eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Computación

Trabajo de Fin de Grado

Caracterización de superficies texturizadas mediante procesamiento de imagen y el uso de Inteligencia Artificial

Autor/a

Xuban Barberena Apezetxea

2023

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Computación

Trabajo de Fin de Grado

Caracterización de superficies texturizadas mediante procesamiento de imagen y el uso de Inteligencia Artificial

Autor/a

Xuban Barberena Apezetxea

Directore/a(s)

Jesús M. Pérez de la Fuente (UPV/EHU)

Fátima Saiz Álvaro (VICOMTECH)

Resumen

En la última década, ha habido un auge en el uso de técnicas de visión artificial para resolver las necesidades que surgen en aplicaciones industriales, como el proceso de control de calidad. El control de calidad es un aspecto fundamental de cualquier industria manufacturera, que tradicionalmente se ha realizado manualmente. Sin embargo, para evitar posibles problemas y carencias derivadas de esta inspección, este proceso se ha tendido a automatizar. El desarrollo del *Deep Learning* en los últimos años ha permitido alcanzar grandes resultados en procesamiento de imágenes, sustituyendo de esta forma los métodos tradicionales de control de calidad y convirtiéndose en la herramienta más utilizada.

Los modelos de *Deep Learning* requieren de una gran cantidad de datos para garantizar que el rendimiento sea bueno. Sin embargo, a menudo el sector de la industria se enfrenta al inconveniente de la poca disponibilidad de los datos o de datos desbalanceados, lo que conlleva a tener unos resultados de menor calidad. Esto es un problema, puesto que en el control de calidad es absolutamente imprescindible que los resultados sean precisos para evitar productos finales defectuosos o falso rechazo.

A raíz del obstáculo de la disponibilidad de pocos datos, han ido surgiendo nuevas técnicas para solventar este problema y así mejorar la capacidad de generalización de los modelos de aprendizaje automático en entornos de pocos datos. Este es el objetivo del paradigma *Few-shot learning*, con el cual se ha trabajado en este Trabajo de Fin de Grado.

La finalidad de este Trabajo de Fin de Grado es realizar un estudio sobre técnicas avanzadas de Inteligencia Artificial aplicada a imágenes en escenarios industriales con escasez y/o desbalanceo de datos entre clases y comprobar si una técnica de *Few-shot learning* puede sustituir a las arquitecturas tradicionales de *Deep Learning* más utilizadas en la literatura para la tarea de control de calidad, en concreto, la detección de defectos superficiales texturizadas a nivel microscópico.

Para ello, se ha llevado a cabo el proceso que ejecutan los sistemas de inspección automá-

tica en la industria para la detección de defectos: adquisición de los datos, generación de la base de datos, diseño y entrenamiento de las redes neuronales, validación y despliegue.

Se han realizado diferentes experimentaciones que demuestran que la técnica de *Few-Shot Learning* ofrece mejor rendimiento en entornos con pocos datos que las arquitecturas de segmentación basadas en *Deep Learning* más utilizadas en la literatura. En concreto, con muy pocas imágenes de entrenamiento, la métrica de validación (IoU) se acerca al 98 %.

Palabras clave: *Control de Calidad, Deep Learning, Machine Vision, Few-Shot Learning, Segmentación de defectos*

Abstract

In the last decade, there was a rise in the use of machine vision techniques to solve the needs that arise in industrial applications, such as the quality control process. Quality control is a fundamental process in any manufacturing industry, which was traditionally performed manually. However, to avoid potential problems and shortcomings derived from this inspection, this process has tended to be automated. The development of the *Deep Learning* in recent years has allowed to achieve great results in image processing, thus replacing the traditional methods of quality control and becoming the most widely used method.

Deep Learning models require a large amount of data to ensure good performance. However, the manufacturing sector often faces the drawback of poor data availability or unbalanced data, which leads to lower quality results. This is a challenge, as accurate results are absolutely essential in quality control to avoid defective final products or false rejects.

As a result of the obstacle of limited data availability, new techniques have emerged to overcome this problem and thus improve the generalizability of machine learning models in data-poor environments. This is the objective of the *Few-shot learning* paradigm, which was used in this Bachelor's Thesis.

The purpose of this Bachelor's Thesis is to carry out a study on advanced Artificial Intelligence techniques applied to images in industrial settings with scarcity and/or imbalance of data between classes and test whether a *Few-shot learning* technique can replace the traditional *Deep Learning* architectures most commonly used in the literature for the task of quality control, specifically, the detection of textured surface defects at the microscopic level.

For this purpose, the process executed by automatic inspection systems in industry for

defect detection was carried out: data acquisition, database generation, neural network design and training, validation and deployment.

Different experiments show that the *Few-Shot Learning* technique offers better performance in low-data environments than the segmentation architectures based on *Deep Learning* that are more used in the literature. Specifically, with very few training images, the validation metric (IoU) is close to 98 %.

Keywords: *Quality Control, Deep Learning, Machine Vision, Few-Shot Learning, Defect Segmentation*

Índice general

Índice de figuras

Índice de tablas

1. CAPÍTULO

Introducción

Durante este Trabajo Fin de Grado (TFG) se ha realizado un estudio sobre técnicas avanzadas de **Inteligencia Artificial** (IA) aplicada a imágenes en escenarios donde hay escasez y/o desbalanceo de datos entre clases. En concreto, se ha trabajado con redes de segmentación del estado del arte y redes basadas en ***Few Shot Learning***. Todo ello se ha hecho bajo un paradigma de **control de calidad industrial**, donde la carencia de datos es uno de los mayores retos a superar.

En los siguientes apartados se expondrá la problemática a tratar y el contexto en el que se enmarca el trabajo y su correspondiente planteamiento. Por último, se explicará la estructura de la presente memoria.

Cabe destacar que este TFG ha sido realizado en colaboración con el Centro Tecnológico Vicomtech, situado en Donostia-San Sebastián. En concreto, con en el departamento de Industria y Fabricación Avanzada, que está orientado a la investigación aplicada en las líneas de visión artificial, realidad virtual, realidad aumentada, robótica y fabricación inteligente.

1.1. Contexto

Las industrias manufactureras son aquellas que producen bienes de algún tipo a través de una combinación de trabajo humano, equipo automatizado, herramientas y sistemas similares. La fabricación se basa en los pasos principales de fabricación, procesamiento

y preparación de productos, a menudo comenzando con materias primas para dar como resultado productos para los consumidores.

La industria manufacturera desempeña un papel fundamental en la economía mundial y europea. Se considera un sector clave debido a su contribución significativa al crecimiento económico, la creación de empleo y la generación de valor agregado.

Esta industria ha experimentado cambios significativos a lo largo del tiempo debido a la globalización, los avances tecnológicos y las demandas de los consumidores y del mercado. Para mantenerse competitivas ante estos desafíos, las empresas manufactureras buscan la innovación en productos y procesos para la mejora continua.

Dentro de esta mejora, el control de calidad es una parte crucial para garantizar que los productos fabricados cumplan con los estándares de calidad establecidos. A través del control de calidad, se busca detectar y corregir cualquier defecto ocurrido durante la producción de los productos antes de que estos sean entregados a los consumidores. Un defecto está asociado a una característica o propiedad que no cumple con ciertos estándares [1]. Sin embargo, el control de calidad no solo evalúa el producto final, sino que está presente durante todo el proceso de producción y sirve para aumentar la eficiencia y productividad, reducir los costes, gestión de riesgos, etc.

Actualmente, y a pesar de los avances tecnológicos, muchos procesos de inspección siguen realizándose manualmente. Esto, dependiendo del sector y la finalidad de los productos, puede ser perjudicial para las empresas, puesto que pueden ocurrir errores humanos, dificultad para gestionar grandes volúmenes de producción, limitaciones de tiempo y capacidad, etc. Por estos motivos, en la actualidad cada vez más organizaciones están integrando técnicas de IA en sus líneas de producción [2][3]. Esto, junto a la integración de sensores, el Internet de las Cosas (IoT) y la mayor disponibilidad de datos, conduce al sector de la manufacturación a una nueva era digital denominada Industria 4.0 [2].

La inspección de calidad automatizada en la industria manufacturera comúnmente es usada junto a la visión artificial [4]. Esta se basa en el uso de cámaras junto a algoritmos de procesamiento de imagen para inspeccionar visualmente cada componente durante todo el proceso de producción. Esto permite asegurar si cada componente cumple con los estándares establecidos, y poder así descartar las piezas defectuosas de la línea de producción, aumentar la productividad y flexibilidad, eliminar la subjetividad de la inspección humana, etc. Por lo tanto, estos sistemas de inspección basados en tecnologías inteligentes traen un balance positivo a la industria manufacturera [5].

En la actualidad, una de las técnicas más empleada en visión artificial es el aprendizaje

automático (*Machine Learning*) (ML). El aprendizaje automático permite a los sistemas de visión artificial aprender patrones y características a partir de imágenes. Existen diferentes enfoques, como el aprendizaje supervisado y no-supervisado, que tienen muchas utilidades diferentes en la industria [6].

Sin embargo, en muchas ocasiones el ML tradicional queda limitado por las exigencias que requieren muchas tareas en la industria. El principal motivo es que requieren del procesamiento de los datos a una forma de representación adecuada para la solución de cada problema específico [7][8]. Este problema se solventa haciendo uso de técnicas más avanzadas como el *Deep Learning* (DL) puesto que permite trabajar con una mayor cantidad de datos complejos.

El DL o aprendizaje profundo es una rama del ML que se centra en el entrenamiento de redes neuronales profundas. El DL suele rendir mejor y es más preciso en situaciones donde la escala y complejidad de los datos es mayor. Además, como se acaba de mencionar, el ML tradicional implica extraer y seleccionar características relevantes de los datos antes del proceso, mientras que el DL tiene la capacidad de aprender automáticamente las representaciones de características y patrones eliminando, en gran medida, la necesidad de la extracción manual.

Por lo general, las técnicas de aprendizaje profundo suelen rendir mejor haciendo uso de grandes conjuntos de datos [9]. Esto se debe a que estas arquitectura tienen muchos parámetros que requieren de una cantidad suficiente de datos para que se ajusten adecuadamente. Sin embargo, en muchas ocasiones la adquisición de los datos en la industria es muy difícil debido a la privacidad, calidad de los datos, costes y más motivos. Esto entorpece el entrenamiento de las redes y puede ser un problema para garantizar la calidad de los resultados.

A raíz de este problema de la disponibilidad de los datos, han ido surgiendo nuevas técnicas para poder trabajar en entornos con pocos datos, donde la mayoría se basan en aumentar el conjunto de datos disponible o usar conocimiento previo de otros datos, como por ejemplo el *Data Augmentation* [10], el aprendizaje por transferencia [11] o la generación de datos sintéticos [12], entre otras. Además de estas técnicas conocidas, en los últimos años han surgido otras más centradas en aprender a partir de un número muy limitado de ejemplos como es el caso del paradigma *Few-shot learning* [13], con el que se trabajará en este proyecto.

1.2. Planteamiento del trabajo

El planteamiento del TFG se centra en el estudio de los beneficios de una red de segmentación basada en *few-shot learning* para detectar defectos en componentes industriales cuando se dispone de un conjunto de datos pequeño. El objetivo principal es abordar el desafío de realizar inspecciones de control de calidad en entornos industriales donde la disponibilidad de datos etiquetados es limitada.

En primer lugar, se han capturado imágenes de los componentes a inspeccionar utilizando un microscopio. Esta elección se justifica por su capacidad para capturar detalles minuciosos y permitir la detección precisa de defectos en los componentes. Estas imágenes se utilizarán como base de datos para el entrenamiento y evaluación de los modelos de segmentación.

Para poder entrenar un modelo de segmentación, se requiere un conjunto de datos anotados. Por lo tanto, se han realizado anotaciones en las imágenes capturadas para identificar y delimitar los defectos presentes en los componentes. Estas anotaciones se utilizan para construir una base de datos anotada que servirá como referencia durante el proceso de entrenamiento y evaluación de los modelos.

Una vez se ha obtenido la base de datos anotada, se procede a realizar experimentos utilizando diferentes hiperparámetros de la red. Esto permite explorar distintas configuraciones y ajustes de la arquitectura de la red de segmentación con el fin de identificar la configuración óptima que maximice la precisión y el rendimiento en la detección de defectos.

Además, se han entrenado redes de segmentación convencionales del estado del arte para realizar una comparación con la solución propuesta basada en *few-shot learning*. Esta comparación es fundamental para evaluar la efectividad y superioridad de la red de segmentación basada en *few-shot learning* en comparación con los enfoques tradicionales.

Los resultados obtenidos durante los experimentos han demostrado que la solución basada en *few-shot learning* ofrece mejores resultados en la detección de defectos en componentes industriales cuando se dispone de un conjunto de datos pequeño. Esto implica que la red de segmentación desarrollada con este enfoque es capaz de aprender eficientemente a identificar y delimitar los defectos a pesar de la limitada cantidad de datos de entrenamiento disponibles.

Finalmente, se destaca que la inferencia realizada por la red de segmentación basada en

few-shot learning es rápida. Esto es un aspecto crucial en un escenario industrial, ya que permite realizar inspecciones en tiempo real y en línea de producción, mejorando así la eficiencia y la calidad del control de calidad industrial.

1.3. Estructura del trabajo

El resto del documento se estructura como se indica a continuación:

En el Capítulo 2 se definen los objetivos generales y específicos planteados en el presente TFG, así como las hipótesis planteadas y la metodología propuesta.

En el Capítulo 3 se analiza el Estado del Arte en temas de *Deep Learning*, control de calidad industrial automatizado, la carencia de datos en este ámbito y el paradigma *few-shot learning*.

Posteriormente en el Capítulo 4, se expone el desarrollo del proyecto en cuanto a los materiales y métodos utilizados. Se presenta el *pipeline* seguido en todo el proceso de desarrollo del sistema automático de detección de defectos. Se presentan los trabajos realizados en términos de adquisición de imágenes, generación de base de datos, entrenamiento de modelos, validación y evaluación para su despliegue en entornos industriales.

En el Capítulo 5 se presenta la experimentación realizada y sus resultados, tanto con la técnica de segmentación basada en FSL como con las redes de segmentación convencionales.

Por último, se exponen las conclusiones obtenidas y las futuras líneas de trabajo en el Capítulo 6.

2. CAPÍTULO

Objetivos, hipótesis y metodología del trabajo

En este capítulo se enuncian los objetivos generales y específicos del trabajo, las hipótesis planteadas y la metodología propuesta para alcanzarlos.

2.1. Objetivos e hipótesis

El objetivo general de este TFG es **realizar un estudio sobre técnicas avanzadas de IA aplicada a imágenes en escenarios industriales con escasez y/o desbalanceo de datos entre clases**. El enfoque se centra en el **control de calidad industrial** y busca **superar el desafío de la falta de datos mediante el uso de redes de segmentación del estado del arte y una red basada en *Few-Shot Learning***.

A continuación, se recogen otros objetivos específicos derivados de este objetivo principal:

- Generar una **base de datos** para poder llevar a cabo la experimentación. Para ello, se debe realizar el montaje de un equipo microscópico para poder materializar la adquisición de imágenes sobre componentes industriales con defectos, además de generar las anotaciones correspondientes a estas imágenes.
- **Optimizar y experimentar con los hiperparámetros de la red** para maximizar los resultados de los modelos de segmentación para la detección de defectos.

- Comprobar si el método de ***Few-shot learning*** es, efectivamente, una **alternativa en entornos de pocos datos** a las arquitecturas convencionales de DL.
- Comprobar la **viabilidad de implementación y despliegue** en entornos reales o productivos.

A partir estos objetivos, se definen las siguientes hipótesis:

- Un ajuste bueno de hiperparámetros y una base de datos de calidad, balanceada y bien anotada permitirá mejorar el rendimiento de los métodos empleados.
- El método de *Few-shot learning* es capaz de generalizar con muy pocas muestras de entrenamiento y va ser capaz de superar en rendimiento a otras redes de segmentación convencionales.
- La arquitectura de *Few-shot learning* es óptima en tiempo de inferencia y se ajusta a la cadencia de producción. Además, es equilibrada en términos de *Precision* y *Recall*.

2.2. Metodología

La hipótesis sobre la que se sustenta el trabajo se basa en que el uso de redes de segmentación basadas en *Few Shot Learning* permite detectar defectos en componentes industriales de una forma robusta cuando no se tiene un gran conjunto de datos para el entrenamiento. Para demostrar esta hipótesis se han realizado una serie de experimentos.

Como punto de partida, para establecer una posible solución al problema, se ha realizado un estudio del Estado del Arte sobre el control de defectos superficiales empleando técnicas de visión artificial, así como la problemática asociada a ellos, y las estrategias seguidas cuando no se tienen datos suficientes disponibles. Este estudio permitirá dimensionar los problemas presentes en la industria y las técnicas disponibles para solucionarlos.

A continuación, se trabajará en el proceso de adquisición de imágenes para crear la base de datos con la que entrenar el sistema de DL. Como los componentes a inspeccionar cuentan con defectos muy pequeños y tienen una superficie reflectante, se empleará un equipo de adquisición acorde a esta problemática. Además, estos componentes cuentan con geometrías complejas y variables, teniendo zonas a diferentes alturas para inspeccionar.

Por ello se emplearán técnicas novedosas de adquisición, como es el sistema TAGLENS. Este sistema de microscopía varifocal permite obtener de manera ultra-rápida una sola imagen con una amplia gama de distancias focales.

Una vez obtenidas las imágenes con las que trabajar, se realizará un proceso de etiquetado para identificar los defectos con claridad en ellas.

Gracias a esas etiquetas, se entrenará un modelo de segmentación basado en FSL que permitirá obtener resultados óptimos en cuanto a detección de defectos, con el objetivo de intentar mejorar los métodos de segmentación empleados hasta la actualidad.

Con el objetivo de mejorar la precisión del modelo, se realizarán diferentes experimentos que permitan elegir los hiperparámetros óptimos para su aprendizaje.

Con los resultados obtenidos a partir de los experimentos, se entrenarán diferentes arquitecturas de segmentación basadas en DL como Unet [14], DeepLabv3+ [15], FPN [16], PAN [17], PSPNet [18] y Unet++ [19] para verificar si los resultados son peores. Como se tienen pocas imágenes en el entrenamiento, se aplicarán técnicas de transferencia de conocimiento usando pesos pre-entrenados en la base de datos pública ImageNet [20].

Se realizará una evaluación y comparativa de resultados con un conjunto de test externo que no ha sido visto nunca por el modelo, con el objetivo de obtener resultados imparciales y los más semejantes al funcionamiento del sistema en el futuro.

3. CAPÍTULO

Estado del arte

El presente capítulo ofrece una revisión exhaustiva y actualizada del estado del arte en el campo de estudio. Se recopilan y analizan los avances más relevantes y las investigaciones más recientes relacionadas con el tema, con el objetivo de establecer un contexto sólido para la investigación actual.

3.1. Control de calidad automatizada

Los productos fabricados requieren un control de calidad que asegure la condición óptima de los mismos. Para realizar estos controles, es importante tener un conocimiento del proceso de fabricación para analizar las posibles causas que puedan originar defectos en los materiales. El control de calidad comprende diferentes tipos de inspecciones, que se pueden dividir en: análisis superficial para detectar desviaciones estéticas, estudios sobre su composición química o sus características mecánicas y análisis de las características geométricas para localizar defectos dimensionales. Este TFG se centra en el primero de ellos, en el análisis de defectos superficiales, por lo que se profundizará en este tema.

El análisis superficial se suele llevar a cabo normalmente de manera visual. Una de las ventajas de realizar de esta manera el análisis es aprovechar la gran capacidad que tienen los humanos para buscar patrones o percibir colores y variaciones de tono. Sin embargo, este proceso también tiene sus desventajas, y es que este proceso repetitivo y tedioso está condicionado a diversos factores físicos y psicológicos humanos, como por ejemplo la fatiga [21]. Por lo que este proceso es propenso a errores, lo que conlleva a un aumento

en las no detecciones, dejando pasar productos defectuosos y afectando a la satisfacción del cliente final.

Este problema ha llevado el proceso de análisis hacia una automatización empleando sistemas de visión. Estos sistemas usan imágenes generadas por cámaras o sensores para la inspección automática o semi-automática (requiere algo de asistencia humana) de tareas como la detección de defectos [22]. Esta automatización se basa en dos etapas fundamentales, la adquisición de imágenes y el procesamiento de las mismas. La fase de adquisición es decisiva para obtener unos datos de calidad, que sirvan como *input* para la etapa de procesamiento. Una imagen que resalte los defectos y que cuente con un nivel de detalle elevado, facilitará la siguiente etapa. Sin embargo, obtener imágenes de calidad no es una tarea sencilla, ya que en la industria hay diferentes retos que hay que superar, como por ejemplo:

- La alta cadencia de producción a la que se fabrican los componentes.
- El sistema tiene que ser robusto para mantener un bajo ratio de falso rechazo y una baja tasa de no detecciones.
- La mayor parte de la producción es de piezas libres de defectos, por lo que conseguir muestras no es sencillo.
- Generalmente se cuenta con superficies metálicas reflectantes que se ven afectadas por la iluminación ambiente.
- El entorno productivo es hostil, se tienen vibraciones, partículas en suspensión, grasa, etc.
- Los catálogos de componentes suelen ser muy extensos, y cada componente suele tener geometrías complejas.

Para superar estos retos, se diseñan sistemas complejos de adquisición que permitan hacer frente a todas las adversidades descritas, y que permitan definir la topografía de la superficie y los defectos lo mejor posible [23] [24]. En cuanto al procesamiento, una de las técnicas más empleada en visión artificial y que mejores resultados está obteniendo es el *Machine Learning* [25]. El ML se refiere a un conjunto de técnicas cuyo objetivo es desarrollar métodos que permitan a los ordenadores aprender a través de la creación de programas que induzcan conocimiento. Estos programas se basan en el análisis de datos y la estadística, buscando patrones en dichos datos para modelar un programa que facilite el

aprendizaje. Este tipo de técnicas tiene aplicaciones en diversos campos como la robótica [26], la clasificación de imágenes [27] o la detección de objetos [28], entre otros muchos ejemplos.

Sin embargo, como se ha mencionado en la Sección 1.1, aunque los métodos de ML tradicionales pueden ser muy útiles en diferentes problemas, presentan el inconveniente de que son estrictamente dependientes a entornos controlados y son muy específicos para cada problema. En concreto, el pre-procesado y la extracción de características son los pasos que más arduos de realizar [29].

Gracias a las técnicas de *Deep Learning* que han surgido estos últimos años, este problema se ve resuelto puesto que no requieren un trabajo previo especializado a cada problema [29] [30]. Esto ha supuesto una mejora significativa en rendimiento para el control de calidad automatizado. A continuación, se mencionan algunas aplicaciones que usan métodos de DL para la inspección de calidad.

En industria, uno de los casos de uso habituales es la detección y clasificación de componentes, o la verificación de ensamblajes [31]. Para ello, se suelen usar arquitecturas para la detección de objetos, las cuales se basan en detectar y localizar objetos de diferentes clases en una determinada imagen. La mayoría de técnicas se basan en redes neuronales convolucionales (CNN), de las cuales las arquitecturas YOLO [32] y R-CNN [33] son algunas de las más conocidas. Existen múltiples aplicaciones desarrolladas en esta línea que han obtenido resultados más que satisfactorios [34] [35].

Por otro lado, otra aplicación del DL en el control de calidad es la detección de anomalías, que trata de identificar patrones inusuales que no se ajustan al comportamiento esperado en el conjunto de datos. Este tipo de técnicas son muy interesantes y útiles, debido a las características de la base de datos necesaria para entrenar que emplean, ya que está compuesta mayormente por imágenes libres de defecto. Como se ha mencionado previamente, este escenario es el más habitual en la industria, por lo que facilita el desarrollo del modelo. Los resultados obtenidos con estas técnicas no son tan precisos como con otras técnicas, pero es adecuado para determinadas aplicaciones [36] [37].

Por último, la segmentación semántica es otra técnica que trata de asignar a cada píxel de una imagen una etiqueta de la clase perteneciente para poder separar las regiones u objetos de una imagen [38]. Es muy útil y precisa para la tarea de detección de defectos y se ha validado en numerosas aplicaciones [39] [40] [41], por lo que este ha sido el enfoque que se ha elegido para este trabajo y sobre el que se va a profundizar.

3.2. Segmentación semántica

La segmentación semántica es una tarea de visión artificial que consiste en clasificar cada píxel de una imagen en una clase [42]. A diferencia de la clasificación de imágenes tradicional, donde el objetivo es asignar una sola etiqueta a toda la imagen, la segmentación semántica tiene como objetivo identificar regiones y objetos en una imagen asignando una etiqueta a cada una de ellos.

La segmentación de imágenes es una técnica muy utilizada en diferentes ámbitos, como el diagnóstico de enfermedades a partir de imágenes médicas [43], aplicaciones de procesamiento 3D [44], en sistemas de vehículos autónomos [45] y muchos más.

En la literatura se puede encontrar una gran variedad de algoritmos de segmentación tradicionales. La segmentación de imágenes comenzó originalmente con el procesamiento de imágenes junto a algoritmos de optimización. Alguno de estos métodos son el *Tresholding* [46], segmentación basada en clústeres [47], *Watershed* [48], etc.

Sin embargo, en la última década las técnicas de DL han avanzado mucho, lo que ha producido una nueva variedad de algoritmos para la segmentación semántica. En muchas de las tareas que afronta la segmentación semántica estos nuevos métodos ofrecen mejor rendimiento que los algoritmos tradicionales mencionados previamente. Esto ha resultado en un cambio de paradigma en el ámbito de visión artificial, donde ahora las redes neuronales profundas (DNN) se usan en la gran mayoría de las tareas de segmentación.

En general, las DNN para la segmentación semántica se basan en arquitecturas que constan de dos partes principales: el codificador (*encoder*) y el decodificador (*decoder*). El codificador es responsable de capturar las características de nivel superior de la imagen de entrada. Existen diferentes tipos de codificadores: convolucionales, recurrentes, autoencoders, etc. El decodificador, por otro lado, toma la representación codificada generada por el codificador y se encarga de procesarlo para generar la salida segmentada de la imagen de entrada. Al igual que los codificadores, existen decodificadores con enfoques totalmente diferentes.

De este modo, algunas de las arquitecturas de este tipo más utilizadas y relevantes en la tarea de detección de defectos son las siguientes: Unet [14], DeepLabv3+ [15], FPN [16], PAN [17], PSPNet [18] y Unet++ [19]. Estas son las que se han utilizado en este trabajo y se detallan un poco más en la Sección 4.3.2 del informe.

Estas redes se pueden diseñar y entrenar desde cero, pero para aprender las representacio-

nes adecuadas requieren de muchos datos. Puesto que en muchas ocasiones no es posible obtener la cantidad adecuada, para afrontar el problema de la poca disponibilidad de datos se hace uso de técnicas de *transfer learning* [49]. La idea detrás de estas técnicas es aprovechar el conocimiento y representaciones obtenidas previamente en una tarea similar con el objetivo de incrementar el rendimiento [30].

Una técnica dentro del aprendizaje por transferencia es la extracción de características, que implica utilizar una parte de una red pre-entrenada, generalmente las capas convolucionales, como un extractor de características para obtener representaciones generales de las imágenes, como bordes, texturas y formas. Estas características extraídas se usan después para entrenar otra red separada para la tarea específica.

En el contexto de las arquitecturas de segmentación semántica, estas redes pre-entrenadas son los codificadores. Algunas de las más conocidas con las que se ha trabajado en este proyecto son ResNet [50], EfficientNet [51], VGG [52] y Mix Vision Transformer [53]. Estas redes son entrenadas en conjuntos de datos públicos que contienen miles de imágenes de todo tipo, entre otros, ImageNet [54], The PASCAL VOC 2012 [55] y MS COCO [56]. Cabe mencionar que en este trabajo las redes pre-entrenadas han sido entrenadas en la base de datos ImageNet.

Sin embargo, las imágenes que contienen no tienen nada que ver con las necesidades que pueden surgir en la industria y en el control de calidad. Además, teniendo en cuenta que se disponen de pocos datos, en este contexto el uso de aprendizaje por transferencia no asegura que el rendimiento vaya a mejorar.

Por lo tanto, como resumen, en la tarea de segmentación semántica habitualmente se usan redes pre-entrenadas para la extracción de características antes de entrenar las arquitecturas de segmentación especificas. Es decir, se puede utilizar las redes pre-entrenadas (ResNet y EfficientNet, por ejemplo) como codificador en las arquitecturas de segmentación como Unet y las otras mencionadas previamente.

Por último, cabe mencionar que para poder entrenar los modelos se requiere de las imágenes de entrada, además de los datos etiquetados o anotados correspondientes a cada instancia del conjunto de datos. Una etiqueta o máscara es la propia imagen de entrada en la que cada píxel está marcado en la clase a que pertenece. En el contexto de detección de defectos, un píxel blanco indica que pertenece a la clase 'defecto' y uno negro a la clase 'no defecto'. La Figura 3.1 muestra una imagen con su correspondiente etiqueta.

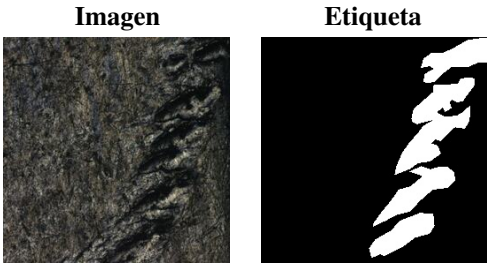


Figura 3.1: Ejemplo imagen y etiqueta correspondiente

En entornos industriales las cámaras permiten obtener muchas imágenes, pero para que luego estos datos sean útiles hay que pasar por este proceso de etiquetado [57]. Aquí reside el problema de la poca disponibilidad de los datos puesto que el proceso de obtención es manual, requiere mucho tiempo y puede ser caro.

Por lo tanto, para poder trabajar con pocos datos han surgido nuevas técnicas como *few-shot learning* que permiten trabajar sin la necesidad de tener miles de datos etiquetados.

3.3. Few-shot learning

Few-shot learning (FSL) es un paradigma dentro del ML que tiene como objetivo entrenar modelos para generalizar a partir de una cantidad limitada de datos [13].

Para ello, en general, se usan 2 conjuntos de datos diferentes:

- *Support* (soporte): Contiene un pequeño número de ejemplos etiquetados para cada clase que se usa para el entrenamiento.
- *Query* (consulta): Contiene ejemplos no vistos previamente por el modelo durante el entrenamiento y es utilizado para evaluar el rendimiento del modelo entrenado.

Existen variantes dependiendo de cuántas instancias se tienen por cada clase en el conjunto de soporte [58]. Con una sola imagen por cada clase se conoce como *One-shot learning*, con ninguna imagen por clase se le llama *Zero-shot learning* y con varias (no muchas) imágenes se le llama *Few-shot learning*. Aunque los términos parezcan estar relacionados, el contexto y los conceptos teóricos pueden ser totalmente diferentes.

En general, en estas técnicas aplican el aprendizaje por transferencia al usar modelos pre-entrenados como punto de partida, lo que permite mayor adaptabilidad y generalización

a diferentes problemas y ámbitos. Aunque este paradigma sea nuevo en la literatura y necesite más investigación, actualmente se usan en algunas aplicaciones como la clasificación de imágenes, detección de objetos, robótica, procesamiento del lenguaje natural y segmentación semántica.

Para poder trabajar con problemas relacionados concretamente con la segmentación semántica, se hace uso de una variante del FSL: *Few-shot segmentation* (FSS). La diferencia con el FSL es que clasifica las imágenes u objetos en categorías, mientras que FSS trata de clasificar cada píxel para obtener una máscara que delimita las diferentes clases. Por lo tanto, el conjunto de soporte debe de tener las imágenes con sus correspondientes máscaras, además de decodificadores para generar las máscaras.

4. CAPÍTULO

Materiales y métodos

En este capítulo se detalla el desarrollo realizado en este TFG paso a paso, exponiendo los materiales y métodos empleados para ello. El objetivo ha sido llevar a cabo el proceso que ejecutan los sistemas de inspección automática en la industria para la detección de defectos. Generalmente, este proceso se divide en varias etapas: adquisición de los datos, generación de la base de datos, diseño y entrenamiento de las redes neuronales, validación y despliegue.

En la Figura 4.1 se muestra este *pipeline* gráficamente. Es importante destacar que estos procesos son iterativos y suelen requerir ajustes y mejoras continuas para obtener un sistema de inspección automática eficaz y preciso.

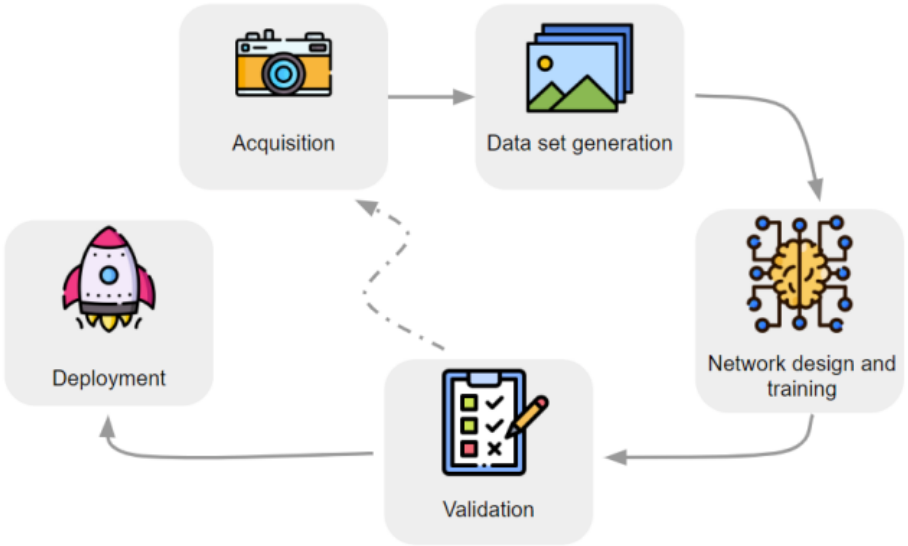


Figura 4.1: Proceso (*pipeline*) de los sistemas de inspección automática [8]

4.1. Primer paso: adquisición de imágenes

El primer paso es recopilar imágenes relevantes para el sistema de inspección automática. Esta adquisición tiene que ser óptima para las piezas a inspeccionar, de forma que represente lo mejor posible la superficie y sus defectos.

En este TFG los componentes a inspeccionar han sido espadines reflectantes metálicos con un recubierto de níquel, con diferentes variaciones, y una dimensión de 5cm de longitud y 0.8cm de anchura. En la Figura 4.2 se muestran una serie de espadines, donde el primer espadín en la parte derecha presenta un defecto muy grande que se puede ver visualmente, pero en la mayoría de los casos no se da este caso. Además, se puede apreciar la gran reflectividad que tiene la superficie, ya que al estar capturados con una iluminación no controlada, aparecen reflejos sobre ellos.

Figura 4.2: Pieza de trabajo

La adquisición de imágenes se ha realizado con un microscopio óptico. Sin embargo, no es un microscopio convencional puesto que utiliza la tecnología TAGLENS [1]. Es una tecnología de lentes de enfoque variable desarrollada por la empresa Mitutoyo. La tecnología TAGLENS utiliza una lente líquida que se puede cambiar de forma mediante la aplicación de una corriente eléctrica. Esto permite cambiar el enfoque de la lente sin necesidad de moverla físicamente. De este modo, se obtiene un enfoque rápido en diferentes distancias de trabajo, lo que permite una inspección más rápida y precisa.

Si los defectos son profundos, esto permite capturar también la parte interior del defecto. En cambio, con microscopios convencionales el interior de los defectos se vería desenfocado o completamente en negro. Por lo tanto, la tecnología TAGLENS permite mejorar el control de calidad considerablemente puesto se adquieren los defectos en su totalidad.

La Figura 4.3 enseña una muestra de como el sistema usado permite adquirir defectos con profundidad:

[1] https://www.mitutoyo.com/taglens/

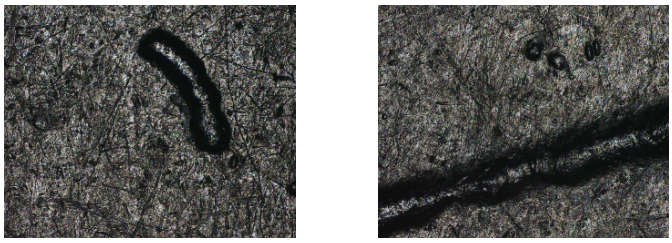


Figura 4.3: Ejemplo adquisición en defectos con profundidad

La lente utilizada captura una distancia de 3.39mm de anchura y 2.71mm de altura reales y guarda las imágenes en una resolución de 1280 x 1024 píxeles, lo que equivale a una distancia de 2.65 micras reales entre píxeles. Por lo cual, cada imagen representa un área muy pequeña del material y los defectos resultan muy difíciles de detectar por el ojo humano.

La Figura 4.4 muestra el sistema de adquisición de imágenes. Además de la lente que se acaba de aludir, el sistema cuenta con diferentes componentes: PLS (*Pulsed Light Source with fibre*) para iluminar la muestra que se está observando, un PC para utilizar el software asociado al microscopio, una cámara *USB3 Vision* para pasar las imágenes obtenidas por el microscopio al PC y un controlador para gestionar la comunicación entre los componentes. El montaje de este sistema se ha llevado a cabo desde cero, lo que ha requerido un proceso de aprendizaje para comprender correctamente como funciona cada componente y como usarlos con el fin de obtener una adquisición óptima.

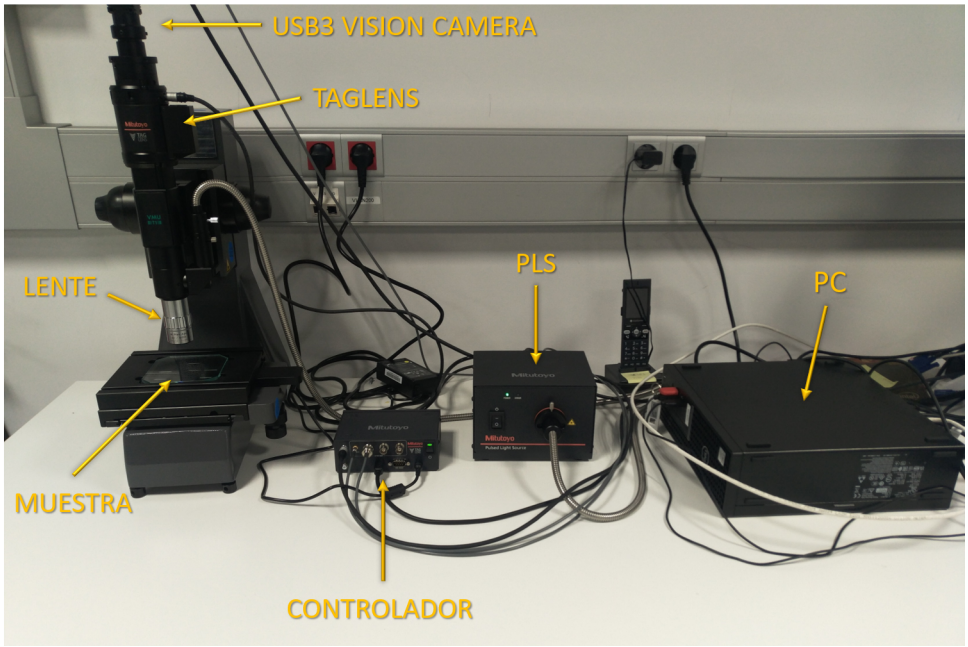


Figura 4.4: Sistema de adquisición de imagenés

Por último, la Figura 4.5 incluye algunas adquisiciones realizadas sobre muestras con y sin defectos. Se puede apreciar la variabilidad de los defectos, puesto que la forma, profundidad e incluso el color son diferentes.

Figura 4.5: Muestra adquisición de imagenés

4.2. Segundo paso: generación de la base de datos

Los métodos de segmentación semántica haciendo uso del aprendizaje profundo requieren de, además de la imagen de entrada, su anotación correspondiente. Un dato anotado o etiquetado, como se ha comentado previamente, se refiere a la propia imagen de entrada donde cada píxel está asociado con una etiqueta o clase específica que indica a qué categoría o región pertenece. La tarea de detección de defectos se trata de segmentación binaria, donde cada píxel se etiqueta como perteneciente a la clase 'defecto' o 'no defecto'.

Los modelos de segmentación requieren imágenes etiquetadas para el entrenamiento porque necesitan aprender a asignar las etiquetas correctas a los píxeles de una imagen. De este modo, el modelo ajusta sus parámetros para que pueda generalizar y segmentar correctamente nuevas imágenes no vistas durante la fase de entrenamiento. Además, son necesarios para poder sacar las métricas y validar el modelo.

La Figura 4.6 muestra la anotación de una imagen con y sin defecto. Los píxeles en blanco pertenecen a la clase 'defecto' y los negros a la clase 'no defecto'.

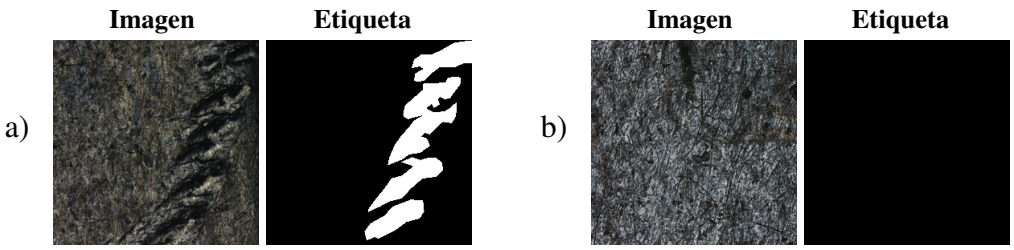


Figura 4.6: Ejemplo imagen y etiqueta correspondiente. (a) Con defecto; (b) Sin defecto.

Para crear anotaciones manualmente se ha usado la herramienta LabelMe[2] que permite anotar cada instancia de forma rápida y sencilla seleccionando manualmente las regiones de interés de las imágenes.

Con el sistema de captura mencionado y con la herramienta de anotación se ha compuesto una base de datos de un total de **120 imágenes**. Este *dataset* ha sido divididos en 3 conjuntos diferentes: entrenamiento, validación y test. Los conjuntos obtenidos por tanto se componen de la distribución de imágenes mostrada en la Tabla 4.1. En la Sección 5.1 se explica la razón de esta distribución.

Conjunto	Imágenes con defecto	Imágenes sin defecto	Imágenes en total
Entrenamiento	100 %	0 %	60
Validación	100 %	0 %	20
Test	50 %	50 %	40

Tabla 4.1: Distibución del dataset

Los defectos presentes en estos datos pueden ser muy variados como se aprecia en las imágenes de ejemplo de la Figura 4.5: desde pequeños defectos que en teoría deberían de ser fáciles de detectar hasta defectos complejos que incluso la anotación manual es complicado de realizar correctamente. Por este motivo, se ha intentado balancear la división de los datos. Es decir, poner en los 3 conjuntos instancias que en teoría se suponen que son fáciles y difíciles de segmentar. Si no se balancea los datos esto puede suponer un sesgo. Por ejemplo, si para la fase de test se guardan las imágenes sencillas de segmentar, esto podría dar como resultado mejores resultados que lo que en realidad debería de dar.

A continuación, en la Figura 4.7 se muestran tres ejemplos de a lo que nos referimos con imágenes fáciles y difíciles de segmentar. En la primera imagen es muy fácil delimitar el defecto. En cambio, en la segunda puede haber dudas si considerar la parte del medio de las dos líneas como defecto o no (en realidad sí es defecto, pero como el color es el mismo que el fondo de alrededor lleva a la confusión y es posible realizar una segmentación errónea). Por último, la tercera imagen es prácticamente imposible realizar una correcta

[2]https://github.com/wkentaro/labelme

y precisa anotación manualmente (se puede considerar que toda la imagen es defecto o anotar cuidadosamente región por región).

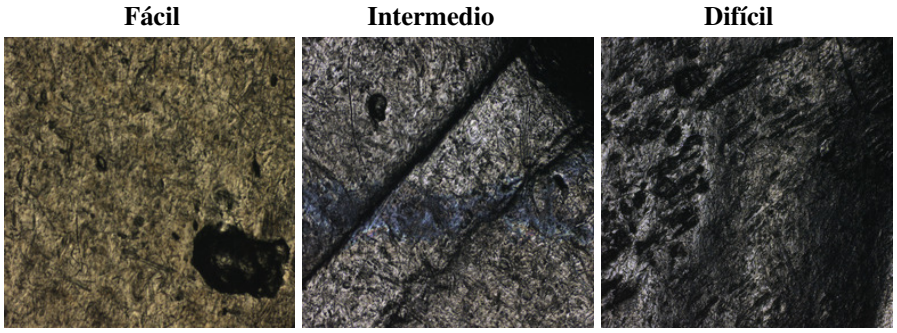


Figura 4.7: Ejemplos de imágenes con distintos grados de dificultad para la anotación.

Se ha mencionado previamente el problema que supone tener una base de datos reducida y que, además, esta sea muy variable. Pero, el hecho de que dentro de esta variabilidad haya instancias muy difíciles de segmentar supone un problema añadido para entrenar los modelos, puesto que la generalización será más complicada.

Además, es crucial etiquetar las imágenes de manera precisa y cuidadosa para garantizar la calidad de los modelos. Si las imágenes se etiquetan incorrectamente, por ejemplo, asignando una etiqueta incorrecta a una región específica o no etiquetando regiones importantes, la red de segmentación aprenderá patrones incorrectos o incompletos. Esto añade un sesgo y puede llevar a predicciones erróneas y resultados no confiables. Esto en el ámbito de la inspección de calidad es un aspecto muy importante a tener en cuenta.

En este caso, en las instancias difíciles de segmentar puede llegar a ser difícil saber dónde acaba un defecto o si una región es parte del defecto. En general, tanto los píxeles con defecto, como sin defecto son de un color de escala de grises, lo que dificulta la delimitación. Por este motivo, se ha llevado a cabo un proceso de etiquetado riguroso para poder asegurar la calidad de los datos. Sin embargo, idealmente habría que utilizar expertos en el dominio para generar etiquetas precisas y realizar una validación de la calidad de las etiquetas antes del proceso de entrenamiento.

4.3. Tercer paso: diseño y entrenamiento de las redes

En la Sección 3 se ha comentado los conceptos teóricos de las redes de segmentación y del *Few-Shot Learning (FSL)*. Ahora, en cambio, se detalla brevemente las diferentes arquitecturas utilizadas para las tareas contempladas este TFG.

4.3.1. Few-shot learning: DCAMA

Existen muchos algoritmos de *Few-Shot Learning* con fundamentos teóricos completamente diferentes. Por lo tanto, el objetivo ha sido encontrar dentro de la literatura algún método que sea eficaz para la tarea del presente proyecto.

A finales del año 2022 se publicó un artículo donde se propone un nuevo algoritmo de *FSS*: DCAMA (*Dense Cross-Query-and-Support Attention Weighted Mask Aggregation for Few-Shot Segmentation*) [59].

Este artículo indica que los enfoques de FSS propuestos hasta el momento no maximizan la información mediante la correlación entre los datos del conjunto de soporte y consulta. De este modo, gran parte de la información del fondo de las imágenes se pierde y se centra principalmente en los píxeles de primer plano.

Por lo tanto, el enfoque de este nuevo artículo es tratar de utilizar al máximo tanto la información del fondo como la del primer plano. Esto puede ser muy útil en este proyecto puesto que en muchas ocasiones no es fácil diferenciar el fondo y el defecto. Si este enfoque permite valorar de la misma forma el fondo y el defecto, es posible que aprenda más fácil delimitar las dos. Este es el motivo de la elección de este método.

La idea conceptual, de forma sencilla, es la siguiente: se analiza cada píxel de la imagen de consulta con todos los píxeles de la imagen de soporte. Si el píxel es similar (de la misma clase) se le asigna un peso grande y si no es similar un peso pequeño. De este modo, para la predicción de ese píxel específico de la consulta se realiza una suma ponderada entre todos los pesos calculados. Esto se puede ver en la Figura 4.8:

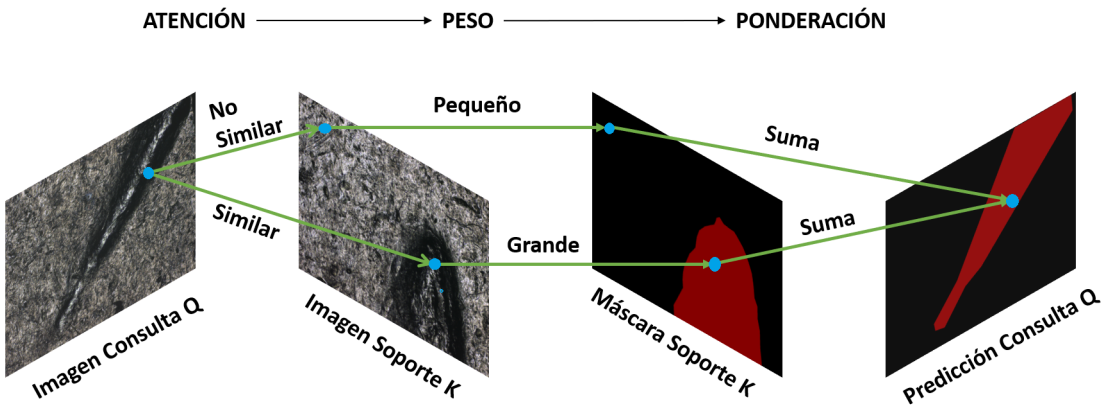


Figura 4.8: DCAMA idea conceptual

Esta arquitectura emplea técnicas de *Deep Learning* muy avanzadas sobre mecanismos de Atención y *Transformers* de Visión para la segmentación semántica, donde no se ha entrado a profundizar. En el artículo original [59] se profundiza más en la arquitectura y en los conceptos teóricos. El código es abierto[3], por lo que el trabajo ha consistido en entender la implementación y adaptarlo al presente problema.

Por último, mencionar que esta arquitectura permite trabajar con *Few-Shot Learning* y *One-shot Learning*. De este modo, para poder apreciar las diferencias entre estos dos casos en nuestro contexto, la experimentación se realizará usando una imagen de soporte para *One-Shot learning* y 5 imágenes de soporte para *Few-Shot learning*.

4.3.2. Redes de segmentación semántica

En la Sección 4.3.2 de este informe se ha explicado que son las redes neuronales de segmentación. A continuación, se profundizará en las diferentes arquitecturas que se han utilizado para la tarea que abarca este proyecto.

En resumen a lo explicado previamente, estas arquitecturas se dividen en dos módulos: codificador y decodificador. El codificador (*backbone*) suele ser una red de clasificación pre-entrenada. El decodificador, en cambio, toma la representación codificada generada por el codificador y lo procesa para generar la salida segmentada.

En la literatura se pueden encontrar muchas arquitecturas para diferentes propósitos, por lo que se ha escogido las más frecuentemente usadas y las más relevantes en la tarea de detección de defectos: Unet [14], DeepLabv3+ [15], FPN [16], PAN [17], PSPNet [18] y Unet++ [19]. A continuación, se detallan brevemente:

- **Unet:**
  Es una red neuronal convolucional (CNN) inspirada en FCNs (*Fully Convolutional Network*) desarrollada para aplicaciones de segmentación semántica en imágenes biomédicas a nivel microscópico, aunque ahora también se utiliza en otros campos. Esta arquitectura se caracteriza por su capacidad para realizar segmentación precisa en imágenes médicas, donde se requiere una alta resolución y detalles finos.

  El codificador (parte izquierda de la Figura 4.9) se encarga de reducir la resolución espacial de la imagen de entrada y extraer características relevantes disminuyendo

[3] https://github.com/pawn-sxy/dcama

los parámetros de la red, mientras que el decodificador (parte izquierda de la Figura 4.9) se encarga de aumentar gradualmente la resolución espacial y generar la salida segmentada. De la combinación de estas dos partes viene la forma 'U' de la arquitectura, y de ahí su nombre. En el artículo original [14] se profundiza en esta arquitectura.

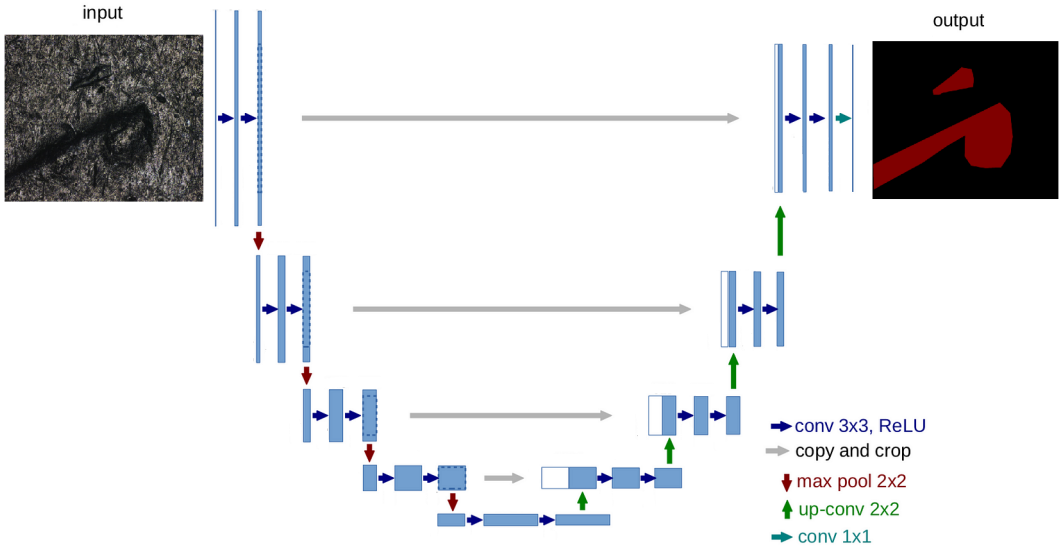


Figura 4.9: Arquitectura Unet [14]

- **DeepLabv3+:**
  DeepLabv3+ es una arquitectura de segmentación avanzada de Google, diseñado específicamente para segmentar objetos en imágenes con detalles de granularidad fina y fondos complejos. Esto puede ser útil para delimitar los bordes de los defectos y el fondo, puesto que la diferencia entre las dos clases puede ser muy pequeña.

  DeepLabv3+ también está diseñado en una estructura codificador-decodificador, pero, la peculiaridad es que se centra en capturar información contextual a través de convoluciones dilatadas [60] y *Atrous Spatial Pyramid Pooling* (ASPP) [61]. Haciendo uso de estas técnicas se extrae información relevante a diferentes escalas sin reducir significativamente la resolución espacial de la imagen de entrada (como ocurre en la arquitectura Unet, por ejemplo). En el artículo original [15] se detalla esta arquitectura y en los artículos [60, 61] se profundiza también estos dos métodos que utiliza, respectivamente. La figura 4.10 muestra la arquitectura:

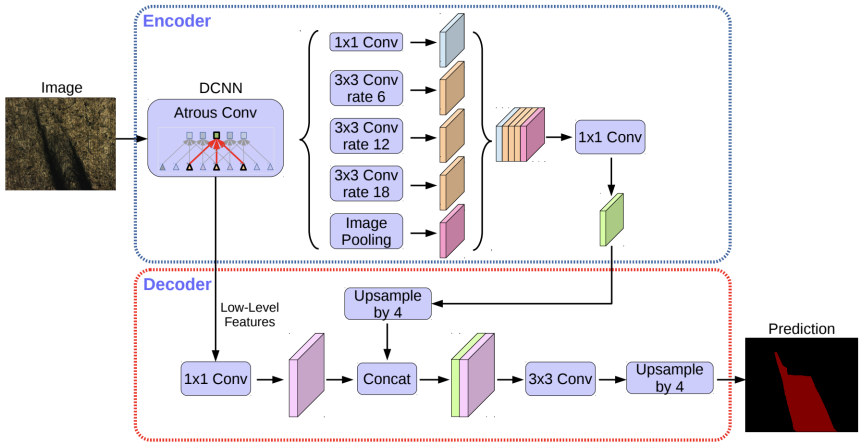


Figura 4.10: Arquitectura DeepLabv3+ [15]

- **FPN (*Feature Pyramid Network*):**

  Permite extraer características de la imagen de entrada en varios niveles. Hace uso de *backbones* para extraer la información de la imagen de entrada para crear una pirámide de características, y comparte estas características con las capas vecinas en el flujo de abajo hacia arriba y de arriba hacia abajo. Este flujo viene representado por las flechas negras en la Figura 4.11. En el artículo original [16] se profundiza en esta arquitectura.

  En el control de calidad es común que los defectos varíen en tamaño y forma. Algunos defectos pueden ser pequeños y sutiles, mientras que otros pueden ser más grandes y evidentes. La detección precisa de estos defectos requiere que el modelo sea capaz de capturar características relevantes en diferentes escalas. Esta arquitectura aborda este desafío al construir la pirámide de características.

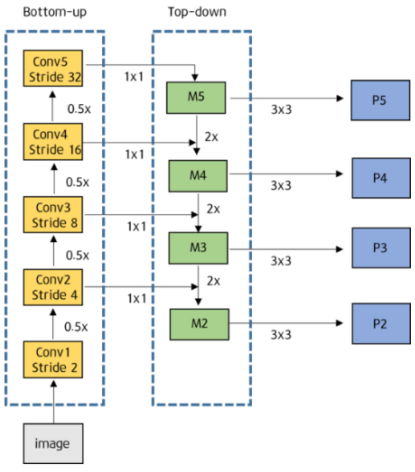


Figura 4.11: Arquitectura FPN [62]

- **PAN (*Path Aggregation Network*):**
  Es una extensión a la arquitectura FPN, por lo que está basada en la idea de agregar información para obtener una mayor precisión. Esta arquitectura está diseñada para mejorar el flujo de información entre los diferentes niveles de la pirámide, aumentando las rutas de información en este flujo para que la información sea más fácil de propagar. Este aumento y cambios de flujo vienen representados en la Figura 4.12 por la flecha roja y verde. En el artículo original [17] se profundiza en esta arquitectura.

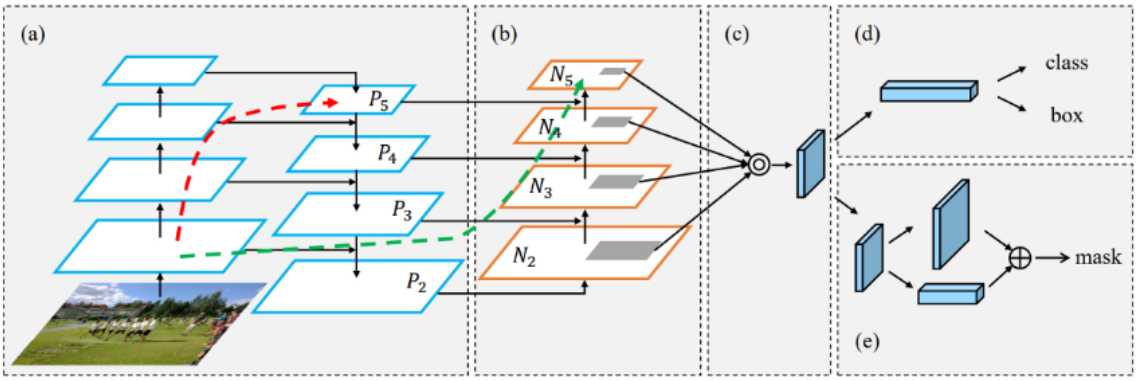


Figura 4.12: Arquitectura PAN [17]

- **PSPNet (*Pyramid Scene Parsing Network*):**
  PSPNet es un modelo de segmentación semántica que utiliza un módulo de análisis piramidal para explotar la información de contexto global, agregando también contexto de diferentes regiones para obtener una predicción final más robusta y confiable. Esta red realiza una extracción de características de la imagen de entrada con convoluciones dilatadas, lo que permite aumentar el campo receptivo. Estas capas se colocan en el final de la red principal, permitiendo obtener características más descriptivas. En el artículo original [18] se profundiza en esta arquitectura.

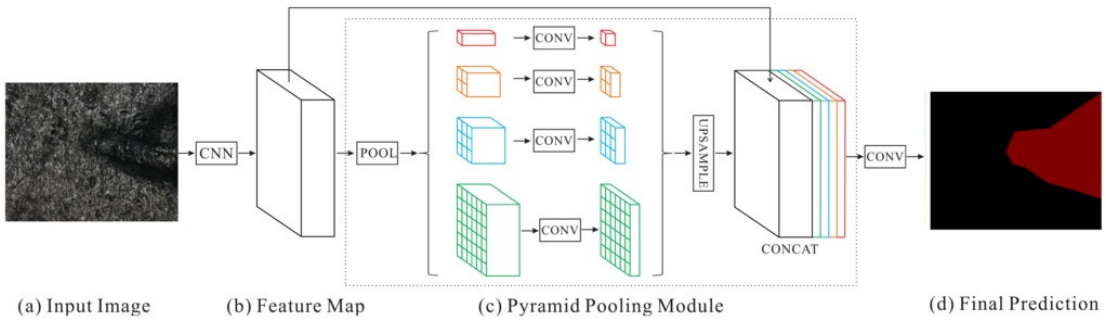


Figura 4.13: Arquitectura PSPNet [18]

- **Unet++:**
  Es una variante creada como mejora a la arquitectura Unet para la segmentación de imágenes médicas. Siendo una arquitectura más profunda y compleja, Unet++ ha demostrado ofrecer mejores resultados a la Unet original [63].

  A diferencia de la arquitectura original Unet, en vez de utilizar un solo codificador y decodificador, la arquitectura Unet++ utiliza bloques de codificador-decodificador que se conectan entre sí para formar una red más profunda y compleja que la Unet original. En el artículo original [19] se profundiza en esta arquitectura.

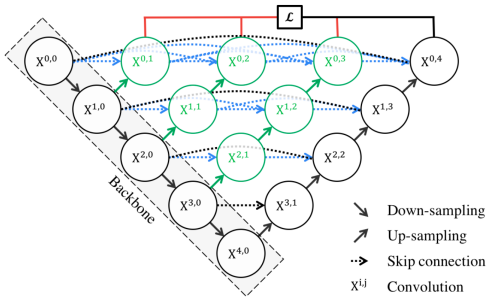


Figura 4.14: Arquitectura Unet++ [19]

Cabe mencionar que la implementación de estas arquitecturas y el entrenamiento de los modelos se ha llevado a cabo con el lenguaje de programación Python. Dentro de este lenguaje existen varias librerías que facilitan la implementación y el entrenamiento de las arquitecturas de segmentación semántica. A continuación, se detalla brevemente algunas de la principales librerías:

- **PyTorch**[4]**:** es una librería de aprendizaje automático y *deep learning* que permite aprovechar el poder de las GPUs para acelerar el entrenamiento y la inferencia de modelos.
- **Segmentation Models PyTorch**[5]**:** es una librería de PyTorch que facilita la implementación de modelos de segmentación semántica. Además, permite cargar modelos pre-entrenados como entrada a estas arquitecturas. Por lo tanto, se pueden implementar todas las arquitecturas de segmentación mencionadas previamente.
- **TensorBoardX**[6]**:** permite visualizar y analizar gráficamente los datos generados durante la fase de entrenamiento, lo que facilita el monitoreo y la depuración.

[4]https://pytorch.org/docs/stable/index.html
[5]https://smp.readthedocs.io/en/latest/
[6]https://tensorboardx.readthedocs.io/en/latest/

4.4. Cuarto paso: Validación

Después de entrenar las redes hay que evaluar el rendimiento y la precisión de los modelos utilizando datos no vistos durante el entrenamiento. Esto proporciona una medida objetiva de la calidad de los modelos y facilita la selección del mejor.

El proceso de validación generalmente implica tomar un conjunto de datos de validación separado que no se ha utilizado durante el entrenamiento. Este conjunto debe de estar bien construido y balanceado para que represente a todos los casos posibles que se pueda encontrar en un entorno de producción. Se ha profundizado en este tema previamente en la Sección 4.2.

Para esta evaluación y comparación entre los modelos en la segmentación semántica, que implica asignar una etiqueta a cada píxel de una imagen, las siguientes métricas son comúnmente las más utilizadas (por ende, las que se han usado en este trabajo):

- ***True Positives*** (TP): el defecto se ha detectado como defecto.
- ***True Negatives*** (TN): el fondo se ha detectado como fondo.
- ***False Positives*** (FP): el fondo se ha detectado erróneamente como defecto.
- ***False Negatives*** (FN): el defecto se ha detectado erróneamente como fondo.
- **IoU:** (*Intersection over Union*) o el Índice de Jaccard [64]: En el contexto de la segmentación semántica, se utiliza para evaluar qué tan bien se superponen la máscara de segmentación predicha por el modelo y la máscara de segmentación real o de referencia. Se calcula dividiendo el área de intersección por el área de unión de ambas máscaras.

$$IoU = J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.1}$$

  El resultado varía entre 0 y 1, donde 0 significa que la predicción no coincide en absoluto con la máscara de referencia y el valor 1, en cambio, indica que la predicción coincide exactamente con la máscara de referencia.

  Una predicción se considera valida si el valor de IoU es mayor que un umbral determinado. El valor de este umbral varia dependiendo del problema y contexto que se esta solucionando. En algunas tarea se puede considerar el valor 0.5 como umbral,

sin embargo, en la segmentación semántica en el campo de la detección de defectos superficiales requiere utilizar umbrales mas altos para garantizar la precisión. En este caso, se ha decidido escoger el umbral de 0.8.

- ***Recall:*** es la tasa de *True Positives*. Es decir, de todos los defectos, cuántos se han detectado como defectos. En segmentación semántica indica la capacidad de detectar correctamente todas las instancias de una clase específica (que no marque una muestra negativa como positiva), lo que se traduce en una baja tasa de *False Positives*.

$$Recall = \frac{TP}{TP+FN} \tag{4.2}$$

  Un *Recall* alto indica que el modelo está capturando adecuadamente las instancias de las clases, mientras que un Recall bajo indica que hay muchas regiones que no se han detectado correctamente, lo que implica la presencia de *False Negatives*.

- ***Precision:*** en la segmentación semántica mide la proporción de píxeles con defecto predichos correctamente en relación al total de píxeles clasificados como defecto. Una alta precisión indica que la mayoría de los píxeles positivos predichos son verdaderamente positivos y que hay pocos casos de *False Positives*.

$$Precision = \frac{TP}{TP+FP} \tag{4.3}$$

Es importante tener en cuenta que la *Precision* y el *Recall* pueden estar relacionados. Mientras que la precisión se centra en la calidad de los resultados positivos, el *Recall* se centra en la capacidad de detectar todos los casos positivos. En general, existe una relación inversa entre la *Precision* y el *Recall*: a medida que se aumenta la precisión, es posible que el *Recall* disminuya, y viceversa. Por ello, a la hora de evaluar es necesario considerar el equilibrio entre estas dos métricas.

4.5. Quinto paso: Despliegue

Una vez que el modelo ha sido validado y se considera adecuado, hay que desplegarlo en un sistema de inspección automática en producción. Esto implica implementar el modelo entrenado en un entorno en tiempo pseudo-real, donde recibe nuevas imágenes para realizar predicciones. El sistema de inspección automática puede estar integrado en una línea de producción o en un dispositivo independiente, según los requisitos del sistema.

El control de calidad, dependiendo de la componente que se está inspeccionando, tiene requisitos de velocidad muy exigentes. Las llaves examinadas en este trabajo son creadas en masa en una línea de producción, por lo que requiere que la inspección individual sea inmediata.

Los sistemas basados en *Deep Learning* necesitan hardware de cómputo avanzados para garantizar el rendimiento y la velocidad. En concreto, el uso de GPUs es imprescindible en sistemas de control de calidad puesto que permite acelerar el tiempo de entrenamiento y el procesamiento en tiempo real. Además, hay que seleccionar la arquitectura software óptima e ideal para la tarea específica puesto que esto también tiene impacto en el rendimiento del sistema.

Además, si el objetivo es trabajar con imágenes a nivel microscópico, hay que asegurar que el microscopio pueda ser acoplado a la línea de producción y que la captura sea rápida también. No sirve de nada que el software y hardware estén optimizados pero que luego no se pueda obtener las imágenes a la frecuencia necesaria. Esto no ocurre con la tecnología TAGLENS que se ha usado puesto que tiene una gran adaptabilidad a diferentes sistemas y entornos.

En este TFG se evaluará el tiempo que se tarda en realizar la inferencia de las imágenes, así como la estabilidad del modelo en cuanto a que no genere falso rechazo y no detecciones, para comprobar que sería viable su despliegue en una aplicación real.

5. CAPÍTULO

Resultados de la experimentación

En este Sección se detalla los resultados obtenidos con las diferentes arquitecturas mencionadas en la Sección 4.3. El objetivo es comparar los resultados obtenidos con la arquitectura DCAMA con las redes de segmentación convencionales para ver con cuál se obtiene el mejor resultado. Con esta experimentación se podrá afirmar si esta técnica de *Few-Shot Learning* puede ser una alternativa en la tarea de detección de defectos en entornos con pocos datos a las arquitecturas más frecuentes en la literatura.

5.1. Planteamiento de la experimentación

Para que la comparación sea justa y significativa, es importante que todos los modelos se prueben con los mismos parámetros. Para ello, se ha definido un *benchmark*. El *benchmark* en el aprendizaje automático se refiere a la evaluación del rendimiento de diferentes modelos de aprendizaje para un conjunto de datos y parámetros determinados. De esta manera, si todos los modelos están utilizando los mismos parámetros y los mismos conjuntos de datos, cualquier diferencia en el rendimiento se puede atribuir a las diferencias inherentes en los modelos en sí mismos. Esto permitirá una comparación justa y precisa del rendimiento de los diferentes modelos.

El primer parámetro a ajustar será el **tamaño del *batch*** puesto que puede suponer un impacto significativo en la convergencia y el rendimiento del modelo [65]. El tamaño del *batch* es el número de datos que se utilizan en cada iteración del algoritmo. Al disponer de pocos datos es recomendable utilizar un tamaño de *batch* pequeño para evitar el sobre-

ajuste (*overfitting*) y permitir que el modelo generalice mejor [66]. La experimentación se realizará con entre 10 y 60 imágenes de entrenamiento, por lo que los tamaños de *batch* seleccionados han sido 1, 2, 4 y 6.

El siguiente parámetro a ajustar será el **número de épocas**. Una época se refiere a una pasada completa del conjunto de entrenamiento durante el proceso de entrenamiento del modelo. Si el modelo se entrena con muy pocas épocas, es probable que no tenga suficiente tiempo para aprender las características importantes de los datos de entrenamiento y, por lo tanto, puede subajustarse (*underfitting*) a los datos de entrenamiento y no lograr buenos resultados en los datos de validación. En cambio, si el modelo se entrena con un número excesivo de épocas, puede que el modelo comience a memorizar los datos de entrenamiento en lugar de aprender patrones generales, lo que resultará una baja precisión en los datos de validación y que el modelo sobreajuste.

Cabe resaltar que el proceso de entrenamiento se ha realizado con instancias que contienen solo defectos. Agregar al modelo una gran cantidad de imágenes con una sola clase, como es el caso de las imágenes sin defecto, puede suponer un sesgo en favor a esta clase mayoritaria. Tan solo devolviendo la máscara segmentada completamente en negro todas las imágenes sin defecto serán predichas correctamente, por lo que las métricas obtenidas se verán 'infladas' y no proporcionará una evaluación realista del rendimiento del modelo.

En otras palabras, aunque el modelo sea malo las métricas obtenidas serán muy buenas porque segmentará bien las instancias sin defectos. Por lo tanto, el entrenamiento ha sido realizado solo en el conjunto de defectos para evitar este sesgo. Además, en general, los defectos forman una pequeña parte de la imagen y el fondo restante es igual a las imágenes sin defectos, por lo que la división de las clases sigue estando balanceado.

Por otro lado, la validación de los modelos se ha realizado en 2 escenarios diferentes: utilizando solo el conjunto con defectos otra vez por el motivo que se acaba de mencionar y utilizando todo el conjunto de test (imágenes con y sin defectos). En un entorno industrial real, la mayoría de piezas creadas en una línea de producción no tendrán defectos, por lo que se quiere comprobar el comportamiento en un caso real.

5.2. DCAMA

Además de obtener el mejor modelo posible, se quiere comprobar la cantidad mínima de datos con el que esta técnica de *Few-Shot Learning* puede adaptarse y generalizar correctamente. Para ello, las primeras pruebas han sido realizadas con una cantidad muy

limitada de datos, para luego ir aumentando y así poder apreciar cómo se comporta en entornos con más datos.

5.2.1. Base de datos reducida

La primera prueba ha sido con tan solo 10 imágenes de entrenamiento, 6 de validación y 40 de test (de las cuales, 20 son instancias con defectos y las otras 20 sin defectos). Se puede suponer que los resultados no serán buenos puesto que no son suficientes datos de entrenamiento como para generalizar correctamente.

El primer parámetro a ajustar será el **tamaño del *batch***. Como es de esperar, los resultados son malos. Al hacer uso de solo 10 imágenes de entrenamiento y 6 de validación no son suficiente para que la red converja. Esto significa que el modelo no ha aprendido a reconocer las características y patrones necesarios para la segmentación. Con diferentes valores del tamaño del *batch* se obtienen las mismas métricas, por lo que esto significa que el modelo no detecta nada, simplemente devuelve una máscara en negro. Además, esto ocurre tanto con las pruebas de 1-shot como con 5-shot. A continuación, en la Figura 5.1 se muestra los resultados junto a una gráfica de la métrica IoU durante el entrenamiento, que muestran que la red no ha convergido.

Solo defectos	
IoU	83.38
Recall	1.0
Precision	0.83

Todo el dataset	
IoU	91.68
Recall	1.0
Precision	0.92

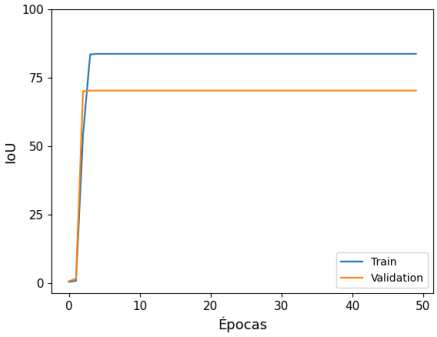


Figura 5.1: DCAMA: Resultados tamaño de *batch* dataset reducida

El motivo de que devuelva ese valor alto de la métrica IoU sin detectar ningún defecto es lo que se ha comentado en la Sección 5.1, la clase mayoritaria es el fondo de la imagen que son predichas correctamente, lo que hace que el valor de IoU suba. Esta información es valiosa puesto que ahora se tiene un punto de referencia en los dos escenarios.

La Tabla 5.1 muestra un resumen de los resultados con diferentes valores del **número de épocas**:

Dataset test con solo defectos				
		25	**50**	**100**
1-shot	IoU	83.38	83.38	94.75
	Recall	1.0	1.0	0.88
	Precision	0.83	0.83	0.83
5-shot	IoU	83.38	83.38	94.46
	Recall	1.0	1.0	0.88
	Precision	0.83	0.83	0.86

Dataset test entero				
		25	**50**	**100**
1-shot	IoU	91.68	91.68	97.30
	Recall	1.0	1.0	0.93
	Precision	0.92	0.92	0.92
5-shot	IoU	0.0	0.0	97.14
	Recall	1.0	1.0	0.94
	Precision	0.92	0.92	0.92

Tabla 5.1: DCAMA: Resultados épocas *dataset* reducida.

Con épocas 25 y 50 ocurre un subajuste de la red al igual que en la prueba anterior: la red no converge y no predice nada devolviendo una máscara en negro. Sin embargo, se aprecia como los resultados en la época 100 cambian drásticamente y la red empieza a aprender. Esto se puede ver en la Figura 5.2 donde se muestra la métrica IoU en la fase de entrenamiento:

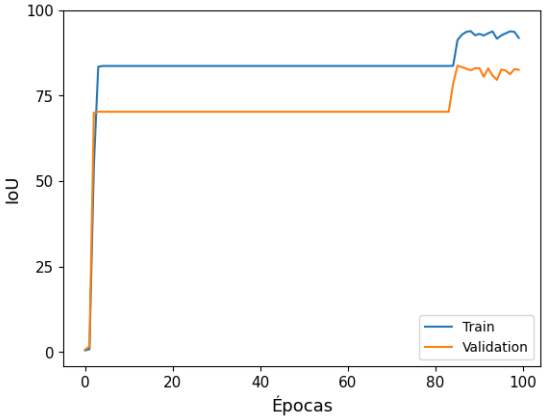


Figura 5.2: DCAMA: Gráfica entrenamiento épocas *dataset* reducida

La figura 5.3 muestra un ejemplo de la segmentación obtenida con las diferentes épocas:

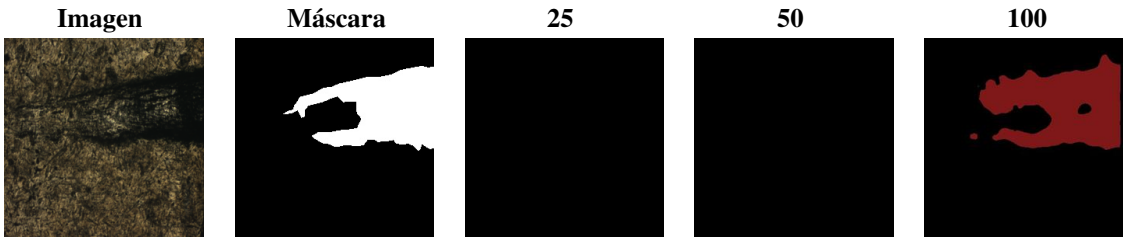


Figura 5.3: DCAMA: Ejemplo épocas *dataset* reducida

En un principio parecía que la red no respondía correctamente al número de datos de entrada introducidos. Sin embargo, con diversas pruebas se ha visto que con un número alto

de épocas se puede mejorar el resultado, aunque todavía no es un resultado lo suficientemente bueno. El modelo sitúa correctamente donde se encuentran los defectos, pero el resultado final no es muy preciso.

A continuación, se ha ampliado la base de datos para comprobar como reacciona ahora la arquitectura y ver si los resultados mejoran.

5.2.2. Base de datos ampliada

Se han realizado las mismas pruebas pero con un conjunto de datos mayor. En concreto, 30 imágenes de entrenamiento, 12 de validación y 40 de test (de las cuales, 20 son instancias con defectos y las otras 20 sin defectos).

La Tabla 5.2 muestra los resultados obtenidos con diferentes tamaños de *batch*:

		Dataset test con solo defectos						Dataset test entero			
		1	**2**	**4**	**6**			**1**	**2**	**4**	**6**
1-shot	IoU	0.928	**0.934**	0.932	0.933	**1-shot**	IoU	0.944	**0.976**	0.971	0.973
	Recall	0.86	**0.87**	0.86	0.86		Recall	0.91	**0.93**	0.92	0.92
	Precision	0.86	**0.89**	0.89	0.88		Precision	**0.94**	0.93	0.92	0.92
5-shot	IoU	0.919	**0.928**	0.927	0.922	**5-shot**	IoU	0.947	**0.959**	0.962	0.961
	Recall	0.86	**0.88**	0.87	0.87		Recall	0.93	0.93	**0.94**	0.93
	Precision	0.85	**0.87**	0.86	0.86		Precision	**0.95**	0.93	0.92	0.92

Tabla 5.2: DCAMA: Resultados tamaño de *batch dataset* ampliada.

Se ve que los resultados no varían mucho y por eso todas las métricas son muy parecidas, pero el tamaño 2 ofrece ligeramente mejores métricas. Sin embargo, la mayor diferencia es que con un tamaño pequeño del *batch* la red converge antes. Esto se puede ver en la Figura 5.4 donde se muestra la fase de entrenamiento:

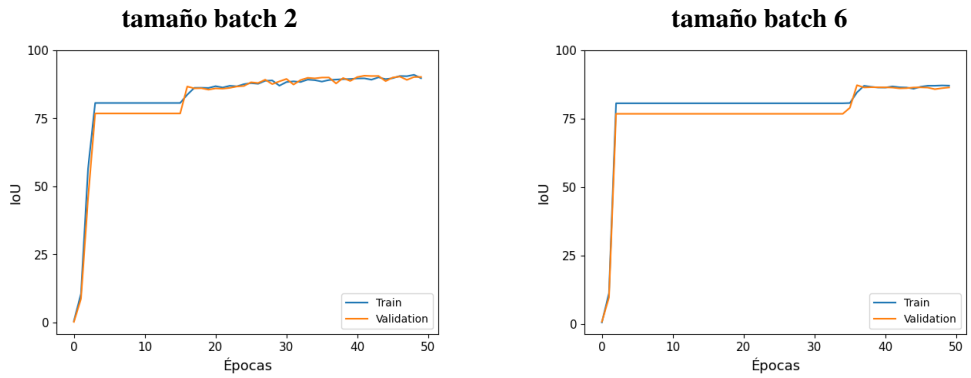


Figura 5.4: DCAMA: Gráfica entrenamiento diferentes tamaños de *batch dataset* ampliada.

A continuación, la Tabla 5.3 muestra los resultados de realizar las pruebas con diferentes valores del **número de épocas**:

Dataset test con solo defectos				
		25	**50**	**100**
1-shot	IoU	0.931	**0.934**	0.916
	Recall	0.86	**0.87**	0.85
	Precision	0.87	**0.88**	0.87
5-shot	IoU	0.923	**0.928**	0.919
	Recall	0.87	**0.88**	0.86
	Precision	0.87	0.87	**0.88**

Dataset test entero				
		25	**50**	**100**
1-shot	IoU	0.971	**0.976**	0.955
	Recall	0.92	**0.93**	0.92
	Precision	**0.94**	0.93	0.92
5-shot	IoU	0.948	**0.959**	0.956
	Recall	0.92	**0.93**	0.92
	Precision	**0.94**	0.93	0.93

Tabla 5.3: DCAMA: Resultados épocas *dataset* ampliada.

Con solo 25 y 50 épocas las redes convergen y ofrecen buenos resultados en general. Por otro lado, con 100 épocas se pasa del punto de convergencia y los resultados empeoran considerablemente (se sobreajusta). A medida que el modelo continúa entrenando durante más épocas, se adapta cada vez más a los detalles y el ruido del conjunto de datos de entrenamiento, en lugar de aprender patrones generales que se pueden aplicar a otros datos. Como resultado, el modelo obtiene buenas métricas en el conjunto de datos de entrenamiento, pero no en el conjunto de validación y test. En la Figura 5.5 se muestra un ejemplo con una imagen sin defecto donde se aprecia un falso rechazo principalmente con 100 épocas:

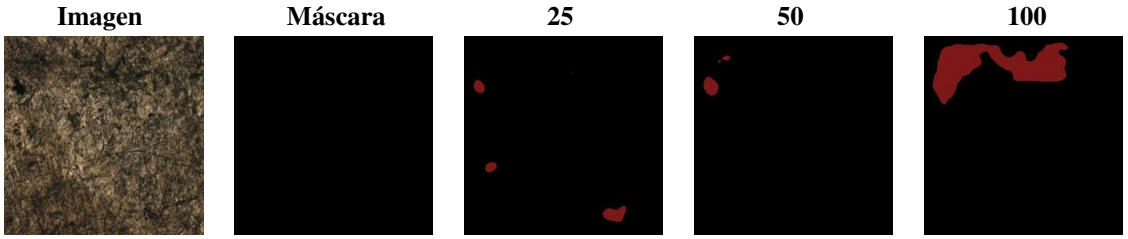


Figura 5.5: DCAMA: Ejemplo épocas *dataset* ampliada

La arquitectura mejora considerablemente al ampliar el conjunto de datos, como era de esperar. Sin embargo, las métricas finales no son tan diferentes comparando con los resultados del conjunto reducido. La principal diferencia es que con la base de datos ampliada la red converge mucho antes, lo que supone un menor tiempo de entrenamiento.

El hecho de usar solo 30 imágenes de entrenamiento y obtener métricas de IoU cercanas a 98 % es un resultado realmente bueno.

Como una última prueba para analizar el comportamiento del modelo, se ha repetido el mismo procedimiento volviendo a ampliar el tamaño del conjunto de datos. En concreto,

60 imágenes de entrenamiento, 20 de validación y 40 de test (de las cuales, 20 son instancias con defectos y las otras 20 sin defectos). Por no repetir otra vez el mismo proceso, la Tabla 5.4 muestra el resultado del mejor modelo obtenido:

	Dataset test con solo defectos	
1-shot	IoU	0.941
	Recall	0.86
	Precision	0.88
5-shot	IoU	0.94
	Recall	0.86
	Precision	0.87

	Dataset test entero	
1-shot	IoU	0.978
	Recall	0.92
	Precision	0.93
5-shot	IoU	0.965
	Recall	0.92
	Precision	0.93

Tabla 5.4: DCAMA: Resultado final *dataset* ampliada.

Aunque se haya vuelto a ampliar el conjunto de entrenamiento, el rendimiento obtenido es ligeramente superior a la anterior prueba. El mejor modelo ha llegado a obtener **97.8 %** la métrica IoU (**94.1 %** con las imágenes de solo defectos). Por lo tanto, se puede considerar que la red converge muy rápido y luego se estabiliza mejorando ligeramente los resultados a medida que se aumenten los datos. Para este caso en concreto, haciendo uso de entre 10 y 30 instancias de entrenamiento, la arquitectura llega a este punto de convergencia. A partir de aquí, a medida que se aumente la cantidad de datos, el rendimiento irá aumentando ligeramente.

Por otro lado, en cuanto a las diferencias entre 1-shot y 5-shot, se ha podido ver que la primera ofrece ligeramente mejores resultados. La Figura 5.6 muestra un ejemplo de la segmentación obtenida con las dos variantes. La primera imagen es la versión 1-shot con una única imagen de soporte, mientras que la segunda es la versión 5-shot con 5 imágenes de soporte.

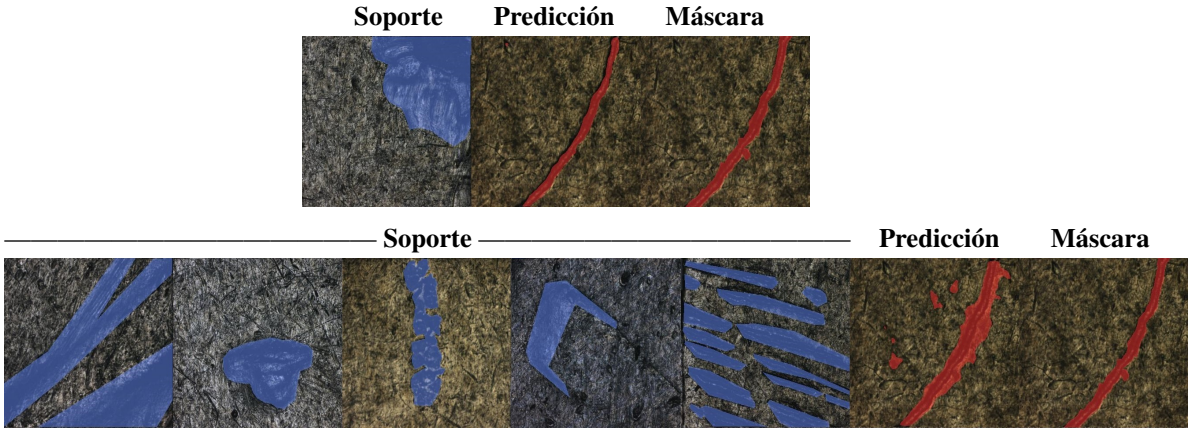


Figura 5.6: DCAMA: Ejemplo diferencia 1-shot vs 5-shot

Por último, hay que considerar el rendimiento en el despliegue que se consigue con esta arquitectura. Tanto con el conjunto de datos reducido como con el ampliado, el tiempo de entrenamiento tarda alrededor de 10 minutos. Es un buen tiempo de entrenamiento considerando los resultados obtenidos.

Por otro lado, como se ha mencionado en la Sección 4.5, el tiempo de predicción en producción en entornos reales es muy importante. Esta arquitectura con los componentes hardware utilizados tarda por cada predicción una media de 0.3 segundos con la versión 1-shot y 0.7 segundos con la versión 5-shot.

5.3. Redes de segmentación

En esta Sección se quiere obtener el mejor modelo entre las diferentes arquitecturas de segmentación semántica mencionadas previamente. Para ello, de la misma forma que se ha hecho en la sección anterior, primero hay que establecer los parámetros más relevantes de una red de segmentación para poder determinar el mejor modelo. Esta vez, la experimentación se ha llevado a cabo solo con en el conjunto de datos ampliado: 60 imágenes de entrenamiento, 20 de validación y 40 de test (de las cuales, 20 son instancias con defectos y las otras 20 sin defectos)

El primer paso ha sido escoger el mejor codificador *(backbone)* de entre todas las opciones disponibles. Existen muchos codificadores para cada arquitectura, además de los diferentes pesos pre-entrenados que ofrece cada codificador [1]. El criterio de selección de los codificadores para el estudio han sido tomar en cuenta los más relevantes que han demostrado buenos resultado en la tarea de detección de defectos: ResNet [50], EfficientNet [51], VGG [52] y Mix Vision Transformer [53]. Se ha probado con tres variante de ResNet con diferentes grados de complejidad. ResNet18 es más liviana y menos profunda, mientras que ResNet50 es más profunda y compleja. Por otro lado, ResNeXt50_32x4d es una variante de ResNet50 con una estructura interna diferente más compleja.

Sin embargo, el uso de diferentes arquitecturas para diferentes codificadores dificulta la comparación del rendimiento entre los codificadores. El motivo es que los resultados podrían ser influenciados no solo por las características extraídas por los codificadores, también por la arquitectura de la red de segmentación. Por lo tanto, las pruebas para la elección del mejor codificador se han hecho con una única arquitectura, para más en adelante probar con todas las arquitecturas. Hay que ser consciente que limitar la evaluación a una sola arquitectura puede afectar a la capacidad para encontrar la mejor solución.

[1] https://pypi.org/project/segmentation-models-pytorch/#encoders

La elección de esta arquitectura inicial debe de ser en base a cuál se cree que sería la mejor para esta tarea específica en función de la literatura. En el Sección 4.3.2 se ha detallado cada arquitectura dando motivos de la razón del porqué de la elección para la tarea de detección de defectos. En realidad, la elección de cualquiera sería justificada, puesto que hay evidencias suficientes de que pueda ser útil para esta tarea. Sin embargo, se ha decidido experimentar con la arquitectura Unet puesto que originalmente se diseñó para trabajar con imágenes a nivel microscópico, al igual que las imágenes capturadas en este proyecto. Además, esta arquitectura se caracteriza por su capacidad para realizar segmentación precisa en tareas donde se requiere una alta resolución y detalles finos.

Los resultados de los diferentes **codificadores** son las que se muestran en la Tabla 5.7 Además, la Figura 5.7 muestra un ejemplo para ver las diferencias:

Dataset test con solo defectos						
Backbone	**resnet18**	**resnet50**	**resnext50_32x4d**	**efficientnet-b1**	**vgg16**	**mit_b0**
IoU	0.727	0.86	**0.864**	0.828	0.799	0.735
Recall	0.749	0.898	**0.899**	0.846	0.813	0.749
Precision	0.973	0.955	0.959	0.977	**0.981**	0.98

Dataset test entero						
Backbone	**resnet18**	**resnet50**	**resnext50_32x4d**	**efficientnet-b1**	**vgg16**	**mit_b0**
IoU	0.863	0.928	**0.95**	0.895	0.871	0.858
Recall	0.874	0.947	**0.949**	0.905	0.878	0.866
Precision	0.986	0.977	0.979	0.988	**0.991**	0.99

Tabla 5.5: Segmentación semántica: Resultados codificadores.

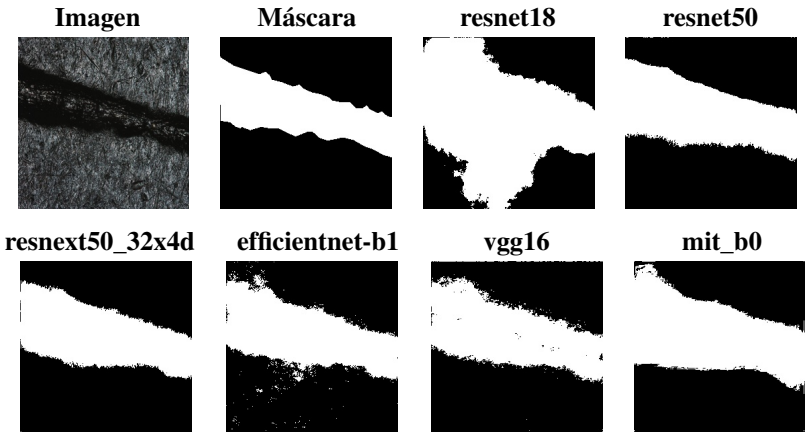


Figura 5.7: Segmentación semántica: Ejemplo codificadores

El *backbone* que mejor resultado ofrece y, por ende, el que se usará en las pruebas posteriores es **resnext50_32x4d**. Ofrece un IoU mayor que al resto y los valores de *Precision* y *Recall* son más equilibrados.

El siguiente paso será realizar la experimentación del *benchmark* definido previamente. El primer parámetro a comprobar es el **tamaño del *batch***. Los resultados obtenidos son los que se muestran en la Tabla 5.6:

Dataset test con solo defectos				
	1	**2**	**4**	**6**
IoU	0.862	**0.864**	0.708	0.828
Recall	0.891	**0.899**	0.723	0.821
Precision	0.972	0.959	**0.98**	0.972

Dataset test entero				
	1	**2**	**4**	**6**
IoU	0.928	**0.95**	0.842	0.877
Recall	0.939	**0.949**	0.85	0.887
Precision	0.983	0.979	**0.99**	0.986

Tabla 5.6: Segmentación semántica: Resultados tamaño de *batch*

Se puede apreciar que con tamaños de batch más pequeños se obtienen mejores métricas. Esto era previsible puesto que los tamaños de batch más pequeños pueden ser más beneficiosos en conjuntos de datos pequeños [66]. Se hará uso de tamaño de batch 2 para las siguientes pruebas.

A continuación, la Figura 5.8 muestra un ejemplo para ver las diferencias entre las pruebas:

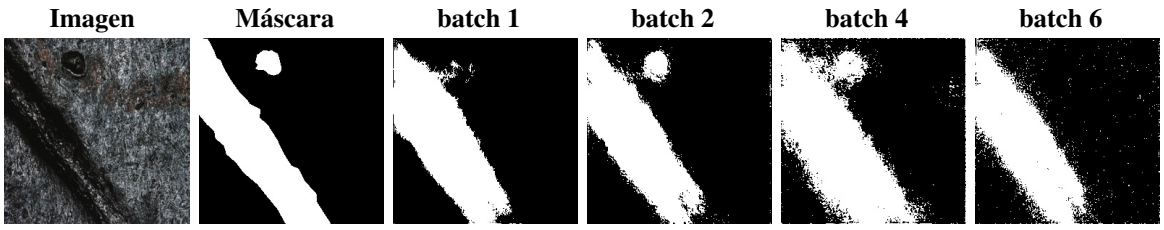


Figura 5.8: Segmentación semántica: Ejemplo tamaño de *batch*

La siguiente comprobación ha sido ver como afecta el **número de épocas**. La Tabla 5.7 muestra los resultados:

Dataset test con solo defectos			
	25	**50**	**100**
IoU	0.805	**0.864**	0.635
Recall	0.82	**0.899**	0.642
Precision	0.98	0.959	**0.988**

Dataset test entero			
	25	**50**	**100**
IoU	0.896	**0.95**	0.805
Recall	0.904	**0.949**	0.804
Precision	0.991	0.979	**0.994**

Tabla 5.7: Segmentación semántica: Resultados épocas

Con 50 épocas se obtienen los mejores valores en cuanto a métricas, el mayor valor de IoU y los valores más equilibrados de *Precision* y *Recall*. Con 25 épocas la red no llega a

converger completamente. En cambio, con 100 épocas supera el punto de convergencia y la red se sobreajusta (*overfitting*) dando significativamente peores resultados.

A continuación, la Figura 5.9 muestra un ejemplo para apreciar las diferencias:

Figura 5.9: Segmentación semántica: Ejemplo épocas

Por último, una vez que se ha optimizado la red seleccionado el *backbone* y los parámetros adecuados, el último paso ha sido comprobar el rendimiento de cada **arquitectura** de segmentación. La Tabla 5.8 muestra los resultados::

Dataset test con solo defectos						
	unet	**deepLabv3+**	**fpn**	**pan**	**pspnet**	**unet++**
IoU	0.864	0.838	0.881	0.892	0.884	**0.929**
Recall	0.899	0.894	0.944	0.955	**0.978**	0.957
Precision	0.959	0.931	0.929	0.937	0.929	**0.969**

Dataset test entero						
	unet	**deepLabv3+**	**fpn**	**pan**	**pspnet**	**unet++**
IoU	0.95	0.919	0.938	0.946	0.942	**0.964**
Recall	0.96	0.94	0.96	0.97	**0.988**	0.97
Precision	0.976	0.965	0.964	0.961	0.951	**0.984**

Tabla 5.8: Segmentación semántica: Resultados arquitecturas.

Se ve que la arquitectura Unet++ ofrece significativamente mejores resultados que el resto, especialmente en el conjunto con solo defectos. Otras arquitecturas muy utilizadas en la literatura donde se ha demostrado su eficacia en detección de defectos, como el caso de DeepLabv3+, ofrece resultados muy malos. Esto era de esperar puesto que estas arquitectura suelen ser entrenadas empleando conjuntos muy grandes de datos donde las redes aprenden mejor.

A continuación, la Figura 5.10 muestra algunos ejemplos para ver las diferencias entre las diferentes arquitecturas. Se verifica que, efectivamente, la arquitectura Unet++ es la que más se acerca a la máscara original:

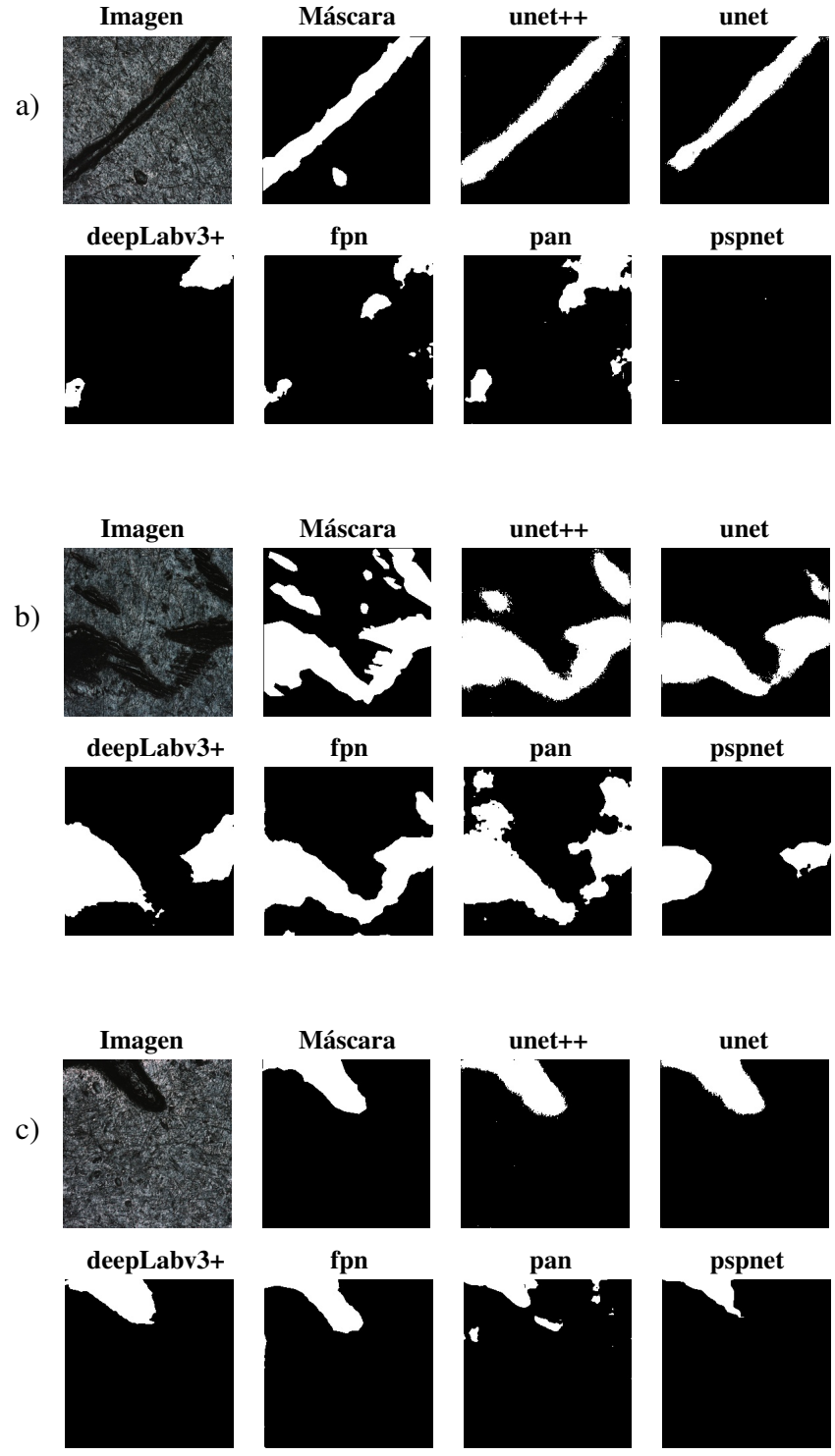


Figura 5.10: Segmentación semántica: Ejemplos diferencia entre arquitecturas

5.4. Comparación

En las dos secciones anteriores se ha llevado a cabo la experimentación de manera individual de la técnica de *Few-Shot Segmentation* DCAMA y las arquitecturas de segmentación convencionales para obtener el mejor modelo posible de cada uno. El objetivo de esta experimentación ha sido poder determinar cuál es el mejor para la tarea de detección de defectos con un conjunto de datos reducido. De este modo, en esta última sección se hará una breve recopilación y comparación entre las dos para poder apreciar de forma más sencilla las diferencias entre ambas.

Como resumen de la experimentación, se ha visto que la versión 1-shot es la que mejor funciona para la arquitectura DCAMA. En cuanto a las redes de segmentación, la arquitectura Unet++ ha mostrado significativamente mejor rendimiento que las otras. En la Tabla 5.9 se recoge los resultados de los dos mejores modelos:

Dataset test con solo defectos		
	DCAMA	**unet++**
IoU	**0.941**	0.929
Recall	0.88	**0.957**
Precision	0.86	**0.969**

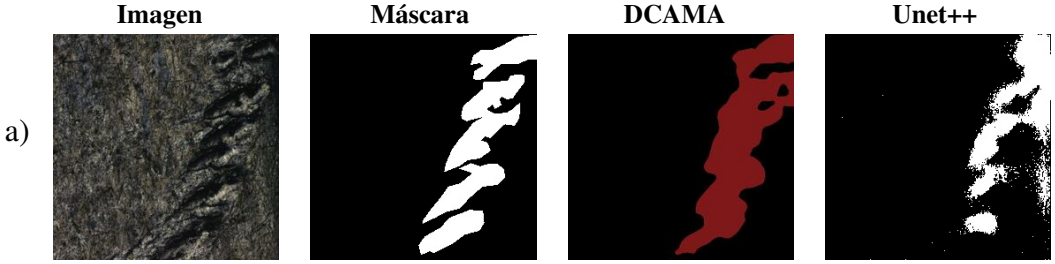
Dataset test entero		
	DCAMA	**unet++**
IoU	**0.978**	0.964
Recall	0.93	**0.97**
Precision	0.92	**0.984**

Tabla 5.9: DCAMA vs Red de Segmentación.

Estos resultados indican que DCAMA es mejor arquitectura para la tarea del presente proyecto.

Otro aspecto muy importante a considerar es el rendimiento en el despliegue obtenido por cada uno. DCAMA tarda una media de 0.2 segundos por cada predicción. En cambio, el modelo de Unet++ tarda una media de 0.4 segundos. Por otro lado, los dos modelos tardan alrededor de 10 minutos en entrenar, por lo que no hay diferencia en este aspecto.

Por ultimo, la Figura 5.11 muestra algunos ejemplos para ver las diferencias entre ambas, donde se puede ver que, efectivamente, la técnica DCAMA ofrece mejores resultados:



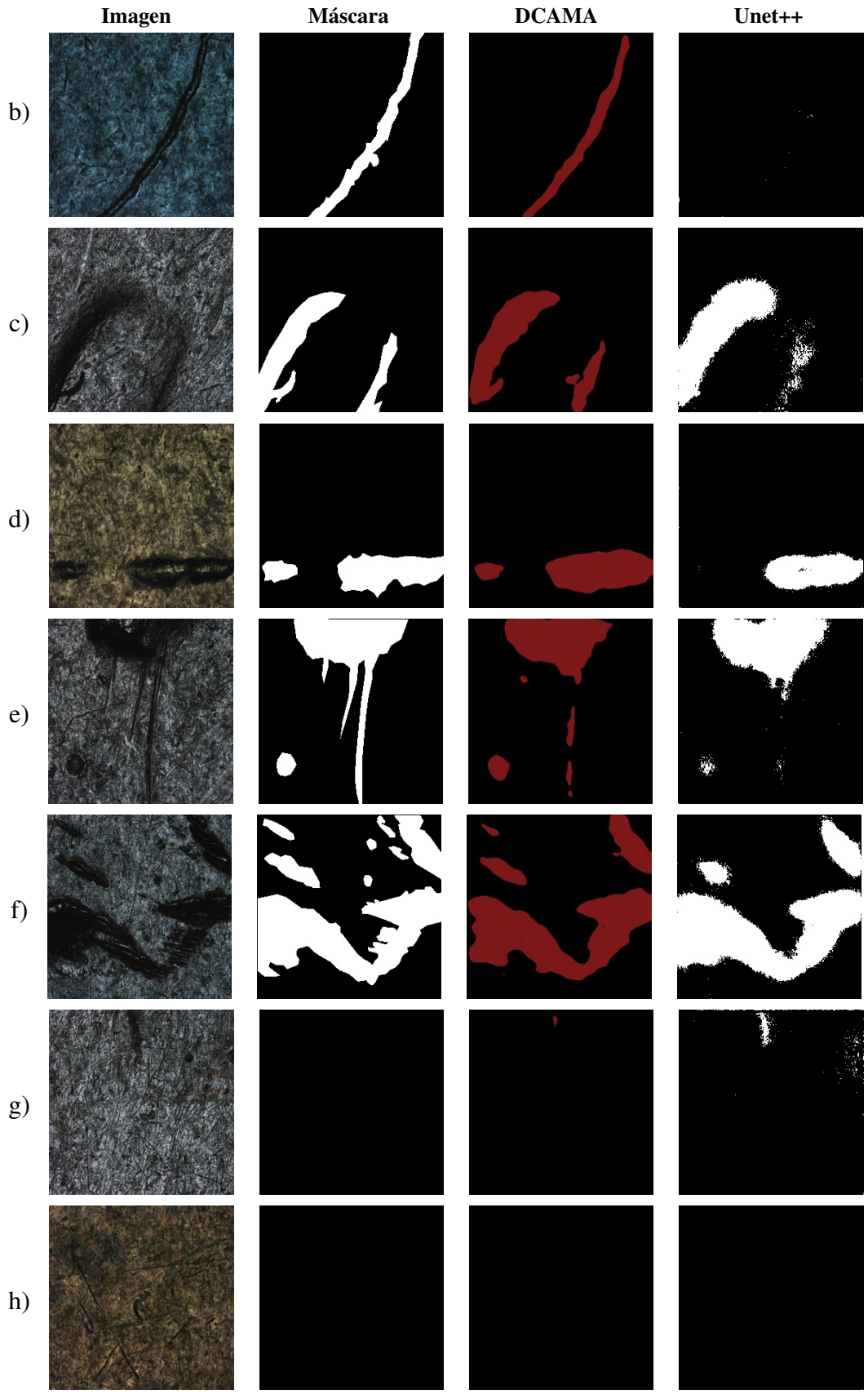


Figura 5.11: Ejemplos DCAMA vs mejor Red de Segmentación.

6. CAPÍTULO

Conclusiones

El objetivo general de este TFG ha sido realizar un estudio sobre técnicas avanzadas de Inteligencia Artificial aplicadas a imágenes en escenarios de control de calidad industrial con escasez y/o desbalanceo de datos entre clases. Se ha podido ver que los modelos de *Deep Learning* requieren de una gran cantidad de datos para garantizar que el rendimiento sea bueno. Sin embargo, en muchas ocasiones la adquisición de los datos en la industria es muy difícil debido a diversos factores. Por lo tanto, los métodos convencionales de control de calidad no ofrecen unos resultados adecuados y esto es un problema, puesto que en el control de calidad es absolutamente imprescindible que los resultados sean precisos.

De este modo, se ha realizado un estudio de los beneficios de una red de segmentación basada en *Few-Shot Learning* para detectar defectos en componentes industriales de una forma robusta cuando se dispone de un conjunto de datos pequeño. La finalidad de este estudio ha sido comprobar si este método de *Few-Shot Learning* es una alternativa en entornos de pocos datos a las arquitecturas convencionales de DL más frecuentemente utilizadas en la literatura para la detección de defectos.

Para ello, se ha llevado a cabo el proceso general que sigue el desarrollo de los sistemas de inspección automática: adquisición de los datos, generación de la base de datos, diseño y entrenamiento de las redes neuronales, validación y análisis de la viabilidad del despliegue en entornos productivos.

Se han realizado contribuciones a lo largo de todo este proceso. En un primer paso, se ha compuesto una base de datos capturando las superficies a analizar mediante el micros-

copio TAGLENS y se han anotado las imágenes para componer un conjunto de entrenamiento, validación y test. Se han realizado también experimentaciones con diferentes arquitecturas de segmentación de defectos y optimización de sus parámetros en el entrenamiento. Y por último se han validado los modelos mediante métricas de evaluación y métricas temporales.

Los resultados demuestran que el enfoque de *Few-Shot Learning* ofrece considerablemente mejores soluciones que las arquitecturas convencionales de DL para la tarea del presente trabajo. En concreto, con muy pocas imágenes de entrenamiento, la métrica de validación (IoU) se acerca al 98 %.

Además de este objetivo principal, se ha podido validar las hipótesis planteadas. Para empezar, se ha podido ver que los algoritmos de DL necesitan una base de datos de calidad, balanceada y bien anotada para un correcto funcionamiento. Esta base de datos tiene que ser amplia y cubrir en la medida de lo posible todos los escenarios en los que el sistema tenga que trabajar.

Además, se ha demostrado que el ajuste de los hiperparámetros y distintas configuraciones de las arquitecturas permite mejorar el rendimiento de los métodos empleados, obteniendo así unos resultados más precisos.

Por otro lado, se ha comprobado que el método DCAMA de *Few-Shot Learning* utilizado es capaz de generalizar con muy pocas muestras de entrenamiento. En concreto, se ha podido ver que con solo entre 10 y 30 imágenes de entrenamiento la arquitectura generaliza correctamente ofreciendo unos buenos resultados. Además, esta arquitectura es óptima en tiempo de inferencia y se ajusta a la cadencia de producción puesto que tarda una media de 0.2 segundos por cada predicción. En cuanto al tiempo de entrenamiento, la arquitectura tarda alrededor de 10 minutos en entrenar.

En definitiva, teniendo en cuenta todos estos motivos, se puede llegar a la conclusión de que los resultados son prometedores y que los objetivos se han cumplido satisfactoriamente.

Como líneas futuras de investigación, dentro del marco de la visión artificial y el control de calidad industrial, se propone examinar este método de *Few-Shot Learning* en otras aplicaciones industriales con entornos más cambiantes y/o con diferentes tipos de materiales. La adquisición en este trabajo se ha realizado a nivel microscópico para la tarea de detección de defecto, por lo que sería interesante analizar el comportamiento en un entorno diferente para poder fortalecer la evidencia de que, efectivamente, es una alternativa en entornos de pocos datos.

Bibliografía

[1] Amitava Mitra. *Fundamentals of quality control and improvement*. John Wiley & Sons, 2016.

[2] Ricardo Silva Peres, Xiaodong Jia, Jay Lee, Keyi Sun, Armando Walter Colombo, and Jose Barata. Industrial artificial intelligence in industry 4.0-systematic review, challenges and outlook. *IEEE Access*, 8:220121–220139, 2020.

[3] Jorge F Arinez, Qing Chang, Robert X Gao, Chengying Xu, and Jianjing Zhang. Artificial intelligence in advanced manufacturing: Current status and future outlook. *Journal of Manufacturing Science and Engineering*, 142(11), 2020.

[4] Longfei Zhou, Lin Zhang, and Nicholas Konz. Computer vision techniques in manufacturing. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2022.

[5] Ajay Agrawal, Joshua S Gans, and Avi Goldfarb. Artificial intelligence: the ambiguous labor market impact of automating prediction. *Journal of Economic Perspectives*, 33(2):31–50, 2019.

[6] B Bajic, I Cosic, M Lazarevic, N Sremcev, and A Rikalovic. Machine learning techniques for smart manufacturing: Applications and challenges in industry 4.0. *Department of Industrial Engineering and Management Novi Sad, Serbia*, 29, 2018.

[7] Neha Sharma, Reecha Sharma, and Neeru Jindal. Machine learning and deep learning applications-a vision. *Global Transitions Proceedings*, 2(1):24–28, 2021.

[8] Fátima Aurora Sáiz Álvaro. Artificial intelligence for advanced manufacturing quality. 2023.

[9] Jayme Garcia Arnal Barbedo. Impact of dataset size and variety on the effectiveness of deep learning and transfer learning for plant disease classification. *Computers and electronics in agriculture*, 153:46–53, 2018.

[10] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[11] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

[12] Fabio Henrique Kiyoiti dos Santos Tanaka and Claus Aranha. Data augmentation using gans. *arXiv preprint arXiv:1904.09135*, 2019.

[13] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)*, 53(3):1–34, 2020.

[14] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[15] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018.

[16] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[17] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8759–8768, 2018.

[18] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.

[19] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++: A nested u-net architecture for medical image segmentation. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical*

Decision Support: 4th International Workshop, DLMIA 2018, and 8th International Workshop, ML-CDS 2018, Held in Conjunction with MICCAI 2018, Granada, Spain, September 20, 2018, Proceedings 4, pages 3–11. Springer, 2018.

[20] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[21] I-Chun Chen, Rey-Chue Hwang, and Huang-Chu Huang. Pcb defect detection based on deep learning algorithm. *Processes*, 11(3):775, 2023.

[22] Nirbhar Neogi, Dusmanta K Mohanta, and Pranab K Dutta. Review of vision-based steel surface inspection systems. *EURASIP Journal on Image and Video Processing*, 2014(1):1–19, 2014.

[23] Fátima A Saiz, Iñigo Barandiaran, Ander Arbelaiz, and Manuel Graña. Photometric stereo-based defect detection system for steel components manufacturing using a deep segmentation network. *Sensors*, 22(3):882, 2022.

[24] Atiya Khan, Amol D Vibhute, Shankar Mali, and CH Patil. A systematic review on hyperspectral imaging technology with a machine and deep learning methodology for agricultural applications. *Ecological Informatics*, page 101678, 2022.

[25] Bo Tang, Li Chen, Wei Sun, and Zhong-kang Lin. Review of surface defect detection of steel products based on machine vision. *IET Image Processing*, 17(2):303–322, 2023.

[26] Kriti Aggarwal, Sunil K Singh, Muskaan Chopra, Sudhakar Kumar, and Francesco Colace. Deep learning in robotics for strengthening industry 4.0.: opportunities, challenges and future directions. *Robotics and AI for Cybersecurity and Critical Infrastructure in Smart Cities*, pages 1–19, 2022.

[27] Yinglong Li. Research and application of deep learning in image recognition. In *2022 IEEE 2nd International Conference on Power, Electronics and Computer Applications (ICPECA)*, pages 994–999. IEEE, 2022.

[28] Syed Sahil Abbas Zaidi, Mohammad Samar Ansari, Asra Aslam, Nadia Kanwal, Mamoona Asghar, and Brian Lee. A survey of modern deep learning based object detection models. *Digital Signal Processing*, page 103514, 2022.

[29] Muriel Mazzetto, Marcelo Teixeira, Érick Oliveira Rodrigues, and Dalcimar Casanova. Deep learning models for visual inspection on automotive assembling line. *arXiv preprint arXiv:2007.01857*, 2020.

[30] Omar Elharrouss, Younes Akbari, Noor Almaadeed, and Somaya Al-Maadeed. Backbones-review: Feature extraction networks for deep learning and deep reinforcement learning approaches. *arXiv preprint arXiv:2206.08016*, 2022.

[31] Rahul Rai, Manoj Kumar Tiwari, Dmitry Ivanov, and Alexandre Dolgui. Machine learning in manufacturing and industry 4.0 applications. *International Journal of Production Research*, 59(16):4773–4778, 2021.

[32] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[33] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[34] Hafiz Mughees Ahmad and Afshin Rahimi. Deep learning methods for object detection in smart manufacturing: A survey. *Journal of Manufacturing Systems*, 64:181–196, 2022.

[35] Jyunrong Wang, Huafeng Dai, Taogen Chen, Hao Liu, Xuegang Zhang, Quan Zhong, and Rongsheng Lu. Toward surface defect detection in electronics manufacturing by an accurate and lightweight yolo-style object detector. *Scientific Reports*, 13(1):7062, 2023.

[36] Jie Liu, Jianlin Guo, Philip Orlik, Masahiko Shibata, Daiki Nakahara, Satoshi Mii, and Martin Takáč. Anomaly detection in manufacturing systems using structured neural networks. In *2018 13th world congress on intelligent control and automation (wcica)*, pages 175–180. IEEE, 2018.

[37] Jiaqi Liu, Guoyang Xie, Jingbao Wang, Shangnian Li, Chengjie Wang, Feng Zheng, and Yaochu Jin. Deep visual anomaly detection in industrial manufacturing: A survey. *arXiv preprint arXiv:2301.11514*, 2023.

[38] Swarnendu Ghosh, Nibaran Das, Ishita Das, and Ujjwal Maulik. Understanding deep learning techniques for image segmentation. *ACM Computing Surveys (CSUR)*, 52(4):1–35, 2019.

[39] Max K Ferguson, AK Ronay, Yung-Tsun Tina Lee, and Kincho H Law. Detection and segmentation of manufacturing defects with convolutional neural networks and transfer learning. *Smart and sustainable manufacturing systems*, 2, 2018.

[40] Manan Mehta and Chenhui Shao. Federated learning-based semantic segmentation for pixel-wise defect detection in additive manufacturing. *Journal of Manufacturing Systems*, 64:197–210, 2022.

[41] Rongge Xu, Ruiyang Hao, and Biqing Huang. Efficient surface defect detection using self-supervised learning strategy and segmentation network. *Advanced Engineering Informatics*, 52:101566, 2022.

[42] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3523–3542, 2021.

[43] Imran Qureshi, Junhua Yan, Qaisar Abbas, Kashif Shaheed, Awais Bin Riaz, Abdul Wahid, Muhammad Waseem Jan Khan, and Piotr Szczuko. Medical image segmentation using deep semantic-based methods: A review of techniques, applications and emerging trends. *Information Fusion*, 2022.

[44] Alok Jhaldiyal and Navendu Chaudhary. Semantic segmentation of 3d lidar data using deep learning: a review of projection-based methods. *Applied Intelligence*, 53(6):6844–6855, 2023.

[45] Li-Hua Wen and Kang-Hyun Jo. Deep learning-based perception systems for autonomous driving: A comprehensive survey. *Neurocomputing*, 2022.

[46] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[47] Nameirakpam Dhanachandra, Khumanthem Manglem, and Yambem Jina Chanu. Image segmentation using k-means clustering algorithm and subtractive clustering algorithm. *Procedia Computer Science*, 54:764–771, 2015.

[48] Laurent Najman and Michel Schmitt. Watershed of a continuous function. *Signal Processing*, 38(1):99–112, 1994.

[49] Javaria Amin, Muhammad Almas Anjum, and Muhammad Malik. Fused information of deeplabv3+ and transfer learning model for semantic segmentation and rich features selection using equilibrium optimizer (eo) for classification of npdr lesions. *Knowledge-Based Systems*, 249:108881, 2022.

[50] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[51] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

[52] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[53] Hongyi Wang, Shiao Xie, Lanfen Lin, Yutaro Iwamoto, Xian-Hua Han, Yen-Wei Chen, and Ruofeng Tong. Mixed transformer u-net for medical image segmentation, 2021.

[54] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[55] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111:98–136, 2015.

[56] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[57] Joao Fonseca and Fernando Bacao. Research trends and applications of data augmentation algorithms. *arXiv preprint arXiv:2207.08817*, 2022.

[58] Archit Parnami and Minwoo Lee. Learning from few examples: A summary of approaches to few-shot learning. *arXiv preprint arXiv:2203.04291*, 2022.

[59] Xinyu Shi, Dong Wei, Yu Zhang, Donghuan Lu, Munan Ning, Jiashun Chen, Kai Ma, and Yefeng Zheng. Dense cross-query-and-support attention weighted mask

aggregation for few-shot segmentation. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XX*, pages 151–168. Springer, 2022.

[60] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[61] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[62] Jonathan Hui. Understanding feature pyramid networks for object detection (fpn), Apr 2020.

[63] Guang Huo, Dawei Lin, and Meng Yuan. Iris segmentation method based on improved unet++. *Multimedia Tools and Applications*, 81(28):41249–41269, 2022.

[64] Seyed Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian D. Reid, and Silvio Savarese. Generalized intersection over union: A metric and A loss for bounding box regression. *CoRR*, abs/1902.09630, 2019.

[65] Pavlo M Radiuk. Impact of training set batch size on the performance of convolutional neural networks for diverse datasets. *Information Technology and Management Science*, 20(1):20–24, 2017.

[66] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.