,

se tienen que adaptar a la dinámica del mercado, ofreciendo la posibilidad de desarrollar más variedad de productos y optimizar la producción para mejorar la competitividad.

El gran desafío a que se tienen que afrontar estas máquinas es ganar la autonomía para ajustar sus parámetros a los valores específicos de cada producto, y optimizar la producción para gestionar mejor los cambios y minimizar los fallos. Para alcanzar este objetivo, las máquinas tienen que ser preparadas para tener la capacidad de conocer las características fundamentales de cada producto en tiempo real y como tienen que reaccionar anti cualquier situación que puede afectar a la producción.

Siguiendo la misma línea, este trabajo intenta abordar el problema dándole a las máquinas formadoras y plegadoras de cajas de cartón la posibilidad de ver y aprender por sí solas a reconocer cada modelo de plancha de cartón. Se emplea un sistema de visión artificial incorporado en la línea de producción para adquirir las imágenes de las planchas de cartón a formar. Estas imágenes son transferidas a un algoritmo de inteligencia artificial que analiza y destaca las características principales de cada modelo de caja empleando la segmentación semántica.

1.2. Motivación

En el sector de embalaje existen muchos modelos de cajas según los requerimientos del producto a que van destinados y las condiciones que le afectan en su paso por su ciclo de vida. Cada modelo de caja se diferencia del otro mediante un conjunto de características relacionadas con el diseño y propiedades del material. A veces varias cajas del mismo modelo, aunque en líneas generales comparten las mismas características principales, se diferencian mucho en el diseño especifico de cada elemento.

Para convertir las planchas de cartón en cajas, las máquinas formadoras y plegadoras de cajas de cartón disponen de sistemas que transforman la plancha pasando por diferentes etapas que se complementan para posicionar y fijar cada uno de los elementos en su posición adecuada en la caja. Debido a la gran variedad de diseños que hay en el mercado, las máquinas tienen que disponer de mecanismos intercambiables específicos para formar un modelo

determinado, y cualquier cambio ligero en el diseño puede suponer un cambio significativo en los sistemas utilizados y un reajuste global de la máquina.

Muchas de las empresas del sector de embalaje son fábricas que se dedican exclusivamente a la formación de cajas de cartón para otro cliente final y no para el uso propio en sus líneas de producción. Lo que les obliga a responder a frecuentes pedidos de diferentes modelos y diseños de cajas. Para satisfacer la demanda del mercado sus máquinas tienen que ser capaces de asumir todas las exigencias de los clientes en cuanto a la adaptación rápida a la variación de diseños y el cumplimiento de los plazos establecidos. En la práctica, para realizar los ajustes necesarios para adaptar una máquina a un determinado modelo de caja, hay que disponer de profesionales con un amplio conocimiento de todas las variedades de cajas y los ajustes que afectan a la transformación correcta de la plancha. Además, los tiempos de ajuste de medida y formato pueden ser muy elevados si las diferencias entre un modelo y otro son mayores.

Después de muchos años diseñando máquinas formadoras y plegadoras de cajas de cartón, y tratando muy de cerca las necesidades que afectan a este tipo de máquinas, surgió la idea de este trabajo para darles a las máquinas la capacidad de ser inteligentes y poder actuar de forma autónoma para solucionar los diferentes problemas del día a día.

Para una máquina, las cajas no son más que un conjunto de parámetros que se relacionan entre sí de una forma o de otra. Solo hace falta detectar estos parámetros y vincularlos con los parámetros de la máquina para determinar los ajustes necesarios en la adaptación al cambio. Para conseguir este objetivo primero hay que detectar estos elementos de la plancha. Sin un sistema de visón adecuado, la mayoría de las características no se pueden detectar, como pueden ser los pliegues internos y el sentido de los ondulados. Para solucionar este problema, se ha utilizado la técnica de iluminación de campo oscuro *dark-field* que destaca mejor el diseño basándose en las sombras generadas en los cortes y pliegues de la plancha por causa de la iluminación lateral. Luego hay que analizar las imágenes de manera precisa para destacar sus características más importantes. Sin embargo, sin la existencia de un sistema inteligente, la máquina no puede seguir los cambios frecuentes que sufre cada modelo y aprender a reconocerlos en un futuro. Por eso se ha optado por entrenar un

algoritmo de inteligencia artificial para poder realizar la tarea de reconocimiento de las características de la plancha. Por último, para delimitar mejor los elementos detectados de cada diseño, se ha utilizado un algoritmo de segmentación semántica.

1.3. Objetivos

Aparte de lo que se ha comentado en los apartados anteriores, los objetivos principales de este trabajo se pueden dividir en tres puntos:

- Estudiar la mejor técnica de iluminación capaz de destacar las características de planchas de cartón ondulado de los modelos *Columna* y *Plaform*. Como resultado a este estudio, utilizaremos la técnica de Campo oscuro para resaltar la sombra generada en los pliegues para trazar el contorno de cada característica.
- Construir un sistema de visión artificial basado en la técnica de iluminación seleccionada para capturar las imágenes y crear un algoritmo para preprocesarlas y adaptarlas a los requerimientos de las siguientes etapas.
- Construir una base de datos con las imágenes preprocesadas de los dos modelos de planchas. Esta base de datos se constituye de conjuntos de imágenes para el entrenamiento, validación y test.
- Crear un algoritmo de inteligencia artificial y entrenarlo con la base de datos anterior para reconocer las características de las planchas de forma optimizada utilizando la técnica de segmentación semántica.

En los próximos capítulos se van a detallar más los pasos seguidos para realizar estos objetivos.

1.4. Estructura del documento

El presente trabajo de fin de máster se encuentra dividido en capítulos principales más un resumen en el cual se sintetizan los aspectos principales y más significativos del trabajo.

El documento seguirá por tanto la siguiente estructura:

- Capítulo 1: se incluye una breve introducción del trabajo en la cual se podrá llevar a cabo una primera toma de contacto con el contexto y la

temática tratada en él, como la motivación y los objetivos para justificar la realización del mismo.

- Capítulo 2: este capítulo sirve como base teórica para la realización del trabajo, introduciéndose los conceptos fundamentales que se van a manejar en el trabajo mediante la realización de una descripción del estado del arte.
- Capítulo 3: en este capítulo se describe el sistema de visión artificial construido y los ajustes de técnicas de iluminación para la adquisición de las imágenes.
- Capítulo 4: se realiza una descripción de los pasos seguidos y las herramientas usadas para el preprocesamiento de las imágenes adquiridas.
- Capítulo 5: en este capítulo se explica el proceso de segmentación incluido el etiquetado de las imágenes, la creación de la red neuronal VGG16 y su entrenamiento para la segmentación semántica.
- Capítulo 6: En este capítulo se analizan los resultados obtenidos y se evalúa el rendimiento de la red neuronal creada presentando las posibles modificaciones para optimizar el sistema.
- Capítulo 7: en este último capítulo se exponen las conclusiones extraídas del trabajo y las futuras líneas de trabajo a desarrollar.

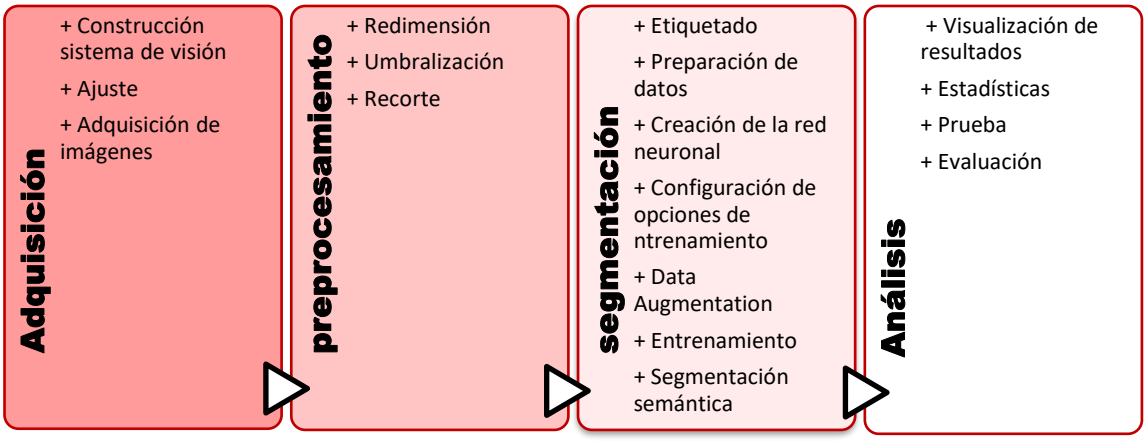


Figura 1 Flujo de trabajo

Capítulo 2

2. Estado de arte

2.1. Visión artificial

2.1.1. Introducción

La visión artificial consiste básicamente en la deducción automática de la estructura y propiedades de un mundo tridimensional, posiblemente dinámico, a partir de una o varias imágenes bidimensionales de ese mundo. En esta área de conocimiento se aúnan conceptos de la física del color, óptica, electrónica, geometría, algorítmica, sistemas de computación, etc [6].

El propósito de este capítulo es introducir los conceptos básicos sobre la tecnologías que compone un sistema de visión. Empezando por los diferentes sistemas de visión, describiendo brevemente los dispositivos que permiten controlar la transmisión de la luz, además de las diferentes técnicas de iluminación.

2.1.2. Sistemas de visión artificial

Los sistemas de visión artificial son fundamentales a la hora de resolver los problemas que acontecen en el control de procesos de fabricación que se llevan a cabo en continuo. Además, constituye una herramienta de gran ayuda en la producción de la mayoría de los procesos industriales donde se ha implementado la automatización de dichos procesos.

Según la Automated Imaging Association (AIA), la visión artificial abarca todas las aplicaciones industriales y no industriales en las que una combinación de hardware y software brinda un guiado operativo a los dispositivos en la ejecución de sus funciones de acuerdo con la captación y procesamiento de imágenes.

Aunque la visión artificial aplicada a la industria utiliza los mismos algoritmos y enfoques que las aplicaciones académicas/educativas y gubernamentales/militares de visión artificial, las limitaciones son diferentes.

Los sistemas de visión industrial exigen una mayor robustez, fiabilidad y estabilidad en comparación con un sistema de visión académico/educativo y, normalmente, cuestan mucho menos que los que se utilizan en aplicaciones gubernamentales/militares. Por tanto, la visión artificial industrial implica bajos costes, precisión aceptable, resistencia elevada, alta fiabilidad y una gran estabilidad mecánica y de temperatura.

Los sistemas de visión artificial cuentan con sensores digitales protegidos en el interior de cámaras industriales con ópticas especializadas para adquirir imágenes, de forma que el hardware y software informático pueden procesar, analizar y medir diversas características a la hora de tomar decisiones [7].

2.1.2.1. Ventajas

Mientras que la visión humana es mejor para la interpretación cualitativa de una escena desestructurada compleja, la visión artificial destaca en la medida cuantitativa de una escena estructurada debido a su velocidad, precisión y repetibilidad. Por ejemplo, en una línea de producción, un sistema de visión artificial puede inspeccionar cientos, o incluso miles, de piezas por minuto. Un sistema de visión artificial gira en torno a la resolución correcta de la cámara y a la óptica que le permiten inspeccionar fácilmente detalles de un objeto demasiado pequeños para que el ojo humano pueda llegarlos a ver.

Al eliminar el contacto físico entre el sistema de prueba y las piezas que van a verificarse, la visión artificial evita daños en las piezas y elimina el tiempo y los costes de mantenimiento asociados al desgaste de los componentes mecánicos. La visión artificial aporta beneficios operativos y de seguridad adicionales al reducir la participación humana en el proceso de fabricación. Además, evita la contaminación humana de salas limpias y protege a las personas frente a entornos peligrosos [8].

Objetivo estratégico	Aplicaciones de visión artificial
Calidad superior	Inspección, medición, calibración y verificación de montaje
Mayor productividad	Las tareas repetitivas que se hacían antes manualmente se realizan ahora con el sistema de visión artificial
Flexibilidad de producción	Medición y calibración / Guiado de robots / Verificación previa a la operación
Menos tiempo de inactividad de las máquinas y reducción del tiempo de configuración	Cambios programados de antemano
Información más completa y control de procesos más estricto	Las tareas manuales pueden ahora ofrecer retroalimentación de datos por ordenador.
Reducción de gastos de bienes de capital	La adición de visión a una máquina mejora su rendimiento y evita la obsolescencia
Menos costes de producción	Un sistema de visión vs. muchas personas/Detección de taras en una etapa temprana del proceso
Reducción de la tasa de deshechos	Inspección, medición y calibración
Control de inventarios	Reconocimiento óptico de caracteres e identificación
Espacio reducido	Sistema de visión vs. operador

Tabla 1 Objetivos estratégicos de la visión artificial [8]

2.1.2.2. Aplicaciones

Normalmente, la tecnología de correspondencia de patrones representa el primer paso en cualquier aplicación de visión artificial, ya sea la verificación del montaje más simple o la selección/extracción de piezas amontonadas aleatoriamente en un contenedor mediante un complejo sistema robótico 3D, para buscar el objeto o característica de interés dentro del campo de visión de la cámara. La localización del objeto de interés suele determinar el éxito o el

fracaso de la aplicación. Si las herramientas de software de correspondencia de patrones no pueden localizar con precisión la pieza dentro de la imagen, no podrán guiar, identificar, inspeccionar, contar ni medir la pieza. Aunque la búsqueda de una pieza pueda parecer sencillo, las diferencias de apariencia en entornos de producción reales pueden hacer que el próximo paso sea excesivamente difícil (Figura 1). A pesar de que los sistemas de visión están entrenados para reconocer piezas basándose en patrones, incluso los procesos controlados más ajustados permiten alguna variabilidad en la apariencia de una pieza (Figura 2) [8].

Figura 2 Los cambios de apariencia debidos a la iluminación.

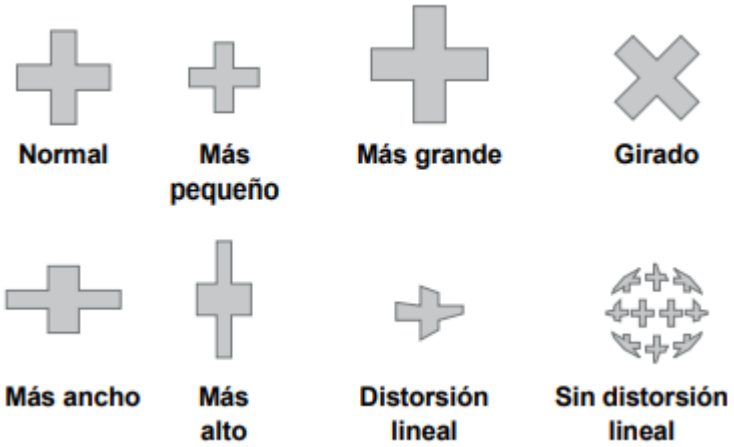


Figura 3 La distorsión de pose o presentación de la pieza [8]

Para lograr unos resultados precisos, fiables y repetibles, las herramientas de localización de piezas de un sistema de visión deben incluir suficiente inteligencia para comparar de forma rápida y precisa los patrones de formación con los objetos reales (correspondencia de patrones) que bajan por la línea de producción. La localización de piezas es el primer paso crítico en las cuatro categorías principales

de aplicaciones de visión artificial: guiado, identificación, medición e inspección.

- **Guiado.**

El guiado se puede realizar por varias razones. En primer lugar, los sistemas de visión artificial pueden localizar la posición y orientación de una pieza, compararla con una tolerancia especificada, y garantizar que está en el ángulo correcto para verificar el montaje apropiado. A continuación, el guiado se puede usar para notificar la localización y orientación de una pieza en el espacio 2D o 3D a un robot o unidad de control de una máquina, para que el robot pueda localizar la pieza o la máquina pueda alinearla. El guiado por visión artificial permite obtener una velocidad y una precisión muy superiores que el posicionamiento manual en tareas como la disposición de piezas dentro o fuera de palés, el embalaje de piezas al salir de una cinta transportadora, la búsqueda y alineación de piezas para el montaje con otros componentes, la colocación de piezas en un estante, o la retirada de piezas de los almacenes.

El guiado también puede utilizarse para la alineación de otras herramientas de visión artificial. Se trata de una característica muy potente de la visión artificial, puesto que la presentación de las piezas a la cámara se puede producir en orientaciones desconocidas durante la producción. Al localizar la pieza y alinear las demás herramientas de visión artificial respecto a ella, la visión artificial permite el posicionamiento/sujeción automática de las herramientas. Esto implica la localización de las características claves de una pieza para permitir posicionar con precisión un calibre, nivel, perfil u otras herramientas de software de visión para que puedan interactuar correctamente con la pieza. Este enfoque permite a los fabricantes desarrollar diversos productos en la misma línea de producción y reduce la necesidad de herramientas de hardware costosas para mantener la posición de la pieza durante la inspección [8].

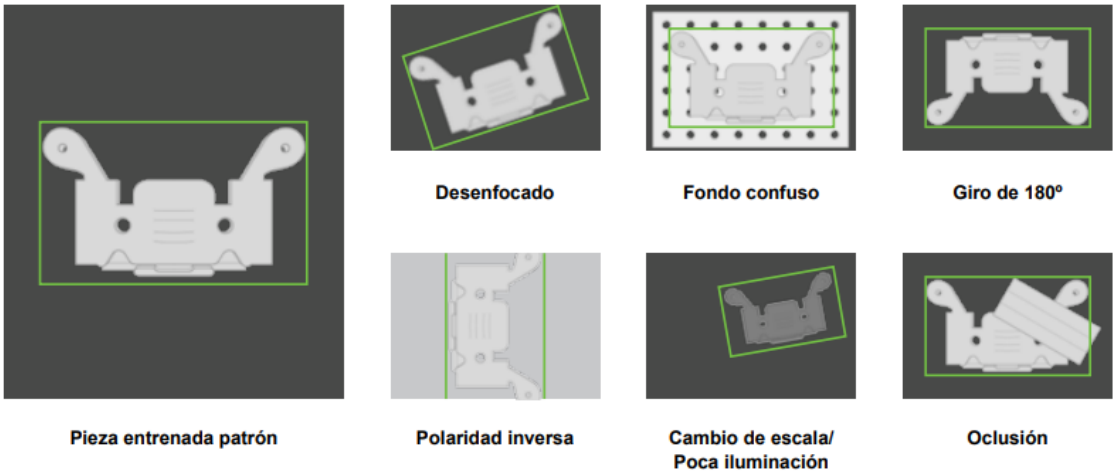


Figura 4 La correspondencia de patrones [8]

- **Identificación**

Un sistema de visión artificial para identificación y reconocimiento de piezas lee códigos de barras (1-D), códigos de matrices de datos (2-D), marcajes directos en piezas (DPM) y caracteres impresos en piezas, etiquetas y paquetes. Un sistema de reconocimiento óptico de caracteres (OCR) lee caracteres alfanuméricos sin conocimiento previo, mientras que un sistema de comprobación óptica de caracteres (OCV) confirma la presencia de una cadena de caracteres. Además, los sistemas de visión artificial pueden identificar piezas mediante la localización de un patrón único o identificar elementos según su color, forma o tamaño.

Las aplicaciones DPM marcan un código o una cadena de caracteres directamente en la pieza. Fabricantes de todas las industrias suelen usar esta técnica para detectar errores, habilitar estrategias eficaces de contención, supervisar el control de procesos y las métricas de control de calidad y cuantificar áreas problemáticas en una planta como cuellos de botella. La trazabilidad mediante el marcaje directo en las piezas mejora el seguimiento de los activos y la verificación de autenticidad de las piezas. Además, proporciona datos a nivel unitario para impulsar un soporte técnico y un servicio de reparación en garantía superior al documentar la genealogía de las piezas en un submódulo que prepara el producto terminado [8].

Figura 5 Técnicas de identificación [8]

- **Medición**

Un sistema de visión artificial para medición calcula las distancias entre dos o más puntos o ubicaciones geométricas de un objeto y determina si esas medidas cumplen las especificaciones. Si no, el sistema de visión envía una señal de fallo a la unidad de control de la máquina, activando un mecanismo de rechazo que expulsa el objeto de la línea.

En la práctica, una cámara fija capta imágenes de las piezas cuando pasan por su campo de visión y el sistema utiliza software para calcular las distancias entre distintos puntos de la imagen. Como muchos sistemas de visión artificial pueden medir características de un objeto con una precisión de hasta 0,0254 milímetros, abordan una serie de aplicaciones que tradicionalmente se han gestionado a través de la medición por contacto [8].

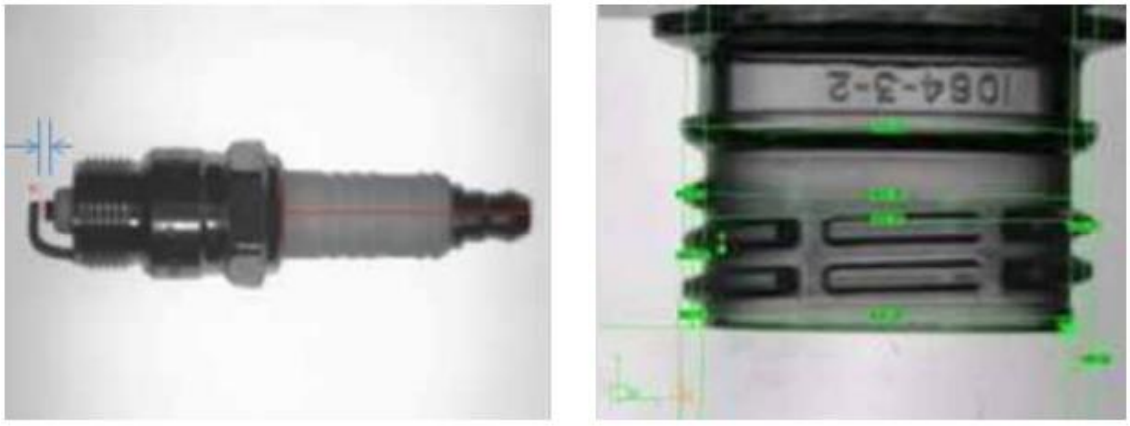


Figura 6 Aplicaciones de medición [8]

- **Inspección**

Un sistema de visión artificial para inspección detecta defectos, contaminantes, imperfecciones funcionales y otras irregularidades en productos manufacturados.

Como ejemplos cabe citar la inspección de tabletas de medicamentos en busca de taras, de pantallas para verificar los iconos o confirmar la presencia de píxeles, o de pantallas táctiles para medir el nivel de contraste de fondo. La visión artificial también puede inspeccionar la integridad de los productos, como garantizar la coincidencia entre producto y paquete en las industrias de alimentación y farmacéutica, y la comprobación de precintos de seguridad, tapones y anillas en las botellas.

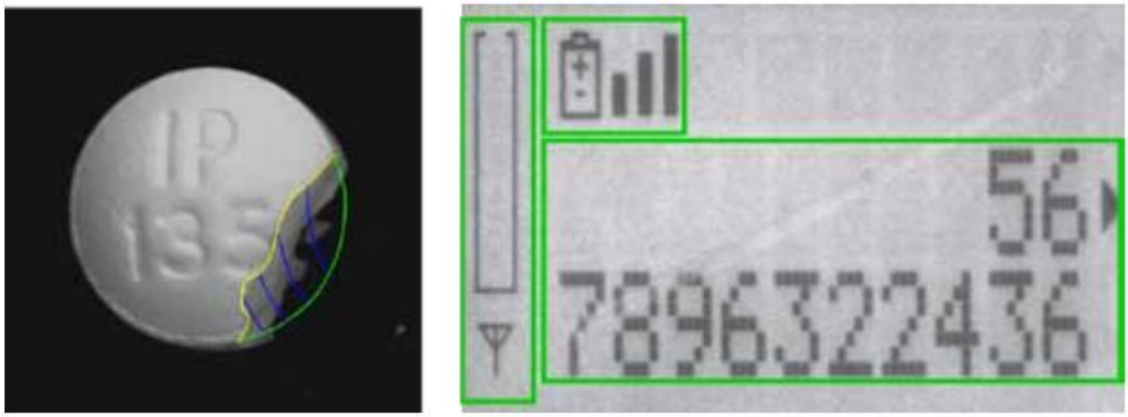


Figura 7 detección de defectos y irregularidades [8]

2.1.2.3. Componentes

Los componentes principales de un sistema de visión artificial (Figura 1) son la iluminación, la lente, el sensor de imagen, el procesamiento de visión y las comunicaciones. La iluminación ilumina la pieza que va a inspeccionarse permitiendo que sus características destaquen y puedan ser vistas claramente por la cámara. La lente capta la imagen y la presenta al sensor en forma de luz. El sensor de la cámara de visión artificial convierte esta luz en una imagen digital que se envía al procesador para su análisis.

El procesamiento de visión consiste en unos algoritmos que revisan la imagen y extraen de ella la información precisa, realizan la inspección necesaria y toman una decisión. Por último, la comunicación se lleva a cabo normalmente a través de una señal de E/S discreta o los datos se envían a través de una conexión serie hacia un dispositivo que registra la información o la utiliza.

La mayoría de los componentes de hardware de los sistemas de visión artificial, como módulos de iluminación, sensores y procesadores, son productos comercialmente disponibles (COTS). Los sistemas de visión artificial se pueden

montar a partir de productos COTS, o adquirirse como un sistema integrado con todos los componentes en un único dispositivo [8].

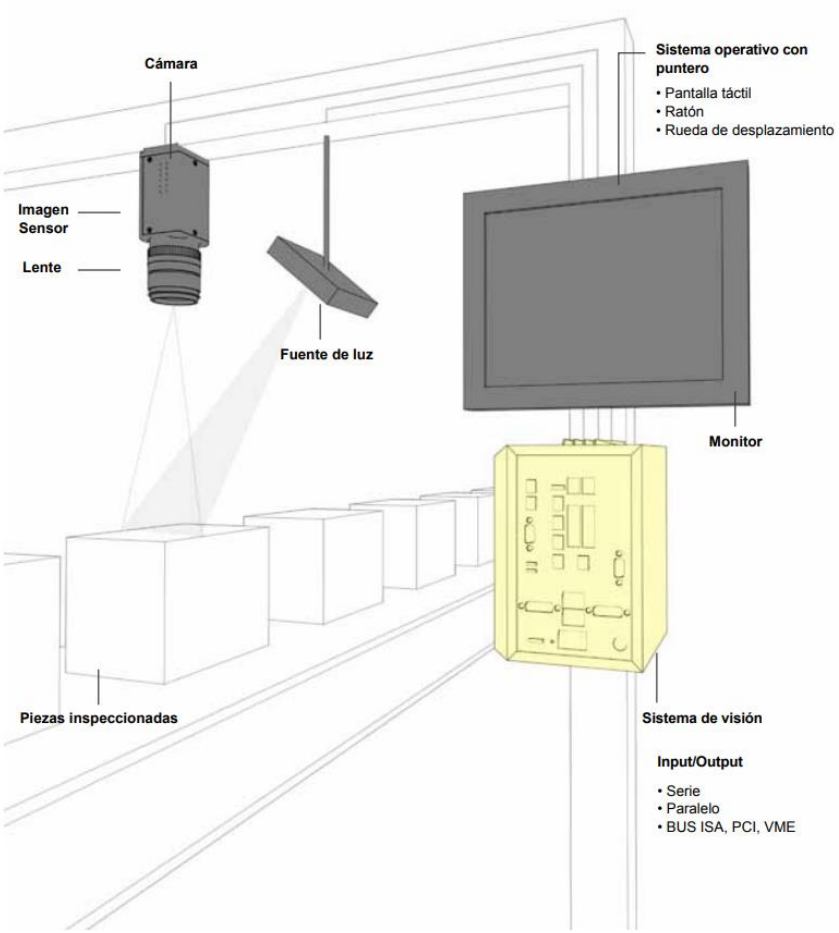


Figura 8 Principales componentes de un sistema de visión artificial industrial [8]

En las páginas siguientes se indican los distintos componentes claves de un sistema de visión artificial: iluminación, lentes, sensor de visión, procesamiento de imágenes, procesamiento de visión y comunicaciones.

2.1.2.3.1. Iluminación

En las aplicaciones de visión la importancia de la iluminación es en muchas ocasiones subestimada.

La iluminación se puede considerar la parte más crítica dentro de un sistema de visión. Las cámaras, de momento, son mucho menos sensibles y versátiles que la visión humana y las condiciones de iluminación deben optimizarse al máximo, para que una cámara pueda capturar una imagen que

el ojo humano podría distinguir sin necesidad de una iluminación tan especializada. Esto se hace mucho más evidente cuando el objeto a iluminar presenta formas complejas o superficies muy reflectantes.

Las cámaras de visión artificial capturan la luz reflejada de los objetos. El propósito de la iluminación utilizada en las aplicaciones de visión es controlar la forma en que la cámara va a ver el objeto. La luz se refleja de forma distinta si se ilumina una bola de acero, que si se ilumina una hoja de papel blanco y el sistema de iluminación por tanto debe ajustarse al objeto a iluminar.

Si se utiliza una iluminación adecuada, la aplicación se resolverá más fácilmente, mientras que si la misma aplicación recibe una iluminación incorrecta puede que sea imposible de resolver. Si para solucionar una aplicación es necesario utilizar muchos filtros de software, significa que la iluminación que se está aplicando no es la adecuada. Una iluminación correcta permitirá emplear menos filtros en la imagen y por tanto aumentar la velocidad de proceso en esa aplicación.

La primera pregunta que hay que hacerse cuando se debe resolver una nueva aplicación es: ¿Cuál es la mejor iluminación para nuestra aplicación? La tecnología de la iluminación recoge un número de consideraciones a tener en cuenta para determinar la mejor iluminación para una aplicación [9].

- **Fuentes de iluminación y contenido espectral**

Las fuentes de iluminación que más se usan en visión artificial son fluorescentes, halógenos, LED, haluros metálicos (mercurio) y xenón, particularmente para estaciones de inspección de pequeña a mediana escala. El haluro metálico, el xenón y el sodio a alta presión se usan más comúnmente en aplicaciones a gran escala o en áreas que requieren una fuente muy brillante. El haluro metálico, también conocido como mercurio, a menudo se usa en microscopía porque tiene muchos picos de longitud de onda discretos, lo que complementa el uso de filtros para estudios de fluorescencia. Una fuente de xenón es útil para aplicaciones que requieren una luz estroboscópica muy brillante. La Figura 9 muestra las ventajas y desventajas de los tipos de iluminación fluorescente, halógena y LED y los criterios de selección relevantes, tal como se aplican a la visión artificial. Por ejemplo, mientras que la iluminación LED tiene una mayor vida útil, la iluminación halógena puede ser

la opción para una inspección en particular porque ofrece una mayor intensidad.

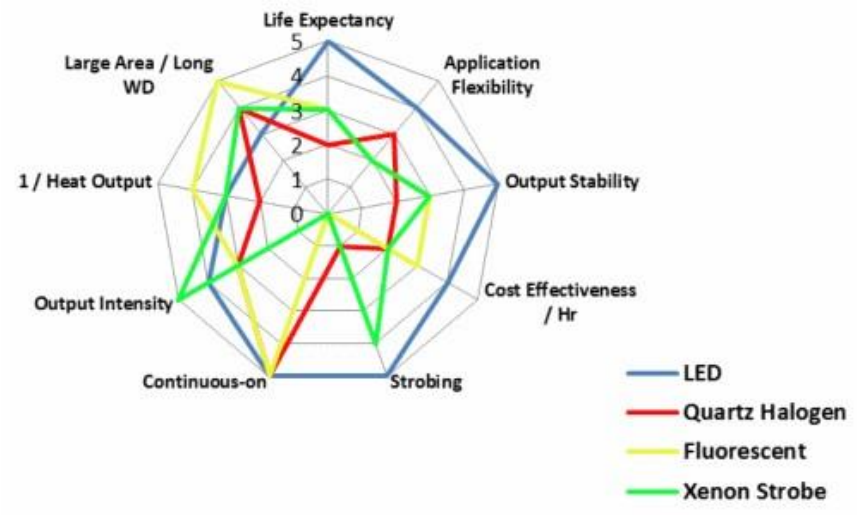


Figura 9 Comparación de fuentes de iluminación comunes [9]

Históricamente, las fuentes de iluminación fluorescentes y halógenas se han utilizado con mayor frecuencia. En los últimos años, la tecnología LED ha mejorado en estabilidad, intensidad y rentabilidad; sin embargo, todavía no es tan rentable para la iluminación de grandes áreas, particularmente en comparación con las fuentes fluorescentes. Sin embargo, si la flexibilidad de la aplicación y la estabilidad de salida son parámetros importantes, entonces la iluminación LED podría ser más apropiada. Dependiendo de los requisitos de iluminación exactos, a menudo se puede usar más de un tipo de fuente para una implementación específica, y la mayoría de los expertos en visión están de acuerdo en que un tipo de fuente no puede resolver adecuadamente todos los problemas de iluminación.

No hay que considerar solo el brillo de una fuente sino también su contenido espectral (Figura 10). Por ejemplo, las aplicaciones de microscopía a menudo usan una fuente de halógeno de espectro completo, xenón o mercurio, particularmente cuando se toman imágenes en color. Sin embargo, una fuente LED monocromática también es útil para una cámara CCD en blanco y negro, y ahora también para aplicaciones en color, con la llegada de "todo color - RGB" y cabezales de luz LED blanca.

En aquellas aplicaciones que requieren alta intensidad de luz, como inspecciones de alta velocidad, puede ser útil hacer coincidir la salida espectral de la fuente con la sensibilidad espectral de la cámara de visión particular (Figura

10). Por ejemplo, las cámaras basadas en sensores CMOS son más sensibles al IR que sus contrapartes (CCD), lo que confiere una ventaja de sensibilidad significativa en las configuraciones de inspección con poca luz cuando se utilizan LED IR o fuentes de tungsteno ricas en IR [9].

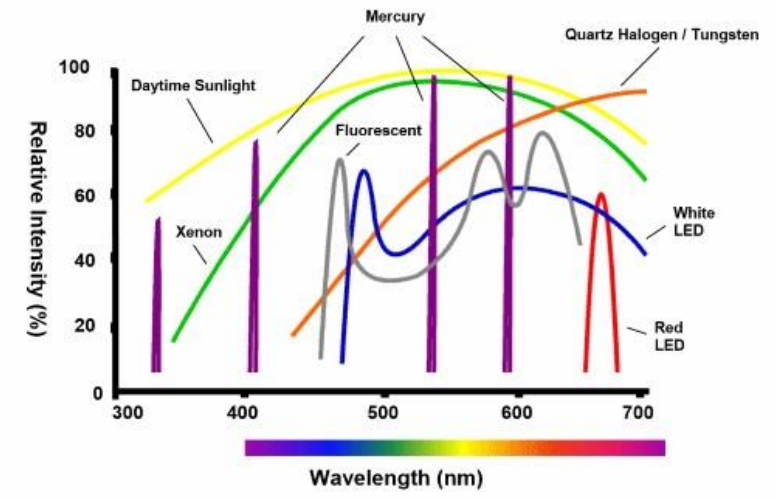


Figura 10 intensidad relativa vs contenido espectral [9]

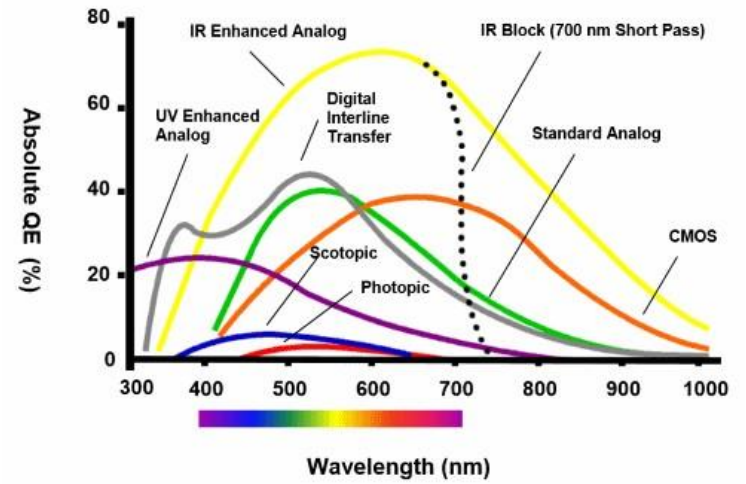


Figura 11 Eficiencia cuántica absoluta del sensor de la cámara vs longitud de onda [9]

Además, las Figuras anteriores ilustran varios otros puntos relevantes a considerar al seleccionar una cámara y fuente de iluminación:

- Intente hacer coincidir la sensibilidad máxima del sensor con la longitud de onda máxima de la fuente de iluminación para aprovechar al máximo la salida.
- Las fuentes de longitud de onda estrecha, como los LED monocromos o el mercurio son beneficiosas para pasar longitudes de onda estratégicas

cuando se combinan con filtros de paso estrecho. Por ejemplo, el filtro de paso de banda rojo de 660 nm, cuando se combina con la luz LED roja, es efectivo para bloquear la luz ambiental en una planta iluminada con fuentes fluorescentes o de mercurio.

- La luz del sol tiene la intensidad bruta y el contenido espectral de banda ancha para cuestionar los resultados de la inspección de la visión: utilice una carcasa opaca.
- Aunque la mente es buena para interpretar lo que ven los ojos, el sistema visual humano es lamentablemente inadecuado en términos de sensibilidad máxima y rango dinámico espectral: hay que dejar que los ojos vean la imagen tal como la adquirió la cámara de visión [7]

Tipo de iluminación	Ventajas	Inconvenientes
Incandescente/Halógena	• Bajo coste y fáciles de utilizar. • Permiten ajustar la intensidad de luz. • Oscila 50 veces por segundo.	• Desprenden una gran cantidad de calor • Su espectro se centra en el rojo siendo deficiente para azules verdes o amarillos
Fluorescentes	• Se calienta menos que el incandescente • Su espectro se centra en los colores del ojo humano • La duración está estimada en torno a 10.000 horas	• La longitud de onda de la luz cambia con el uso • Para que sean válidos en aplicaciones industriales tienen que trabajar a una frecuencia del orden de 25 KHz
Led	• Gran durabilidad (100.000 horas) • Posibilidad de encender y apagar solamente en el tiempo de captura de la imagen • Fácil elección de la longitud de onda de la fuente de luz dentro del espectro visible e infrarrojo • Las fuentes de luz se pueden construir en multitud de formas	• Precio
Láser	• Se utilizan para generar luz estructurada con forma diversas tales como líneas, líneas paralelas, líneas cruzadas, retículas, puntos y matriz de puntos. Para generar las formas se utilizan ópticas específicas. • Están disponibles en multitud de longitud de ondas desde el visible al infrarrojo cercano. • Dado que el ojo humano es muy sensible al verde, un diodo láser en esta longitud de onda genera un mejor contraste en los bordes especialmente sobre superficies rojas.	• Precio
Fibra óptica	• Se utiliza para llevar la luz a cualquier punto distante de la fuente de luz. • Permite iluminar pequeñas áreas. • Proporciona luz fría, es decir, no se calienta.	• Precio. • Sólo sirve para iluminar pequeñas áreas.

Tabla 2 ejemplos de distintos sistemas de iluminación

- **Los pilares de la iluminación**

Los 4 pilares de la iluminación en visión artificial son:

- Geometría: la relación espacial tridimensional entre muestra, luz y cámara.
- Estructura o patrón: la forma de la luz proyectada sobre la muestra.
- Longitud de onda o color: cómo la luz se refleja o absorbe de manera diferencial por la muestra y su fondo.
- Filtros: bloqueo y paso diferencial de longitudes de onda y / o

direcciones de luz.

Comprender cómo manipular y mejorar el contraste de la muestra utilizando las cuatro piedras angulares es crucial para cumplir con los tres criterios de aceptación para evaluar la calidad y la solidez de la iluminación. Efectuar cambios de contraste a través de la geometría implica mover la muestra, la luz y / o las posiciones de la cámara hasta que encuentre una configuración adecuada. Por ejemplo, una luz de anillo coaxial (una montada alrededor de la cámara) puede generar un punto de reflejo en una superficie de código de barras semirreflectante, pero simplemente moviendo la luz fuera del eje, el reflejo también se aleja del campo de visión de la cámara. Los cambios de contraste a través de la estructura o la forma de la luz proyectada en la muestra generalmente son específicos de la fuente de luz o de la técnica de iluminación. Los cambios de contraste a través de la iluminación de color están relacionados con la absorbancia de color diferencial versus la reflectancia [10].

- **Consideraciones para una solución de iluminación óptima**

Hay que considere todas estas evaluaciones junto con lo comentado anteriormente, para desarrollar una solución de iluminación adecuada:

- **Entorno:**

Comprender completamente los requisitos físicos y las limitaciones del área de inspección inmediata, en un espacio 3D, es crítico. En particular, dependiendo de los requisitos de inspección específicos, el uso de máquinas robóticas de pick-and-place o estructuras de soporte preexistentes, pero necesarias, puede limitar severamente la elección de soluciones de iluminación efectivas al añadir nuevas restricciones, no solo en el tipo de iluminación, pero también su geometría, distancia de trabajo, intensidad y patrón. Por ejemplo, puede determinar que se requiere una fuente de luz difusa pero que no se puede aplicar debido al acceso limitado a la zona de inspección.

- **Luz ambiental:**

La presencia de luz ambiental puede tener un tremendo impacto en la calidad y la consistencia de las inspecciones, particularmente cuando se usa una fuente multiespectral como la luz blanca. Los contribuyentes ambientales más comunes son las luces techo de la fábrica y la luz solar, pero ocasionalmente la iluminación

de tareas específicas de visión de otras estaciones de inspección o incluso otras estaciones en la misma celda de trabajo puede tener un impacto.

Existen tres métodos activos para lidiar con la luz ambiental: estroboscopio de alta potencia con pulsos de corta duración, recintos físicos y filtros de paso. El método que se aplica es una función de muchos factores, la mayoría de los cuales se analizan con cierto detalle en secciones posteriores. El estroboscopio de alta potencia simplemente elimina la contribución ambiental, pero tiene desventajas en ergonomía, costo y esfuerzo de implementación, además de que no todas las fuentes, como fluorescentes, pueden ser estroboscopias. Si no se puede utilizar el estroboscopio, y si la aplicación requiere el uso de una cámara en color, es necesaria una luz blanca multiespectral para una reproducción y equilibrio precisos del color. En esta circunstancia, un filtro de paso de longitud de onda estrecha es ineficaz, ya que bloqueará una parte importante de la contribución de la luz blanca y, por lo tanto, un recinto cerrado es la mejor opción [10].

- **Interacción objeto – luz:**

La forma en que la superficie de un objeto interactúa con la luz ambiental y la iluminación específica de la tarea está relacionada con muchos factores, incluida la forma superficial, la geometría y la reflectividad, así como su composición, topografía y color. Una combinación de estos factores determina la cantidad de luz, y de qué manera, se refleja en la cámara y, posteriormente, la que pasará para adquisición, procesamiento y medición.

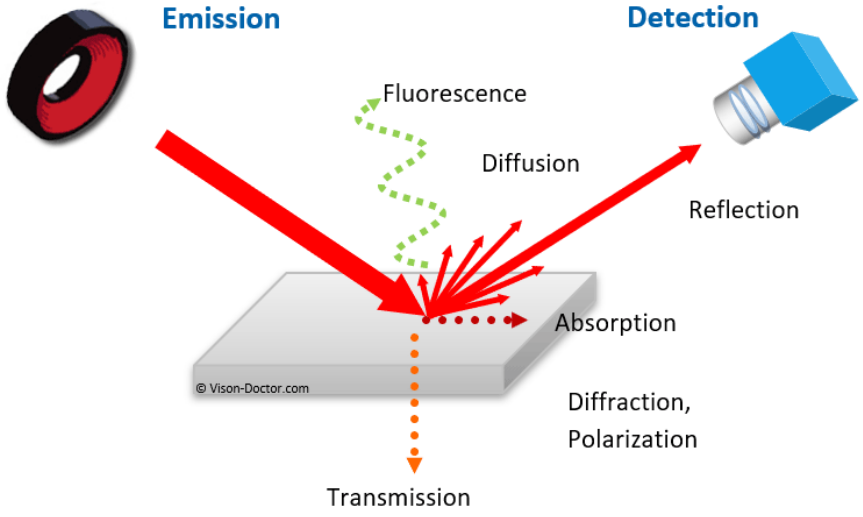


Figura 12 Interacción entre la radiación electromagnética y el objeto [10]

Por ejemplo, una superficie especular curvada, como el fondo de una lata de refresco (Figura 13), refleja una fuente de luz direccional de manera diferente a una superficie plana y difusa, como el papel. De manera similar, una superficie topográfica, como una PCB, se refleja de manera diferente a una superficie plana pero finamente texturizada (Figura 14), dependiendo del tipo de luz y la geometría [9].

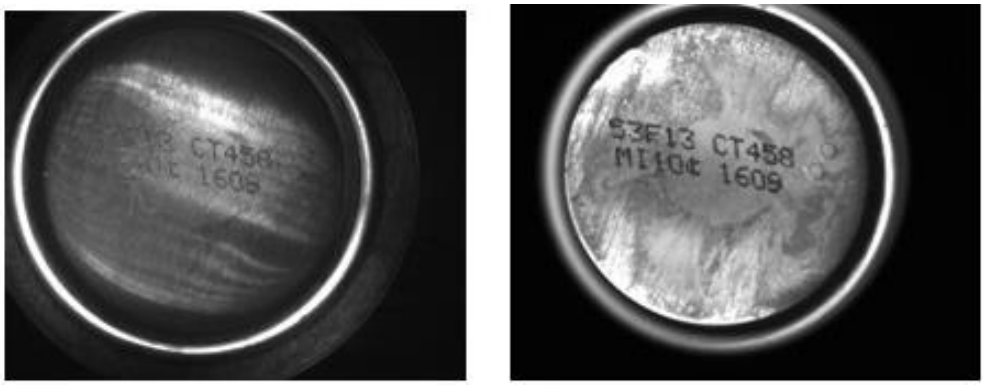


Figura 13 luz direccional vs difusa [9]

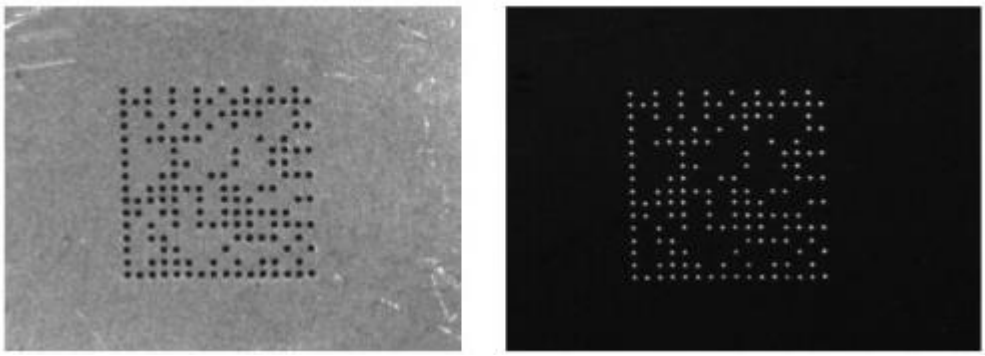


Figura 14 luz direccional vs de campo oscuro [9]

- **Análisis de color**

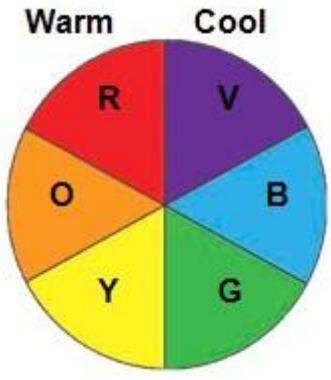


Figura 15 Rueda de colores [9]

Los materiales reflejan y / o absorben varias longitudes de onda de luz de manera diferente, un efecto válido tanto para el espacio de imágenes en blanco y negro como en color. Los colores similares se reflejan y las superficies se iluminan; por el contrario, los colores opuestos absorben y las superficies se oscurecen. Usando una simple rueda de colores cálidos versus fríos (Figura 15), puede generar un contraste diferencial entre una parte y su fondo (Figura 16), e incluso diferenciar partes de color, dada una paleta de colores limitada y conocida, con un negro y cámara blanca (Figura 17) [9].

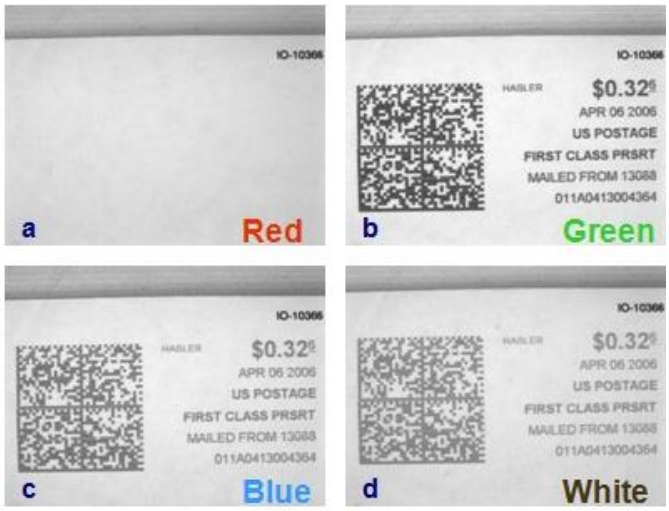


Figura 16 Un sello fotografiado bajo luz de varios colores [9]

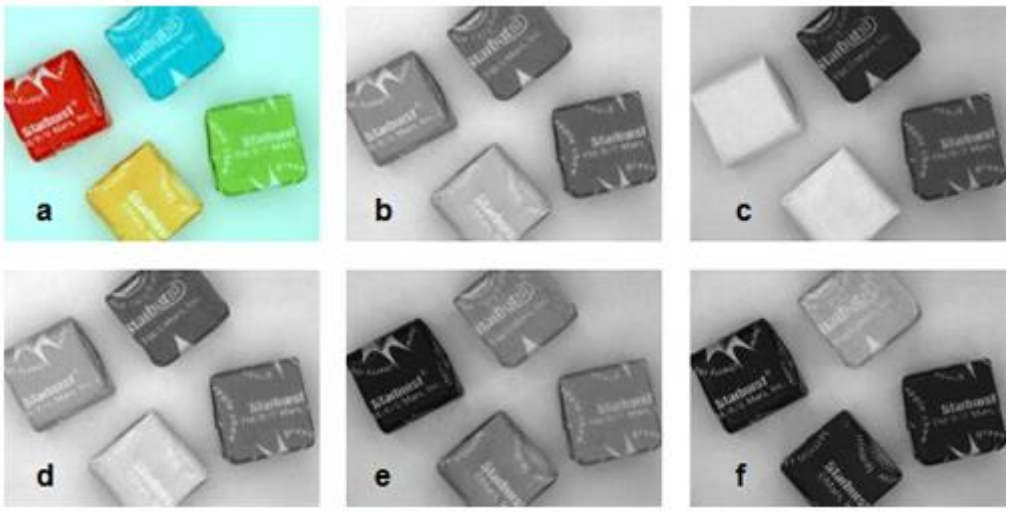


Figura 17 piezas de caramelo se fotografían bajo luces de varios colores y cámara B/N [9]

- **Composición del objeto y transmitancia**

La composición del objeto/muestra puede afectar en gran medida lo que sucede con la iluminación. Algunos plásticos pueden transmitir luz solo de ciertos rangos de longitud de onda y son opacos; algunos pueden no transmitir, sino que difunden internamente la luz; y algunos pueden absorber la luz solo para reemitirla en la misma longitud de onda o en una longitud de onda diferente (fluorescencia). Las etiquetas y tintes fluorescentes también se usan comúnmente en la industria de la impresión (Figura 18) [9].

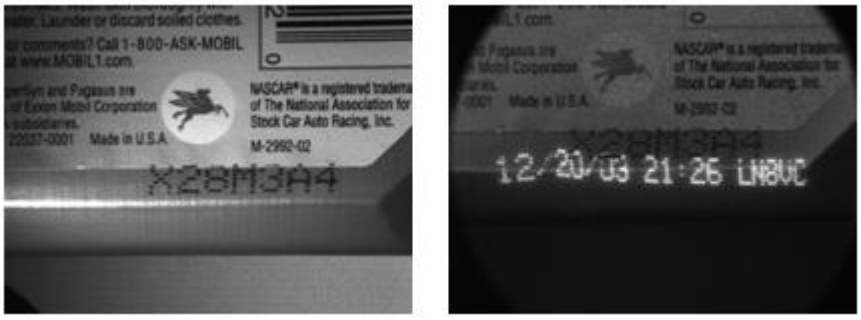


Figura 18 La botella de aceite se ilumina con un anillo rojo de 660nm. a la derecha se ilumina con luz fluorescente UV a 360 nm [9].

Particularmente cuando se quiere bloquear reflexiones en objetos, cualquier uso de filtros de polarización conlleva compromisos inherentes. Las imágenes representadas en la Figura 19 demuestran el uso menos efectivo y altamente efectivo de los filtros de polarización específicamente para bloquear el deslumbramiento. En las muestras representadas en la Figura, se observa que el reflejado ocurre sobre una superficie curva. Esto ocurre porque se producen múltiples direcciones de reflexión en la superficie curva desde una fuente de luz direccional, y los filtros de polarización no pueden bloquear todas las direcciones simultáneamente, dejando siempre algunas áreas con viñetas. En este caso, un enfoque más efectivo para el control del deslumbramiento, si permiten las restricciones de la aplicación para hacerlo, es reconsiderar la geometría de la iluminación. Simplemente moviendo la luz desde una posición coaxial alrededor de la lente a un ángulo relativamente alto, pero fuera del eje, puede eliminar por completo toda la reflexión [9].

Figura 19 El efecto del cambio de la posición de la cámara [9]

2.1.2.3.2. Técnicas de iluminación

El éxito de una aplicación de visión está altamente condicionado por la iluminación que se utilice. En el caso de no utilizar una iluminación adecuada, aparecen problemas de contrastes, brillos y sombras que complican el algoritmo de inspección o que incluso pueden llegar a que el algoritmo no pueda obtener una solución. Para mejorar notablemente la eficiencia del sistema de visión es necesario utilizar la técnica de iluminación adecuada que permita obtener una imagen correcta para ser procesada. Una imagen correcta para el procesado es aquella en la que los píxeles que representan los objetos de interés en la misma tienen características de luminosidad parecida y son muy distintas de los píxeles que no representan objetos de interés. Una iluminación apropiada es crítica para obtener una imagen correcta en la que no aparecen zonas saturadas o sombras que oculten información dentro de la imagen. Además, las sombras causan falsas detecciones de bordes resultando en medidas incorrectas. También, una iluminación pobre puede resultar en una baja relación señal/ruido, lo que puede provocar una imagen con píxeles ruidosos. Igualmente, una iluminación no uniforme puede además dificultar notablemente operaciones de segmentación [6].

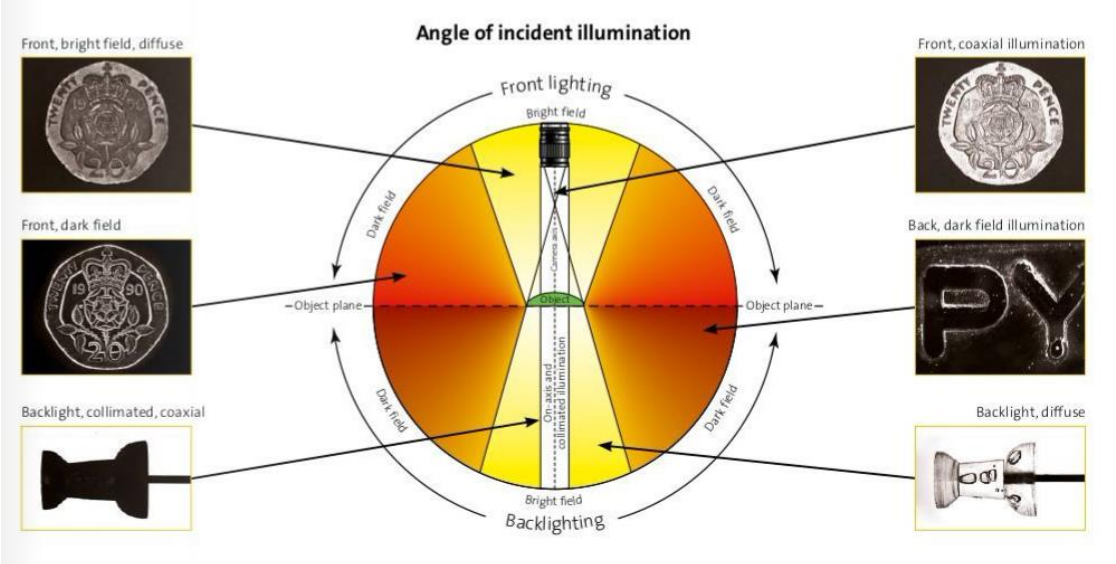


Figura 20 Iluminación según el ángulo de incidencia [11]

- **Iluminación frontal**

La cámara se posiciona mirando al objeto en la misma dirección que la luz. Esto reduce las sombras, suaviza las texturas y minimiza la influencia de rayas, polvo e imperfecciones que pueda tener el objeto. La cámara recibe la luz reflejada del objeto. Este tipo de iluminación se consigue mediante anillos de luz [6].

- Aplicaciones: indicada para superficies con pocos reflejos: papel, tela… para la detección de marcas de diferentes colores, caracteres y detección de todo lo que suponga un cambio de color en prácticamente cualquier superficie.

- Ventajas: elimina sombras, se puede utilizar a grandes distancias cámara/objeto.

- Inconvenientes: intensos reflejos sobre superficies reflectantes.

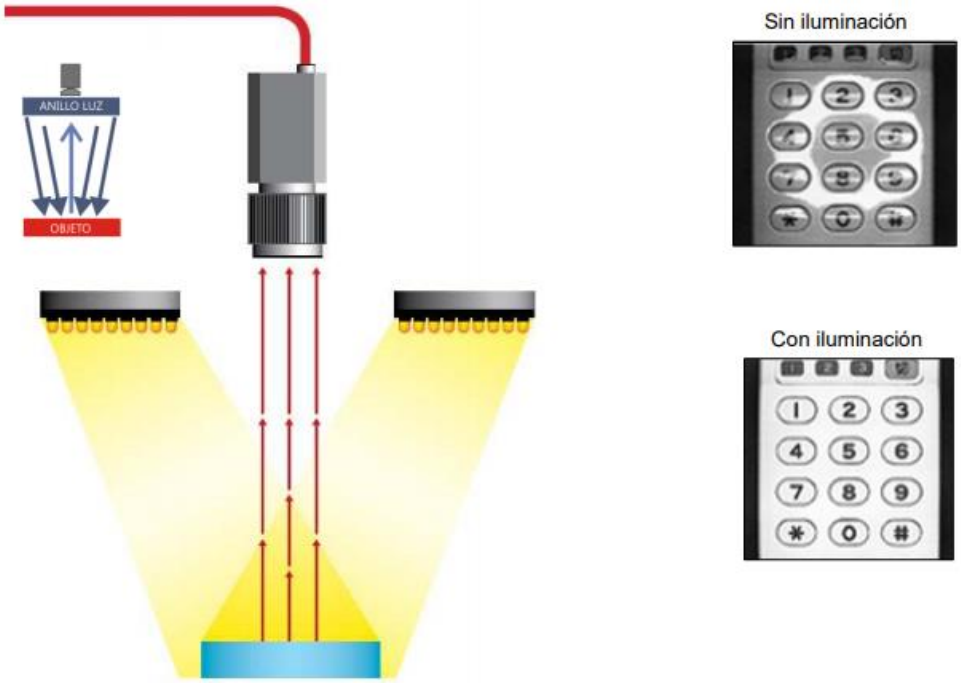


Figura 21 Iluminación frontal [11]

- **Iluminación lateral**

La cámara se posiciona mirando al objeto mientras que la dirección de la luz es lateral al objeto. El grado de inclinación del elemento emisor de luz vendrá determinado por el grado deseado de resalte de los relieves [6].

- Aplicaciones: indicada para resaltar bordes, rayas y fisuras en una dirección determinada.
- Ventajas: resalta los relieves por pequeños que sean de los objetos, resultando una sombra muy definida.
- Inconvenientes: con ángulos pequeños respecto a la horizontal, la luz producirá sombras en todos los relieves y en el contorno de la pieza.

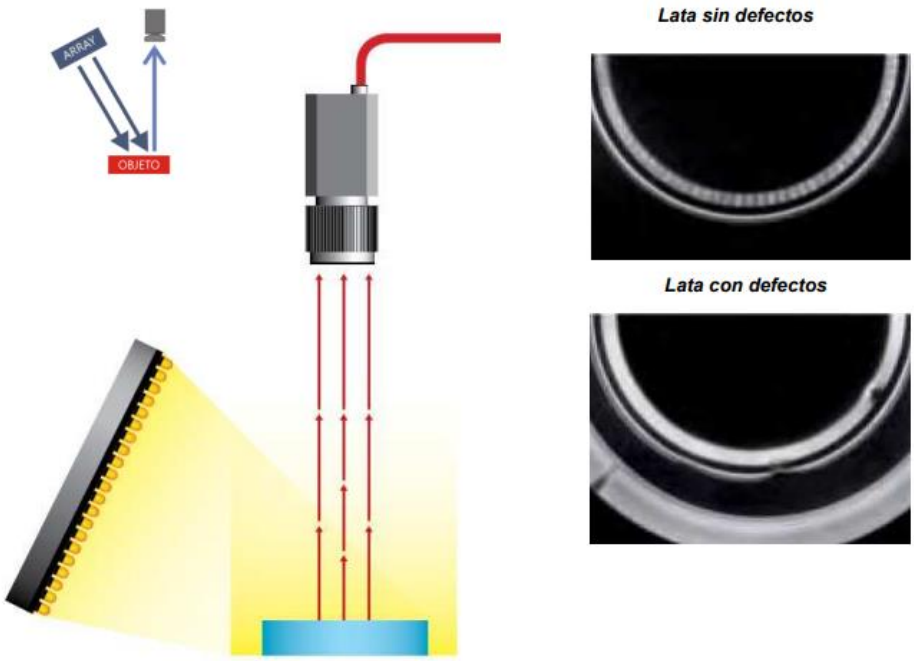


Figura 22 Iluminación lateral [11]

- **Iluminación por campo oscuro (Dark field)**

La luz es emitida lateralmente con un ángulo muy pequeño mediante un anillo en todas las direcciones, rebotando en los defectos del objeto a analizar e incidiendo en la cámara [6].

- Aplicaciones: indicada para resaltar incrustaciones y códigos alfanuméricos con poco contraste en metal sobre metal o gris sobre gris. Muy utilizada en la verificación de grabados tipo láser o troquel.
- Ventajas: destaca los detalles en superficies con muy poco contraste.
- Inconvenientes: no es recomendable en superficies que absorban la luz.

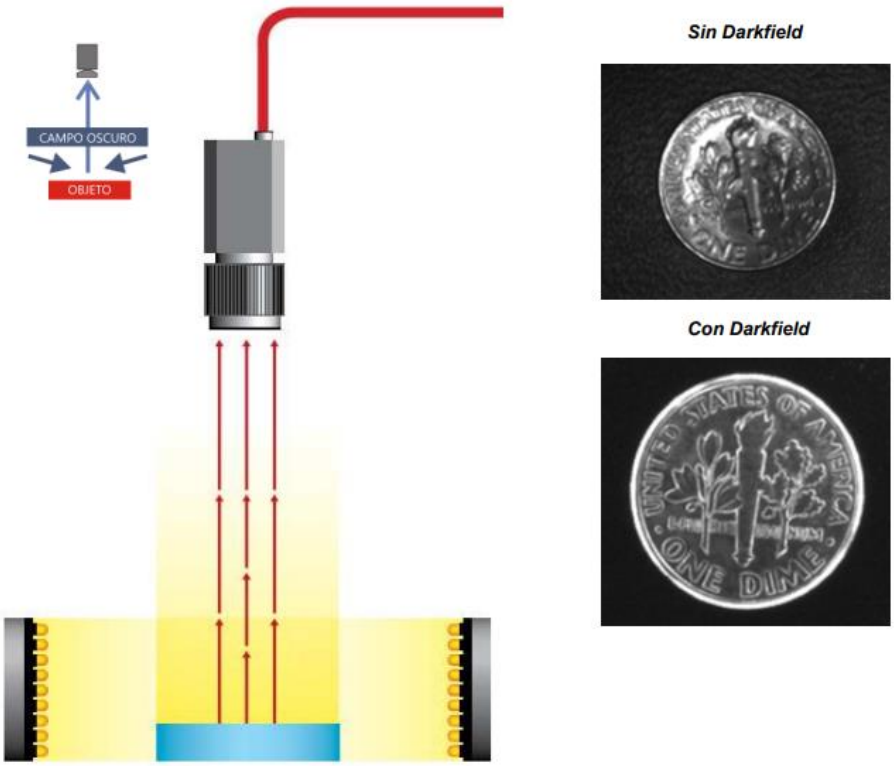


Figura 23 Iluminación de campo oscuro [11]

- **Iluminación por contraste (Back light)**

La luz es emitida desde la parte posterior del objeto quedando este entre la fuente de iluminación y la cámara. La iluminación tiene que ser uniforme en toda la superficie del objeto. La cámara inspecciona la silueta del objeto por contraste pudiendo realizar mediciones muy precisas, ya que se eliminan por completo las sombras producidas por la iluminación.

- Aplicaciones: indicada para la inspección de la silueta del objeto. Utilizada también en materiales translúcidos o transparentes para detectar manchas, rayas, grietas…
- Ventajas: permite inspecciones de siluetas con mediciones muy precisas y de impurezas en los objetos transparentes o translúcidos.
- Inconvenientes: no permite reconocer los detalles superficiales del objeto, códigos, inscripciones, etc [6].

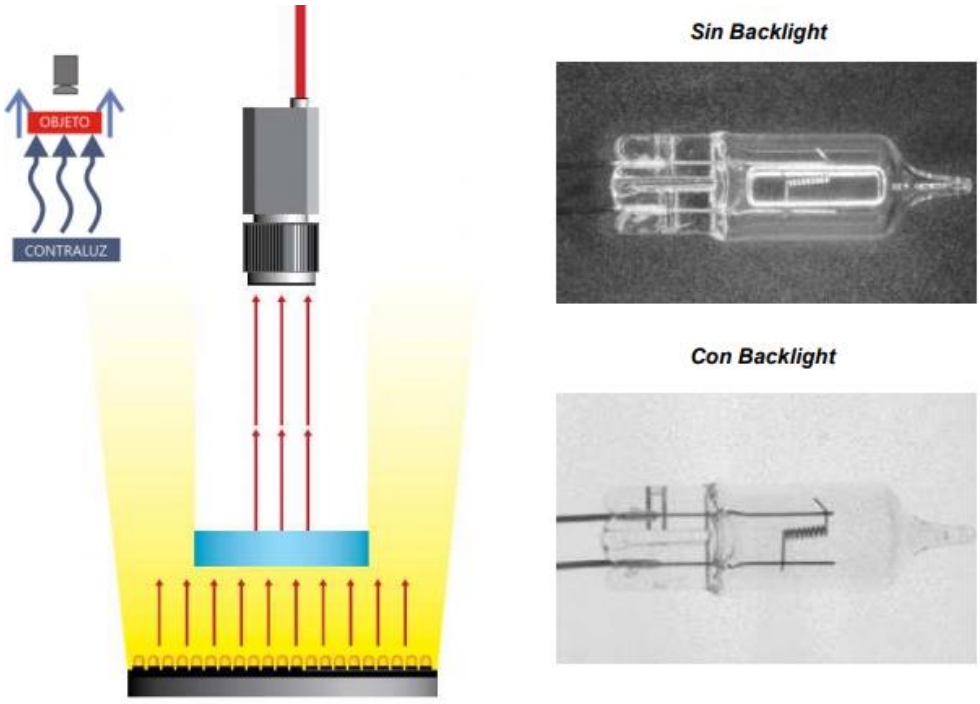


Figura 24 Iluminación por contraste [11]

- **Iluminación axial difusa**

La luz es emitida lateralmente siendo reflejada 90º por un espejo semitransparente que desvía los haces de luz en la misma dirección que el eje de la cámara, consiguiendo una luz difusa homogénea. En superficies planas reflectantes si no se utiliza este método de iluminación, la cámara vería reflejado su propio objetivo.

- Aplicaciones: indicada para la inspección superficies planas reflectantes, como PCB, etiquetas reflectantes, inspección de impresión sobre aluminio o cavidades profundas.

- Ventajas: permite inspecciones de códigos en materiales altamente reflectantes.

- Inconvenientes: no permite reconocer relieves en el objeto [6].

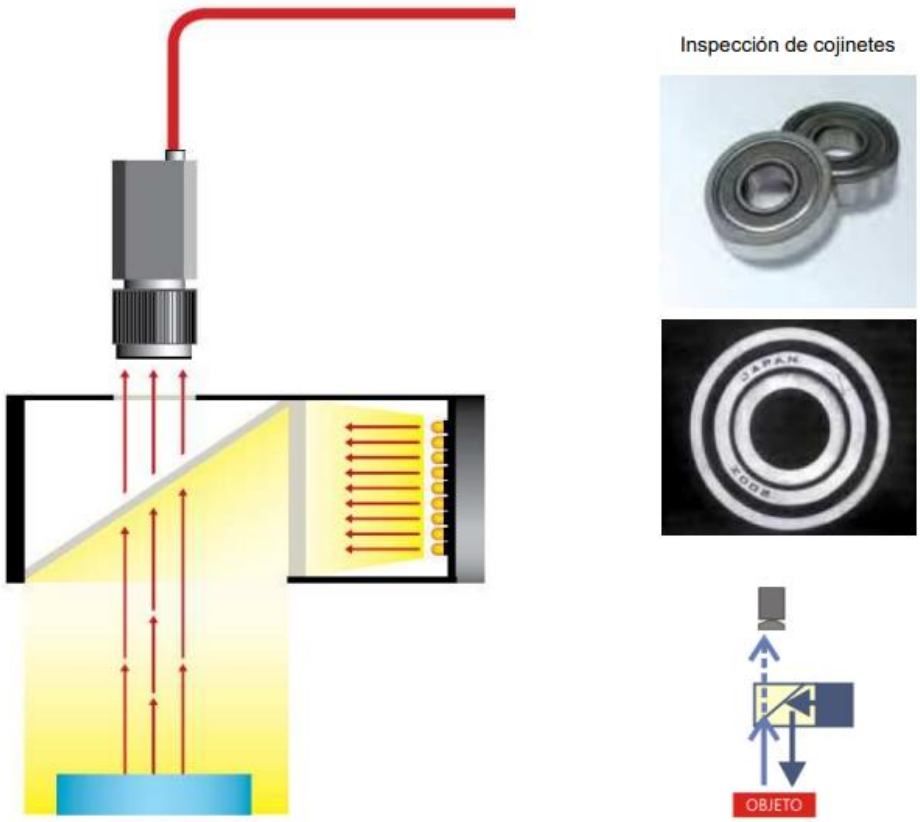


Figura 25 Iluminación axial difusa [11]

- **Iluminación difusa tipo DOMO**

La luz es emitida dentro de una cúpula esférica resultando una luz difusa desde todas direcciones, eliminando sombras y reflejos, suavizando texturas y minimizando la influencia de rayas, polvo, relieves y curvaturas que pueda tener el objeto inspeccionado. A este tipo de iluminación también se le denomina iluminación de día nublado por no producir ningún tipo de sombra al objeto.

- Aplicaciones: indicada para la inspección de superficies tales como instrumental médico, espejos, compact disk, latas, etc.
- Ventajas: eliminación de sombras y minimizador de arrugas, polvo y relieves.
- Inconvenientes: coste elevado [6].

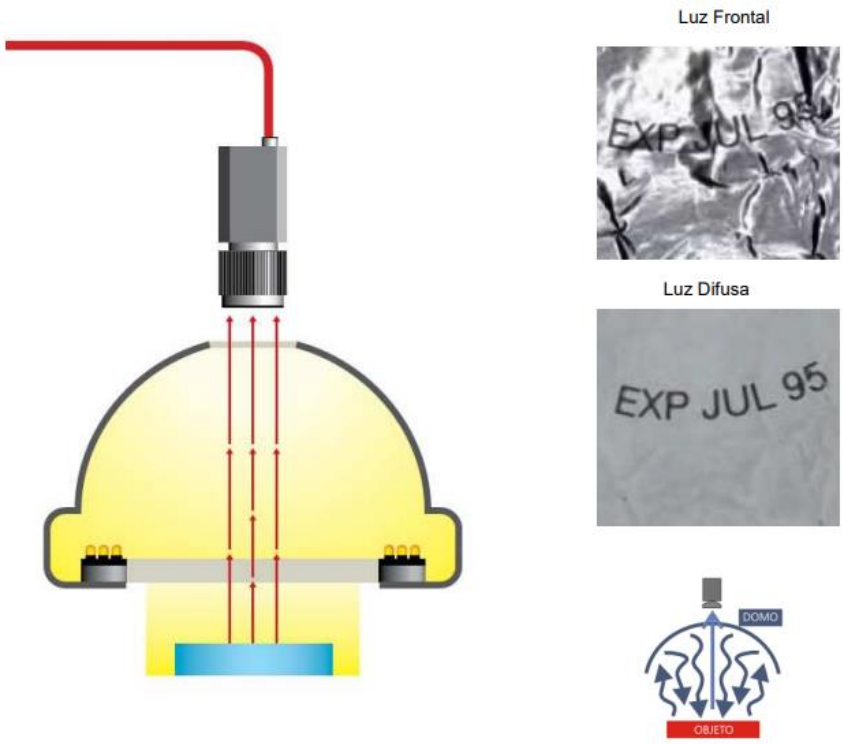


Figura 26 Iluminación DOMO [11]

- **Iluminación por Láser**

La iluminación mediante láser o luz estructurada se utiliza normalmente para resaltar o determinar una tercera dimensión de un objeto. Se trata de colocar la fuente de luz láser en un ángulo conocido con respecto al objeto a iluminar y a la cámara, de forma que viendo la distorsión de la luz pueda interpretarse la profundidad de los objetos a medir. También se utiliza para indicar el trazado por el que se debe ajustar un proceso, por ejemplo, en aplicaciones de corte.

Para realizar una inspección en 3D de un objeto, se proyecta una línea de luz. Las distorsiones en la línea se traducen en variaciones en la altura. De aquí se puede desprender una forma en 3D detectando la falta o exceso de material o llegar a hacer una reconstrucción en tres dimensiones del objeto.

- Aplicaciones: ajuste de procesos de corte, control de profundidad de objetos...
- Ventajas: no le influye la iluminación externa.
- Inconvenientes: coste elevado. Si se utilizan lentes cilíndricas para conseguir

una línea o un patrón concreto, el láser no tiene la misma intensidad lumínica a lo largo del patrón [6].

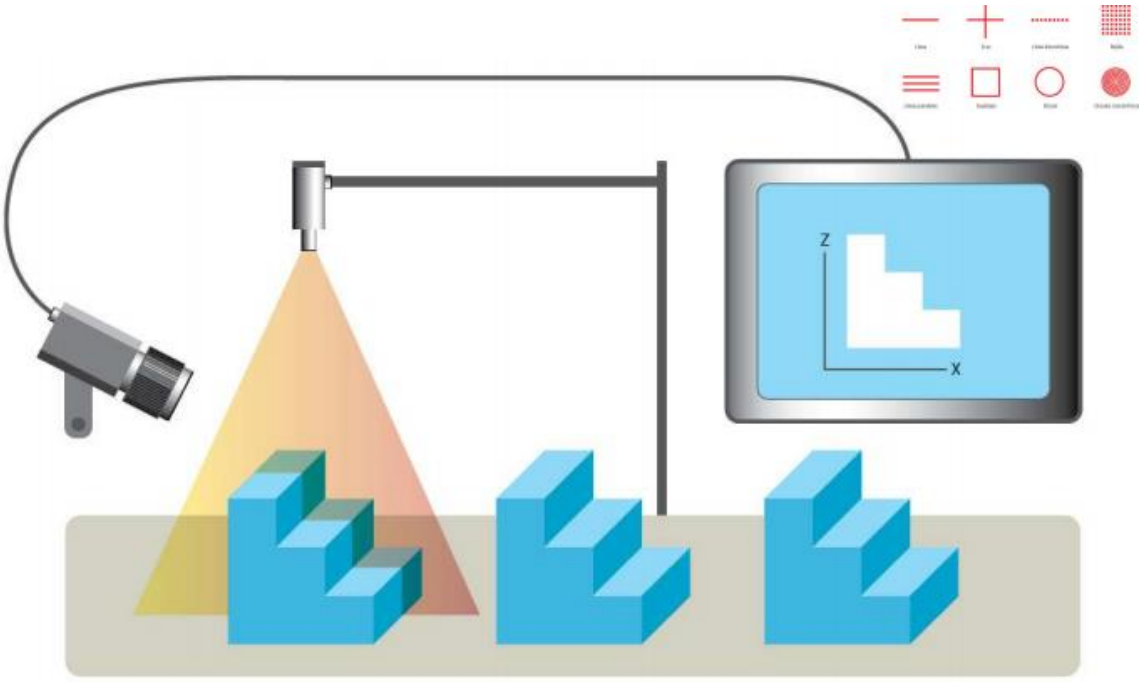


Figura 27 Iluminación por Láser [11]

2.1.2.3.3. Cámaras

La cámara es el dispositivo que, utilizando un objetivo formado por un juego de lentes y el diafragma, construye una imagen sobre el plano del sensor compuesto de elementos fotosensibles, la digitaliza y la transmite hacia la tarjeta de adquisición del procesador. Están compuestas por un sensor y la electrónica asociada. Las cámaras proporcionan una señal de vídeo en un formato estándar para su digitalización (en el caso analógico) o directamente la información en formato digital que constituye la imagen captada por la misma (en el caso de cámaras digitales) [6].

- **Montura entre la óptica y el cuerpo de cámara**

La montura de la cámara puede ser C, CS o F. La montura C es la más común la cual deja 17,5 mm de separación entre la óptica y el sensor. La montura CS es compatible con la C si se utiliza un separador de 5mm ya que la distancia entre el sensor y la óptica con esta montura es de 12,5 mm. La montura F se utiliza en cámara con sensores grandes. La distancia entre la

óptica y el sensor con este tipo de monturas es de 46,5 mm la cual la hace ideales para cámaras con sensores lineales [6].

- **Formato del sensor**

CCD (Charge Coupled Device) y CMOS (Complementary Metal Oxide Semiconductor) son diferentes tecnologías utilizadas en la construcción del sensor para convertir la luz en señales eléctricas. Los sensores CMOS se utilizan en aplicaciones donde es necesario un bajo consumo de energía o en las que el espacio es reducido. El nivel de ruido en una imagen capturada con tecnología CMOS es mayor que una captura con tecnología CCD. Además, el rango dinámico del sensor es notablemente menor que el de un sensor de las mismas características desarrollado con tecnología CCD. Por el contrario, los sensores CMOS son más rápidos y tienen un menor consumo que los sensores desarrollados con tecnología CCD. Los sensores CCD se recomiendan en aplicaciones donde es necesaria una mayor calidad de imagen.

El elemento sensible de la inmensa mayoría de las cámaras actuales se compone de una matriz de sensores de acoplamiento de carga o CCDs. Cuando un CCD se ve expuesto a la luz, cada elemento del mismo absorbe una cierta cantidad de fotones, los cuales provocan la aparición de cargas eléctricas que se almacenan en un condensador MOS. Una propiedad importante de un sensor CCD es la posibilidad de transferir la carga en el momento que se decida, obteniéndose así un efecto de obturación electrónica, ya que sólo se convertirá en señal la luz durante el tiempo de exposición. Estas cargas se transmiten inmediatamente a través de la puerta (gate) a un registro de desplazamiento que se encargará de almacenarla y luego volcarla secuencialmente al exterior.

El sensor puede tener formato de área o lineal. Los sensores con formato de área son rectangulares y suelen tener un ratio típico 4:3(H:V). Los sensores de área son más grandes que los sensores lineales, ya que los sensores lineales capturan simplemente una línea de píxeles. El tamaño de la línea medida en píxeles es el que define la resolución del sensor. Los sensores de área son mucho más baratos que los sensores lineales ya que se utilizan en multitud de aplicaciones y son muy fáciles de configurar. Los sensores lineales se utilizan para capturar imágenes de objetos grandes ya que para capturar una imagen se

van acumulando líneas sucesivas hasta que se forma la imagen. Para esto se necesita un movimiento relativo entre la cámara y el objeto de forma que el escaneo sea posible, lo que complica notablemente la captura de la imagen. Por el contrario, dado que sólo es necesario iluminar una línea de la escena, el apartado de iluminación es mucho más simple y las imágenes que se obtienen tienen una iluminación más homogénea [6].

Para obtener una imagen en color o monocroma existen diversas configuraciones. La siguiente tabla muestra los pros y los contras de cada una de ellas.

MONOCROMO	COLOR CON UN SOLO SENSOR	COLOR CON 3 SENSORES
• Un solo sensor que recoge imágenes en escala de grises. • 10% más de resolución si se compara con una imagen en color capturada con un solo sensor. • Mejor relación señal/ruido. • Mayor contraste. • Mejor sensibilidad en condiciones de poca luz. • Ideal para aplicaciones de medida	• Usa un filtro de Bayer color RGB. • Más barato. • Menor resolución de imagen para un sensor con los mismos elementos sensibles (son necesarios más elementos sensibles para reconocer el color de un píxel). 	• Utiliza un prisma para separar la luz blanca en tres sensores. • Más caro. • Mejor resolución para una imagen en color. • Menor sensibilidad • Menos ópticas disponibles para este tipo de sensores.

Tabla 3 Técnicas para imágenes monocromas o en color [6]

El tamaño del sensor determina el campo de visión del sistema y el zoom necesario para poder capturar una imagen de la escena. El tamaño puede variar desde ¼ de pulgada (3,6x2,7 mm) hasta de 35 mm (36x24 mm) pasando por diferentes medidas tales como 1/2 pulgada (6,4x4,8 mm), 1 pulgada (12,8x9,6 mm), etc.

Para poder seleccionar la cámara y la óptica que compongan el sistema de visión que mejor se adecue a la aplicación que se desea resolver es necesario tener en cuenta los siguientes parámetros:

- Campo de visión (Field Of View, FOV): área del objeto o escena del que se desea capturar una imagen.
- Resolución: tamaño de la característica más pequeña del objeto que se desea que se vea en la imagen.

- Distancia de trabajo: separación que existe entre el objeto que se captura la imagen y la óptica.
- Profundidad de campo: máxima profundidad del objeto necesaria para conseguir un enfoque adecuado [6].

2.1.2.3.4. Lentes

Una lente se caracteriza por un índice de refracción, una reflexión y una dispersión definida según el índice de Abbe. El índice de refracción n, describe la capacidad que tiene la lente para reducir la velocidad de la luz cuando pasa a través de ella. Se define como un cociente entre la velocidad de la luz en el vacío y la velocidad de la luz al cuando pasa a través de la lente [6].

$$n = \frac{velocidad\ de\ la\ luz\ en\ vacio}{velocidad\ de\ la\ luz\ en\ el\ material} = \frac{c}{v} \qquad [1]$$

Las lentes con un bajo índice de refracción tienen menos reflexión de forma natural. Para aumentar el índice de reflexión en las lentes con un alto índice de refracción, se utilizan recubrimientos que permiten reducir la reflexión. La dispersión describe la variación del índice de refracción con la longitud de la onda de la luz que atraviesa la lente. Se mide utilizando el índice de Abbe, y permite conocer la separación de colores que producirá la lente cuando la luz atraviese la misma. Un índice de Abbe pequeño indica una alta dispersión y una significativa separación de los colores, mientras que un índice de Abbe grande indica una baja dispersión y una menor separación de los colores.

$$vd = \frac{nd - 1}{nf - nc} \qquad [2]$$

Siendo:

nd = índice de refracción en 587.6nm (verde).
nf = índice de refracción en 486.1nm (azul).
nc = índice de refracción en 656.3nm (rojo).

Finalmente, la reflexión es el fenómeno que se produce cuando la luz cambia de medio y una parte de ella es devuelta al medio de procedencia. Por ejemplo, cuando la luz pasa del aire al cristal de la lente, se tiene entre un 8% y un 10% de pérdidas, las cuales dependen de la longitud de onda de la misma.

Para reducir la reflexión, se añaden recubrimientos a la lente que permiten tener unas pérdidas de hasta el 0.25% o menos. Sin embargo, aunque el recubrimiento anti-reflexión mejora significativamente el comportamiento de la lente, es necesario tener en cuenta, que esta mejora sólo se produce para las longitudes de onda para las cuales el recubrimiento anti reflexión ha sido diseñado. Fuera de estas longitudes de onda el comportamiento de la lente puede ser diferente al esperado. Este efecto se utiliza para construir lentes que filtren unas determinadas longitudes de onda [6].

2.1.2.3.5. Ópticas

La óptica se encuentra dispuesta en el objetivo, que es un conjunto complejo de lentes y mecánica dispuestas en un cilindro metálico y que dispone de una rosca o bayoneta que permite su incorporación al cuerpo de la cámara y su intercambio.

Las ópticas permiten enfocar luz procedente de un punto en un lado de la óptica en otro punto detrás de la misma. De esta forma, la óptica permite determinar la escena que se va a capturar en forma de imagen de un objeto situado a una distancia finita de la misma y la enfoca en el sensor que también está a una distancia fija al otro lado.

Para determinar cuál es la lente apropiada para la aplicación que se desea resolver, se deben tener en cuenta una serie de parámetros: tamaño del sensor de la cámara, distancia entre la cámara y el objeto y campo de visión o tamaño del objeto. Con estos datos y utilizando una fórmula podemos calcular la óptica a utilizar en cada aplicación, pero antes de realizar estos cálculos necesitamos conocer unos conceptos básicos de las ópticas [6].

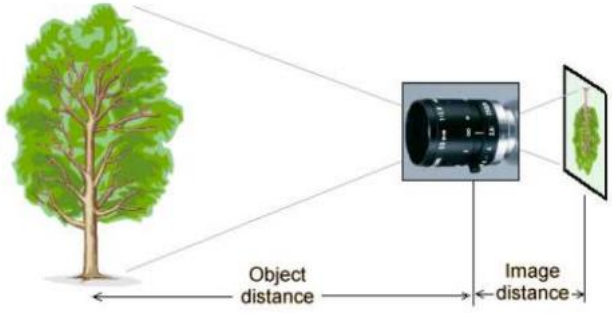


Figura 28 Distancia al objeto [6]

- **Distancia focal**

La distancia focal es la medida en mm relativa a la distancia entre la lente y el elemento sensor. La imagen sale invertida debido a la lente del objetivo, que invierte la imagen al recibir los rayos de luz. Los objetivos de las cámaras tienen una distancia focal fija o variable, dependiendo del tipo de objetivo. Al variar la distancia focal conseguimos un menor o mayor acercamiento del objeto, comúnmente llamado zoom.

Las ópticas pueden estar formadas por una única lente o varias lentes. La distancia focal f de las ópticas formadas por una única lente coincide con la distancia focal de la misma. En el caso de utilizar varias lentes el coste aumenta y la distancia focal de la óptica depende de las distancias focales de las lentes y de la distancia que separa las mismas [6].

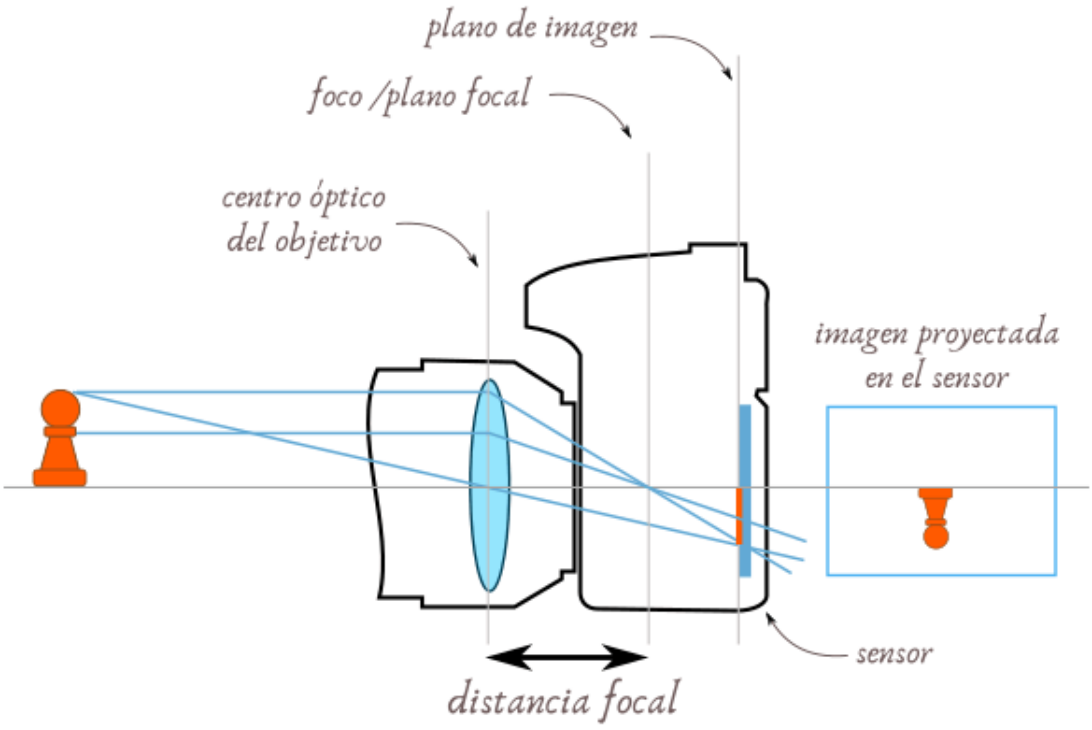


Figura 29 Distancia focal [12]

Estando a la misma distancia de trabajo, lentes con diferente distancia focal muestran la misma imagen en distinto tamaño.

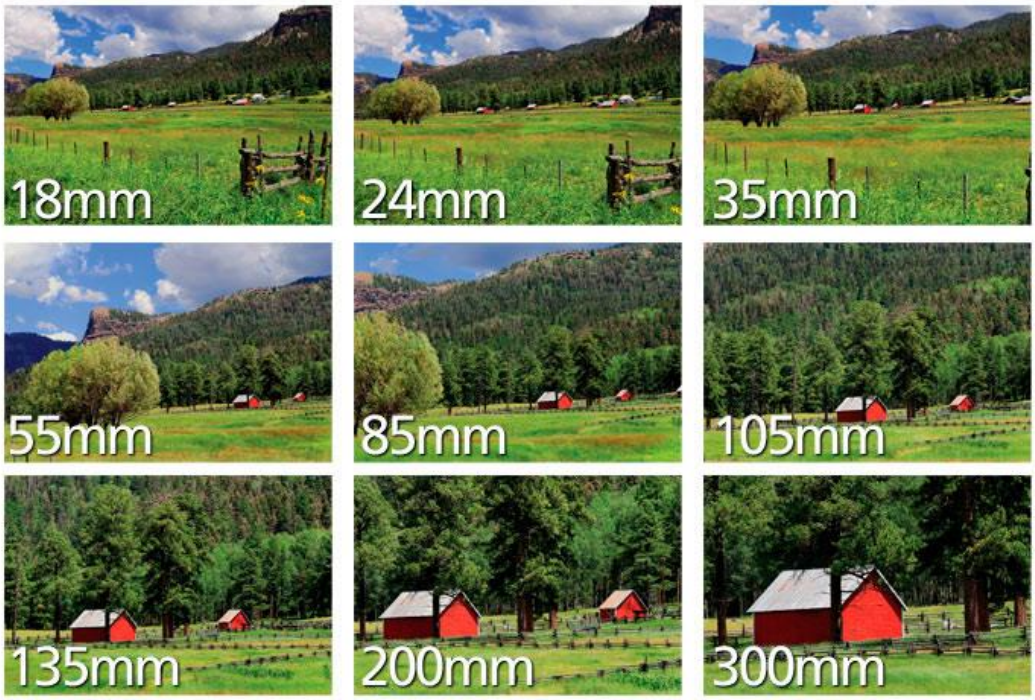


Figura 30 Imágenes a diferentes distancias focales [13]

- **Foco o enfoque**

Normalmente las ópticas utilizadas en sistemas de visión artificial tienen una distancia focal variable, también denominada foco o enfoque. Es muy importante que el objeto a analizar esté perfectamente enfocado para su posterior análisis. El rango en que una lente ofrece nitidez está entre el infinito y una distancia mínima esta distancia depende de la distancia focal [14] .

Figura 31 Anillo de enfoque [15]

- **Profundidad de campo**

Es la distancia en la que los objetos aparecen enfocados. A menor apertura del diafragma, mayor será la profundidad de campo.

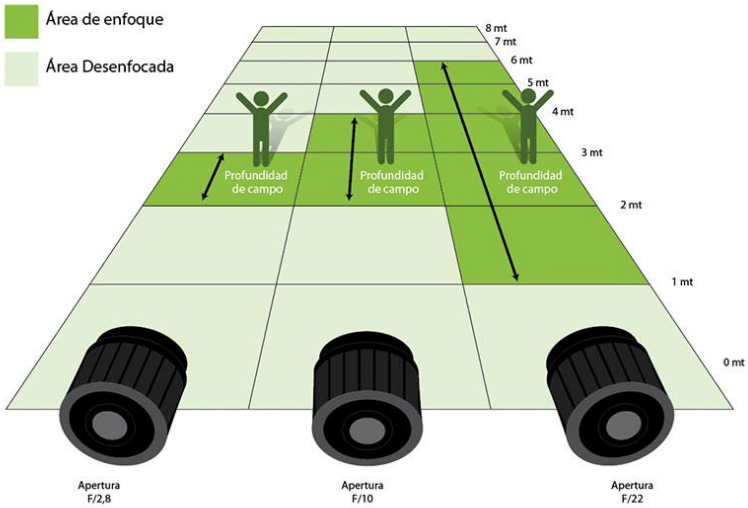


Figura 32 Profundidad de campo [16]

- **Apertura del diafragma**

Las ópticas disponen de un diafragma que permiten regular la cantidad de luz que llega al sensor. La apertura del diafragma condiciona la profundidad de campo, la sensibilidad del sensor y la cantidad de luz necesaria en la escena para capturar una imagen de calidad. El número f máximo, que es la mayor apertura del diafragma en la escala F. Ésta es: 1, 1.4, 2, 2.8, 4, 5.6, 8, 11, 16, 22, 32 y cada una de las cifras de esta escala (de izquierda a derecha) indican que el objetivo deja pasar el doble de luz que la cifra anterior. En dicha escala el paso de una cifra a la siguiente, en cualquier sentido, se denomina un punto del diafragma o stop [14].

Figura 33 Exposición [17]

Las ópticas también se definen en base al ángulo de visión que abarcan. Generalmente se utiliza el ángulo de la diagonal, aunque se suele incluir el horizontal que resulta mucho más expresivo para el ser humano, acostumbrado a una visión de tipo panorámico en sentido horizontal. El campo de visión más amplio corresponde a los objetivos denominados gran angular y el más pequeño a los teleobjetivos. En consecuencia, al cambiar la distancia focal, lo que se hace en la práctica es variar el tamaño en la imagen de un determinado objeto. Así, por ejemplo, si el objeto tiene un tamaño de 1 milímetro en la imagen capturada con un objetivo de una focal de 25 mm, con un objetivo de una focal cuádruple (100mm) tendrá un tamaño de 4 milímetros [14].

Los objetivos se pueden clasificar en objetivos normales, de focal larga y de focal corta, siendo la focal normal la que capta imágenes dentro de un ángulo similar al de la visión humana, con una perspectiva de los objetos equivalente a la que percibe el ser humano con sus ojos.

Existen diversos tipos de ópticas cada una de las cuales es útil para unas determinadas aplicaciones. Las ópticas con distancia focal fija tienen como característica un ángulo de campo de visión único definido por una distancia focal fija. Tienen una mínima distancia al objeto (MOD) a partir de la cual es posible enfocar el mismo. Los objetos que están más cercanos a la lente aparecen

más grandes que los objetos que se sitúan más lejos. Se caracterizan porque son ligeras y pequeñas, por lo que son ampliamente utilizadas en aplicaciones de visión artificial.

Teniendo en cuenta la perspectiva, un objeto que está más cerca de la lente aparece más grande en la imagen que uno que está más lejos. Este hecho puede producir un error de perspectiva que es apreciable si la relación entre la distancia del objeto a la lente y la profundidad del mismo es del mismo orden. Para solucionar este error, en lugar de utilizar ópticas convencionales, se utilizan ópticas telecéntricas que evitan este error de perspectiva y las hacen ideales para aplicaciones de medición.

Las ópticas telecéntricas ofrecen una muy buena calidad de imagen y el orden de amplificación de las mismas es de 0.06x hasta 6x. Las ópticas con zoom o varifocales permiten cambiar la distancia focal o el ángulo del campo de visión de la lente. Los sistemas ópticos con zoom mantienen enfocado el objeto mientras la distancia focal se modifica. Por el contrario, con las ópticas varifocales, es necesario reenfocar de nuevo el objeto cada vez que se cambia la distancia focal. El cambio de distancia focal permite diferentes campos de visión para ser capturados sin necesidad de cambios en el resto del sistema de visión. Las ópticas con zoom son ideales para aplicaciones donde es necesario inspeccionar con un único sistema de visión objetos grandes teniendo en cuenta algunos pequeños detalles.

Los objetivos y las lentes para microscopio tienen un nivel de amplificación elevado ofreciendo una calidad de imagen superior. Estos objetivos pueden trabajar en un amplio rango de distancias al objeto de trabajo. Además, estos objetivos tienen la posibilidad de fácil integración con el sistema de iluminación o filtros específicos [14].

2.1.2.3.6. Estándares de vídeo

Las cámaras digitales disponen de muchas opciones de comunicación, cada una de las cuales se debe ajustar a las necesidades de la aplicación. Algunos formatos como por ejemplo Firewire permiten obtener una salida de vídeo y alimentación en un solo conector, lo que posibilita reducir notablemente los elementos que componen el sistema de visión. La velocidad de transferencia de

datos afecta directamente al número de capturas o frames por segundo que se pueden obtener de la cámara. Este dato también se ve condicionado por la resolución de la imagen y la profundidad de color (número de bits con el que se representa la información de cada píxel). La siguiente tabla muestra una comparativa de los diferentes tipos de interface de comunicaciones que existen en la actualidad [14].

	FireWire 1394.a	FireWire 1394.b	Camera Link	USB 2.0 (USB 3.0)	GigE(POE)
Tasa Transferencia de datos	400Mb/s	800Mb/s	Hasta 3.6Gb/s	480Mb/s (5Gb/s)	1000Mb/s
Longitud máxima de cable	4,5 m	100 m	10m	5m	100m
Número de cámaras	Hasta 63	Hasta 63	1	Hasta 127	Ilimitada
Conector	6 pines-6 pines	9 pines-9 pines	26 pines	USB	RJ45/CAT5e ó 6
Tarjeta de captura	Opcional	Opcional	Necesaria	Opcional	No es necesaria
Alimentación	Opcional	Opcional	Necesaria	Opcional	Necesaria (Opcional)

Tabla 4 interfaces de comunicación

2.1.2.3.7. Tarjetas de adquisición

Es la parte encargada de recibir la transmisión de la señal digital de vídeo que proporciona la cámara. En su funcionamiento cabe destacar como parámetros:

- La resolución o definición espacial, esto es, el número de puntos discretos por imagen o cuadro (píxeles).
- La resolución digital, que es el número de bits que se utilizan para codificar los niveles de intensidad luminosa de cada punto resultante de la digitalización.

Estas tarjetas (frame grabbers) agrupan los elementos de decodificación electrónica del protocolo de comunicación junto con una memoria local. Permiten básicamente recibir y almacenar imágenes, pudiéndose encontrar

incluida en ellas varias funciones, como son:

- Adquisición de secuencias de imágenes.
- Aceleración del proceso de visualización en pantalla.
- Procesado de imágenes [14].

2.1.2.3.8. Software

Las técnicas de procesado y análisis de imagen para entornos industriales y científicos son relativamente recientes, sus inicios los podemos encontrar hace unos 30 años, y han evolucionado muy rápidamente ayudados a su vez por el rápido avance de los ordenadores y su potencia de cálculo [18].

En el pasado más reciente no era posible hacer los procesos en tiempo real debido a que los ordenadores no eran lo suficientemente rápidos para realizar los cálculos con las imágenes. Los procesos en tiempo real en ese momento se debían hacer en procesadores DSP a bordo de las placas, con el fin de poder alcanzar las velocidades requeridas, para la mayoría de las aplicaciones.

Con la llegada del bus PCI y con la rápida evolución de los procesadores de los PC se ha conseguido visualizar las imágenes en tiempo real y realizar la mayoría de procesos en tiempos suficientemente cortos, como para que puedan resolver aplicaciones de visión en entornos científicos e industriales, con los resultados esperados en su justo tiempo.

Esta evolución del hardware ha comportado el desarrollo de librerías de visión que puedan funcionar en entornos estándar, tanto de sistemas operativos como de procesadores.

El sistema operativo más utilizado en la actualidad en las aplicaciones de visión es el Windows, en cualquiera de sus variedades.

Sin embargo, existen muchas aplicaciones desarrolladas en UNIX, QNX y últimamente se está utilizando con gran asiduidad el Linux, tanto en su versión estándar como en la RT (RealTime).

Hasta hace pocos años la implementación de sistemas de visión artificial

requería un extenso conocimiento del software de bajo nivel y del hardware de visión. Actualmente, el panorama ha cambiado radicalmente, ya que se encuentran disponibles numerosos entornos de programación escalables y fáciles de utilizar, que combinados con los nuevos procesadores hacen muy fácil la implementación de un sistema de visión [18].

La base del software de un sistema de visión es la interpretación y análisis de los píxeles. El resultado final puede ser, desde la medida de una partícula, a la determinación o lectura de una serie de caracteres (OCR), pasando por cualquier otro proceso que podamos imaginar sobre las imágenes.

Dependiendo de si la aplicación se realiza en entorno industrial o científico, los pasos a seguir en un sistema de visión serán algo distintos.

Básicamente, hay cuatro niveles de software de visión. Cada uno de estos niveles requiere diferentes grados de programación y conocimiento de los entornos de visión por parte del programador o usuario.

- Sistemas de programación a bajo nivel basado en kits de desarrollo de software (sdk): Normalmente utilizan librerías en DLL o Active X y requiere un amplio conocimiento de hardware y software de visión y un conocimiento de programación en lenguajes estándar tales como Visual C o Visual Basic.

- Sistemas de programación por menú: Son sistemas de fácil utilización que no requieren ningún tipo de experiencia en programación.

- Aplicaciones específicas: Son programas específicamente diseñados para resolver aplicaciones concretas [18].

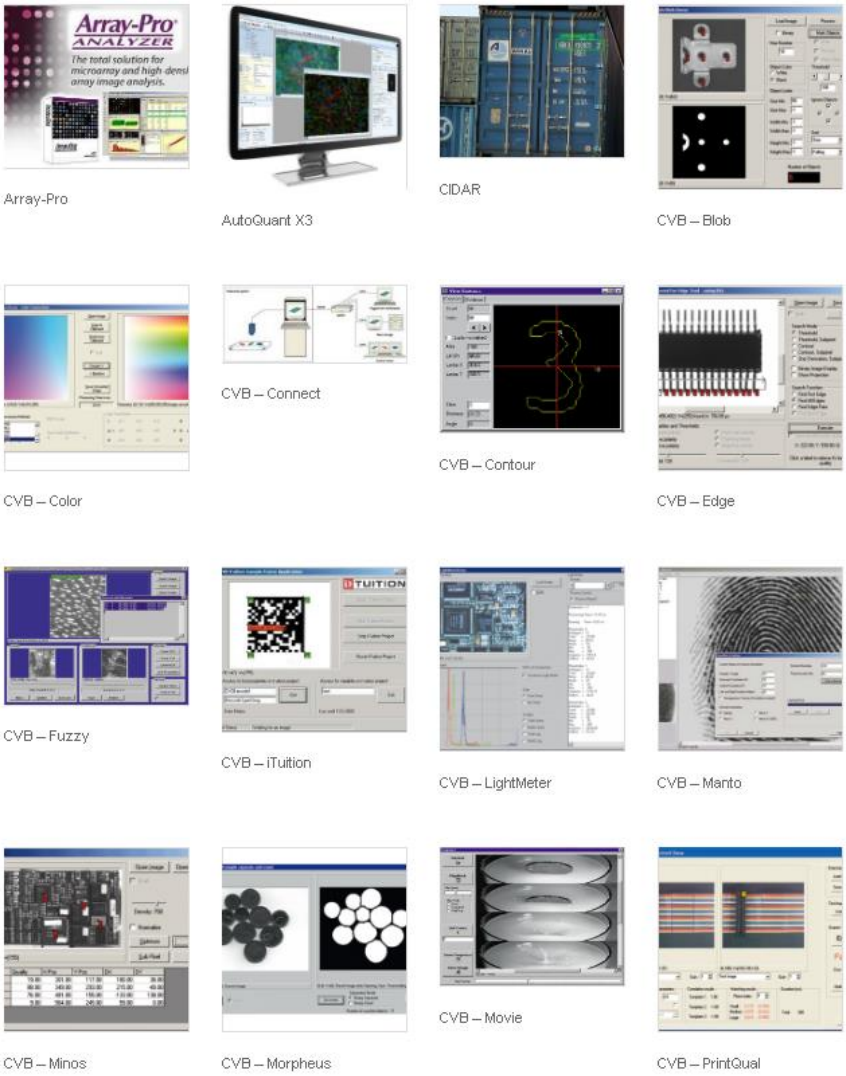


Figura 34 Ejemplo de software de visión artificial [18]

2.2. Inteligencia artificial

2.2.1. Introducción

Según Takeyas (2007) la IA es una rama de las ciencias computacionales encargada de estudiar modelos de cómputo capaces de realizar actividades propias de los seres humanos en base a dos de sus características primordiales: el razonamiento y la conducta [19].

Los expertos en ciencias de la computación Stuart Russell y Peter Norvig diferencian varios tipos de inteligencia artificial:

- Sistemas que piensan como humanos: automatizan actividades como la toma de decisiones, la resolución de problemas y el aprendizaje. Un ejemplo son las redes neuronales artificiales.

- Sistemas que actúan como humanos: se trata de computadoras que realizan tareas de forma similar a como lo hacen las personas. Es el caso de los robots.

- Sistemas que piensan racionalmente: intentan emular el pensamiento lógico racional de los humanos, es decir, se investiga cómo lograr que las máquinas puedan percibir, razonar y actuar en consecuencia. Los sistemas expertos se engloban en este grupo.

- Sistemas que actúan racionalmente: idealmente, son aquellos que tratan de imitar de manera racional el comportamiento humano, como los agentes inteligentes.

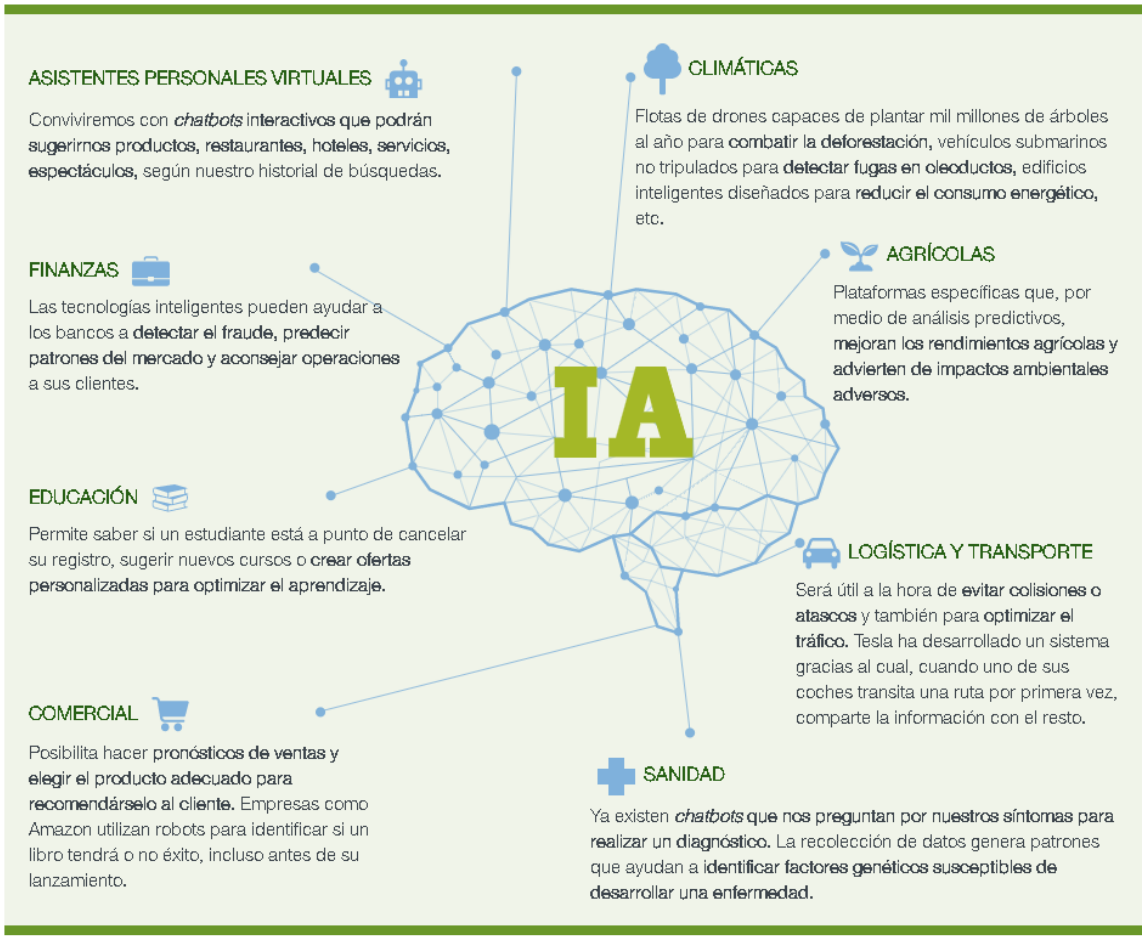


Figura 35 Principales aplicaciones de la inteligencia artificial [20]

2.2.2. Redes neuronales artificiales

La red neuronal artificial es un modelo de inteligencia artificial, inspirado en la red neuronal biológica. Se fundamenta en la utilización de una colección de nodos, llamados neuronas artificiales, conectados con otros nodos de diferente capa neuronal. En cada conexión entre neuronas artificiales se transmite información de forma unidireccional. La neurona receptora recibe las señales de una o más neuronas y las procesa para enviar una nueva señal procesada a las neuronas que están en una capa neuronal superior [21].

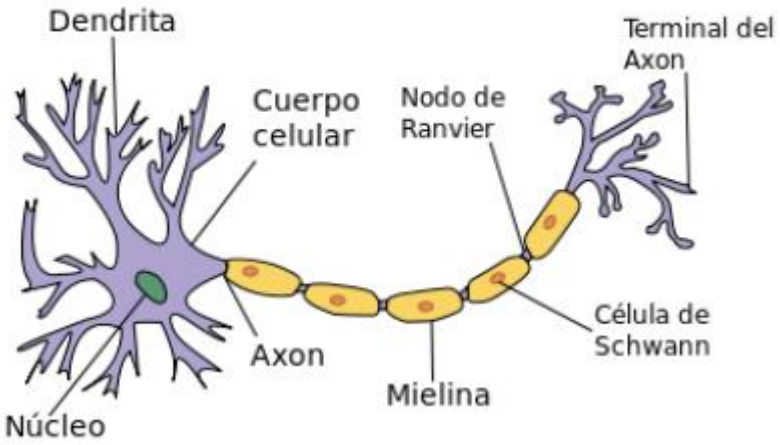


Figura 36 Morfología de una neurona biológica [22]

En la red neuronal artificial la señal transmitida entre conexiones es un número real y el procesado de dichas señales se calcula mediante una función suma no lineal de todas ellas. Para que el algoritmo pueda "aprender" es necesario la utilización y el ajuste de constantes modificadoras -también conocidos como pesos (*weights*)- sobre las neuronas o conexiones neuronales, que incrementen o disminuyan la señal procesada por la neurona.

Generalmente la red neuronal artificial está organizada en capas. Las diferentes capas llevan a cabo diferentes transformaciones sobre las señales de entrada. En una red con n capas la conformación es la siguiente: la primera capa de *input*, n-2 capas neuronales ocultas (*hidden neural layers*) y una última capa de clasificación. La señal viaja desde la primera capa hasta la última después de atravesar las diferentes capas ocultas [21].

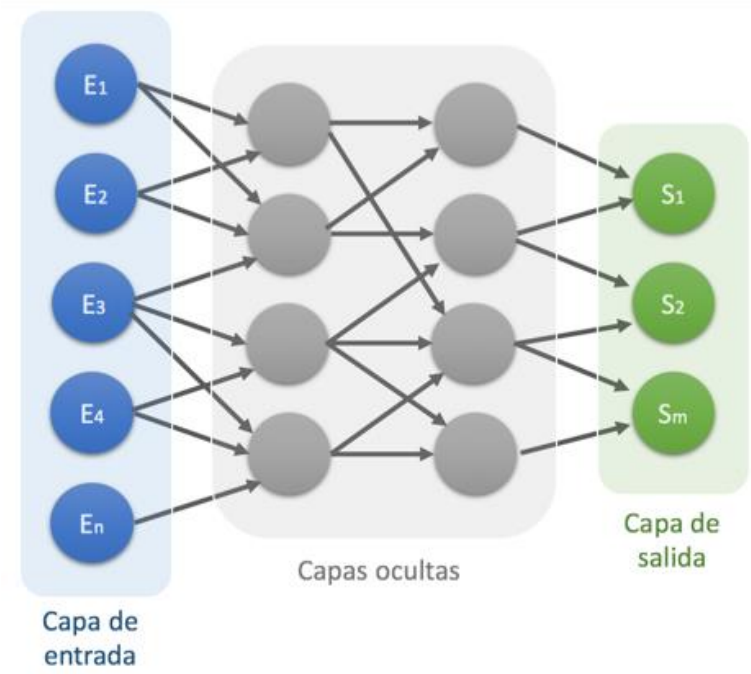


Figura 37 Red neuronal artificial [23]

Los primeros científicos que desarrollaron un modelo computacional de redes neuronales fueron Warren McCulloch y Walter Pitts [24] utilizando el cálculo lógico. Este modelo sentaba las bases de la utilización de redes neuronales en la inteligencia artificial. Más tarde, el psicólogo D. O. Hebb [25] describió la hipótesis de la plasticidad neuronal. Esta hipótesis sirvió de base para que en 1948 Alan Turing [26] desarrollara el concepto de máquinas desorganizadas en modelos computacionales. El primer modelo de máquina desorganizada -conocido como máquina de Turing tipo A- conectaba al azar redes de puertas lógicas NAND. El segundo modelo desarrollado –conocido como máquina de Turing Tipo B- podría ser creado tomando la máquina de Turing tipo A y reemplazando cada conexión inter-nodo con un modificador, éste permitiría a la máquina de tipo B llevar a cabo la "educación" de la red.

En 1975, el científico Paul J. Werbos desarrolló eficazmente un algoritmo de entrenamiento de las capas de redes neuronales mediante un entrenamiento hacia atrás (backpropagation). Éste distribuye el término error por las diferentes capas neuronales de manera reversa, mediante la modificación de los pesos de cada nodo [27].

A pesar de que la comunidad científica mostró un gran interés por las redes neuronales, éstas no fueron muy utilizadas debido a la complejidad y el

alto coste computacional que presentaban. Por ello, otros modelos de machine learning más simples, como: support vector machines, random forest o árboles de decisión, han tenido un mayor éxito en la clasificación de datos. Aunque, por otra parte, dichos algoritmos obtienen un pobre resultado en la clasificación de imágenes.

Los problemas intrínsecos al uso de redes neuronales artificiales han podido ser solventados debido tanto a una mejora en el hardware como a una optimización de los algoritmos. La mejora a nivel de hardware vino por la utilización de GPUs, dada la posibilidad de poder llevar a cabo multi-paralelización. La mejora de los algoritmos vino por el uso de modelos neuronales pre-entrenados, que han podido disminuir el tiempo de entrenamiento y el número de información necesaria para llevar a cabo el entrenamiento. Por otra parte, la utilización de la función max-pooling (función no- lineal que reduce la resolución) mejora la reproducibilidad de la red neuronal [21].

2.2.3. Redes neuronales convolucionales CNN

Entre los diferentes tipos de redes neuronales, la red neuronal convolucional (CNN) ha sido la que más éxito ha tenido debido al análisis de imágenes. La CNN es una subclase de red neuronal centrada en la estructura espacial de los inputs. La configuración de la CNN alterna capas convolucionales con capas max-pooling y por encima de ellas hay varias capas completa o escasamente conectadas (fully/sparsely connected layers) seguidas de una capa final de clasificación.

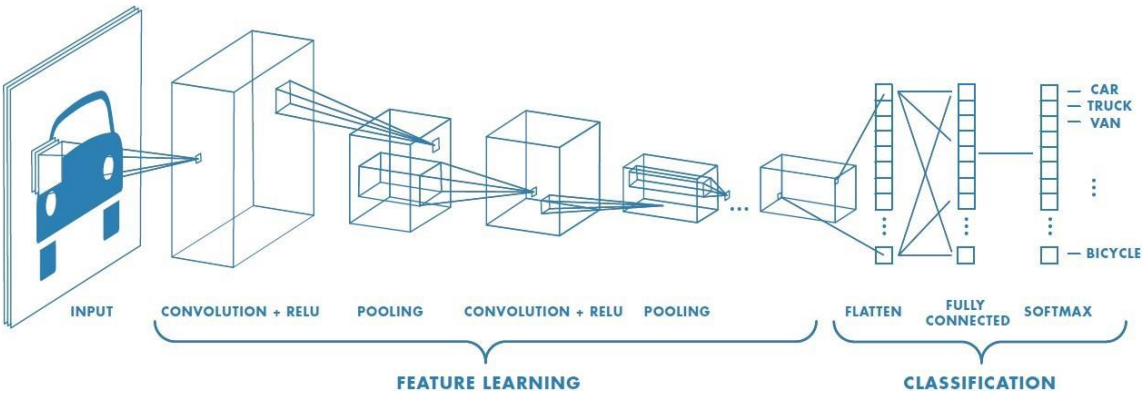


Figura 38 Arquitectura de una red convolucional [28]

2.2.3.1. Estructura

Las CNN son redes feedforward (prealimentadas) en las cuales la información fluye de manera unidireccional, desde los inputs hasta los outputs. Así como, las redes neuronales artificiales están inspiradas biológicamente, también lo están las CNNs. Las arquitecturas de las CNNs están basadas en el cortex visual del cerebro, que consiste en capas simples y complejas alternantes.

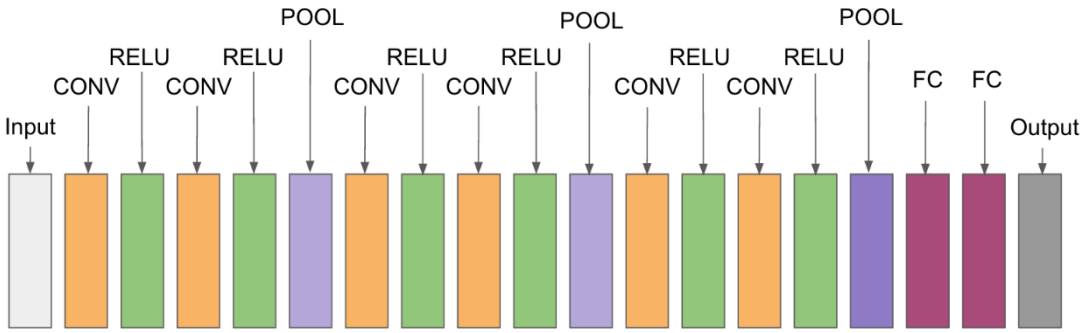


Figura 39 ejemplo de una estructura de la CNN [29]

La composición de las CNNs presenta muchas variantes. Sin embargo, en general éstas consisten en agrupaciones de módulos que contienen capas convolucionales y de reducción de resolución (pooling). En el caso del modelo de CNN más sencillo (modelo secuencial), los módulos se apilan unos encima de otros para formar un modelo profundo [30].

La capa convolucional está compuesta por varios kernels convolucionales los cuales son utilizados para calcular los mapas de características, con el fin de extraer información de las éstas [31]. Específicamente, cada neurona del mapa de características está conectada con otras neuronas de regiones vecinas de la siguiente capa. Dicha región es considerada como el campo receptivo de la neurona de la capa previa. El nuevo mapa de características puede ser obtenido primeramente por convolución del input con el kernel aprendido, para posteriormente aplicar una función no-lineal de activación. El mapa de características completo es obtenido mediante la utilización de diferentes kernels. El valor de cada característica en una determinada localización (i, j) en el k mapa de características de la capa l es calculado por:

$$z_{i,j,k}^{l} = w_k^{l^T} x_{i,j}^{l} + b_k^{l} \quad [3]$$

Donde $w_k^{l^T}$ y b_k^l representa al vector de pesos y la tendencia del filtro k de la capa 1 respectivamente. La región input centrada en la localización (i, j) de la capa 1 se representa como $x_{i,j}^l$.

Por otra parte, la función de activación introduce la no-linealidad al CNN, lo cual es deseable para detectar características no lineales. Siendo a(...) la función de activación. Las funciones de activación más utilizadas son: las sigmoideas, la tangente hiperbólica [32] y la unidad lineal de rectificación (ReLU) [33]. El vector de activación $a_{i,j,k}^l$ de la característica convolucional $z_{i,j,k}^l$ puede ser calculado como:

$$a_{i,j,k}^l = z(a_{i,j,k}^l) \quad [4]$$

Posteriormente, se suele utilizar una capa de *pooling* para disminuir la variabilidad (frente a la distorsión y translación de los datos de entrada) reduciendo la resolución espacial del mapa de características. Habitualmente el procedimiento de *pooling* es realizado entre dos capas convolucionales. El resultado de aplicar la función pool al valor de activación es:

$$y_{i,j,k}^{l+1} = pool(a_{m,n,k}^l), \forall (m,n) \in \mathcal{R}_{ij} \quad [5]$$

Donde $\mathcal{R}_{ij}$ representa a la vecindad local alrededor de la localización (i,j). Las funciones *pooling* más utilizadas son *average pooling* [34] y *max-pooling* [35] [19]. En especial, la capa de agregación por *max-pooling* mejora sustancialmente la generalización -y por ende la robustez del modelo-mediante la propagación del máximo valor del campo receptivo a la siguiente capa.

Generalmente, los kernels convolucionales de las capas más bajas están diseñados para detectar características básicas como curvas y límites, mientras que los kernels de capas más altas se centran en detectar características más abstractas.

Después de varias capas convolucionales y de reducción de resolución, puede haber una o más capas completamente conectadas con el fin de interpretar las características extraídas y de llevar a cabo un alto nivel de razonamiento computacional [37].

La última capa de las CNNs es una capa de output. Si la finalidad del modelo es la clasificación, se llevará a cabo una operación softmax [38], es decir se comprime un vector K-dimensional, z, en un vector K-dimensional σ(z) en el rango de (0,1). Siendo ϴ todos los parámetros de la CNN (vectores de pesos y la constante de tendencia). El parámetro óptimo para una tarea específica puede ser obtenido por la minimización de la función definida para cada tarea. Si se tienen N relaciones input-output $\{(x^{(n)}, y^{(n)}); n \in [1,\dots,N]\}$ donde $x^{(n)}$ son los n datos de entrada e $y^{(n)}$ es la correspondiente etiqueta para dichos datos, el $o^{(n)}$ es el output de la CNN. Por tanto, la función de la CNN puede ser calculada con [21]:

$$L = \frac{1}{N}\sum_{n=1}^{N} l(\theta; y^{(n)}, o^{(n)}) \quad [6]$$

El entrenamiento de la CNN es un problema global de optimización. El mejor ajuste de los parámetros de la CNN puede ser realizado por minimización de la función loss.

Por otra parte, si la capa final es un clasificador softmax, la probabilidad de pertenecer a cada una de las clases viene dado por:

$$y_i = \frac{\exp(-z_i)}{\sum_{j=i}^{k} \exp(z_j)} \quad [7]$$

2.2.3.2. Modelos de CNN.

- **Modelo LeNet-5**

Es un modelo de 7 capas desarrollado por LeCun et al [36] en 1998, que clasifica dígitos, se aplicó por varios bancos para reconocer números escritos a mano en los cheques digitalizados en imágenes en escala de grises con una resolución de 32x32 pixeles. La habilidad para procesar imágenes con mayor resolución requiere mayores capas de convolución, por eso este modelo está restringido por la disponibilidad de recursos computacionales.

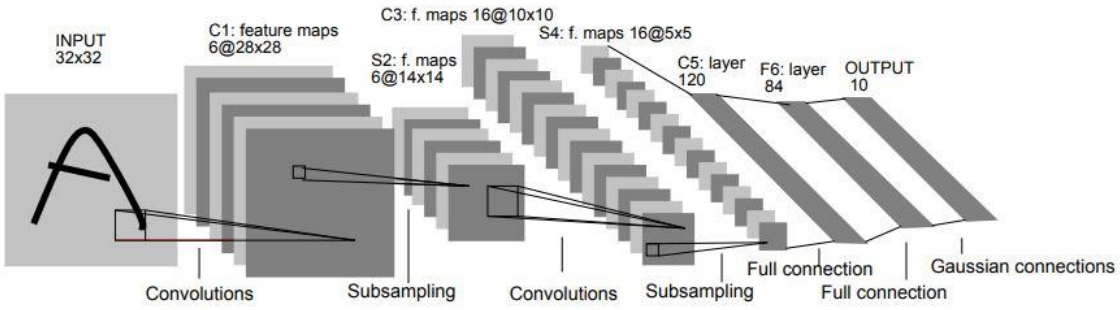


Figura 40 Modelo LeNet-5 [36]

- **Modelo AlexNet.**

El campo del aprendizaje profundo resurgió en el 2006 [39] debido al gran éxito obtenido en una variedad de tareas, como: la clasificación, reconocimiento de imágenes y objetos [40] [41], reconocimiento facial [42] y segmentación de imágenes [43]. A pesar de éstos avances, las CNNs no habían sido muy utilizadas ni en el ámbito de la visión computacional ni el del aprendizaje profundo. Esto cambió después del congreso ILSVRC en el 2012, cuando una CNN desarrollada por Krizhevsky et al [44]. (AlexNet) consiguió el mejor resultado en la clasificación de imágenes de conjunto de imágenes de ImageNet. Este trabajo revolucionó el campo de la visión computacional y desde entonces las CNNs han sido la arquitectura que mejores resultados ha obtenido en la mayoría de tareas visuales y en particular en la clasificación de imágenes.

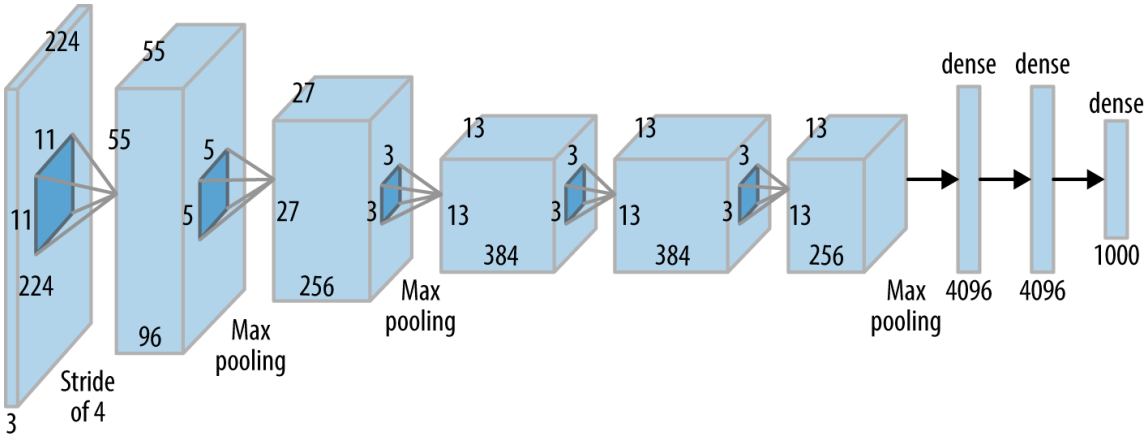


Figura 41 Modelo AlexNet [45]

El éxito del modelo de Krizhevsky fue debido a la utilización de métodos novedosos. El modelo consiste de cinco capas convolucionales: tres de las cuales

están seguidas de capas de max-pooling y tres capas completamente conectadas. Como función de activación fue utilizada la función ReLU, que permite un rápido entrenamiento. Para no llegar al overfitting los autores emplearon la técnica dropout [46], que consiste en silenciar unas determinadas neuronas al azar, antes de que cada caso sea presentado a la CNN en la fase de entrenamiento. De esta manera, se previenen co-adapataciones superfluas con los datos de entrenamiento. Por otra parte, el overfitting también fue reducido por un aumento artificial de los datos de entrada, modificando las imágenes input mediante la aplicación de translaciones, reflexiones y alteraciones de intensidades.

- **Modelo ZFNet.**

Ha sido también ganadora del ILSVRC 2013. Se ha conseguido afinando los hiper-parametros del modelo AlexNet mientras mantiene la misma estructura de antes.

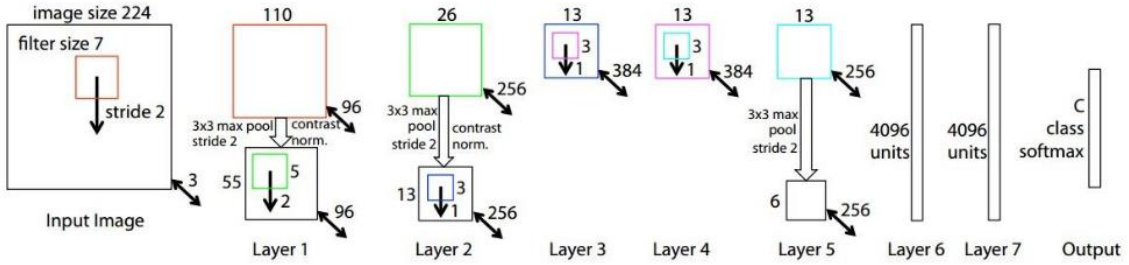


Figura 42 Modelo ZFNet [47]

- **Modelo VGGNet.**

El modelo Visual Geometry Group (VGG) desarrollado por Simonyan y Zisserman [48], consiguió el segundo mejor resultado en la clasificación de imágenes del ILSVRC en 2014. El modelo (VGG-16) utilizaba una red convolucional profunda, que consistía en 16 capas apiladas secuencialmente. Esto fue posible utilizando un pequeño filtro convolucional (3x3) por toda la red, minimizando de este modo los campos receptivos. El resultado es que había menos parámetros y más no-linealidades, facilitando la discriminación de la función de decisión y por tanto haciendo más fácil su entrenamiento.

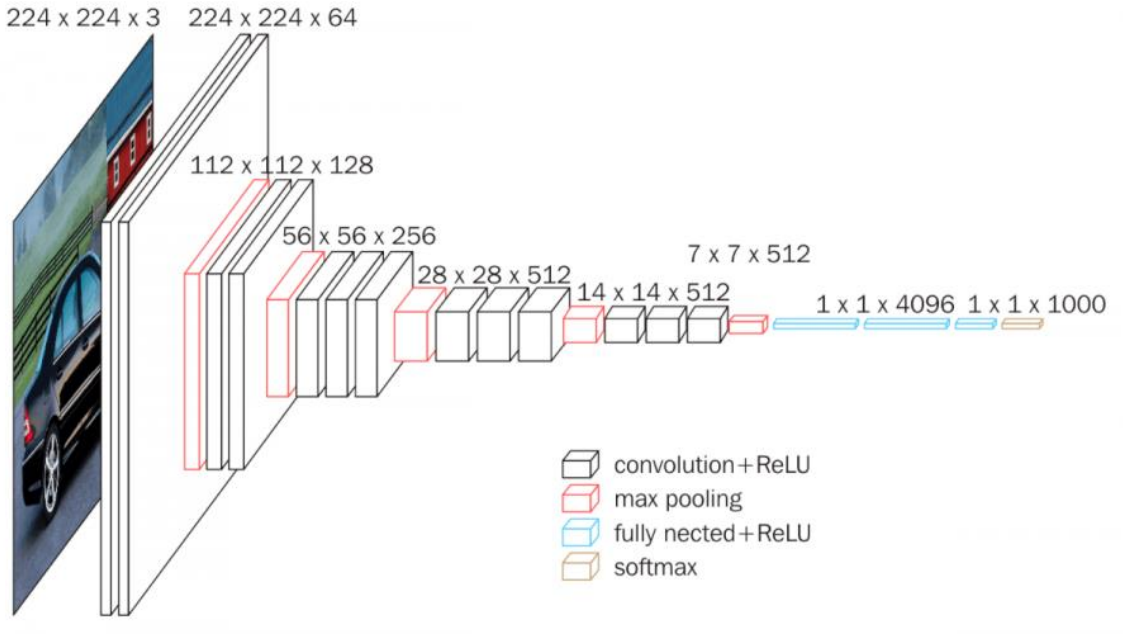


Figura 43 Modelo VGG16 [49]

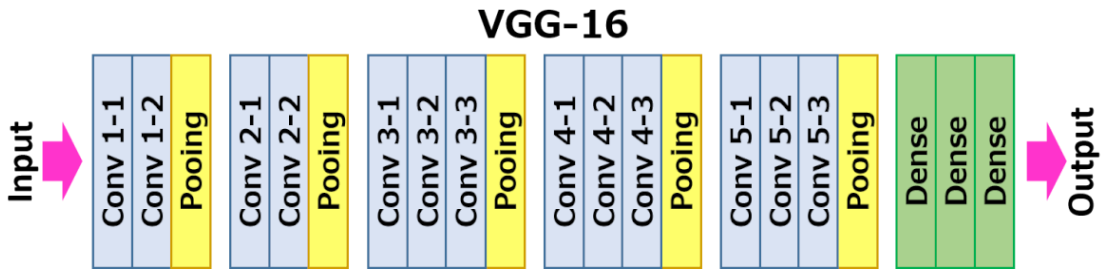


Figura 44 Estructura VGG16 [50]

- **Modelo GoogleNet.**

El modelo que ganó el concurso ILSVRC del 2014 fue GoogLeNet [51], desarrollado por Szeged y et al. Introduciendo una novedosa arquitectura: el módulo Inception, que se basa en la utilización de convoluciones con diferente tamaño de filtro de forma paralela para posteriormente concatenar los resultados. Dicho modelo utilizaba 22 capas. Para superar el alto coste de utilizar un gran número de parámetros –ya que la red es más propensa al overfitting -se diseñó una nueva arquitectura de la red basada en los principios de Hebbian, que permitía cambiar de una red convolucional totalmente conectada a una escasamente conectada. Específicamente, su arquitectura utilizaba convoluciones que servían como bloques de reducción de la dimensionalidad, para posteriormente llevar a cabo convoluciones y computacionalmente más costosas. De esta manera, se podía incrementar la profundidad y la

anchura de la red sin aumentar considerablemente el coste computacional.

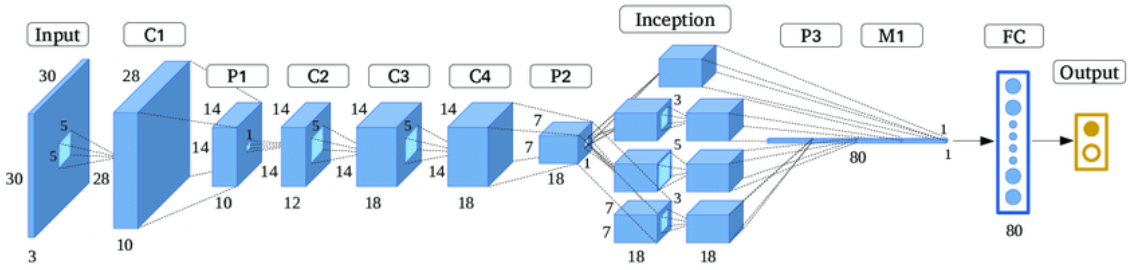


Figura 45 Estructura GoogleNet [51]

- **Modelo ResNet.**

Este modelo Nació de una simple observación: "¿por qué agregar más capas a las redes neuronales profundas no mejora la precisión, o incluso empeora" [52].

Los desarrolladores de ResNet han devuelto este problema a la hipótesis de que las asignaciones directas son difíciles de entrenar. Por lo tanto, propusieron un remedio: en lugar de tratar de aprender a partir de mapeos subyacentes de x y H (x), es posible aprender la diferencia entre los dos, que es el "residuo" y, posteriormente, ajustar el último a la entrada.

Supongamos que el residuo es F (x) = H (x) -x. Ahora nuestra red intenta aprender de F (x) + x. Esto dio origen a los famosos bloques de ResNet (red residual):

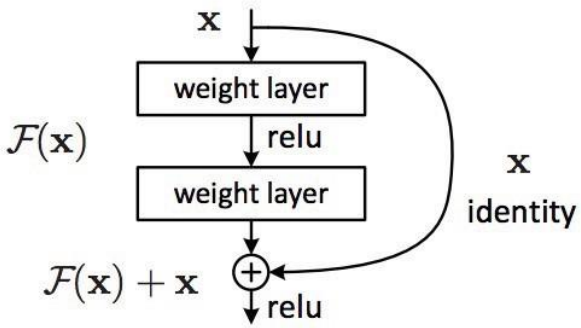


Figura 46 Bloque de ResNet [52]

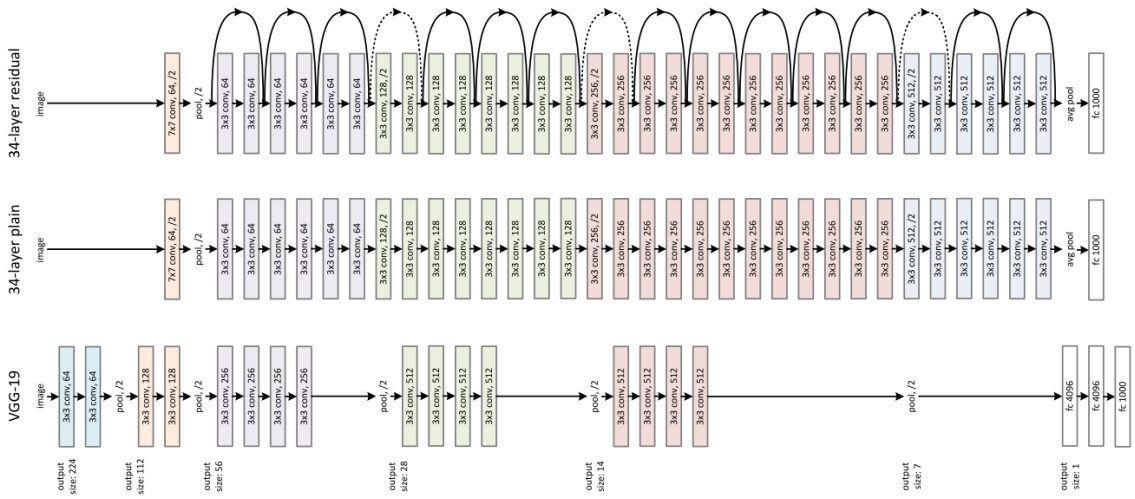


Figura 47 Estructura de una ResNet [52]

- **Comparativa.**

Año	CNN	Desarrollado por	Nº parámetros
1998	LeNet	Yann LeCun et al	60 mil
2012	AlexNet	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	60 millones
2013	ZFNet	Matthew Zeiler y Rob Fergus	
2014	GoogleNet	Szeged y et al	4 millones
2014	VGG16	Simonyan, Zisserman	138 millones
2015	ResNet	Kaiming He	

Tabla 5 Resumen de los principales modelos de CNN

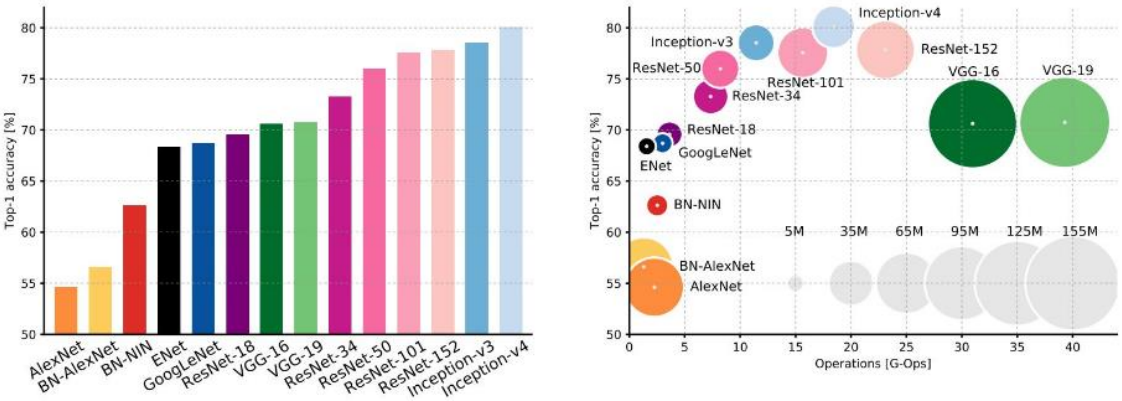


Figura 48 Comparación entre diferentes modelos de CNN [53]

2.2.4. Aprendizaje transferido

El aprendizaje transferido se usa muy a menudo en aplicaciones de Deep Learning. Se aprovecha una red neuronal pre-entrenada como un punto de partida para aprender una nueva tarea [54]. Afinar una red neuronal con el aprendizaje transferido es mucho más rápido y fácil que entrenar una red neuronal con unos valores de pesos arbitrarios. Se pueden transferir las características aprendidas de una red neuronal a una nueva tarea usando menos cantidad de imágenes de entrada ya que la red ha aprendido ya a reconocer características más generales como líneas, bordes o formas.

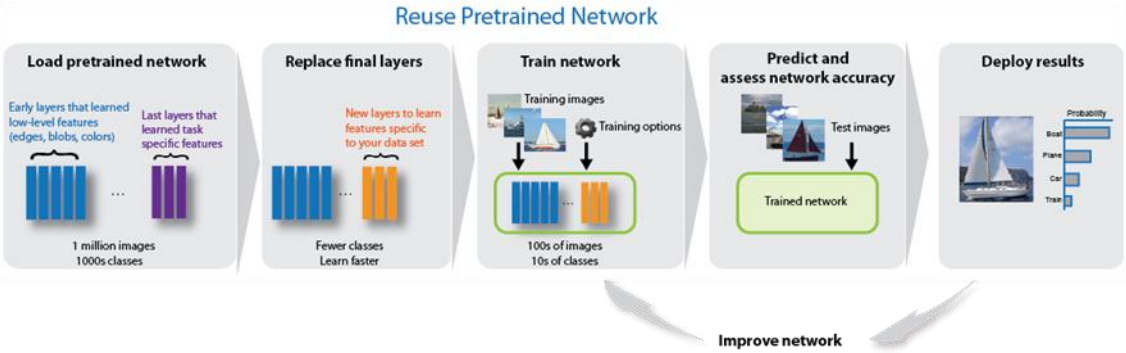


Figura 49 Aprendizaje transferido [55]

2.2.5. Segmentación semántica

A diferencia de los sistemas de detección y reconocimiento de objetos tradicionales, que dan como resultado la ventana rectangular donde se ha detectado un objeto (Blobs) y son evaluados por la precisión de estas ventanas, los sistemas de segmentación semántica tienen como objetivo delimitar precisamente los objetos de las distintas categorías a nivel de píxel, dando como resultado cualquier forma arbitraria [56].

Figura 50 Localización de objetos vs segmentación semántica [56]

Hasta ahora los modelos descritos fueron ideados para la clasificación de imágenes en categorías, realizando una predicción global de las clases de elementos que están presentes en el input. Es decir, pueden predecir el contenido global pero no el número ni la posición. Las siguientes evoluciones lógicas en la inferencia de una imagen son mediante, localización de objetos, segmentación semántica y segmentación semántica de instancias [57]. La localización de objetos obtiene información de las clases presentes y de la localización de las mismas. La segmentación semántica realiza inferencias por pixel de las diferentes clases posibles. Por otra parte, la segmentación semántica de instancias puede segmentar elementos que pertenecen a la misma clase y que están pegados entre sí.

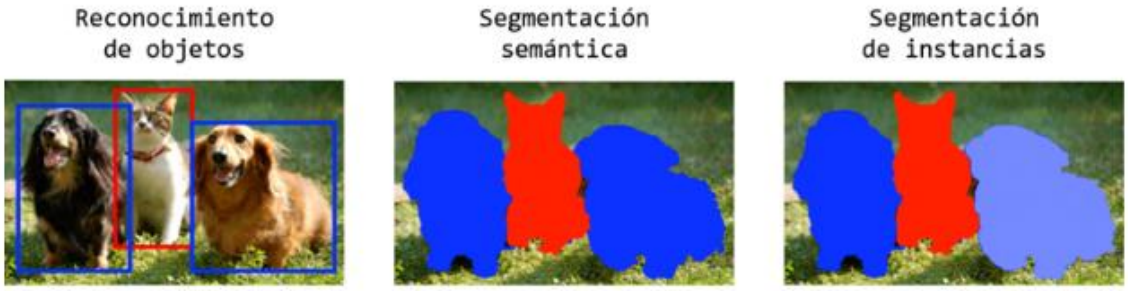


Figura 51 los 3 tipos de segmentación [58]

El precursor de la segmentación semántica fue el modelo de Red Neuronal Completa (FCN) desarrollado por Long et al [59]. La aproximación del modelo fue mediante el uso de CNNs ya conocidas y ampliamente usadas en la clasificación de imágenes. Para ello, transformaron los modelos de clasificación -AlexNet, VGG, GoogLeNety ResNet–en modelos completamente convolucionales remplazando las últimas capas completamente conectadas con capas convolucionales cuyo output son mapas espaciales en vez de una puntuación para la clasificación de clases. Estos mapas de características son aumentados de resolución mediante el uso de convoluciones de pasos fraccionados (también conocido como de convolución) para producir un output con un aumento de resolución, en el que cada pixel es etiquetado según la clase inferida. Este modelo es considerado un hito en el aprendizaje profundo ya que muestra como las CNNs pueden ser entrenadas de forma end-to-end.

A pesar de la efectividad y flexibilidad del modelo FCN presenta

varias carencias como son: la invariancia espacial que presenta no tiene en cuenta información global que forma parte del contexto, no separa instancias similares que no presenta un discontinuo entre éstas y el entrenamiento de estos modelos es computacionalmente muy costosos (aproximadamente entre uno y tres días utilizando un equipo de altas prestaciones como es la GPU NVIDIA Tesla K40c).

A partir de la arquitectura FCN se han descrito otras variantes que han tenido bastante éxito. En general, todos ellas utilizan una red pre-entrenada para la clasificación eliminando las capas completamente conectadas, a esta parte de la arquitectura se la conoce como "codificador": que produce mapas de características o imágenes con baja resolución. Por otra parte, la parte específica de cada modelo es el "decodificador", cuya función es mapear las imágenes de baja resolución para crear imágenes de mayor resolución con predicciones de clase por pixel.

Uno de los modelos que más éxito ha tenido es SegNet [60]. En el modelo SegNet el decodificador está compuesto por un conjunto de capas de aumento de resolución y de capas convolucionales que finalmente están seguidas por un clasificador softmax para predecir las etiquetas por pixel para un output que tiene la misma resolución que la imagen input. Cada aumento de resolución de la parte decodificadora corresponde con la función max-pooling utilizada en la parte codificadora.

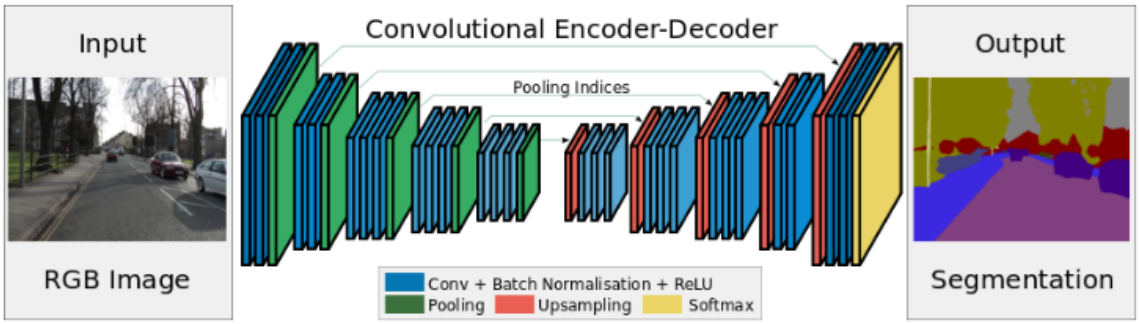


Figura 52 Arquitectura SegNet [60]

Generalmente, el proceso de entrenamiento de una red de segmentación semántica para clasificar imágenes consta de estos cuatro pasos:

- Analizar un conjunto de imágenes con píxeles etiquetados.
- Crear una red de segmentación semántica.
- Entrenar la red para clasificar imágenes en categorías de píxeles.
- Evaluar la precisión de la red.

Capítulo 3

3. Adquisición de imágenes

3.1. Introducción

La primera fase de este trabajo consta de la construcción de un sistema de visión artificial para la adquisición de imágenes que posteriormente formarán parte de la base de datos del algoritmo de segmentación. Se va a proceder en primer lugar al estudio de la técnica de iluminación adecuada para la aplicación. Después, se crea una *Lighing-Box* para sostener los elementos del sistema de visión y aislarlo del entorno. Y finalmente, se ajustará la configuración de la cámara para mejorar el contraste.

3.2. Técnicas de iluminación.

El objeto que vamos a analizar con este sistema se trata de una plancha de cartón ondulado. Esta plancha contiene ciertos cortes y pliegues que le permiten luego ser transformada en la caja mediante un proceso de formado y plegado.

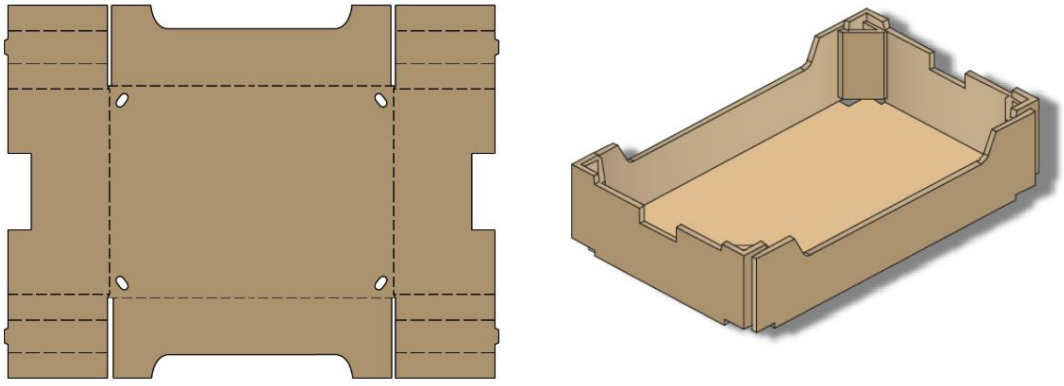


Figura 53 Ilustración de una plancha de cartón ondulado antes y después de la transformación [61]

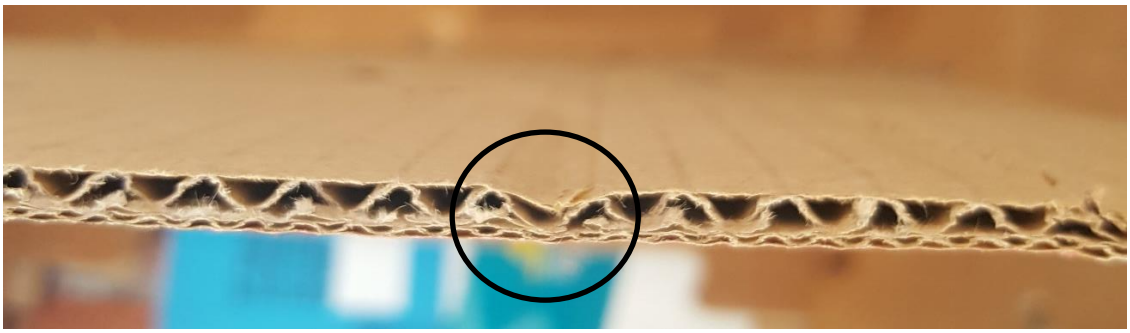


Figura 54 vista seccionada de la profundidad de un pliegue

Tomando como referencia la caja formada, cada zona de la plancha recibe un nombre que la describe según la función que tiene en la caja. Estas zonas están delimitadas mediante una combinación de cortes y pliegues determinados mediante un diseño. Aunque los detalles de estos recortes cambian de un diseño a otro, la forma general de cada característica viene definida mediante varios aspectos comunes que le permiten ser reconocida.

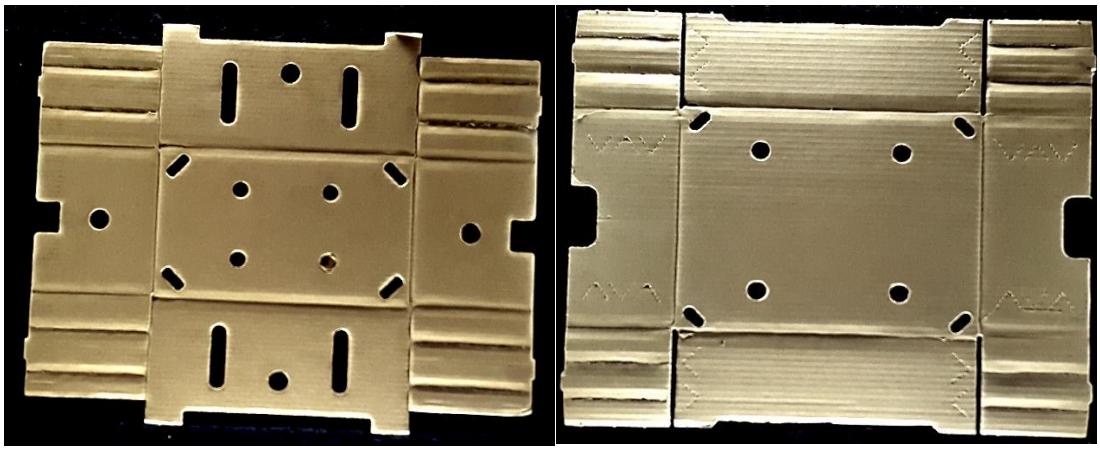


Figura 55 Planchas del mismo modelo y diferente diseño

A continuación, se van a comparar dos de las técnicas de iluminación más comunes en visión artificial para valorar sus ventajas y desventajas a la hora de aplicarlas sobre los objetos seleccionados.

- **Iluminación frontal directa.**

Es la técnica de iluminación más común. En la iluminación frontal, la cámara se posiciona mirando al objeto en la misma dirección de la luz y la cámara recibe la luz reflejada del objeto.

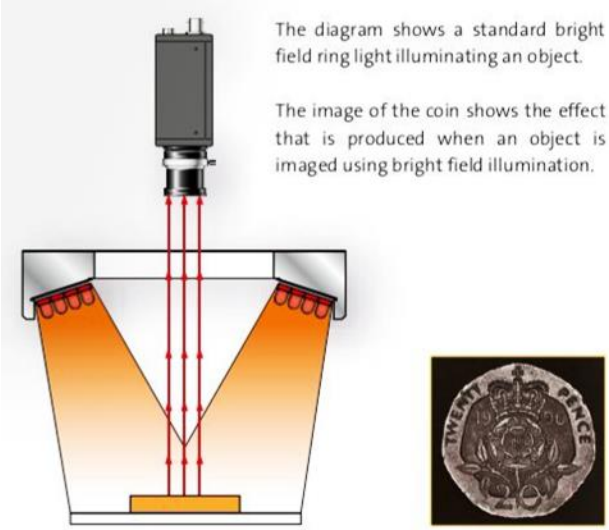


Figura 56 Ejemplo de una iluminación frontal directa [11]

Este tipo de iluminación se consigue mediante anillos de luz, iluminadores puntuales, de área y lineales. Esta iluminación es especialmente útil en superficies con pocos reflejos [62].

Para aplicar esta técnica, se ha utilizado una fuente de iluminación formada por cuatro tiras de LED posicionados a cada lado a una altura de 1 metro de la superficie de la plancha. La cámara se ha situado en el frontal de la plancha a la misma altura para evitar los reflejos directos de la iluminación. Los parámetros del sistema de iluminación utilizado se detallarán más adelante.

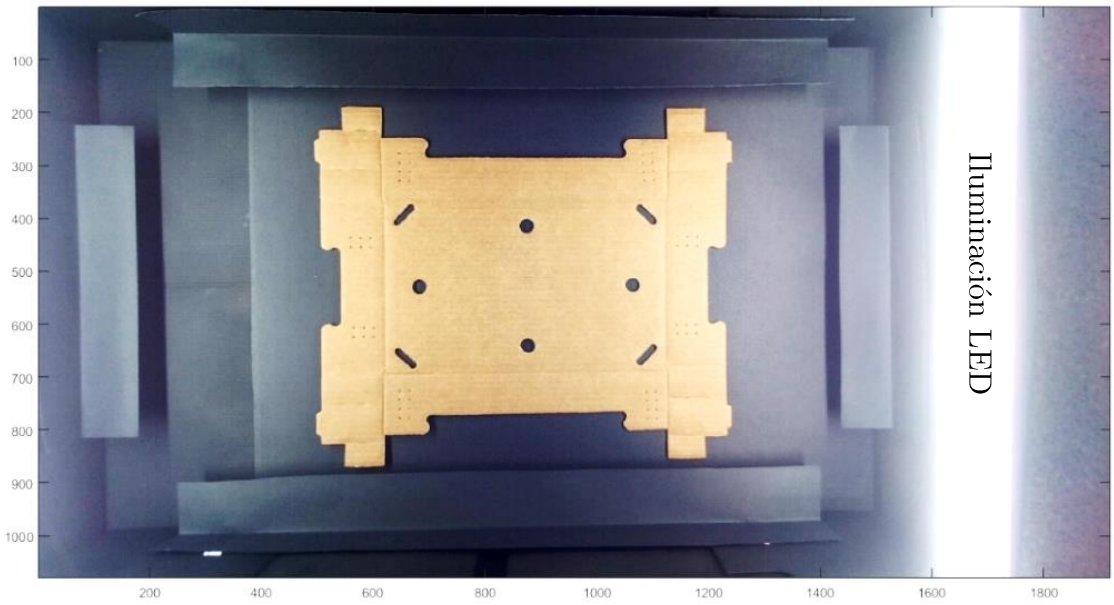


Figura 57 Sistema con iluminación frontal directa

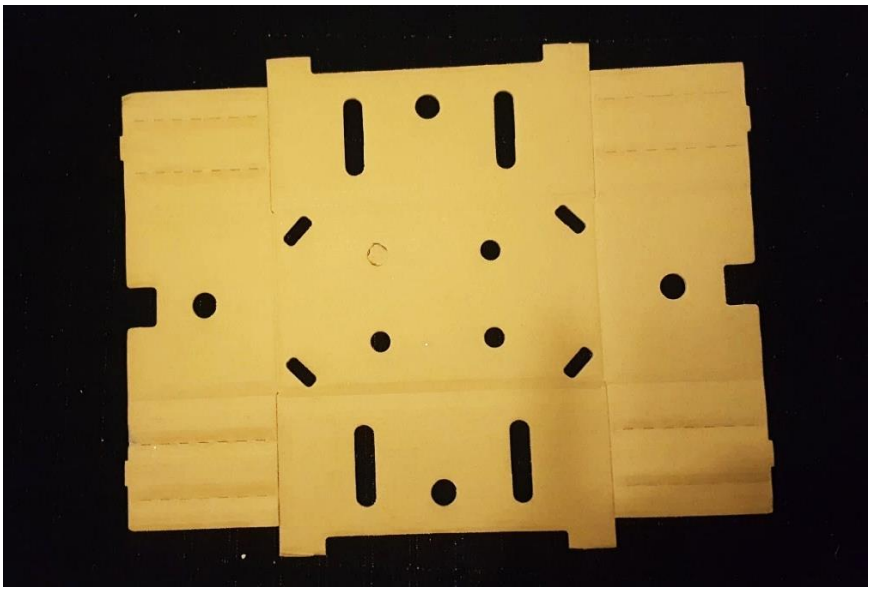


Figura 58 Imagen de la plancha aplicando una iluminación frontal directa

Esta técnica ha destacado de forma eficaz los contornos de la plancha definidos por los recortes. Sin embargo, ha reducido considerablemente el contraste de la mayoría de las características internas definidas por los pliegues. Esto es debido a que la iluminación frontal ha eliminado gran parte de la sombra en los pliegues, lo que ha provocado la pérdida del efecto de profundidad. Esta técnica puede servir para analizar únicamente el contorno externo de las planchas, cosa que no nos sería de mucha utilidad sabiendo que existen diseños de diferentes modelos de planchas que tienen contornos externos similares.

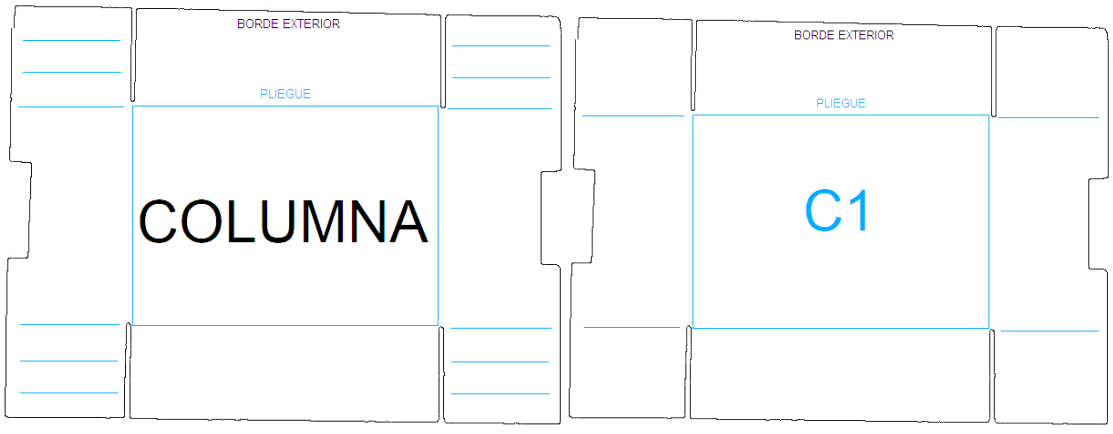


Figura 59 Comparación de contornos exteriores de los modelos de planchas Columna y C1

- **Iluminación de campo oscuro (Dark Field).**

La luz se aplica teniendo en cuenta el principio de "ángulo de incidencia = ángulo de reflexión". Cuando el haz de la luz forma un ángulo muy pequeño respecto a la superficie plana del objeto, la luz se refleja fuera del campo de visión de la cámara lo que provoca que la superficie se ve oscura.

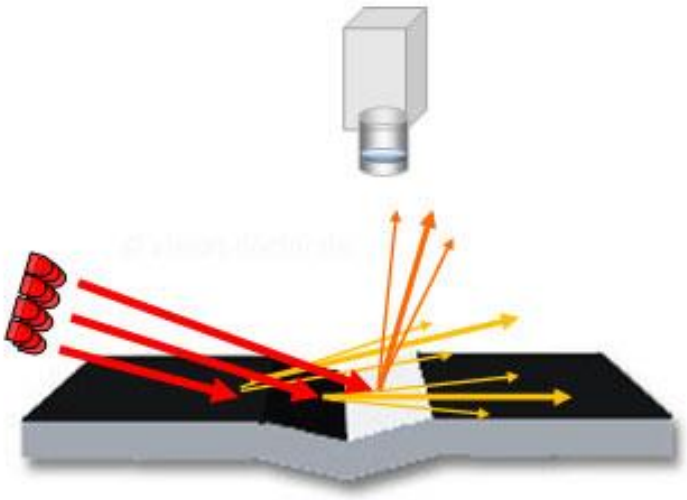


Figura 60 diferentes superficies bajo una iluminación de campo oscuro [10]

Las caras inclinadas, los bordes, las ranuras, los relieves y las elevaciones interfieren con el haz de luz. En estos casos, la luz se refleja hacia la cámara, lo que hace que estos objetos aparecen brillantes en la imagen.

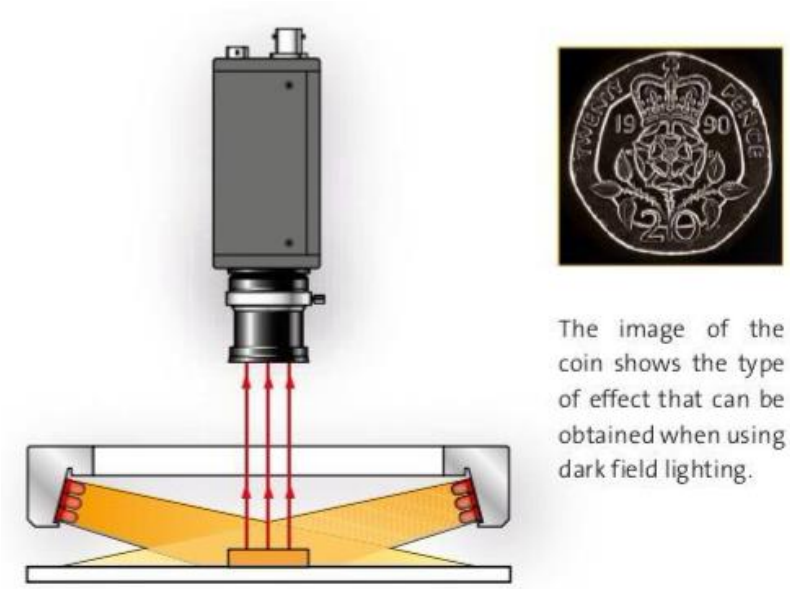


Figura 61 Iluminación de campo oscuro [11]

Para aplicar esta técnica de iluminación, se ha colocado la fuente de iluminación a una altura de 20 mm de la superficie de la plancha. Debido a que las planchas tienen pliegues y cortes en las dos direcciones. Se han utilizado lámparas LED para iluminarlas desde los cuatro laterales para destacar todas las características.

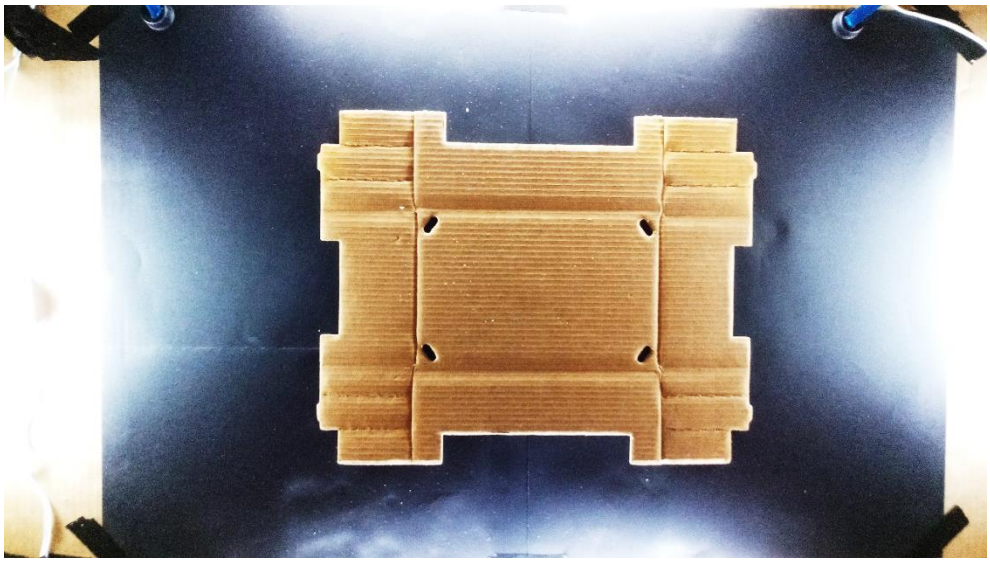


Figura 62 distribución de la luz alrededor de la plancha

No se puede usar una luz muy direccional ni un sistema pinole para dirigir la luz al plano superior de la plancha porque además de que el espesor de las planchas cambia de un diseño a otro, la superficie de estas suele sufrir algunas deformaciones en el proceso de fabricación y transporte.

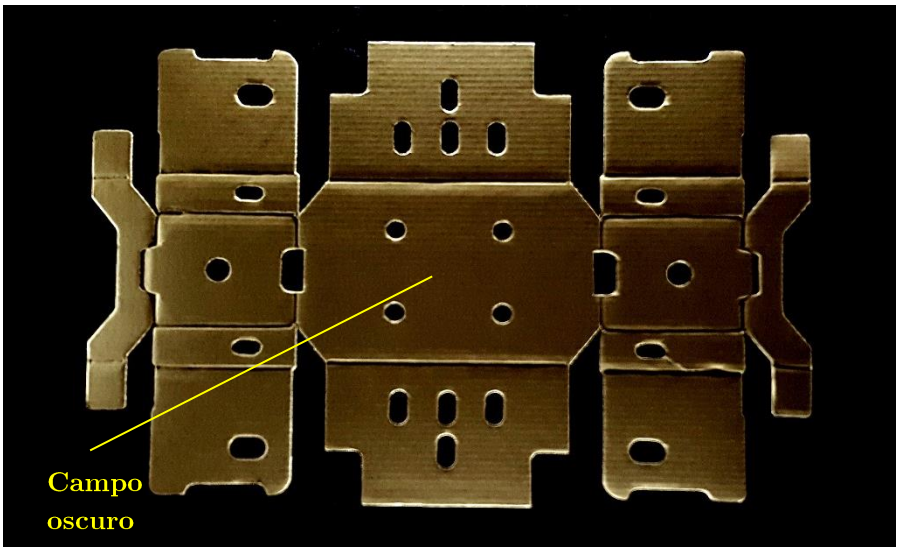


Figura 63 Imagen de la plancha aplicando una iluminación de campo oscuro

Con esta técnica de iluminación, se nota una mejora considerable en el contraste de los pliegues internos de la plancha. Sin embargo, pueden aparecer reflejos fuertes en los tramos más próximos al sistema de iluminación en planchas de mayor tamaño como consecuencia de disminuir la altura de las luces. Este problema se puede evitar alejando el sistema de iluminación lo suficiente para abarcar todos los tamaños de planchas considerados.

A continuación, se muestra una tabla de comparación entre los dos sistemas de iluminación utilizados.

Parámetro/ técnica de iluminación	Frontal directa	Dark Field
Contraste de contornos externos	Bueno	Bueno
Resalte de pliegues internos	Pobre	Bueno
Reflejos fuertes en superficie	Bueno	Bueno

Tabla 6 Comparación entre las técnicas de iluminación

3.3. Caja de iluminación (Lighting-box).

Después de elegir la técnica de iluminación que más se adapta a los requerimientos de la aplicación, se procede a la construcción de una caja de iluminación para la adquisición de imágenes. A continuación, se describe con mayor detalle este proceso.

- **Caja.**

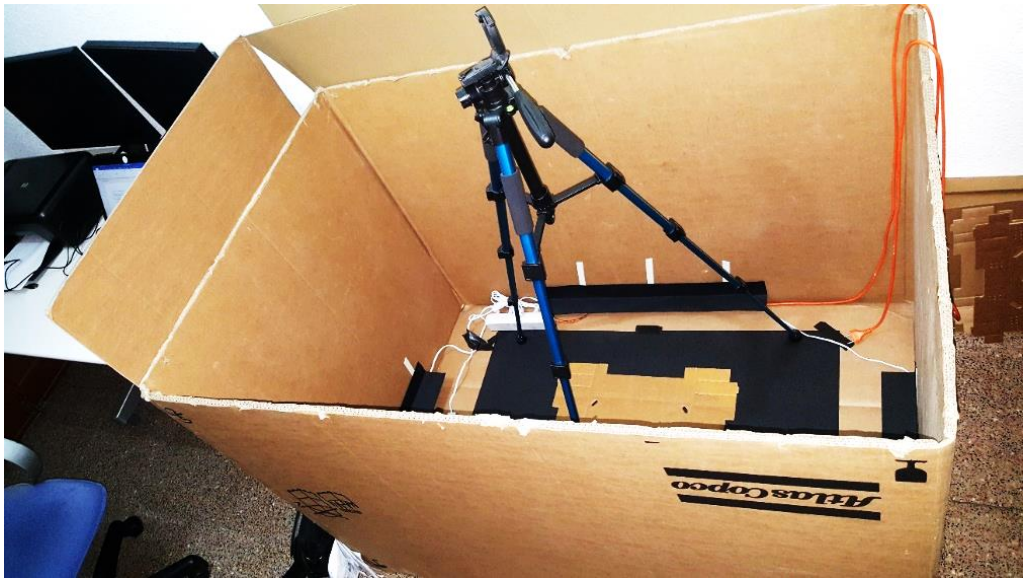


Figura 64 Caja utilizada

Para sostener el sistema y aislarlo de la iluminación exterior, se ha utilizado una caja de cartón como una base. El tamaño de la caja elegida viene dado por la dimensión de plancha máxima y el margen que hay que dejar entre la iluminación y la plancha para evitar reflejos fuertes en su superficie.

- **Iluminación.**

Para distribuir la luz de forma uniforme tanto en el largo como en el ancho de la plancha, se ha utilizado un total de 4 tiras de LED de 6000k y de 600 y 300 milímetros de largo respectivamente.

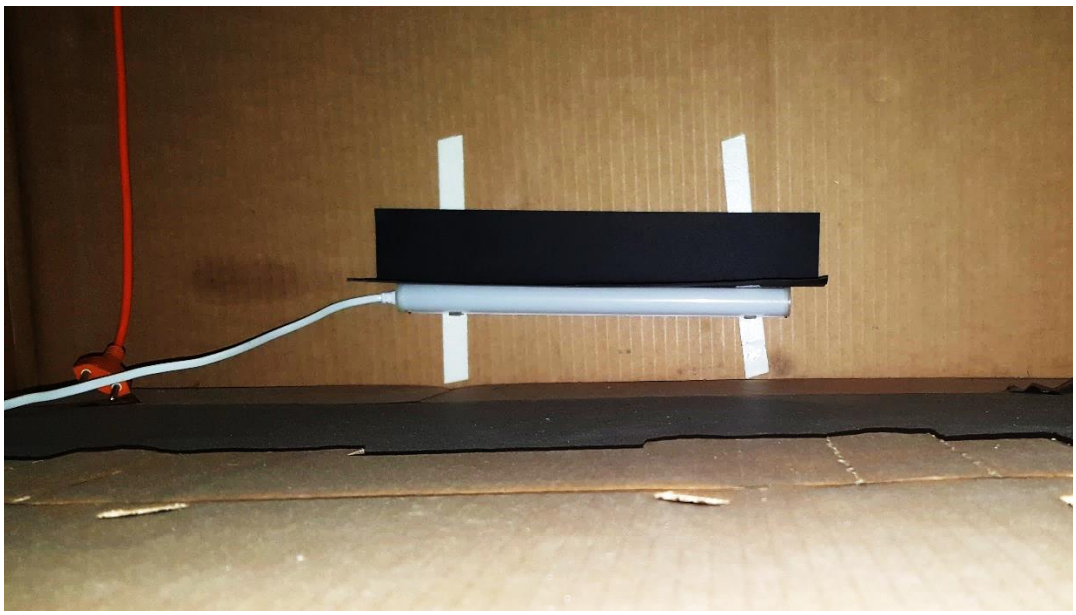


Figura 65 lámpara instalada en el ancho

Figura 66 Lámpara de 600 mm con tiras de LED instalada en el largo

Las lámparas han sido fijadas en los cuatro lados de la caja a una altura de 20 milímetros de la superficie de la plancha. Lo que se consigue con esta medida es el contraste óptimo entre el color negro de la sombra en el fondo de los pliegues y el color blanco del reflejo sobre el borde del pliegue.

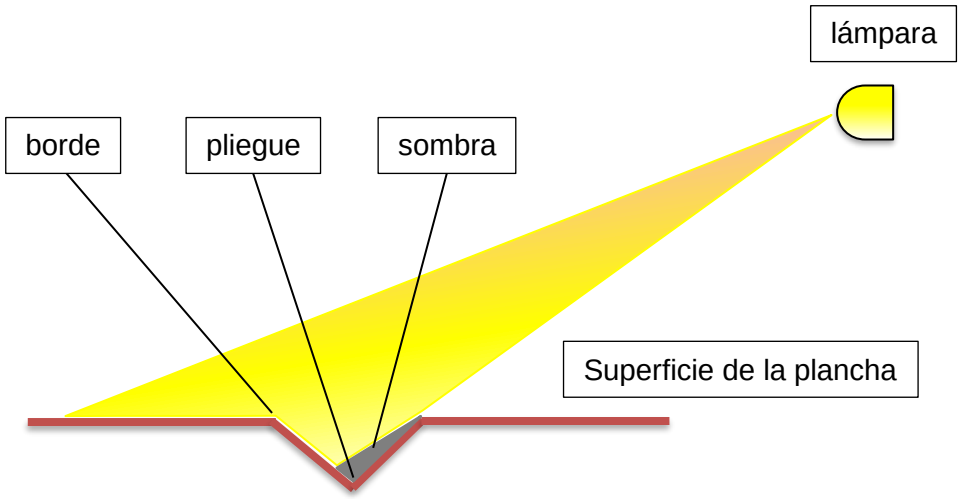


Figura 67 Altura de lámpara demasiado alta

Si se elevan demasiado las lámparas, la sombra en los pliegues pierde intensidad y la superficie tendrá más luz, lo que reduce el contraste (Figura 67). Por otro lado, si la altura de las lámparas es demasiado baja, la superficie es más oscura y la sombra en los pliegues se extiende mucho. Debido a esto, los contornos de los elementos de la plancha no quedan bien delimitados (Figura 68).

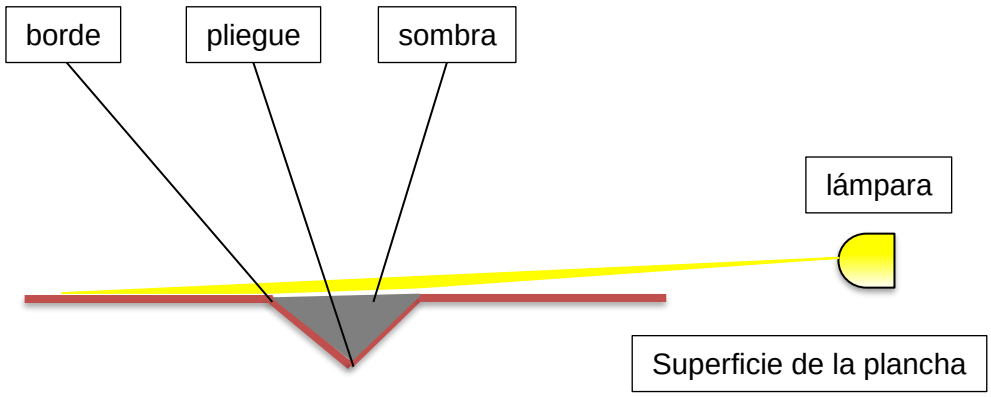


Figura 68 Altura de lámpara demasiado baja

- **Fondo.**

Para mejorar el contraste del contorno externo y los recortes de la plancha, se ha utilizado un papel de color negro mate como fondo.

- **Sombrillas.**

Una de las consecuencias de utilizar la técnica de iluminación dark-field en esta aplicación es el reflejo directo de la luz en la cámara (Figura 62). Para solucionar este problema, se ha tapado la parte superior de las lámparas con sombrillas fabricadas con trozos de papel de color negro (Figuras 65, 69).

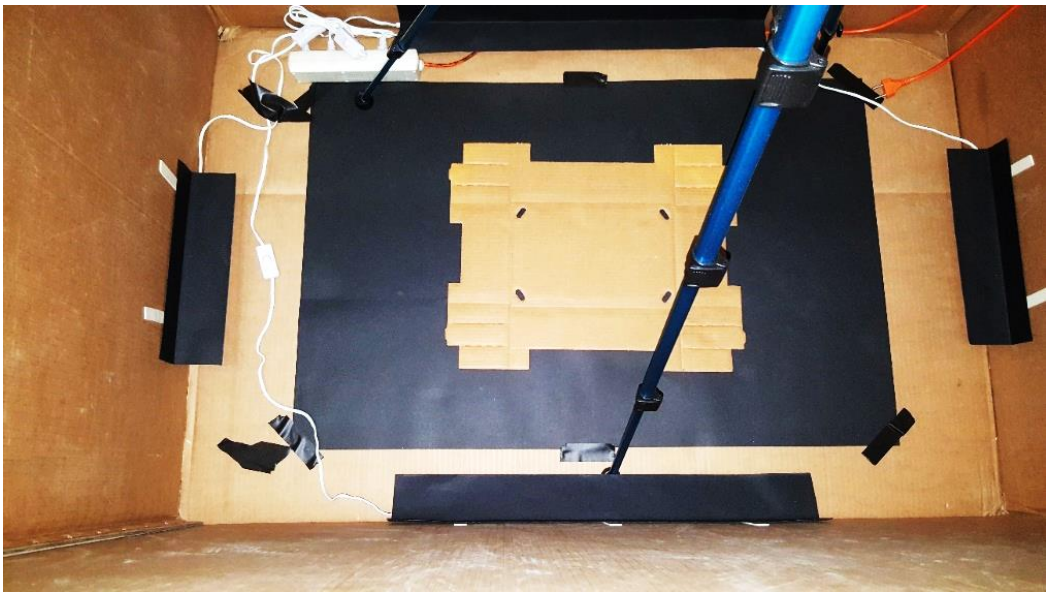


Figura 69 vista en planta

- **Cámara.**

Las fotos que necesitamos inicialmente para la base de datos han sido capturadas con la cámara del móvil y transferidas posteriormente al ordenador. Sin embargo, para la ejecución del algoritmo de segmentación sobre imágenes en tiempo real, se podría utilizar la aplicación '*Android IP WebCam* ' [63] junto con el código necesario en Matlab, con el propósito de usar la cámara del móvil desde Matlab a través de la red WIFI usando la dirección IP.

A continuación, se describen algunos parámetros de la cámara utilizada [64]:

- **Modelo**: Cámara trasera del Samsung Galaxy s6 Edge Plus.
- **Sensor**: CMOS.
- **Apertura máxima**: f1.9.
- **Tamaño de imagen**: 8 MP usado de 16 MP máximo (2448x3264 px).
- **Modo**: Pro
- **Sensibilidad ISO**: 200.
- **Velocidad de obturación**: 1/10.
- **Enfoque**: Infinito.
- **Balance de blancos**: 5500k.
- **Zoom**: x1.

En la siguiente figura, se muestran los ajustes de color y contraste que se han llevado a cabo para aclarar más las características de la plancha sin renunciar a las ventajas conseguidas con la técnica de iluminación.

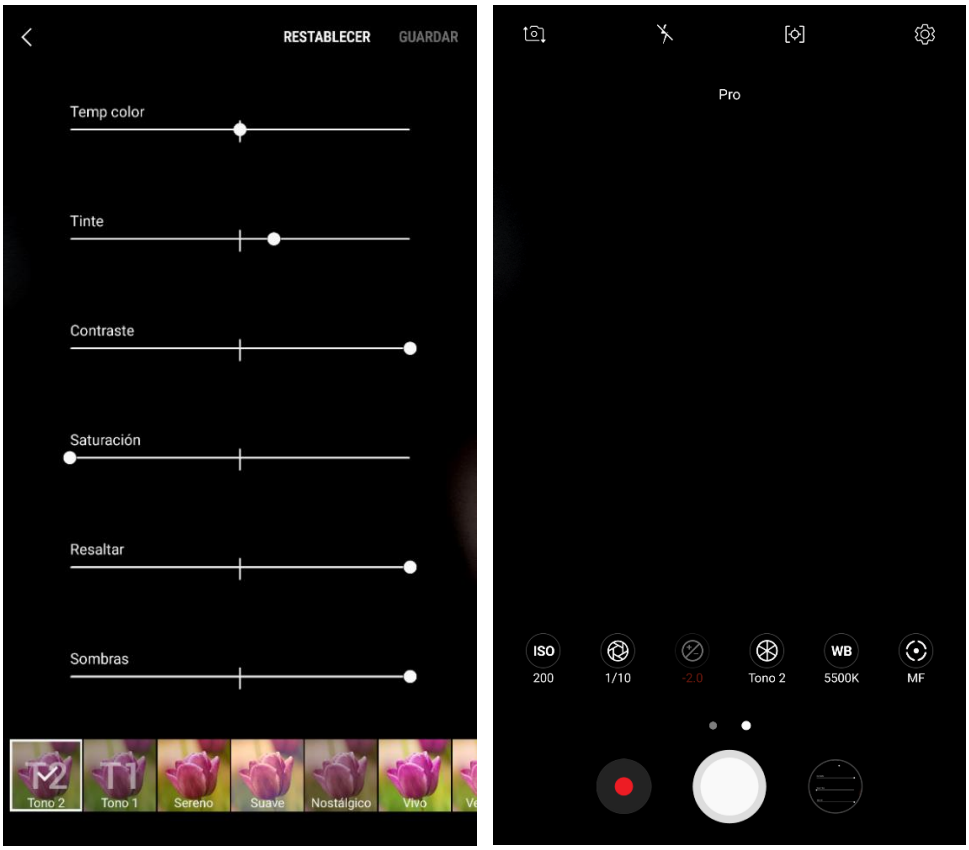


Figura 70 Ajustes de color y contraste

Los ajustes realizados han permitido conseguir el siguiente resultado.

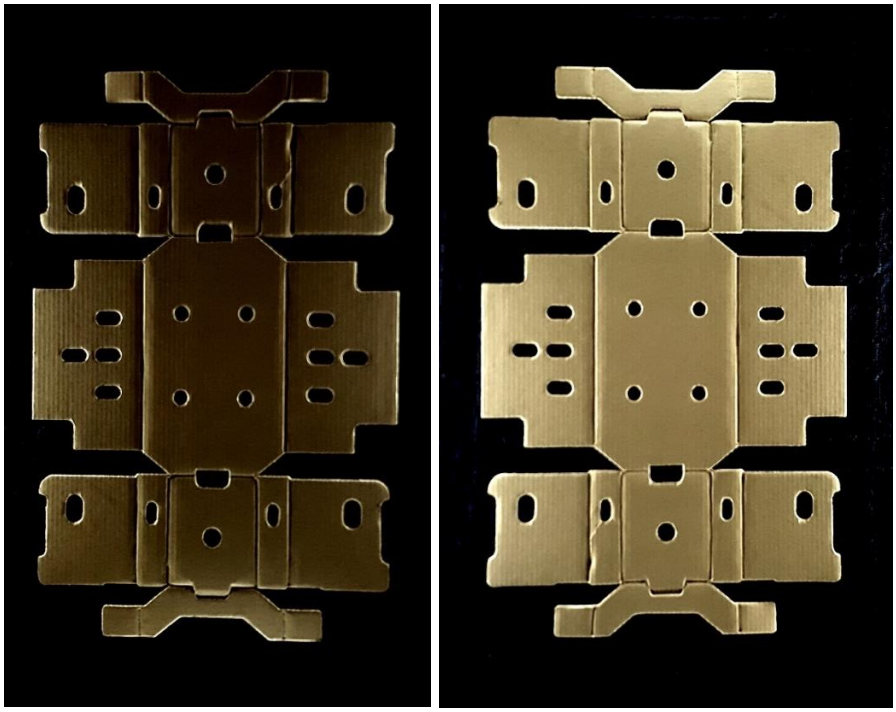


Figura 71 antes y después de realizar los ajustes

En la siguiente figura, se detallan todos los elementos del sistema de visión construido y las medidas principales.

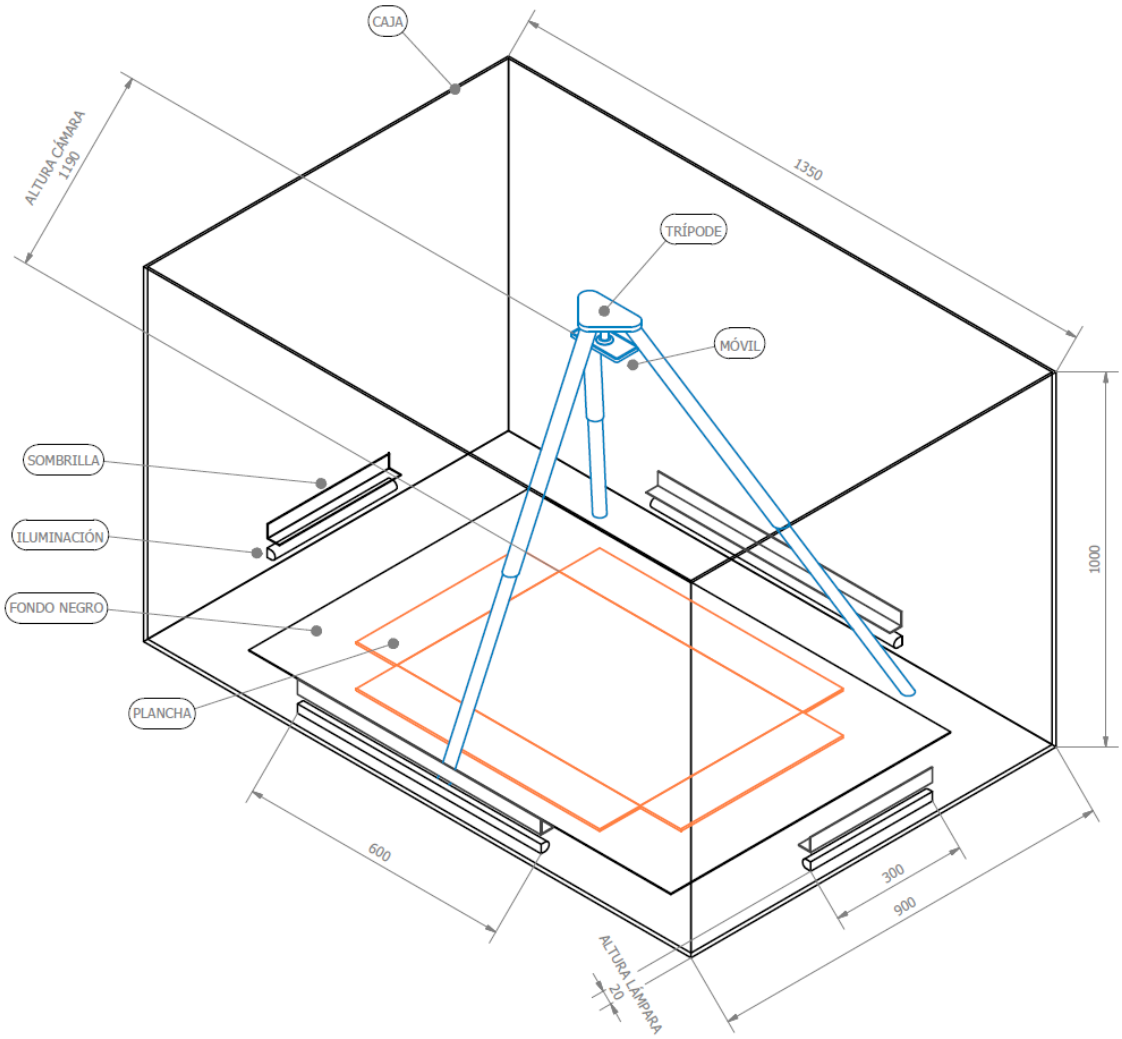


Figura 72 elementos del sistema de visión

Se ha adquirido un total de 110 imágenes de los modelos de plancha *Columna* y *Plaform ®* desde diferentes ángulos. Esta cantidad es pequeña debido a que la mayoría de las planchas encontradas están sujetas a contratos de confidencialidad con el cliente y no se pueden utilizar fuera de la empresa. Las imágenes adquiridas son de planchas que no requieren ninguna licencia especial para utilizarlas en este proyecto. Para solucionar este problema, se utilizará más adelante la función *DataAumenter* para multiplicar la cantidad total de imágenes e incrementar la robustez del algoritmo de segmentación.

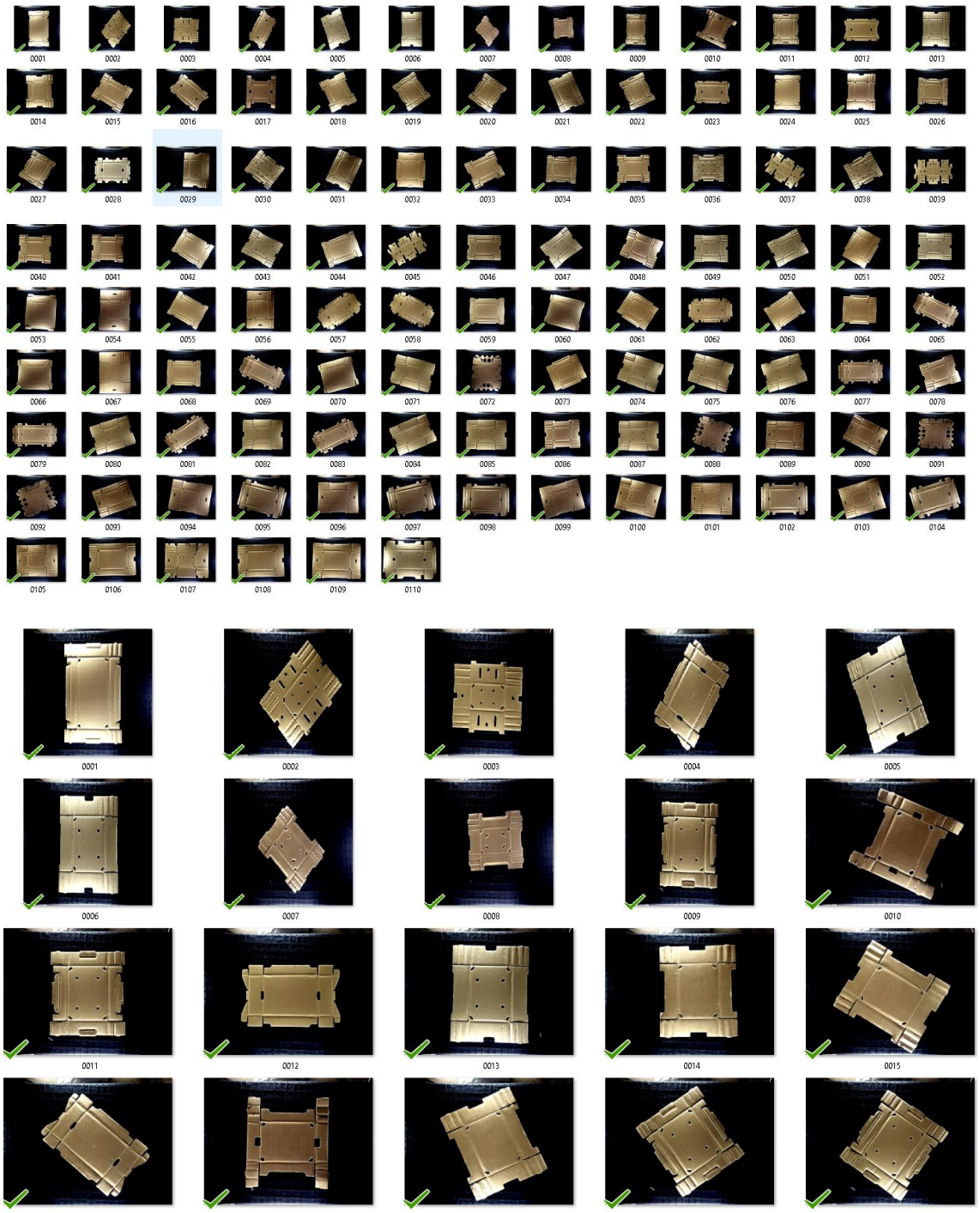


Figura 73 vista previa de las imágenes capturadas

Capítulo 4

4. Preprocesamiento

4.1. Introducción

Las imágenes capturadas no pueden ser utilizadas directamente como base de datos para el algoritmo de segmentación porque necesitan adaptación previa del formato y la dimensión para satisfacer los requisitos solicitados por la red neuronal. También necesitan algunas mejoras para optimizar el proceso de segmentación que consisten en la eliminación del fondo y recorte de la plancha.

4.2. Conversión del formato.

Las imágenes capturadas con la cámara del móvil tienen un formato JPG. Al ser un formato diseñado para la compresión de imágenes, tiende a perder la calidad de estas en las transformaciones [65]. Por esta razón las imágenes han sido convertidos al formato PNG que utiliza un algoritmo de compresión sin perdida [66]. Esta tarea se ha llevado a cabo mediante la herramienta de '*export*' del programa *Microsoft Office Picture Manager*.

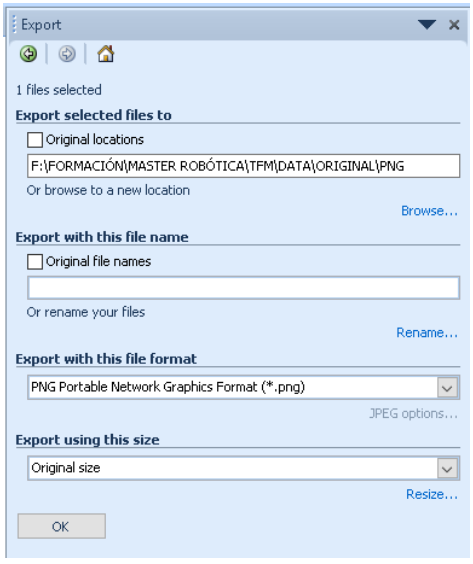


Figura 74 Herramienta Export en Microsoft Office Picture Manager

4.3. Algoritmo de preprocesamiento en Matlab.

En este capítulo se describen los pasos seguidos para crear el código en Matlab para el preprocesamiento de las imágenes. Las tareas principales realizadas por el algoritmo se describen en el siguiente diagrama de flujo.

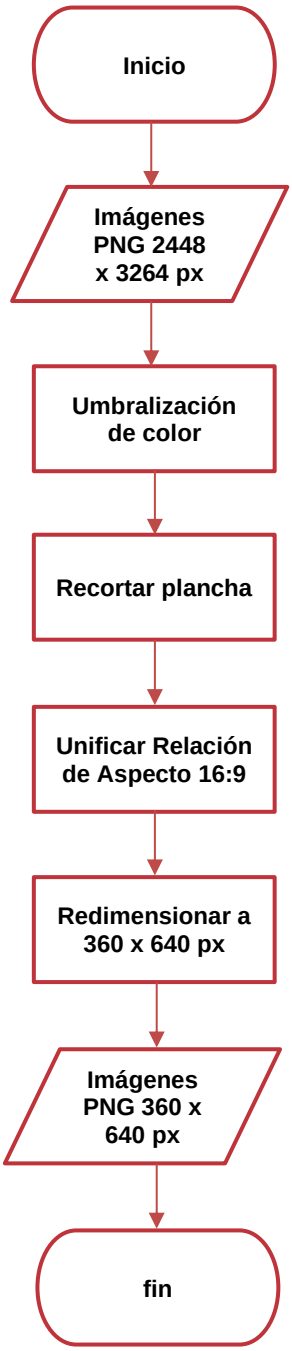


Figura 75 Algoritmo de preprocesamiento

En primer lugar, especificamos la ruta para las imágenes de entrada y la ruta donde se guardan las imágenes preprocesadas.

```
clc
clear all
close all

% ruta de imágenes de entrada
RutaOrigen = ('F:\FORMACIÓN\MASTER ROBÓTICA\TFM\DATA\ORIGINAL\PNG');

% ruta donde se guardan las imágenes preprocesadas
RutaDestino = ('F:\FORMACIÓN\MASTER ROBÓTICA\TFM\DATA\PREPROCESADO');
% añadir "\" a la ruta de origen
RutaOrigenPrep= [RutaOrigen '\'];
```

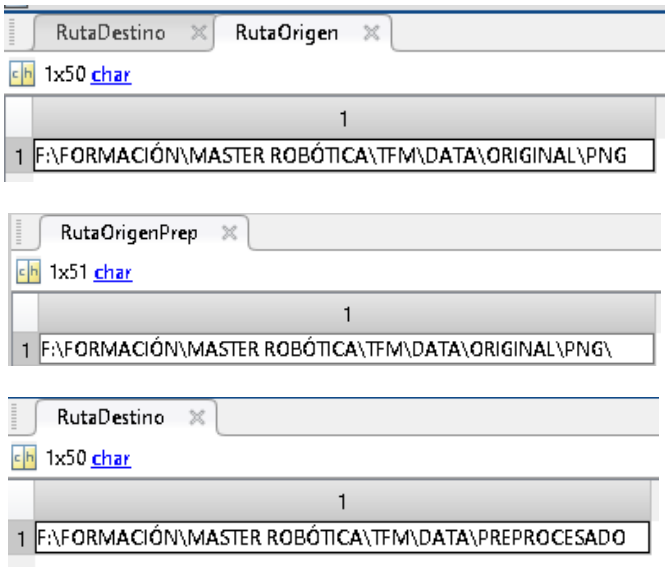


Figura 76 Rutas de entrada y salida de datos

En los siguientes pasos, se crea un listado con los nombres de las imágenes en la ruta de entrada.

```
% obtener la lista con los nombres de los elementos de la ruta de origen
ordenados
Elementos= ls(RutaOrigenPrep);
```

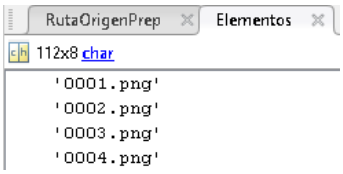


Figura 77 listado elementos de entrada

```
% insertar en una tabla los elementos de la lista
Tabla= cellstr(Elementos);

% ajustar la tabla
Tabla2= Tabla(3:length(Tabla));

% calcular la dimensión de la tabla
S= size(Tabla2);
a= 1;
```

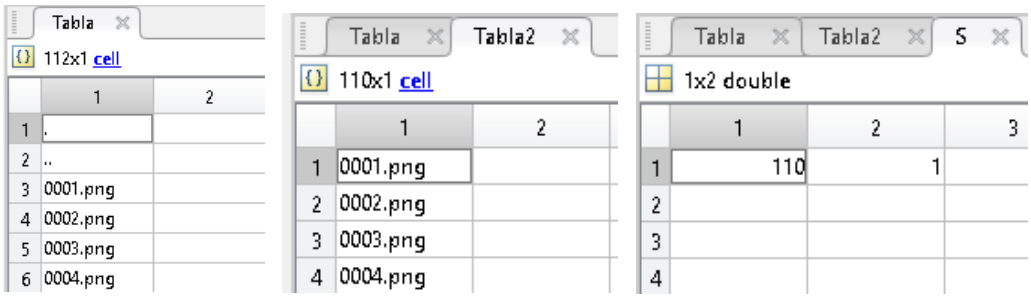


Figura 78 Tabla de elementos de entrada

A continuación, se crea el inició del bucle '*While*' para aplicar las tareas de preprocesamiento a cada una de las imágenes de entrada y guardar cada una de las imágenes preprocesadas en la carpeta de destino con un nombre correlativo.

```
23.   while a <= S(1)
24.       close all
25.
26.   %nombre con extensión del elemento número (a)
27.       NElemento= char(Tabla2(a));
28.
29.   %Nombre de la imágen
30.       NImagen = (NElemento);
31.
32.   % Construir el link al elemento (a)
33.       LinkImagen = [RutaOrigen '\' NImagen];
34.
35.   %leer la imágen que corresp al elemento (a)
36.       ImOriginal = imread(LinkImagen);
37.
38.   % obtener las dimensiones de la imágen
39.       [Altura,Ancho,z1] = size(ImOriginal);
```

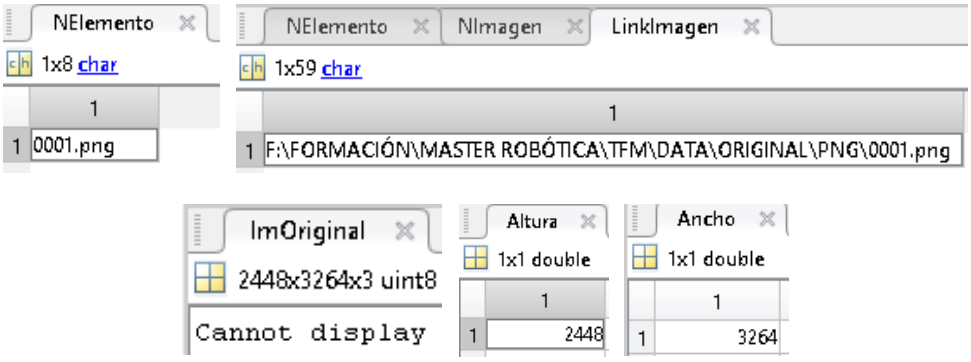


Figura 79 Inicio Bucle While

4.3.1. Umbralización y eliminación de fondo.

El objetivo de este apartado es realizar una umbralización de color para eliminar el fondo que contiene reflejos y degradaciones de color que pueden distraer al algoritmo de segmentación y alterar su precisión (Figura 80).

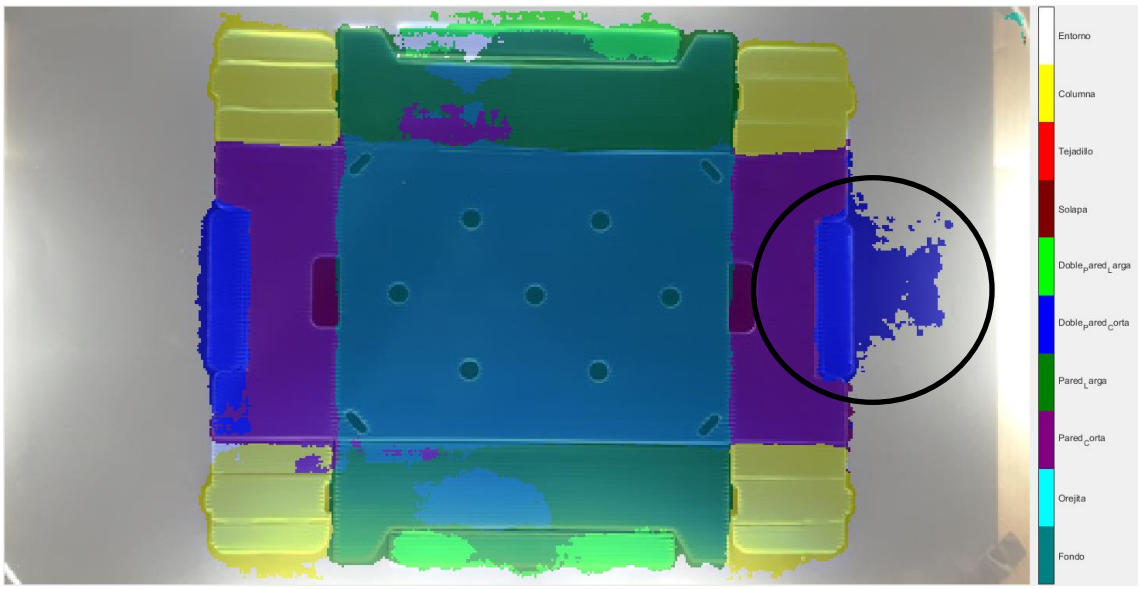


Figura 80 Ejemplo de segmentación semántica sobre una imagen sin eliminación de fondo

Con la ayuda de la aplicación de Matlab *Color Thresholder* [67] de la *Image Processing and computer visión toolbox*, identificamos el umbral de colores de la plancha y lo aislamos del resto de colores que pertenecen al fondo reemplazándolos con un color negro.

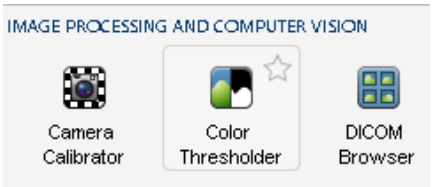


Figura 81 Color Thresholder

Con el resultado de esta umbralización, se exportan los valores de los umbrales de color hacia el espacio de trabajo para crear una función y aplicarla a todas las imágenes de forma automática utilizando el algoritmo de preprocesamiento que estamos desarrollando. El umbral de colores es similar para todas las planchas ya que tanto el color del cartón como las condiciones de iluminación no presentan ninguna variación.

Esta aplicación permite crear una máscara de segmentación manipulando los componentes de color de una imagen, basándose en diferentes espacios de color.

En primer lugar, hay que cargar una imagen a la aplicación. Ya que todavía no tenemos ninguna en el espacio de trabajo, la cargamos directamente desde la ruta de imágenes de entrada.

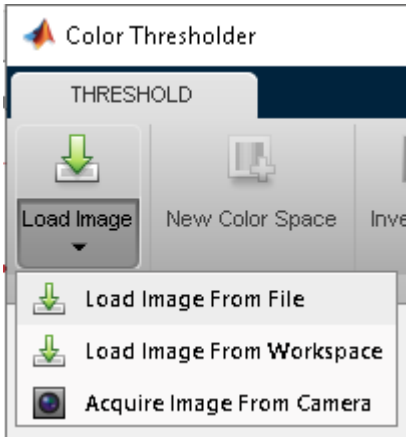


Figura 82 Cargar imágenes

En el próximo paso, seleccionamos un espacio de color donde trabajar. Consiste en encontrar el espacio de color que mejor separa los umbrales de colores de la plancha y del fondo en la imagen. Para ello, probamos cada uno de los espacios de colores disponibles hasta encontrar el más adecuado.

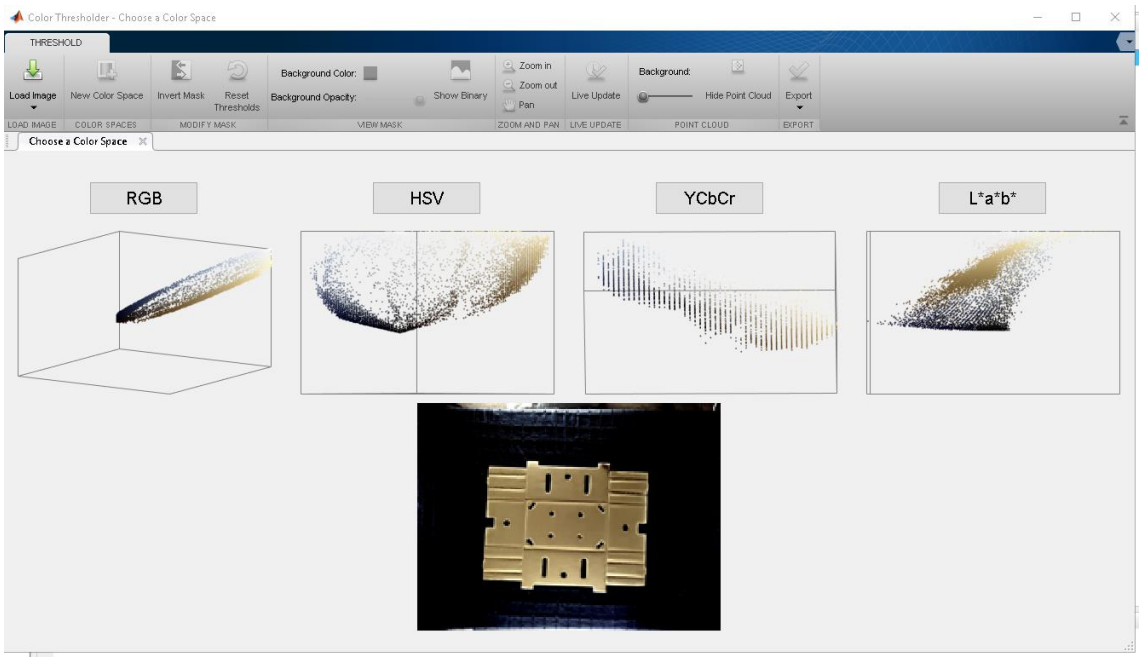


Figura 83 Espacios de colores

- **El espacio de color RGB.**

El modelo de color RGB utiliza una mezcla de colores aditivos, porque describe qué tipo de luz necesita ser emitida para producir un color dado. RGB almacena valores individuales para el rojo, el verde y el azul [68].

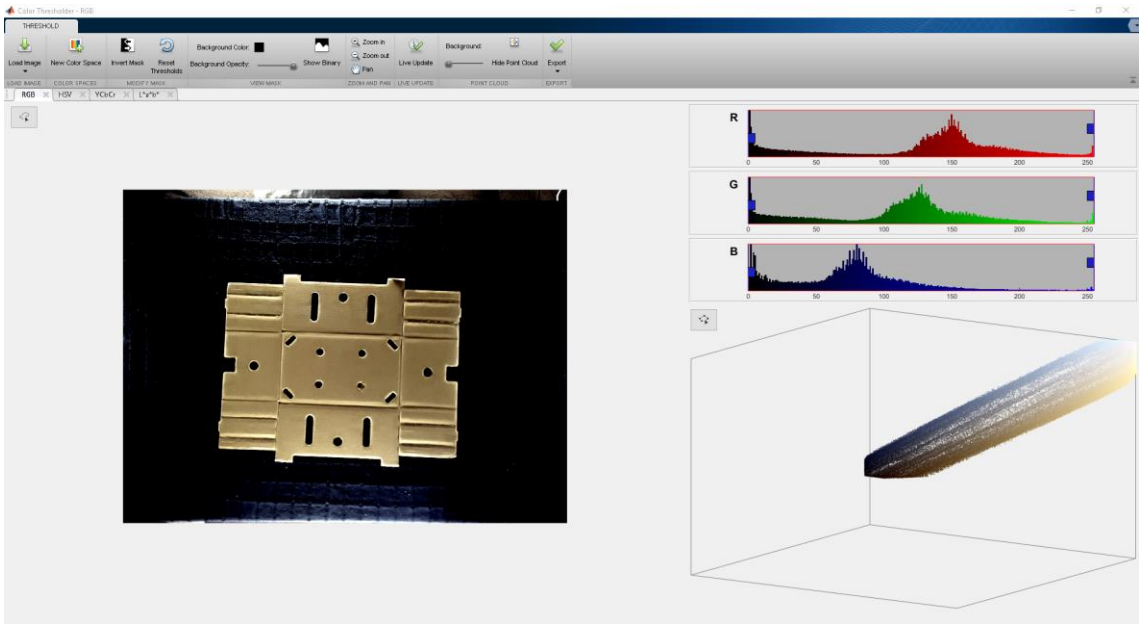


Figura 84 Espacio de color RGB

Ajustamos los tres canales intentando conseguir el objetivo deseado.

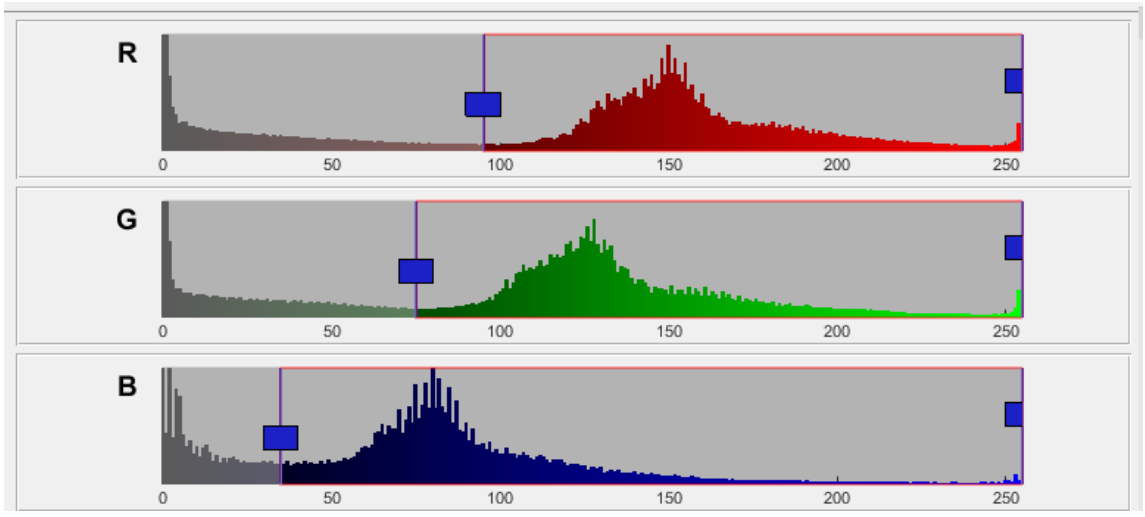


Figura 85 Ajuste de valores RGB

Lo único que permite hacer este espacio de color es disminuir la presencia del fondo oscuro, pero no es capaz de eliminar las degradaciones de colores del fondo causados por los reflejos de la luz ya que algunos de estos colores existen también en la plancha y si se eliminan se alterará la calidad de esta.

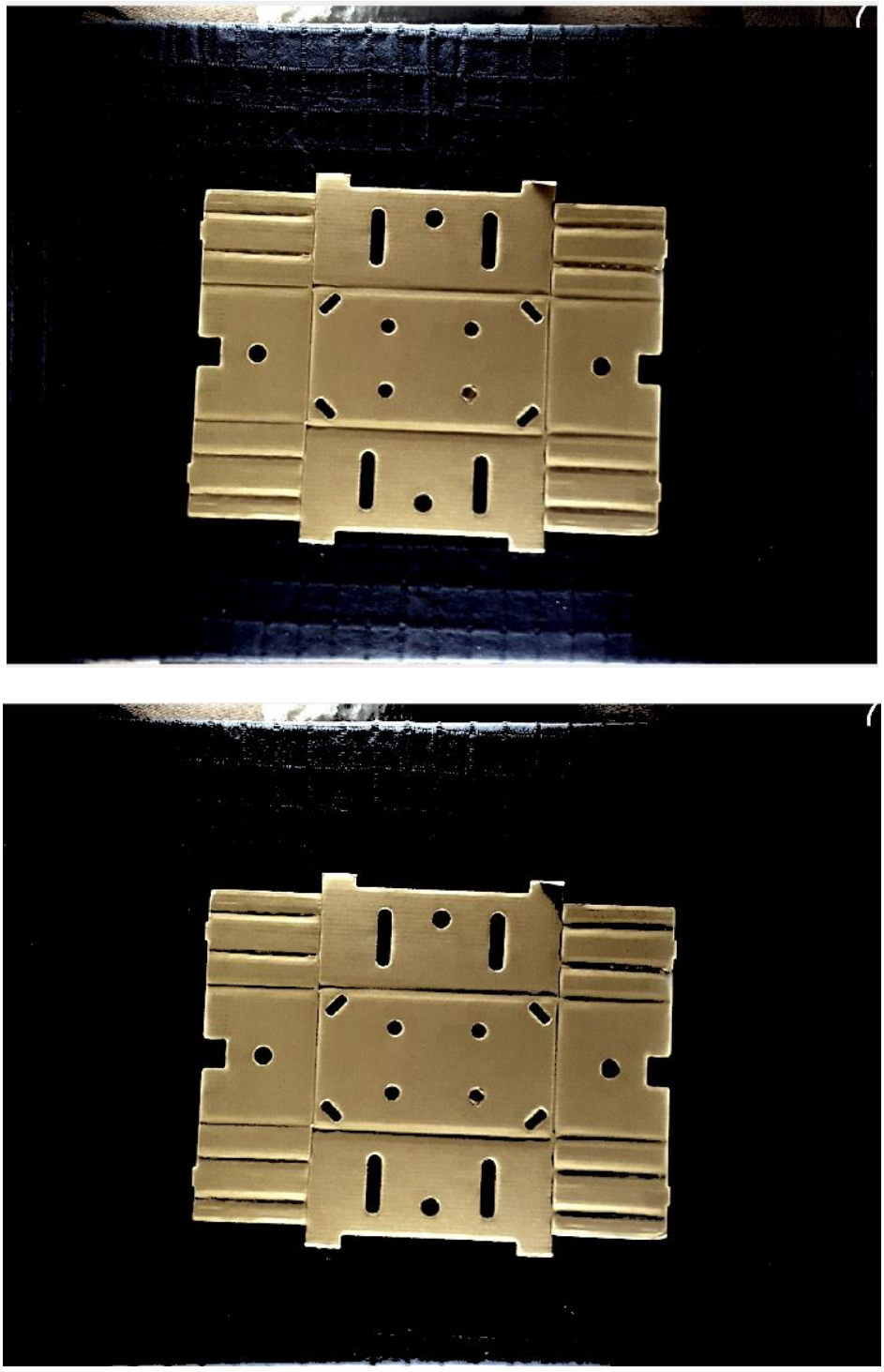


Figura 86 Antes y después de manipular el color de la plancha en el espacio RGB

- **El espacio de color HSV.**

El espacio de color HSV es una representación tridimensional del color basado en los componentes de tinte, matiz o tonalidad (hue, en inglés), saturación (saturation) y brillo o valor (value) [69].

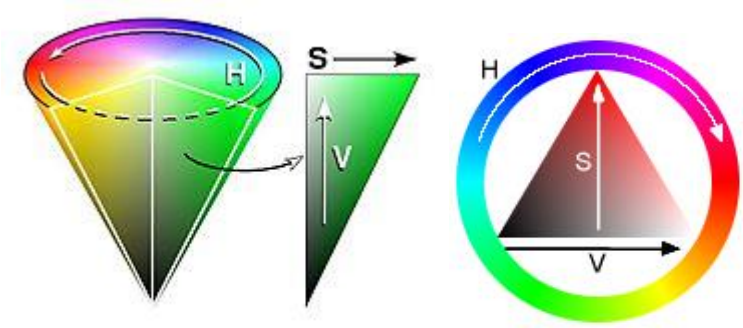


Figura 87 Cono del espacio de color HSV [69]

- **Matiz (H)**: Se representa como un grado de ángulo cuyos valores posibles van de 0 a 360°. Cada valor corresponde a un color.

- **Saturación (S)**: Se representa como la distancia al eje de brillo negro-blanco. Los valores posibles van del 0 al 100%.

- **Valor (V)**: Representa la altura en el eje blanco-negro. Los valores posibles van del 0 al 100%. 0 siempre es negro. Dependiendo de la saturación, 100 podría ser blanco o un color más o menos saturado.

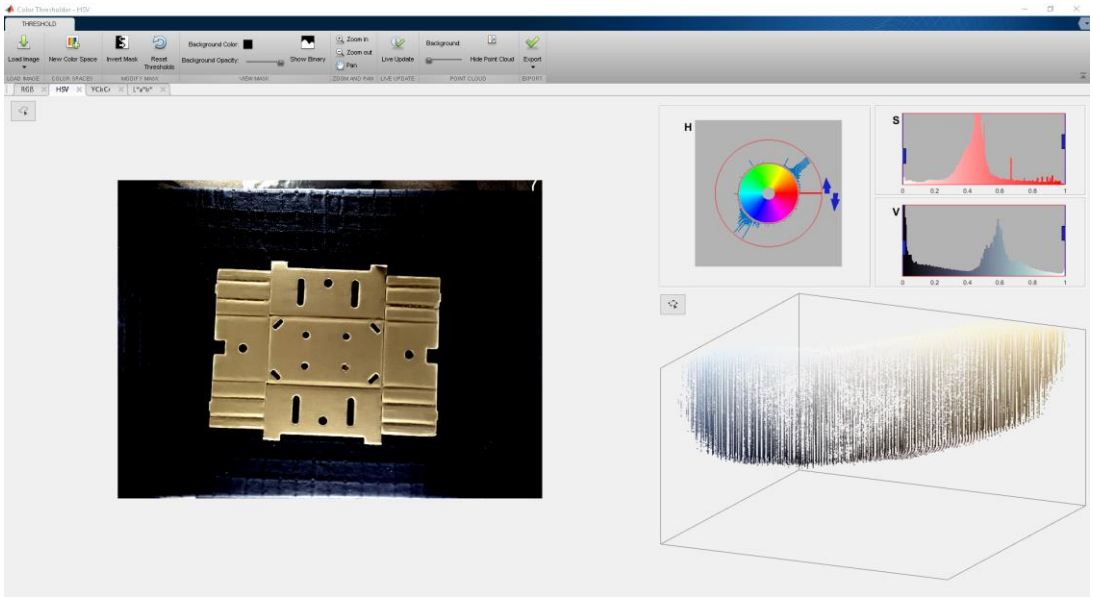


Figura 88 Espacio de color HSV

Se observa que los colores de interés se concentran en la zona superior derecha de la cara plana del cono que representa el canal H. Los histogramas de los canales S y V no los podemos tocar porque podemos eliminar algunos colores que forman la plancha como el blanco que define el reflejo en los cortes, el negro que define la sombra en los pliegues, y la degradación del marrón que define el color del cartón.

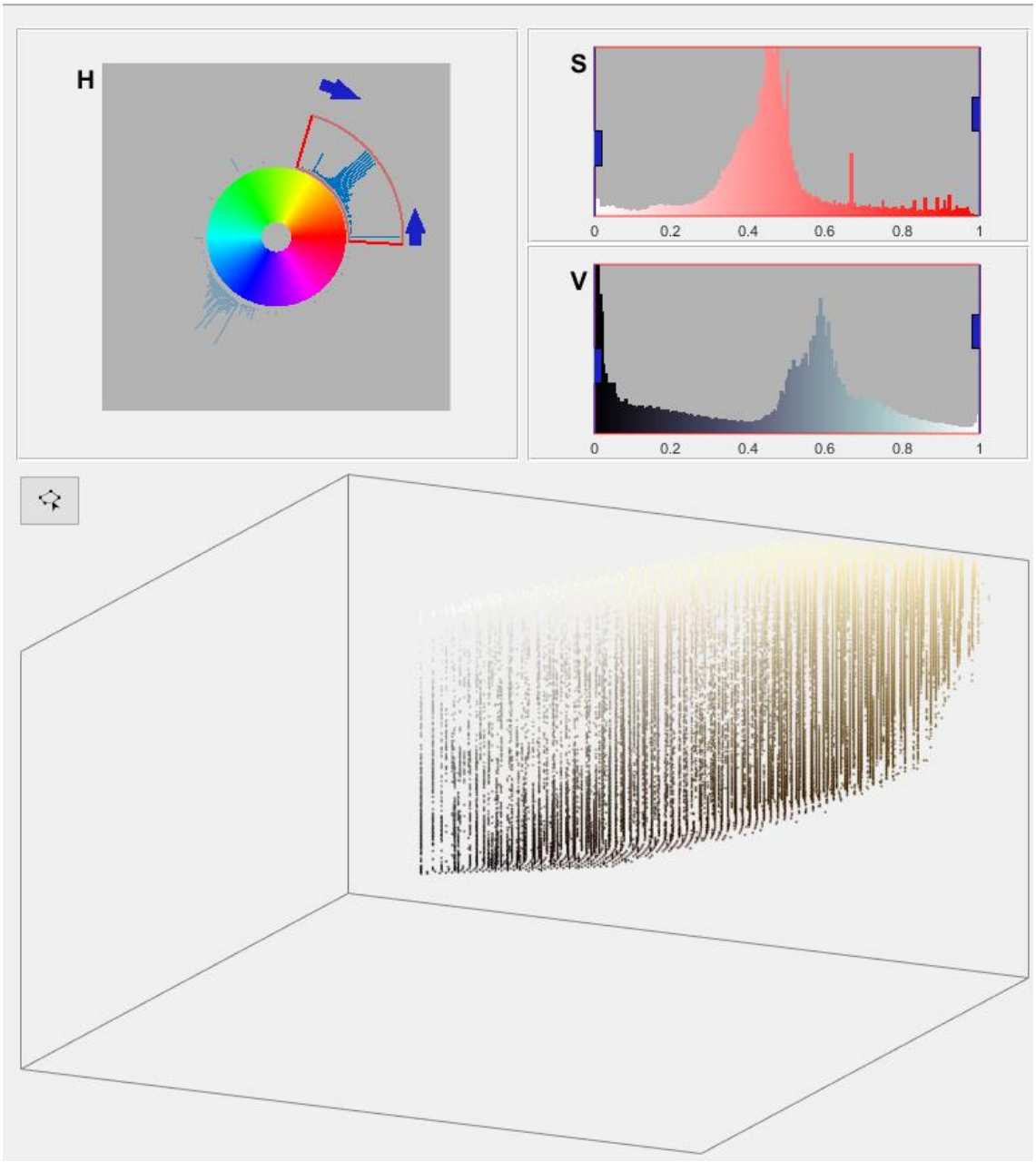


Figura 89 Histogramas del espacio HSV

Este espacio de color elimina más colores del fondo que el espacio RGB. Sin embargo, quedan algunas zonas en los bordes de la imagen sin eliminar porque

contienen colores similares a los colores de la plancha.

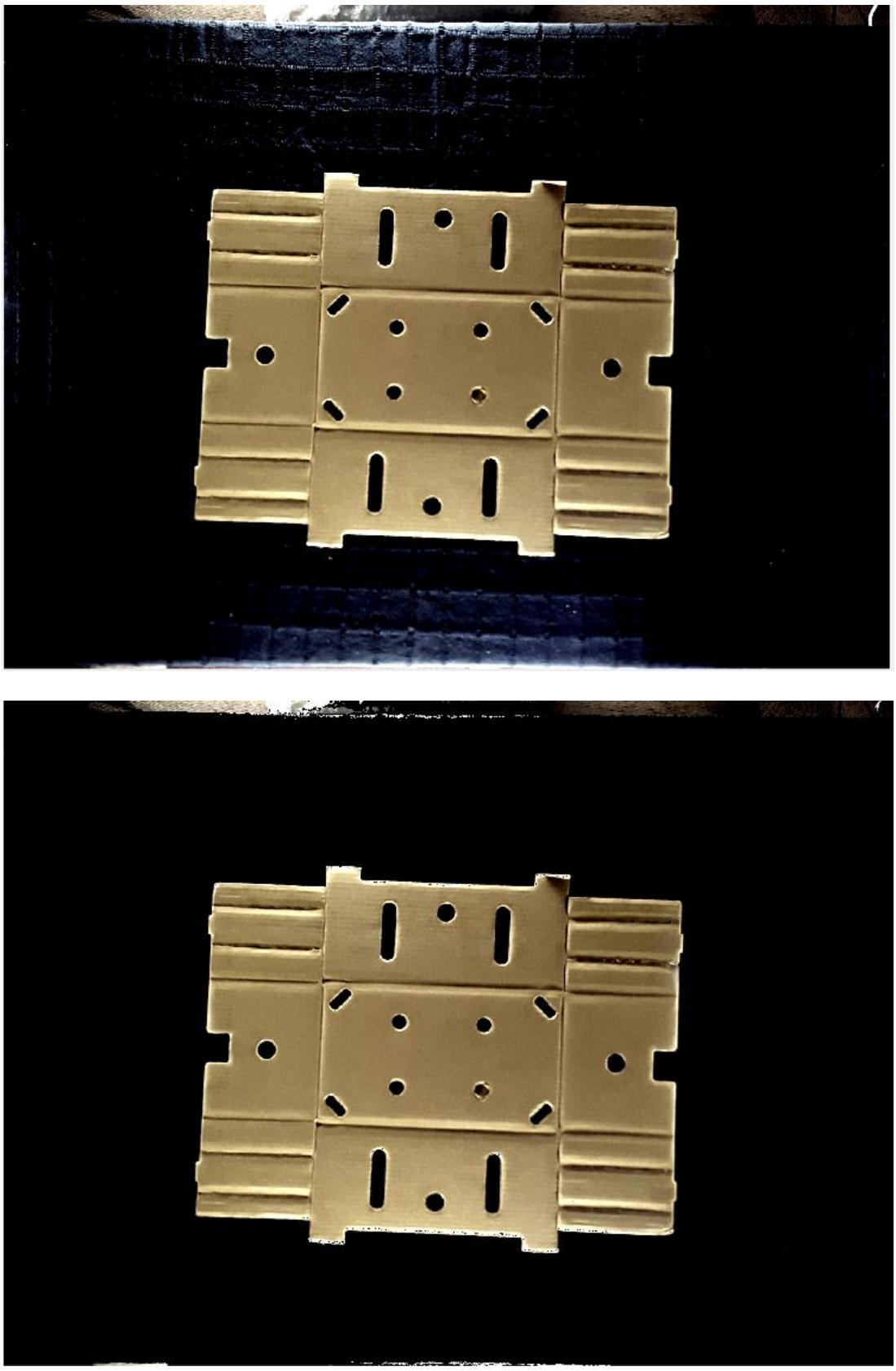


Figura 90 Antes y después de manipular el color de la plancha en el espacio HSV

- **El espacio de color YCbCr.**

Es una familia de espacios de color utilizados como parte de la canalización de imágenes [70].

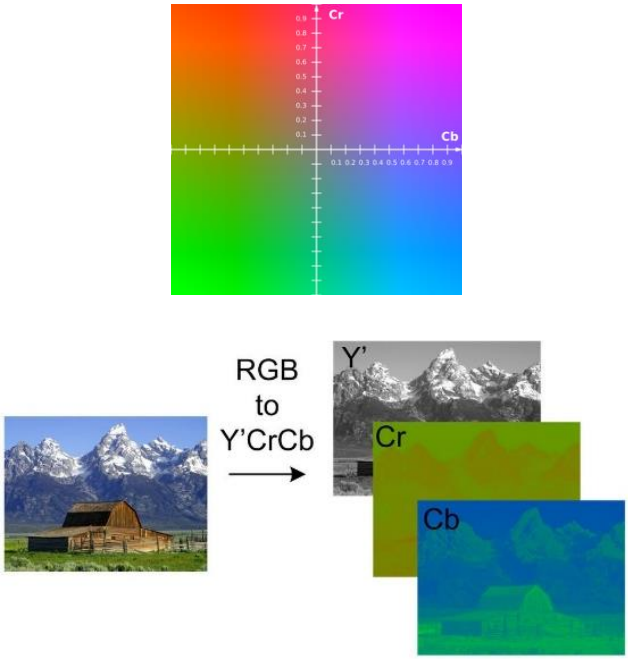


Figura 91 Canales del espacio de color YCbCr [70]

- **Y**: representa la luminancia (luma) y se encuentra en el rango de 0 a 255.

- **Cb y Cr**: representan los componentes de crominancia (croma) de diferencia azul y diferencia de rojo. Se encuentran en el rango 0 a 255.

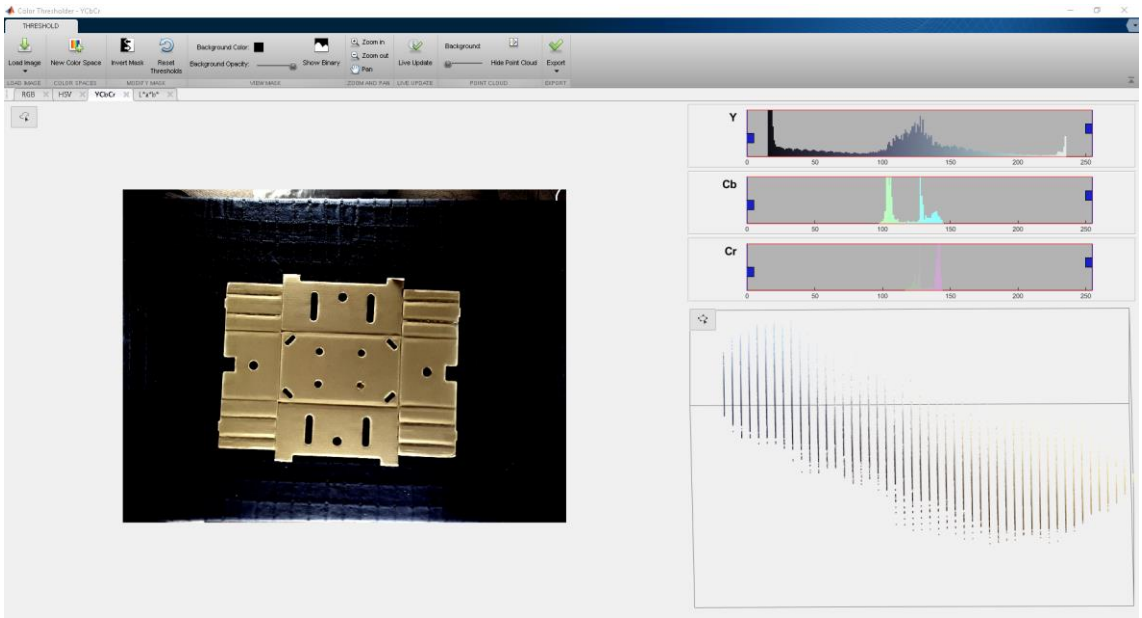


Figura 92 Espacio de color YCbCr

Ajustamos manualmente el histograma de los componentes Cb y Cr para conseguir nuestro objetivo. No podemos modificar el histograma del componente Y porque podemos eliminar los colores blanco y negro que también forman parte de los colores de la plancha.

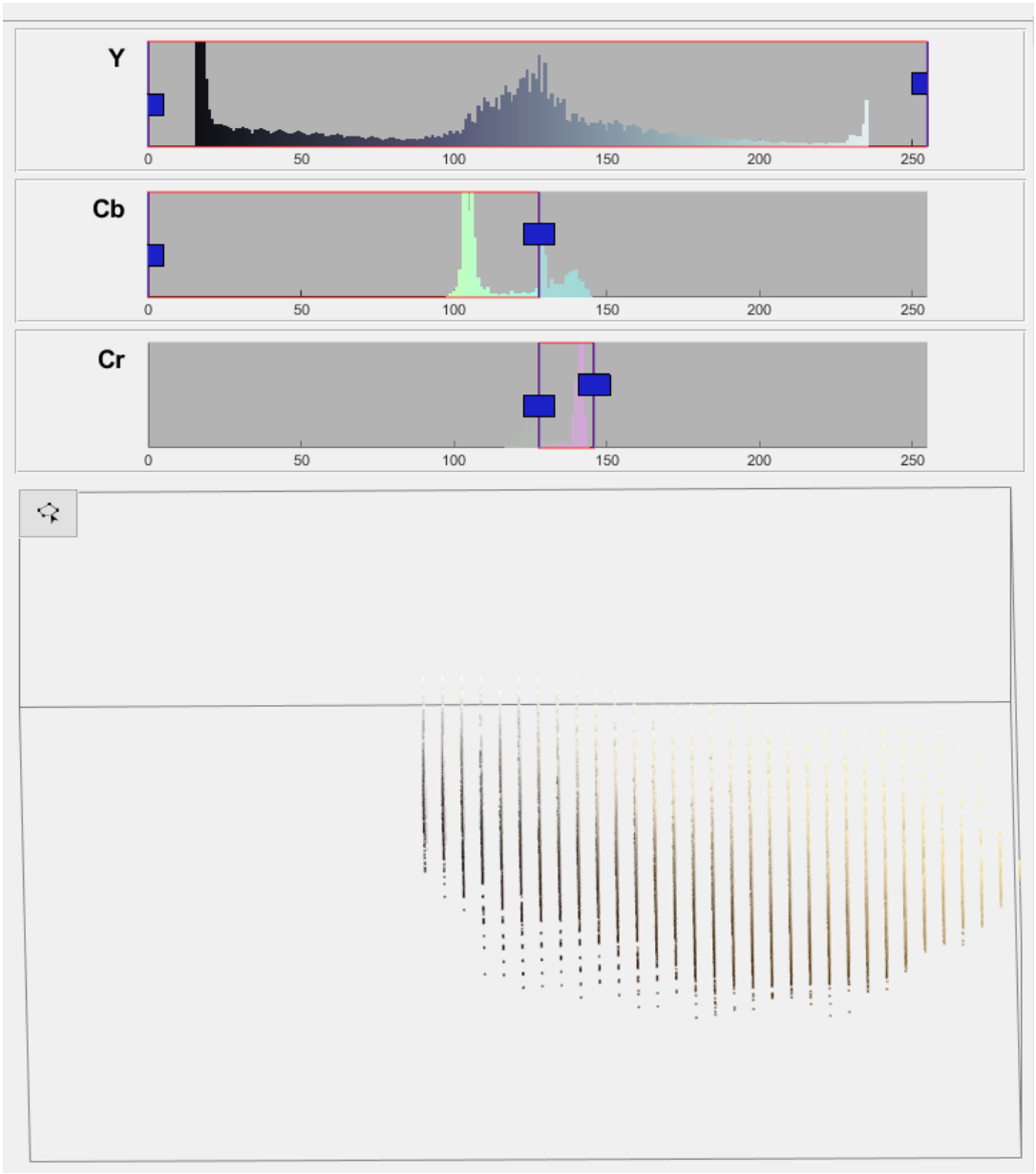


Figura 93 Histogramas del espacio YCbCr

En la siguiente figura, se observa la diferencia una vez manipulados los histogramas. No hay prácticamente ninguna diferencia entre los resultados de este espacio de color y el espacio HSV.

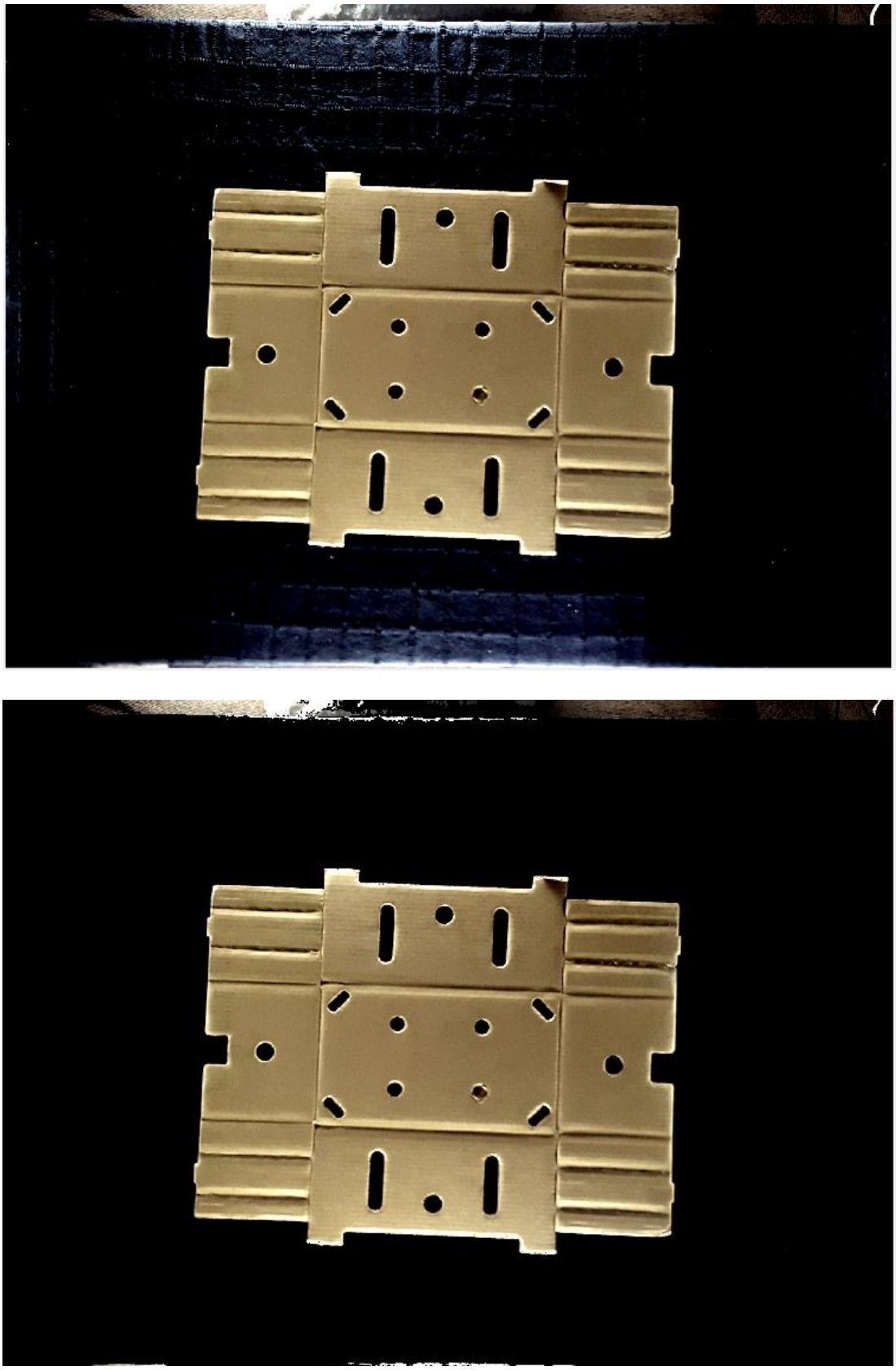


Figura 94 Antes y después de manipular el color de la plancha en el espacio YCbCr

- **El espacio de color L*a*b.**

El espacio de color CIELAB es una transformación matemática del espacio XYZ en el cual se fija un blanco de referencia y cuyos valores de triestímulo son (Xn, Yn, Zn). Ese blanco de referencia puede ser, por ejemplo, una fuente luminosa, el iluminante al que se haya adaptado el observador, un difusor perfecto o el color neutro más reflectante o transmisor de un medio de reproducción (entonces es (media-relative)).

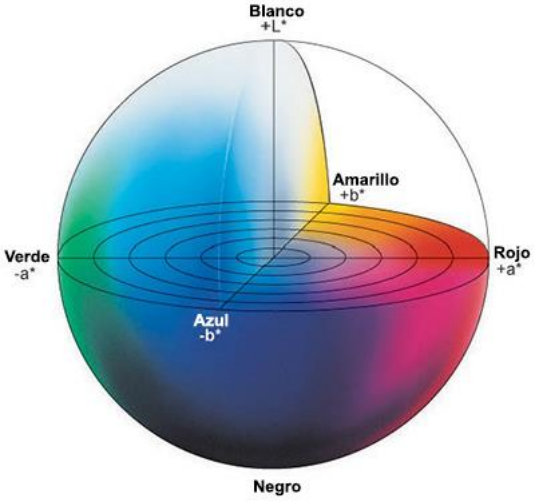


*Figura 95 los canales del espacio L*a*b [71]*

Los tres ejes del sistema CIELAB se indican con los nombres L*, a* y b*. Representan, respectivamente Luminosidad (lightness), tonalidad de rojo a verde (redness-greenness) y tonalidad de amarillo a azul (yellowness-blueness) (los dos últimos ejes están inspirados en la teoría de los colores oponentes) [71].

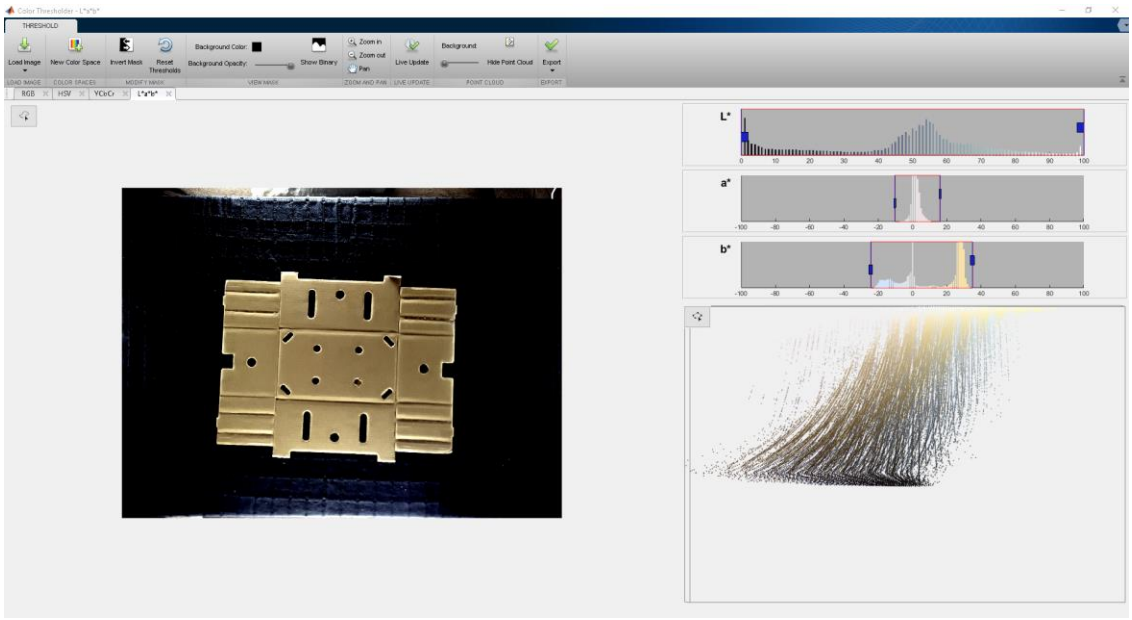


*Figura 96 Espacio de color L*a*b*

En el histograma del componente b* que representa la tonalidad del amarillo a azul podemos distinguir fácilmente entre dos zonas bien separadas. La graduación del azul que representa la mayor parte de los reflejos sobre el fondo de la plancha y que nos interesa eliminar. Y la graduación del amarillo que representa el color de la plancha. Hay una parte en la graduación del azul que no se puede eliminar porque representa un brillo que está presente en los reflejos sobre los bordes y pliegues de la plancha.

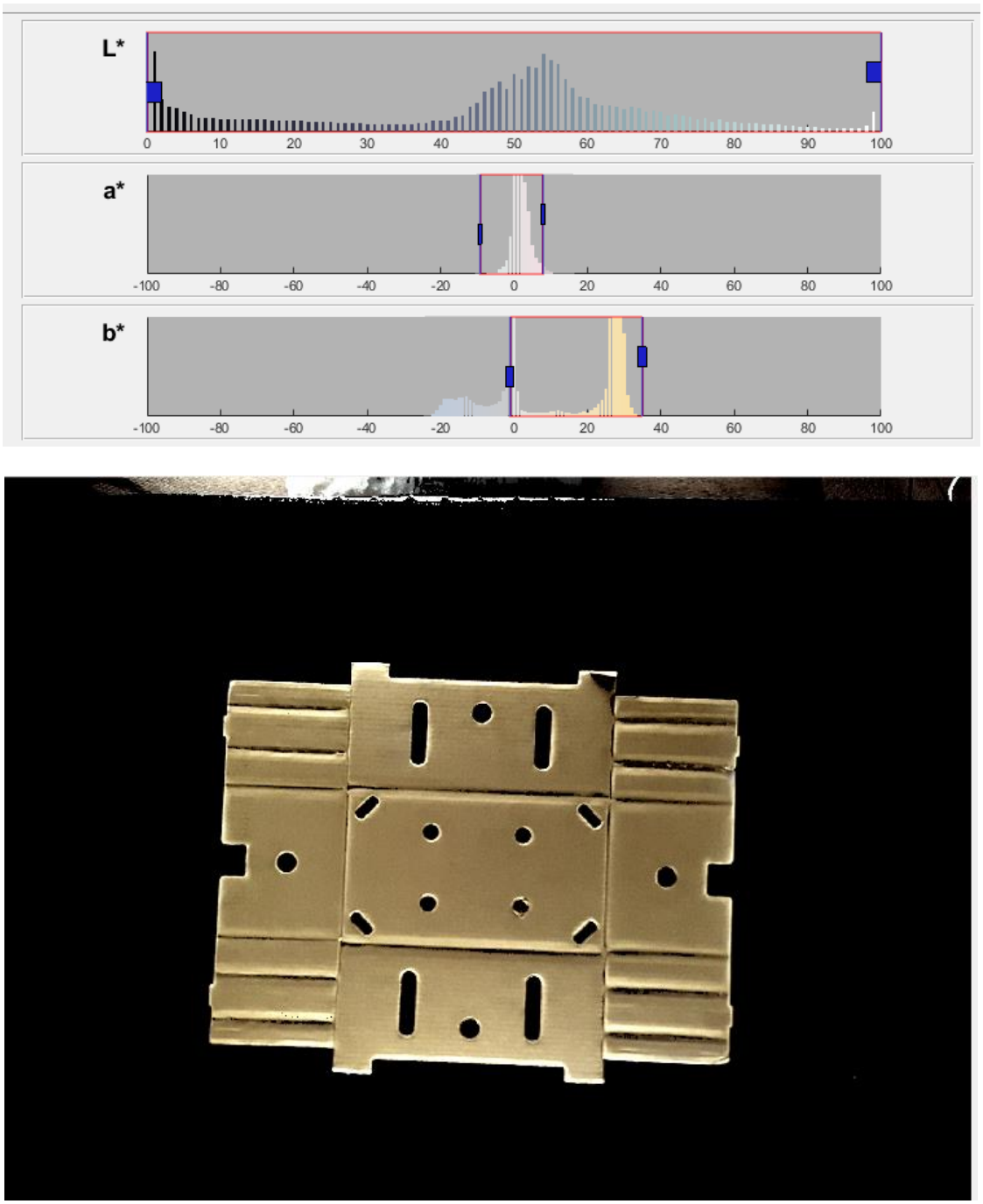


*Figura 97 Histogramas del espacio L*a*b*

Igual que los espacios de color vistos anteriormente, quedan algunos colores sin eliminar en los bordes de la imagen para no afectar a la calidad de la plancha. Si eliminamos estos colores del fondo, se eliminan también de la plancha.

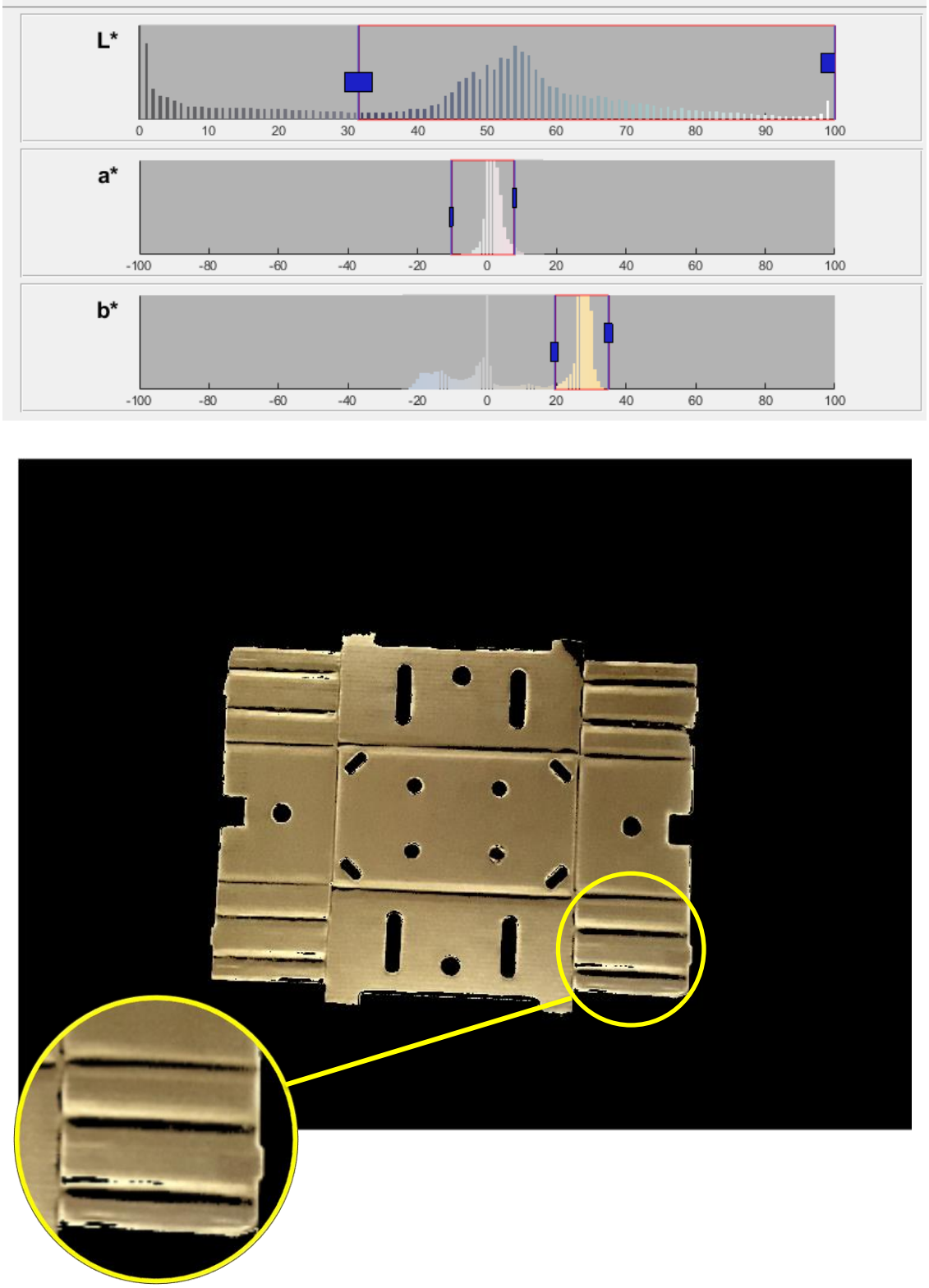


Figura 98 Resultado de eliminar todos los colores del fondo de la plancha

Valorando todos los espacios de color disponibles en la herramienta, el espacio de color L*a*b es el que mejor separa los colores de la plancha representados principalmente por el azul que define la degradación de los colores producidos por los reflejos en el fondo y el amarillo definido por la degradación de los colores de la plancha.

Una vez ajustados los umbrales sobre el espacio de color seleccionado, exportamos los valores de la umbralización al espacio de trabajo de Matlab para crear una función que se puede aplicar sobre cada una de las imágenes utilizando el algoritmo de preprocesamiento.

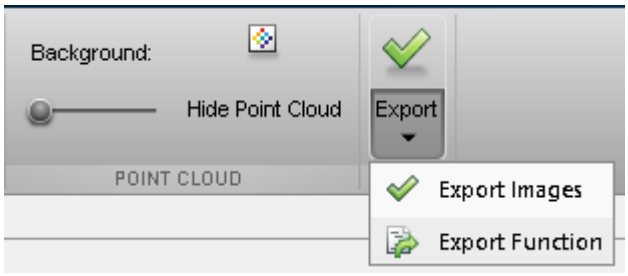


Figura 99 Exportar función al espacio de trabajo

```
function [BW,ImMascara] = EliminarFondo(ImRGB)
    %createMask  Threshold RGB image using auto-generated code from
    colorThresholder app.
    %  [BW,MASKEDRGBIMAGE] = createMask(RGB) thresholds image RGB using
    %  auto-generated code from the colorThresholder app. The colorspace and
    %  range for each channel of the colorspace were set within the app. The
    %  segmentation mask is returned in BW, and a composite of the mask and
    %  original RGB images is returned in maskedRGBImage.
    % Auto-generated by colorThresholder app on 01-May-2019
    %------------------------------------------------------
    % Convertir la imágen del espacio de colore RGB al espacio de color L*a*b
    I = rgb2lab(ImRGB);
    % Definir la umbralizacion sobre el canal 1 basando en el histograma
    Canal1Min = 0.000;
    Canal1Max = 100.000;
    % Definir la umbralizacion sobre el canal 2 basando en el histograma
    Canal2Min = -10.405;
    Canal2Max = 16.208;
    % Definir la umbralizacion sobre el canal 3 basando en el histograma
    Canal3Min = -2.037;
    Canal3Max = 34.813;
    % Crear una máscara basada en la umbralización ajustada sobre el
    histograma
    MascaraBW = (I(:,:,1) >= Canal1Min ) & (I(:,:,1) <= Canal1Max) & ...
        (I(:,:,2) >= Canal2Min ) & (I(:,:,2) <= Canal2Max) & ...
        (I(:,:,3) >= Canal3Min ) & (I(:,:,3) <= Canal3Max);
    BW = MascaraBW;
    % Inicializar la imagen de salida con máscara basada en la imagen de
    entrada.
    ImMascara = ImRGB;
    % reemplazar por cero el valor de los pixeles del fondo donde la imagen
    binaria BW es False
    ImMascara(repmat(~BW,[1 1 3])) = 0;
end
```

La función tiene como entrada la imagen de la plancha en espacio RGB y como salida una imagen umbralizada a partir de los valores ajustados en los canales del espacio L*a*b.

La primera tarea consiste en convertir la imagen RGB al espacio de color L*a*b. Luego, una vez especificados los valores máximos y mínimos de los umbrales para los 3 canales, se crea una máscara binaria donde se asigna el valor 1 (True) a los píxeles representados por el color blanco y que pertenecen a las zonas que se quieren mantener, y el valor 0 (False) a los pixeles representados por el color negro y que pertenecen a las zonas que se quieren eliminar.

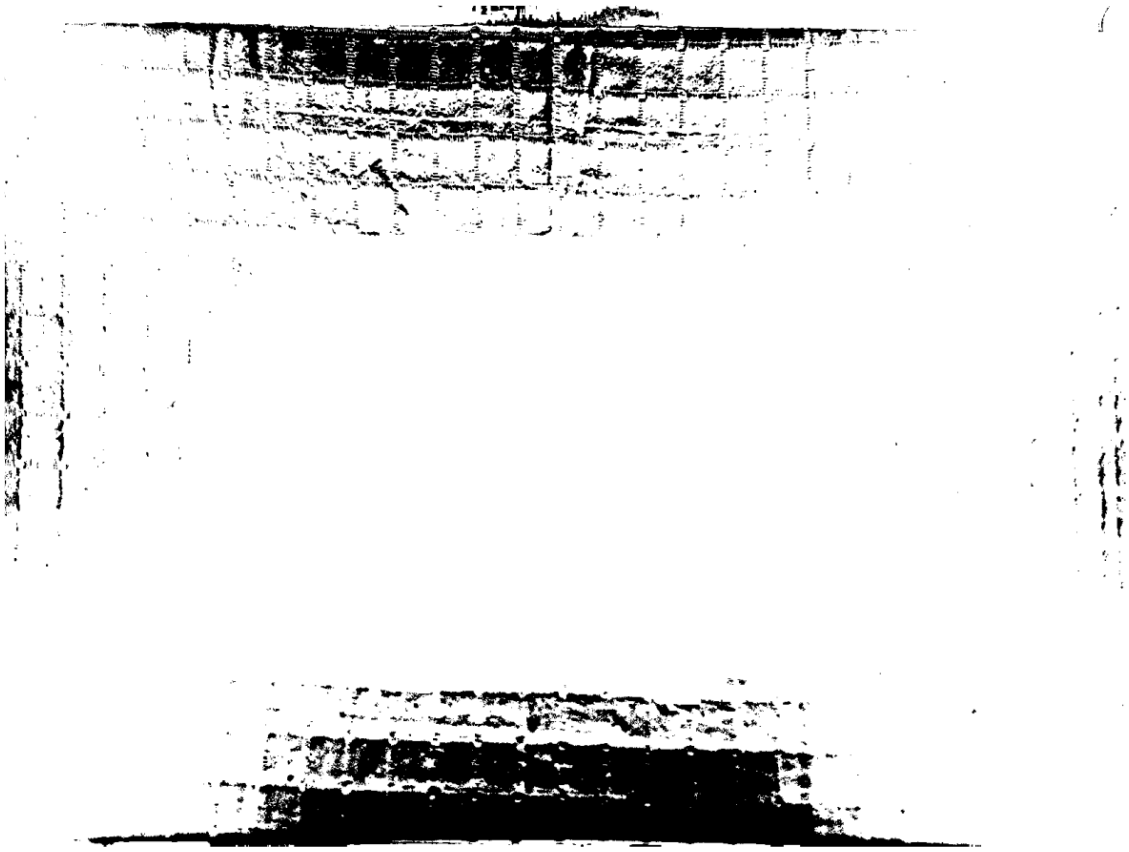


Figura 100 Máscara binaria (MascaraBw)

La tarea siguiente consiste en reemplazar por 0 (negro) los pixeles en la imagen original RGB que coinciden con los píxeles con valor 0 (False) en la máscara binaria. En la siguiente figura se ve la imagen RGB superpuesta sobre la máscara binaria para visualizar mejor el efecto de la umbralización.

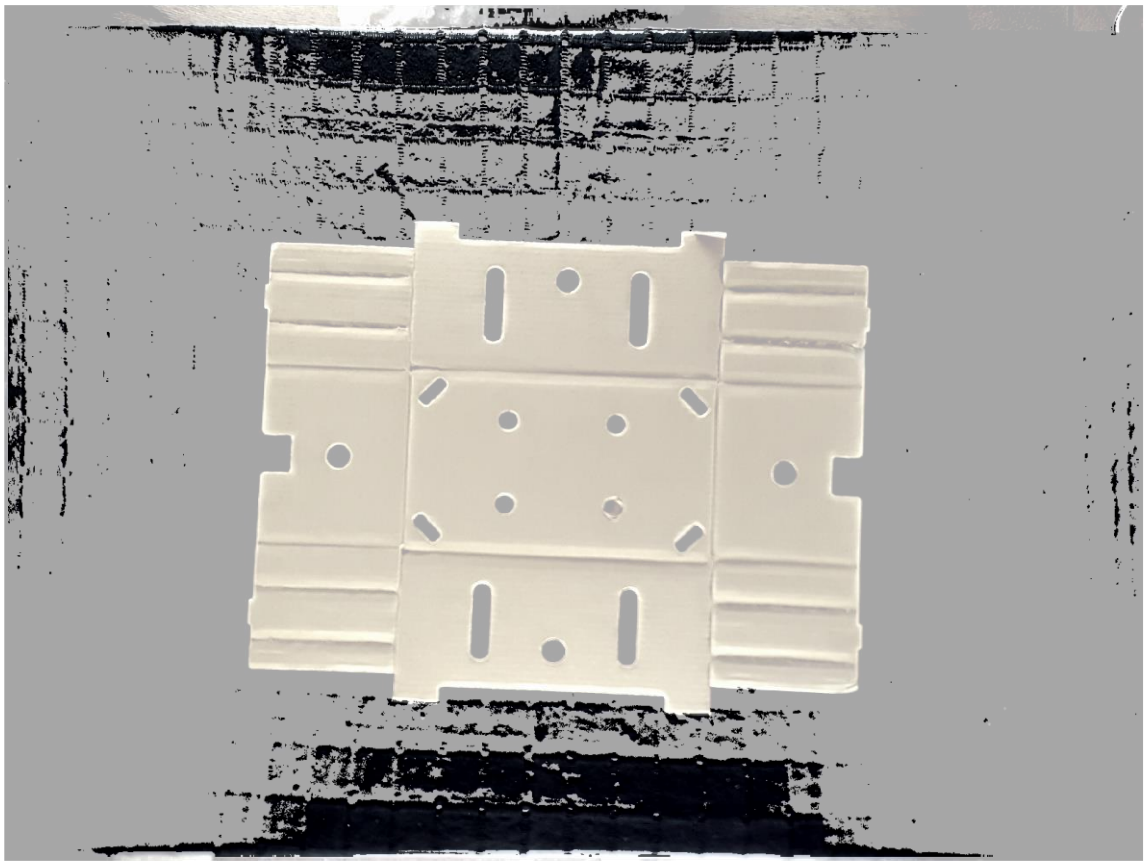


Figura 101 Imágen RGB superpuesta sobre la máscara binaria

Llamamos a esta función desde el bucle '*While*' del algoritmo de preprocesamiento que estamos desarrollando.

```
40.   %Eliminar el fondo de la plancha
41.   [BW,ImSinFondo]=EliminarFondo(ImOriginal);
```

En la siguiente imagen se muestra el resultado de aplicar la función sobre la imagen número 0008. Hay que mencionar que los ajustes de la umbralización se han realizado sobre el histograma de la imagen número 0003.

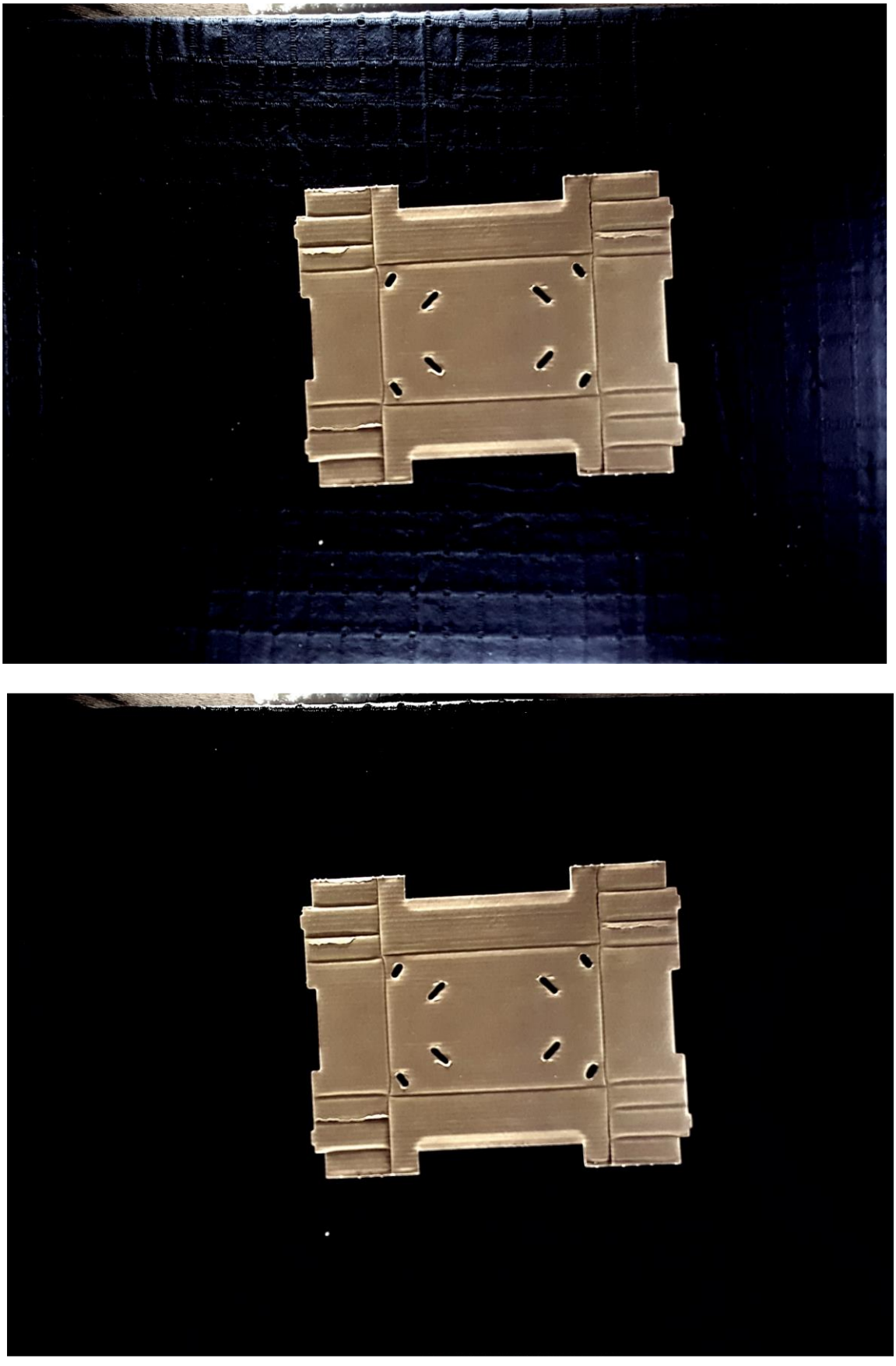


Figura 102 Antes y despés de aplicar la función de umbralización sobre la imágen numero 0008

4.3.2. Recorte.

Para eliminar las zonas que quedan en los bordes de la imagen, creamos una función para recortar la imagen a los extremos de la plancha. Consiste en detectar las zonas con aglomeraciones de pixeles y clasificarlas según tamaño. Al final se extrae el '*Blob*' de mayor tamaño que representará los píxeles de la plancha y eliminamos el resto.

En primer lugar, creamos la imagen binaria. Esta función sustituye todos los pixeles de la imagen de entrada con luminancia mayor que 0.1 con el valor 1 (blanco) y sustituye todos los demás pixeles por el valor 0 (negro).

```
42.   ImagenBinaria = im2bw(ImSinFondo,0.1); %ajustar y Generar la imagen
      binaria
```

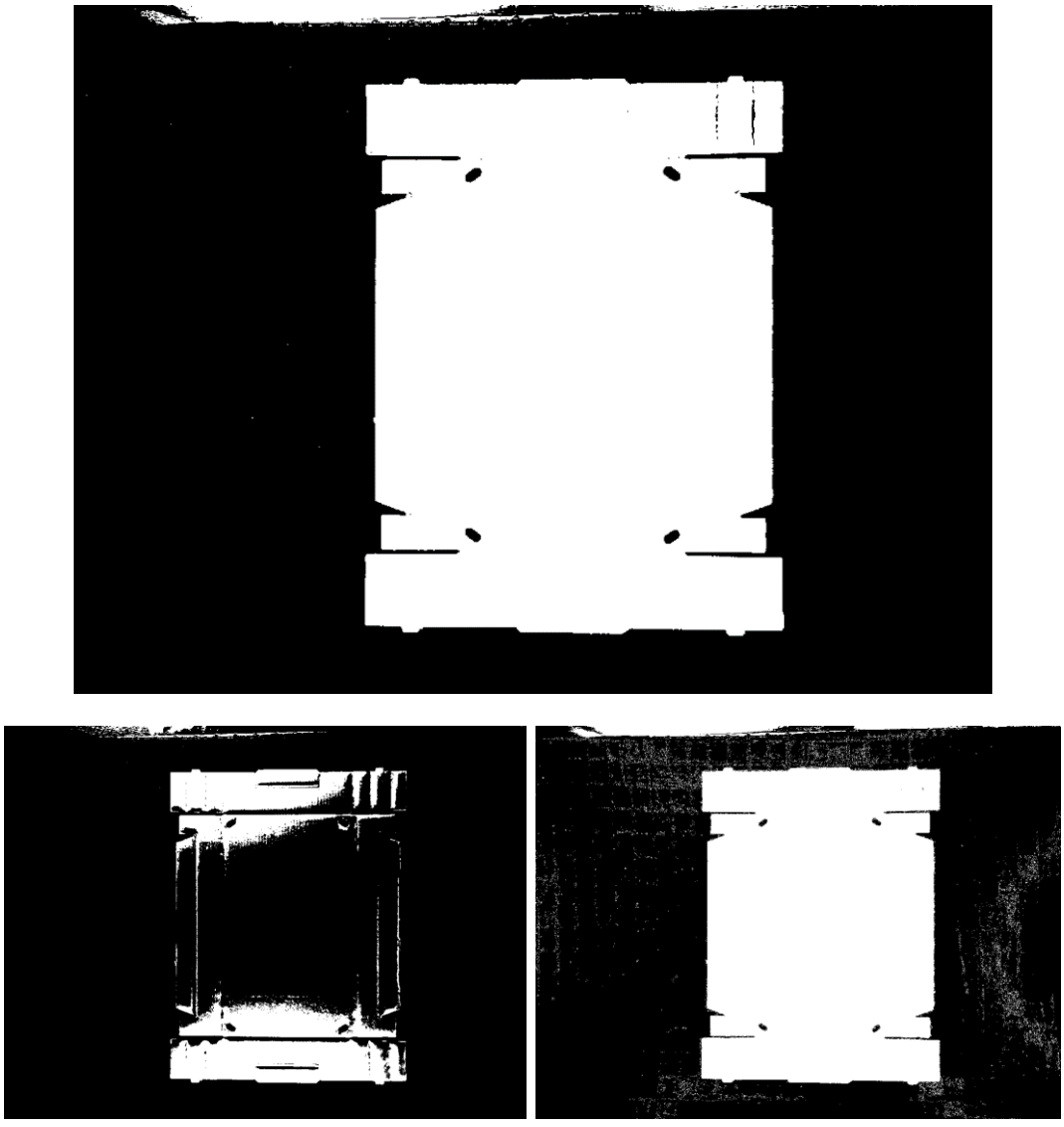


Figura 103 Imagen binaria según diferentes valores de luminancia (0.1 arriba – 0.8 izquierda – 0 derecha)

```
function [SoloPlancha,Coordinadas] = ExtraerPlancha(ImagenBinaria,
CantidadEtiqExtraer)

    % Etiquetar y Obtener las propiedades de las zonas etiquetadas
    [ImEtiquetada, numeroEtiq] = bwlabel(ImagenBinaria);

    % obtener las areas de cada etiqueta
    AreaEtiq = regionprops(ImEtiquetada, 'area');
    TAreas = [AreaEtiq.Area];

    % ordenar las areas de mayor a menor
    [OrdenArea, IndiceArea] = sort(TAreas, 'descend');

    % extraer la etiqueta más grande
    MayorEtiq = ismember(ImEtiquetada,IndiceArea(1:CantidadEtiqExtraer));

    % convertir de integer a binary (logical).
    ImagenBinaria = MayorEtiq > 0;

    %Recortar la imagen
    ImEtiquetada2 = bwlabel(ImagenBinaria);
    Coordinadas = regionprops(ImEtiquetada2, 'BoundingBox');

    % encontrar el cuadro delimitador obteniendo la lista de pixeles en el blob.
    Cdelimitador = Coordinadas.BoundingBox;

    % recortar la imagen al cuadro delimitador
    SoloPlancha = imcrop(ImagenBinaria, Cdelimitador);

 end
```

La primera tarea de la función consiste en etiquetar y contar las zonas con píxeles conectados en la imagen binaria mediante la función '*bwlabel*'. Luego, se calculan las áreas mediante la función *'regionprops'* y se ordenan de mayor a menor. En la siguiente figura se muestran los valores de las áreas sobre la imagen.

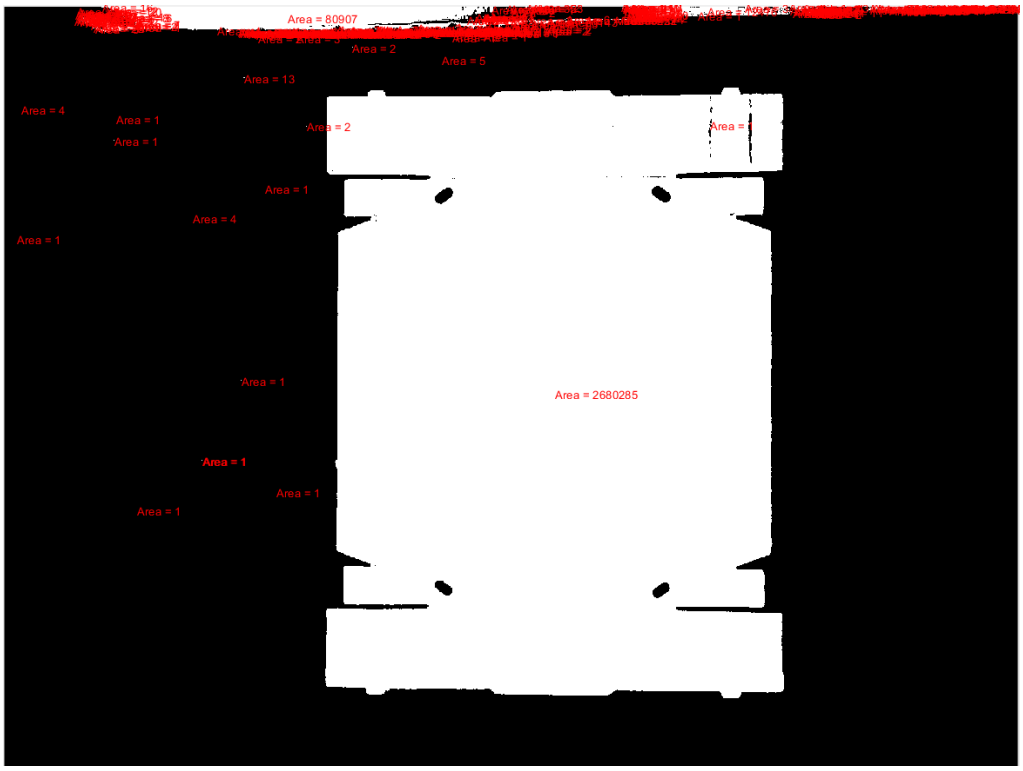


Figura 104 Áreas de los blobs de la imagen

OrdenArea | AreaEtiq

303x1 struct with 1 field

Fields	Area
1	1
2	4
3	2
4	2
5	2

OrdenArea | AreaEtiq | TAreas

1x303 double

	1	2	3	4
1	1	4	2	2

OrdenArea

1x303 double

	1	2	3	4
1	2680285	80907	12476	1108

Figura 105 valores calculados de las áreas de cada blob ordenados de mayor a menor

Como se puede observar en las imágenes anteriores, el área más grande corresponde al blob de la plancha. Por lo tanto, en la tarea siguiente se conserva solo el primer elemento en el listado de las áreas y se elimina el resto. En la siguiente figura, se muestra la imagen resultante de esta tarea.

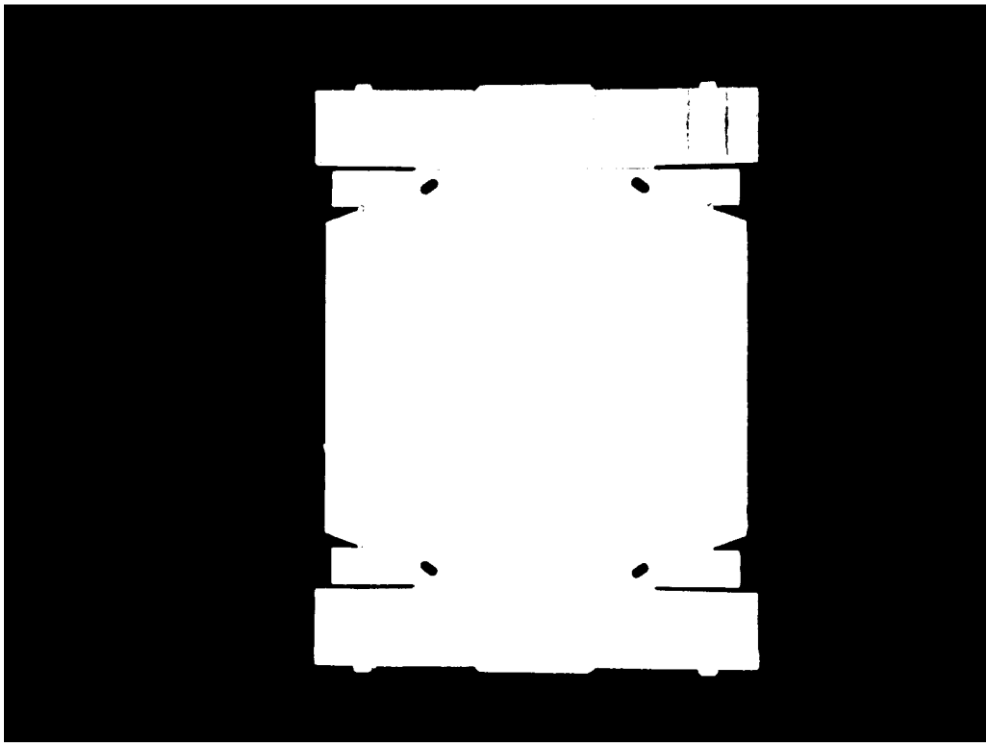
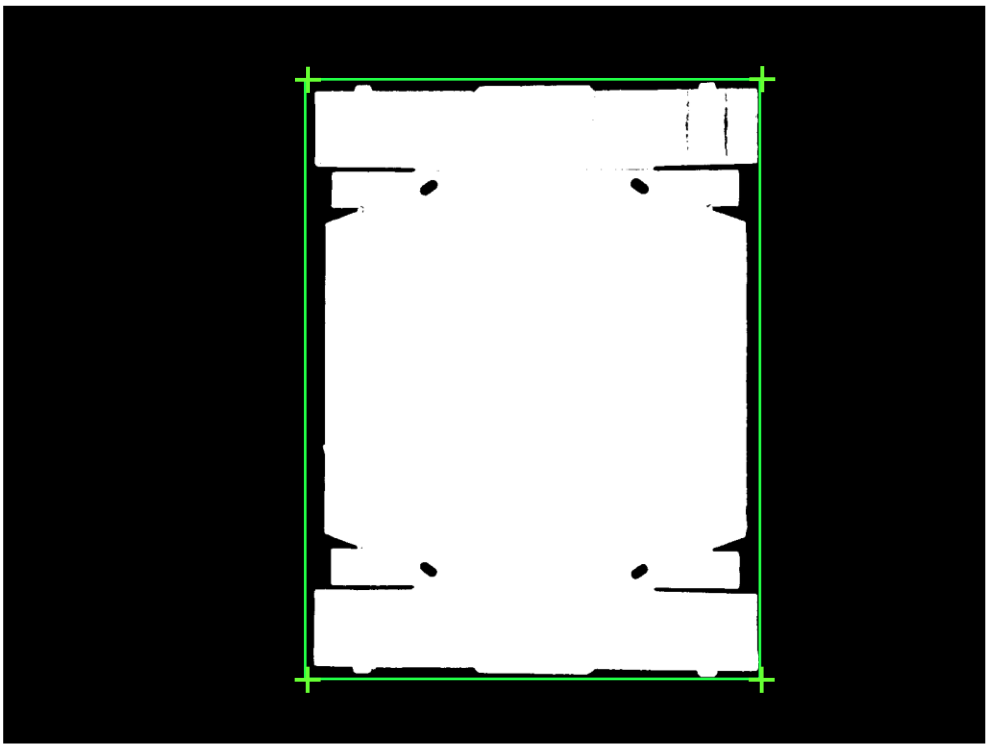


Figura 106 Imagen con el blob de mayor area (MayorEtiq)

A continuación, se procede a crear un cuadro delimitador sobre el blob de la plancha para recortarlo. Para ello se vuelve a etiquetar la imagen obtenida en el paso anterior con la función *'bwlabel'* y se calculan las coordenadas del cuadro delimitador alrededor de la única área en la imagen que corresponde a la plancha.



Coordinadas | Coordinadas.BoundingBox

Coordinadas.BoundingBox

	1	2	3	4
1	1.0355e+03	261.5000	1477	1970

Figura 107 Coordinadas de los 4 extremos del cuadro delimitador

Finalmente, se recorta la imagen utilizando la función *'imcrop'* y las coordenadas del cuadro delimitador.

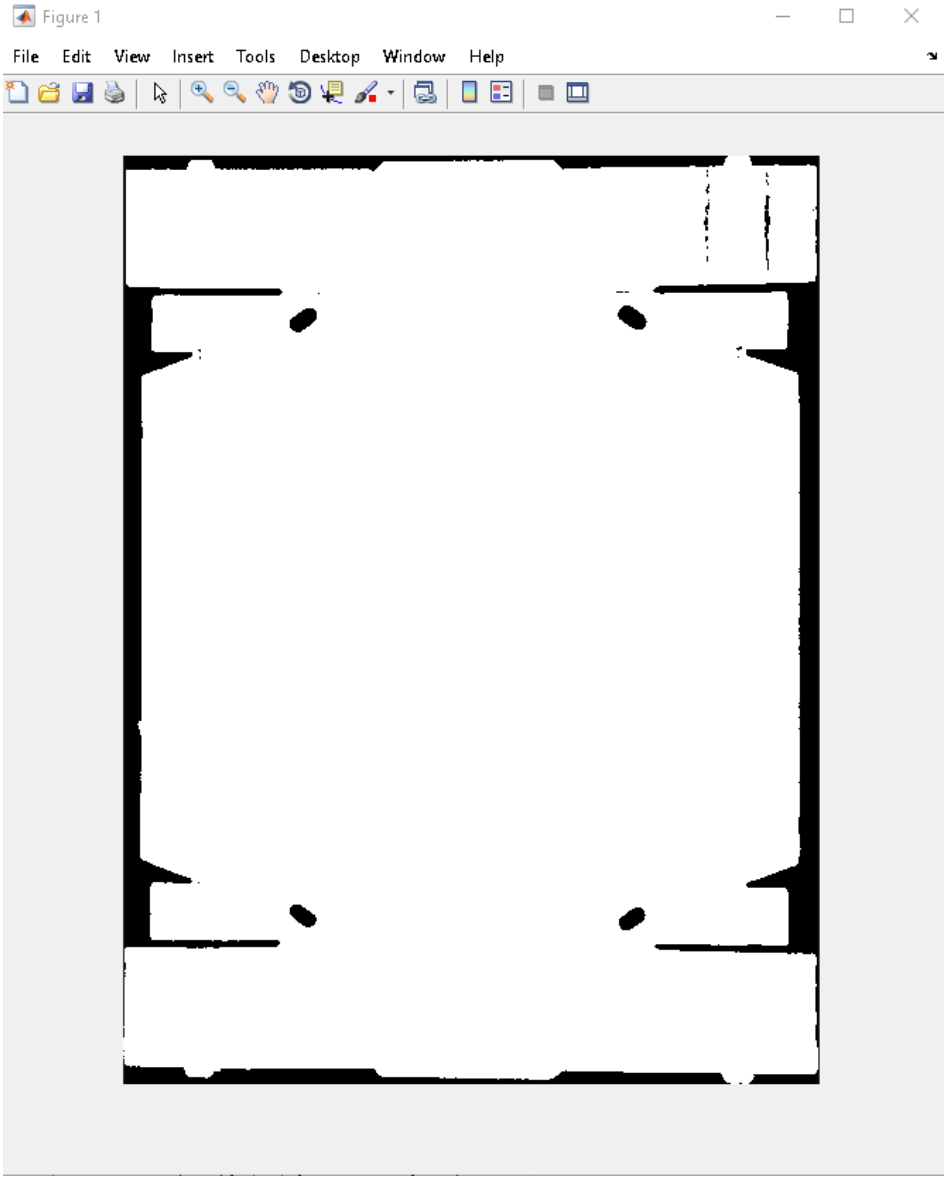


Figura 108 Imagen binaria recortada

Llamamos a esta función desde el bucle *'While'* del algoritmo de preprocesamiento que estamos desarrollando y utilizamos las mismas coordenadas del cuadro delimitados de cada imagen binaria para recortar su correspondiente imagen RGB.

```
43.   % recortar la imagen al blob de la plancha, 1 ctd de blobs a extraer
44.    [imagenBlob,pixel] = ExtraerPlancha(ImagenBinaria, 1);
45.
46.   %usar el mismo cuadro delimitador para recortar la imagen en color
47.    ImRecortada = imcrop(ImSinFondo,pixel.BoundingBox);
```

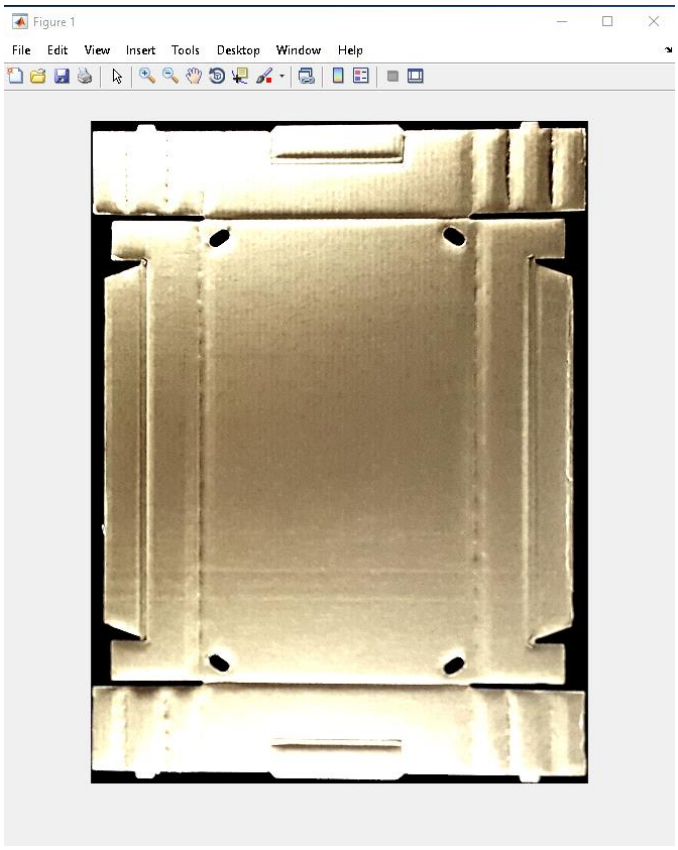


Figura 109 Imagen RGB recortada

4.3.3. Redimensión.

Debido a que las planchas en las imágenes tienen diferente tamaño y orientación, las imágenes obtenidas hasta ahora tendrán diferentes dimensiones y relaciones de aspecto. Estas imágenes no pueden ser proporcionadas así a la red neuronal ya que deben tener toda la misma dimensión exigida por la red.

En primer lugar, obtenemos las dimensiones de las imágenes recortadas.

```
48.   [Altura2,Ancho2,z2] = size(ImRecortada); % Obtener las dimensiones de la
      imagen recortada
```

Luego, creamos el código necesario para unificar la relación de aspecto de las imágenes. Teniendo como objetivo la relación de aspecto 16:9, realizamos una regla de tres para calcular el ancho necesario en cada imagen sabiendo su altura. Utilizamos los valores de la dimensión auxiliar 1920 x 1080 como constantes para el cálculo.

Altura plancha → Altura auxiliar
Ancho Plancha → Ancho auxiliar

$$Ancho\ Plancha = \frac{Altura\ Plancha \cdot\ Ancho\ Auxiliar}{Altura\ Auxiliar} = Altura\ Plancha \cdot \frac{1920}{1080}$$ [8]

A partir del resultado anterior, rellenamos con color negro los dos lados de la plancha hasta conseguir el ancho necesario.

```
49.   % rellenar con negro hasta conseguir la relación de aspecto 16:9
50.     AnchoObjetivo=1920;
51.     AltoObjetivo=1080;
52.
53.   % el ancho equivalente % regla de tres para una relacion 16:9
54.     AnchoEq=(Altura2*AnchoObjetivo)/AltoObjetivo;
55.
56.   % margen a rellenar a cada lado de la imagen redondeado a un número
      entero double
57.     Marg=round((AnchoEq-Ancho2)/2);
58.
59.   % rellenar solo en un eje de la imagen
60.     ImRell = padarray(ImRecortada,[0 Marg],'both');
```

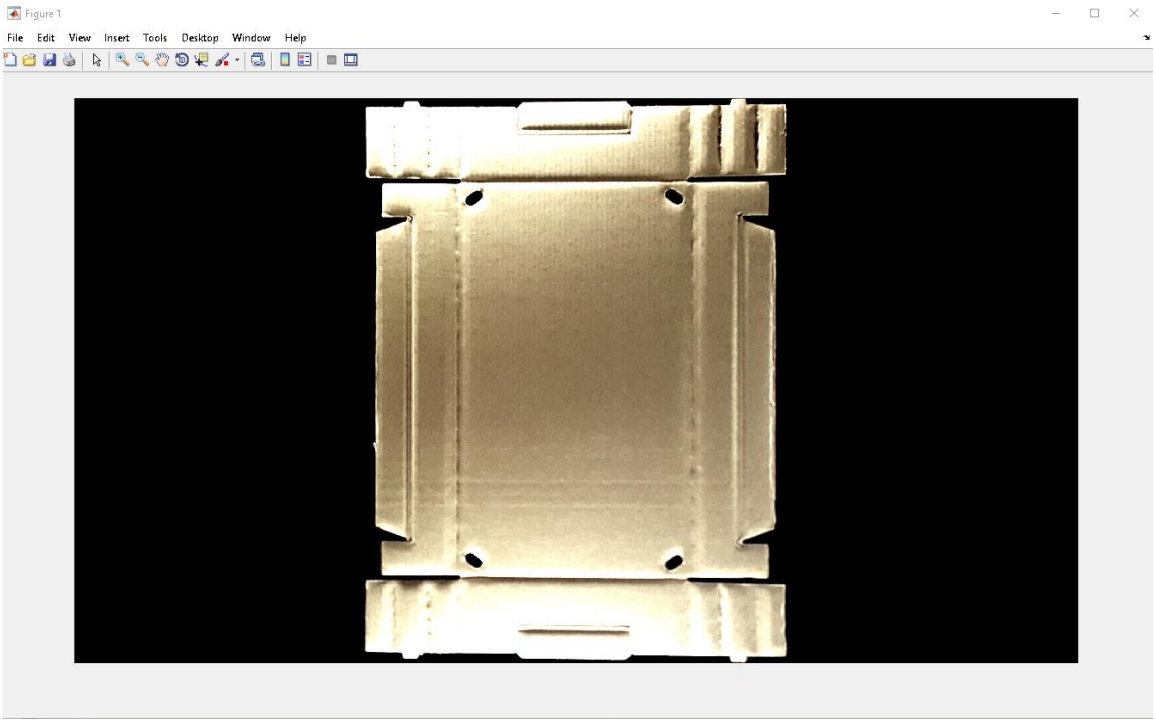


Figura 110 Imagen rellenada

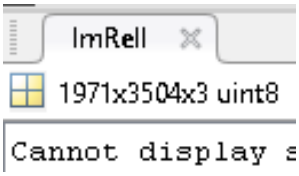


Figura 111 Dimensión de la imagen rellenada

$$\frac{3504}{1971} = \frac{16}{9} = 1{,}777..8 \qquad [9]$$

Las imágenes obtenidas hasta ahora tienen la misma relación de aspecto, pero diferente dimensión. A continuación, redimensionamos todas las imágenes para tener la misma dimensión. Para ahorrar memoria y acelerar el entrenamiento de la red neuronal, elegimos una dimensión de 640 x 360 (un tercio de la dimensión 1920*1080).

```
61.   ImRed = imresize(ImRell, [360 640]); % redimensionar hasta un tercio de
      la dimension 1920x1080
```

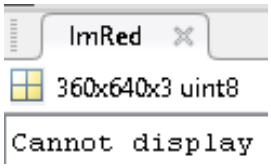


Figura 112 Dimensión definitiva de las imágenes

Para cerrar el bucle *'While'*, guardamos todas las imágenes preprocesadas en formato ".png" en la carpeta de destino conforme se vayan procesando. Añadimos la letra "R" al nombre de las imágenes para distinguirlas de las imágenes originales.

```
62.   % guardar las imagenes en la carpeta de destino
63.    imwrite(ImRed,[RutaDestino '\' 'R' NImagen ],'png','Comment','R')
64.
65.          a= a+1;
66.   end
```

Figura 113 Vista previa de algunas imágenes preprocesadas en la carpeta de destino

Capítulo 5

5. Segmentación semántica

5.1. Introducción

En el presente capítulo se comentan los pasos seguidos para realizar una segmentación semántica sobre las imágenes obtenidas hasta ahora. En primer lugar, se etiquetan las diferentes características de los dos modelos de planchas y se prepara la base de datos para el entrenamiento. Después, se crea la red neuronal SegNet a partir de una red VGG-16 y se configuran sus parámetros para el entrenamiento.

5.2. Etiquetado

Los dos modelos de planchas elegidos contienen diferentes partes que representan a los elementos funcionales de la caja. Los nombres asignados a estas partes fueron elegidos por los fabricantes de las planchas y los fabricantes de las máquinas montadoras para unificar el vocabulario utilizado a la hora de describir los diferentes modelos de planchas/cajas.

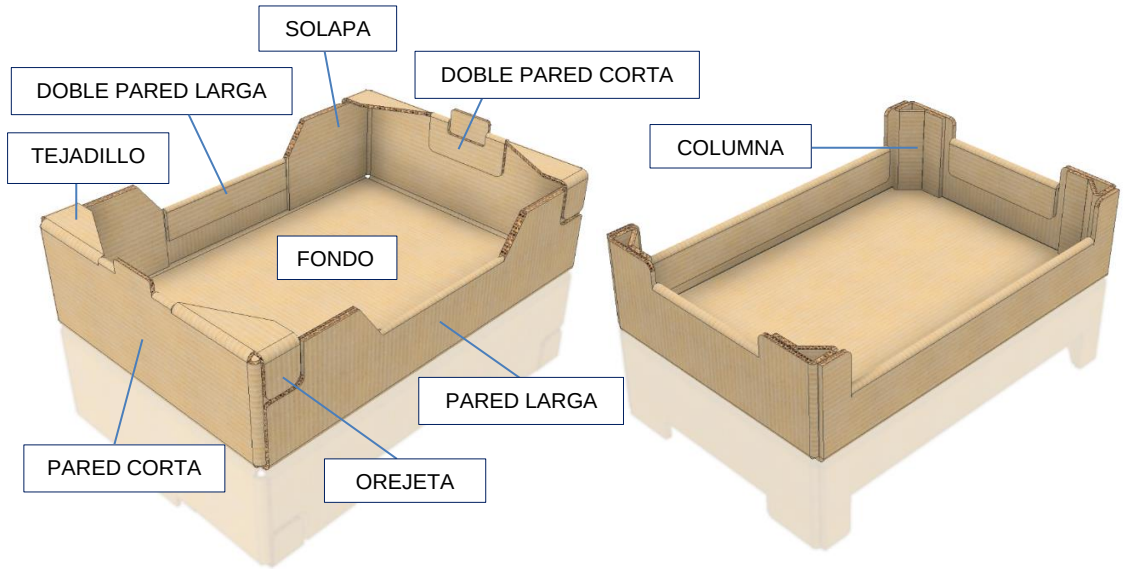


Figura 114 Características de los modelos de cajas

A continuación, se describen las funciones de cada una de las partes de la caja:

- **Fondo**: Es la base de la caja y el elemento que sostiene el producto.
- **Pared corta**: Es el lateral más corto que determina el ancho de la caja.
- **Pared larga**: Es el lateral más largo que determina el largo de la caja.
- **Doble pared**: Es la pestaña plegada que se añade a una o ambas paredes para darle más rigidez a la caja.
- **Solapa**: Es la pestaña plegada que se utiliza para unir la pared corta con la pared larga de la caja.
- **Tejadillo**: Es la pestaña plegada y paralela al fondo de la caja que se añade a la parte superior de las paredes cortas para mejorar el apoyo de las cajas apiladas encima y minimizar la flexión de la base debido al peso del producto.
- **Orejeta**: Es la pestaña plegada que se utiliza para unir el tejadillo con la pared larga.
- **Columna**: Es la parte de la plancha que se pliega para formar una columna en cada esquina de la caja para conseguir la rigidez necesaria para aguantar el peso del producto.

La doble pared se añade a la caja solo cuando existe el riesgo de flexión en las paredes de la caja debido a la naturaleza del producto envasado o la calidad del cartón. En los casos normales las cajas son simples con el objetivo de ahorrar cartón y cola y simplificar el proceso de montaje al no necesitar los mecanismos especiales para formar la doble pared.

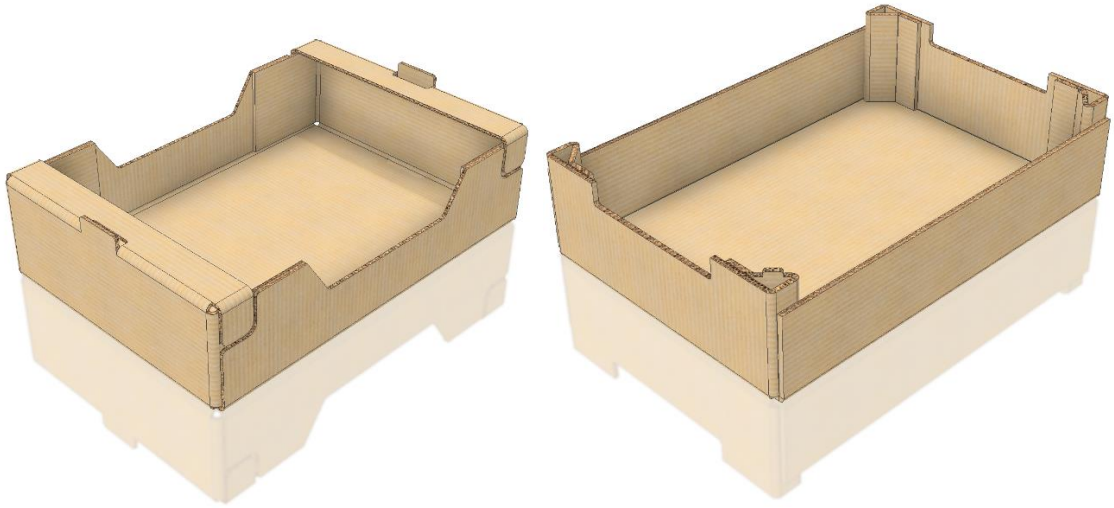


Figura 115 Cajas de Plaform y Columna sin doble pared

Las características mencionadas anteriormente se pueden categorizar en 3 pequeños grupos. Estos grupos nos permiten identificar más rápido el modelo de caja:

Características comunes

- Fondo.
- Pared Corta.
- Pared Larga.
- Doble pared (Pared corta).
- Doble pared (Pared larga).

Características específicas de Plaform

- Orejeta
- Solapa
- Tejadillo

Características específicas de Columna

- Columna

Para identificar las características, se etiquetan con 9 etiquetas diferentes:

1- Fondo.
2- Orejeta.
3- Pared Corta.
4- Pared Larga.
5- Doble pared (Pared corta).
6- Doble pared (Pared larga).
7- Solapa.
8- Tejadillo.
9- Columna.

En los siguientes pasos se explica cómo asociar cada etiqueta con la zona correspondiente de la plancha en las imágenes obtenidas utilizando las herramientas de Matlab.

Utilizando la aplicación *Image Labler* de *Image Processing and computer vision Toolbox* de Matlab (R2018a), empezamos el proceso para etiquetar las imágenes.

Figura 116 Image Labeler de Matlab

El primer paso consiste en crear las etiquetas con los nombres de cada característica y asignarle un color determinado que se utilizará para delimitar las regiones de interés (ROI).

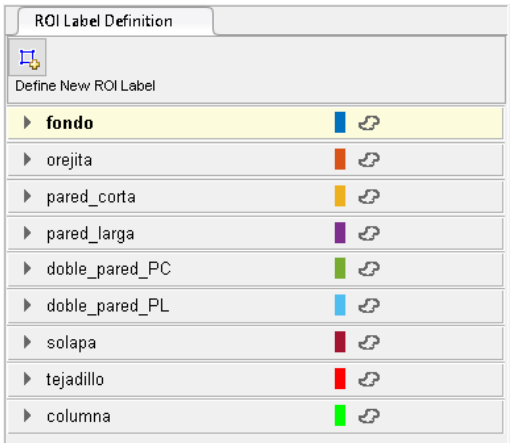


Figura 117 Creación de etiquetas y asignación de colores

El siguiente paso consiste en cargar las imágenes preprocesadas a la aplicación. Ya que las imágenes se encuentran guardadas en un directorio, elegimos la opción *Add images from folder.*

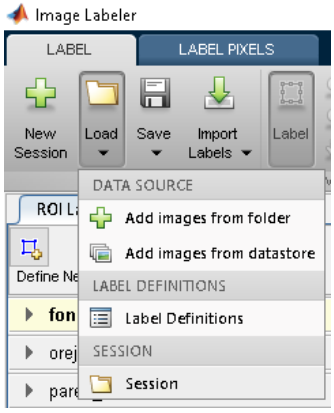


Figura 118 Carga de imágenes a la aplicación

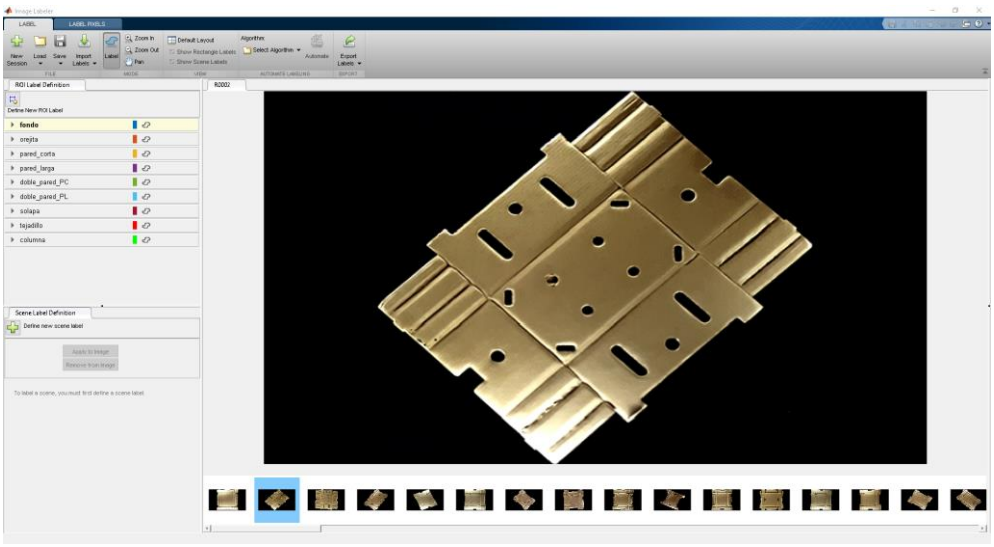


Figura 119 Imágenes cargadas

A continuación, empezamos a etiquetar las características de las planchas delimitando las regiones de interés con el color correspondiente. Este proceso consume mucho tiempo al tener que dibujar manualmente el contorno alrededor de cada región de todas las imágenes. Para reducir el tiempo total, se utiliza la herramienta de polígono.

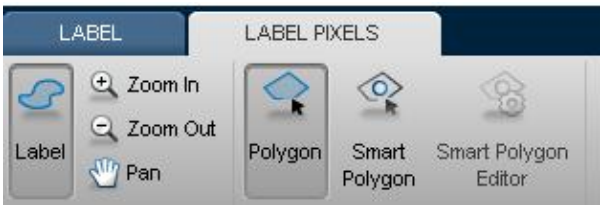


Figura 120 Herramienta de polígono

Esta opción permite marcar únicamente los extremos de cada región para crear un lazo cerrado y rellenar automáticamente con el color de la etiqueta correspondiente.

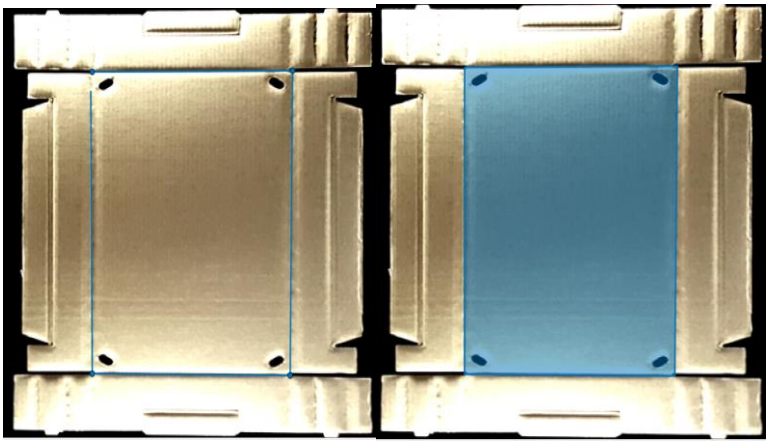


Figura 121 Utilización de la herramienta de polígono

Para mayor precisión y comodidad a la hora de marcar las regiones de interés, se ha utilizado una tableta gráfica. Al utilizar esta herramienta, el tiempo necesario para etiquetar una plancha modelo Plaform ha disminuido en comparación con el uso del ratón, de 4:50 minutos a 2:33 minutos ahorrando más de 2:17 minutos por plancha. Si multiplicamos este tiempo por la cantidad de imágenes a etiquetar ahorraríamos un total de 239 minutos que equivalen aproximadamente a 4 horas de trabajo.

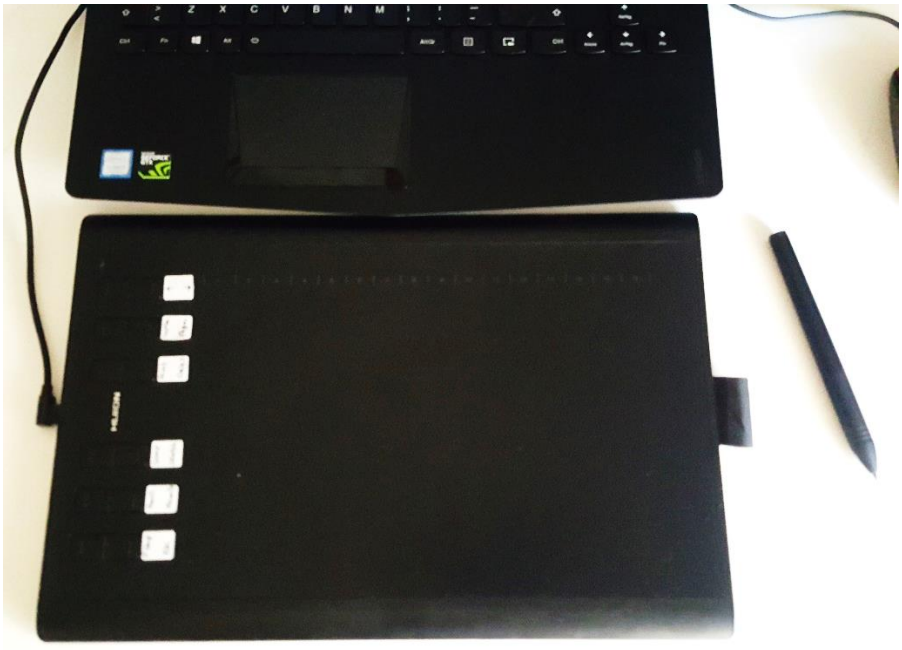


Figura 122 Tableta gráfica

Para dibujar los polígonos, se han trazado las líneas justo en el centro de las sombras cuando estas delimitan dos características diferentes para conseguir que cada característica aproveche parte de la sombra. En cuanto al contorno exterior, intentamos dejar los cortes dentro del área de etiquetado.

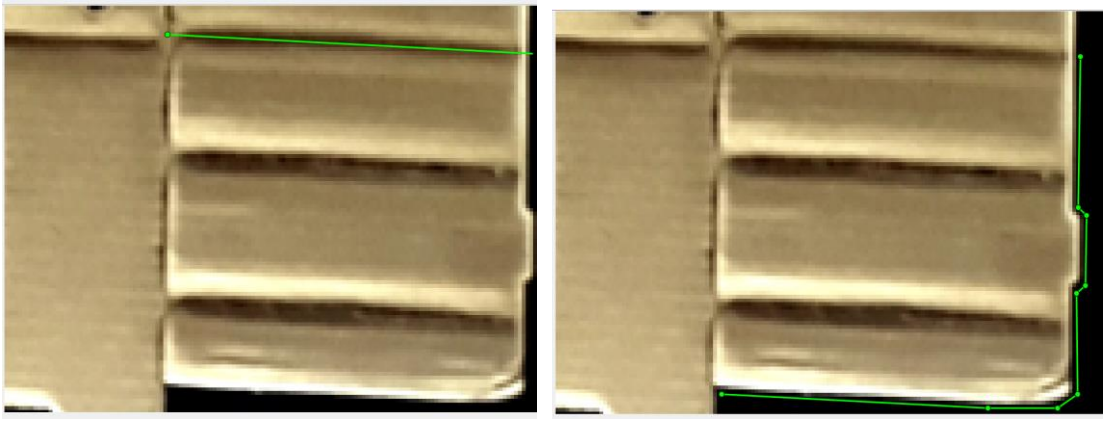


Figura 123 Líneas de trazado del polígono.

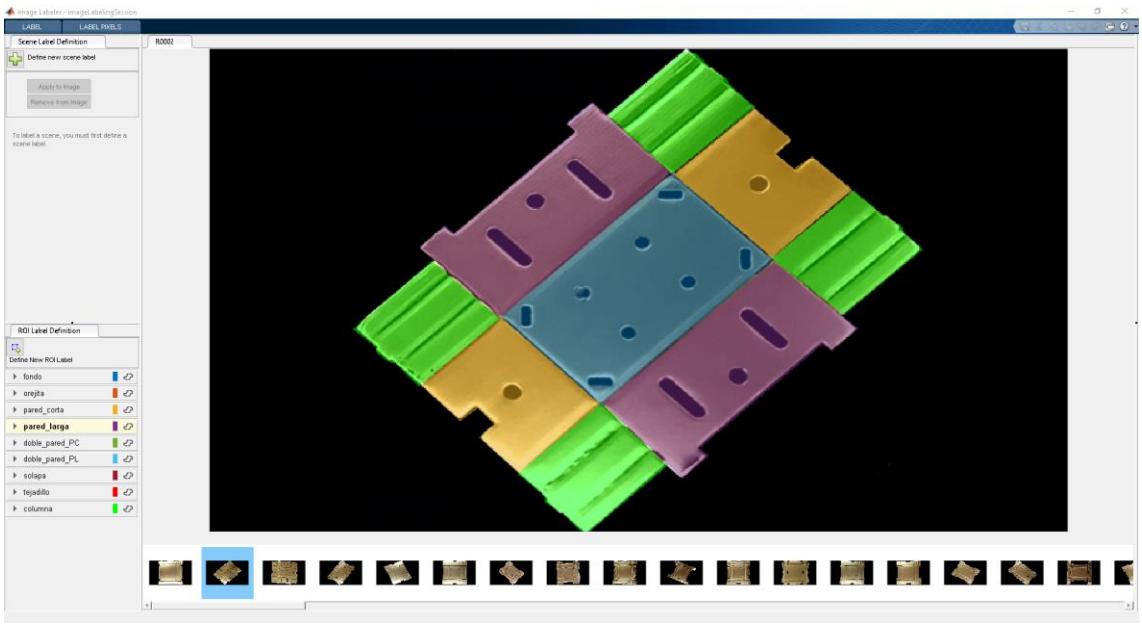


Figura 124 Ejemplo de plancha modelo Columna etiquetada

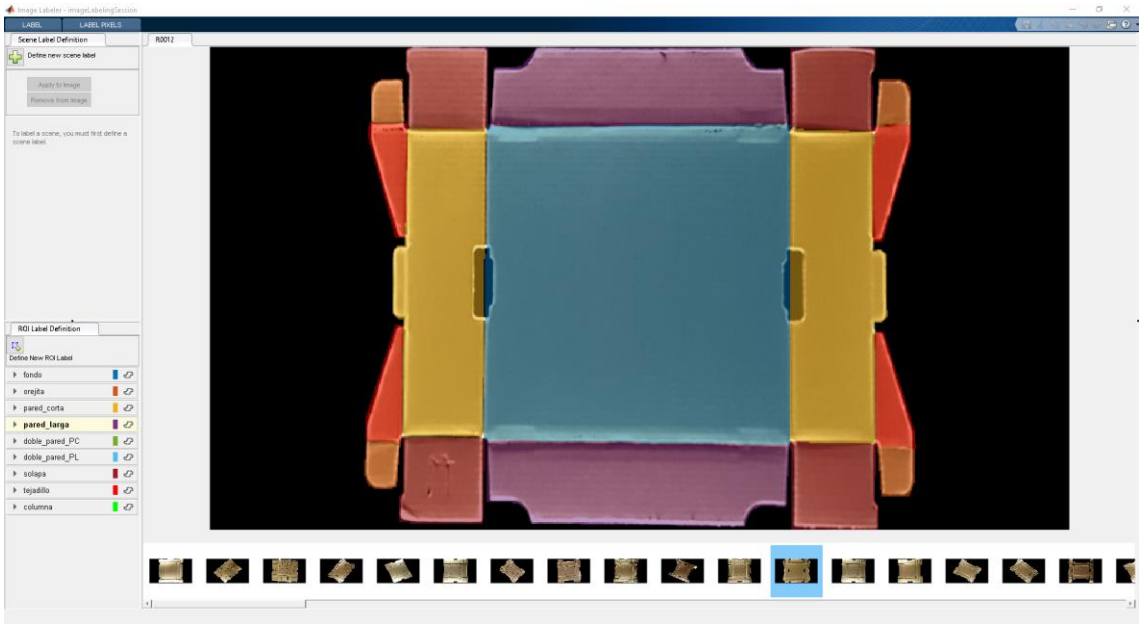


Figura 125 Ejemplo de plancha modelo Plaform etiquetada

La información generada por la app de etiquetado *Image Labeler* se almacena en un archivo tipo *GroundTruth* compuesto por las siguientes secciones:

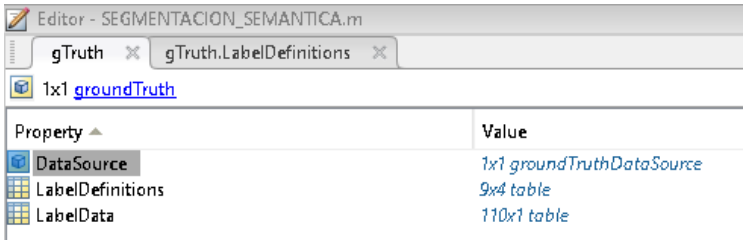
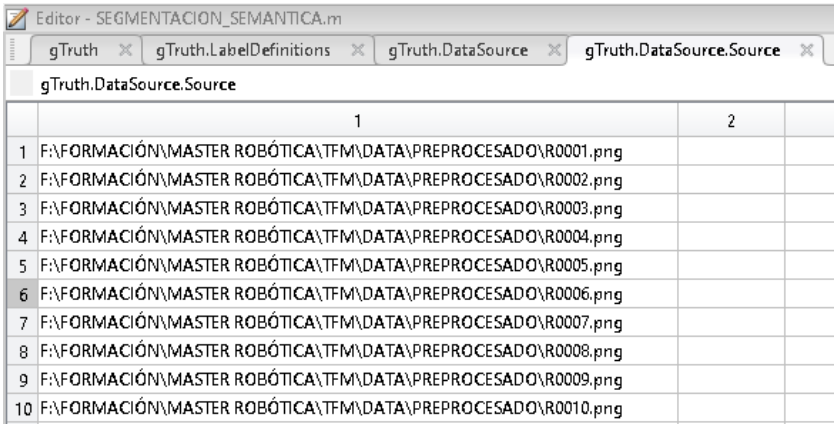


Figura 126 GroundTruth

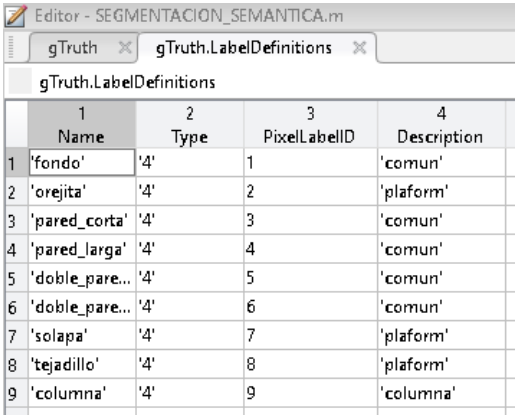
- **DataSource**: compuesto por dos vectores; El vector *Source* que contiene las rutas de las imágenes originales. Y el vector *TimeStamps* que almacena la secuencia de tiempos de captura de las imágenes en el caso de videos.



	1	2
1	F:\FORMACIÓN\MASTER ROBÓTICA\TFM\DATA\PREPROCESADO\R0001.png	
2	F:\FORMACIÓN\MASTER ROBÓTICA\TFM\DATA\PREPROCESADO\R0002.png	
3	F:\FORMACIÓN\MASTER ROBÓTICA\TFM\DATA\PREPROCESADO\R0003.png	
4	F:\FORMACIÓN\MASTER ROBÓTICA\TFM\DATA\PREPROCESADO\R0004.png	
5	F:\FORMACIÓN\MASTER ROBÓTICA\TFM\DATA\PREPROCESADO\R0005.png	
6	F:\FORMACIÓN\MASTER ROBÓTICA\TFM\DATA\PREPROCESADO\R0006.png	
7	F:\FORMACIÓN\MASTER ROBÓTICA\TFM\DATA\PREPROCESADO\R0007.png	
8	F:\FORMACIÓN\MASTER ROBÓTICA\TFM\DATA\PREPROCESADO\R0008.png	
9	F:\FORMACIÓN\MASTER ROBÓTICA\TFM\DATA\PREPROCESADO\R0009.png	
10	F:\FORMACIÓN\MASTER ROBÓTICA\TFM\DATA\PREPROCESADO\R0010.png	

Figura 127 DataSource

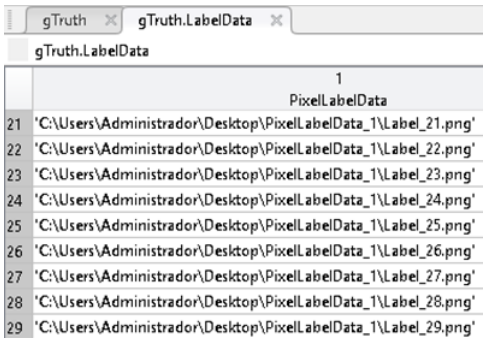
- **LabelDefinition**: Contiene el listado de las etiquetas e información relacionada.



	1 Name	2 Type	3 PixelLabelID	4 Description
1	'fondo'	'4'	1	'comun'
2	'orejita'	'4'	2	'plaform'
3	'pared_corta'	'4'	3	'comun'
4	'pared_larga'	'4'	4	'comun'
5	'doble_pare...	'4'	5	'comun'
6	'doble_pare...	'4'	6	'comun'
7	'solapa'	'4'	7	'plaform'
8	'tejadillo'	'4'	8	'plaform'
9	'columna'	'4'	9	'columna'

Figura 128 LabelDefinition

- **LabelData**: Contiene las rutas de las imágenes etiquetadas guardadas automáticamente en el disco.



	1 PixelLabelData
21	'C:\Users\Administrador\Desktop\PixelLabelData_1\Label_21.png'
22	'C:\Users\Administrador\Desktop\PixelLabelData_1\Label_22.png'
23	'C:\Users\Administrador\Desktop\PixelLabelData_1\Label_23.png'
24	'C:\Users\Administrador\Desktop\PixelLabelData_1\Label_24.png'
25	'C:\Users\Administrador\Desktop\PixelLabelData_1\Label_25.png'
26	'C:\Users\Administrador\Desktop\PixelLabelData_1\Label_26.png'
27	'C:\Users\Administrador\Desktop\PixelLabelData_1\Label_27.png'
28	'C:\Users\Administrador\Desktop\PixelLabelData_1\Label_28.png'
29	'C:\Users\Administrador\Desktop\PixelLabelData_1\Label_29.png'

Figura 129 LabelData

Debido a que el proceso de etiquetado se hizo en varias sesiones de trabajo, he tenido que guardar por una parte la sesión de trabajo para no tener que volver a cargar las imágenes y crear las etiquetas, y por otra parte exportar el archivo *GroundTruth* que contiene toda la información del proceso de etiquetado, para poder cerrar Matlab y apagar el ordenador.

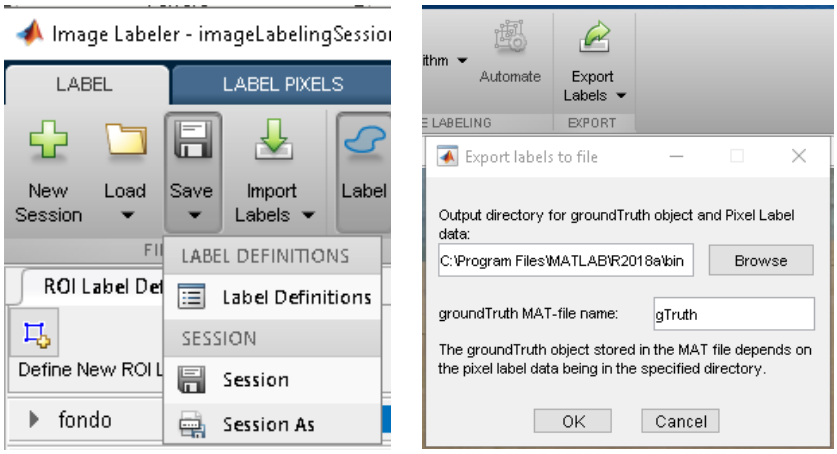


Figura 130 Guardar sesión de trabajo y exportar las etiquetas

Al volver a abrir la información guardada anteriormente para continuar el proceso de etiquetado, Matlab devuelve una ventana de error desconocido, y carga el archivo *GroundTruth* con todas las secciones vacías (Comparar las figuras 126 y 132).

Figura 131 Error en la carga de información de etiquetado

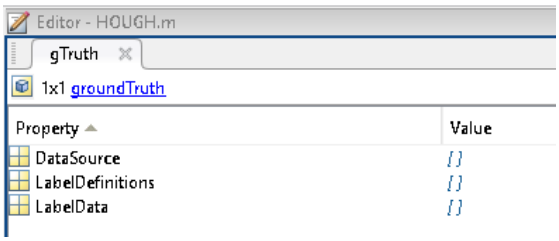


Figura 132 Archivo GroundTruth vacío

Para poder continuar el proceso de etiquetado sin perder las etiquetas, se ha decidido construir el archivo *GroundTruth* manualmente rellenando todas las secciones mencionadas anteriormente, y utilizar las imágenes etiquetadas

generadas por la aplicación *Image Labeler.*

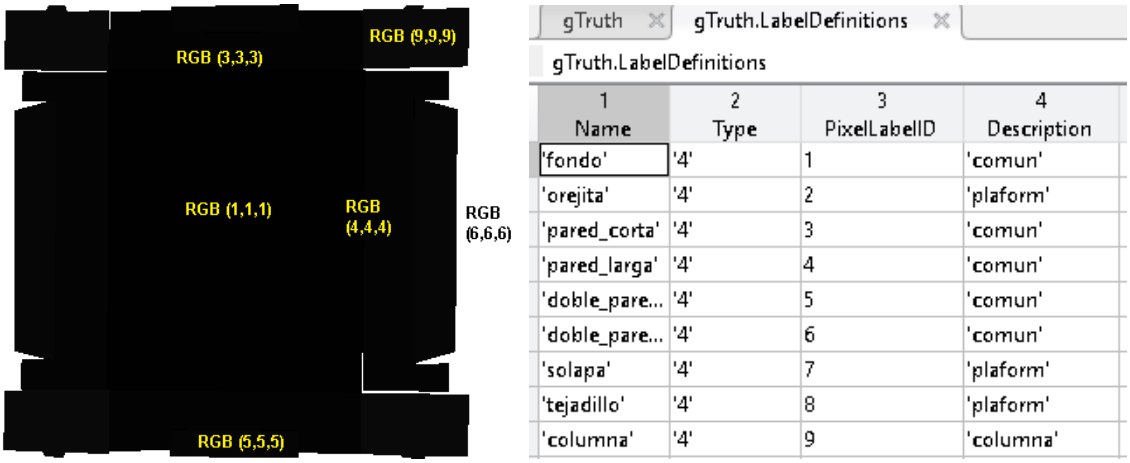
Al abrir alguna imagen etiquetada generada por la aplicación no se aprecia ninguna etiqueta o color que habíamos marcado.

Figura 133 Imagen etiquetada generada por la app

Al observar la inversa de estas imágenes, se descubre que la aplicación convirtió los colores de las etiquetas en escalas de grises dando a cada una de ellas un color con valores RGB iguales a su ID correspondiente en la tabla de *PixelLabelID.*

Figura 134 Inversa de una imagen etiquetada

Con el ayuda del programa *Paint* podemos ver los valores RGB de estos colores. Para facilitar la visualización de la plancha, se ha rellenado el fondo de la imagen con color blanco.



1 Name	2 Type	3 PixelLabelID	4 Description
'fondo'	'4'	1	'comun'
'orejita'	'4'	2	'plaform'
'pared_corta'	'4'	3	'comun'
'pared_larga'	'4'	4	'comun'
'doble_pare...	'4'	5	'comun'
'doble_pare...	'4'	6	'comun'
'solapa'	'4'	7	'plaform'
'tejadillo'	'4'	8	'plaform'
'columna'	'4'	9	'columna'

Figura 135 Valores RGB de una imagen etiquetada y su correspondiente Label ID

Estas imágenes etiquetadas no las podemos utilizar directamente para entrenar la red neuronal ya que no presentan el contraste adecuado para distinguir entre las diferentes etiquetadas.

A continuación, creamos una función para reemplazar cada uno de los colores de escala de grises con un color nuevo que va a representar a la etiqueta en el algoritmo de entrenamiento.

Etiqueta	Label ID	Colores en escala de grises	Nuevos colores RGB
Fondo	1	1-1-1	0-255-255
Orejita	2	2-2-2	255-128-0
Pared corta	3	3-3-3	255-0-255
Pared larga	4	4-4-4	0-255-0
Doble pared PC	5	5-5-5	255-0-128
Doble pared PL	6	6-6-6	128-0-255
Solapa	7	7-7-7	255-0-0
Tejadillo	8	8-8-8	0-0-255
Columna	9	9-9-9	255-255-0
Entorno	10	0-0-0	255-255-255

Tabla 7 Valores RGB de los colores reemplazados

La función abrirá automáticamente cada una de las imágenes etiquetadas para reemplazar los colores y volverá a guardarlas en un directorio nuevo.

```
clc
clear all

DirEtiqOr= fullfile(pwd,'ALGORITHM','IMAGE LABELER','PixelLabelData_1'); % directorio etiquetas con colores grises

DirEtiqOrCol= fullfile(pwd,'DATA','ETIQUETADO'); % directorio etiquetas con colores reemplazados

DrEt= [DirEtiqOr '\'];

lr_DrEt= ls(DrEt); %primer elemento

CEt= cellstr(lr_DrEt); % tabla

CCEt= CEt(3:length(CEt));

SEt= size(CCEt); %dimensión de la tabla
a= 1;

% Imagenes
while a <= SEt(1)
    close all

    ArchEt= char(cellstr([DrEt char(CCEt(a))]));

    datos_Et= char(CCEt(a));

    NombreEt= char(CCEt(a));

    NombEt = (NombreEt);

    LeerEt = [DirEtiqOr '\' NombEt];

    OrEt = imread(LeerEt);

    [AltoEt,LargoEt,z1Et] = size(OrEt);

   Gray= OrEt;

RGB = Gray(:,:,[1 1 1]);
[r c z]=size(RGB);

  for i=1:r
      for j=1:c

   if     (RGB(i,j,1)==1 && RGB(i,j,2)==1 && RGB(i,j,3)==1) % Fondo
          RGB(i,j,1)=0; RGB(i,j,2)=255; RGB(i,j,3)=255; % reemplazar color

   elseif(RGB(i,j,1)==2 && RGB(i,j,2)==2 && RGB(i,j,3)==2) % Orejita
          RGB(i,j,1)=255; RGB(i,j,2)=128; RGB(i,j,3)=0;

   elseif(RGB(i,j,1)==3 && RGB(i,j,2)==3 && RGB(i,j,3)==3) % Pared corta
          RGB(i,j,1)=255; RGB(i,j,2)=0; RGB(i,j,3)=255;

   elseif(RGB(i,j,1)==4 && RGB(i,j,2)==4 && RGB(i,j,3)==4) % Pared larga
          RGB(i,j,1)=0; RGB(i,j,2)=255; RGB(i,j,3)=0;

   elseif(RGB(i,j,1)==5 && RGB(i,j,2)==5 && RGB(i,j,3)==5) % Doble pared pc
          RGB(i,j,1)=255; RGB(i,j,2)=0; RGB(i,j,3)=128;

   elseif(RGB(i,j,1)==6 && RGB(i,j,2)==6 && RGB(i,j,3)==6) % Doble pared pl
          RGB(i,j,1)=128; RGB(i,j,2)=0; RGB(i,j,3)=255;

   elseif(RGB(i,j,1)==7 && RGB(i,j,2)==7 && RGB(i,j,3)==7) % Solapa
          RGB(i,j,1)=255; RGB(i,j,2)=0; RGB(i,j,3)=0;

   elseif(RGB(i,j,1)==8 && RGB(i,j,2)==8 && RGB(i,j,3)==8) % Tejadillo
          RGB(i,j,1)=0; RGB(i,j,2)=0; RGB(i,j,3)=255;

   elseif(RGB(i,j,1)==9 && RGB(i,j,2)==9 && RGB(i,j,3)==9) % Columna
          RGB(i,j,1)=255; RGB(i,j,2)=255; RGB(i,j,3)=0;

   else   RGB(i,j,1)=255; RGB(i,j,2)=255; RGB(i,j,3)=255; % Fondo Blanco

```

```
    end
       end
   end
     imwrite(RGB,[DirEtiqOrCol '\' 'COL' NombEt ],'png','Comment','colorada ETIQUETA')
%guardar las imágenes
      a= a+1;
end
```

La siguiente figura muestra un ejemplo de las imágenes después de reemplazar los colores. Estas imágenes son las que se van a utilizar en el proceso de entrenamiento.

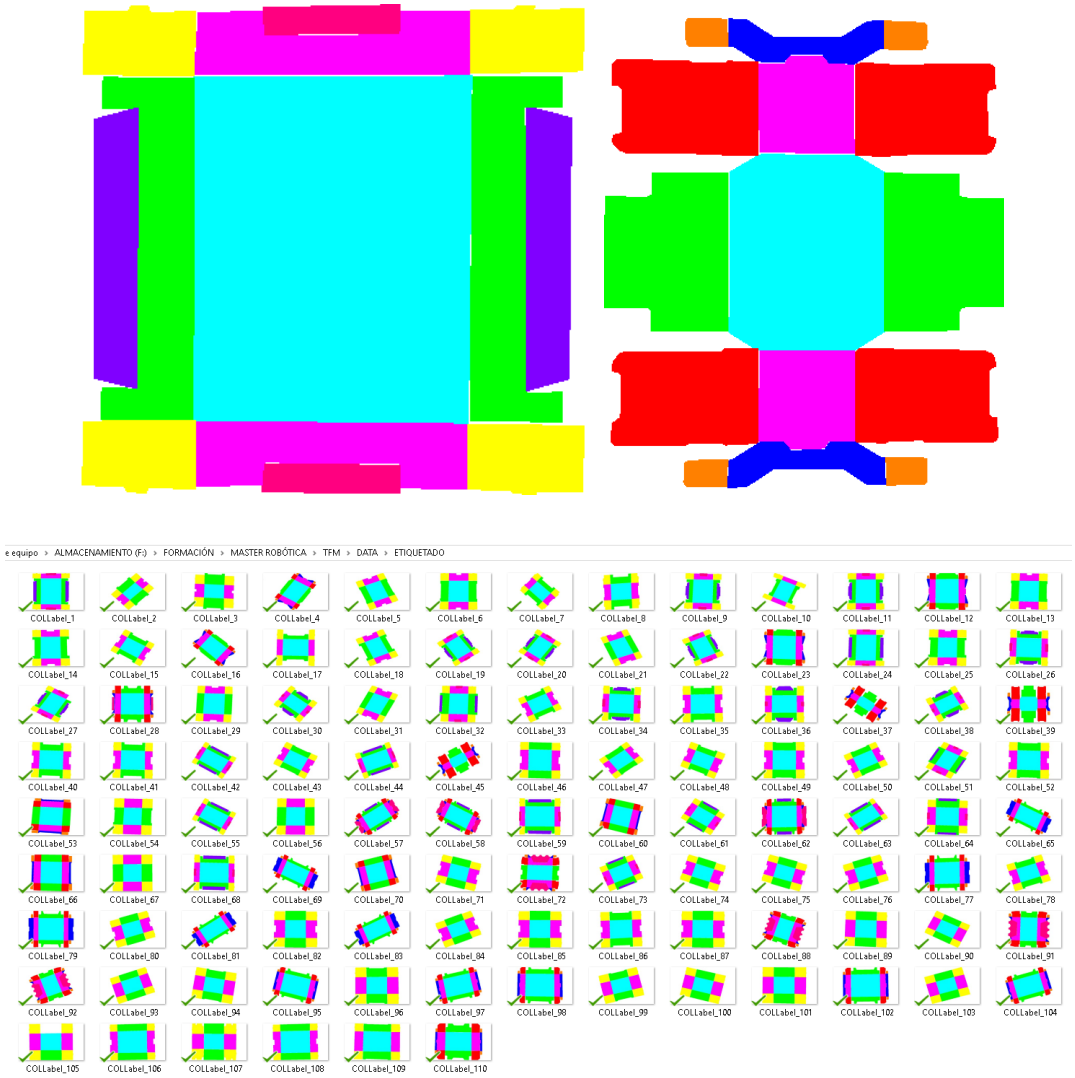


Figura 136 Imágenes etiquetadas con colores reemplazados

5.3. Entrenamiento

5.3.1. Preparación de datos.

A continuación, preparamos el algoritmo para el entrenamiento con los datos generados anteriormente.

- **Instalación de la red VGG16.**

El entrenamiento de la SegNet se basa en la red neuronal VGG-16 (del equipo de **O**xford **V**isual **G**eometry **G**roup). Esta red ha sido entrenada con más de un millón de imágenes extraídos de la base de datos ImageNet, y puede clasificar más de 1000 categorías de objetos. Como resultado, este modelo ha aprendido distinguir entre muchas características de una gran cantidad de imágenes.

Esta función requiere instalar en Matlab la Neural Network Toolbox para VGG-16.

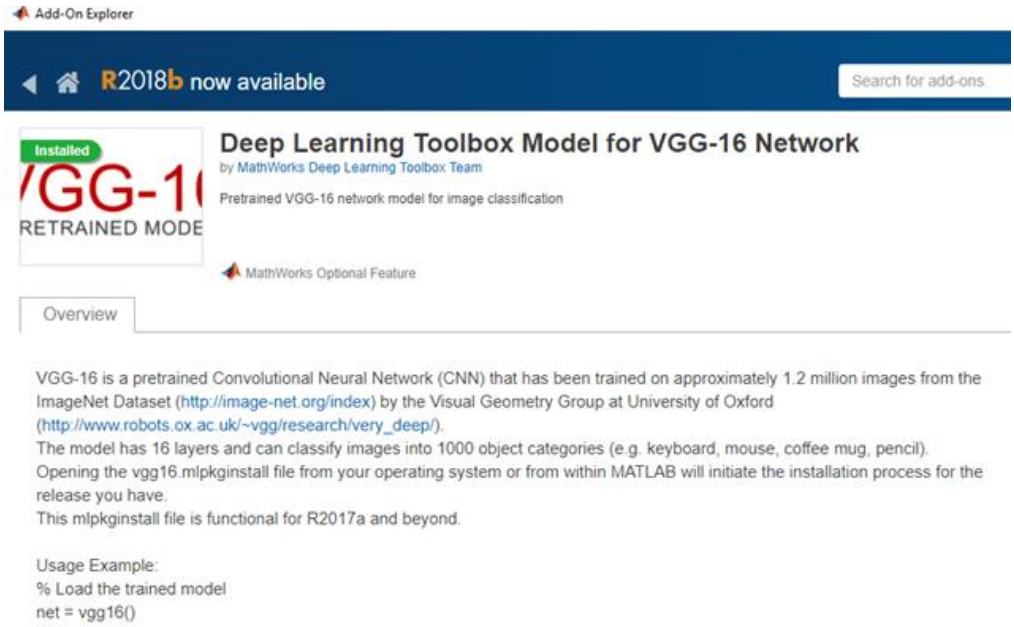


Figura 137 Instalar VGG16

```
1.  % DESCARGAR E INSTALAR EL MODULO DE NEURAL NETWORK TOOLBOX PARA LA VGG16.
2.  clc
3.  clear all;
4.  vgg16
```

- **Carga de imágenes de planchas.**

Como se comentó anteriormente, debido al error producido en el proceso de etiquetado en Matlab, hay que construir el archivo *GroundTruth* manualmente. En primer lugar, se configuran las rutas de los datos de entrenamiento.

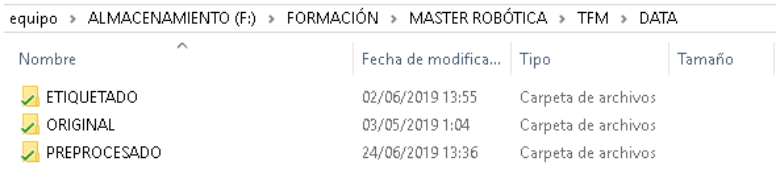


Figura 138 Rutas de datos de entrenamiento

```
%% PREPARACIÓN

DirImagR = fullfile(pwd,'DATA','PREPROCESADO'); % directorio imagenes
preprocesadas
DirEtiqR= fullfile(pwd,'DATA','ETIQUETADO'); % directorio imágenes etiquetadas
```

Usamos la función ImageDataStore para cargar las imágenes de las planchas. Esta función nos permite cargar eficientemente una gran cantidad de imágenes guardadas en el disco, y crea un almacén (*Datastore*) que contiene solo las rutas de las imágenes para cargarlas directamente cuando se necesitan sin tener que cargar todas las imágenes a la vez y ocupar mucho espacio en la memoria del programa.

```
%% CARGAR IMAGENES

ImDataStore = imageDatastore(DirImagR);

% mostrar una imagen
Im_N=100;
I_ra=readimage(ImDataStore,Im_N);
figure
imshow(I_ra)
drawnow
title('imagen número 100')
```

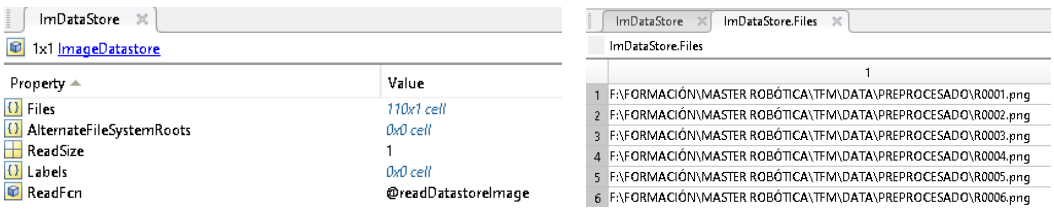


Figura 139 ImageDataStore

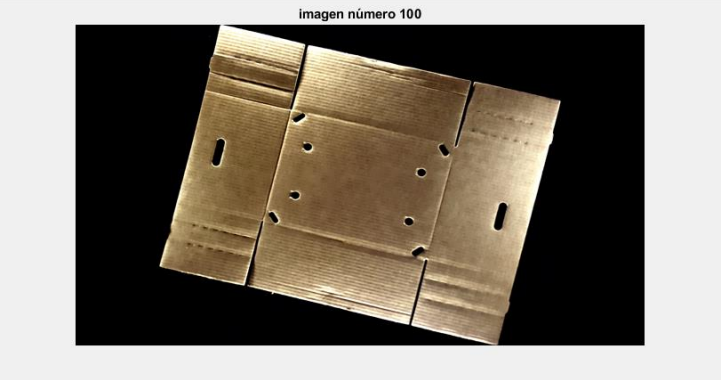


Figura 140 Visualización de una imagen cargada

- **Carga de imágenes etiquetadas.**

Usamos la función *pixelLabelDataStore* para cargar los datos relacionados con las imágenes etiquetadas. Esta función contiene los datos de las etiquetas a nivel de píxel y el ID de cada etiqueta para identificar las clases. Las clases que usaremos son:

```
%% CARGAR IMAGENES ETIQUETADAS

classes = [
    "Fondo"
    "Orejita"
    "Pared_Corta"
    "Pared_Larga"
    "Doble_Pared_PC"
    "Doble_Pared_PL"
    "Solapa"
    "Tejadillo"
    "Columna"
    "Entorno"
    ];
```

A continuación, creamos un mapa de colores personalizado para asignarle a cada etiqueta su color correspondiente. Tomamos como referencia los valores de los colores RGB mostrados en la tabla 7.

```
% mapa de colores
cmap=[
    0 255 255     % Fondo
    255 128 0     % Orejita
    255 0 255     % Pared_Corta
    0 255 0       % Pared_Larga
    255 0 128     % Doble_Pared_PC
    128 0 255     % Doble_Pared_PL
    255 0 0       % Solapa
    0 0 255       % Tejadillo
    255 255 0     % Columna
    255 255 255   % Entorno
    ];
% normalizar
cmap=cmap ./255;
```

En las imágenes etiquetadas, cada ID de píxel etiquetado se proporciona

como valor de un color RGB. Para asociar las etiquetas con su clase, hay que asociar los valores de los colores RGB con su clase correspondiente.

```
labelIDs= {...
    [
    0 255 255; ... % "Fondo"
    ]
    [
    255 128 0; ... % "Orejita"
    ]
    [
    255 0 255; ... % "Pared_Corta"
    ]
    [
    0 255 0; ... % "Pared_Larga"
    ]
    [
    255 0 128; ... % "Doble_Pared_PC"
    ]
    [
    128 0 255; ... % "Doble_Pared_PL"
    ]
    [
    255 0 0; ... % "Solapa"
    ]
    [
    0 0 255; ... % "Tejadillo"
    ]
    [
    255 255 0; ... % "Columna"
    ]
    [
    255 255 255; ...% "Entorno"
    ]
    };
```

Utilizamos las clases y los IDs de las etiquetas para crear *pixelLabelDataStore*. Esta función asociará los valores RGB con las etiquetas y cuando se lee una imagen, devuelve una matriz categórica. Funciona de la misma manera que la función *ImageDataStore* con la principal diferencia de que el etiquetado no es para un único objeto, sino para cada píxel individualmente.

```
LbDataStore = pixelLabelDatastore(DirEtiqOr,classes,labelIDs);
```

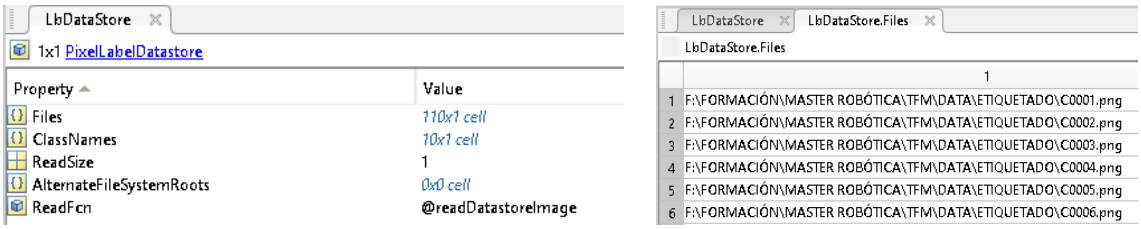


Figura 141 Vista previa de LbDataStore

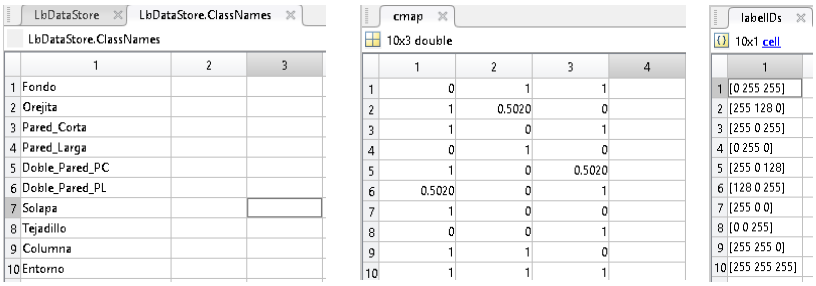


Figura 142 Vista previa de los parámetros de LbDataStore

Es muy importante que las imágenes de las planchas y las imágenes etiquetadas tengan una correspondencia de 1-1 en el orden cuando se consultan en el disco. Por eso los nombres de las imágenes deben tener la misma numeración en el disco.

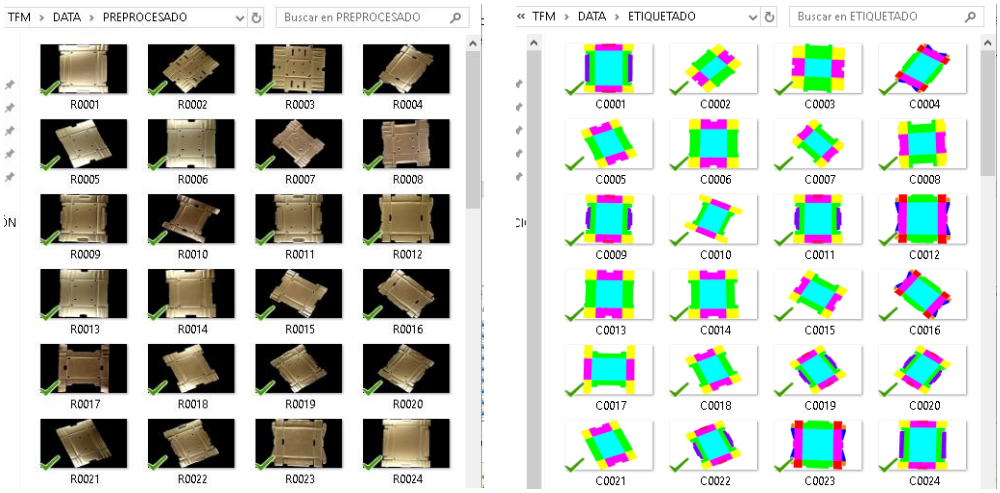


Figura 143 Correspondencia 1-1 de las imágenes en el disco

Visualizamos una muestra del mapeado realizado sobre un trozo de una imagen.

```
%% Mostrar mapeado
 C_rgb = imread(LbDataStore.Files{100});
 C = readimage(LbDataStore, 100);

            if size(C_rgb,1) == 720
                B =
labeloverlay(C_rgb(600:650,600:650,:),C(600:650,600:650,:),'ColorMap',cmap);
                % Show a ground-truth array
                C(600:650,600:650)
            else
                B =
labeloverlay(C_rgb(200:225,200:225,:),C(200:225,200:225,:),'ColorMap',cmap);
                C(200:225,200:225)
            end
            reset(LbDataStore);
            imshow(B,'InitialMagnification',400)
```

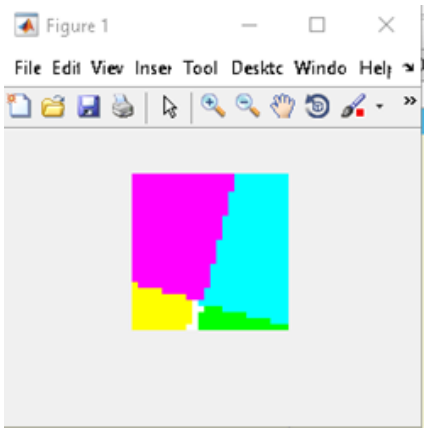


Figura 144 Vista previa del mapeado sobre una imagen

```
ans =

  26×26 categorical array

  Columns 1 through 11

    Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta
    Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta
    Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta
    Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta
    Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta
    Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta
    Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta
    Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta
    Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta
    Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta
    Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta
    Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta
    Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta
    Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta
    Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta
    Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta
    Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta
    Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta
    Columna          Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta
    Columna          Columna          Columna          Columna          Columna          Pared_Corta      Pared_Corta
    Columna          Columna          Columna          Columna          Columna          Columna          Columna
    Columna          Columna          Columna          Columna          Columna          Columna          Columna
    Columna          Columna          Columna          Columna          Columna          Columna          Columna
    Columna          Columna          Columna          Columna          Columna          Columna          Columna
    Columna          Columna          Columna          Columna          Columna          Columna          Columna
    Columna          Columna          Columna          Columna          Columna          Columna          Columna

  Columns 12 through 22

    Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Fondo
    Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Pared_Corta      Fondo
```

Figura 145 Vista previa del etiquetado píxel a píxel de una imagen

A continuación, se muestra una imagen etiquetada superpuesta con un grado de transparencia sobre su imagen de plancha correspondiente.

```
%% leer y mostrar etiquetas sobre la imagen
drawnow

Cc = readimage(LbDataStore, Im_N);
 % Superponer el resultado de la segmentación sobre la imagen original. - Computer Vision System Toolbox
 Bb = labeloverlay(I_ra,Cc,'ColorMap',cmap);
 figure
 imshow(Bb)
 drawnow
```

Añadimos una barra de colores al lado de la imagen para visualizar el color y el nombre de la etiqueta equivalente.

```
105. colormap(gca,cmap)
106.
107. % añadir una barra de colores a la imagen actual.
108. c = colorbar('peer', gca);
109.
110. % usar los nombres de las clases para cada color.
111. c.TickLabels = classes;
112. numClasses = size(cmap,1);
113.
114. % centrar.
115. c.Ticks = 1/(numClasses*2):1/numClasses:1;
116.
117. % Eliminar marcas.
118. c.TickLength = 0;
```

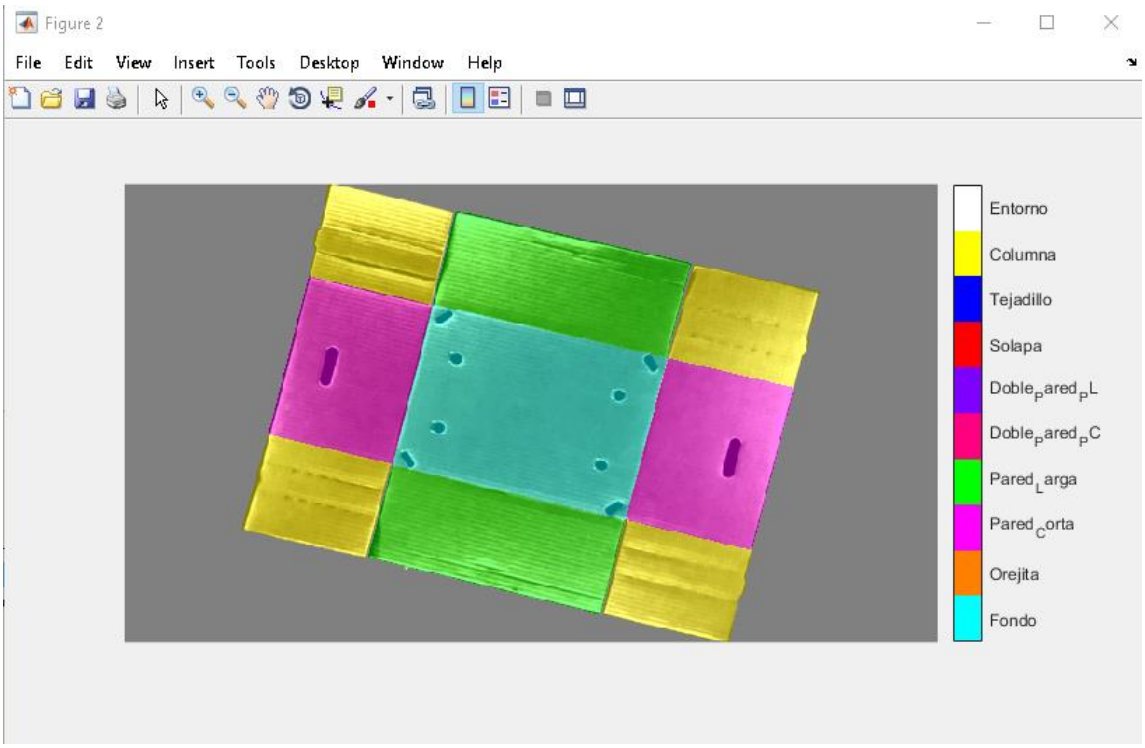


Figura 146 Imagen etiquetada superpuesta sobre la imagen de plancha correspondiente

- **Estadísticas.**

Para visualizar la distribución de las etiquetas de cada clase, utilizamos la función *countEachLabel*. Esta función cuenta el número de píxeles en cada clase y devuelve una tabla con las etiquetas, el número de píxeles asociados a ellas y el número total de pixeles en las imágenes que contienen la etiqueta.

```
119.  %% ESTADISTICAS
120.
121.   tbl = countEachLabel(LbDataStore)
```

tbl =

10×3 table

Name	PixelCount	ImagePixelCount
'Fondo'	3.9727e+06	2.5344e+07
'Orejita'	93681	7.1424e+06
'Pared_Corta'	2.377e+06	2.5344e+07
'Pared_Larga'	2.674e+06	2.5344e+07
'Doble_Pared_PC'	2.0127e+05	7.3728e+06
'Doble_Pared_PL'	2.3897e+05	6.2208e+06
'Solapa'	4.7581e+05	7.1424e+06
'Tejadillo'	2.3202e+05	7.1424e+06
'Columna'	1.8548e+06	1.8202e+07
'Entorno'	1.3224e+07	2.5344e+07

Figura 147 Tabla de estadísticas

PixelCount: numero de pixeles asociados a cada clase.
ImagePixelCount: numero total de pixeles en las imágenes que contienen la clase en cuestión.

La razón por el cual el número total de píxeles en las imágenes (3ª columna) cambia de una clase a otra, es porque la función solo cuenta los pixeles de las imágenes que llevan la clase en cuestión. Por esta razón las etiquetas que aparecen en todas las imágenes como el fondo y las dos paredes tienen el mismo número total de pixeles que es igual al número total de pixeles en todas las imágenes de la base de datos.

Calculamos la frecuencia para obtener el histograma con la información de la tabla.

```
% calculamos la frecuencia para conseguir un histograma de los datos

frequency = tbl.PixelCount/sum(tbl.PixelCount);
figure
bar(1:numel(classes),frequency)
xticks(1:numel(classes))
xticklabels(tbl.Name)
xtickangle(45)
ylabel('Frequency')
```

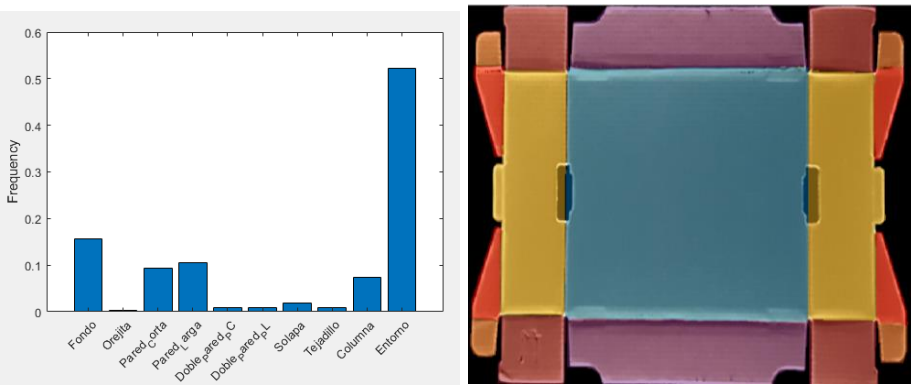


Figura 148 Histograma de frecuencia de las clases

Este histograma nos permite ver el desequilibrio que hay entre las diferentes clases. Este es debido al porcentaje de los pixeles que pertenecen a cada área de las diferentes características de la plancha. Las clases con más participación representan las características con mayor área como el fondo y las paredes. Mientras las clases con participaciones más pequeñas representan a características con menor área como la orejita y la doble pared. Estas últimas clases pueden suponer una dificultad para el aprendizaje. Por lo tanto, es aconsejable, por un lado, incrementar la cantidad de imágenes donde estas clases pueden tener mayor participación. Y, por otro lado, equilibrar todas las clases antes de entrenar la red neuronal para optimizar el proceso de aprendizaje.

- **Preparación de datos de entrenamiento y test.**

Inicialmente utilizamos el 80% de las imágenes para entrenar la red neuronal. El resto de las imágenes se usará para test y validación. El siguiente código divide las imágenes y las imágenes etiquetadas de forma aleatoria en conjuntos de entrenamiento y test.

```
%% PREPARACION DATOS DE ENTRENAMIENTO Y test

% dividir los datos aleatoriamente seleccionando el 80% para el entrenamiento,
% el resto se usara para el test.

% establecer el estado aleatorio inicial.
rng(0);
numFiles = numel(ImDataStore.Files);
% Devuelve un vector de fila que contiene una permutación aleatoria
%de numeros enteros desde 1 hasta n incluido.
shuffledIndices = randperm(numFiles);

% utilizar el 80% para el entrenamiento.
N = round(0.8 * numFiles);
trainingIdx = shuffledIndices(1:N);

% utilizar el resto de imagenes para test.
testIdx = shuffledIndices(N+1:end);

% Crear image datastores para el entrenamiento y test.
trainingImages = ImDataStore.Files(trainingIdx);
testImages = ImDataStore.Files(testIdx);
imdsTrain = imageDatastore(trainingImages);
imdsTest = imageDatastore(testImages);


% extraer la info de las clases y label IDs.
classes = LbDataStore.ClassNames;
labelIDs = 1:numel(LbDataStore.ClassNames);

% Crear pixel label datastores para el netrenamiento y test.
trainingLabels = LbDataStore.Files(trainingIdx);
testLabels = LbDataStore.Files(testIdx);
pxdsTrain = pixelLabelDatastore(trainingLabels, classes, labelIDs);
pxdsTest = pixelLabelDatastore(testLabels, classes, labelIDs);
DataSourceTest=pixelLabelImageSource(imdsTest,pxdsTest);

% numero total de imagenes de entrenamiento
numTrainingImages = numel(imdsTrain.Files)
% numero total de imagenes de test
numTestingImages = numel(imdsTest.Files)
```

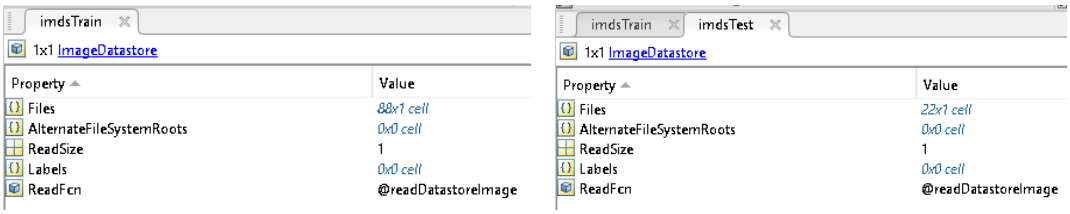


Figura 149 Proporciones de datos de entrenamiento y test.

5.3.2. Creación de la red neuronal.

Utilizamos la función *SegnetLayers* para crear una red neuronal *SegNet* inicializada usando los pesos de la red neuronal *VGG-16*. Esta función realiza automáticamente las operaciones necesarias para transferir los pesos de la red neuronal VGG-16 y añadir las capas adicionales que hacen falta para la segmentación semántica según la dimensión de imagen elegida y el número de clases disponibles.

En primer lugar, se configura la dimensión de las imágenes de entrada de la red neuronal según el tamaño de las imágenes disponibles en la base de datos, y se ajusta el número de valores de la capa de salida igual al número de clases a clasificar.

```
. %% Crear la red neuronal
imageSize = [360 640 3];
numClasses = numel(classes); % número de clases

% segnetLayers devuelve las capas de la red SegNet que está ya inicializada
% con los pesos y capas del modelo preintrenado.
lgraph = segnetLayers(imageSize,numClasses,'vgg16');
% lgraph inicialmente tiene 91 capas.
lgraph.Layers
```

Como resultado obtenemos las 91 capas de la red neuronal creada:

1	**'inputImage'**	**Image Input**	**360x640x3 images with 'zerocenter' normalization**
2	'conv1_1'	Convolution	64 3x3x3 convolutions with stride [1 1] and padding [1 1 1 1]
3	'bn_conv1_1'	Batch Normalization	Batch normalization
4	'relu1_1'	ReLU	ReLU
5	'conv1_2'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
6	'bn_conv1_2'	Batch Normalization	Batch normalization
7	'relu1_2'	ReLU	ReLU
8	'pool1'	Max Pooling	2x2 max pooling with stride [2 2] and padding [0 0 0 0]
9	'conv2_1'	Convolution	128 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
10	'bn_conv2_1'	Batch Normalization	Batch normalization
11	'relu2_1'	ReLU	ReLU
12	'conv2_2'	Convolution	128 3x3x128 convolutions with stride [1 1] and padding [1 1 1 1]
13	'bn_conv2_2'	Batch Normalization	Batch normalization
14	'relu2_2'	ReLU	ReLU
15	'pool2'	Max Pooling	2x2 max pooling with stride [2 2] and padding [0 0 0 0]
16	'conv3_1'	Convolution	256 3x3x128 convolutions with stride [1 1] and padding [1 1 1 1]
17	'bn_conv3_1'	Batch Normalization	Batch normalization
18	'relu3_1'	ReLU	ReLU
19	'conv3_2'	Convolution	256 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
20	'bn_conv3_2'	Batch Normalization	Batch normalization
21	'relu3_2'	ReLU	ReLU
22	'conv3_3'	Convolution	256 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
23	'bn_conv3_3'	Batch Normalization	Batch normalization
24	'relu3_3'	ReLU	ReLU
25	'pool3'	Max Pooling	2x2 max pooling with stride [2 2] and padding [0 0 0 0]
26	'conv4_1'	Convolution	512 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]

27	'bn_conv4_1'	Batch Normalization	Batch normalization
28	'relu4_1'	ReLU	ReLU
29	'conv4_2'	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
30	'bn_conv4_2'	Batch Normalization	Batch normalization
31	'relu4_2'	ReLU	ReLU
32	'conv4_3'	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
33	'bn_conv4_3'	Batch Normalization	Batch normalization
34	'relu4_3'	ReLU	ReLU
35	'pool4'	Max Pooling	2x2 max pooling with stride [2 2] and padding [0 0 0 0]
36	'conv5_1'	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
37	'bn_conv5_1'	Batch Normalization	Batch normalization
38	'relu5_1'	ReLU	ReLU
39	'conv5_2'	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
40	'bn_conv5_2'	Batch Normalization	Batch normalization
41	'relu5_2'	ReLU	ReLU
42	'conv5_3'	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
43	'bn_conv5_3'	Batch Normalization	Batch normalization
44	'relu5_3'	ReLU	ReLU
45	'pool5'	Max Pooling	2x2 max pooling with stride [2 2] and padding [0 0 0 0]
46	decoder5_unpool	Max Unpooling	Max Unpooling
47	'decoder5_conv3	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
48	'decoder5_bn_3'	Batch Normalization	Batch normalization
49	decoder5_relu_3	ReLU	ReLU
50	'decoder5_conv2	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
51	'decoder5_bn_2'	Batch Normalization	Batch normalization
52	decoder5_relu_2	ReLU	ReLU
53	'decoder5_conv1	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
54	'decoder5_bn_1'	Batch Normalization	Batch normalization
55	decoder5_relu_1	ReLU	ReLU
56	decoder4_unpool	Max Unpooling	Max Unpooling
57	'decoder4_conv3	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
58	'decoder4_bn_3'	Batch Normalization	Batch normalization
59	decoder4_relu_3	ReLU	ReLU
60	'decoder4_conv2	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
61	'decoder4_bn_2'	Batch Normalization	Batch normalization
62	decoder4_relu_2	ReLU	ReLU
63	'decoder4_conv1	Convolution	256 3x3x512 convolutions with stride [1 1] and padding [1 1 1 1]
64	'decoder4_bn_1'	Batch Normalization	Batch normalization
65	decoder4_relu_1	ReLU	ReLU
66	decoder3_unpool	Max Unpooling	Max Unpooling
67	'decoder3_conv3	Convolution	256 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
68	'decoder3_bn_3'	Batch Normalization	Batch normalization
69	decoder3_relu_3	ReLU	ReLU
70	'decoder3_conv2	Convolution	256 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
71	'decoder3_bn_2'	Batch Normalization	Batch normalization
72	decoder3_relu_2	ReLU	ReLU
73	decoder3_conv1	Convolution	128 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
74	'decoder3_bn_1'	Batch Normalization	Batch normalization
75	decoder3_relu_1	ReLU	ReLU
76	decoder2_unpool	Max Unpooling	Max Unpooling
77	decoder2_conv2	Convolution	128 3x3x128 convolutions with stride [1 1] and padding [1 1 1 1]
78	'decoder2_bn_2'	Batch Normalization	Batch normalization
79	decoder2_relu_2	ReLU	ReLU
80	decoder2_conv1	Convolution	64 3x3x128 convolutions with stride [1 1] and padding [1 1 1 1]
81	'decoder2_bn_1'	Batch Normalization	Batch normalization
82	decoder2_relu_1	ReLU	ReLU
83	decoder1_unpool	Max Unpooling	Max Unpooling

84	decoder1_conv2	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
85	'decoder1_bn_2'	Batch Normalization	Batch normalization
86	decoder1_relu_2	ReLU	ReLU
87	'decoder1_conv1'	Convolution	10 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
88	'decoder1_bn_1'	Batch Normalization	Batch normalization
89	'decoder1_relu_1'	ReLU	ReLU
90	'softmax'	Softmax	softmax
91	'pixelLabels'	Pixel Classification Layer	Cross-entropy loss

Tabla 8 Capas de la red neuronal

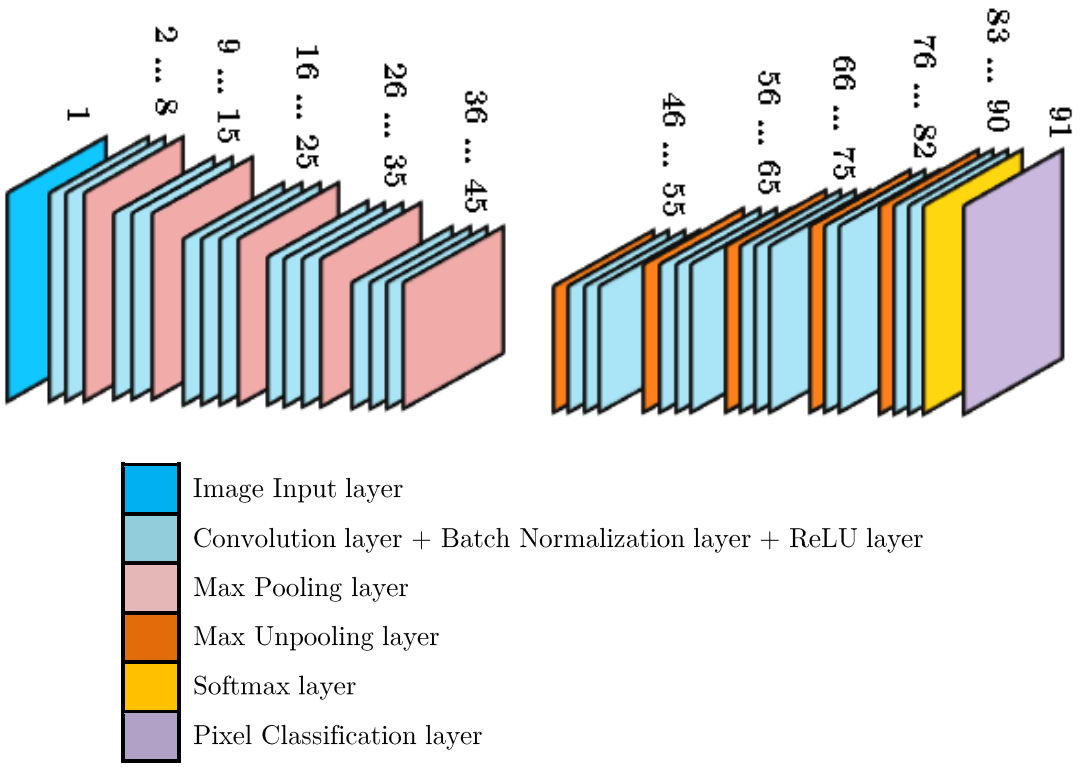


Figura 150 Estructura de capas de la red neuronal

- Image input layer: Esta capa recibe las imágenes y las convierte en valores numéricos organizados en forma matricial según los valores de los canales R, G y B.
- Convolution layer: Esta capa realiza el cálculo de convolución sobre varias porciones de la imagen de entrada con un conjunto de filtros cuyos parámetros se aprenderán en el proceso de entrenamiento. Cada uno de estos filtros puede detectar una determinada característica para generar un mapa de características en la imagen de salida. En las primeras capas, la red neuronal convolucional buscará características de nivel inferior, como bordes horizontales o verticales. Cuanto más avancemos en la red, se buscarán características de nivel superior, como una pared de la plancha, por ejemplo.

La primera capa de convolución utiliza 64 filtros de convolución de 3x3x3, con un paso 'stride' de dimensión 1x1 y un relleno 'padding' de 1x1x1x1.

- Batch Normalization layer: En esta capa las entradas son normalizadas y estandarizadas automáticamente para acelerar el proceso de entrenamiento. La capa primero normaliza las activaciones de cada canal restando la media del mini lote. Luego, la capa desplaza la entrada por un desplazamiento aprendido y la escala por un factor de escala aprendido.
- ReLU layer: Esta capa añade no linealidad a la red. La capa convolucional es una capa lineal ya que es una multiplicación de los pesos del filtro y los valores de entrada. El resultado de una función ReLu es igual a 0 para todos los valores de x<=0. De lo contrario, es igual al valor de x.
- Max Pooling layer: Esta capa reduce aún más la dimensión de las matrices. Esto acelera el tiempo de cómputo a medida que se reduce el número de parámetros a estimar. Además de esto, ayuda a evitar el sobreajuste al hacer que la red sea más robusta. En este proceso de entrenamiento el max pooling se realiza con un tamaño de 2x2 y un paso de 2 como el siguiente ejemplo.

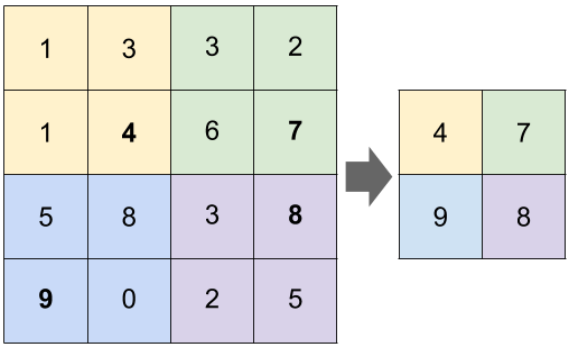


Figura 151 pooling

- Max unpooling layer: para cada capa de pooling, se almacenan las ubicaciones de los máximos para utilizarlas luego en la capa de unpooling.

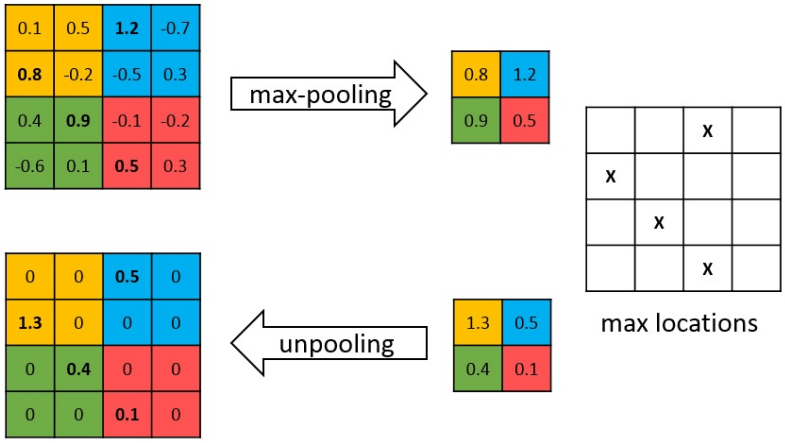


Figura 152 max pooling y unpooling

- Softmax layer : La función Softmax es una función que limita su salida al rango de 0 a 1. Esto permite que la salida se interprete directamente como una probabilidad. Al mismo tiempo, las funciones Softmax son sigmoides de múltiples clases que se utilizan para determinar la probabilidad de múltiples clases a la vez. Esta capa Softmax permite que la red neuronal ejecute una función multiclase para determinar la probabilidad de existencia de varios objetos en la imagen a la vez. [73]

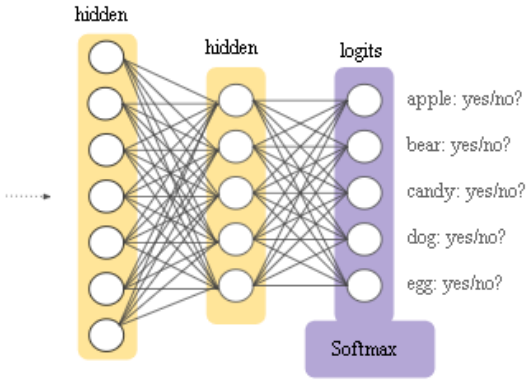


Figura 153 Softmax layer [73]

- Pixel Classification layer: Esta capa produce una clasificación a nivel de pixel asignando una etiqueta categórica a cada píxel de imagen. La capa ignora automáticamente los píxeles no definidos durante el entrenamiento.

A continuación, mostramos todas estas capas como están unidas entre sí, y el detalle de las últimas 9 capas.

```
% Plot the 91-Layer lgraph
fig1=figure('Position', [100, 100, 1000, 1100]);
subplot(1,2,1)
plot(lgraph);
axis off
axis tight
title('Complete Layer Graph')
subplot(1,2,2)
plot(lgraph);
xlim([2.862 3.200])
ylim([-0.9 10.9])
axis off
title('Last 9 Layers Graph')
```

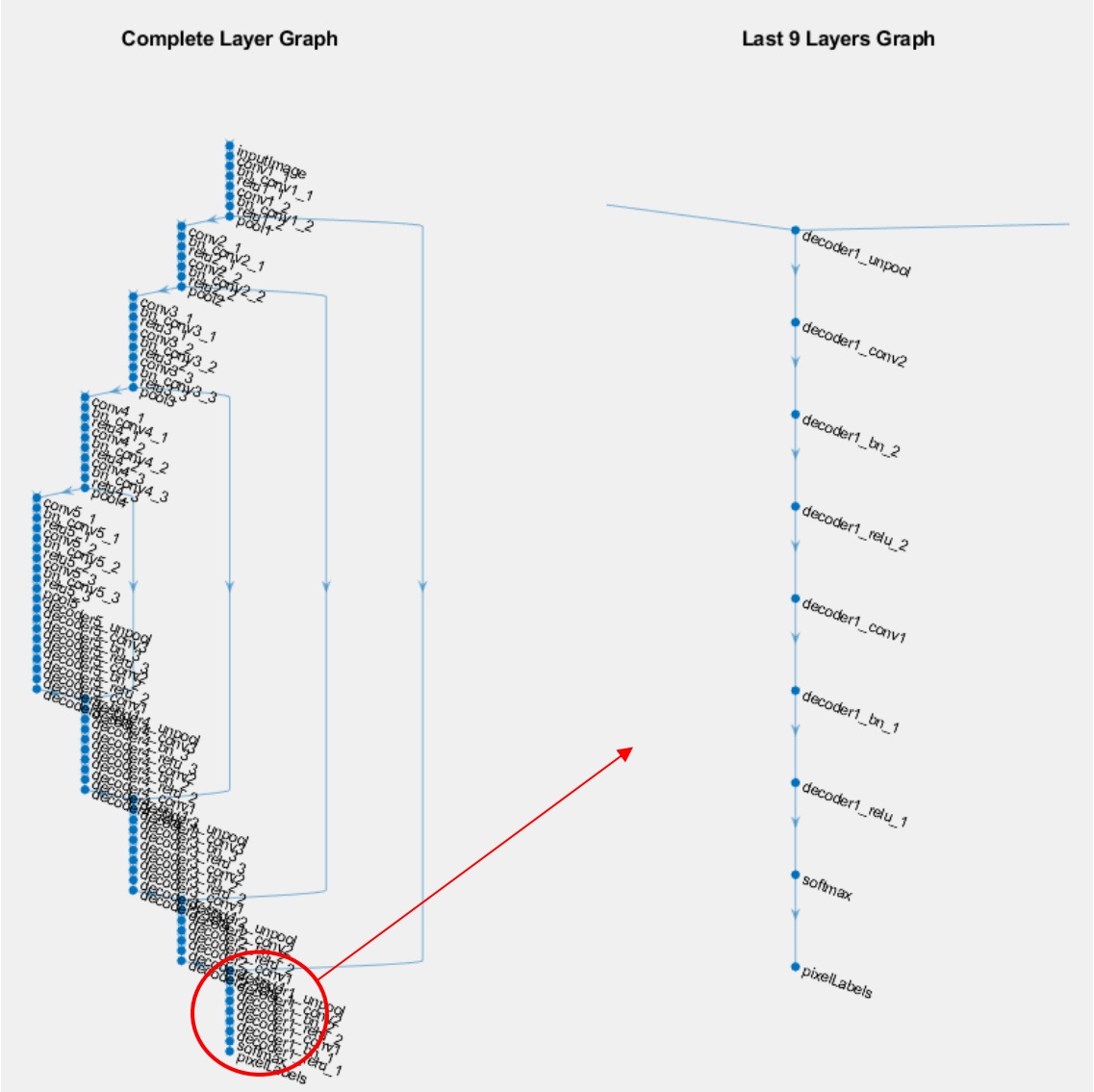


Figura 154 Arquitectura de la red neuronal creada

5.3.3. Equilibrar las clases.

Como se ha mencionado anteriormente, las clases en nuestro conjunto de datos no están equilibrados. Para mejorar el proceso de entrenamiento, los pesos de las clases se pueden usar para equilibrarlas. Usando las estadísticas calculadas anteriormente podemos calcular el valor de la mediana de la frecuencia de las clases para saber el valor de frecuencia promedio, y dividir este valor sobre cada una de las frecuencias para obtener los pesos y conseguir que cuanto mayor es la frecuencia menor es el peso. De esta forma conseguimos darle más resalte a las clases con pixeles de menor frecuencia en las imágenes.

$$frecuencia = \frac{PixelCount}{ImagePixelCount} \quad [10]$$

$$pesos_{clases} = \frac{mediana_{frecuencia}}{frecuencia} \quad [11]$$

La mediana de la frecuencia es la media de los dos valores medianos de la columna de frecuencias ordenada de menor a mayor.

$$mediana_{frecuencia} = 0.0802 \quad [12]$$

clases	PixelCount	ImagePixelCount	Frecuencias	pesos
Fondo	3.9727e+06	2.5344e+07	0,156	0,5117
Orejeta	93681	7.1424e+06	0,0131	6,1148
Pared_Corta	2.377e+06	2.5344e+07	0,093	0,8552
Pared_Larga	2.674e+06	2.5344e+07	0,106	0,7548
Doble_Pared_PC	2.0127e+05	7.3728e+06	0,027	2,9380
Doble_Pared_PL	2.3897e+05	6.2208e+06	0,038	2,0878
Solapa	4.7581e+05	7.1424e+06	0,066	1,2039
Tejadillo	2.3202e+05	7.1424e+06	0,032	2,4690
Columna	1.8548e+06	1.8202e+07	0,102	0,7859
Entorno	1.3224e+07	2.5344e+07	0,520	0,1540

Tabla 9 Estadísticas

```
194.  %% equilibrar clases usando sus pesos
195.
196.  % Get the imageFreq using the data from the countEachLabel function
197.  imageFreq = tbl.PixelCount ./ tbl.ImagePixelCount;
198.  % The higher the frequency of a class the smaller the classWeight
199.  classWeights = median(imageFreq) ./ imageFreq
```

```
classWeights =

    0.5117
    6.1149
    0.8551
    0.7602
    2.9380
    2.0879
    1.2039
    2.4690
    0.7871
    0.1537
```

Figura 155 Pesos equilibrados

Para introducir los valores de los pesos en la capa de clasificación de la red neuronal, utilizamos la función *pxLayer* y el nombre de la capa *pixelClassificationLayer.*

```
200.  pxLayer = pixelClassificationLayer('Name','labels','ClassName', tbl.Name,
    'ClassWeights', classWeights)
```

```
pxLayer =

  PixelClassificationLayer with properties:

            Name: 'labels'
      ClassNames: {1×10 cell}
   ClassWeights: [10×1 double]
      OutputSize: 'auto'

  Hyperparameters
   LossFunction: 'crossentropyex'
```

Figura 156 Valores de la capa de clasificación

Ahora podemos usar la capa de clasificación de píxel creada para proporcionar una etiqueta categórica por cada píxel procesado por la red neuronal convolucional. Esta capa reemplazará la última capa de la red neuronal actual.

A continuación, actualizamos la *SegNet* con la nueva capa de clasificación pixelClassificationLayer. Esta operación requiere eliminar la antigua capa y reemplazarla por la nueva. La capa en cuestión tiene un índice de 91 en *lgraph.Layers*. Eliminamos la capa usando la función *removeLayers*, y añadimos la nueva capa usando la función *addLayers*, finalmente reconectamos la nueva capa usando la función *connectLayers*. La razón porqué usamos la función connectLayers es porque estamos trabajando con una red neuronal DAG *(directed acyclic graph)*. Este tipo de red tiene una arquitectura compleja donde una capa tiene entradas desde varias capas y salida hacia otras.

```
201.  % actualizar la red
202.
203.  % Plot the 91-Layer lgraph
204.  fig2=figure('Position', [100, 100, 800, 600]);
205.  subplot(1,2,1)
206.  plot(lgraph);
207.  xlim([2.862 3.200])
208.  ylim([-0.9 10.9])
209.    axis off
210.  title('Initial last 9 Layers Graph')
211.  % Remove last layer of and add the new one we created.
212.  lgraph = removeLayers(lgraph, {'pixelLabels'});
213.  lgraph = addLayers(lgraph, pxLayer);
214.  % Connect the newly created layer with the graph.
215.  lgraph = connectLayers(lgraph, 'softmax','labels');
216.  lgraph.Layers
217.
218.  subplot(1,2,2)
219.  plot(lgraph);
220.  xlim([2.862 3.200])
221.  ylim([-0.9 10.9])
222.  axis off
223.  title(' últimas 9 capas modificadas ')
```

80	decoder2_conv1	Convolution	64 3x3x128 convolutions with stride [1 1] and padding [1 1 1 1]
81	'decoder2_bn_1'	Batch Normalization	Batch normalization
82	decoder2_relu_1	ReLU	ReLU
83	decoder1_unpool	Max Unpooling	Max Unpooling
84	decoder1_conv2	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
85	'decoder1_bn_2'	Batch Normalization	Batch normalization
86	decoder1_relu_2	ReLU	ReLU
87	'decoder1_conv1	Convolution	10 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
88	'decoder1_bn_1'	Batch Normalization	Batch normalization
89	'decoder1_relu_1'	ReLU	ReLU
90	'softmax'	Softmax	softmax
91	'pixelLabels'	Pixel Classification Layer	Class weighted Cross-entropy los with 'Fondo', 'Orejita', and 8 other classes

Tabla 10 las capas modificadas

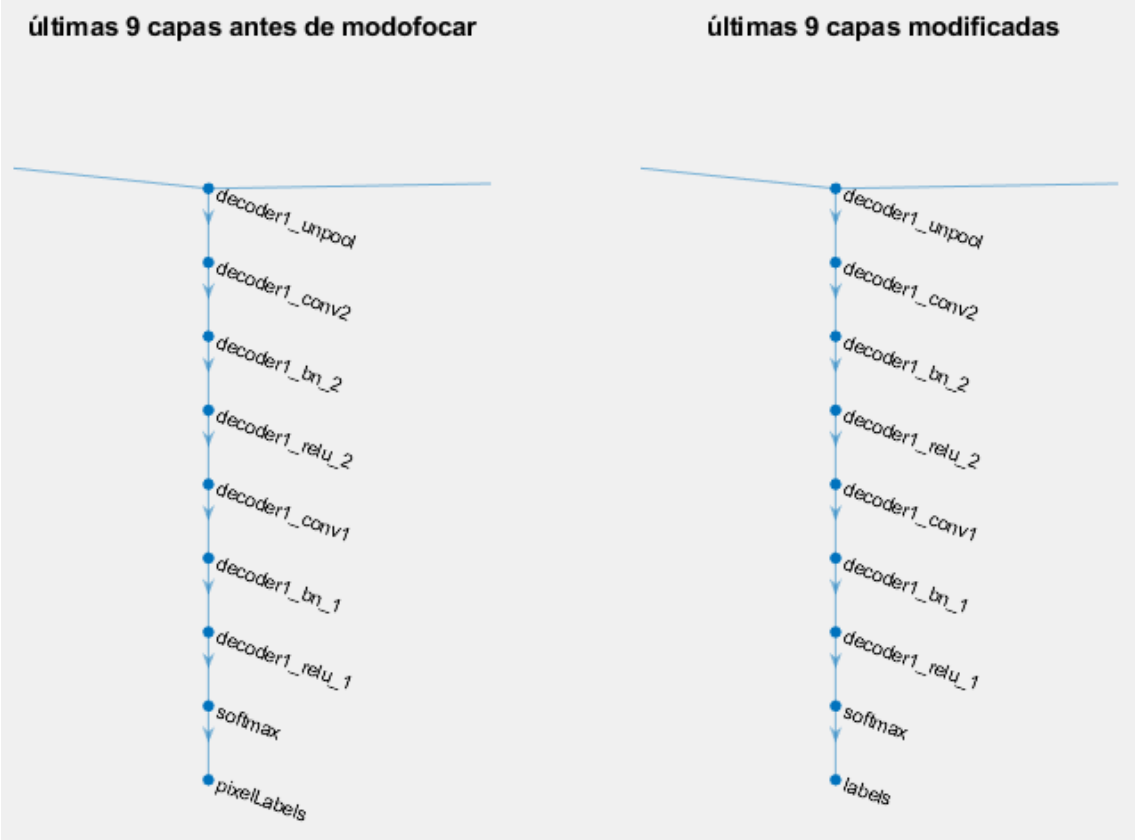


Figura 157 Capas antes y después de modificar

5.3.4. Opciones de entrenamiento.

Existen diferentes opciones para el algoritmo de optimización que se puede usar en el entrenamiento de la red neuronal.

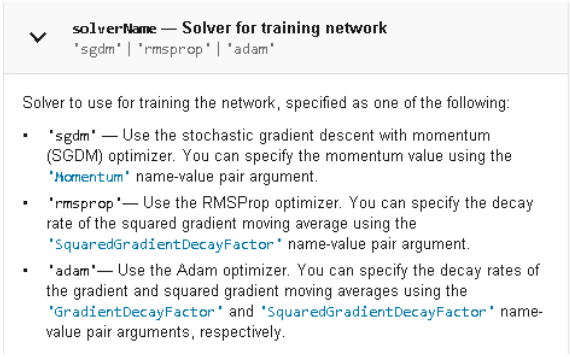


Figura 158 Algoritmos de optimización.

Para el entrenamiento de la red utilizamos el algoritmo de Descenso de gradiente estocástico con momento (sgdm).

- SGDM (Stochastic Gradient Descent with Momentum): El algoritmo de descenso de gradiente estándar actualiza los parámetros de red (pesos y sesgos) para minimizar la función de pérdida al tomar pequeños pasos en cada iteración en la dirección del gradiente negativo de la pérdida,

$$\theta_{\ell + 1} = \theta_{\ell} - \alpha \nabla E(\theta_{\ell}) \text{ [12]}$$

Dónde ℓ es el número de iteración, $\alpha > 0$ es la velocidad de aprendizaje, θ es el vector de parámetros y $E(\theta)$ es la función de pérdida. En el algoritmo de descenso de gradiente estándar, el gradiente de la función de pérdida, $\nabla E(\theta)$, se evalúa utilizando todo el conjunto de entrenamiento, y el algoritmo de descenso de gradiente estándar utiliza todo el conjunto de datos a la vez.

Por el contrario, en cada iteración, el algoritmo de descenso de gradiente estocástico evalúa el gradiente y actualiza los parámetros utilizando un subconjunto de los datos de entrenamiento. Se utiliza un subconjunto diferente, llamado mini-lote, en cada iteración. El paso completo del algoritmo de entrenamiento sobre todo el conjunto de entrenamiento usando mini lotes es una época. El descenso de gradiente estocástico es estocástico porque las actualizaciones de parámetros calculadas usando un mini lote es una estimación ruidosa de la actualización de parámetros que resultaría del uso del conjunto de datos completo. Puede especificar el

tamaño del mini-lote y el número máximo de épocas utilizando los parámetros 'MiniBatchSize' y 'MaxEpochs' especificando nombre-valor, respectivamente.
El algoritmo de descenso de gradiente estocástico puede oscilar a lo largo del camino del descenso más pronunciado hacia el óptimo. Agregar un término de momento a la actualización de parámetros es una forma de reducir esta oscilación. La actualización de descenso de gradiente estocástico con momento *(sgdm)* es [74]:

$$\theta_{\ell} + 1 = \theta_{\ell} - \alpha\nabla E(\theta_{\ell}) + \gamma(\theta_{\ell} - \theta_{\ell} - 1) \quad [13]$$

Donde γ determina la contribución del paso de gradiente anterior a la iteración actual. Se puede especificar este valor utilizando el parámetro *'Momentum'*. Para entrenar una red neuronal usando el descenso de gradiente estocástico con algoritmo de momento, hay que especificar el parámetro *solverName* como *'sgdm'*. Para especificar el valor inicial de la tasa de aprendizaje α, hay que utilizar el parámetro *'InitialLearnRate'*.

Para configurar los parámetros del algoritmo, utilizamos la función *trainingOptions*.

```
%% Ajustar opciones de entrenamiento
options = trainingOptions('sgdm', ...   % algoritmo de optimización sgdm:
                                          stochastic gradient descent with momentum

'LearnRateSchedule','none',...      % metodo para disminuir la tasa de aprendizaje
                                      automaticamente

'Momentum', 0.9, ...                % contribución de una iteración anterior en la
                                      iteración actual.

'LearnRateDropFactor',0.1,...       % factor por el cual se multiplica la tasa de
                                      aprendizaje para disminuir.

'LearnRateDropPeriod',2,...         % número de épocas para disminuir la tasa de
                                      aprendizaje

'InitialLearnRate', 5e-2, ...       % si la tasa de aprendizaje es demasiado
                                      pequeña dura más el entrenamiento, si es
                                      muy grande no converge ( resultados
                                      suboptimos).

'L2Regularization', 0.0001, ...     % decaer los pesos para evitar el sobreajuste

'MaxEpochs', 250,... %120           % n° máximo de epocas. una epoca es el paso del
                                      algoritmo de optimización por todos los
                                      datos de entrenamiento.

'MiniBatchSize', 1, ... %4          % conjunto de datos de entrenamiento usado para
                                      evaluar el gradiente de la función de
                                      pérdida y actualizar los pesos.


'Shuffle', 'once', ...              % mezclar los datos de entrenamiento antes de
```

```
                                        cada epoca y mezclar los datos
                                        de validación antes de cada validación de
                                        la red.

    'ExecutionEnvironment','parallel', ...   % recurso usado en el entrenamiento:
                                        auto, cpu, gpu, parallel

    'Plots','training-progress',...     % gráficos a mostrar durante el
                                        entrenamiento

    'ValidationData',DataSourceTest, ...% datos usados para la validación .

    'ValidationFrecuency',50, ...       % iteraciones entre cada dos evaluaciones
                                        de métricas de validación. (no hace falta %
    en el caso de SGDM).

    'ValidationPatience',5, ...         % las veces en que la pérdida puede ser
                                        mayor o igual que la pérdida más pequeña
                                        antes de parar el entrenamiento
                                        ( validación )

    'Verbose', true,...                 % activar/desactivar la tabla de progreso
                                        mostrada en la ventana de comandos

    'VerboseFrequency',20);             % frecuencia de actualización de valores
                                        de la tabla.

    %'GradientThresholdMethod','l2norm',... % método para recortar los valores
                                            del gradiente que superan cierto
                                            umbral.

    %'GradientThreshold',0.05);             % si el gradiente supera este umbral,
                                            se corta acorde al metodo usado
```

```
options =

  TrainingOptionsSGDM with properties:

                     Momentum: 0.9000
                 InitialLearnRate: 0.0500
        LearnRateScheduleSettings: [1×1 struct]
                L2Regularization: 1.0000e-04
          GradientThresholdMethod: 'l2norm'
                GradientThreshold: Inf
                        MaxEpochs: 250
                    MiniBatchSize: 6
                          Verbose: 1
                  VerboseFrequency: 20
                    ValidationData: [1×1 pixelLabelImageDatastore]
               ValidationFrequency: 50
                ValidationPatience: 5
                          Shuffle: 'once'
                   CheckpointPath: ''
             ExecutionEnvironment: 'gpu'
                       WorkerLoad: []
                        OutputFcn: []
                            Plots: 'training-progress'
                   SequenceLength: 'longest'
             SequencePaddingValue: 0
```

Figura 159 Opciones de entrenamiento

5.3.5. *Data Augmentation.*

La función Data Augmentation se usa durante el entrenamiento para proporcionar a la red neuronal más datos para mejorar la precisión. Se pueden aplicar reflexiones, desplazamientos, giros o escalas de forma aleatoria en uno o dos ejes de la imagen. Usando esta función, los datos disponibles para el entrenamiento se multiplican para contribuir en la mejora del entrenamiento. Usamos imageDataAumenter para especificar los parámetros de esta función.

```
%% Data Aumentation

augmenter =
    imageDataAugmenter('RandXReflection',true,'RandYReflection',true,...
      'RandXTranslation', [-10 10], 'RandYTranslation',[-10 10],'RandRotation',[-
    10,10],'RandXShear',[-5,5],'RandYShear',[-5,5]);
```

Existen más tipos de manipulaciones soportados por esta función. Elegir un tipo u otro requiere un proceso de prueba y error hasta afinar el proceso de entrenamiento. A continuación, vemos un ejemplo de una imagen rotada.

```
% mostrar una rotacion

I = readimage(ImDataStore, Im_N);
              Ib = readimage(LbDataStore, Im_N);
              IIB = labeloverlay(I, Ib, 'Colormap', cmap, 'Transparency',0.8);

              % Convertir a uint8.
              L = uint8(Ib);
              Ir = imrotate(I,10,'nearest','crop');
              Lr = imrotate(L,10,'nearest','crop');
              valueset = 1:10;
              LB = categorical(Lr, valueset,LbDataStore.ClassNames);
              IB = labeloverlay(Ir, LB, 'Colormap', cmap, 'Transparency',0.8);
              figure
              imshowpair(IIB,IB,'montage');
%               HelperFunctions.pixelLabelColorbar(cmap, LbDataStore.ClassNames);
              title('Original vs Rotated image')
```

Aunque la imagen es rotada y sus valores son interpolados, el etiquetado a nivel de píxel se mantiene igual.

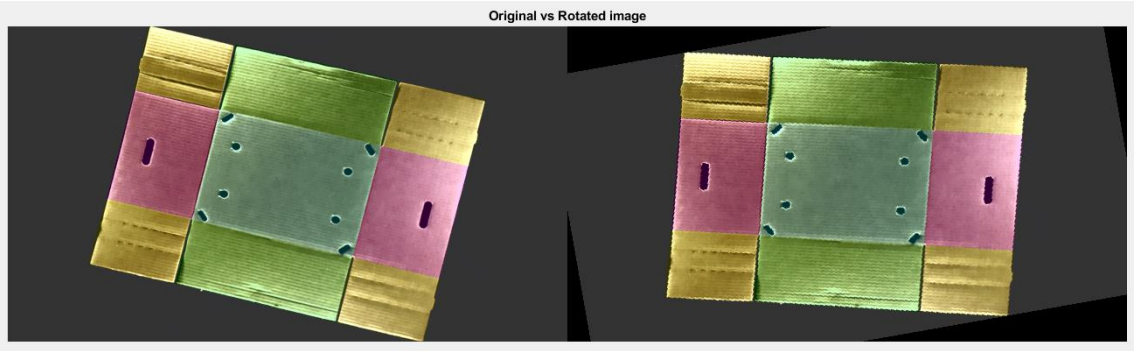


Figura 160 Imagen rotada

5.3.6. Comenzar el entrenamiento.

Combinamos los datos de entrenamiento junto con los datos generados con la función anterior, usando la función pixelLabelImageDatasource. Esta función lee lotes de datos, aplica Data Augmentation, y envía esta información al algoritmo de entrenamiento. Para contar el tiempo total recorrido por el entrenamiento, utilizamos la función tic/toc. Una vez terminado el entrenamiento guardamos la red neuronal entrenada para usar posteriormente.

```
%% Comenzar el entrenamiento

datasource = pixelLabelImageSource(imdsTrain,pxdsTrain,...
    'DataAugmentation',augmenter);

tic
[net, info] = trainNetwork(datasource,lgraph,options);
toc

save('PreTrainedCnn.mat','net','info','options');
disp('NN trained');
```

Como se ha mencionado anteriormente, existen varias opciones de hardware utilizado para el entrenamiento:

- **Auto**: Esta opción utiliza la tarjeta gráfica (GPU) cuando ésta esté disponible, en caso contrario utiliza el procesador (CPU).
- **Cpu**: Utiliza exclusivamente el procesador.
- **Gpu**: Utiliza exclusivamente el procesador gráfico.
- **Multi**-gpu: Utiliza múltiples tarjetas gráficas en una sola máquina.
- **Parallel**: Utiliza un fondo paralelo local (local parallel pool) o un conjunto para el cálculo (compute cluster). Si se tiene acceso a los procesadores gráficos, entonces solo trabajadores con un único procesador grafico realiza el cálculo. Si no se tiene acceso a los procesadores gráficos , el entrenamiento se realiza solo en los conjuntos de CPUs.

La duración de los entrenamientos varía de un ordenador a otro dependiendo de las características de hardware y de las opciones de entrenamiento. La cantidad y el tamaño de imágenes, la cantidad de iteraciones y el tamaño del batch-size, son los factores que más afectan a la duración del entrenamiento. Si aumentamos este último factor, el tiempo total disminuye ya que la CPU/GPU procesará más datos al mismo tiempo, sin embargo, aumenta la carga del ordenador considerablemente. La tasa de aprendizaje puede afectar también a este tiempo. Cuanto más bajo es este factor, más tiempo tarda el entrenamiento en completarse.

Las opciones 'gpu', 'multi-gpu' y 'parallel' requieren Parallel Computing Toolbox . Para utilizar una GPU en el entrenamiento, hace falta tener una tarjeta gráfica Nvidia con CUDA habilitado con una capacidad de cálculo superior a 3[74].

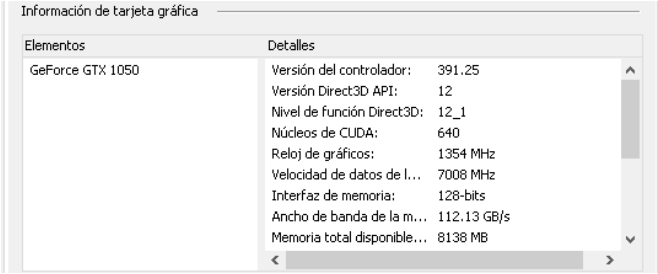


Figura 161 Información tarjeta gráfica

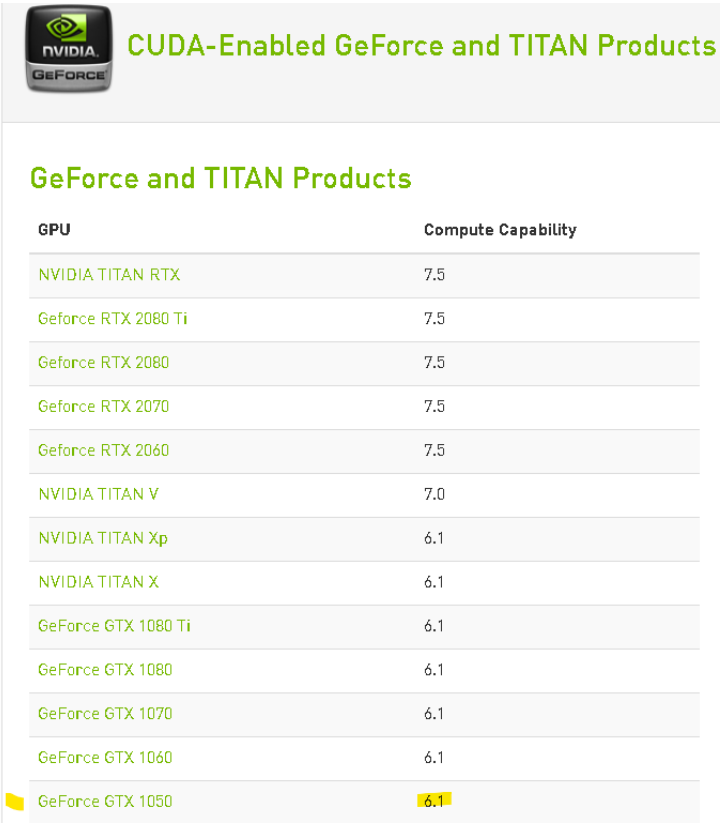


Figura 162 Capacidad de cálculo de la tarjeta gráfica [75]

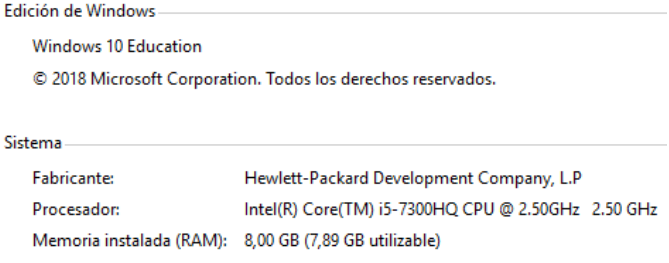


Figura 163 Características del equipo

```
>> license checkout Distrib_Computing_Toolbox

ans =

     1

>> distcomp.feature( 'LocalUseMpiexec', false )

ans =

  logical

    1
```

Figura 164 comprobación Computing toolbox

Capítulo 6

6. Resultados y análisis.

6.1. Introducción

En el siguiente capítulo se exponen los resultados de la segmentación semántica y se valoran las métricas de la red neuronal entrenada.

6.2. Resultados

A continuación, se presentan los resultados de entrenamiento de la red neuronal.

```
Training across multiple GPUs.
Initializing image normalization.
|=========================================================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Validation  |  Mini-batch  |  Validation  |  Base Learning  |
|         |             |    (hh:mm:ss)  |   Accuracy   |   Accuracy   |     Loss     |     Loss     |      Rate       |
|=========================================================================================================================|
|       1 |           1 |       00:00:02 |       10.07% |       10.75% |       2.5131 |       2.4729 |          0.0500 |
|       1 |          20 |       00:00:35 |       33.82% |              |       1.8910 |              |          0.0500 |
|       1 |          40 |       00:00:59 |       68.88% |              |       1.8855 |              |          0.0500 |
|       1 |          50 |       00:01:10 |       53.92% |       64.03% |       1.5729 |       1.8440 |          0.0500 |
|       1 |          60 |       00:01:32 |       64.37% |              |       2.2052 |              |          0.0500 |
|       1 |          80 |       00:01:55 |       67.63% |              |       1.6585 |              |          0.0500 |
|       2 |         100 |       00:02:19 |       70.57% |       74.64% |       0.9443 |       1.5647 |          0.0500 |
|       2 |         120 |       00:02:52 |       69.50% |              |       1.1124 |              |          0.0500 |
|       2 |         140 |       00:03:15 |       72.81% |              |       1.7003 |              |          0.0500 |
|       2 |         150 |       00:03:27 |       59.80% |       72.03% |       1.8213 |       1.5210 |          0.0500 |
|       2 |         160 |       00:03:48 |       59.44% |              |       2.0420 |              |          0.0500 |
|       3 |         180 |       00:04:11 |       76.91% |              |       0.7092 |              |          0.0500 |
|      42 |        3680 |       01:23:31 |       94.80% |              |       0.0799 |              |          0.0500 |
|      43 |        3700 |       01:23:54 |       95.79% |       97.24% |       0.0300 |       0.1478 |          0.0500 |
|      43 |        3720 |       01:24:27 |       95.77% |              |       0.0382 |              |          0.0500 |
|      43 |        3740 |       01:24:50 |       91.56% |              |       0.0620 |              |          0.0500 |
|      43 |        3750 |       01:25:01 |       91.33% |       97.16% |       0.0320 |       0.1341 |          0.0500 |
|      43 |        3760 |       01:25:23 |       93.63% |              |       0.0390 |              |          0.0500 |
|      43 |        3780 |       01:25:45 |       95.40% |              |       0.0508 |              |          0.0500 |
|      44 |        3800 |       01:26:08 |       89.14% |       97.58% |       0.0273 |       0.1131 |          0.0500 |
|      44 |        3820 |       01:26:41 |       87.66% |              |       0.0705 |              |          0.0500 |
|      44 |        3840 |       01:27:04 |       90.75% |              |       0.0499 |              |          0.0500 |
|      44 |        3850 |       01:27:15 |       90.49% |       97.64% |       0.0464 |       0.1145 |          0.0500 |
|      44 |        3860 |       01:27:36 |       92.37% |              |       0.0479 |              |          0.0500 |
|      45 |        3880 |       01:27:59 |       93.00% |              |       0.0437 |              |          0.0500 |
|      45 |        3900 |       01:28:22 |       93.67% |       97.79% |       0.0253 |       0.1080 |          0.0500 |
|      45 |        3902 |       01:28:34 |       96.33% |              |       0.0275 |              |          0.0500 |
|=========================================================================================================================|
Elapsed time is 5409.030826 seconds.
NN trained
```

Figura 165 Datos del proceso de entrenamiento

El entrenamiento se ha llevado a cabo en 45 épocas y 3902 iteraciones con una duración de 1 h : 28 min : 34 s y ha alcanzado una precisión de validación de 97.79% y un valor de pérdida mínimo. El entrenamiento se ha detenido en este

punto para evitar el sobreajuste de la red. La forma de calcular la precisión de validación es la siguiente:

$$\frac{cantidad\ pixeles\ predecidos\ correctamente}{cantidad\ píxeles} \quad [14]$$

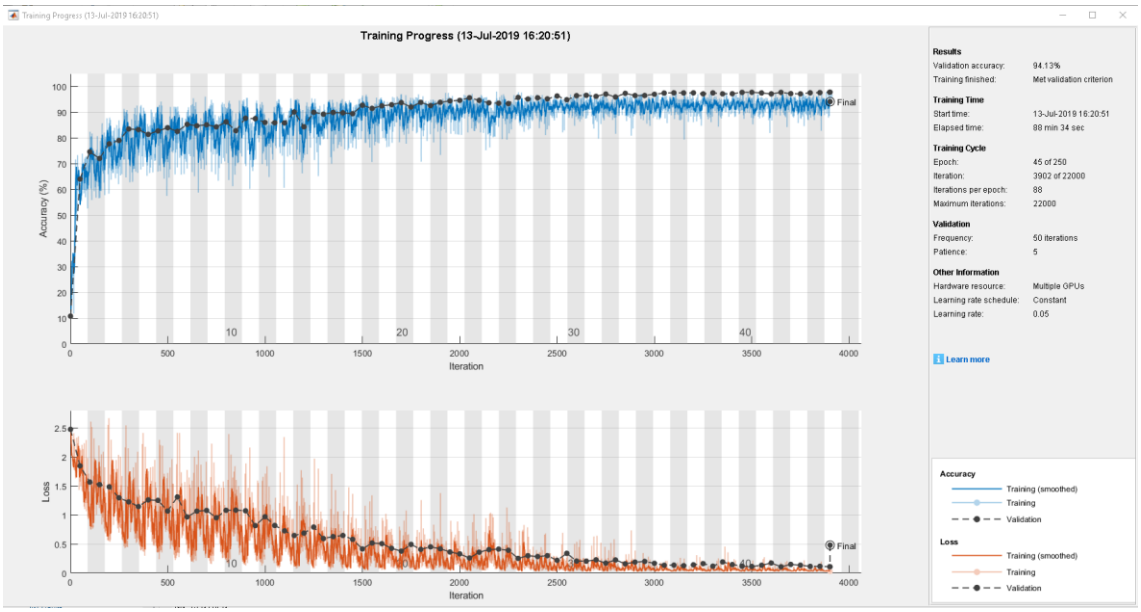


Figura 166 Progreso del proceso de entrenamiento

Figura 167 resultados del proceso de entrenamiento

A continuación, implementamos un código para visualizar una imagen arbitraria de la base de datos de entrenamiento junto con una imagen de la misma base de datos, pero etiquetada por la red, para examinar la eficacia de la red en los datos de entrenamiento.

```
%% visualizar imagenes
Im_N=30;
I=readimage(ImDataStore,Im_N);
Ib = readimage(LbDataStore,Im_N);
IB = labeloverlay(I, Ib, 'Colormap', cmap, 'Transparency',0.5);

% Mostrar el resultado de la segmentación semántica
C = semanticseg(I, net);
CB = labeloverlay(I, C, 'Colormap', cmap, 'Transparency',0.5);
figure
imshowpair(IB,CB,'montage')
title('imagen original vs segmentada')

 % Añadir una barra de colores al eje. La barra de color está ajustada para
mostrar el nombre de la clase junto con el color.

            colormap(gca,cmap)

            % añadir una barra de colores a la imagen actual.
            c = colorbar('peer', gca);

            % usar los nombres de las clases para cada color.
            c.TickLabels = classes;
            numClasses = size(cmap,1);

            % centrar.
            c.Ticks = 1/(numClasses*2):1/numClasses:1;

            % Eliminar marcas.
            c.TickLength = 0;
```

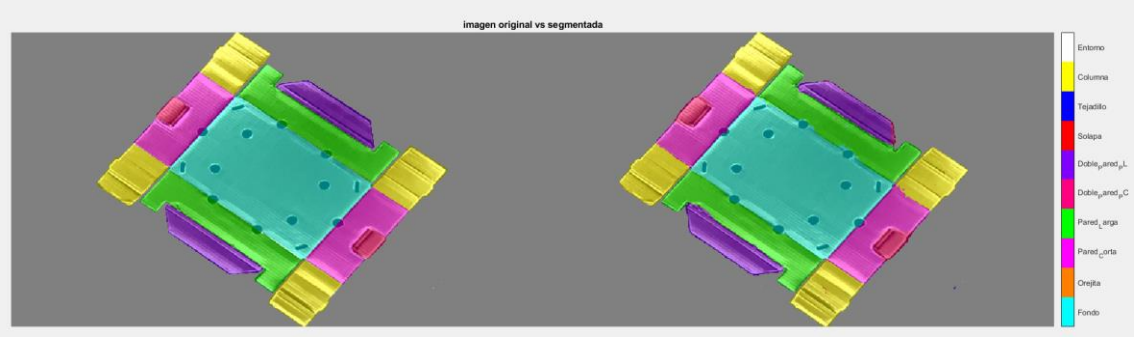


Figura 168 Imagen original vs imagen segmentada

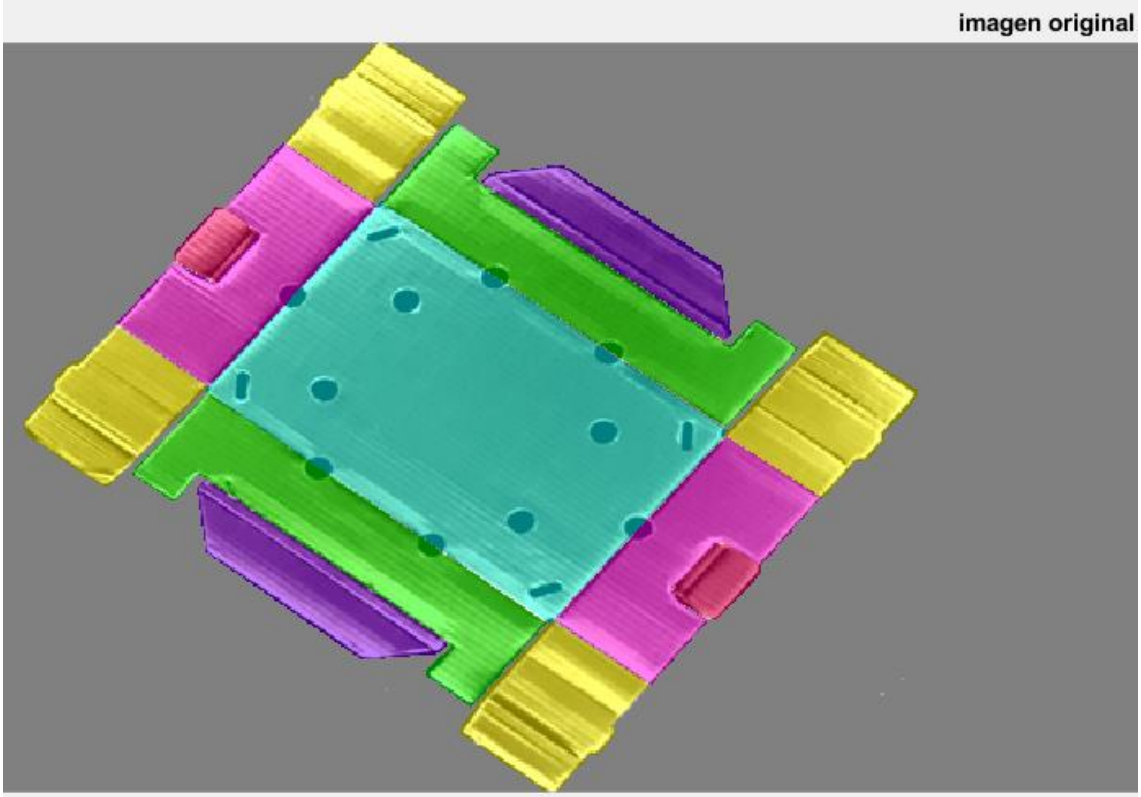


Figura 169 Imagen original

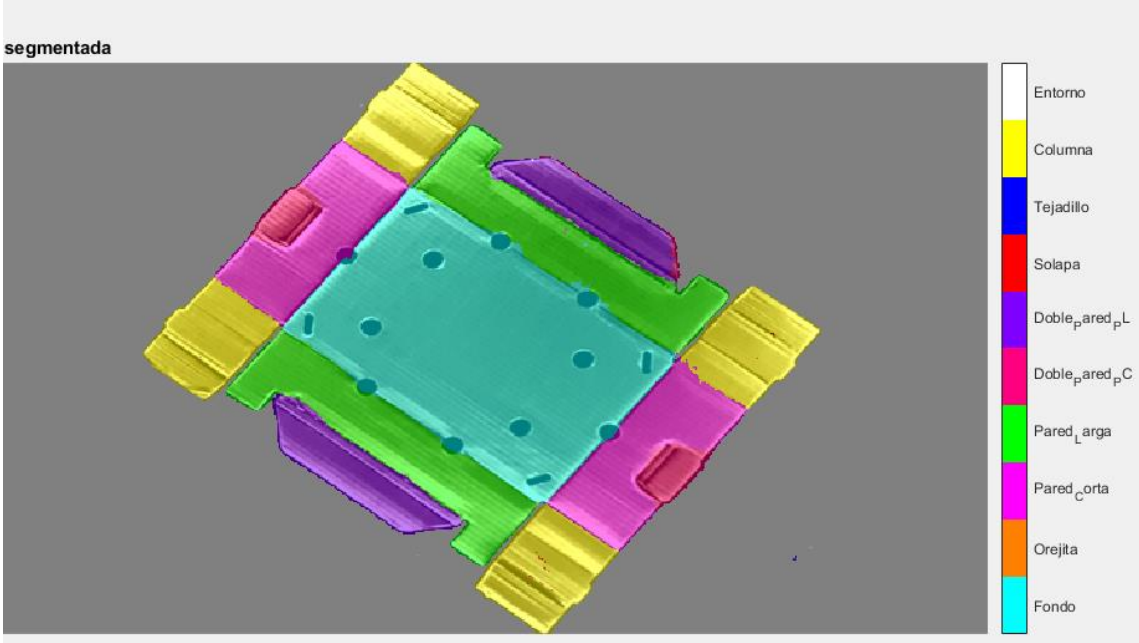


Figura 170 Imagen segmentada

6.3. Evaluación de los resultados

6.3.1. Probar la Red sobre imagen nueva.

Para comprobar la precisión de la segmentación semántica, la aplicamos sobre una imagen de prueba que no pertenece a la base de datos de entrenamiento. Como esta imagen será nueva para la red, ésta intentará aplicar lo que ha aprendido para segmentarla.

```
%% Probar la Red

ITEST = read(imdsTest);
tic
C = semanticseg(ITEST, net);
toc

B = labeloverlay(ITEST, C, 'Colormap', cmap, 'Transparency',0.5);
figure
imshow(B)
title('segmentación semantica')

% Añadir una barra de colores que serve como leyenda para identificar las
 % etiquetas.

            colormap(gca,cmap)

            % Añadir la barra de colores a la imagen.
            c = colorbar('peer', gca);

            % Usar nombres de clases.
            c.TickLabels = classes;
            numClasses = size(cmap,1);

            % Centrar.
            c.Ticks = 1/(numClasses*2):1/numClasses:1;


            c.TickLength = 0;
```

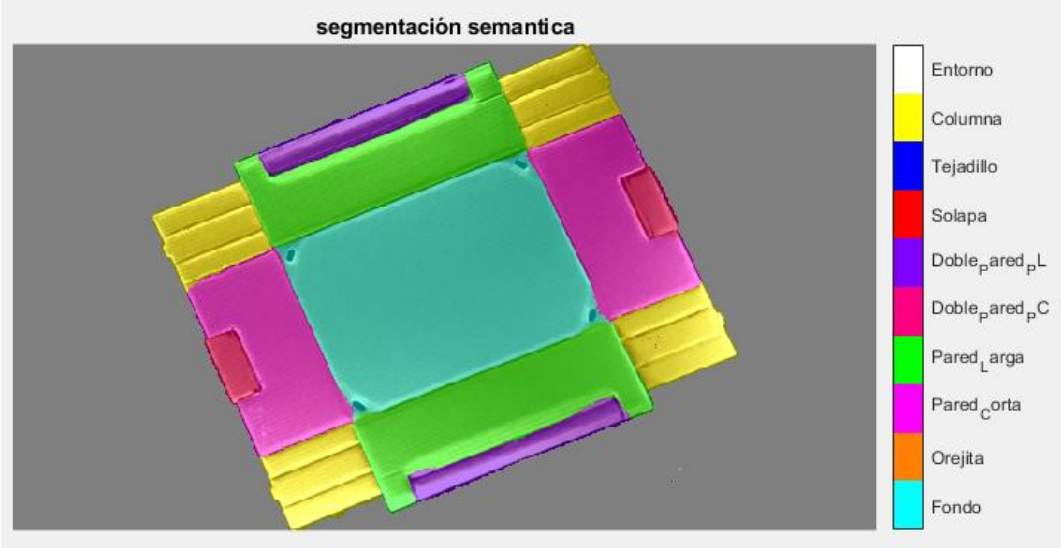


Figura 171 imagen de prueba plancha columna

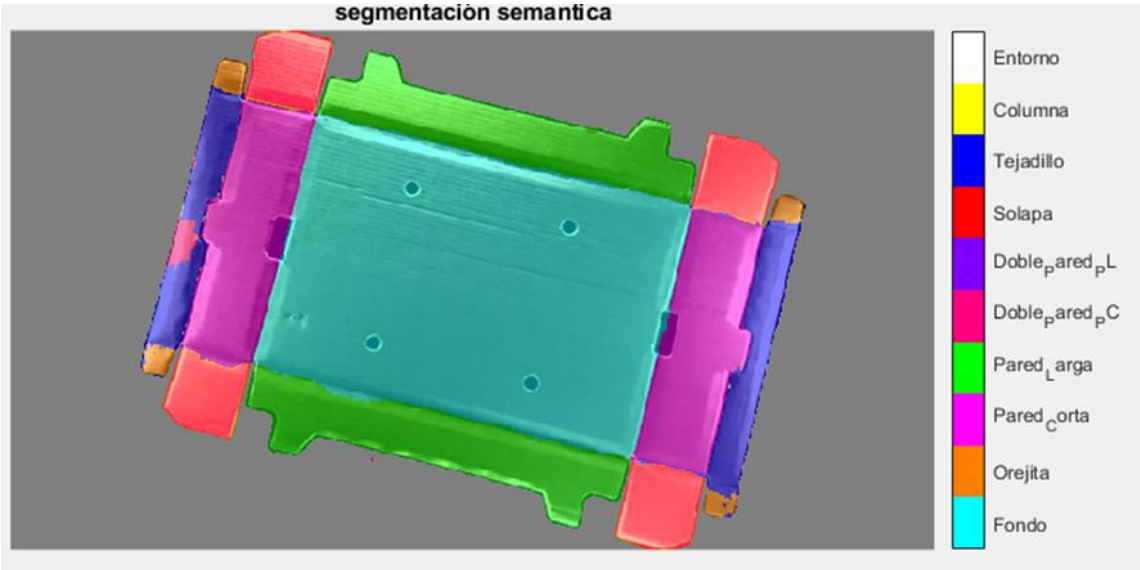


Figura 172 Imagen de prueba plancha Plaform

A pesar de la poca cantidad de imágenes que se han utilizado para entrenar la red, a simple vista estos resultados cumplen en gran medida los requisitos del presente trabajo en cuanto a la localización de las diferentes características y su correcta segmentación, aunque en algunos casos la delimitación entre algunas características no alcanza la precisión esperada debido a la aparición de falsos positivos.

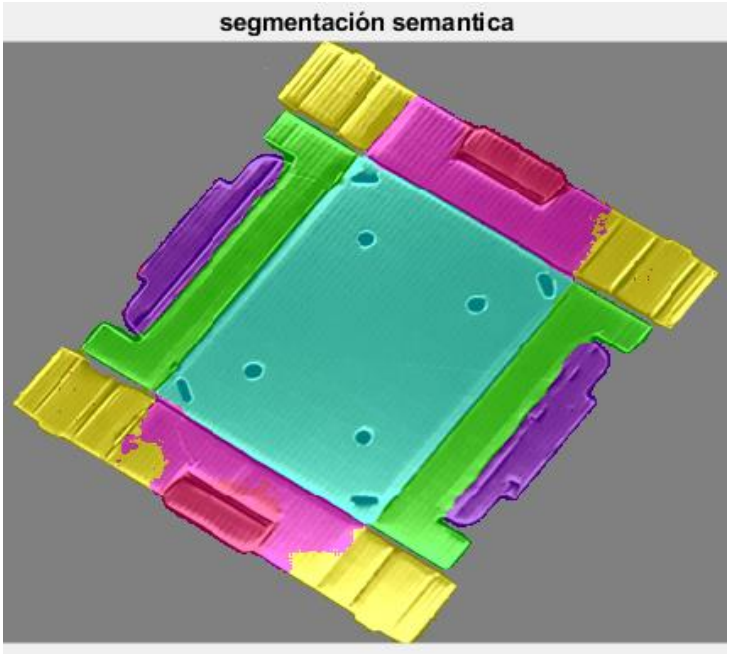


Figura 173 Ejemplo de plancha con falsos positivos

La imagen anterior presenta algunos falsos positivos debido a varios factores relacionados con la precisión de la red entrenada, y la calidad de la imagen original como los reflejos de la luz que disminuyen el contraste entre las características.

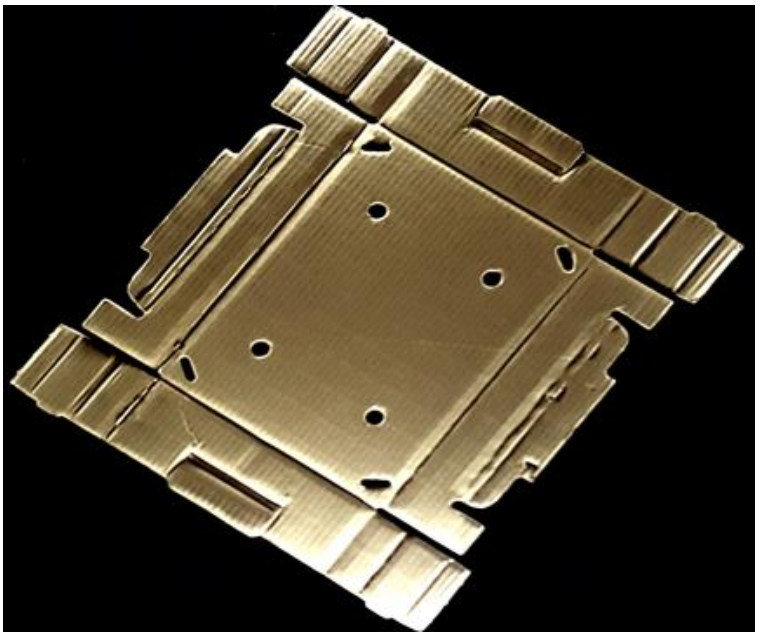


Figura 174 Plancha original con reflejos

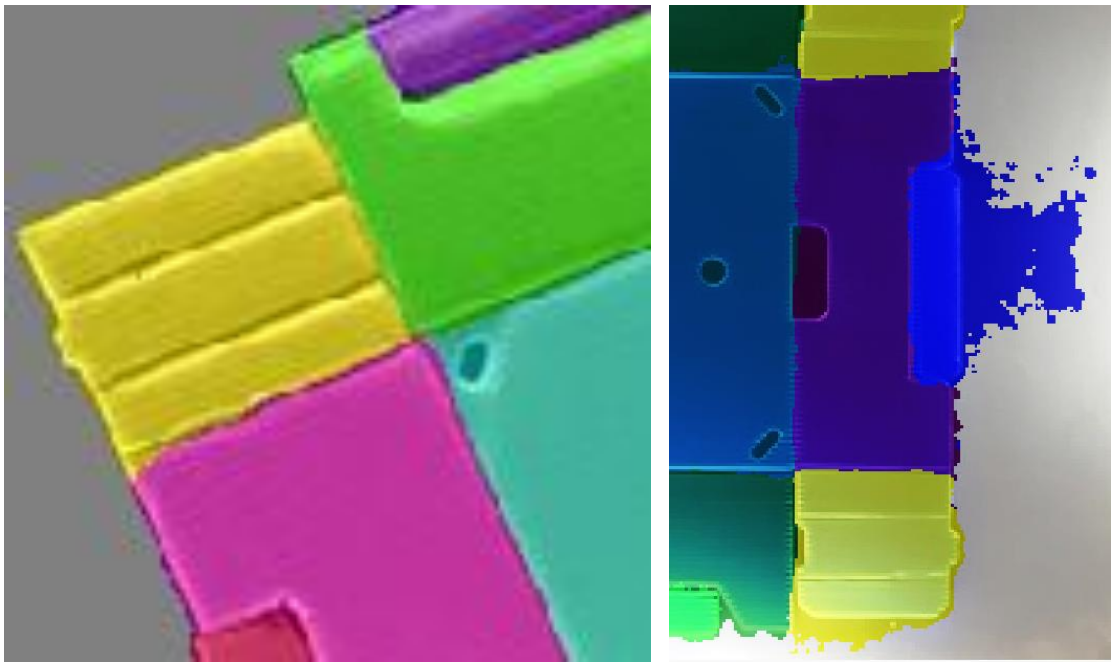


Tabla 11 Delimitación de zonas con eliminación de fondo (izquierda) y sin eliminación de fondo (derecha)

Las operaciones llevadas a cabo en la fase de preprocesamiento para eliminar el fondo y aislar a las planchas ha dado su fruto para delimitar el contorno de la plancha y evitar la aparición de falsos positivos fuera de la plancha causados por los reflejos de iluminación.

6.3.2. Etiquetación manual vs segmentación.

A continuación, realizamos una comparación entre una imagen de la base de datos de test con las etiquetas realizadas manualmente, y la misma imagen sometida a la segmentación semántica. Mostramos las imágenes en escala de grises para poder medir mejor las diferencias.

```
%% Comparar las imagenes
ResultadoEsp = read(pxdsTest); % cargar una imágen con las etiquetas manuales.
ITEST2 = read(imdsTest); % cargar una imagen nueva
C2 = semanticseg(ITEST2, net); % aplicar la segmentación semántica
esperado = uint8(ResultadoEsp); % mostrar en escala de grises
previsto = uint8(C2);
% Image Processing toolbox
imshowpair(esperado, previsto, 'montage')
title('etiquetas manuales vs etiquetas segmentación semantica')
% reset(pxdsTest); %resetear el contador cuando llega al final
% reset(imdsTest);
```

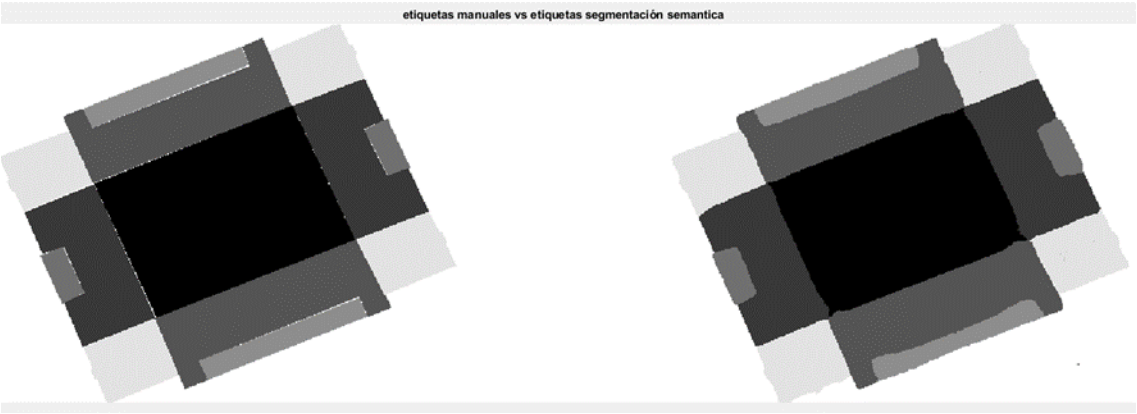


Figura 175 Etiquitas manuales vs la segmentación en plancha de Columna

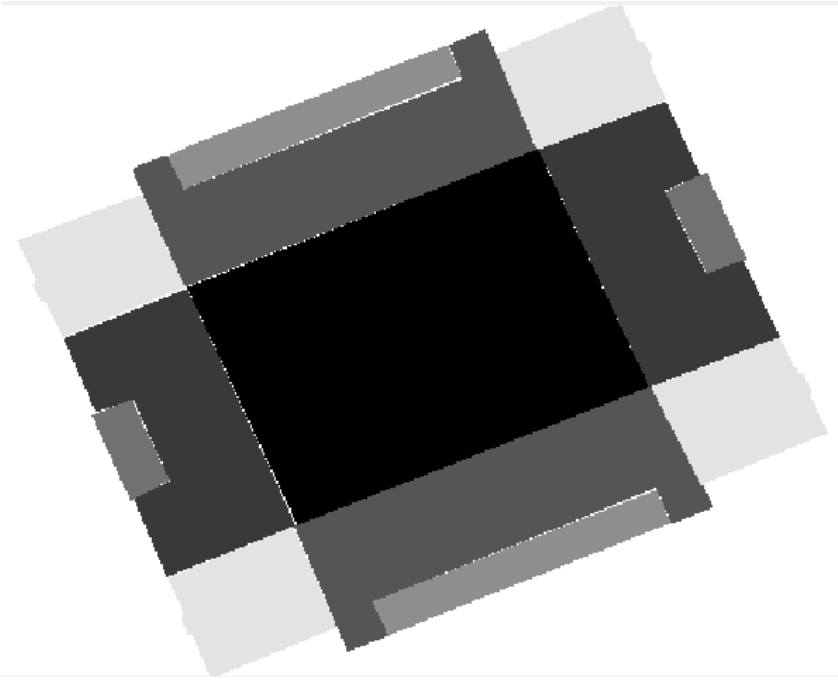


Figura 176 Etiquitas manuales de una plancha Columna

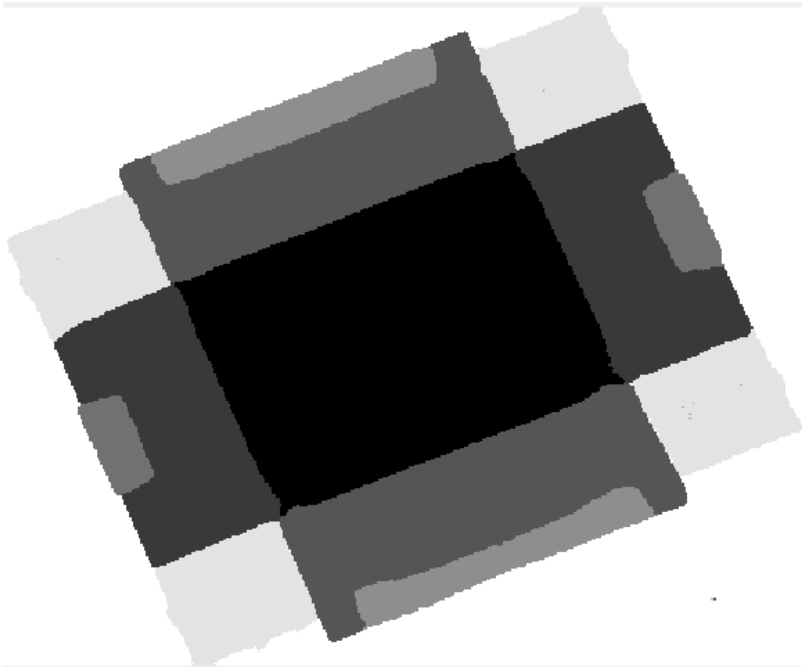


Figura 177 Segmentación semántica equivalente

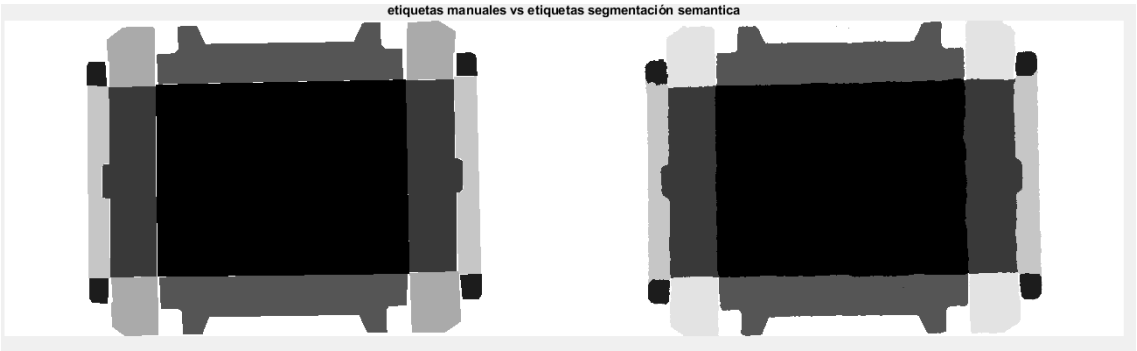


Figura 178 Figura 175 Etiquitas manuales vs la segmentación en plancha de Plaform

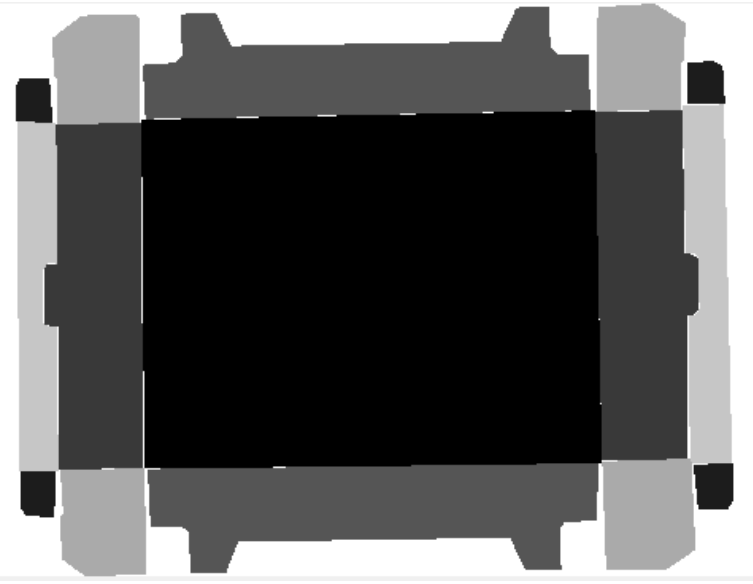


Figura 179 Figura 176 Etiquitas manuales de una plancha Plaform

Figura 180 Figura 177 Segmentación semántica equivalente

Visualmente, los resultados de la segmentación semántica coinciden mejor en clases como el fondo, pared larga, pared corta y columna. Sin embargo, la segmentación de los objetos más pequeños como la doble pared corta y doble pared larga no son tan precisos. El grado de coincidencia de las etiquetas manuales y automáticas por clase se puede medir usando la métrica de intersección sobre unión *(IoU)* conocida como el índice de *Jaccard (Jaccard Index)*, y que se calcula de la siguiente forma.

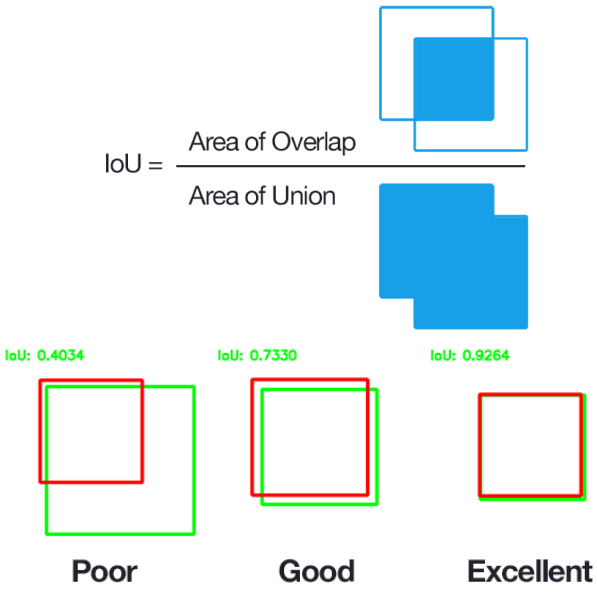


Figura 181 Intersección sobre Unión [76]

```
391.  %% Índice de Jaccard
392.  iou = jaccard(C2, ResultadoEsp);
393.  table(classes,iou) % mostrar el grado de coincidencia por cada clase.
```

```
ans =

  10×2 table

      classes          iou
    ____________    _______

    'Fondo'         0.98071
    'Orejita'           NaN
    'Pared_Corta'    0.9562
    'Pared_Larga'   0.94754
    'Doble_Pared_PC' 0.85925
    'Doble_Pared_PL'  0.856
    'Solapa'             0
    'Tejadillo'          0
    'Columna'       0.95559
    'Entorno'       0.98872
```

```
ans =

  10×2 table

      classes          iou
    ____________    _______

    'Fondo'         0.98441
    'Orejita'       0.85888
    'Pared_Corta'   0.96616
    'Pared_Larga'   0.95871
    'Doble_Pared_PC'     0
    'Doble_Pared_PL'     0
    'Solapa'        0.82641
    'Tejadillo'     0.93326
    'Columna'            0
    'Entorno'       0.97572
```

Figura 182 IoU :Izquierda: modelo Columna - Derecha: modelo Plaform

El índice de Jaccard tiene valores muy próximos a 1 en la mayoría de las clases lo que quiere decir que existe mayor grado de coincidencia entre los resultados de la segmentación semántica y el etiquetado manual. Este valor es más alto para los objetos con más área porque la cantidad de pixeles de los falsos positivos es insignificante comparada con la cantidad total de pixeles del objeto detectado. Los valores nulos que aparecen en la tabla pertenecen a clases que no se han detectado porque no forman parte del modelo de plancha segmentado.

Existen otras métricas comunes para evaluar la coincidencia incluido el índice de Dice (Dice index) y la Puntuación de coincidencia de contorno Boundary-F1.

```
394.  %% Índice de Dice
395.  IndiceDice = dice(C2,ResultadoEsp);
396.  table(classes,IndiceDice)
```

```
ans =

  10×2 table

      classes        IndiceDice
    ____________    ____________

    'Fondo'          0.99026
    'Orejita'            NaN
    'Pared_Corta'    0.97761
    'Pared_Larga'    0.97306
    'Doble_Pared_PC'  0.9243
    'Doble_Pared_PL' 0.92241
    'Solapa'                0
    'Tejadillo'             0
    'Columna'        0.97729
    'Entorno'        0.99433
```

Figura 183 Índice de Dice

6.3.3. Efecto de equilibrar las clases.

Para ver el efecto que tiene el hecho de equilibrar las clases sobre el rendimiento de la red neuronal, se ha entrenado la red ignorando este paso. Así, es más fácil comparar los resultados de la segmentación con datos cuantitativos de los dos modos de entrenamiento.

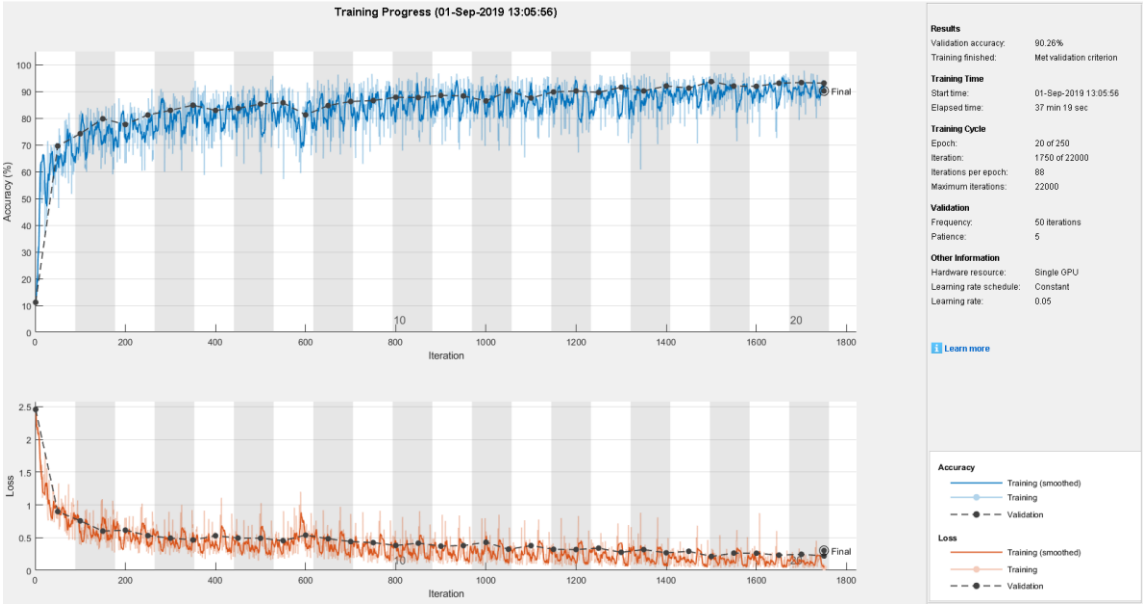


Figura 184 Progreso del proceso de entrenamiento sin equilibrar las clases.

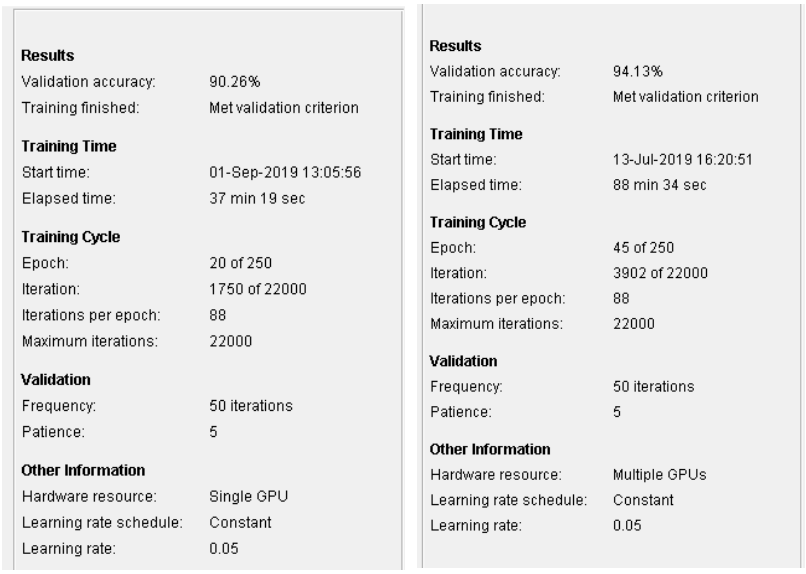


Figura 185 resultados de entrenamiento antes de equilibrar las clases (izquierda) y después (Derecha)

Como se puede observar, equilibrar las clases ha conseguido incrementar la exactitud de la red en un 4.13%. Sin embargo, el entrenamiento tras equilibrar las clases se ha concluido en el doble número de iteraciones tardando el doble de

tiempo.

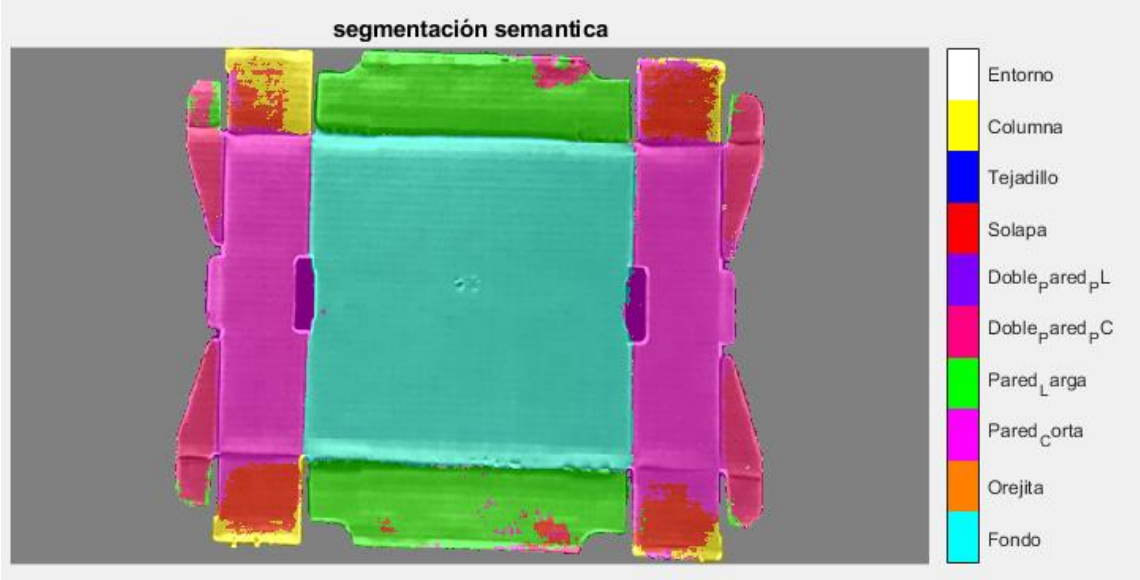


Figura 186 Imagen del modelo Plaform sin equilibrar las clases

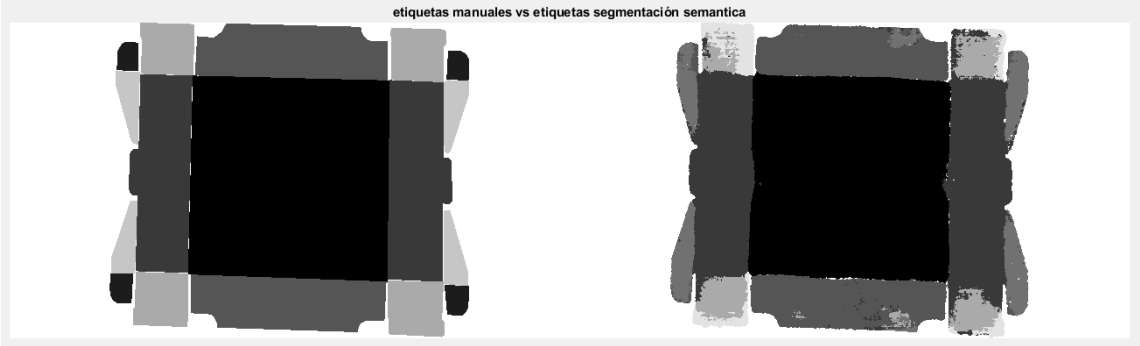


Figura 187 Etiquitas manuales vs la segmentación en plancha de Plaform sin equilibrar las clases

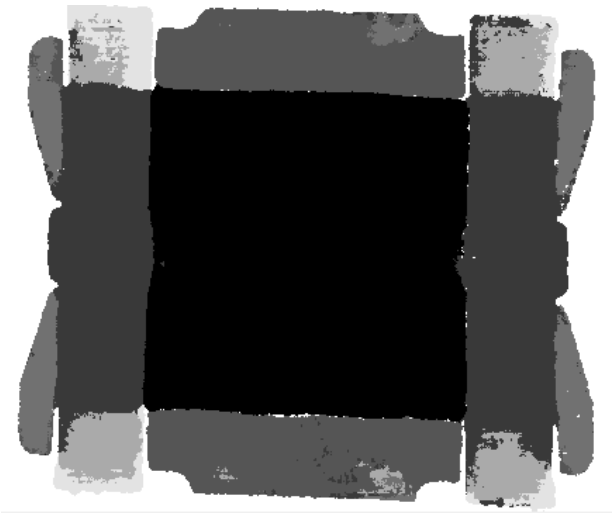


Figura 188 Resultado de la segmentación sobre plancha Plaform sin equilibrar las clases

El resultado final sin equilibrar las clases no alcanza la calidad esperada debido a la aparición de muchos falsos positivos en las clases que representan a los píxeles de menos frecuencia como la *Orejeta,* la *Solapa* y el *tejadillo.*

classes	iou
'Fondo'	0.9791
'Orejita'	0.0015337
'Pared_Corta'	0.83753
'Pared_Larga'	0.88038
'Doble_Pared_PC'	0
'Doble_Pared_PL'	NaN
'Solapa'	0.49122
'Tejadillo'	0.00030414
'Columna'	0
'Entorno'	0.98117

classes	iou
'Fondo'	0.98441
'Orejita'	0.85888
'Pared_Corta'	0.96616
'Pared_Larga'	0.95871
'Doble_Pared_PC'	0
'Doble_Pared_PL'	0
'Solapa'	0.82641
'Tejadillo'	0.93326
'Columna'	0
'Entorno'	0.97572

Figura 189 IoU :Izquierda: Plaform sin equilibrar - Derecha: Plaform después de equilibrar

Aunque el rendimiento global de la red sin equilibrar las clases es relativamente alto, el rendimiento individual en las clases minoritarias es demasiado bajo debido al desequilibrio entre las clases. Pues la red ha enfocado más su aprendizaje en las clases con mayor presencia.

6.3.4. Métricas y matriz de confusión.

Ejecutamos la función *evaluateSemanticSegmentation* para la evaluación de la red neuronal entrenada y el cálculo de las principales métricas sobre los resultados de la segmentación semántica de todo el conjunto de imágenes de Test.

```
%% Evaluación de la red entrenada
reset(pxdsTest); %resetear el contador cuando llega al final
reset(imdsTest);
% realizar la segmentación semántica
pxdsResultados =
semanticseg(imdsTest,net,'WriteLocation',tempdir,'Verbose',false,'ExecutionEnviro
nment','cpu');

% evaluar los resultados de la segmentación semántica frente a ground truth
metrics =
evaluateSemanticSegmentation(pxdsResultados,pxdsTest,'Verbose',false);

metrics.DataSetMetrics % métricas a nivel de conjunto de datos.
metrics.ClassMetrics % métricas a nivel de clases.
```

La función *semanticseg* devuelve los resultados para el conjunto de datos de Test como un objeto de pixelLabelDataStore. Los datos de etiquetas a nivel de pixel de cada imagen de prueba se almacenan en el directorio especificado por la función *WriteLocation.*

ans =

1×5 table

GlobalAccuracy	MeanAccuracy	MeanIoU	WeightedIoU	MeanBFScore
0.94011	0.78226	0.6662	0.89607	0.87389

Figura 190 Métricas globales

Estas métricas proporcionan una visión de alto nivel sobre el rendimiento de la red neuronal. En la siguiente tabla se muestra el impacto que tiene cada clase sobre el rendimiento global de la red.

ans =

10×3 table

	Accuracy	IoU	MeanBFScore
Fondo	0.98434	0.96631	0.93402
Orejita	0.57223	0.55729	0.82779
Pared_Corta	0.9441	0.80252	0.81473
Pared_Larga	0.88494	0.85466	0.88846
Doble_Pared_PC	0.90703	0.46305	0.72073
Doble_Pared_PL	0.76203	0.70437	0.85313
Solapa	0.72231	0.76795	0.70147
Tejadillo	0.79412	0.62717	0.73017
Columna	0.9862	0.70351	0.91351
Entorno	0.98766	0.98311	0.93425

Figura 191 Métricas según clase

Aunque el rendimiento global es alto, las métricas a nivel de clases demuestran que las clases menos representadas en los datos de entrenamiento, como la orejita y la doble pared larga, no se han segmentado bien comparado con las clases que representan objetos de mayor área, como el fondo y las paredes cortas y largas. Esto afecta directamente al rendimiento global de la red. Para disminuir este efecto y mejorar más el rendimiento de la red, hay que incluir más muestras de las clases menos representadas.

Para visualizar el desempeño de nuestra red entrenada, construimos la matriz de confusión con los valores proporcionados por el algoritmo.

metrics | metrics.ConfusionMatrix

metrics.ConfusionMatrix

	1 Fondo	2 Orejita	3 Pared_Corta	4 Pared_Larga	5 Doble_Pared_PC	6 Doble_Pared_PL	7 Solapa	8 Tejadillo	9 Columna	10 Entorno
1 Fondo	793295	0	7129	5208	5	0	0	0	123	155
2 Orejita	0	14455	1	3	37	9	44	9864	230	662
3 Pared_Corta	3111	0	429233	3406	2876	9	0	271	12723	3020
4 Pared_Larga	9726	138	38401	455157	3540	3433	0	82	1146	2714
5 Doble_Pared_PC	0	0	803	16	27921	52	0	1824	0	167
6 Doble_Pared_PL	1	0	39	2429	10862	50564	0	2	1913	544
7 Solapa	513	0	5654	369	1050	4	28180	1686	7291	1458
8 Tejadillo	3	52	1725	34	7386	283	0	44577	211	1863
9 Columna	0	0	925	384	1113	34	132	0	311818	1774
10 Entorno	1680	487	5531	6376	2646	1608	0	1213	13414	2638207

Figura 192Figura 186 Matriz de confusión generada

Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real. Uno de

los beneficios de las matrices de confusión es que facilitan ver si el sistema está confundiendo dos clases.

Si en los datos de entrada el número de muestras de clases diferentes cambia mucho, la tasa de error del clasificador no es representativa de lo bien que realiza la tarea el clasificador. Si por ejemplo hay más muestras de la clase 1 que de la clase 2, el clasificador puede tener fácilmente un sesgo hacia la clase 1. Si el clasificador clasifica todas las muestras como clase 1 su precisión será del 99%. Esto no significa que sea un buen clasificador, pues tuvo un 100% de error en la clasificación de las muestras de la clase 2.[77]

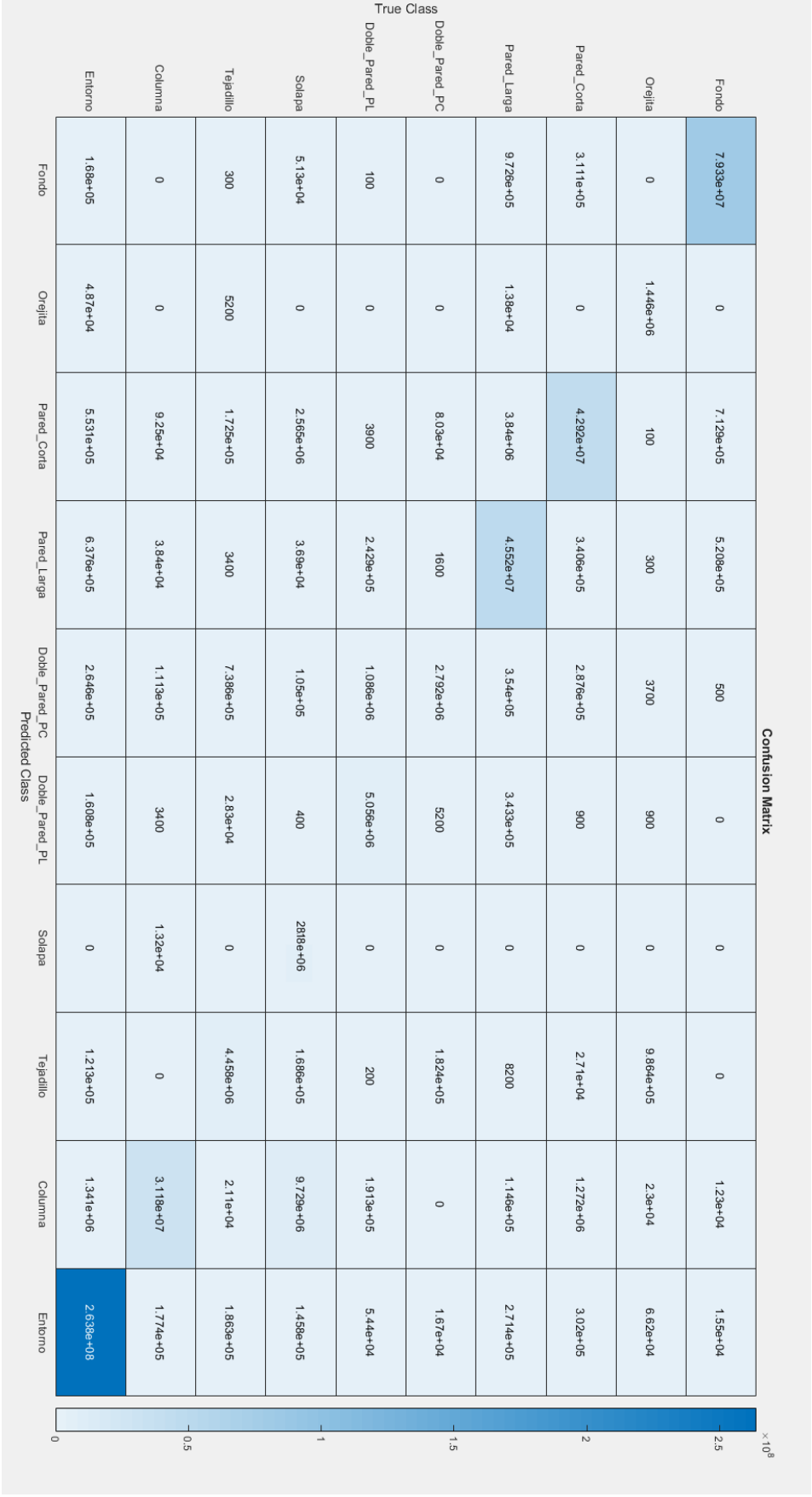


Figura 193 matriz de confusión predicciones vs instancias reales

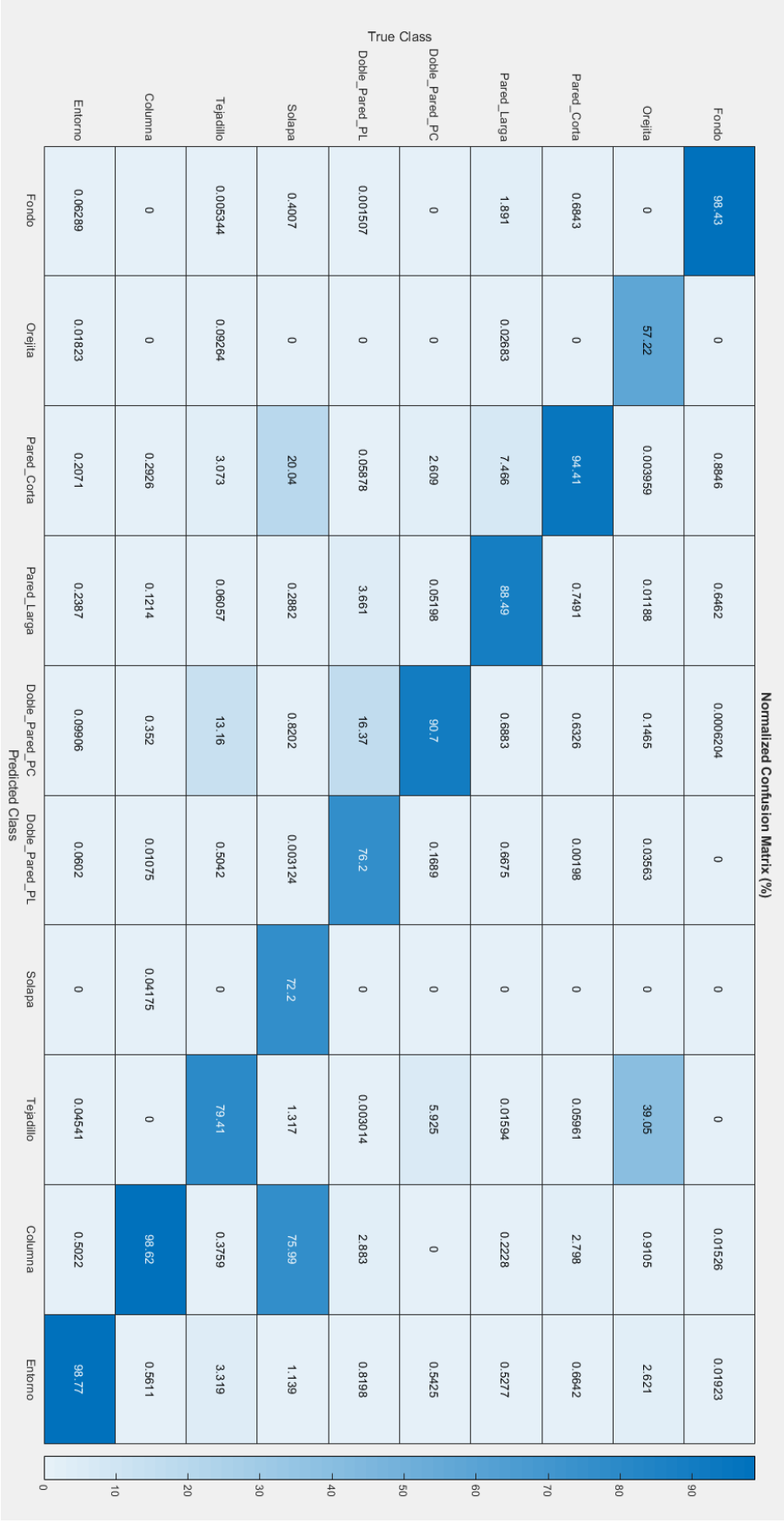


Figura 194 Matriz de confusión normalizada

Para analizar la matriz de confusión, nos apoyamos en sus métricas básicas:

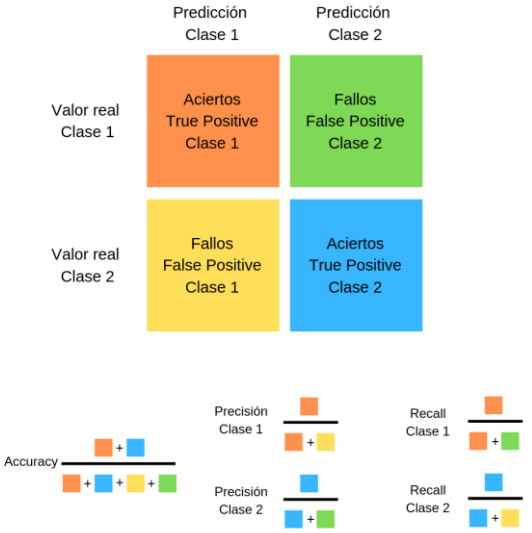


Figura 195 Métricas de una matriz de confusión

		Predicción	
		Positivos	**Negativos**
Observación	**Positivos**	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

Tabla 12 Observación vs Predicción [78]

Siendo:

- **VP** es la cantidad de positivos que fueron clasificados correctamente como positivos por el modelo.
- **VN** es la cantidad de negativos que fueron clasificados correctamente como negativos por el modelo.
- **FN** es la cantidad de positivos que fueron clasificados incorrectamente como negativos.
- **FP** es la cantidad de negativos que fueron clasificados incorrectamente como positivos.

Exactitud (Accuaracy) : Responde a la pregunta ¿Cuál es la proporción de predicciones correctas?

$$Exactitud = \frac{VP + VN}{Total} \qquad [14]$$

Sensibilidad (Recall) : Expresa cuan bien puede el modelo detectar a esa clase. responde a la pregunta ¿Qué proporción de positivos reales se han predicho correctamente?

$$Sensibilidad = \frac{VP}{VP + FN} \qquad [15]$$

Precisión (Precision) : Es básicamente el número total de predicciones correctas dividido por el número total de predicciones. Responde a la pregunta ¿Qué proporción de predicciones positivas es correcta?

$$Precisión = \frac{VP}{VP + FP} \qquad [16]$$

Especificidad: Es la tasa de verdaderos negativos. Responde a la pregunta: Cuando la clase es negativa, ¿qué porcentaje logra clasificar?

$$Especificidad = \frac{VN}{VN + FP} \qquad [17]$$

Tasa de error (Missclassification rate): Es el porcentaje de datos clasificados incorrectamente.

$$Tasa\ de\ error = \frac{FP + FN}{FP + FN + VP + VN} \qquad [18]$$

Conforme a estas métricas hay cuatro posibles casos para cada clase:

- **Alta precisión y alto recall:** el modelo maneja perfectamente esa clase.
- **Alta precisión y bajo recall:** el modelo no detecta la clase muy bien, pero cuando lo hace es altamente confiable.
- **Baja precisión y alto recall:** el modelo detecta bien la clase, pero también incluye muestras de otras clases.
- **Baja precisión y bajo recall:** El modelo no logra clasificar la clase correctamente.

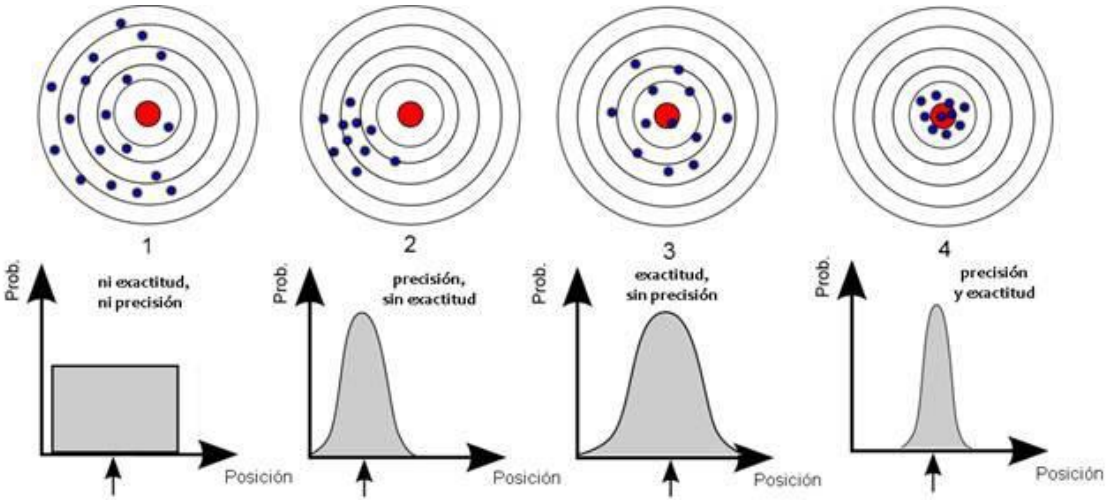


Figura 196 Exactitud vs precisión

En las siguientes tablas se muestran las métricas según cada clase:

	VP	FN	FP	VN
1 - Fondo	793295	12620	15034	4000112
2 - Orejita	14455	10850	677	4778952
3 – Pared Corta	429233	25416	60208	4364174
4 – Pared Larga	455157	59180	18225	4338250
5 – Doble Pared PC	27921	2862	29515	4765486
6 - Doble Pared PL	50564	15790	5432	4742843
7 - Solapa	28180	18025	176	4765227
8 - Tejadillo	44577	11557	14942	4748830
9 - Columna	311818	4362	37051	4481589
10 - Contorno	2638207	32955	12357	2155200

	1	2	3	4	5	6	7	8	9	10
Sensibilidad	0,984	0,571	0,944	0,884	0,907	0,762	0,722	0,794	0,986	0,987
Precisión	0,981	0,955	0,876	0,961	0,486	0,902	0,993	0,748	0,893	0,995
Especificidad	0,996	0,999	0,986	0,995	0,993	0,998	0,999	0,996	0,991	0,994
Error	0,0057	0,0023	0,0175	0,0158	0,0067	0,0044	0,0037	0,0054	0,0085	0,0093

Tabla 13 Métricas según cada clase

Las métricas de la tabla anterior muestran que el modelo maneja perfectamente la clase *1-Fondo* con valores de sensibilidad y precisión muy próximos a la unidad y una tasa de error relativamente baja. Esto se puede explicar principalmente por la gran cantidad de muestras disponibles en los datos de entrenamiento ya que se trata de la clase con mayor número de píxeles por imagen según el histograma de frecuencia por clase visto anteriormente. También puede ser debido a que esta clase pertenece a todos los modelos de planchas presentes en las imágenes lo que hace que su tasa de aprendizaje sea muy alta. Otros factores que pueden afectar en el rendimiento de la red pueden ser la posición céntrica de la característica y su lejanía de las fuentes de iluminación lo que disminuye los reflejos de forma considerable, y su forma simple que hace que el modelo la distingue más fácilmente comparada con otras clases con formas más complejas.

Por otro lado, para la segunda clase *2-Orejeta*, el modelo presenta una sensibilidad relativamente baja frente a una precisión muy alta y una tasa de error baja. Lo que puede significar que la red ha tenido algún problema para detectar la característica. Cuando el *dataset* presenta un desequilibrio como en este caso, suele ocurrir que se obtienen resultados similares. Esto es debido a la poca cantidad de muestras en los datos de entrenamiento ya que se trata de la clase con menor número de píxeles por imagen según el histograma de frecuencia por clase visto anteriormente. También hay que tener en cuenta que esta característica solo pertenece al modelo de plancha *Plaform* lo que hace que las imágenes relacionadas con el modelo de plancha *Columna* no se aprovechan para aprender esta clase.

En el caso de las clases 3 y 4 correspondientes a las paredes largas y cortas se obtienen valores de sensibilidad y precisión altos como en el caso de la primera clase debido a las mismas razones. Sin embargo, la tasa de error es superior al resto de clases debido a la confusión con otras clases como se puede observar en la matriz de confusión dado que estas dos características se parecen mucho en cuanto a su forma y posición respecto al centro según la orientación que se le da a la plancha en cada imagen.

La clase 5 correspondiente a la doble pared corta presenta un valor de precisión alto y una sensibilidad relativamente baja. Esto quiere decir que el modelo no detecta bien la clase debido a las confusiones con otras clases como se puede observar claramente en la matriz de confusión. Estas confusiones ocurren sobre todo con la clase 6 correspondiente a la doble pared larga y la clase 8 correspondiente al tejadillo debido a sus formas similares. Hay que observar

también que la característica de *Tejadillo* en el caso de modelos de plancha *Plaform* presenta una similitud alta con la característica de *Doble pared corta* en el caso de modelos de plancha Columna, ya que las dos características tienen una forma y orientación similares y comparten sus fronteras con la misma característica *Pared corta.*

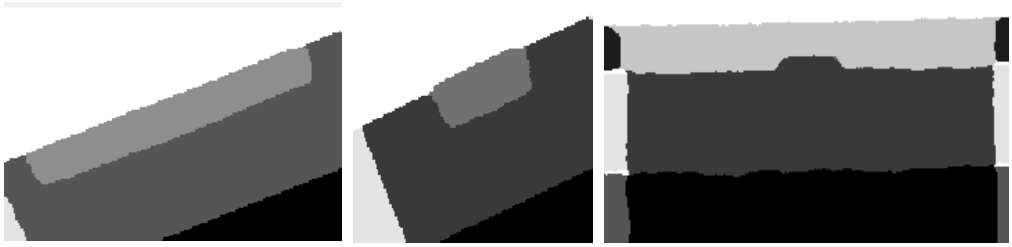


Figura 197 Doble pared larga - doble pared corta (columna) – tejadillo (plaform)

El resto de clases muestra unas métricas similares a la primera clase excepto en la tasa de error para la clase 10 correspondiente al *Contorno* que es relativamente alta debido a algunas confusiones con los píxeles de las otras clases dado que esta clase está en contacto con casi todas ellas.

Capítulo 7

7. Conclusiones y líneas futuras.

7.1. Introducción.

Este capítulo expone el conjunto de conclusiones obtenidas tras el desarrollo del trabajo, así como una serie de propuestas para futuras líneas de investigación que deriven de este trabajo.

7.2. Conclusiones.

El presente trabajo se centra en la segmentación semántica de las características de planchas destinadas a la formación de cajas de cartón ondulado mediante el uso de una red neuronal VGG16 preentrenada.

La primera fase consiste en construir el sistema de visión artificial para la adquisición de imágenes que posteriormente formarán parte de la base de datos del algoritmo de segmentación. para ello, se han valorado diferentes técnicas de iluminación y se ha elegido la técnica de *Dark Field* por aportar una mejora considerable de contraste en los bordes y pliegues internos de las planchas. Después, se ha procedido a la construcción de una caja de iluminación para la adquisición de imágenes, formada por una caja de cartón con un tamaño adecuado para diferentes dimensiones de planchas y un fondo oscuro para mejorar el contraste, 4 lámparas LED fijadas en los cuatro lados de la caja a una altura de 20 mm de la base con unas sombrillas fijadas en su parte superior para evitar reflejos directos, y la cámara de un celular fijada en un trípode con altura regulable.

La segunda fase del presente trabajo consiste en desarrollar el algoritmo de preprocesamiento de las imágenes adquiridas. Como paso previo, se ha convertido el formato de las imágenes a PNG por utilizar un algoritmo de compresión sin pérdidas para evitar la alteración de la calidad de las imágenes en las diferentes transformaciones. El algoritmo de preprocesamiento consiste en primer lugar en la umbralización de color para la eliminación del fondo que

contiene reflejos y degradaciones de color que pueden distraer a la red neuronal durante la segmentación. Para ello, se ha apoyado en la aplicación de *Matlab Color Thresholder* para identificar el umbral de colores de la plancha y eliminar el resto de los colores y reemplazarlos con el color negro. Para la umbralización, se ha utilizado el espacio de color L*a*b por ser el mejor en separar los colores de la plancha de los reflejos del fondo. En segundo lugar, se han recortado las imágenes a los puntos extremos del borde de las planchas para eliminar el resto de los reflejos que quedan en los bordes de las imágenes, mediante la detección de aglomeraciones de píxeles y la extracción del conjunto de píxeles *Blobs* de mayor tamaño. En tercer lugar, se han redimensionado todas las imágenes para unificar su dimensión y relación de aspecto tras el recorte que han sufrido en el paso anterior.

La tercera fase consiste en el desarrollo del algoritmo de segmentación semántica utilizando la red neuronal preentrenada *VGG-16.* Para ello, se ha utilizado en primer lugar, la aplicación de *Matlab Image Labeler* para etiquetar las imágenes manualmente asignando a las diferentes regiones que representan las características de la plancha una etiqueta y un color con el apoyo de una tableta gráfica para mayor precisión y comodidad. En segundo lugar, se ha procedido a la preparación de los datos para el entrenamiento mediante la instalación de la red *VGG16* en *Matlab* y la carga de las imágenes de planchas y sus correspondientes imágenes etiquetadas junto con sus datos de *Gound truth.* Las imágenes se han dividido en conjunto de entrenamiento y conjunto de Test y validación con porcentajes de 80% y 20% respectivamente. En tercer lugar, se ha utilizado la función *SegnetLayers* para crear la red neuronal *SegNet* inicializada usando los pesos de la red neuronal VGG-16. Al consultar las estadísticas, se ha dado cuenta de que las clases en el conjunto de datos no están equilibradas, para solucionar este problema, se han utilizado los pesos de las clases para equilibrarlas mediante el uso de la frecuencia promedio para conseguir darle más resalte a las clases con píxeles de menor frecuencia en las imágenes de la base de datos. Debido a que la cantidad de imágenes es relativamente baja, se ha utilizado la función *DataAumentation* en el entrenamiento para proporcionar a la red más datos y incrementar su robustez frente a diferentes alteraciones geométricas que pueden sufrir algunas imágenes. Como motor de procesamiento se ha utilizado la tarjeta gráfica para multiplicar el numero operaciones por segundo gracias al procesamiento en paralelo.

Una vez vistos los resultados de la segmentación semántica se ha procedido a su evaluación y la exposición de las siguientes conclusiones:

- El entrenamiento se ha llevado a cabo en 45 épocas y 3902

iteraciones con una duración de 1:28 min :34 s y ha alcanzado una precisión máxima de validación de 97.79%.

- A pesar de la poca cantidad de imágenes utilizadas en el entrenamiento, a simple vista se nota que los resultados cumplen en gran medida con los requisitos de este trabajo en cuanto a la localización de las características de las planchas y su correcta segmentación, aunque en algunos casos la delimitación entre algunas regiones no alcanza la precisión máxima debido a la aparición de falsos positivos.

- Las operaciones llevadas a cabo en la fase de preprocesamiento para eliminar el fondo y aislar las planchas ha dado su fruto evitando la aparición de falsos positivos fuera de los bordes de las planchas causados por los reflejos de la iluminación.

- El índice de Jaccard tiene valores muy próximos a la unidad en la mayoría de las clases lo que significa que existe un mayor grado de coincidencia entre los resultados de la segmentación semántica y el etiquetado manual. Este valor es más alto para las clases con mayor área debido a que la cantidad de píxeles de los falsos positivos es insignificante comparada con la cantidad total de píxeles del objeto detectado.

- Aunque el rendimiento global es alto, las métricas a nivel de clases demuestran que las clases menos representadas en los datos de entrenamiento no se segmentan con la misma eficacia que las clases más representadas. Para disminuir el efecto de este hecho sobre el rendimiento global de la red habrá que incluir más muestras de las clases menos representadas en el conjunto de datos.

- Las métricas demuestran que el modelo maneja perfectamente las clases mayoritarias como el *Fondo* con valores de precisión y sensibilidad muy cerca de la unidad y con una tasa de error baja. Esto puede ser explicado principalmente por la gran cantidad de muestras disponibles en los datos de entrenamiento.

- En el caso de la clase *Orejeta*, el modelo presenta una sensibilidad relativamente baja frente a una precisión muy alta y una tasa de error baja. Lo que quiere decir que la red tiene cierta dificultad en la detección de la clase debido a la poca cantidad de muestras

disponibles en los datos de entrenamiento

- La tasa de error de las clases 3 y 4 es superior al resto de clases debido a la confusión entre ellas dado que se parecen mucho en cuanto a su forma y su posición respecto al centro según la orientación de la plancha en cada imagen.

- La clase 5 correspondiente a la doble pared corta presenta un valor de precisión alto y una sensibilidad relativamente baja debido a la confusión con otras clases. Sobre todo, con las clases 6 y 8 correspondientes a la *doble pared larga* y el *tejadillo* respectivamente debido a sus formas similares.

- La tasa de error de la clase 10 correspondiente al Entorno es relativamente alta debido a algunas confusiones con los píxeles de las otras clases dado que esta clase está en contacto con la mayoría de las clases.

- Para mejorar el rendimiento de red neuronal habrá que incluir mayor cantidad de imágenes y procurar mantener un equilibrio entre todas las clases.

7.3. Líneas futuras.

La línea de investigación de este trabajo queda abierta al no haberse completado el objetivo final de implementar el sistema de visión en una máquina montadora de cajas. El trabajo futuro derivado de este proyecto abarca tareas asociadas a la mejora e implementación final del sistema sobre las máquinas entre las cuales se incluyen:

- Mejorar el entrenamiento realizado mediante la inclusión de más cantidad de imágenes de planchas manteniendo un equilibrio entre todas las clases.
- Incluir imágenes de más modelos de planchas para ampliar la capacidad del sistema para afrontar la mayoría de las aplicaciones reales.
- Mejorar el sistema de visión mediante la optimización de sus componentes.
- Realizar la segmentación semántica en diferentes tipologías de redes y comparar el resultado para asegurar el rendimiento óptimo.
- Desarrollar un sistema de visión y un algoritmo de inteligencia artificial para darle a las máquinas no solo la capacidad para reconocer las características, sino también para extraer información dimensional.
- Desarrollar un sistema de control completo para utilizar la información recibida de la segmentación semántica para controlar automáticamente los parámetros de la máquina y darle la capacidad de autoregulación.
- Darle al algoritmo la capacidad de aprendizaje continuo para incluir automáticamente imágenes nuevas en la base de datos y entrenarse para reconocer nuevas características y aumentar su robustez.
- Desarrollar una aplicación informática basada en el algoritmo para escanear las planchas y convertirlas en planos dimensionados.
- Desarrollar una aplicación informática basada en el algoritmo para convertir la información escaneada de las planchas en archivos 3D con la capacidad de plegar las cajas automáticamente conociendo la relación entre las diferentes regiones de la plancha durante su transformación en caja.
- Desarrollar un sistema de realidad aumentada basado en el algoritmo para mostrar el aspecto final de la caja montada a partir de la foto de la plancha sin plegar.

BIBLIOGRAFÍA

[1] ElPais, «Marvin Minsky, cerebro de la inteligencia artificial,» 26 01 2016. [En línea]. Available: https://elpais.com/elpais/2016/01/26/ciencia/1453809513_840043.html.

[2] D. Poole, Computational Intelligence: A Logical Approach, Oxford University Press, 1998.

[3] Ministerio de industria comercio y turismo, «La industria 4.0: El estado de la cuestión,» España, 2018.

[4] Spri, «Inteligencia artificial para apuntalar la Industria 4.0,» [En línea]. Available: https://www.spri.eus/es/basque-industry-comunicacion/inteligencia-artificial-para-apuntalar-la-industria-4-0/.

[5] manutencionyalmacenaje, «Datos y robots transforman el sector del envase y embalaje,» [En línea]. Available: https://www.manutencionyalmacenaje.com/articulos/240733-Datos-y-Robots-transforman-el-sector-del-Envase-y-Embalaje.

[6] E. Alegre, Conceptos y Métodos en Visión por Computador, CEA, 2016.

[7] Infaimon, «Sistemas de visión artificial: tipos y aplicaciones,» 2019. [En línea]. Available: https://blog.infaimon.com/sistemas-de-vision-artificial-tipos-aplicaciones/.

[8] Cognex, «Introduccuón a la visión artificial,» 2017. [En línea]. Available: https://www.cognex.com/es-es/what-is/machine-vision.

[9] National Instruments, «A practical guide to machine vision lighting,» 2019. [En línea]. Available: https://www.ni.com/es-es/innovations/white-papers/12/a-practical-guide-to-machine-vision-lighting.html.

[10] Vision-Doctor, «Lighting techniques for machine vision,» [En línea]. Available: https://www.vision-doctor.com/en/illumination-techniques.html.

[11] S. IMAGING, Imaging & Vision Handbook 2016/17, 2017.

[12] quecamarareflex, «Qué es la distancia focal en fotografía,» [En línea]. Available: https://quecamarareflex.com/distancia-focal-para-torpes/.

[13] drescuela, «Todo sobre la distancia focal en fotografía,» [En línea]. Available: https://drescuela.com/distancia-focal/.

[14] visionartificial, «Aplicación práctica de la visión artificial,» [En línea]. Available: http://visionartificial.fpcat.cat/wp-content/uploads/UD_1_didac_Conceptos_previos.

[15] nikon, « lente nikon af s 55 300mm,» [En línea]. Available: https://www.nikon.es.

[16] diligent, «La importancia de la profundidad de campo en la fotografia de producto,» [En línea]. Available: https://www.diligent.es/la-importancia-de-la-profundidad-de-campo-en-la-fotografia-de-producto/.

[17] .thewebfoto, «Exposición,» [En línea]. Available: http://www.thewebfoto.com/2-hacer-fotos/207-exposicion.

[18] infaimon, «SOFTWARE DE IMAGEN,» [En línea]. Available: https://www.infaimon.com/categoria-producto/software-de-imagen/.

[19] L. T. Bruno, «Introducción a la inteligencia artificial,» *Instituto Tecnológico de Nuevo Laredo,* 2007.

[20] unipython, «urso de inteligencia artificial,» [En línea]. Available: https://unipython.com/curso-de-inteligencia-artificial/.

[21] D. G. Seisdedos, Segmentación de núcleos celulares en imágenes de microscopia ayudados por redes neuronales convolucionales, Universitat Oberta de Catalunya, 2018.

[22] Lareserva, «¿Cuáles son las partes de una neurona?,» [En línea]. Available: https://www.lareserva.com/Cuales_son_las_partes_de_una_neurona.

[23] diegocalvo, «Definición de red neuronal artificial,» [En línea]. Available: http://www.diegocalvo.es/definicion-de-red-neuronal/.

[24] M. W, «A logical calculus of ideas immanent in,» *Bulletin of Mathematical Biophysics,* 1943.

[25] H. D, «The organization of behaviour,» *McGill University,* 1949.

[26] T. AM, «Inteligent machinery, a heretical theory,» *Elsevier Science Publishers,* 1992.

[27] W. PJ, «Beyound regression: new tools for prediction and analysis in the behavioural science,» *Harvard University,* 1975.

[28] BBVAData, «Una Introducción "peculiar" al Deep Learning,» [En línea]. Available: https://www.bbvadata.com/es/a-weird-introduction-to-deep-learning/.

[29] L. Schelkes, «Evaluation of Neural Networks for Image Recognition Applications,» *arxiv,* 2018.

[30] H. DH, «Receptive fields of single neurones in the cat's striate cortex,» *Physiol,* 1959.

[31] G. J, «Recent advances in convolutional neural networks,» *ElSevier,* 2017.

[32] L. YA, «Efficient backpropagation,» *Tricks of the trade,* 2012.

[33] N. V, «Rectified linear units improve restricted Boltzmann machines,» *Proceedings of the international conference on machine learning,* 2010.

[34] W. T, «End to end text recognition with convolutional neural networks,» *Proceedings of the international conference on pattern recognition,* 2012.

[35] B. Y, «A teoretical analysis of feature pooling in visual recognition,» *Proceedings of the international conference on machine learning,* 2010.

[36] L. Y, «Gradient-based learning applied to document recognition,» *Proceedings of the IEEE,* 1998.

[37] S. K, «Very deep convolutional networks for large-scale image recognition,» *Proceedings of the international conference on learning representations,* 2015.

[38] R. O, «Imagenet large scale visual recognition challenge,» *International journal of conflict and violence,* 2015.

[39] H. GE, «Dimensionality of data with neural networks,» *Toronto University,* 2006.

[40] R. MA, «Unsupervised learning of invariant feature hierarchies with appplications to object recognition,» *New York University,* 2006.

[41] C. K, «High performance convolutional neural networks for document processing,» *Inventeurs du monde numérique,* 2006.

[42] N. F, «Face detection using GPU-based convolutional neural networks,» *International conference on computer analysis of image and patterns,* 2009.

[43] T. SC, «Convolutional networks can learn to generate affinity graphs for image segmentation,» *Neural Computation,* 2010.

[44] K. A, «ImageNet classification with deep convolutional neural networks,» *University of Toronto,* 2012.

[45] L. Y. Yang, «Inductive Biases,» [En línea]. Available: https://leowyy.github.io/.

[46] H. GE, «Improving neural networks by preventing co-adaptation of feature detectors,» *University of Toronto,* 2012.

[47] Mc, «Review of ZFNet —The Winner of ILSVLC 2013,» [En línea]. Available: https://mc.ai/paper-review-of-zfnet-the-winner-of-ilsvlc-2013-image-classification/.

[48] S. K, «Very deep convolutional networks for large-scale image recognition,» *Computer vision pattern recognition,* 2014.

[49] inovex, «Robustifying Machine Perception for Image Recognition Systems,» [En línea]. Available: https://www.inovex.de/blog/machine-perception-face-recognition/.

[50] Mc, «Loss functions based on feature activation and style loss,» [En línea]. Available: https://mc.ai/loss-functions-based-on-feature-activation-and-style-loss/.

[51] S. C, «Going deeper with convolutions,» *Computer vision foundation,* 2015.

[52] K. He, «Deep residual learning for image recognition,» *Cornell University,* 2015.

[53] X. Tang, «Categories of NoSQL,» [En línea]. Available: https://xinxintang.github.io/page/6/.

[54] W. Rawat, «Deep convolutional neural networks for image classification: A comprehensive review,» *Neural computation,* 2017.

[55] mathworks, «Get Started with Transfer Learning,» [En línea]. Available: https://es.mathworks.com/help/deeplearning/gs/get-started-with-transfer-learning.html.

[56] Universitat Oberta de Catalunya, «La segmentación semántica y sus benchmarks,» [En línea]. Available: http://informatica.blogs.uoc.edu/2016/05/26/la-segmentacion-semantica-y-sus-benchmarks/.

[57] G. A, «A review on deep learning techniques applied to semantic segmentation,» *Arxiv,* 2017.

[58] avansis, «APLICACIONES DE LA IA: VISIÓN COMPUTACIONAL,» [En línea]. Available: https://www.avansis.es/2019/06/20/aplicaciones-de-la-ia-vision-computacional/.

[59] L. J, «Fully convolutional networks for semantic segmentation,» *Proceedings of the IEEE Conference on computer vision pattern recognition,* 2015.

[60] B. V, «Segnet: a deep convolutional encoder-decoder architecture for image recognition,» *IEEExplore,* 2015.

[61] Boix, «Caja Columna Cerrada,» [En línea]. Available: https://www.boix.es/productos/maquinas/aplicacion/caja-columna-cerrada/.

[62] Infaimon, «ILUMINACIÓN FRONTAL,» [En línea]. Available: https://www.infaimon.com/enciclopedia-de-la-vision/iluminacion-frontal/.

[63] P. Khlebovich, «IP Webcam,» [En línea]. Available: https://play.google.com/store/apps/details?id=com.pas.webcam&hl=es.

[64] Samsung, «Galaxy S6 edge+ (Plus)-¿Para qué sirve cada uno de los modos de la cámara?,» [En línea]. Available: https://www.samsung.com.

[65] Wikipedia, «Joint Photographic Experts Group (JPEG);,» [En línea]. Available: https://es.wikipedia.org/wiki/Joint_Photographic_Experts_Group.

[66] Wikipedia, «Portable Network Graphics (PNG),» [En línea]. Available: https://es.wikipedia.org/wiki/Portable_Network_Graphics.

[67] Mathworks, «Color Thresholder,» [En línea]. Available: https://es.mathworks.com/help/images/ref/colorthresholder-app.html.

[68] Wikipedia, «RGB,» [En línea]. Available: https://es.wikipedia.org/wiki/RGB.

[69] Ecured, «Modelo HSV,» [En línea]. Available: https://www.ecured.cu/Modelo_HSV.

[70] Hisour, «Espacios de color YCbCr,» [En línea]. Available: https://www.hisour.com/es/ycbcr-color-spaces-26075/.

[71] Gugsm, «Espacios de color L*a*b,» [En línea]. Available: http://www.gusgsm.com/el_espacio_de_color_lab.

[72] L. Y, «Gradient-based learning applied to document recognition,» *Proceedings of the IEEE,* 1998.

[73] https://deepai.org/machine-learning-glossary-and-terms/softmax-layer

[74] Mathlab «Training options»

https://es.mathworks.com/help/deeplearning/ref/trainingoptions.html#bu80qkw-3_head

[75] Nvidia «GPU Compute Capability» https://developer.nvidia.com/cuda-gpus

[76] Pyimagesearch «Intersection over Union (IoU) for object detection» https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

[77] Juanbarrios «Balanceo de clases en casos de salud» https://www.juanbarrios.com/balanceo-de-clases-en-casos-de-salud/

[78] Carlos Zelada «Evaluación de modelos de clasificación» https://rpubs.com/chzelada/275494

[79] Portilla Jose «Data Science and Machine Learning Bootcamp with R». https://www.udemy.com/data-science-and-machine-learning-bootcamp-with-r/.

ANEXOS

Listado de códigos.

- ### Algoritmo de preprocesamiento.

```
clc
clear all
close all
%% Cargar la imágen
% ImOriginal=imread('F:\FORMACIÓN\MASTER ROBÓTICA\TFM\DATA\RAW\PNG\0001.png');
% imshow(ImOriginal);
% mostrar el histograma
% imhist(ImOriginal);

%% Recortar la imagen al marco que contiene la plancha

RutaOrigen = ('F:\FORMACIÓN\MASTER ROBÓTICA\TFM\DATA\ORIGINAL\PNG'); % ruta de donde importamos las imágenes
RutaDestino = ('F:\FORMACIÓN\MASTER ROBÓTICA\TFM\DATA\PREPROCESADO'); % ruta donde se guardan las imágenes preprocesadas

RutaOrigenPrep= [RutaOrigen '\']; % añadir "\" a la ruta de origen
Elementos= ls(RutaOrigenPrep); % obtener la lista con los nombres de los elementos de la ruta de origen ordenados
Tabla= cellstr(Elementos); % insertar en una tabla los elementos de la lista
Tabla2= Tabla(3:length(Tabla)); % ajustar la tabla
S= size(Tabla2); % calcular la dimensión de la tabla
a= 1;

while a <= S(1) % Ejecutamos un bucle para realizar las operaciones de preprocesamiento sobre cada una de las imágenes de la carpeta de origen
    close all
    NElemento= char(Tabla2(a)); %nombre con extensión del elemento numero (a) seleccionado
    NImagen = (NElemento); %Nombre de la imágen
    LinkImagen = [RutaOrigen '\' NImagen]; % Construir el link al elemento seleccionado
    ImOriginal = imread(LinkImagen); %leer la imágen que corresponde al elemento seleccionado
    [Altura,Ancho,z1] = size(ImOriginal); % obtener las dimensiones de la imágen


    [BW,ImSinFondo]=EliminarFondo(ImOriginal); %crear máscara y eliminar el fondo de la plancha
 imshow(ImSinFondo);
    ImagenBinaria = im2bw(ImSinFondo,0.1); %ajustar y Generar la imagen binaria
%     imshow(ImRell);

    [imagenBlob,pixel] = ExtraerPlancha(ImagenBinaria, 1); % recortar la imagen para visualizar solo la plancha , 1 es el numero de blobs a extraer
    ImRecortada = imcrop(ImSinFondo,pixel.BoundingBox); %usar el cuadro delimitador para recortar la imagen en color
  [Altura2,Ancho2,z2] = size(ImRecortada); % obtener las dimensiones de la imágen recortada

  % rellenar con negro hasta conseguir la relación de aspecto 16:9
  AnchoObjetivo=1920;
  AltoObjetivo=1080;
```

```
43.     AnchoEq=(Altura2*AnchoObjetivo)/AltoObjetivo; % el ancho equivalente % regla
    de tres para una relacion 16:9
44.     Marg=round((AnchoEq-Ancho2)/2); % el margen que hay que rellenar a cada lado
    de la imagen redondeado a un numero entero double
45.     ImRell = padarray(ImRecortada,[0 Marg],'both'); %rellenar en un eje de la
    imagen
46.     % Redimensionar para ahorrar memoria y acelerar el cálculo
47.     ImRed = imresize(ImRell, [360 640]); % redimensionar hasta un tercio de la
    dimension 1920x1080
48.     imwrite(ImRed,[RutaDestino '\' 'R' NImagen ],'png','Comment','R') % guardar
    las imagenes en la carpeta de destino
49.         a= a+1;
50. end
```

- **Función para extraer la plancha.**

```
1.  function [SoloPlancha,Coordinadas] = ExtraerPlancha(ImagenBinaria,
    CantidadEtiqExtraer)
2.
3.      % Etiquetar y Obtener las propiedades de las zonas etiquetadas
4.      [ImEtiquetada, numeroEtiq] = bwlabel(ImagenBinaria);
5.      % obtener las areas de cada etiqueta
6.      AreaEtiq = regionprops(ImEtiquetada, 'area');
7.      TAreas = [AreaEtiq.Area];
8.
9.      % ordenar las areas de mayor a menor
10.     [OrdenArea, IndiceArea] = sort(TAreas, 'descend');
11.
12.     % extraer la etiqueta más grande
13.     MayorEtiq = ismember(ImEtiquetada, IndiceArea(1:CantidadEtiqExtraer));
14.     % convertir de integer a binary (logical).
15.     ImagenBinaria = MayorEtiq > 0;
16.
17.     %Recortar la imagen
18.      ImEtiquetada2 = bwlabel(ImagenBinaria);
19.      Coordinadas = regionprops(ImEtiquetada2, 'BoundingBox');
20.      % encontrar el cuadro delimitador obteniendo la lista de pixeles en el blob.
21.      Cdelimitador = Coordinadas.BoundingBox;
22.      % recortar la imagen al cuadro delimitador
23.      SoloPlancha = imcrop(ImagenBinaria, Cdelimitador);
24.
25. end
26.
27. imshow(ImEtiquetada2);
28.
```

- **Función para eliminar el fondo.**

```
1.  function [BW,ImMascara] = EliminarFondo(ImRGB)
2.  %createMask  Threshold RGB image using auto-generated code from colorThresholder
    app.
3.  %  [BW,MASKEDRGBIMAGE] = createMask(RGB) thresholds image RGB using
4.  %  auto-generated code from the colorThresholder app. The colorspace and
5.  %  range for each channel of the colorspace were set within the app. The
6.  %  segmentation mask is returned in BW, and a composite of the mask and
7.  %  original RGB images is returned in maskedRGBImage.
8.
9.  % Auto-generated by colorThresholder app on 01-May-2019
10. %------------------------------------------------------
11.
12.
13. % Convertir la imágen del espacio de colore RGB al espacio de color L*a*b
14. I = rgb2lab(ImRGB);
15.
16. % Definir la umbralizacion sobre el canal 1 basando en el histograma
17. Canal1Min = 0.000;
18. Canal1Max = 100.000;
19.
```

```
% Definir la umbralizacion sobre el canal 2 basando en el histograma
Canal2Min = -10.405;
Canal2Max = 16.208;

% Definir la umbralizacion sobre el canal 3 basando en el histograma
Canal3Min = -2.037;
Canal3Max = 34.813;

% Crear una máscara basada en la umbralización ajsutada sobre el histograma
MascaraBW = (I(:,:,1) >= Canal1Min ) & (I(:,:,1) <= Canal1Max) & ...
    (I(:,:,2) >= Canal2Min ) & (I(:,:,2) <= Canal2Max) & ...
    (I(:,:,3) >= Canal3Min ) & (I(:,:,3) <= Canal3Max);
BW = MascaraBW;

% Inicializar la imagen de salida con máscara basada en la imagen de entrada.
ImMascara = ImRGB;

% Set background pixels where BW is false to zero.
% reemplazar por cero el valor de los pixeles del fondo donde la imagen binaria
BW es False
ImMascara(repmat(~BW,[1 1 3])) = 0;

end
```

- **Algoritmo de segmentación.**

```
% DESCARGAR E INSTALAR EL MODULO DE NEURAL NETWORK TOOLBOX PARA LA VGG16.
% clc
% clear all;
% vgg16
%% PREPARACIÓN

DirImagR = fullfile(pwd,'DATA','PREPROCESADO'); % directorio imagenes
preprocesadas
DirEtiqR= fullfile(pwd,'DATA','ETIQUETADO'); % directorio imágenes etiquetadas


%% CARGAR IMAGENES

ImDataStore = imageDatastore(DirImagR);
% mostrar una imagen
Im_N=4;
I_ra=readimage(ImDataStore,Im_N);
figure
imshow(I_ra)
drawnow
title('imagen número 100')


%% CARGAR IMAGENES ETIQUETADAS

classes = [
    "Fondo"
    "Orejita"
    "Pared_Corta"
    "Pared_Larga"
    "Doble_Pared_PC"
    "Doble_Pared_PL"
    "Solapa"
    "Tejadillo"
    "Columna"
    "Entorno"
    ];

% mapa de colores
cmap=[
    0 255 255     % Fondo
    255 128 0     % Orejita
    255 0 255     % Pared_Corta
    0 255 0       % Pared_Larga
    255 0 128     % Doble_Pared_PC
```

```
    128 0 255     % Doble_Pared_PL
    255 0 0       % Solapa
    0 0 255       % Tejadillo
    255 255 0     % Columna
    255 255 255   % Entorno
    ];
% normalizar
cmap=cmap ./255;

labelIDs= {...
    [
    0 255 255; ... % "Fondo"
    ]
    [
    255 128 0; ... % "Orejita"
    ]
    [
    255 0 255; ... % "Pared_Corta"
    ]
    [
    0 255 0; ... % "Pared_Larga"
    ]
    [
    255 0 128; ... % "Doble_Pared_PC"
    ]
    [
    128 0 255; ... % "Doble_Pared_PL"
    ]
    [
    255 0 0; ... % "Solapa"
    ]
    [
    0 0 255; ... % "Tejadillo"
    ]
    [
    255 255 0; ... % "Columna"
    ]
    [
    255 255 255; ...% "Entorno"
    ]
    };

LbDataStore = pixelLabelDatastore(DirEtiqR,classes,labelIDs);

%% Mostrar mapeado
 C_rgb = imread(LbDataStore.Files{100});
 C = readimage(LbDataStore, 100);

            if size(C_rgb,1) == 720
                B =
labeloverlay(C_rgb(600:650,600:650,:),C(600:650,600:650,:),'ColorMap',cmap);
                % Show a ground-truth array
                C(600:650,600:650)
            else
                B =
labeloverlay(C_rgb(200:225,200:225,:),C(200:225,200:225,:),'ColorMap',cmap);
                C(200:225,200:225)
              end
              reset(LbDataStore);
              imshow(B,'InitialMagnification',400)

 %% leer y mostrar etiquetas sobre la imagen
 drawnow

 Cc = readimage(LbDataStore, Im_N);
 % Overlay segmentation results onto original image. - Computer Vision System
Toolbox
 Bb = labeloverlay(I_ra,Cc,'ColorMap',cmap);
 figure
 imshow(Bb)
 drawnow

  % Add a colorbar to the current axis. The colorbar is formatted
             % to display the class names with the color.

```

```
117.              colormap(gca,cmap)
118.
119.              % añadir una barra de colores a la imagen actual
120.              c = colorbar('peer', gca);
121.
122.              % usar los nombres de las clases para cada color.
123.              c.TickLabels = classes;
124.              numClasses = size(cmap,1);
125.
126.              % centrar.
127.              c.Ticks = 1/(numClasses*2):1/numClasses:1;
128.
129.              % eliminar marcas.
130.              c.TickLength = 0;
131.
132.
133.   %% ESTADISTICAS
134.
135.   tbl = countEachLabel(LbDataStore)
136.
137.   % calculamos la frecuencia para conseguir una histograma de los datos
138.  frequency = tbl.PixelCount/sum(tbl.PixelCount);
139.  figure
140.  bar(1:numel(classes),frequency)
141.  xticks(1:numel(classes))
142.  xticklabels(tbl.Name)
143.  xtickangle(45)
144.  ylabel('Frequency')
145.
146.
147.  %% PREPARACION DATOS DE ENTRENAMIENTO Y test
148.
149.  % dividir los datos aleatoriamente seleccionando el 80% para el entrenamiento,
150.  % el resto se usara para el test.
151.
152.              % establecer el estado aleatorio inicial.
153.              rng(0);
154.              numFiles = numel(ImDataStore.Files);
155.              % Devuelve un vector de fila que contiene una permutación aleatoria
    de numeros enteros desde 1 hasta n incluido.
156.              shuffledIndices = randperm(numFiles);
157.
158.              % utilizar el   80% para el entrenamiento.
159.              N = round(0.8 * numFiles);
160.              trainingIdx = shuffledIndices(1:N);
161.
162.              % utilizar el resto de imagenes para test.
163.              testIdx = shuffledIndices(N+1:end);
164.
165.              % Crear image datastores para el entrenamiento y test.
166.              trainingImages = ImDataStore.Files(trainingIdx);
167.              testImages = ImDataStore.Files(testIdx);
168.              imdsTrain = imageDatastore(trainingImages);
169.              imdsTest = imageDatastore(testImages);
170.
171.
172.              % extraer la info de las clases y label IDs.
173.              classes = LbDataStore.ClassNames;
174.              %labelIDs = 1:numel(LbDataStore.ClassNames);
175.
176.              % Crear pixel label datastores para el entrenamiento y test.
177.              trainingLabels = LbDataStore.Files(trainingIdx);
178.              testLabels = LbDataStore.Files(testIdx);
179.              pxdsTrain = pixelLabelDatastore(trainingLabels, classes, labelIDs);
180.              pxdsTest = pixelLabelDatastore(testLabels, classes, labelIDs);
181.
182.
183.              DataSourceTest=pixelLabelImageSource(imdsTest,pxdsTest);
184.
185.              % numero total de imagenes de entrenamiento
186.              numTrainingImages = numel(imdsTrain.Files)
187.              % numero total de imagenes de test
188.              numTestingImages = numel(imdsTest.Files)
189.
190.   %% Crear la red neuronal
```

```
imageSize = [360 640 3];
numClasses = numel(classes);

% segnetLayers devuelve las capas de la red SegNet, lgraph,que está ya
% inicializado con los pesos y capas del modelo preintrenado.
lgraph = segnetLayers(imageSize,numClasses,'vgg16');
% lgraph inicialmente tiene 91 capas.
lgraph.Layers

% Plot the 91-Layer lgraph
fig1=figure('Position', [100, 100, 1000, 1100]);
subplot(1,2,1)
plot(lgraph);
axis off
axis tight
title('Complete Layer Graph')
subplot(1,2,2)
plot(lgraph);
xlim([2.862 3.200])
ylim([-0.9 10.9])
axis off
title('Last 9 Layers Graph')

%% equilibrar clases usando sus pesos

% Get the imageFreq using the data from the countEachLabel function
imageFreq = tbl.PixelCount ./ tbl.ImagePixelCount;
% The higher the frequency of a class the smaller the classWeight
classWeights = median(imageFreq) ./ imageFreq

pxLayer = pixelClassificationLayer('Name','labels','ClassName', tbl.Name,
'ClassWeights', classWeights)

% actualizar la red

% Plot the 91-Layer lgraph
fig2=figure('Position', [100, 100, 800, 600]);
subplot(1,2,1)
plot(lgraph);
xlim([2.862 3.200])
ylim([-0.9 10.9])
axis off
title('últimas 9 capas antes de modofocar')
% Remove last layer of and add the new one we created.
lgraph = removeLayers(lgraph, {'pixelLabels'});
lgraph = addLayers(lgraph, pxLayer);
% Connect the newly created layer with the graph.
lgraph = connectLayers(lgraph, 'softmax','labels');
lgraph.Layers

subplot(1,2,2)
plot(lgraph);
xlim([2.862 3.200])
ylim([-0.9 10.9])
axis off
title(' últimas 9 capas modificadas')

%% Ajustar opciones de entrenamiento
options = trainingOptions('sgdm', ...   % algoritmo de optimización sgdm:
stochastic gradient descent with momentum
      'LearnRateSchedule','none',...      % metodo para disminuir la tasa de
aprendizaje automaticamente
      'Momentum', 0.9, ...                % contribución de una iteración
anterior en la iteración actual.
      'LearnRateDropFactor',0.1,...       % factor por el cual se multiplica la
tasa de aprendizaje para disminuir.
      'LearnRateDropPeriod',2,...         % número de epocas para disminuir la
tasa de aprendizaje
      'InitialLearnRate', 5e-2, ...       % si la tasa de aprendizaje es
demasiado pequeña dura más el entrenamiento, si es muy grande no converge (
resultados suboptimos).
      'L2Regularization', 0.0001, ...     % decaer los pesos para evitar el
sobreajuste (overfitting)
      'MaxEpochs', 250,... %120           % n° máximo de epocas. una epoca es el
paso del algoritmo de optimización por todos los datos de entrenamiento.
```

```
256.      'MiniBatchSize', 1, ... %4          % conjunto de datos de entrenamiento
    usado para evaluar el gradiente de la función de pérdida y actualizar los pesos.
257.      'Shuffle', 'once', ...              % mezclar los datos de entrenamiento
    antes de cada epoca y mezclar los datos de validación antes de cada validación de
    la red.
258.      'ExecutionEnvironment','gpu', ...   % recurso usado en el entrenamiento :
    auto, cpu, gpu, parallel
259.      'Plots','training-progress',...     % gráficos a mostrar durante el
    entrenamiento
260.      'ValidationData',DataSourceTest, ...% datos usados para la validación .  %
    LEER ESTE ENLACE IMPORTANTE :  https://towardsdatascience.com/train-validation-
    and-test-sets-72cb40cba9e7
261.      'ValidationPatience',5, ...         % las veces en que la pérdida puede ser
    mayor o igual que la pérdida más pequeña antes de parar el entrenamiento (
    validación )
262.      'Verbose', true,...                 % activar/desactivar la tabla de
    progreso mostrada en la ventana de comandos
263.      'VerboseFrequency',20);             % frecuencia de actualización de
    valores de la tabla.
264.
265.      %'GradientThresholdMethod','l2norm',... % método para recortar los valores
    del gradiente que superan cierto umbral.
266.      %'GradientThreshold',0.05); % si el gradiente supera este umbral, se corta
    acorde al metodo usado
267.     %%
268.
269.  %% Data Aumentation
270.
271.  augmenter =
    imageDataAugmenter('RandXReflection',true,'RandYReflection',true,...
272.      'RandXTranslation', [-10 10], 'RandYTranslation',[-10 10],'RandRotation',[-
    10,10],'RandXShear',[-5,5],'RandYShear',[-5,5]);
273.
274.  % mostrar una rotacion
275.
276.  I = readimage(ImDataStore, Im_N);
277.              Ib = readimage(LbDataStore, Im_N);
278.
279.
280.
281.              IIB = labeloverlay(I, Ib, 'Colormap', cmap, 'Transparency',0.8);
282.
283.              % Convertir de categorico a uint8.
284.              L = uint8(Ib);
285.
286.              Ir = imrotate(I,10,'nearest','crop');
287.              Lr = imrotate(L,10,'nearest','crop');
288.
289.              valueset = 1:10;
290.
291.              LB = categorical(Lr, valueset,LbDataStore.ClassNames);
292.              IB = labeloverlay(Ir, LB, 'Colormap', cmap, 'Transparency',0.8);
293.
294.              figure
295.              imshowpair(IIB,IB,'montage');
296.  %             HelperFunctions.pixelLabelColorbar(cmap, LbDataStore.ClassNames);
297.              title('Original vs Rotated image')
298.
299.  %% Comenzar el entrenamiento
300.
301.  datasource = pixelLabelImageSource(imdsTrain,pxdsTrain,...
302.      'DataAugmentation',augmenter);
303.
304.  tic
305.  [net, info] = trainNetwork(datasource,lgraph,options);
306.  toc
307.
308.  save('PreTrainedCnn.mat','net','info','options');
309.  disp('NN trained');
310.
311.  %% visualizar imagenes
312.  Im_N=30;
313.  I=readimage(ImDataStore,Im_N);
314.  Ib = readimage(LbDataStore,Im_N);
315.  IB = labeloverlay(I, Ib, 'Colormap', cmap, 'Transparency',0.5);
```

```
316.
317.  % Mostrar el resultado de la segmentación semántica
318.  C = semanticseg(I, net);
319.  CB = labeloverlay(I, C, 'Colormap', cmap, 'Transparency',0.5);
320.  figure
321.  imshowpair(IB,CB,'montage')
322.  title('imagen original vs segmentada')
323.
324.   % Add a colorbar to the current axis. The colorbar is formatted
325.              % to display the class names with the color.
326.
327.              colormap(gca,cmap)
328.
329.              % añadir una barra de colores a la imagen actual.
330.              c = colorbar('peer', gca);
331.
332.              % usar los nombres de las clases para cada color.
333.              c.TickLabels = classes;
334.              numClasses = size(cmap,1);
335.
336.              % centrar.
337.              c.Ticks = 1/(numClasses*2):1/numClasses:1;
338.
339.              % Eliminar marcas.
340.              c.TickLength = 0;
341.
342.              %% Probar la Red
343.
344.  ITEST = read(imdsTest);
345.  tic
346.  C = semanticseg(ITEST, net);
347.  toc
348.
349.  B = labeloverlay(ITEST, C, 'Colormap', cmap, 'Transparency',0.5);
350.  figure
351.  imshow(B)
352.  title('segmentación semantica')
353.
354.  % Añadir una barra de colores que serve como leyenda para identificar las
355.   % etiquetas.
356.
357.              colormap(gca,cmap)
358.
359.              % Añadir la barra de colores a la imagen.
360.              c = colorbar('peer', gca);
361.
362.              % Usar nombres de clases.
363.              c.TickLabels = classes;
364.              numClasses = size(cmap,1);
365.
366.              % Centrar.
367.              c.Ticks = 1/(numClasses*2):1/numClasses:1;
368.
369.
370.              c.TickLength = 0;
371.
372.  %% Comparar las imagenes
373.  ResultadoEsp = read(pxdsTest); % cargar una imágen con las etiquetas manuales.
374.  ITEST2 = read(imdsTest); % cargar una imagen nueva
375.  C2 = semanticseg(ITEST2, net); % aplicar la segmentación semántica
376.  esperado = uint8(ResultadoEsp); % mostrar en escala de grises
377.  previsto = uint8(C2);
378.  % Image Processing toolbox
379.  imshowpair(esperado, previsto, 'montage')
380.  title('etiquetas manuales vs etiquetas segmentación semantica')
381.  % reset(pxdsTest); %resetear el contador cuando llega al final
382.  % reset(imdsTest);
383.
384.  %% Índice de Jaccard
385.  iou = jaccard(C2, ResultadoEsp);
386.  table(classes,iou) % mostrar el grado de coincidencia por cada clase.
387.
388.  %% Índice de Dice
389.  IndiceDice = dice(C2,ResultadoEsp);
390.  table(classes,IndiceDice)
```

```
391.
392.  %% Índice de Boundary-F1
393.  IndiceBF= bfscore(C2,ResultadoEsp);
394.  table(classes,bfscore)
395.  %% Evaluación de la red entrenada
396.  reset(pxdsTest); %resetear el contador cuando llega al final
397.  reset(imdsTest);
398.  % realizar la segmentación semántica
399.  pxdsResultados =
    semanticseg(imdsTest,net,'WriteLocation',tempdir,'Verbose',false,'ExecutionEnviro
    nment','cpu');
400.
401.  % evaluar los resultados de la segmentación semántica frente a ground truth
402.  metrics =
    evaluateSemanticSegmentation(pxdsResultados,pxdsTest,'Metrics',"all");
403.
404.  metrics.DataSetMetrics % métricas a nivel de conjunto de datos.
405.  metrics.ClassMetrics % métricas a nivel de clases.
406.  %%
407.  normConfMatData = metrics.NormalizedConfusionMatrix.Variables;
408.  figure
409.  h = heatmap(classes,classes,100*normConfMatData);
410.  h.XLabel = 'Predicted Class';
411.  h.YLabel = 'True Class';
412.  h.Title = 'Normalized Confusion Matrix (%)';
```