Universidad de Oviedo

Máster en Ingeniería de Automatización e Informática Industrial

MAIIND

Diseño e implementación de técnicas de Machine Learning para la detección de defectos superficiales en piezas sometidas a procesos de estampado o fundición

Autor:	*Tutor:*	*Co-Tutor:*
Daniel García Peña	Ignacio Díaz Blanco	Diego García Pérez
uo250234@uniovi.es	idiaz@uniovi.es	garciaperdiego@uniovi.es

Universidad de Oviedo

9 de febrero de 2021

Índice

Índice de figuras

Índice de cuadros

1. Introducción

Desde los comienzos de esta década, la industria ha comenzado una carrera para aprovechar las nuevas tecnologías de la información con el fin de optimizar su funcionamiento, dando lugar a la cuarta revolución industrial bajo el nombre de **Industria 4.0**. Esta revolución está basada en los denominados habilitadores digitales (Internet de las cosas, *Cloud computing*, *Big Data*, Inteligencia Artificial, etc.), sin los cuales la digitalización de la industria no sería posible.
El Machine Learning (ML), como una disciplina del campo de la Inteligencia Artificial (IA), se encuentra en pleno auge, gracias sobre todo al aumento de la capacidad de computación y al aumento exponencial de los datos recogidos en una gran variedad de ámbitos. La complejidad y rapidez en la que se genera la información, tal y como se recoge en la regla de las tres v's (velocidad, volumen y variedad) que define el concepto de Big Data, ha propiciado el desarrollo de tecnologías como el ML capaces de encontrar patrones en datos masivos, de tal forma que estos son capaces de predecir comportamientos futuros a partir de una información suministrada. Este aprendizaje permite a los computadores realizar tareas específicas de forma autónoma, es decir, sin necesidad de ser programados específicamente para ello. Aunque se trata de una tecnología relativamente nueva, ya se han visto diversas aplicaciones en el entorno industrial:

- ML aplicado en producción, fabricación y calidad.
- ML aplicado en logística.
- ML aplicado en mantenimiento predictivo.
- ML aplicado en seguridad.
- ML aplicado en Recursos Humanos.

Siguiendo esta tendencia, en la empresa *Desarrollo Soluciones Integrales plus* (DSIplus) [1] surge la necesidad de mejorar sus técnicas de detección de defectos superficiales, hasta ahora realizadas con técnicas clásicas de visión artificial, mediante algoritmos de *Machine Learning*.

La estructura del proyecto será la siguiente:

- En la sección 2 se expondrán las diferentes técnicas y herramientas ne-

cesarias para realizar el proyecto, desde la extracción de características, hasta la evaluación de los modelos generados.

- En la sección 3 se expondrán los resultados obtenidos tras la aplicación de las distintas técnicas.
- En la sección 4 se discutirán los resultados y se propondrán futuras líneas de trabajo.

1.1. Antecedentes y estado del arte

Hoy en día los controles de calidad en la industria de la manufactura son cada vez más rigurosos e importantes, demandando un control del 100 % de la producción, tanto en las características dimensionales como en la detección de existencia de defectos, tanto superficiales como internos de los productos. Ante esta tesitura, con el fin de mejorar los procesos productivos y disminuir los costes de fabricación, es necesario el desarrollo de nuevos sistemas capaces de llevar a cabo el control de calidad con un alto grado de precisión y sin comprometer los tiempos de producción.
Hasta hace unos años los controles de calidad se realizaban de forma estadística mediante la toma de muestras aleatorias de la producción y llevando a cabo sobre ellas las correspondientes mediciones e inspecciones, ya que la inspección de la totalidad de la producción era una tarea imposible de realizar sin que se influyese en la velocidad del proceso productivo.
Este problema sigue presente, principalmente, en los métodos de inspección que requieren un contacto entre el elemento medidor y la muestra que está siendo estudiada, puesto que la medida ha de realizarse de forma relativamente lenta para evitar colisiones que lo deterioren, además de que, para realizar mediciones con gran precisión, es necesario que no existan vibraciones, cosa que en un proceso real es difícil de conseguir. Con la aparición de los métodos de medida sin contacto, como los ultrasonidos, la visión artificial o algunas técnicas interferométricas, las limitaciones de velocidad de medición han desaparecido con lo que se puede realizar una inspección completa de la producción.
Estos sistemas de medición sin contacto aún se encuentran en plena fase de investigación y desarrollo en algunos campos de aplicación para conseguir un sistema adecuado y que cumpla con todas las especificaciones que requiere la industria para su correcta implementación y uso en los procesos industriales.

En este contexto, DSIplus ha centrado sus esfuerzos en el desarrollo de sistemas de de medición sin contacto basados en la tecnología de visión artificial Holografía Conoscópica [2] en configuración de perfilómetro, que permite solventar los inconvenientes y limitaciones de la tecnología más usada hasta el momento (la triangulación láser).
Pese a que estos sistemas de visión artificial son capaces de detectar con gran precisión posibles defectos en la superficie de la pieza, resulta complicado establecer manualmente una serie de umbrales para diferenciar los defectos reales de los denominados falsos positivos (FP).
El Machine Learning se presenta como la tecnología adecuada para complementar el procesamiento de visión artificial. Aunque esta tecnología ya se encuentra presente en aspectos de nuestra vida cotidiana con aplicaciones como el reconocimiento de voz y facial [3] [4], el reconocimiento y procesamiento de texto [5] o incluso en la conducción de vehículos autónomos [6] mediante el reconocimiento de imágenes, se ha constatado que esta tecnología todavía se encuentra en las etapas iniciales de su aplicación en procesos de fabricación avanzada y visión artificial.
Los algoritmos de Machine Learning que se aplican para realizar el reconocimiento de imágenes son capaces de encontrar patrones particulares, clasificarlos y agruparlos por zonas [7]. Esto ofrece un abanico de posibilidades para la detección de defectos en piezas industriales. De esta forma, disponer de un software de análisis de imagen basado en Machine Learning ofrece una herramienta muy útil para solucionar retos de visión complejos en los que estén implicadas piezas de geometrías complejas o requisitos de control de calidad muy estrictos, ya que, entre otras cosas, permite distinguir defectos difíciles de detectar con otras técnicas de análisis, ya que esta aporta detalles como la textura o la rugosidad de la superficie, difíciles de extraer de una captura en 2D de la pieza.
Ante este contexto, han comenzado a surgir en el mercado algunas soluciones software de análisis de imágenes industriales basados en Machine Learning [8]. Sin embargo, a pesar de que ya existen algunas soluciones de este tipo, estas no están aplicadas a la detección de defectos en piezas con una geometría compleja. Además, estas otras aplicaciones ya existentes solo son capaces de procesar imágenes 2D para detectar defectos superficiales. Por el contrario, DSIplus integra información 3D de la pieza, lo que proporciona una ventaja en el análisis de las piezas mencionadas.

1.2. Contexto del proyecto

Hoy en día, la fundición es uno de los procesos claves dentro de la industria, suministrando piezas en sectores como la automoción, aeronáutica o el armamentístico. Estas piezas suelen jugar un papel clave en el funcionamiento del producto final, por lo que el más mínimo defecto puede ser fatal. Por este motivo, estas piezas se ven sometidas a procesos de calidad exhaustivos.
Existen varios tipos de defectos que pueden aparecer en piezas de fundición, pero este proyecto se centrará en los denominados defectos superficiales. En la figura 1 se pueden ver algunos de los defectos superficiales más comunes, entre los que se encuentran los rechupes, las sopladuras, las gotas frías, las zonas sin material, las rebabas y las piezas rotas. Algunos de estos defectos se muestran en la Figura 1.
Desgraciadamente, estos procesos de calidad aún se realizan en su mayoría de forma manual, siendo un operario el que revisa la calidad de la pieza, permitiendo que factores como el cansancio o el estado emocional afecten al proceso de toma de decisión. Por este motivo, la visión artificial se plantea como el candidato perfecto para resolver esta tarea.
A pesar de la alta precisión aportada por la sensórica de inspección dimensional, basada fundamentalmente en tecnologías como la triangulación láser o la holografía conoscópica, establecer manualmente mecanismos deterministas para la detección de muestras anómalas es una tarea compleja y costosa en escenarios de alta variabilidad, donde la discriminación entre defecto y geometrías intrínsecas a la pieza no es trivial. Asimismo, el propio proceso de medida introduce incertidumbre en los datos obtenidos (ruidos propios del sensor, diferentes colocaciones, condiciones de contorno variables, etc.), reafirmando la necesidad de técnicas capaces de encontrar patrones complejos en datos multidimensionales y establecer relaciones invariantes a las condiciones de contorno entre estos patrones y la existencia de defecto. Las técnicas ML son capaces de aprender estas relaciones mediante un aprendizaje basado en ejemplos, habiendo demostrado sobradamente su valía en la detección de anomalías en ámbitos de aplicación similares, tales como la detección de consumos anómalos en grandes edificios [9], detección de fallos en máquinas rotatorias [10] o la monitorización de salud estructural de grandes infraestructuras [11].
Este proyecto se centrará en la detección de defectos superficiales en pinzas de frenos, también denominadas *calipers*. Como se ve en la Figura 2, esta

Figura 1: Ejemplos de defectos presentes en las piezas de fundición.

pieza destaca por tener una geometría compleja, por lo que son necesarios 8 sensores de triangulación para poder analizarla en su totalidad.

Las pinzas avanzan por una cinta transportadora que permite obtener 8 imágenes 3D de cada pinza (ver Figura 5), capturando por completo todas las superficies de la misma. Pese a que la combinación de estas imágenes podría generar la nube de puntos 3D de la pieza entera, para el desarrollo de este proyecto se analiza cada imagen por separado, de forma que hay zonas de la pieza que se analizan desde distintos ángulos, lo que ofrece mayor se-

Figura 2: Pieza de fundición a estudiar.

guridad en las decisiones.
La información capturada por este sistema será el punto de partida para el desarrollo de los modelos de ML por lo que se mantendrá la misma división en lo que resta de trabajo.
Estos sensores están situados en diferentes posiciones con el objetivo de evitar oclusiones y capturar la totalidad de la superficie de la pieza, por lo que se encuentran a diferentes distancias de la misma y trabajan en rangos de valores distintos, agrupados por pares de sensores simétricos, de forma que un sensor y su simétrico recogen una distribución de distancias similar.
Mediante los mecanismos deterministas mencionados previamente, es posible realizar una primera detección de posibles defectos. Debido a la naturaleza del problema, explicada en la sección 1.1, se define la función a optimizar durante el proyecto como:

$$\text{mín}\, FP\, t.q.\, FN < \epsilon \tag{1}$$

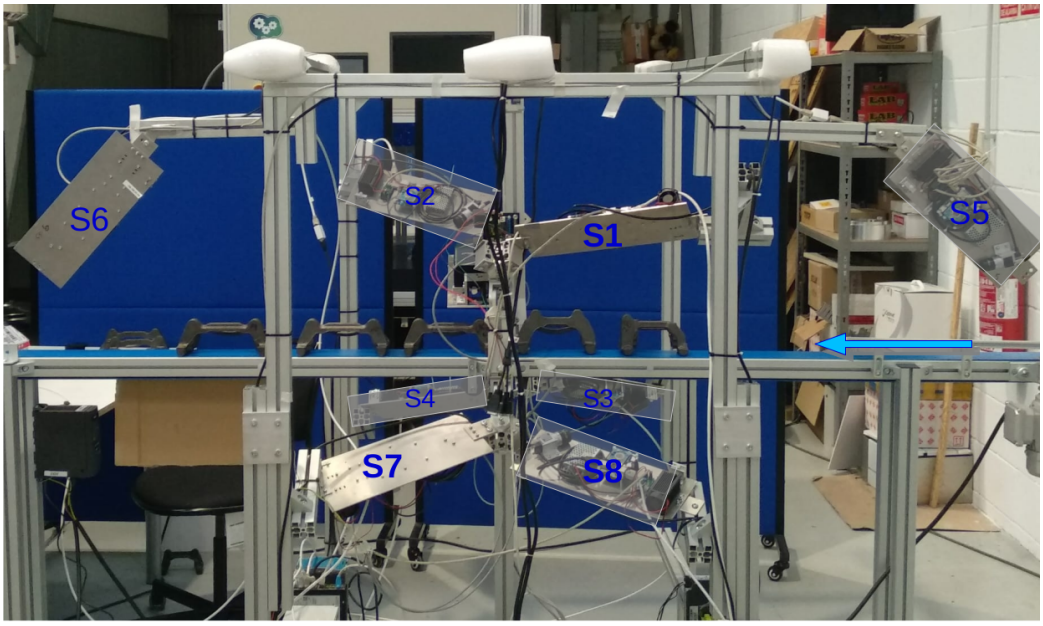


Figura 3: Disposición de los sensores de medición.

donde FP hace referencia a "Falsos Positivos"(puntos normales clasificados como defectos), FN hace referencia a "Falsos Negativos"(puntos de defecto clasificados como puntos normales) y ϵ es el valor de tolerancia de falsos negativos aceptado por el cliente. El objetivo del proyecto será mejorar los resultados obtenidos con técnicas clásicas de visión artificial, mediante algoritmos de ML.

Aunque el presente trabajo se centra en mejorar el procesamiento actual

Métrica	**Porcentaje**
Falsos positivos	50 %
Detección de defectos	90 %

Cuadro 1: Métricas de partida.

en el proyecto mencionado previamente, se considera que puede ser interesante evaluar la aplicabilidad de las técnicas desarrolladas en la sección 2 en diferentes proyectos de la empresa. Los proyectos escogidos son los siguientes:

- *Calipers*. Al igual que las piezas del estudio original, esta pieza es anali-

zada tras un proceso de fundición. Tanto el proceso productivo como la complejidad geométrica de la pieza son similares a los del proyecto original, aunque el tamaño y la forma cambian. Debido a que se trata de un estudio de la viabilidad de la tecnología, la infraestructura disponible para el escaneo de las piezas es limitada, por lo que son escaneadas mediante un sensor situado sobre un brazo robotizado.

- *Puertas*. Estas piezas son el resultado de un proceso de estampado. Al contrario que las piezas del estudio principal, la geometría es más parecida a un plano, por lo que no hay similitud en el proceso productivo ni en la geometría de la pieza.

1.3. Planteamiento del problema

Si consideramos como datos de partida una imagen de distancias a uno de los 8 sensores $\mathbf{X} \in \mathbb{R}^{l \times n}$, se plantea calcular sobre esta imagen una extracción de m características

$$\mathcal{F}(\mathbf{X}^{(i)}) = \{\mathbf{F}_1^{(i)}, \mathbf{F}_2^{(i)}, \ldots, \mathbf{F}_m^{(i)}\} \tag{2}$$

donde las imágenes $\mathbf{F}_m^{(i)} \in \mathbb{R}^{l \times n}$ de la misma dimensión que la entrada son el resultado de la operación de extracción de características $\mathcal{F}(.)$ sobre la i-ésima imagen del conjunto de entrenamiento. Teniendo esto en cuenta, el objetivo de este proyecto es encontrar una función $C(.)$ que, partiendo de las características extraídas, estime la probabilidad de defecto.

$$C(\mathcal{F}(\mathbf{X}^{(i)})) = p(y = 1|\mathbf{X}^{(i)}) \tag{3}$$

En una primera aproximación, $C(.)$ fue diseñada como un conjunto de reglas fijas sobre $\mathbf{F}_m^{(i)}$ seguidas de un umbralizado final para obtener una imagen binaria. Esta imagen final destaca las áreas sospechosas de fallo, denominadas de ahora en adelante como 'manchas' (blobs). Este enfoque heurístico no captura con precisión los patrones que discriminan 'fallo' de 'normal' en las características, marcando como mancha partes de las piezas que corresponden a geometrías complejas pero que no son defectos (falsos positivos correspondientes a rebabas, números de serie, manillas de las puertas, etc.). Este procedimiento se ejecuta en paralelo para cada uno de los 8 sensores

con el objetivo de detectar los posibles defectos antes de procesar la siguiente pieza. El tiempo que se dispone para procesar la pieza se denomina *tiempo de ciclo*.
En este trabajo se propone como enfoque alternativo obtener $C(.)$ por métodos de *machine learning* (ML) capaces de aprender, mediante ejemplos, relaciones no lineales en el espacio de características, llegando a encontrar patrones complejos que discriminan las partes de las piezas con mayor precisión que las reglas heurísticas, evitando el problema de los falsos positivos.
A la hora de realizar la extracción de características se plantearon dos métodos distintos: extracción a nivel de píxel y extracción a nivel de mancha.

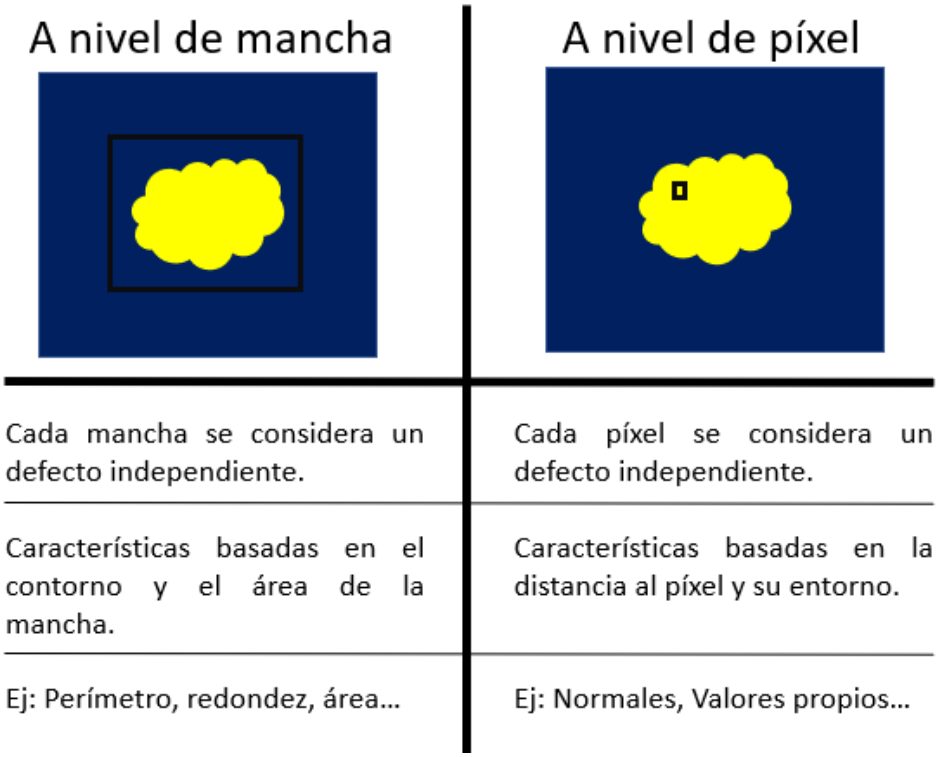


Figura 4: Diferencias entre los tipos de extracción de características propuestos.

Finalmente se decide realizar la extracción a nivel de píxel por los siguientes motivos:

- Debido a la geometría de las pinzas, un defecto puede situarse en más de un plano al mismo tiempo y que éste sea capturado desde distintos ángulos en cada una de las capturas, haciendo que carezca de sentido el cálculo de características sobre los contornos de las manchas detectadas en la clasificación heurística.
- El número de ejemplos disponibles para el entrenamiento al clasificar por píxel aumenta considerablemente previniendo el sobreentrenamiento [12] de los modelos ML.

De este modo se parte de un clasificador

$$C(\mathbf{f}_{l,n}) = p(y_{l,n} = 1|x_{l,n}) \tag{4}$$

donde a partir de un vector de características $\mathbf{f}_{l,n} = [f_{l,n}^{(1)}, f_{l,n}^{(2)}, \ldots, f_{l,n}^{(m)}]$ calculados sobre el punto $x_{l,n}$ de la imagen de partida $\mathbf{X}$ se estima la probabilidad de defecto $p(y_{l,n} = 1|x_{l,n})$ para ese punto.

1.4. Objetivos

El objetivo principal del presente proyecto es el diseño y desarrollo de algoritmos que permitan aplicar la tecnología de Machine Learning o Aprendizaje Automático a la detección de defectos en piezas de producción industrial, para mejorar la eficiencia en la detección y clasificación de estos defectos.
DSIplus trabaja con clientes cuyos requisitos de control de calidad son muy elevados por tratarse, en gran medida, de empresas del sector de la automoción, y cuyas piezas a inspeccionar son frecuentemente complejas y con geometrías difíciles. Mediante este proyecto se llevará a cabo el diseño y desarrollo de algoritmos de Machine Learning para mejorar la fiabilidad de sus sistemas de detección de defectos en esas piezas. El objetivo final es, por tanto, la mejora de la clasificación de defectos en piezas complejas y la generalización de su uso con la intención de disminuir los tiempos de ajuste o entrenamiento a nuevas piezas con geometrías distintas pero de misma naturaleza que se incorporen a los procesos productivos de sus clientes.

Del objetivo principal de proyecto se derivan una serie de objetivos tecnológicos específicos que se desglosan a continuación:

- **Extracción y curado de los datos**. En esta fase se recopilarán y procesarán los datos de entrenamiento. El etiquetado manual de los datos se realizará mediante un software propio, desarrollado por DSIplus, mientras que el procesamiento de los datos se realizará mediante el lenguaje de programación *Python* [13], con librerías como *Numpy* [14] o *Pandas* [15].
- **Computo de características o descriptores relevantes**. A partir de las imágenes devueltas por los sensores, se procesará un conjunto de descriptores por muestra, capaces de sintetizar la información relevante para la detección.
- **Desarrollo y aplicación de algoritmos de detección de anomalías**. Se utilizarán como punto de partida técnicas ML del estado del arte en detección de anomalías a nuestro problema, las cuales podrán ser tanto no supervidadas (one-class classifiers, deep auto-encoders, etc.), como supervisadas (SVC, árboles de decisión, redes neuronales profundas, etc). Estos algoritmos se desarrollarán en *Python*, utilizando las librerías *scikit-learn* [16], *Tensorflow* [17] y *Keras* [18].
- **Evaluación**. Las técnicas propuestas serán evaluadas con datos de test ajenos al entrenamiento, analizando su rendimiento mediante métricas específicas de clasificación (recall, precisión, F1-score, etc.). Además, las soluciones aportadas también pueden ser evaluadas en otros términos relacionados con su despliegue en planta, como por ejemplo el tiempo en obtener una predicción (complejidad de los algoritmos) o la transferibilidad de la solución a equipos industriales.

Por lo tanto, el resultado de este proyecto será un sistema que, fusionado con la visión artificial convencional, mejorará drásticamente el desempeño en la clasificación de defectos en los productos de DSIplus, y que consecuentemente, tendrá un impacto elevado en la productividad de los procesos productivos de sus clientes.

2. Métodos y técnicas

2.1. Adquisición del dataset

Con el objetivo de facilitar la extracción posterior de las características, es necesario definir un método de adquisición y etiquetado de los datos que sirva como guía para el desarrollo de futuros proyectos. Los métodos explicados a lo largo de esta sección son propiedad de DSIplus y, aunque el proyectante ha participado activamente, tanto en el escaneo, como en el etiquetado de los datos, no son el objeto principal del presente proyecto.

2.1.1. Escaneo de las piezas

La adquisición de los datos se realiza mediante una técnica de escaneo 3D denominada holografía conoscópica [2]. La holografía conoscópica es una técnica interferométrica por la que un haz reflejado en una superficie atraviesa un cristal birrefringente, esto es, un cristal que posee dos índices de refracción, uno ordinario y fijo y otro extraordinadio que es función del ángulo de incidencia del rayo en la superficie del cristal.
Como resultado de atravesar el cristal se obtienen dos rayos paralelos que se hacen interferir utilizando para ello una lente cilíndrica, esta interferencia es capturada por el sensor de una cámara convencional obteniendo un patrón de franjas. La frecuencia de esta interferencia determina la distancia del objeto en el que se proyectó el haz. Esta técnica permite la medición de orificios en su configuración colineal, alcanzando precisiones mejores que una micra. La ventaja de esta técnica es que permite utilizar luz no coherente, esto quiere decir que la fuente de iluminación no tiene porqué ser un láser, la única condición es que sea monocromática.
A partir de estas franjas se genera una imagen de distancias que será el punto de partida del presente proyecto.

2.1.2. Etiquetado de los datos

Para modelar un clasificador (4), cada conjunto de características $\mathbf{f}_{l,n} = [f_{l,n}^{(1)}, f_{l,n}^{(2)}, \ldots, f_{l,n}^{(m)}]$ debe estar asociado a una etiqueta $y_{l,n}$. El proceso de

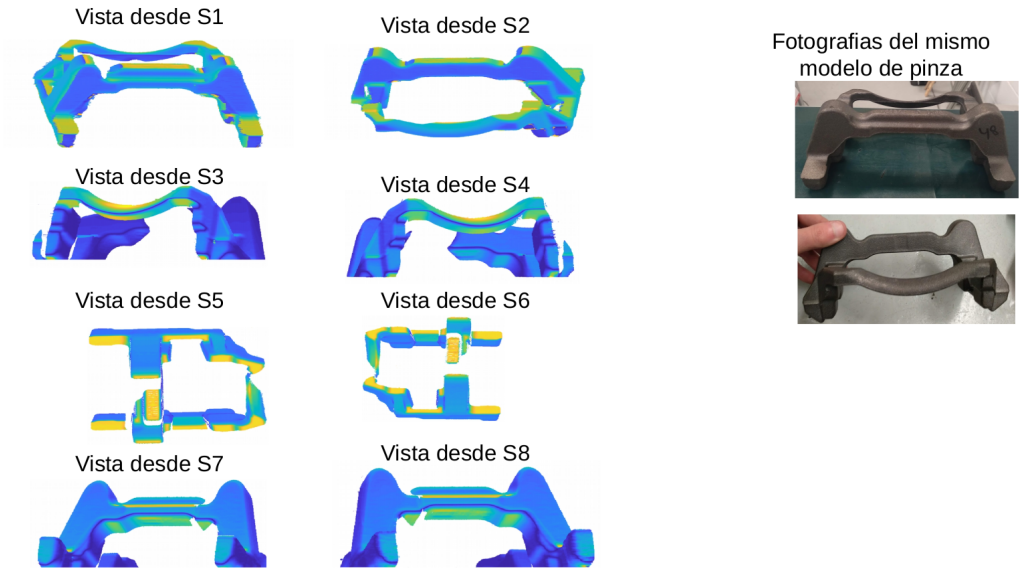


Figura 5: Análisis completo de pinza.

etiquetado de las piezas se realiza de forma manual mediante de un software de etiquetado propio desarrollado en la plataforma **Qt** [19] mediante el lenguaje de programación **c++**.
En cuanto al número de categorías, en fases previas del proyecto se propuso dividir los blobs en 10 subcategorías dentro de las dos categorías principales: defectos (D) y falsos positivos (FP).

1	D-Defecto genérico
2	FP-Falso positivo genérico
3	D-Arena
4	D-Depresión
5	D-Grieta
6	D-Agujero
7	D-Junta fría
8	FP-Rebaba
9	FP-Forma
10	FP-Ruido

Cuadro 2: Categorías de defectos / falsos positivos.

Esta división finalmente se descartó debido a que el objetivo final del proyecto no es clasificar tipos de defectos sino diferenciar entre defectos y no defectos, por lo que el esfuerzo y el tiempo extra invertido no se vería reflejado en el resultado final. Finalmente los blobs se dividieron en dos categorías:

0	Normal
1	Defecto
2	Falso positivo

Cuadro 3: Categorías de blobs.

Estas etiquetas se guardan en una máscara de etiquetas que se genera junto con el resto de características.

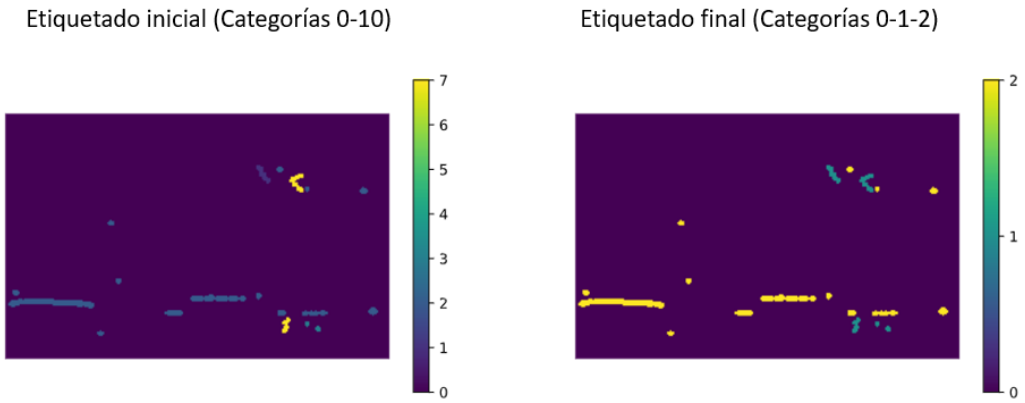


Figura 6: Ejemplo de máscara de etiquetas.

Por último, con el objetivo de facilitar la identificación de cada mancha, se genera una máscara adicional que asigna un identificador a cada blob y se genera una máscara similar a la anterior. Estos identificadores se asignan de izquierda a derecha y de arriba a abajo, siendo los identificadores:

$$idx = [1, 2, 3..., n] \tag{5}$$

siendo n el número de blobs en la imagen.

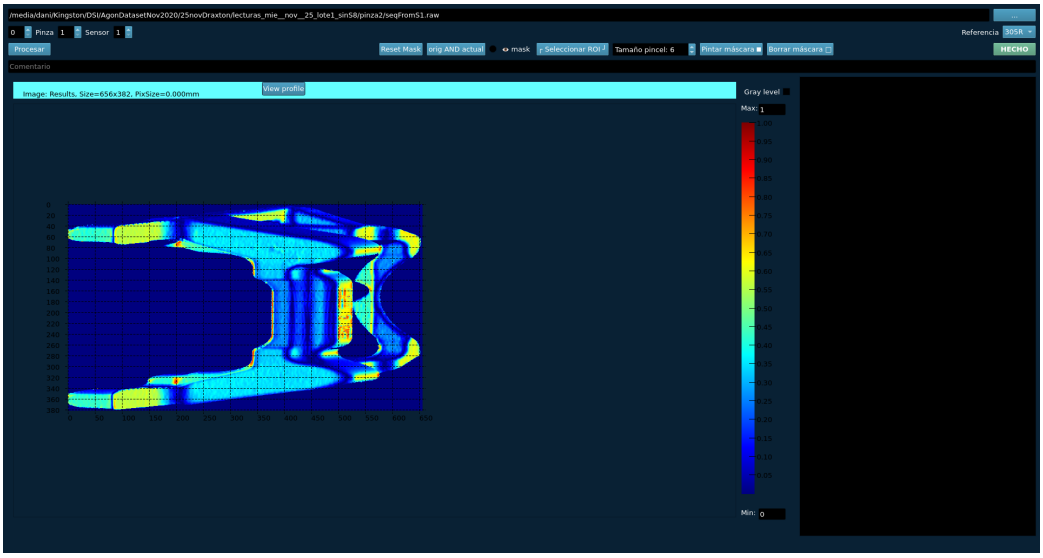


Figura 7: Interfaz de aplicación de etiquetado.

2.2. Preprocesamiento de los datos

Como ya se ha comentado previamente, resolver un problema de ML consiste en optimizar una función matemática a partir de una serie de ejemplos. Para poder cumplir esta premisa, es necesario trabajar con datos numéricos que permitan ser usados en un contexto matemático.
Por este motivo, una buena preparación de los datos es de vital importancia y afectará en gran medida al resultado final del proyecto. Se estima que el tiempo invertido en esta sección corresponde al 60 % del tiempo total dedicado al proyecto.

2.2.1. Extracción de características

Una vez escaneada y etiquetada la nube de puntos, es necesario generar un conjunto de características $\mathbf{f}_{l,n} = [f_{l,n}^{(1)}, f_{l,n}^{(2)}, \ldots, f_{l,n}^{(m)}]$ que permitan diferenciar los puntos normales de los puntos de defectos.
Debido a la falta de estudios previos en piezas de características similares, se decide extraer la mayor cantidad posible de características, intentando minimizar la cantidad de información redundante con el objetivo de evitar

problemas como la maldición de la dimensión (*curse of dimensionality*) introducida por primera vez por Bellman [20].
A partir de la nube de puntos se extraen las siguientes características $f_{l,n}^{(m)}$:

- Distancia al sensor obtenida a partir de la imagen 3D de partida.
- Normales en los tres ejes n_x, n_y, n_z. Contienen el valor descompuesto en coordenadas del vector unitario normal a la superficie en cada píxel.
- Valores propios $\lambda_1, \lambda_2\ \lambda_3$. Contienen para cada píxel los valores propios de la matriz de covarianza de la vecindad que le rodea.

Los valores propios, generados a partir de la información espacial de la nube de puntos 3D, pueden ser usados para generar nuevas características que describan la estructura local de cada píxel, así como características geométricas adicionales, definidas en [21]:

$$Linearity : L_\lambda = \frac{\lambda_1 - \lambda_2}{\lambda_1} \tag{6}$$

$$Planarity : P_\lambda = \frac{\lambda_2 - \lambda_3}{\lambda_1} \tag{7}$$

$$Sphericity : S_\lambda = \frac{\lambda_3}{\lambda_1} \tag{8}$$

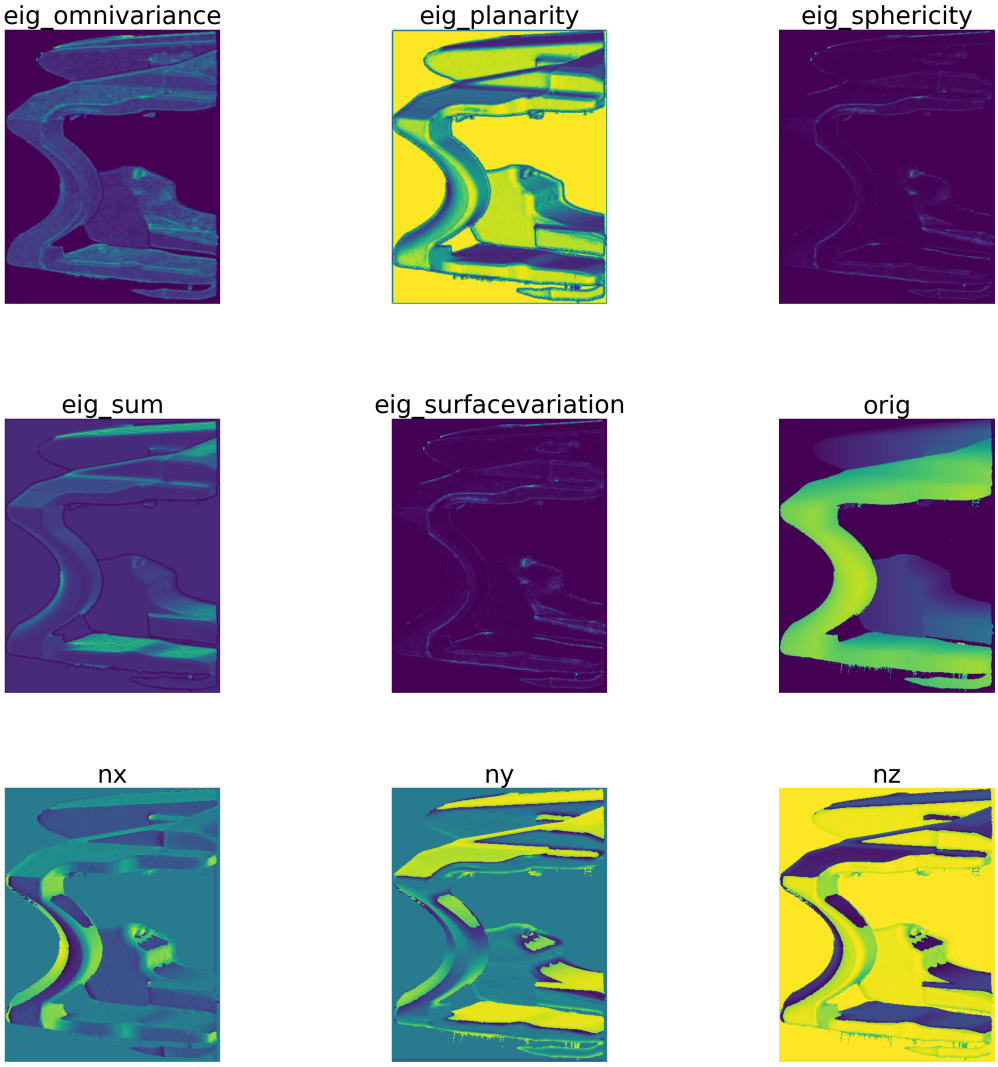
$$Omnivariance : O_\lambda = \sqrt[3]{\lambda_1 \lambda_2 \lambda_3} \tag{9}$$

$$Anisotropy : S_\lambda = \frac{\lambda_1 - \lambda_3}{\lambda_1} \tag{10}$$

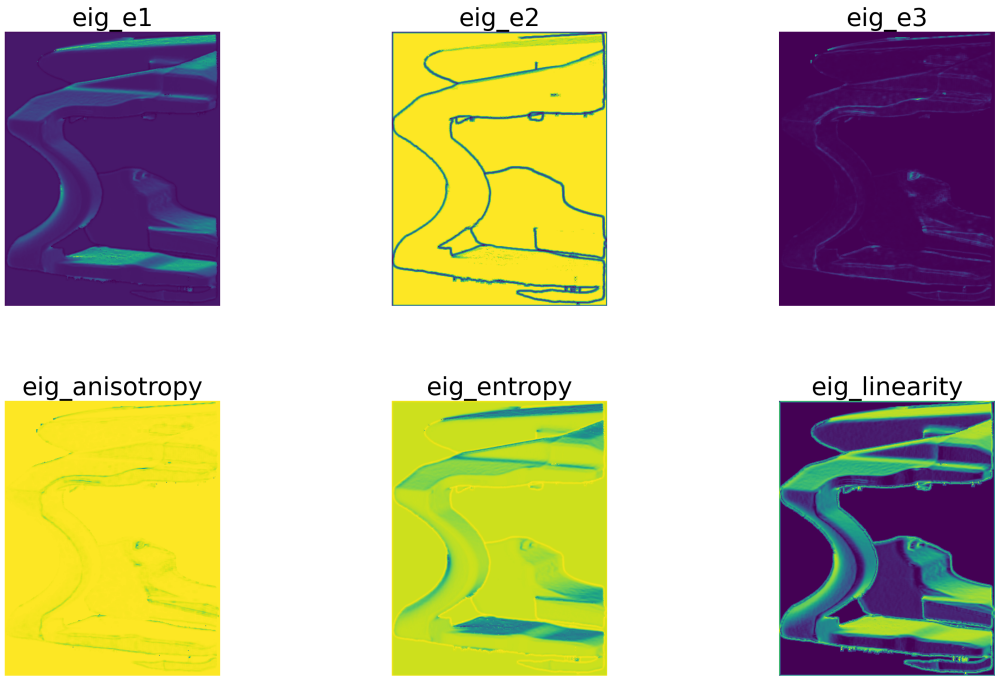
$$Eigenentropy : E_\lambda = \sum_{i=1}^{3} \lambda_i \ln(\lambda_i) \tag{11}$$

$$Suma\ de\ \lambda s : \sum_{\lambda} = \lambda_1 + \lambda_2 + \lambda_3 \tag{12}$$

$$surface\ variation : C_\lambda = \frac{\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3} \tag{13}$$



Pese a que las características previamente mencionadas son calculadas a partir de información, no sólo del píxel a analizar, sino también de su contorno, se proponen dos alternativas para aumentar la información del contorno introducida al modelo. Para ello, se define la *imagen de características* f, siendo el número total de píxeles N^f y siendo $f_{l,m}$ el píxel correspondiente a la fila l y columna m, con valor $f_{x_i,y_i} = f(x_i, y_i)$. Suponemos que $W_{f_{l,m}}$ es una ventana



Cuadro 4: Características de entrada.

de tamaño $ws \times ws$ centrada sobre el píxel $f_{l,m}$.

$$W_{f_{l,m}} = \{f_{l,m}\} \quad \begin{array}{l} \forall\, l \subset [l - \frac{ws}{2}, l + \frac{ws}{2}] \\ \forall\, m \subset [m - \frac{ws}{2}, m + \frac{ws}{2}] \end{array} \tag{14}$$

El número total de píxeles en $W_{f_{l,m}}$ es n_i^f ($n_i^f < ws * ws$). Las dos operaciones propuestas sobre f son:

1. **Generando características agregadas**. Para cada imagen de características original f, se genera una imagen de características derivada f_{ws} formada por el valor medio de cada píxel en una ventana:

$$f_{ws} = \frac{1}{n_i^L} \sum_{k,l \in W_{f_{l,m}}^L} f(k,l) \tag{15}$$

   Tras comparar varios valores de ws, se decidió que el valor óptimo es $ws = 5$.

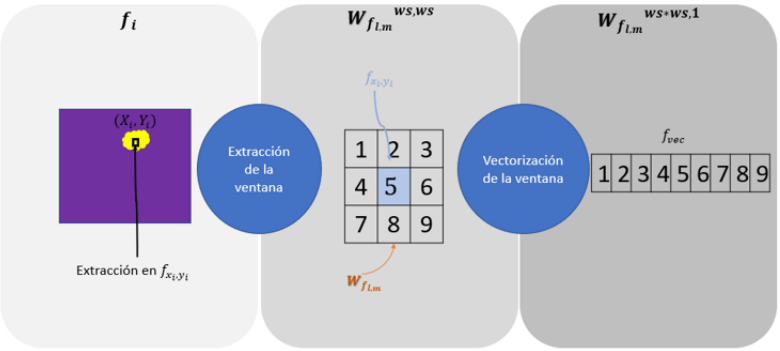
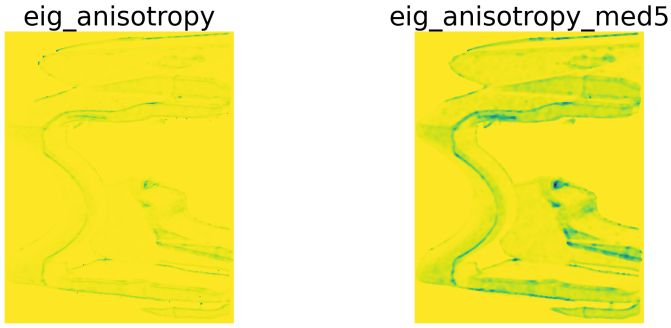


Figura 8: Ejemplo de vectorización en P_i con ws=3.

2. **Vectorizando los valores de f en una ventana $\mathbf{W}_{\mathbf{P_i}}^{\mathbf{f}}$ centrada en** P_i. Para cada píxel P_i de la imagen original f, se extrae un vector de características f_{vec} de tamaño n^f correspondiente a los valores de f en $W_{P_i}^f$.

$$f_{vec} = W_{f_{l,m}}{}^{ws,ws} \rightarrow W_{f_{l,m}}^{ws*ws,1} \tag{16}$$

De esta forma se consigue aumentar el vector de entrada con un factor de ws. Se hicieron pruebas con distintos valores de ws en busca del valor óptimo que maximice la cantidad de información de entrada en los modelos sin caer en problemas de sobreentrenamiento [22].



Cuadro 5: Comparativa ente f y f_{ws} en ventana de 5x5.

2.2.2. Limpieza de los datos

El proceso de limpieza de datos es un proceso clave dentro de cualquier proyecto de ML. El objetivo es encontrar los *outliers* (datos incorrectos, incompletos, inprecisos, irrelevantes o faltantes dentro del dataset) y modificarlos o eliminarlos en función de las necesidades. Debido a la gran cantidad de datos disponibles, se decidió eliminar todos los datos que pudiesen afectar negativamente al rendimiento final del modelo.
Tras una primera inspección de los datos en crudo, se pudo observar que la mayoría de datos defectuosos se producen por los siguientes motivos:

- *Valores no finitos en la extracción de características.* Características como linearity, anisotropy, sphericity o planarity tienen en común que se calculan a partir de una combinación lineal de los valores propios y, más en concreto, una división. Este hecho hace posible que, en situaciones puntuales, **se produzca una división entre 0**. Durante el entrenamiento se eliminan estos valores ya que se dispone de una cantidad suficiente de datos. Por el contrario, durante la inferencia del modelo no es posible descartar dichos puntos por lo que se sustituyen por el valor medio en una ventana de tamaño 3x3.

- *Problemas en la conversión.* La nubes de puntos escaneadas por los sensores son almacenadas en formato *.raw*. Este formato no es compatible con el lenguaje de programación Python, por lo que es necesario realizar la conversión de *.raw* a un array de flotantes de 32 bits. Esta conversión se realiza mediante una función propia, que se ha comprobado que en situaciones puntuales genera valores nulos o infinitos. Del mismo modo, se ha comprobado que **genera valores fuera del rango de las características que es necesario eliminar**. Un ejemplo es la aparición de valores propios negativos, imposible por definición. Es posible que aparezcan más valores erróneos pero resulta imposible su detección si el valor está dentro de la distribución normal de la variable. La frecuencia de estos valores no es significativa ($<1\,\%$) por lo que el entrenamiento no se ve afectado por estos valores. Debido a que la inferencia se realiza en *c++*, donde no aparece este problema, se decide considerar este fallo como aceptable en lo que resta de proyecto.

Por último, se han encontrado filas de ceros, correspondientes a errores en el proceso de extracción de características que deben ser eliminados también.

Este fallo se produce en el proceso de almacenamiento de las características, por lo que tampoco afectará durante la inferencia.

2.2.3. Visualización de los datos

El desarrollo de sistemas basados en ML es, de forma inherente, un proceso iterativo. El tiempo empleado en la preparación de los datos, así como en la selección y optimización del modelo generalmente suele ser grande, por lo que este proceso iterativo puede volverse frustrante. Las técnicas de visualización de datos resultan muy útiles para adquirir un mayor conocimiento e intuición sobre los datos, así como identificar los patrones principales que puedan aparecer, reduciendo el número de iteraciones necesarias.

En primer lugar se analizan las distribuciones característica por característi-

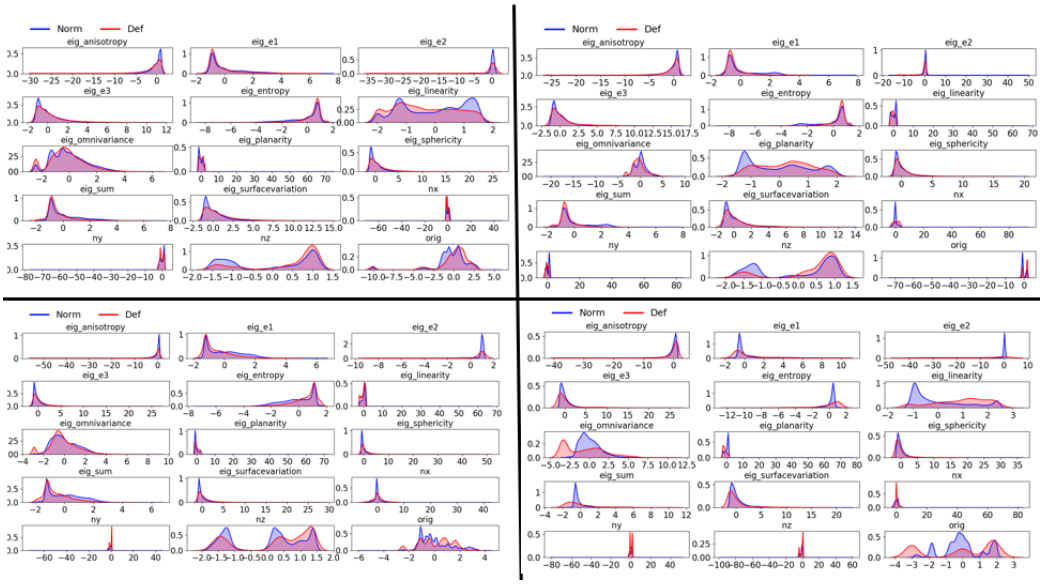


Figura 9: Distribución de características por sensores. Esquina sup. izq. (Sensores 1 y 2), esquina sup. der. (Sensores 3 y 4), esquina inf. izq. (Sensores 5 y 6) y esquina inf. der. (Sensores 7 y 8).

ca en busca de diferencias entre los puntos de defecto y los puntos normales. Las distribuciones son calculadas por pares de sensores debido a que trabajan en rango distintos tal y como se mencionó previamente. De forma adicional, esta primera inspección permite comprobar que la limpieza de datos previos

ha sido correcta y no quedan valores erróneos que puedan modificar la distribución de las variables.
Pese a que se pueden apreciar diferencias en las distribuciones de puntos normales y puntos de defecto, resulta díficil establecer un conjunto de reglas de forma manual para diferenciar ambas clases, aunque sí que se puede apreciar que las características elegidas son sensibles al defecto y, por tanto, portan la información adecuada para la clasificación posterior.
Con el objetivo de analizar la estructura de los datos del conjunto de características, se propone reducir la dimensionalidad de los vectores de entrada $\mathbf{f}$ para poder realizar una visualización en 2D. Algoritmos como t-SNE [23], permiten reducir el número de dimensiones de forma que puntos cercanos en el espacio de alta dimensión de entrada permanezcan cercanos en el espacio de baja dimensión de salida, también conocido como espacio latente. Por este motivo, aplicar estos algoritmos puede ser de gran ayuda para evaluar la capacidad de agrupar puntos de defecto / normales con las características seleccionadas.
Debido a la gran cantidad de datos disponibles para los sensores 5-6 y 7-8 se

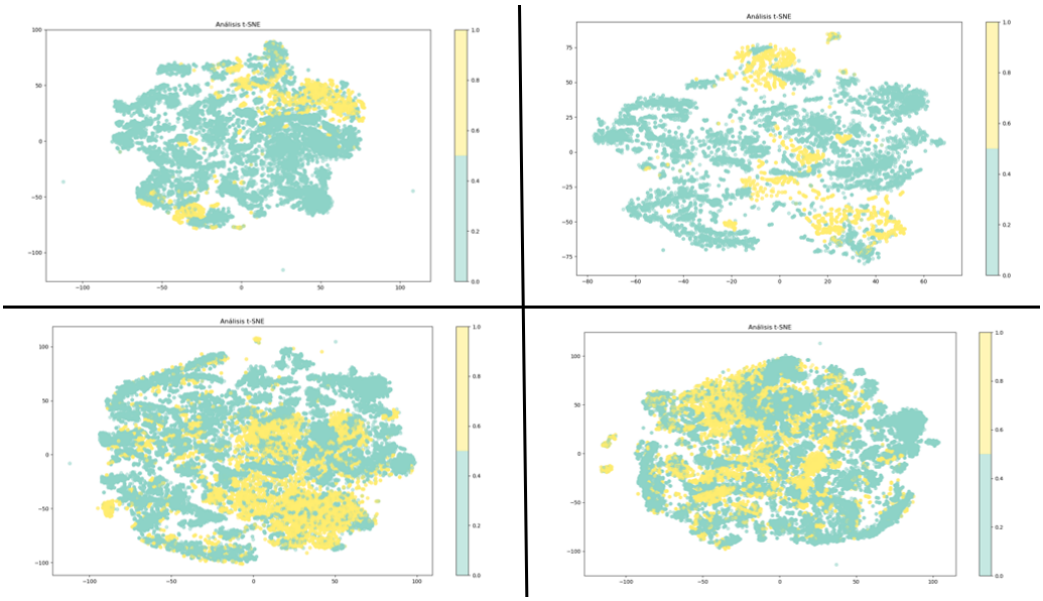


Figura 10: Visualización 2D de las características. Esquina sup. izq. (Sensores 1 y 2), esquina sup. der. (Sensores 3 y 4), esquina inf. izq. (Sensores 5 y 6) y esquina inf. der. (Sensores 7 y 8).

decide realizar la visualización sobre una muestra aleatoria de 50.000 puntos. Tras los resultados obtenidos, se concluye que la estructura de los datos es lo suficientemente compleja como para que el t-SNE no pueda resumirla en un mapa 2D, por lo que son necesarias técnicas más complejas para realizar esta tarea.

2.3. Técnicas de análisis

Como ya se ha comentado previamente, el objetivo del ML es crear un modelo que nos permita resolver una tarea dada. Una vez definido el modelo, se entrena usando una gran cantidad de datos de entrenamiento. El modelo aprende de estos datos y es capaz de hacer predicciones.
En función de la naturaleza de la tarea que se quiera resolver, se pueden dividir los modelos de ML disponibles en tres grandes categorías:

- *Métodos supervisados* [24]. En los métodos supervisados, los modelos se entrenan con datos etiquetados, intentando encontrar una función que, dadas las variables de entrada, les asigne la etiqueta de salida adecuada.
- *Métodos semi-supervisados* [25]. Para que los métodos supervisados tengan un buen desempeño son necesarios grandes volúmenes de datos etiquetados, los cuales no siempre están disponibles y son difíciles de obtener, ya que requieren del tiempo y esfuerzo de un especialista para etiquetarlos. Sin embargo, en la mayoría de los casos abundan grandes cantidades de datos sin etiquetar. Los métodos semi-supervisados utilizan una combinación de datos etiquetados y no etiquetados para obtener resultados.
- *Métodos no supervisados* [26]. Los métodos no supervisados se utilizan cuando no se dispone de datos etiquetados para el entrenamiento. Por tanto el objetivo de estos modelos no será predecir un valor de salida, sino describir la estructura de los datos.
- *Métodos de aprendizaje por refuerzo* [27]. En estos métodos, los modelos aprenden a base de pruebas y errores en un número de diversas situaciones. Aunque conoce los resultados desde el principio, no sabe cuáles son las mejores decisiones para llegar a obtenerlos. Lo que sucede es que el modelo progresivamente va asociando los patrones de éxito, para repetirlos una y otra vez hasta perfeccionarlos y volverse infalible.

En el presente trabajo se planteará la detección de defectos superficiales tanto como un problema supervisado, como no supervisado. En el primer caso el objetivo será clasificar los puntos de defecto y los puntos normales mediante ejemplos etiquetados de ambos casos, mientras que en el caso de los métodos no supervisados, el objetivo será detectar los puntos anómalos entrenando el modelo solo con muestras normales.

2.3.1. Métodos supervisados

Se denominarán **clasificadores** a los modelos entrenados con pares de ejemplos $\{\mathbf{f}_{l,n}, y_{l,n}\}$ siendo $y_{l,n} = 0$ para puntos sin defecto y $y_{l,n} = 1$ en el caso de puntos con defecto.
En los estudios previos realizados para evaluar la viabilidad del proyecto, se estudiaron diferentes arquitecturas de ML como los árboles de decisión [28, 29] o los clasificadores basados en máquinas de soporte vectorial (SVC) [30, 31] o las redes neuronales 'feedforward'. Tras este estudio se decidió utilizar la arquitectura de **red neuronal** para el desarrollo del presente trabajo.
Una red neuronal, al igual que el cerebro humano, está formada por un conjunto de neuronas interconectadas entre sí y agrupadas en capas. Existen tres tipos de capas en cualquier red neuronal: de entrada, de salida y ocultas. Las capas de entrada reciben las señales desde el entorno (un vector de características $f_{l,n}$); las capas de salida envían la señal fuera de la red (la predicción $y_{l,n}$), y las capas ocultas son aquellas cuyas entradas y salidas se encuentran dentro de la red.
Las conexiones que unen las neuronas que forman la red tienen asociadas un peso y definen la transformación que sufre la información de una capa a la siguiente. Se considera que el efecto de la señal es aditivo, de tal forma que la entrada neta que recibe una neurona $Z_{i,j}$ es la suma del producto de cada señal individual $\mathbf{A_{i-1}}$ por el valor de la sinapsis que conecta ambas neuronas $\mathbf{W_i}$ junto con un término independiente b_i y es lo que se conoce como red de propagación.

$$Z_{i,j} = \mathbf{A_{i-1}} * \mathbf{W_i} + b_i \tag{17}$$

Asociada a cada neurona hay una función de activación $g(x)$ que transforma el estado actual $Z_{i,j}$ de la neurona en una señal de salida $A_{i,j}$.

$$A_{i,j} = g(Z_{i,j}) \tag{18}$$

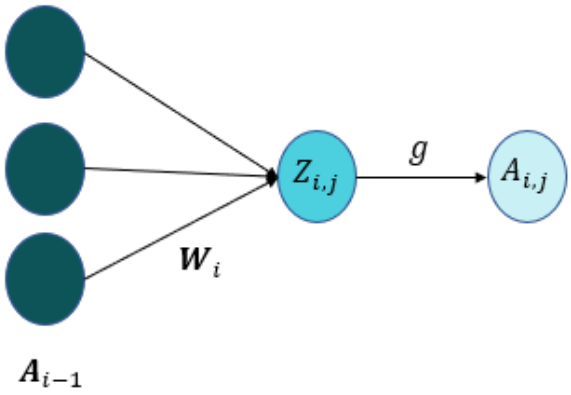


Figura 11: Estructura de cada neurona de la red neuronal.

El entrenamiento de $\mathbf{W_i}$ y de b_i se realiza mediante el método 'backpropagation' [32], basado en el descenso del gradiente, cuyo objetivo será minimizar una función de coste. En este caso, al tratarse de un problema de clasificación, la función de coste utilizada es *binary crossentropy*:

$$\mathcal{L} = -[y_{i,j} \log(\hat{y}_{i,j}) + (1 - y_{i,j}) \log(1 - \hat{y}_{i,j})] \tag{19}$$

donde $\hat{y}_{i,j}$ hace referencia a la predicción del modelo e $y_{i,j}$ hace referencia a la clase real (Defecto / No defecto).
Con el objetivo de optimizar el rendimiento de las redes neuronales es necesario ajustar los siguientes hiperparámetros:

- Número de capas L.
- Neuronas en cada capa. En el caso de la capa de entrada, el número de neuronas n_x viene determinado por el preprocesamiento realizado en 2.2.1. El número de neuronas de cada capa oculta n_l para ($l = 1, 2, ...L - 1$) se determina de forma manual y el número de neuronas en la capa final n_y viene determinado por el número de clases a predecir.
- Función de activación para cada capa. Se comparan los resultados utilizando las funciones *'rectified linear activation function'* (ReLU) e *'hyperbolic tangent Activation Function'* (tanh) para las capas ocultas y la función *'Sigmoid'* para la última capa de forma que la salida del modelos será un valor en el rango [0,1].

- Optimizador. Método de optimización de la matriz de pesos **W**. En este proyecto se compararon los resultados utilizando *Adam* [33], *stochastic gradient descent* (SGD), *RMSProp*.
- 'Learning rate'. Se considera el hiperparámetro más importante a la hora de configurar una red neuronal, ya que una mala parametrización puede provocar una convergencia muy lenta del modelo, o incluso una divergencia del modelo.
- Número de iteraciones. A partir de un número determinado de iteraciones, se observa que el modelo no mejora su rendimiento e incluso puede llegar a sobreentrenar, por lo que es necesario limitar el número de iteraciones durante el entrenamiento. La librería de keras [34] permite deterner el entrenamiento de forma automática en el momento en el que no se aprecie mejora en el modelo tras un mínimo de iteraciones, lo que se conoce como *early stopping*.
- Tamaño de cada lote. El parámetro varía en función de la memoria del dispositivo donde se entrene el modelo.

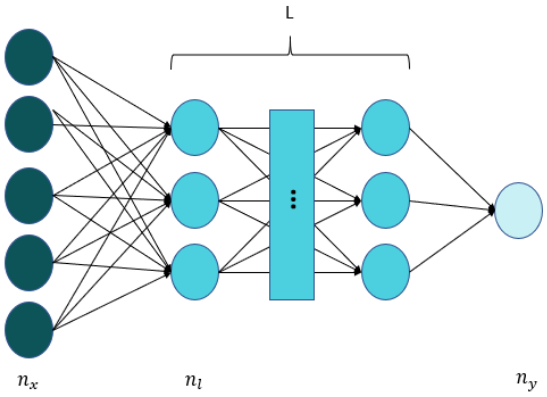


Figura 12: Estructura de una red neuronal 'feedforward'.

La selección del valor óptimo de cada uno de los parámetros previamente mencionados se determina de forma empírica utilizando las métricas de validación 2.4.2 como referencia. Se utiliza la librería de optimización *hyperas*

[35] con el objetivo de automatizar este proceso.
Estos parámetros fueron optimizados con un conjunto de datos disponible en la fecha de realización de este documento. A medida que el proyecto avance y se disponga de más datos una vez que el modelo esté en funcionamiento, se realizarán reentrenamientos periódicos con el objetivo de mantener un modelo óptimo en producción pese a las variaciones que pueda sufrir la distribución de los datos de entrada.
Con el objetivo de evitar el sobreentrenamiento se añade una capa de regularización tras cada capa densa. La librería de *Keras* ofrece varias opciones pero para este proyecto se utiliza una capa de "Dropout"[36] que aleatoriamente multiplica por 0 los valores de entrada en la capa con un ratio 'rate' definido manualmente.

2.3.2. Métodos no supervisados

Aunque estos modelos se suelen utilizar para tareas de *clustering* [37], el objetivo en este proyecto será detectar anomalías en los datos de entrada. Por lo tanto, el objetivo del modelo será reproducir en la salida el mismo vector asignado a su entrada.

Para resolver esta tarea se utilizarán modelos con estructura de autoencoder. Los autoencoders son redes neuronales con el objetivo de generar nuevos datos primero comprimiento la entrada en un espacio de variables latentes de menor dimensión que la entrada y luego reconstruyendo la salida en base a la información adquirida. Estos modelos constan de dos partes bien diferenciadas:

- *Encoder*: Es la parte de la red que comprime la entreada en un espacio de variables latentes y que puede representarse mediante la función de codificación $h = e(x)$.
- *Decoder*: Es la parte de la red que trata de reconstruir la entrada en base a la información recolectada previamente. Se representa mediante la función de decodificación $\hat{x} = d(h)$.

De tal forma que la ecuación del autoencoder se puede decribir como:

$$d(e(h)) = \hat{x} \tag{20}$$

donde $\hat{x}$ es la reconstrucción de la entrada original x, tal y como se muestra en la Figura 13.

Lo que se espera es que, cuando entrenamos un autoencoder y reproducimos

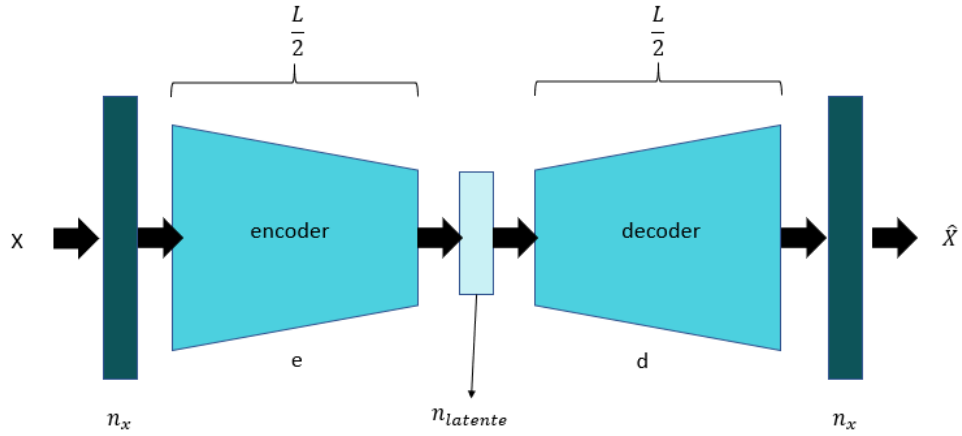


Figura 13: Estructura de un autoencoder.

la entrada a la salida del modelo, el espacio de variables latentes h sea capaz de capturar las características más relevantes de los datos de entrenamiento. Si no se imponen suficientes restricciones en la codificación (tamaño del espacio latente demasiado grande), es posible que la red se dedique a copiar la entrada en la salida. Para el desarrollo del proyecto se compararán los resultados con dos arquitecturas distinas:

- *Auto-Encoder* (AE) [38].
- *Variational Auto-Encoder* (VAE) [39].

La principal diferencia entre ambas arquitecturas reside en que los autoencoders 'aprenden' una representación comprimida de la entrada a través de la compresión de la entrada (encoder) y la descompresión (decoder) para reconstruir la entrada original. El aprendizaje se realiza mediante la medición de la distancia entre la reconstrucción y la entrada original de forma que se mide la cantidad de información perdida en el proceso de compresión. Por otro lado, los modelos VAEs, además de aprender una función que represente el espacio comprimido como los AEs, aprenden la distribución de los datos de entrada.

En el caso de los AEs, la función de coste es el error cuadrático medio, mientras que la función de coste en el VAE es compleja y debe ser implementada manualmente. Esta función está compuesta por dos términos:

- *Error de reconstrucción.* Al igual que en el AE, se evalúa la capacidad del modelo de reconstruir el vector de entrada. Siendo K es tamaño del vector de entrada $\mathbf{X}$, se calcula el error de reconstrucción para cada muestra (i) $\mathcal{L}_{recon}^{(i)}$:

$$\mathcal{L}_{recon}^{(i)} = \frac{-1}{K} \sum_{k=1}^{K} (\hat{x_k}^{(i)} - x_k^{(i)})^2 \tag{21}$$

- *Divergencia Kullback-Leibler* [40]. Se trata de un término de regularización entre la distribución resultante y la distribución Gaussiana. Siendo J el tamaño del espacio latente, se calcula la divergencia Kullback-Leiber para una muestra (i) $\mathcal{L}_{latent}^{(i)}$ donde μ y σ^2 corresponden a la media y la varianza respectivamente del epacio latente:

$$\mathcal{L}_{latent}^{(i)} = -\frac{1}{2J} \sum_{j=1}^{J} (1 + \log(\sigma_j^{(i)})^2 - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2) \tag{22}$$

Ambos términos se encuentran en rangos de valores distintos ya que $\mathcal{L}_{recon}$ se calcula sobre tamaño del espacio de entrada y $\mathcal{L}_{latent}$ se calcula sobre el tamaño del espacio latente. Para poder comparar ambos valores, se normaliza cada función de coste en función del tamaño del espacio K y J respectivamente.
En la ecuación 23 se puede ver la función de coste final, donde M es el tamaño del lote a evaluar, $\mathcal{L}_{total}^{(m)}$ hace referencia al coste para el lote m, y c1 y c2 son parámetros configurables que permiten modificar la importancia de cada uno de los términos durante el entrenamiento. Tras varias pruebas, se consideró que unos buenos valores para estos términos son c1=0.1 y c2=1.

$$\mathcal{L}_{total}^{(m)} = \frac{1}{M} \sum_{i=1}^{M} (c1\mathcal{L}_{latent}^{(i)} + c2\mathcal{L}_{recon}^{(i)}) \tag{23}$$

De forma similar a los métodos supervisados, se optimizan los siguientes hiperparámetros:

- Número de capas L. El modelo se divide en dos modelos simétricos e inversos denominados 'encoder' y 'decoder'. El tamaño de cada uno de estos modelos será $\frac{L}{2}$.
- Tamaño del espacio latente $n_{latente}$. Número de neuronas en el cuello de botella.
- Neuronas en cada capa. En el caso del encoder, el tamaño de la capa de entrada n_x viene determinado por el vector de entrada. El número de neuronas de cada capa oculta n_l para $(l = 1, 2, ...\frac{L}{2} - 1)$ se determina de forma manual y el número de neuronas en la capa final $n_{\frac{L}{2}}$ viene determinado por el tamaño del espacio latente. De forma similar, el tamaño de la capa de entrada del decoder viene determinado por el tamaño del espacio latente y el tamaño de la capa de salida viene determinado por el vector de entrada.
- Función de activación para cada capa. Se comparan los resultados utilizando las funciones *'rectified linear activation function'* (ReLU) e *''hyperbolic tangent Activation Function'* (tanh) para las capas ocultas y la función *'Sigmoid'* para la última capa de forma que la salida del modelos será un valor en el rango [0,1].
- Optimizador. Método de optimización de la matriz de pesos W. En este proyecto se compararon los resultados utilizando Adam [33], stochastic gradient descent (SGD), RMSProp.
- 'Learning rate'.
- Número de iteraciones.
- Tamaño de cada lote.

Mientras que en los métodos supervisados la salida del modelo es directamente la probabilidad de fallo, en los métodos no supervisados es necesario definir un método para la clasificación normal/defecto. Esta clasificación se realiza a partir del **residuo**. Se define *residuo* como la diferencia entre la entrada x y la salida $\hat{x}$ calculada mediante el error cuadrático medio de forma que:

$$res = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 \tag{24}$$

Debido a que el objetivo tanto en los métodos supervisados como en los no supervisados es obtener una máscara binaria que diferencie los puntos normales y defectos, es necesario seleccionar un **umbral** de forma que, siendo P el valor a la salida del modelo en los métodos supervisados y el residuo en los métodos no supervisados, el proceso de umbralizado devuelve:

$$\begin{cases} 0, & if P < umbral \\ 1, & if P \geq umbral \end{cases} \tag{25}$$

2.3.3. Selección de los datos de entrenamiento

Al trabajar con nubes de puntos de gran resolución, es necesario escoger la cantidad de píxeles que se extrae de cada nube, ya que trabajar con la totalidad de los puntos supondría un esfuerzo computacional ingente, que implicaría tiempos de ejecución innecesariamente largos ya que muchos de los puntos son muy similares entre sí. Se utiliza la máscara generada en el etiquetado (ver Figura 6) para seleccionar los puntos de interés (defectos y FPs). Con el objetivo de crear un modelo robusto ante nuevas referencias de piezas, se añaden puntos normales de la pieza al conjunto de entrenamiento. La cantidad de datos normales añadidos por pieza es:

$$N_{puntosNormales} = N_{puntosDefecto} + N_{puntosFPs} \tag{26}$$

Al realizar el etiquetado de los datos de forma manual, pese a realizarse con la mayor precisión posible, se introduce un error en el etiquetado de los bordes de los blobs. Por este motivo se decide aplicar un procesamiento previo a la máscara generada durante el etiquetado. El procesamiento consiste en aplicar un filtro de erosión [41] con un kernel de tamaño 5. De esta forma se considera como defecto únicamente la parte central del defecto etiquetado en primera instancia. Un beneficio adicional de este procesamiento es que, eliminando los bordes (zona de transición), se genera una mayor diferencia entre los puntos normales y de defecto, con lo que se espera mejorar el rendimiento final del modelo.

Los puntos normales se seleccionan utilizando una máscara binaria que define los límites de la pieza para evitar seleccionar puntos fuera de la misma. De la misma forma que se 'reducen' los blobs para extraer los puntos de defecto, se aplica un filtro de dilatación con kernel de tamaño 5 con el objetivo de

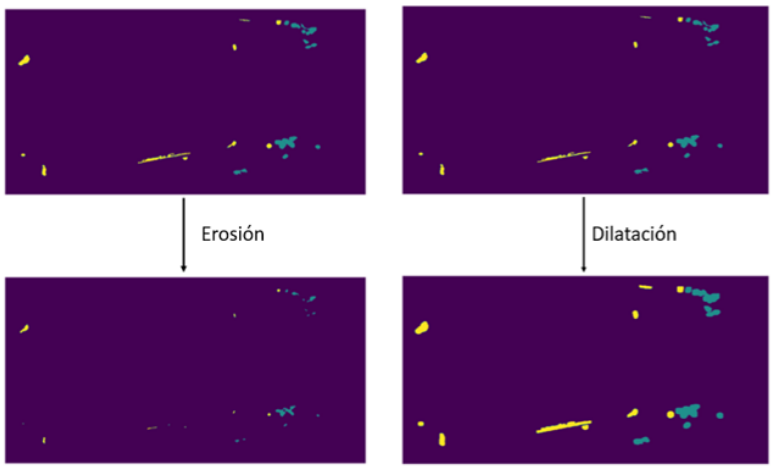


Figura 14: Proceso de erosión de máscara de defectos.

'aumentar' los bordes de los blobs y, por tanto, reducir la posibilidad de seleccionar puntos de defecto como puntos normales.

Finalmente, tal y como se muestra en la figura 15, se combina la máscara

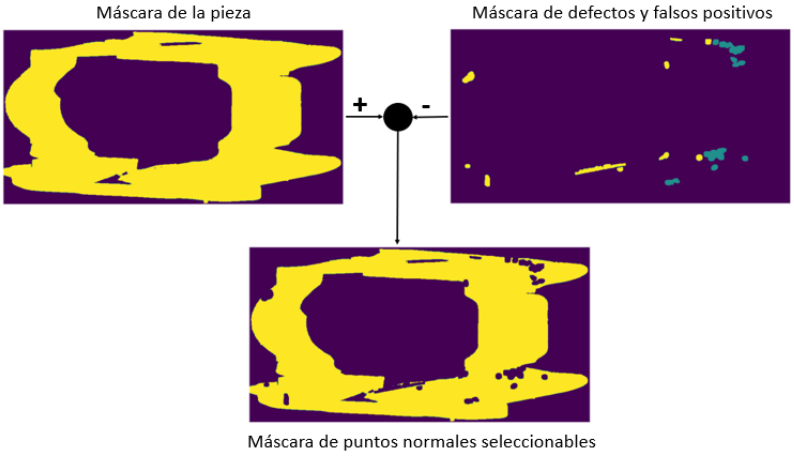


Figura 15: Máscara procesada de selección de puntos normales.

dilatada de la figura 14 con la máscara del fondo de la pieza para extraer los

puntos normales seleccionables de tal forma que:

$$máscara_{puntosSeleccionables} = máscara_{pieza} - máscara_{DefectosYFalsosPositivos} \tag{27}$$

2.3.4. Normalización de los datos

En los problemas de ML, es común que las características de entrada tengan rangos de valores diferentes. Se ha comprobado que una incorrecta normalización de los datos, no solo no mejora los resultados, si no que en algunos casos llega a empeorar el rendimiento de los modelos [42]. Pese a esto, la estandarización de los datos de entrada es una de las técnicas de preprocesamiento más comunes. Esto se debe a que, pese a que en ambos casos se llegue a la solución óptima, la optimización de la función de coste utilizando datos normalizados es más rápida.
En el caso de que no se normalicen los datos de entrada, la función de coste tendrá una forma alargada, por lo que será necesario un learning rate más pequeño y más iteraciones para converger en la solución óptima.
Para el presente proyecto, la normalización, al igual que el resto de etapas de preprocesamiento, se realiza por parejas de sensores, tal y como se explicó en los apartados anteriores.
Por otro lado, como se puede ver en la Figura 16, cuando se trabaja con características en rangos de valores similares, la función de coste es más simétrica y necesita menos iteraciones y un valor de learning rate más alto para llegar a la solución óptima.
Para cada característica f, se divide el proceso de normalización en dos etapas:

1. Se calcula el valor medio de la característica.

$$\mu = \frac{1}{m}\sum_{i=1}^{m} f^{(i)} \tag{28}$$

2. Se calcula la desviación estándar.

$$\sigma = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(f_i - \mu)^2} \tag{29}$$

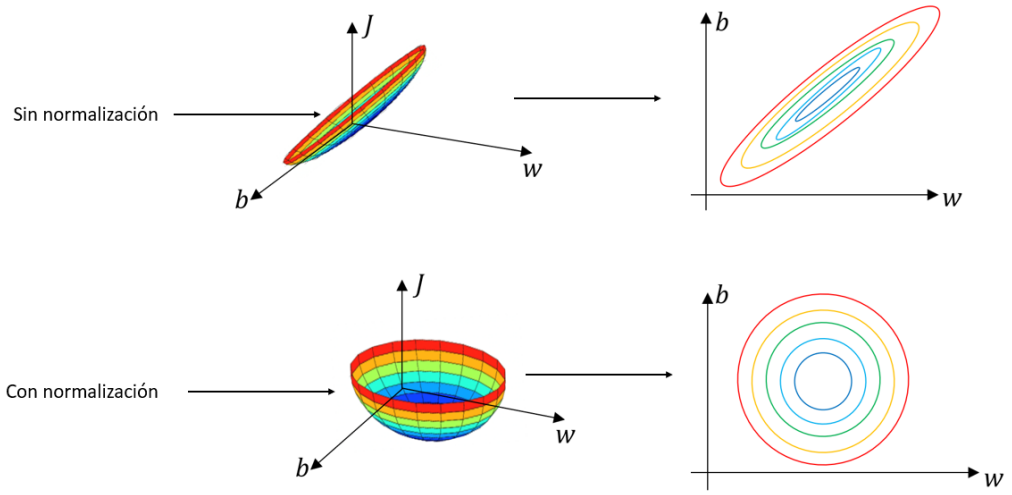


Figura 16: Efecto de la normalización sobre la función de coste.

De esta forma, el conjunto de características normalizado f_{norm} se define como:

$$f_{norm} = \frac{f - \mu}{\sigma} \tag{30}$$

2.3.5. Almacenamiento de los datos

El último paso del preprocesamiento es el almacenamiento de los datos. Debido a la cantidad de datos disponibles ($\sim$ 15 millones de puntos), es necesario utilizar un formato que optimice el espacio de almacenamiento en disco y proporcione un acceso eficiente a los datos. Formatos como csv (comma-separated values), son muy cómodos para almacenar cantidades pequeñas de datos debido a su compatibilidad con librerías como Pandas [15], pero tienen problemas de escalabilidad, ya que para acceder a los datos, aunque sea solo una parte de los mismos, exige la carga del archivo al completo.
El formato escogido para almacenar los datos es **hdf** (Hierarchical Data Format) [43]. Este formato está basado en bases de datos y grupos, lo que permite organizar de forma eficiente, no solo las características, sino información extra como la etiqueta de cada muestra, el sensor de adquisición y

otros metadatos. Adicionalmente, permite el acceso a los datos sin necesidad de cargar la totalidad de los datos en memoria por lo que permite trabajar con conjuntos de datos más grandes aun con recursos de hardware limitados.

2.3.6. División de los datos

Debido a la naturaleza de los datos, donde el número de clases está tan desequilibrado a causa de a la escasez de puntos defectuosos, se utiliza una modificación de la división estándar Entrenamiento-Validación-Test. Esta modificación se realiza porque, de realizar la separación estándar, es muy probable que en alguna de las divisiones no haya sufientes puntos de defectos. Por este motivo se decide generar manualmente un conjunto de entrenamiento eliminando la gran mayoría de puntos normales. El procedimiento de división de los conjuntos de entrenamiento permite generar un conjunto de datos lo suficientemente grande para entrenar los métodos no supervisados a la vez que se genera un sub-conjunto de datos para entrenar los métodos supervisados.

1. Se separan las muestras de cada clase en un archivo .hdf5 distinto.
2. Se divide el conjunto formado por puntos normales en Entrenamiento-Validación-Test, siendo la división de los datos 80-10-10 respectivamente. Estos 3 conjuntos se usarán para entrenar los métodos no supervisados. Para la evaluación de los métodos no supervisados se utilizará el conjunto de datos de defectos generado para el entenamiento de métodos supervisados.
3. El 20 % perteneciente a los conjuntos de Validación y Test se usa para entrenar los métodos supervisados junto con los datos de defectos. Se realiza una división 80-20 para el entrenamiento y test de estos métodos.

Esta división se muestra gráficamente en la Figura 17.

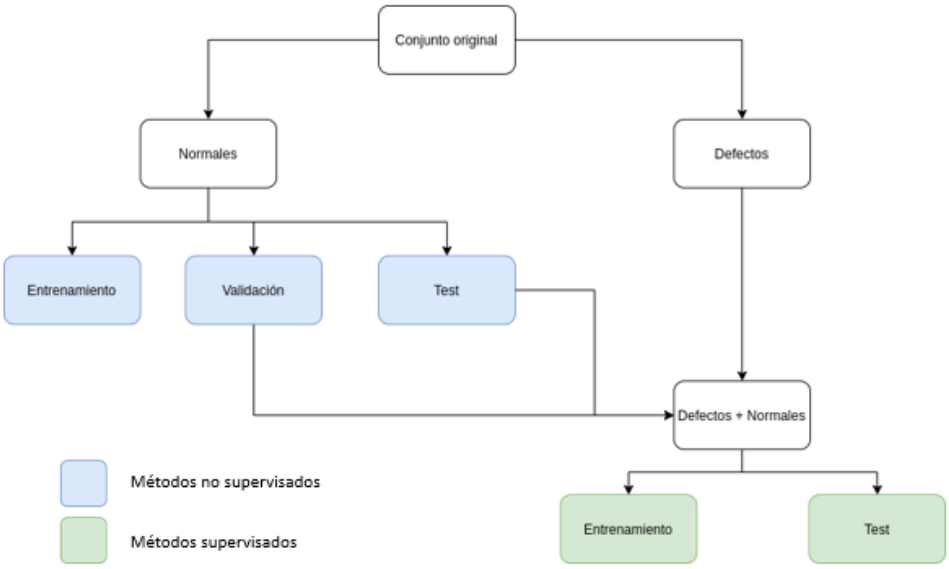


Figura 17: División de los datos Entrenamiento-Validación-Test.

2.4. Evaluación de resultados

2.4.1. Desbalance de clases

El problema del desbalance de clases es muy habitual en los proyectos de ML basados en clasificación donde hay un ratio desproporcionado de observaciones de cada clase. En este caso, como se puede ver en la Figura 18, la cantidad de muestras normales es muy superior a las muestras correspondientes a puntos defectuosos. Este desbalance puede provocar un error de "bias"[44] que afecte al rendimiento final del clasificador. El balance de clases según la división del apartado anterior era la que se muestra en la Figura 18.

Tras los resultados del Cuadro 6, se puede observar que el desbalance es demasiado grande, por lo que es necesario aplicar técnicas de balance de datos [45] para entrenar el clasificador correctamente.

Se decide dividir el balance de datos en varias etapas: un filtrado previo de la cantidad de datos normales, un aumento artificial de las muestras de la clase 'defecto' y una modificación de la importancia de las muestras al aplicar la

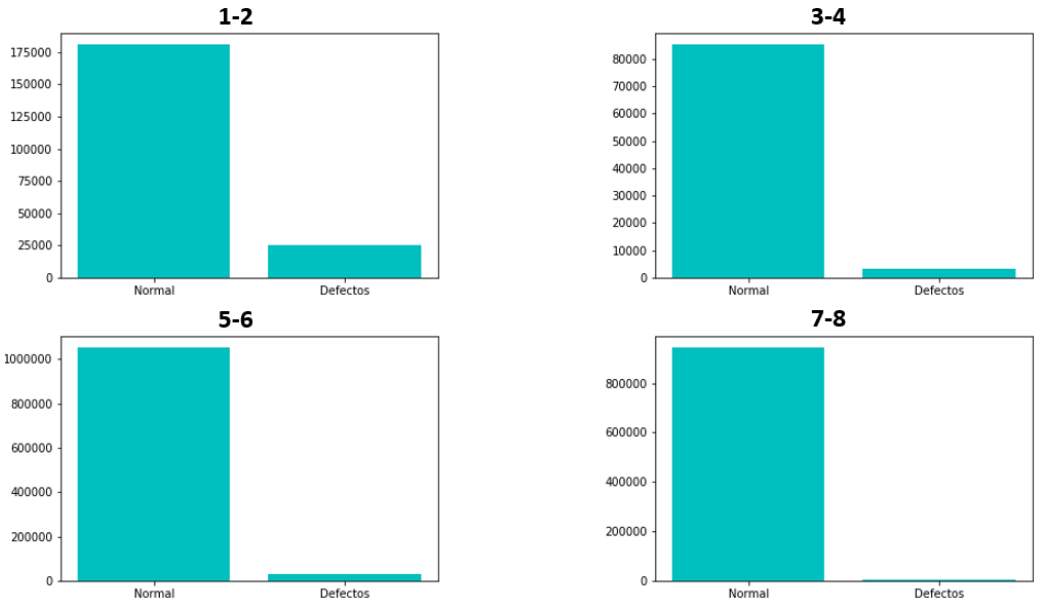


Figura 18: Desbalance de clases inicial.

	S12	**S34**	**S56**	**S78**
Normales	87.65 %	96.07 %	97.36 %	99.72 %
Defectos	12.34 %	3.97 %	2.63 %	0.28 %

Cuadro 6: Porcentaje puntos normales/defectos por pares de sensores.

función de coste con el objetivo de dar más importancia a las muestras menos frecuentes (defectos).
La generación de muestras artificiales se realiza añadiendo un ruido gaussiano a cada muestra original.

$$\mathbf{f}_{l,n}^{noise} = \mathbf{f}_{l,n} + \mathbf{N}(0, \nu) \tag{31}$$

Para cada muestra original se generan 5 nuevas muestras $\mathbf{f}_{l,n}^{noise}$ adicionales. El número de puntos normales se limita a $\times 2$ veces el número de puntos de defecto, de forma que el nuevo balance de datos es el siguiente:

Como se puede ver en el Cuadro 7, el desbalance se ha reducido pero sigue siendo significativo, por lo que se decide aplicar una modulación de la importancia de las muestras de entrenamiento. Considerando N como el número

	S12	**S34**	**S56**	**S78**
Normales	66.6 %	66.6 %	66.6 %	66.6 %
Defectos	33.3 %	33.3 %	33.3 %	33.3 %

Cuadro 7: Porcentaje puntos normales/defectos tras aumento de datos por pares de sensores.

total de muestras de entrenamiento, siendo n^{normal} el número de muestras de la clase "normal", el peso de cada muestra durante el entrenamiento se define como:

$$W_{normal} = \frac{N}{2 * n^{normal}} \tag{32}$$

De forma análoga, considerando $n^{defecto}$ como el número de muestras de la clase defecto, el peso de cada muestra de dicha clase sera:

$$W_{defecto} = \frac{N}{2 * n^{defecto}} \tag{33}$$

Estos pesos se aplican a la hora calcula la función de coste, de forma que cada muestra de la clase defecto es considerada como $\frac{W_{defecto}}{W_{normal}}$ veces una muestra de la clase normal.
Finalmente, se combinan los datos de todos los sensores para generar un único dataset y **entrenar un único modelo** tanto para el enfoque supervisados como no supervisado. Esto se realiza debido a que, como se muestra en el Cuadro 6, no se dispone de una cantidad suficiente de datos de defectos en todos los sensores para entrenar un modelo lo suficientemente robusto.

2.4.2. Métricas de evaluación

Al igual que las etapas de extracción y preparado de los datos, una buena selección de las métricas de evaluación es clave a la hora de determinar con exactitud el rendimiento del modelo.
Como ya se ha comentado previamente, la particularidad de este problema reside en que la cantidad de falsos negativos tiene que ser próxima a 0 ya que una pieza defectuosa clasificada como pieza normal podría llegar a tener graves consecuencias. Por este motivo se decide realizar la evaluación en base

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

Figura 19: Matriz de confusión.

a la matriz de confusión.

A partir de los parámetros de la figura 19 se obtienen una serie de métricas que son adecuadas para la evaluación de problemas de clasificación con clases desbalanceadas:

- La *precisión* indica el número de defectos clasificados correctamente en función del total de muestras clasificadas como defecto.

$$\text{Precisión} = \frac{\text{VP}}{\text{VP} + \text{FP}} \tag{34}$$

- La *sensibilidad* (TPR) indica el número de defectos clasificados correctamente en función del número total de defectos.

$$\text{Sensibilidad} = \frac{\text{VP}}{\text{VP} + \text{FN}} \tag{35}$$

- El *Ratio de Verdaderos Negativos* (TNR) indica el número de puntos normales clasificados correctamente en función del número total de puntos clasificados como normales.

$$\text{TNR} = \frac{\text{VN}}{\text{VN} + \text{FN}} \tag{36}$$

- La métrica *F1* es una media harmónica de la precisión y la sensibilidad. Aunque es posible aplicar pesos adicionales para dar más importancia

a uno de los dos términos, se ha decidido utilizar un peso de 0.5 para cada métrica.

$$F1 = 2 * \frac{\text{Precisión x Sensibilidad}}{\text{Precisión + Sensibilidad}} \tag{37}$$

- Área debajo de la curva ROC (AUC). La curva ROC es una curva de rendimiento para problemas de clasificación en función del valor del umbral. La curva representa la probabilidad y el AUC representa la capacidad del modelo de distinguir entre las clases.

Según los objetivos propuestos en 1, se puede redefinir la función objetivo como:

$$\text{máx}\, Precisión\, t.q.\, Sensibilidad > 1 - \epsilon \tag{38}$$

2.4.3. Optimización de umbral

En el caso de los clasificadores, el valor devuelto por el modelo está comprendido en el rango [0,1], de forma que se considera que la salida del modelo es la probabilidad de la clase 1 (defecto). En este apartado se definen los métodos seguidos para determinar el punto de corte para la decisión defecto / no defecto.
Se utilizan dos curvas distintas para definir el umbral en el caso de los clasificadores: la figura 21 muestra las curvas ROC [46] y la figura 22 muestra las curvas precisión/sensibilidad (PR) [47]. Aunque de forma general puedan usarse ambas curvas, en casos donde el desbalance de datos es grande, los resultados en ambas curvas pueden ser distintos. La diferencia principal es que las curvas ROC se mantienen constantes independientemente de la probabilidad base, mientras que las curvas PR son más utiles en la práctica para los problemas del tipo 'aguja en un pajar' donde una clase es más interesante que la otra [48].
A partir de los resultados de la figura 21, se selecciona el umbral que optimice la función:

$$\text{máx}(tpr - (1 - fpr)) \tag{39}$$

En el caso de las curvas PR se selecciona el umbral de forma que maximice la métrica f1:

$$\text{máx}(f1) \tag{40}$$

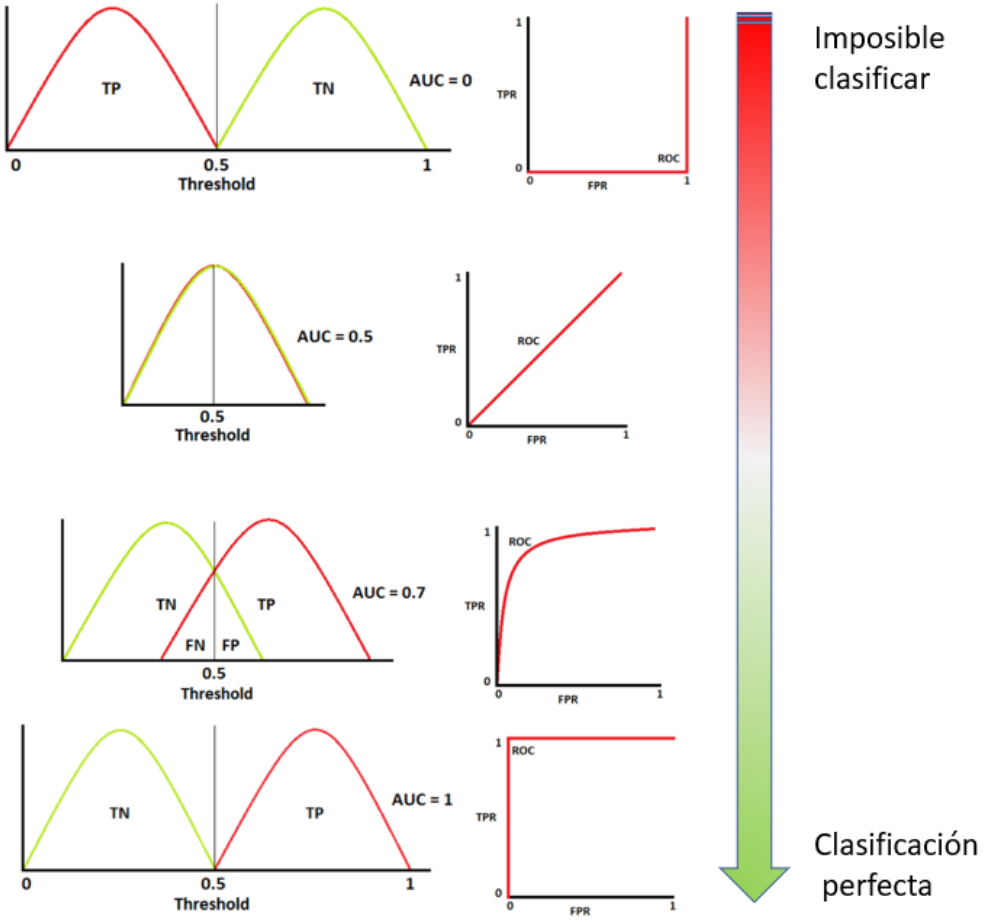


Figura 20: Capacidad del modelo de distinguir entre clases en función del AUC.

En el caso de los modelos que utilizan una estructura de autoencoder, la selección del umbral es diferente ya que no disponemos de puntos de la clase "defecto". Por este motivo se selecciona el umbral con el objetivo de ser capaz de detectar correctamente el **95 %** de los puntos normales. Siendo f_{val} un conjunto de muestras normales no vistas previamente por el modelo, se selecciona el umbral th de forma que:

$$th = percentile_{95}(f_{val}) \tag{41}$$

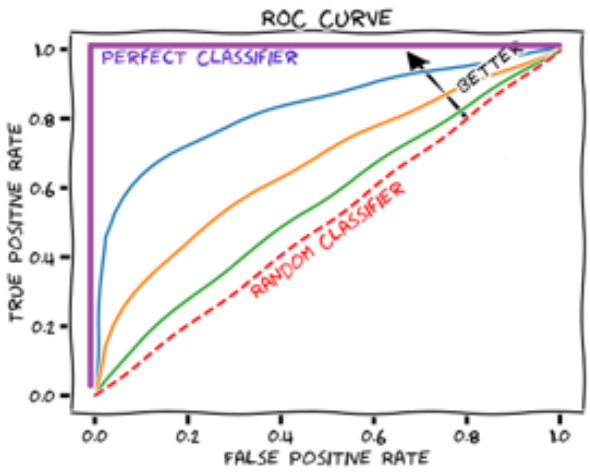


Figura 21: Ejemplo de curva ROC.

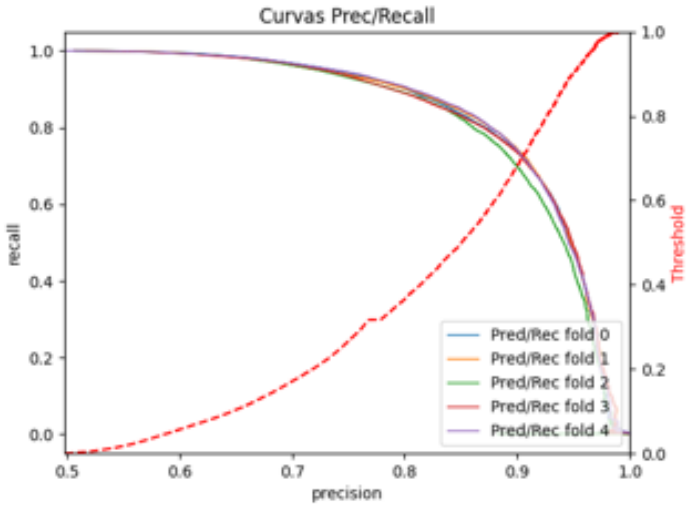


Figura 22: Ejemplo de curva PR.

2.5. Procesamiento de la salida del modelo

Como paso previo a la predicción a nivel de pinza (OK/No OK), es necesario realizar una limpieza de la predicción a nivel de píxel. Patrones como

bordes de la pieza, letras o marcas del bebedero son zonas que el modelo suele detectar como defecto pese a ser parte de la pieza (FP). La pieza se escanea desde una posición constante, de forma que estos falsos positivos se encuentran siempre situados en las mismas zonas de la imagen. Se propone el siguiente procedimiento para solucionar este problema:

1. Se genera una predicción de una pieza sin defectos. Será necesario generar una máscara por cada referencia a procesar.
2. Se aplica un filtro de dilatación ya que, aunque la pieza siempre es escaneada en la misma posición, pequeñas variaciones en la posición, podrían modificar ligeramente la posición de los patrones mencionados previamente.
3. A la predicción generada se le resta la predicción de referencia.

Una vez descartadas las zonas de falsos positivos, se aplica el umbralizado (ver sección 2.4.3) con el objetivo de obtener una máscara binaria.
Con el objetivo de eliminar posibles píxeles aislados detectados como defectos, se aplica un filtro de apertura. El tamaño del kernel de este filtro determina el tamaño mínimo del defecto a detectar. Finalmente, si algún píxel sigue estando clasificado como defecto tras este post-procesamiento, se considera que es un defecto real y por tanto la pieza se considera defectuosa.

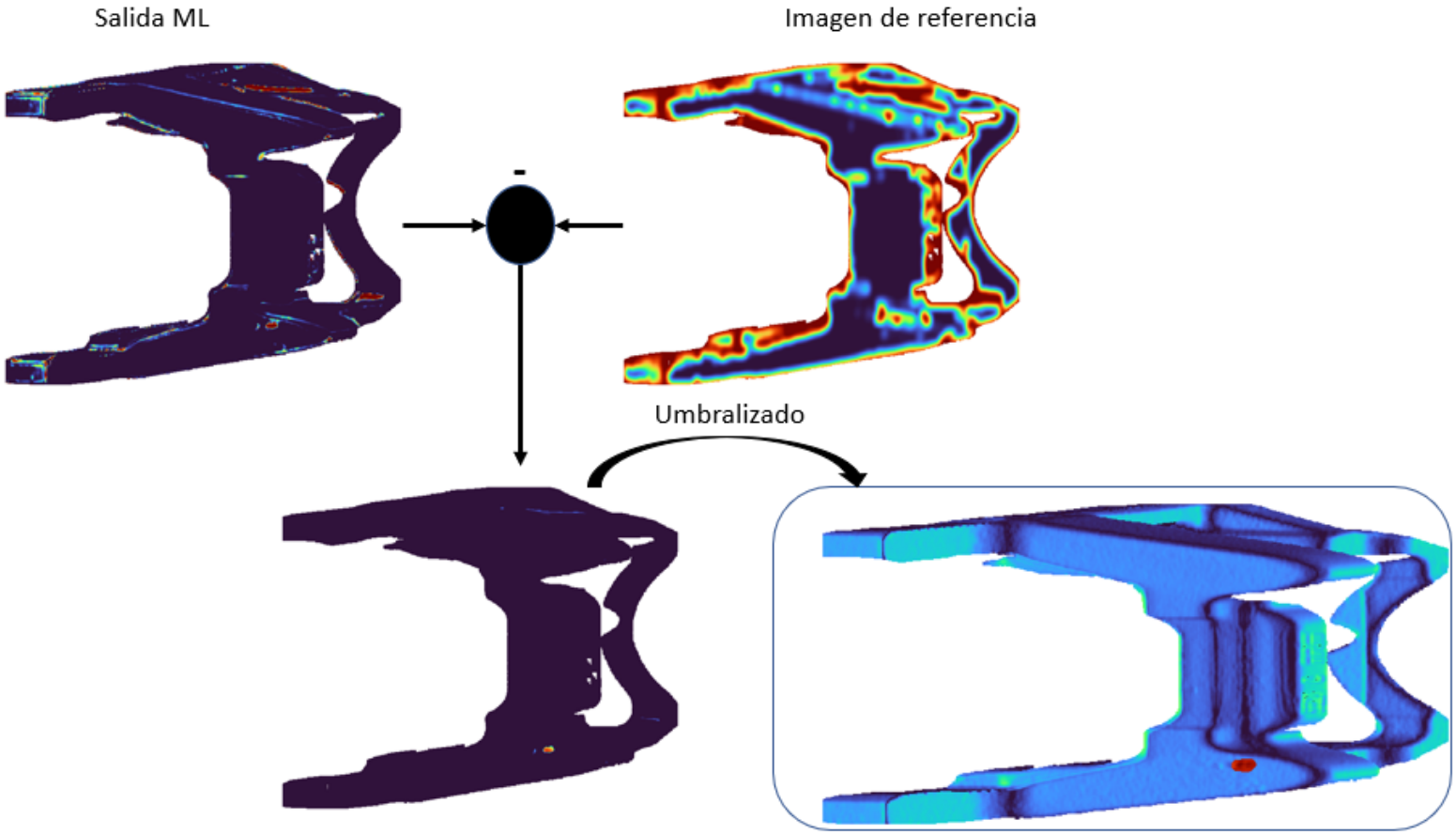


Figura 23: Ejemplo completo de procesamiento de ML.

2.6. Integración en C++

Uno de los mayores retos del presente proyecto reside en el despliegue del mismo en los sistemas embebidos ubicados en el proceso. Los modelos de ML tendrán que adaptarse para ejecutarse junto con el programa principal. Dicho programa se ejecutará en una línea de producción, disponiendo de tiempo limitado para el análisis (escaneo-extracción de características-inferencia del modelo-selección de defectos), por lo que se implementa en el lenguaje c++. Pese a que los modelos podrían ejecutarse desde la nube, se decide intergrarlos junto con el resto del programa para cumplir con los requisitos de tiempo real y evitar que la máquina dependa de tener conexión a internet.
Debido a que el preprocesamiento y entrenamiento de los modelos se realiza en Python, es necesario definir un método de conversión de los modelos, así como una librería para implementar estos modelos en c++. Este proceso se muestra gráficamente en la Figura 24 y cuenta con las siguientes fases:

1. *Preprocesamiento* : Se extraen las características, se eliminan los valores nulos y se normalizan los datos.
2. *Entrenamiento* : Se entrenan los modelos mediante la librería de Tensorflow para Python. Al finalizar el entrenamiento se guarda el modelo en un archivo con formato *.h5*.
3. *Conversión* : Se convierte el archivo .h5 a formato .pb. Aunque la librería dispone de funciones para realizar esta conversión. Es necesario crear una función personalizada que genere el archivo .pb con el formato de Tensorflow 1.
4. *Carga del modelo* : Para realizar la inferencia se utiliza la API de Tensorflow para c++ [49]. Una vez inicializada la estructura del modelo se cargan los pesos mediante el archivo .pb generado previamente. Junto con el modelo, es necesario introducir una serie de parámetros de configuración que se introducen mediante un archivo XML. Los parámetros se almacenan en estructuras dentro de cada una de las clases desarrolladas.
5. Inferencia : Finalmente se replica el proceso de extracción de características y predicción del modelo desde c++.

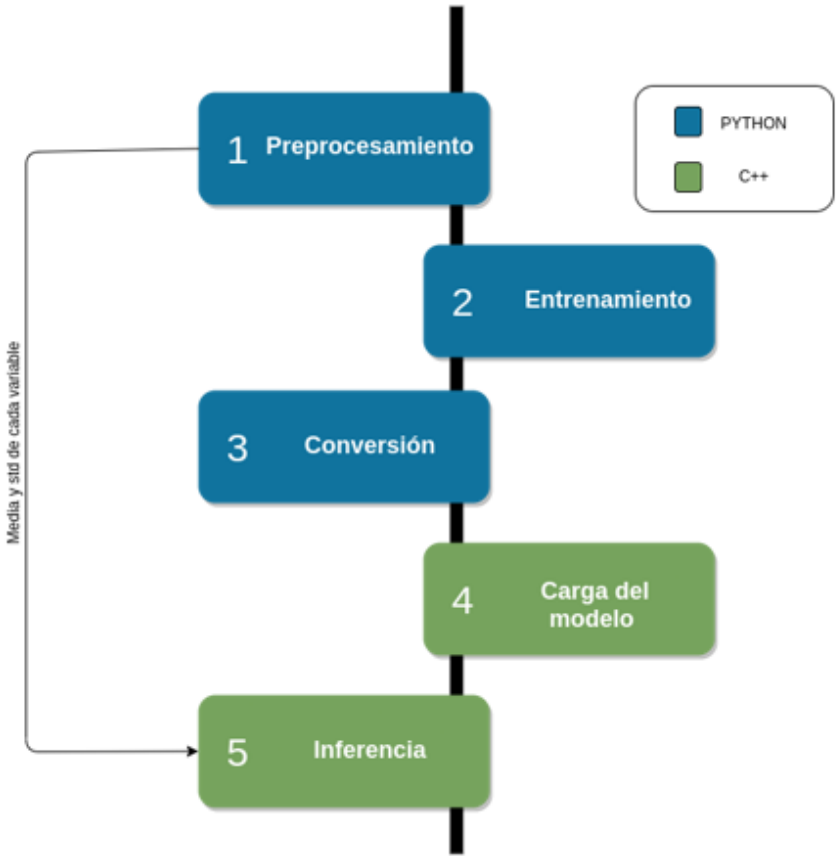


Figura 24: Pipeline de implementación de los modelos.

DSIplus dispone de un conjunto de librerías propias para el desarrollo de aplicaciones de visión artificial en c++ pero, debido a que no dispone de experiencia implementando modelos de ML, será necesario desarrollar una librería que recoja todas las funciones necesarias para la ejecución de estos modelos, desde la extracción de características, hasta la inferencia de los modelos. La librería se desarrollará según el paradigma de programación orientada a objetos (OOP). Los detalles de implementación se incluyen en el apéndice A.

3. Resultados

En esta sección se expondrán los resultados obtenidos tras la aplicación de las técnicas previamente mencionadas.
Pese a que, como ya se ha comentado previamente, el preprocesamiento de los datos se realiza por cada pareja de sensores y, a largo plazo se dispondra de suficientes datos para configurar cuatro modelos independientes, en la fecha de la redacción del documento se plantea el entrenamiento de un único modelo agrupando todos los datos disponibles. Una vez entrenado el modelo se realizará una evaluación de las métricas para cada pareja de sensores de forma individual.

3.1. Métodos supervisados

3.1.1. Selección del tamaño de ventana

Para la selección del tamaño óptimo de ventana se configura un modelo (no óptimo) con el que se han conseguido buenos resultados en pruebas previas.

Parámetro	**Valor**
L	2
N	[512,128,64]
Función de activación	['relu','relu','relu','sigmoid']
Optimizador	'adam'

Cuadro 8: Estructura del modelo para selección del tamaño de ventana.

A la vista de los resultados de la Figura 25, se decide realizar una evaluación cualitativa del tamaño de ventana 7 y el tamaño de ventana 9.
Tras realizar la evaluación visual de los resultados con ambos tamaños de ventana (ver Figura 26), *se decide utilizar tamaño de ventana 9 para el resto del proyecto* debido a que el número de zonas con FPs es menor.

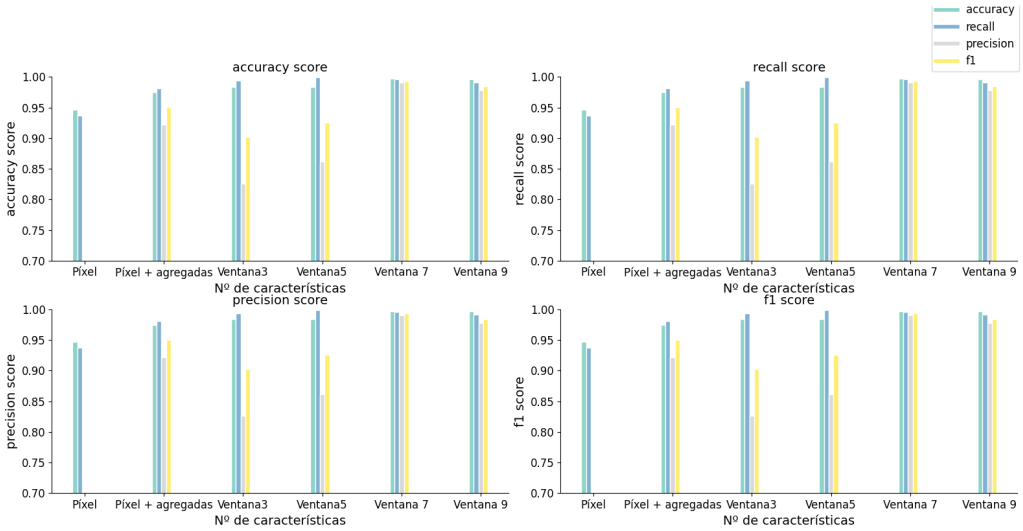


Figura 25: Comparación de métricas en función del vector de entrada.

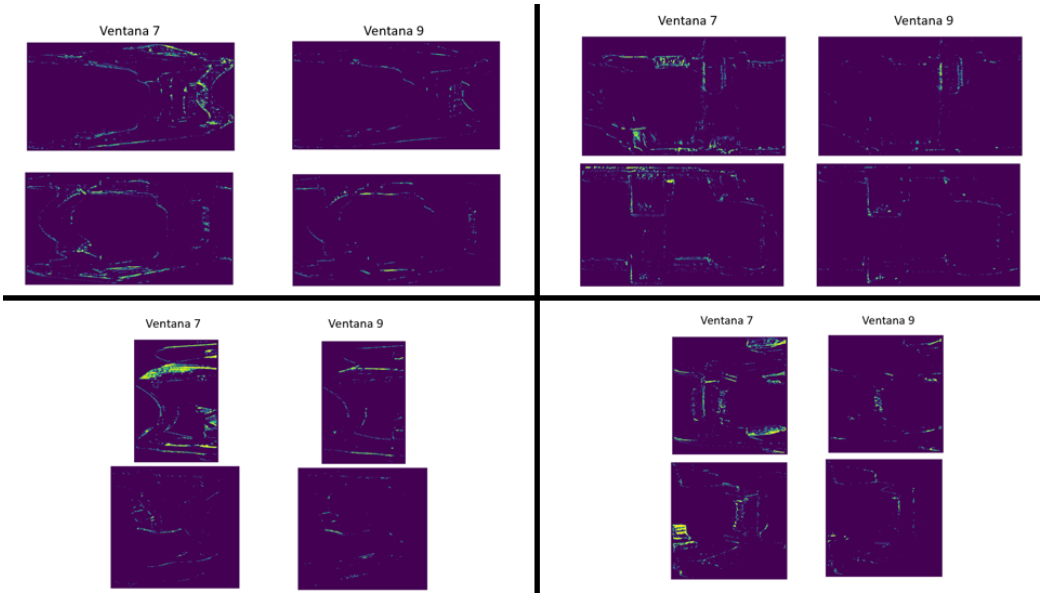


Figura 26: Comparación visual del tamaño de ventana en cada pareja de sensores para una pinza sin defectos.

3.1.2. Optimización del modelo

En primer lugar se realiza la optimización de los hiperparámetros (Ver sección 2.3.1) del modelo. Se prueban los siguientes valores:

- Nº de capas ocultas (L): [1,2,3,4,5,6,7,8,9,10].
- Nº de neuronas por capa oculta (N): [32,64,128,256,512,1024,2048].
- Función de activación : ['relu','tanh'].
- Ratio de "dropout": [0.05,0.1,0.15,0.2].
- Optimizador: ['adam','sgd','rmsprop'].
- Learning rate: [0.1,0.01,0.001].
- Tamaño del lote: [32,128,256,512,1024,2048].
- Nº de épocas: [1,3,10,30,100,300].

Tras realizar un total de 50 evaluaciones, se concluye que la estructura óptima del modelo es la siguiente:

Parámetro	**Valor**
L	8
N	[512,512,256,256,128,128,32,32]
Función de activación	['relu','relu','relu','relu','relu','relu','relu','sigmoid']
Dropout	[0.2,0.2,0.15,0.15,0.1,0.1,0.05,0.05]
Optimizador	'adam'
Learning rate	0.0001
Tamaño del lote	1024
Nº de épocas	50

Cuadro 9: Estructura óptima del modelo.

3.1.3. Resultados con modelo óptimo

Se entrenan los modelos con los parámetros más óptimos según la sección anterior. Para asegurar que el entrenamiento se produce con normalidad se visualizan las métricas 'accuracy' [50], así como el coste ('loss') a lo largo del entrenamiento.

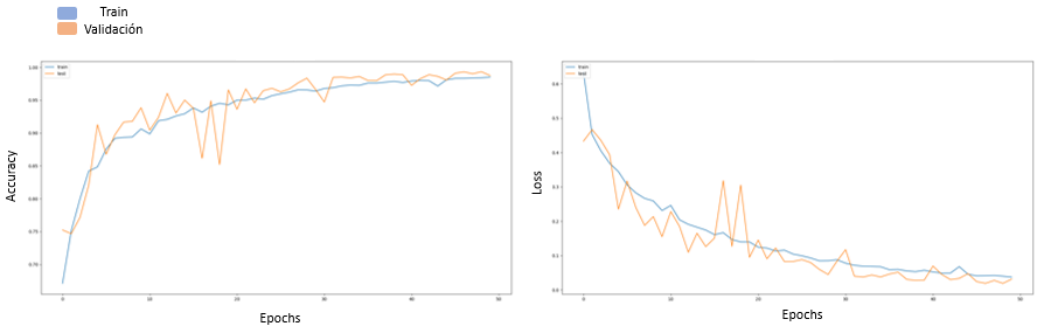
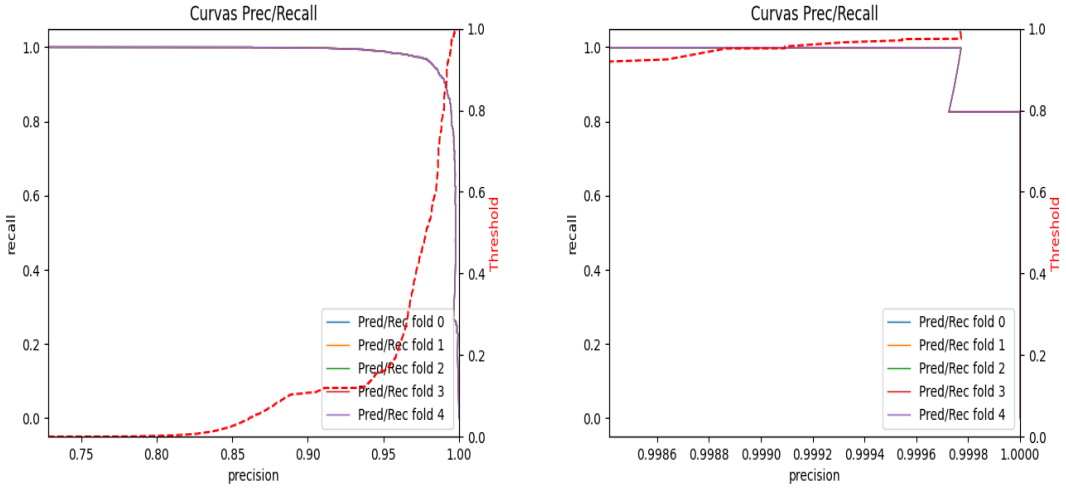


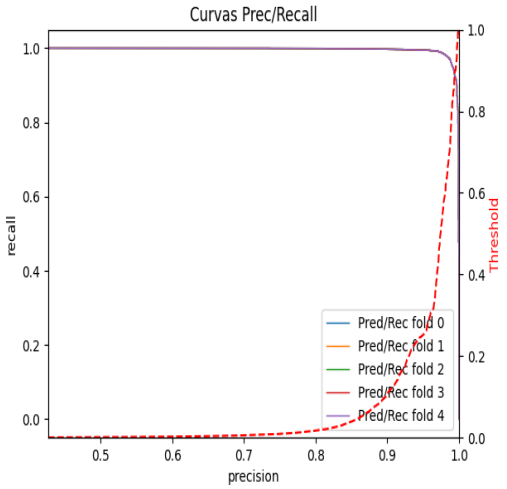
Figura 27: Evolución de métricas durante el entrenamiento para datos de entrenamiento y validación.

Como ya se ha comentado en el apartado 2.4.1 nos enfrentamos a un problema de clasificación con desbalance de datos por lo que se utilizan las curvas PR para seleccionar el valor óptimo del umbral.

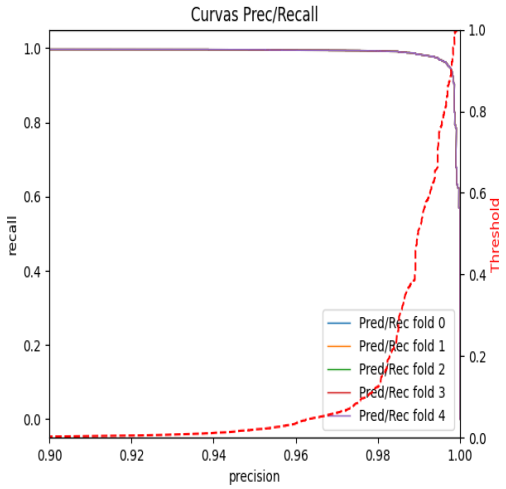


Sensores 1 y 2

Sensores 3 y 4



Sensores 5 y 6



Sensores 7 y 8

Sensores	Umbral
1 y 2	0.44
3 y 4	0.90
5 y 6	0.53
7 y 8	0.38

Cuadro 10: Umbral para cada sensor con clasificador.

Finalmente, se calcula la matriz de confusión. Se utiliza la fracción de los datos no utilizados durante el entrenamiento 'Test' para evaluar los modelos.

Sensores	VN	FP	FN	VP	F1
1 y 2	6538	113	56	3240	0.97
3 y 4	4711	70	36	2448	0.98
5 y 6	83395	487	257	41581	0.99
7 y 8	68948	1391	666	34475	0.97

Cuadro 11: Evaluación de los modelos.

En el apéndice B se incluye un análisis cualitativo de los modelos supervisados tanto para pinzas con defectos como para pinzas ok.

3.2. Métodos no supervisados

Tras los buenos resultados obtenidos entrenando un único modelo para todos los sensores, se decide seguir la misma filosofía para los métodos no supervisados pese a que en este caso se dispone de datos suficientes. Esto se realiza con el objetivo de generar un modelo robusto ante nuevos modelos de piezas y reducir el tiempo de entrenamiento de los modelos.

3.2.1. Comparación entre AE y VAE

En primer lugar es necesario definir cual es la arquitectura óptima para este problema (AE ó VAE). El proceso de entrenamiento de los modelos basados en autoencoders es lento, por lo que la comparación modelo a modelo entre ambas arquitecturas requiere demasiados recursos. Para esta prueba se decide usar una estructura sencilla, evitando el tiempo empleado en procesos de optimización:

- El encoder está formado por dos capas densas [1024,256] con función de activación 'tanh'.
- De forma similar, el decoder está formado por dos capas densas [256,1024] con una función de activación 'tanh'.
- El tamaño del espacio latente es 64.

Debido a que se dispone de un conjunto de datos de defectos usados para entrenar los clasificadores, se evalúa el rendimiento de ambas arquitecturas con un conjunto de datos normales y con un conjunto de defectos y se comparan los resultados en base a dos métricas:

- Accuracy en limpieza de puntos normales.
- Accuracy en detección de defectos.

Accuracy	AE	VAE
Normales	94.7 %	94.8 %
Defectos	47.8 %	57.5 %

Cuadro 12: Comparación de resultados entre AE y VAE.

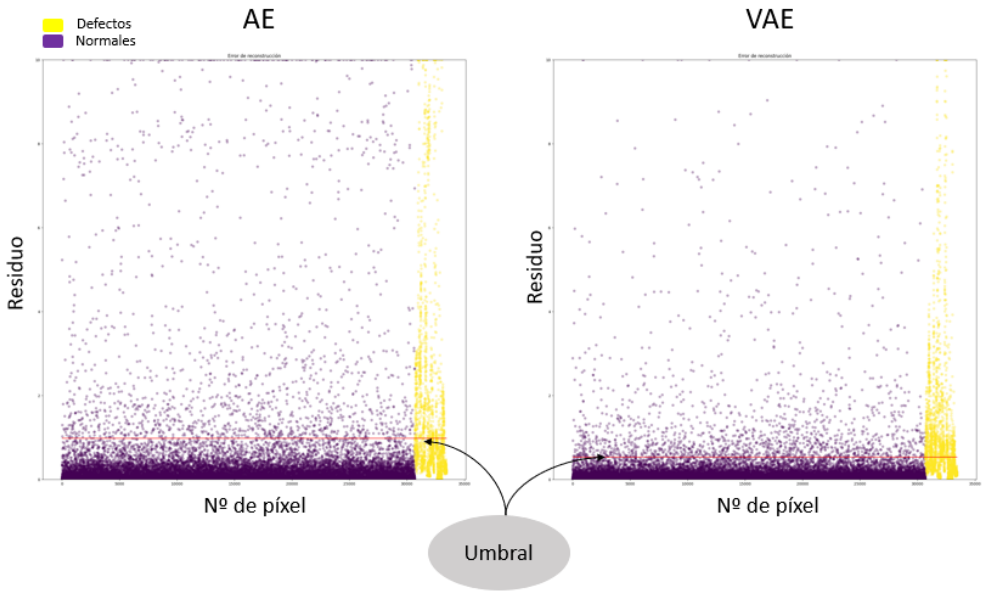


Figura 28: Comparación del error de reconstrucción entre AE y VAE calculado para todas las muestras de test.

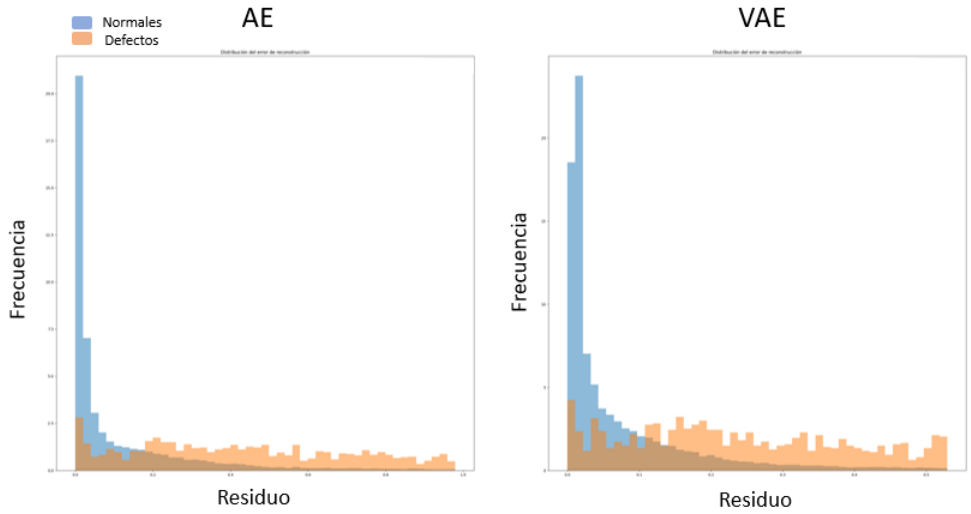


Figura 29: Comparación de la distribución del error de reconstrucción entre AE y VAE calculado para todas las muestras de test.

3.2.2. Optimización del modelo

Tras estos resultados, se decide descartar la arquitectura AE y entrenar los modelos utilizando una arquitectura VAE.
De forma análoga a los clasificadores, se realiza la optimización de la estructura de los autoencoders. Los diferentes parámetros evaluados son:

- Nº de capas ocultas (L): [1,2,3,4].
- Nº de neuronas por capa oculta (N): [32,64,128,256,512,1024].
- Función de activación en las capas ocultas : ['relu','tanh'].
- Tamaño del espacio latente: [4,8,16,32,64]
- Optimizador: ['adam','sgd','rmsprop'].
- Learning rate: [0.1,0.01,0.001].
- Tamaño del lote: [32,128,256,512,1024,2048].
- Nº de épocas: [1,3,10,30,100,300].

Tras realizar un total de 50 evaluaciones, se concluye que la estructura óptima del modelo es la siguiente:

Parámetro	**Valor**
L	3
N	[1024,512,256]
Función de activación	['tanh','tanh','tanh','sigmoid']
Tamaño del espacio latente	64
Optimizador	'adam'
Learning rate	0.001
Tamaño del lote	1024
Nº de épocas	30

Cuadro 13: Estructura óptima del modelo.

3.2.3. Resultados con modelo óptimo

Se entrenan los modelos modelos con los parámetros más óptimos según la sección anterior. Para asegurar que el entrenamiento se produce con normalidad, en la figura 30 se visualiza el error cuadrático medio junto con los diferentes parámetros que forman el coste total.

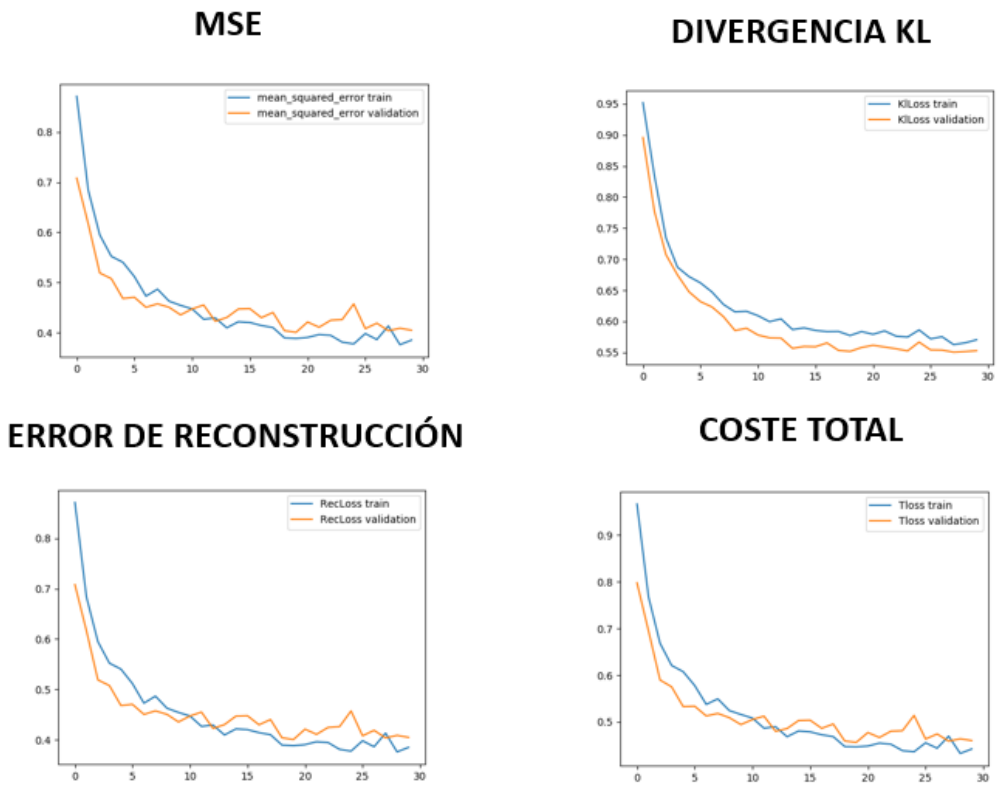


Figura 30: Evolución de métricas durante el entrenamiento.

En la figura 31 se puede ver el valor del residuo para cada muestra de test evaluada. La línea roja horizontal muestra el valor del threshold para asegurar la detección del 95 % de los puntos normales. Idealmente, los puntos amarillos deberían situarse todos por encima de esa línea. Por otro lado, en la figura 32, se muestra la distribución del residuo en las muestras de test.

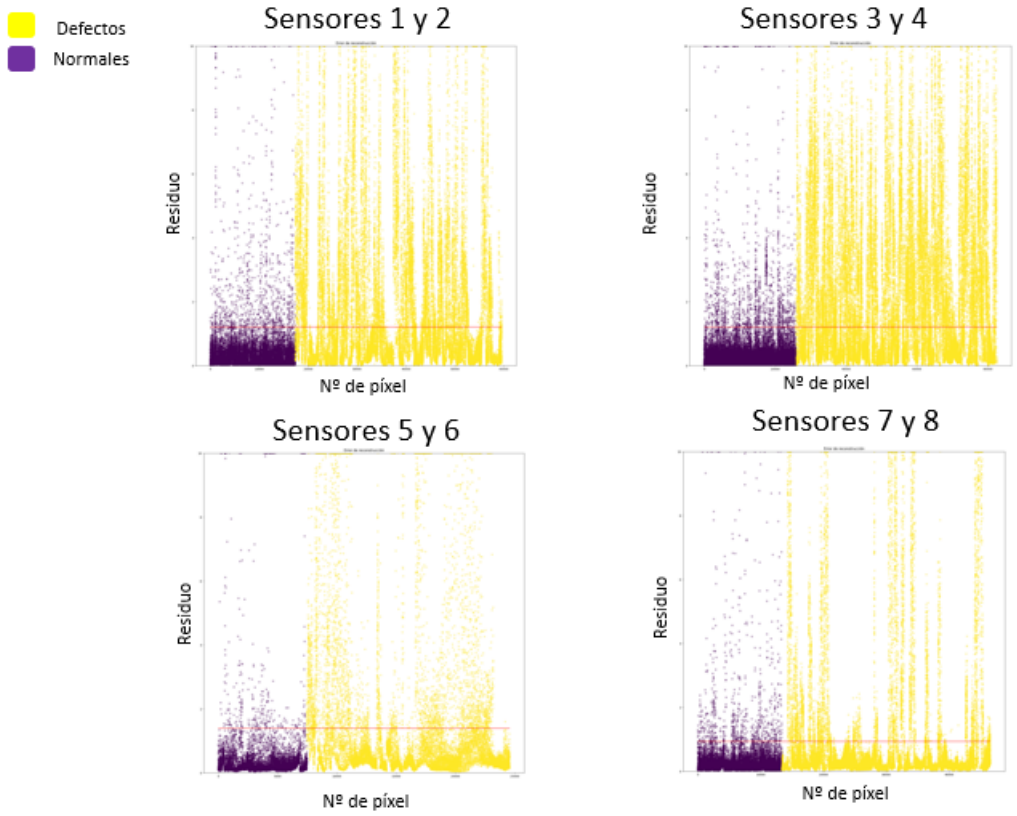


Figura 31: Comparación de valor del residuo entre puntos normales y de defecto.

De acuerdo a la sección 2.4.3 se calcula el valor óptimo del umbral:
Finalmente, se realiza la evaluación sobre los conjuntos de datos normales y

Sensores	Umbral
1 y 2	1.22
3 y 4	1.19
5 y 6	1.36
7 y 8	0.95

Cuadro 14: Umbral para cada sensor con autoencoder.

de tests para cada modelo.
En el apéndice C se incluye un análisis cualitativo de los modelos no super-

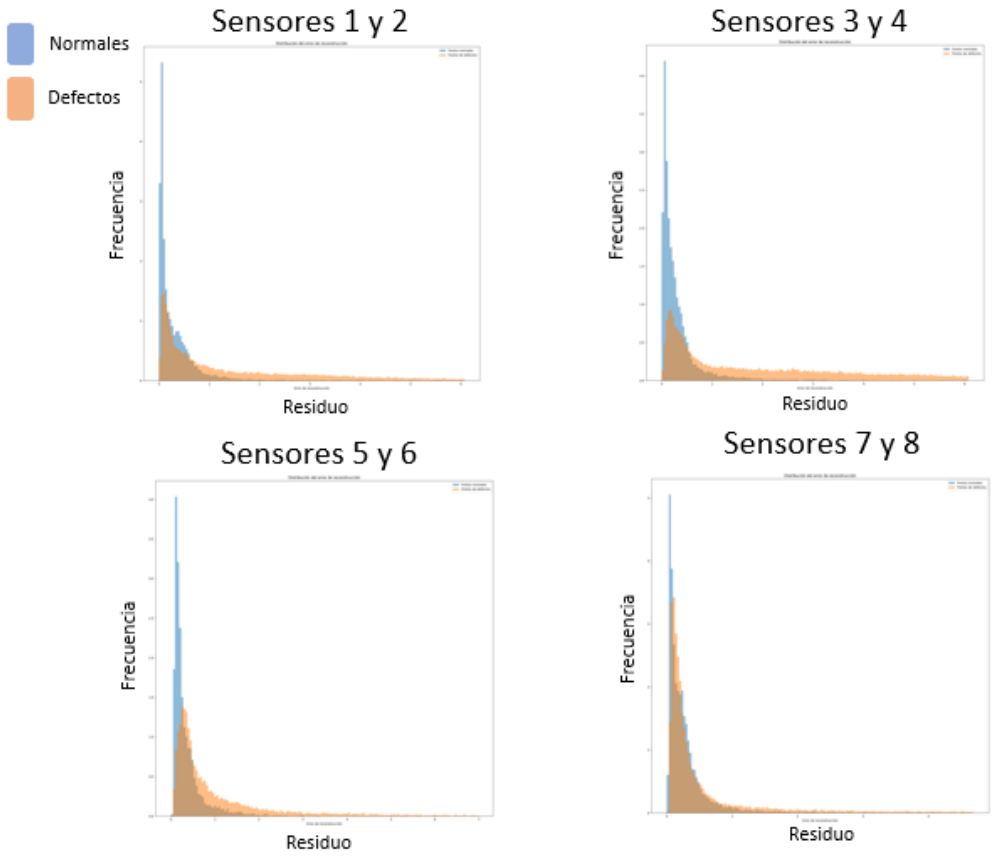


Figura 32: Comparación de valor del residuo entre puntos normales y de defecto.

Sensor	Normales	Defectos
1 y 2	94.3 %	43.1 %
3 y 4	94.0 %	56.3 %
5 y 6	95.7 %	29.8 %
7 y 8	94.4 %	22.8 %

Cuadro 15: Precisión sobre puntos normales/defectos con autoencoder.

visados tanto para pinzas con defectos como para pinzas ok.

4. Discusión

4.1. Aplicación a nuevos proyectos

Uno de los objetivos de este trabajo es desarrollar un procedimiento estándar que permita aplicar los conocimientos adquiridos durante el desarrollo del presente trabajo para facilitar la detección de defectos en futuros proyectos de la empresa. Se evaluaron las técnicas en dos proyectos diferentes.

4.1.1. Resultados en otras piezas de fundición

Como se puede ver en la Figura 33, se trata de piezas similares a las del estudio original. Debido a su geometría compleja, deben ser escaneadas desde varias posiciones (ver Figura 34) para poder realizar el análisis completo. El estudio se encuentra en fases iniciales por lo que el escaneo se realiza con un único sensor situado en un brazo robot, lo que permite ajustar la posición del escaneo. Dicho sensor dispone de una mayor resolución que el sensor original y, por tanto, una mayor cantidad de puntos disponibles.

Debido a que se trata de un estudio previo, no se dispone del tiempo necesario para realizar un etiquetado manual de los defectos por lo que se decide evaluar únicamente los métodos no supervisados.

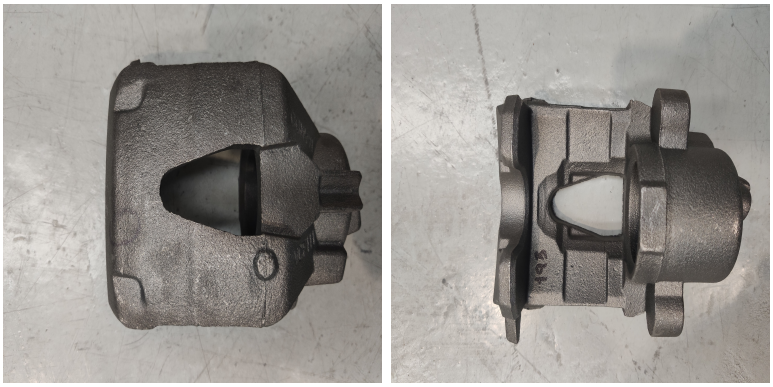


Figura 33: Vistas de pieza de fundición.

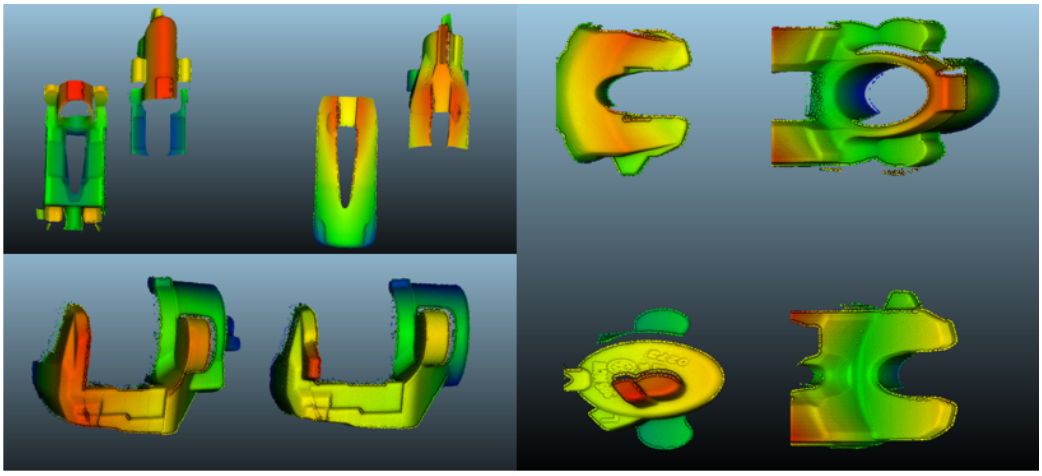


Figura 34: Ejemplo de escaneo en piezas de geometría compleja.

Al no disponer de un conjunto de datos de defectos para evaluar el rendimiento de los modelos, no es posible realizar un análisis cuantitativo de los resultados. El objetivo de las pruebas iniciales es asegurarse de que los modelos son capaces de detectar los defectos, dejando de lado la aparación de falsos positivos. Con el objetivo de facilitar el análisis de los resultados, se decide eliminar el postprocesamiento, generando únicamente dos imágenes: la salida del modelo (residuo) y la imagen binarizada aplicando un umbral a la salida del modelo. A continuación se muestran algunos de los resultados obtenidos durante la primera prueba.

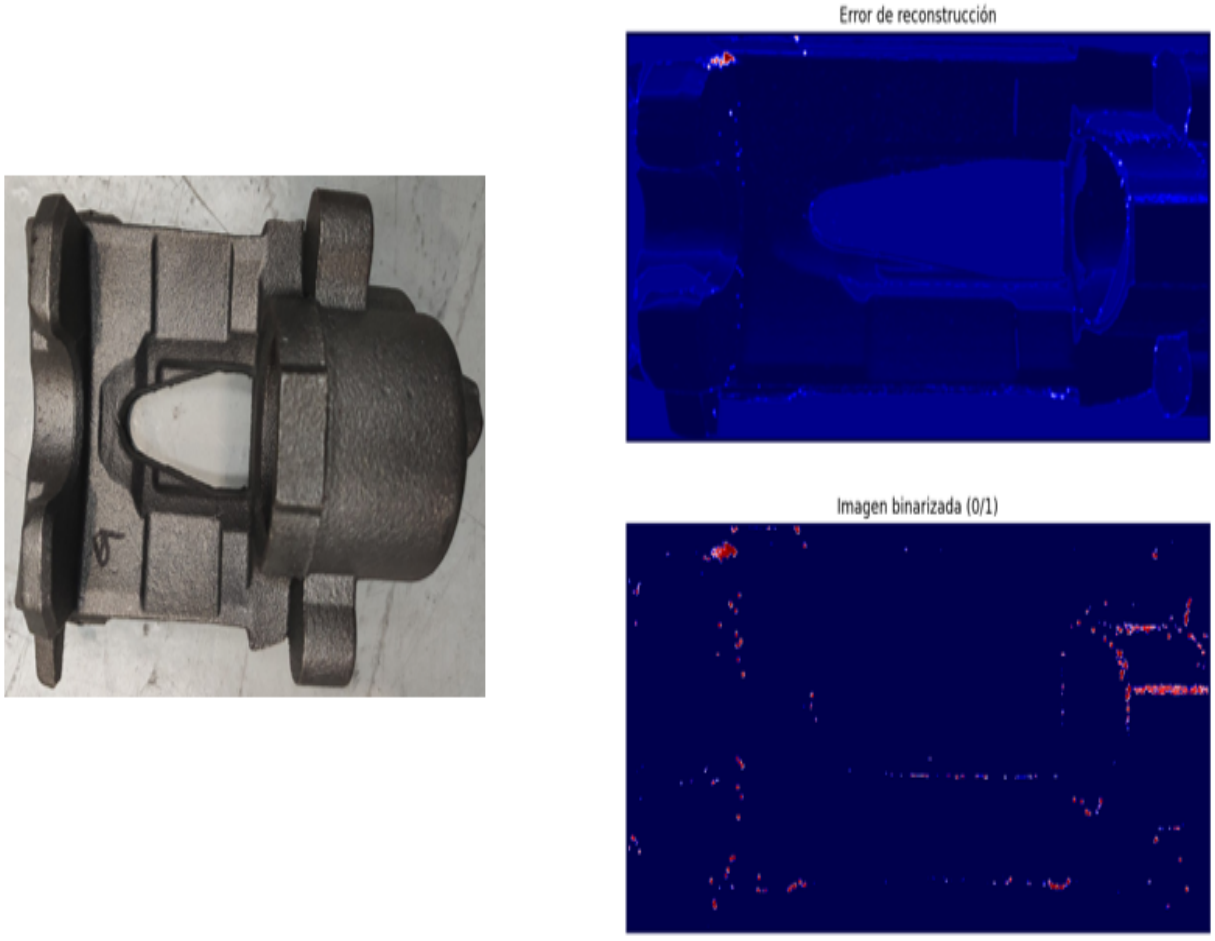


Figura 35: Ejemplo 1 de procesamiento en la prueba inicial sin defecto.

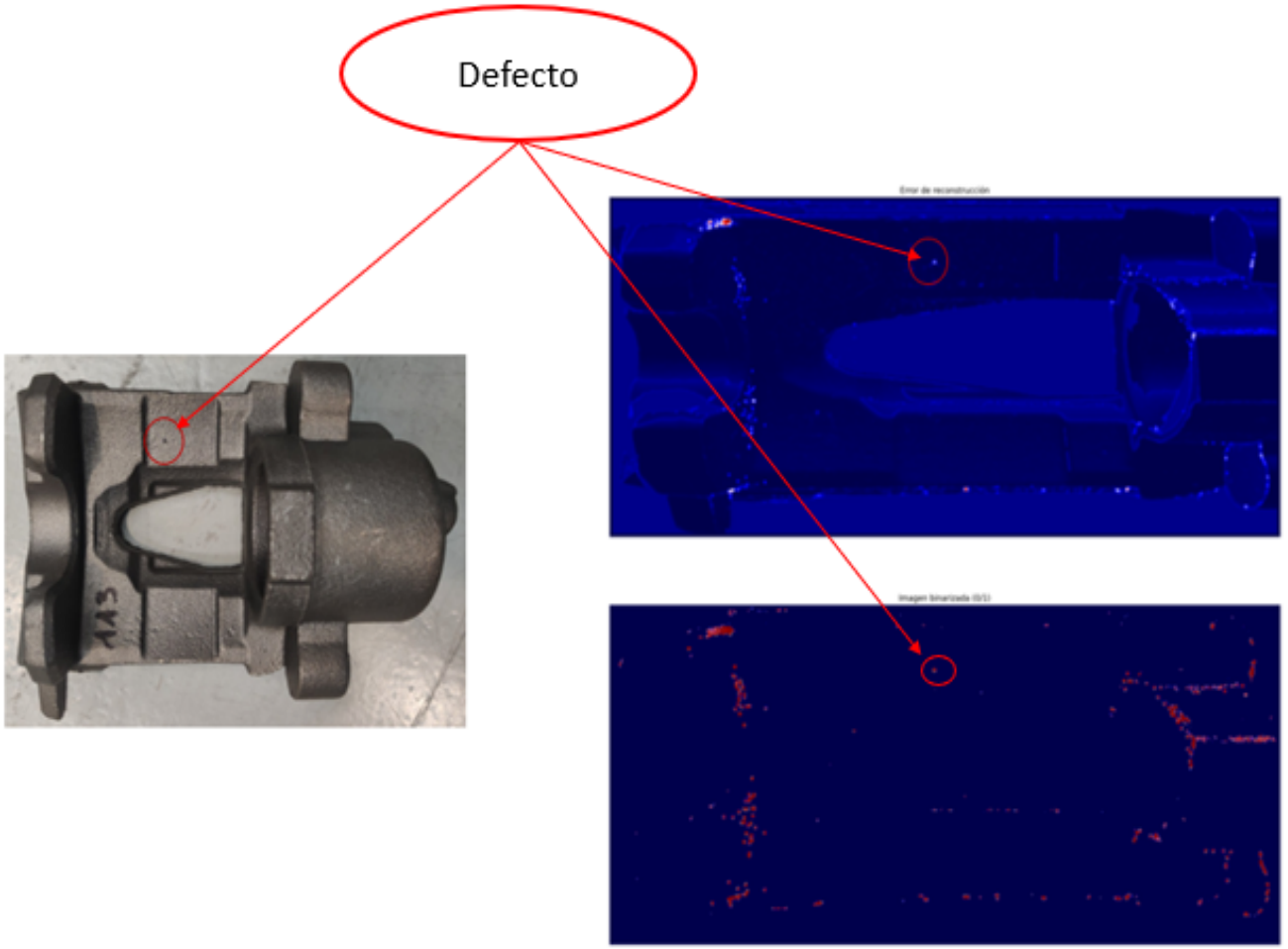


Figura 36: Ejemplo 2 de procesamiento en la prueba inicial con defecto.

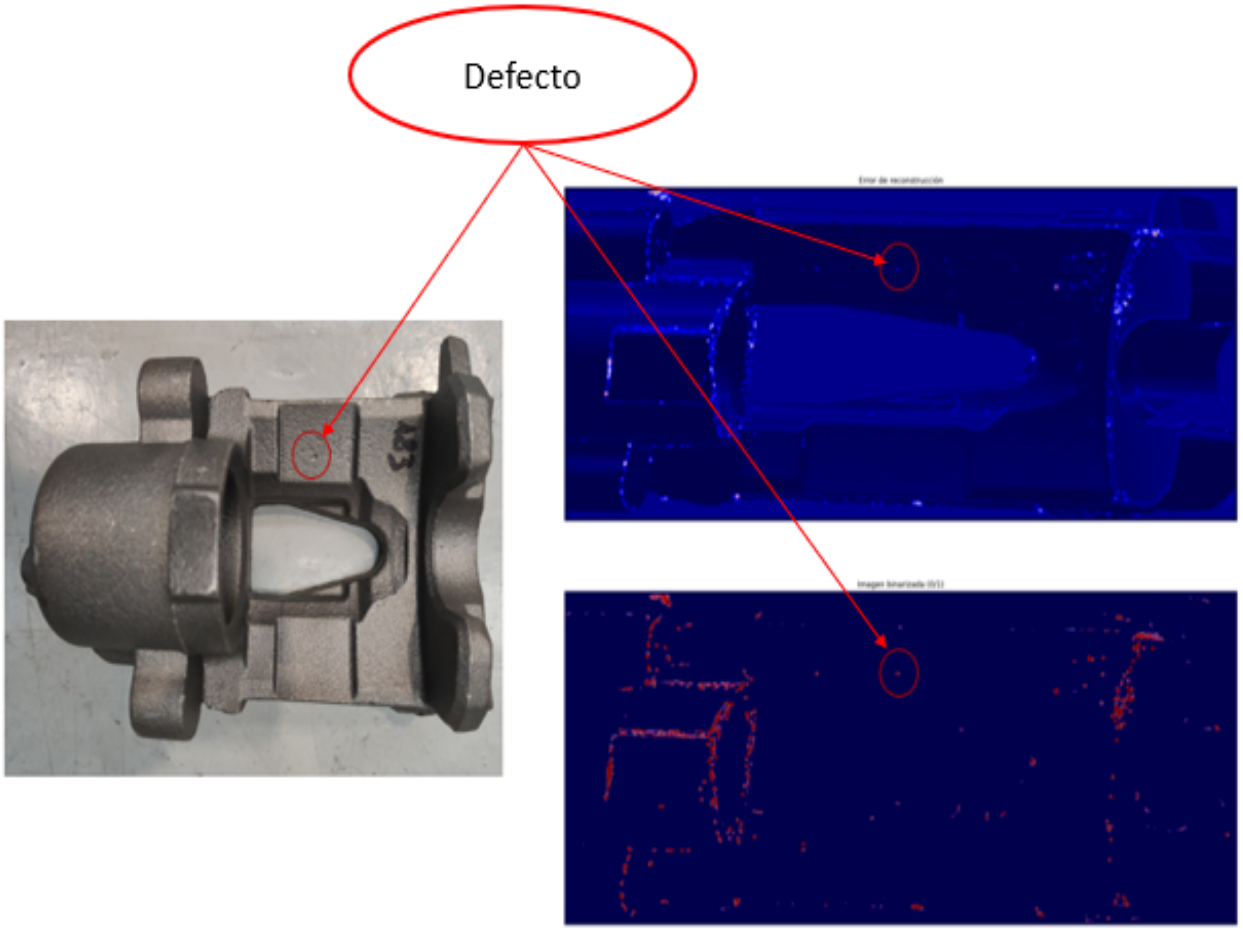


Figura 37: Ejemplo 3 de procesamiento en la prueba inicial con defecto.

Como se puede ver en las Figuras 35, 36 y 37, los modelos son capaces de detectar los defectos. Es evidente que la cantidad de falsos positivos es alta, pero, tras un análisis de las características de entrada (ver Figura 38), se puede observar que la mayoría de falsos positivos se deben a zonas donde se pierde información y/o bordes de la pieza (Puntos blancos en las imágenes). Finalmente, con el objetivo de demostrar al cliente de DSIplus la capacidad de eliminar los falsos positivos, se realiza un postprocesamiento similar al desarrollado en el apartado 2.5. Como se puede observar en la figura 39, la cantidad de falsos positivos ha disminuido drásticamente, manteniendo la detección de los defectos.
Tras estas pruebas se concluye, de forma cualitativa, que las técnicas supervisadas son capaces de detectar los defectos en piezas similares a las del estudio original.

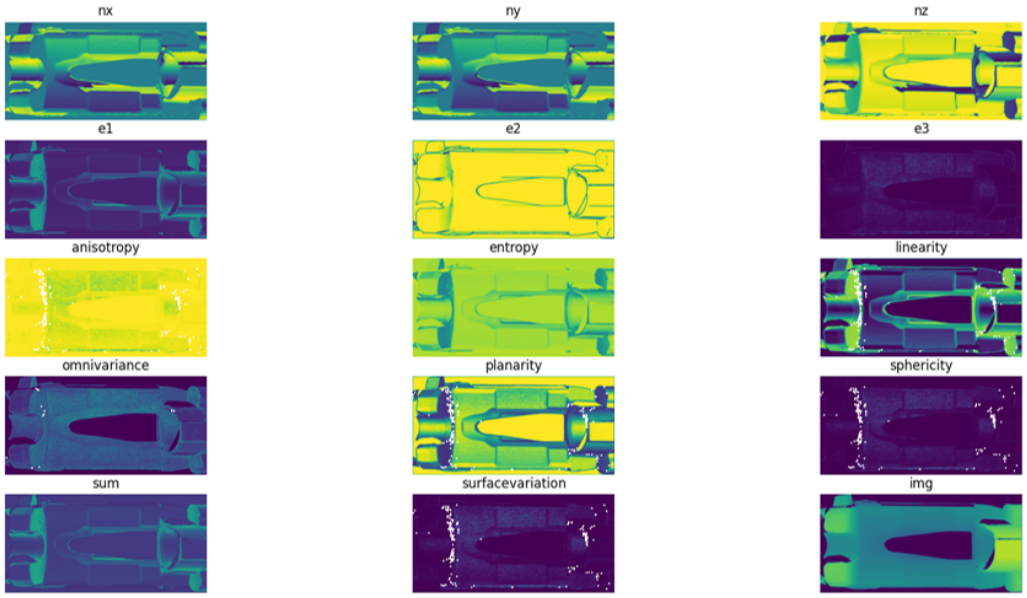


Figura 38: Exploración de características de entrada del autoencoder.

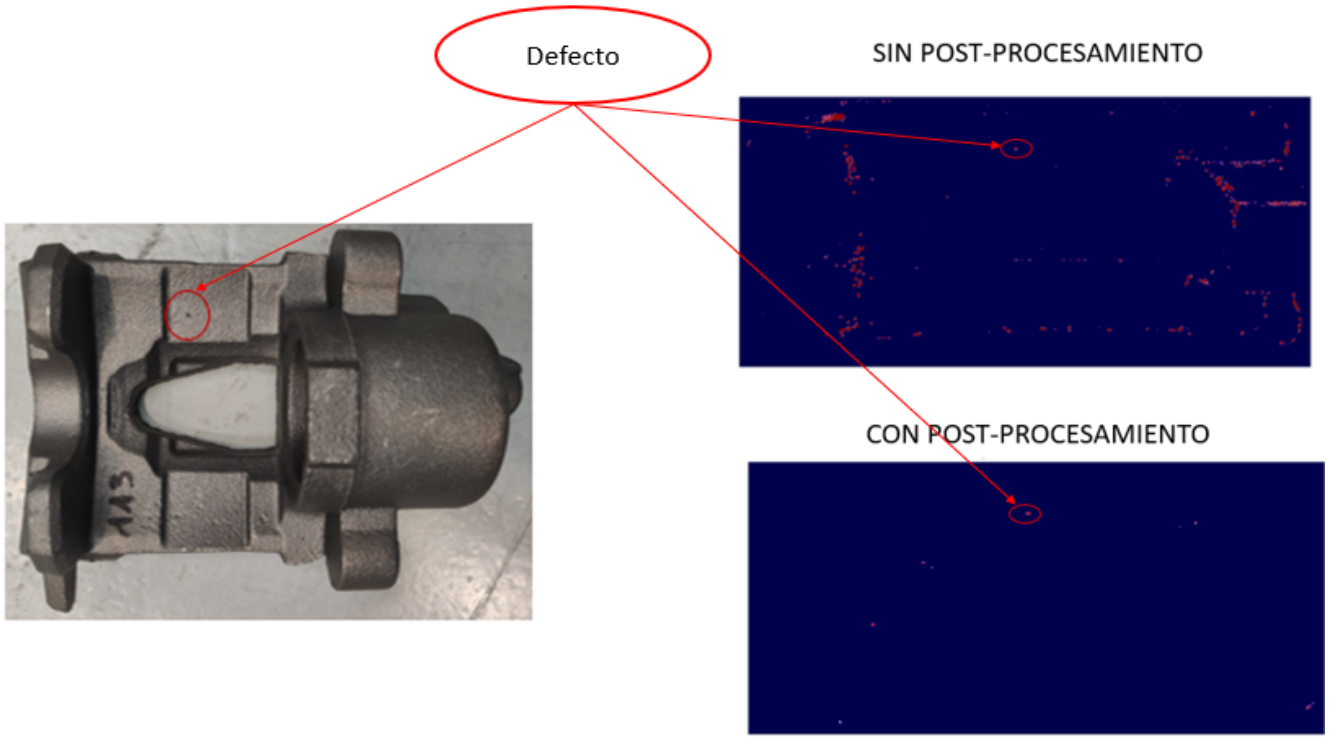


Figura 39: Ejemplo 2 de procesamiento con post-procesamiento.

4.1.2. Resultados en piezas de estampado

El análisis se realiza sobre puertas de coche. En este caso la geometría de la pieza es más sencilla, por lo que el análisis se realiza con un único sensor que se desplaza por la superficie de la puerta. Al contrario que en las piezas de fundición, donde los defectos eran pequeños, en este caso nos encontramos con defectos que pueden llegar a ocupar el ancho de la puerta. Por este motivo se decide descartar el análisis a nivel de píxel.

Al tratarse de una superficie lisa, las proyecciones 2D de los defectos son

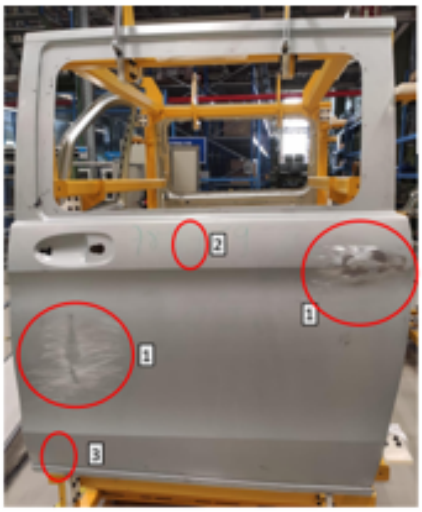


Figura 40: Ejemplo de pieza de estampado defectuosa.

similares a su forma original, por lo que se plantea una extracción de características a partir del contorno de la mancha.

Así como en el análisis píxel a píxel se dispone de suficientes datos para entrenar los modelos, al realizar el análisis a nivel de mancha, la cantidad de defectos disponibles es escasa por lo que se usaron ténicas de aumento de datos para entrenar los modelos (ver Ecuación 31).

Como punto de partida, se calcula un vector de características sobre los contornos identificados

$$\mathbf{f}^{(i)} = [f_1^{(i)}, f_2^{(i)}, \ldots, f_m^{(i)}] \tag{42}$$

donde las $f_m^{(i)}$ corresponden a las propiedades del contorno (*perímetro, área, diámetro equivalente, orientación...*) calculadas para la i-ésima mancha junto con las propiedades de Fourier. Como planteamiento alternativo, se explora una extracción de características basadas en la magnitud y la orientación del gradiente en el área y en el contorno de la mancha [51]. Aunque esta extracción parece prometedora, su complejidad computacional hace que sea difícil de implementar en tiempo real.

De igual manera que en las piezas de fundición, los métodos supervisados fueron los primeros en ser probados. En este caso las etiquetas necesarias para el entrenamiento también fueron asignadas manualmente a cada una de las manchas encontradas. Tras el etiquetado, los métodos propuestos fueron:

- Árboles de decisión.
- SVC.
- Algoritmos 'one class'.

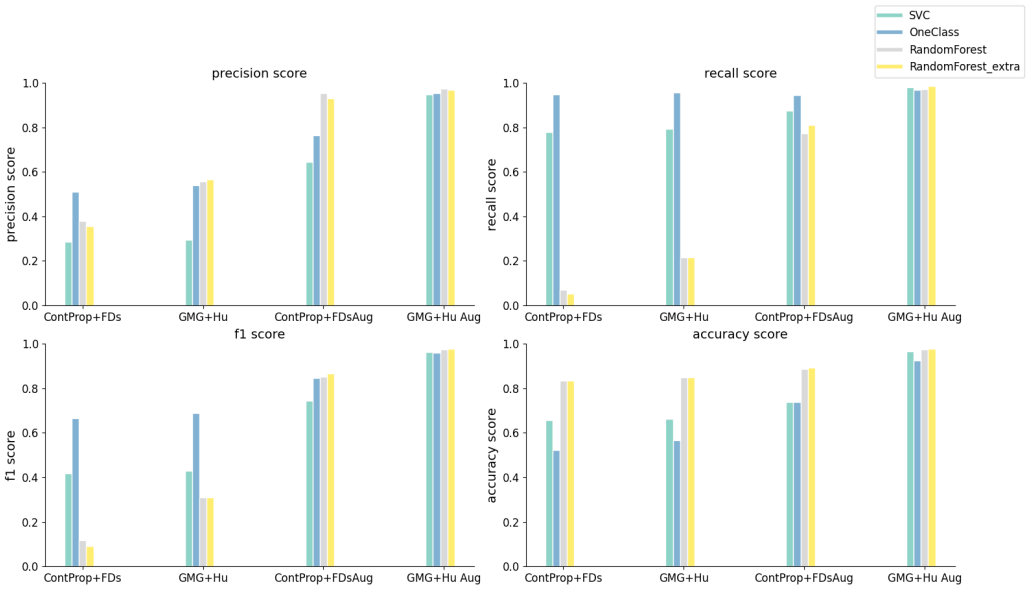


Figura 41: Evaluación de la clasificación.

Actualmente, se está trabajando en intentar resolver problemas relacionados

con el preprocesamiento de los datos, debido a que el proceso de captura y recogida de las imágenes ha sufrido modificaciones a lo largo del proyecto, variando la resolución de las imágenes generadas.
Por otro lado, se están realizando las primeras pruebas con los clasificadores previamente mencionados. En la fecha de redacción de este documento el mejor modelo alcanza una precisión de detección de defectos en torno al 40 %. Se espera que una vez que se resuelvan los problemas de preprocesamiento y se disponga de suficientes datos de defecto, estos resultados mejoren. Por otra parte, tras analizar cualitativamente los resultados sobre puertas reales, se ha observado un comportamiento similar a los métodos no supervisados del proyecto orginal, donde un defecto puede estar formado por varias manchas que se analizan de forma individual, por lo que solo será necesario clasificar como defecto una de ellas.

4.2. Conclusiones

El desarrollo de sistemas de ML en aplicaciones industriales, donde las condiciones de entorno pueden ser cambiantes a lo largo del tiempo resulta complejo.
Uno de los principales problemas durante el desarrollo del proyecto fue la cantidad de datos de defectos. Pese a que se disponía de una gran cantidad de piezas para realizar el estudio (500 piezas), la cantidad de defectos en las mismas no fue suficiente. En la fecha de redacción del documento, la máquina se encuentra en la fábrica del cliente y, aunque no se encuentra en línea, se espera seguir etiquetando defectos durante los próximos meses para conseguir un conjunto de datos lo suficientemente grande y balanceado en todos los sensores.
Uno de los principales requisitos a la hora de introducir una máquina nueva en una línea de producción es el tiempo de ciclo. Durante las pruebas realizadas se llegó a la conclusión de que los métodos no supervisados podían llegar a ser demasiado lentos, no siendo capaces de cumplir los requisitos temporales. Sin embargo, se pudo observar que, como era de esperar, su comportamiento ante nuevos defectos era mejor que en el caso de los métodos supervisados.
A la hora de realizar los test de los métodos no supervisados hay que tener en cuenta que el rendimiento a nivel de píxel no es un rendimiento real. Se ha comprobado que valor de precisión de 40 % a nivel de píxel no equivale a 40 % a nivel de blob. Como se puede ver en la figura 42, en ambos casos

se obtiene un 40 % de precisión en la detección de defectos a nivel de píxel, pero en un caso se obtiene un 40 % de precisión a nivel de blob y en el otro se obtiene un 100 %. Tras las primeras evaluaciones, se observó que el comportamiento del modelo se asemeja más a la imagen c) que a la imagen b). Este mismo fenómeno ocurre en la aparición de falsos positivos por lo que resulta complicado eliminarlos por completo.

En la fecha de la redacción del documento, el proyecto está a la espera de

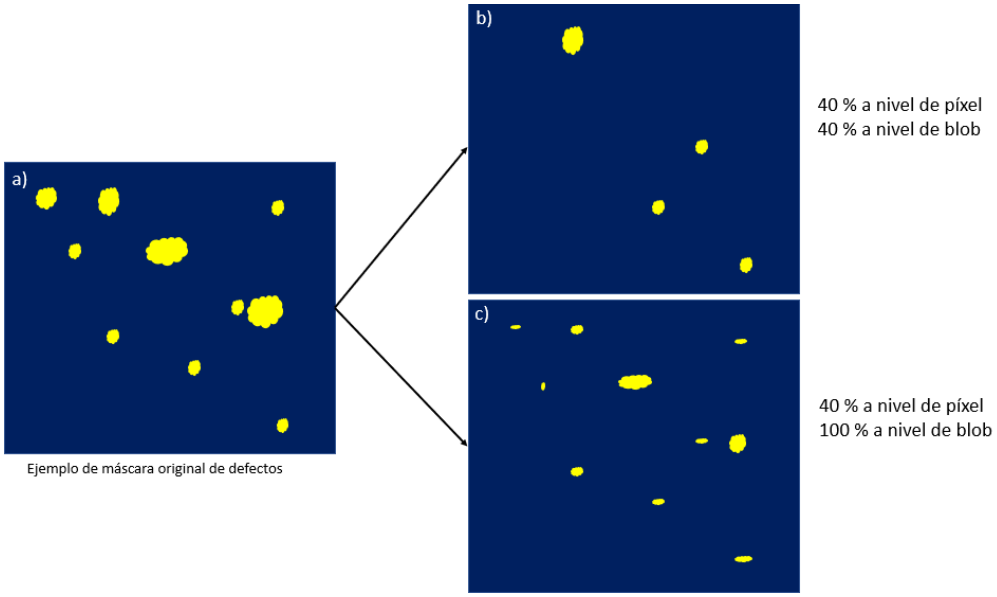


Figura 42: Ejemplo de comparación de precisión a nivel de píxel y a nivel de blob.

la homologación. En el Cuadro 16 se muestran los resultados tras la primera simulación de funcionamiento, donde se analizaron un total de 100 piezas. En

Métrica	**Porcentaje**
Falsos positivos	20 %
Detección de defectos	90 %

Cuadro 16: Métricas finales a nivel de pinza.

vista de estos resultados, se puede confirmar que la aplicación de algoritmos de ML ha mejorado el rendimiento inicial (ver Cuadro 1).

El otro gran problema al que el proyectante se tuvo que enfrentar fue el efecto del sobre-entrenamiento durante la generación de los modelos. A la vista de los resultados de la sección 3, los resultados sobre piezas reales deberían ser mejores a los obtenidos durante las pruebas empíricas. Esto se debe a que la distribución de los datos de entrenamiento y validación no recoge toda la distribución real de la pinza. Un motivo puede ser la limitación de puntos normales durante el entrenamiento de los modelos supervisados. A medida que se disponga de una mayor cantidad de defectos, se aumentará en proporción la cantidad de puntos normales de la pieza, recogiendo la totalidad de la misma.

El escaneo de las piezas usadas para el presente proyecto se realizó utilizando un prototipo de la máquina final. A la hora de montar la máquina final, se han movido ligeramente los sensores por problemas dimensionales, lo que hace que la distribución de los datos se vea ligeramente modificada, afectando al rendimiento final.

En cuanto a la aplicación a nuevos proyectos, se ha comprobado que en el caso de trabajar con piezas de naturaleza similar, como se ha visto en la sección 4.1.1, el procedimiento desarrollado en el presente trabajo puede ser fácilmente aplicado a piezas con geometrías distintas pero de la misma naturaleza. En el caso de piezas como las vistas en la sección 4.1.2, donde los defectos son demasiado grandes para ser analizados a nivel de pixel, los resultados obtenidos en las pruebas preliminares son esperanzadores y se espera que una vez se defina el método de preprocesamiento final y se disponga de una cantidad suficiente de datos, los resultados sobre piezas reales serán similares a los obtenidos durante los estudios previos.

Al margen de los resultados técnicos, el proyecto ha servido al proyectante para desarrollar su habilidades en el manejo de lenguajes como Python o c++, programación orientada a objetos y sobre todo a reforzar los conocimientos en todas las etapas de un proceso de ML, los cuales están estrechamente relacionados con el plan de estudios del máster cursado.

A falta de la puesta en línea de la máquina, este proyecto demuestra la viabilidad de técnologías como el ML para sustituir y/o complementar técnicas de visión artificial clásicas en un sector tan demandante como es el control de calidad a nivel industrial, por lo que servirá como base para el desarrollo de futuros proyectos dentro de DSIplus.

4.3. Líneas futuras de trabajo

Pese a que el trabajo desarrollado cumple con los objetivos propuestos en la sección 1.4, se plantean una serie de mejoras y alternativas a explorar en un futuro.

La totalidad del tiempo empleado en el procesamiento de los datos se enfoca en la extracción de características para introducir en el modelo. Durante los últimos años, las arquitecturas de redes convolucionales [52] han ganado popularidad para clasificación de imagen por su facilidad para encontrar relaciones en el espacio, por lo que parece una alternativa viable a las redes neuronales basadas en capas densas.

El proceso de etiquetado manual de datos de defectos es lento e ineficiente. Por otro lado, con varias decenas de piezas sin defecto, es posible conseguir un conjunto de datos de puntos normales lo suficientemente grande para entrenar modelos no supervisados. Debido a que la implementación de modelos no supervisados en tiempo real resulta complicada, sería interesante explorar la posibilidad de utilizar los métodos no supervisados para realizar el etiquetado de los datos de forma automática. De esta forma sería posible generar fácilmente un conjunto de datos lo suficientemente grande como para entrenar los métodos supervisados, que serán implementados en la máquina final. Una alternativa sería el uso de métodos semi-supervisados [25], los cuales permitan obtener modelos precisos con un menor número de datos etiquetados.

En el caso de las puertas de coche, en ocasiones un mismo defecto está asociado a varias manchas lo que provoca que el modelo procese el mismo fenómeno varias veces con entradas diferentes en cada una de ellas. Sería deseable que las manchas identificadas abarcasen toda la superficie del defecto real, de modo que el modelo sea capaz de entrenar con ejemplos cuya entrada cubra el area completa del defecto real, por lo que sería interesante mejorar el proceso de identificación de manchas.

Una alternativa a la extracción de características a nivel de blob es la exploración de redes convolucionales *Fully-convolutional* (FCN) pensadas especialmente para segmentación de imagen. En concreto se están realizando pruebas con la arquitectura *U-net* [53].

A. Diagrama de clases

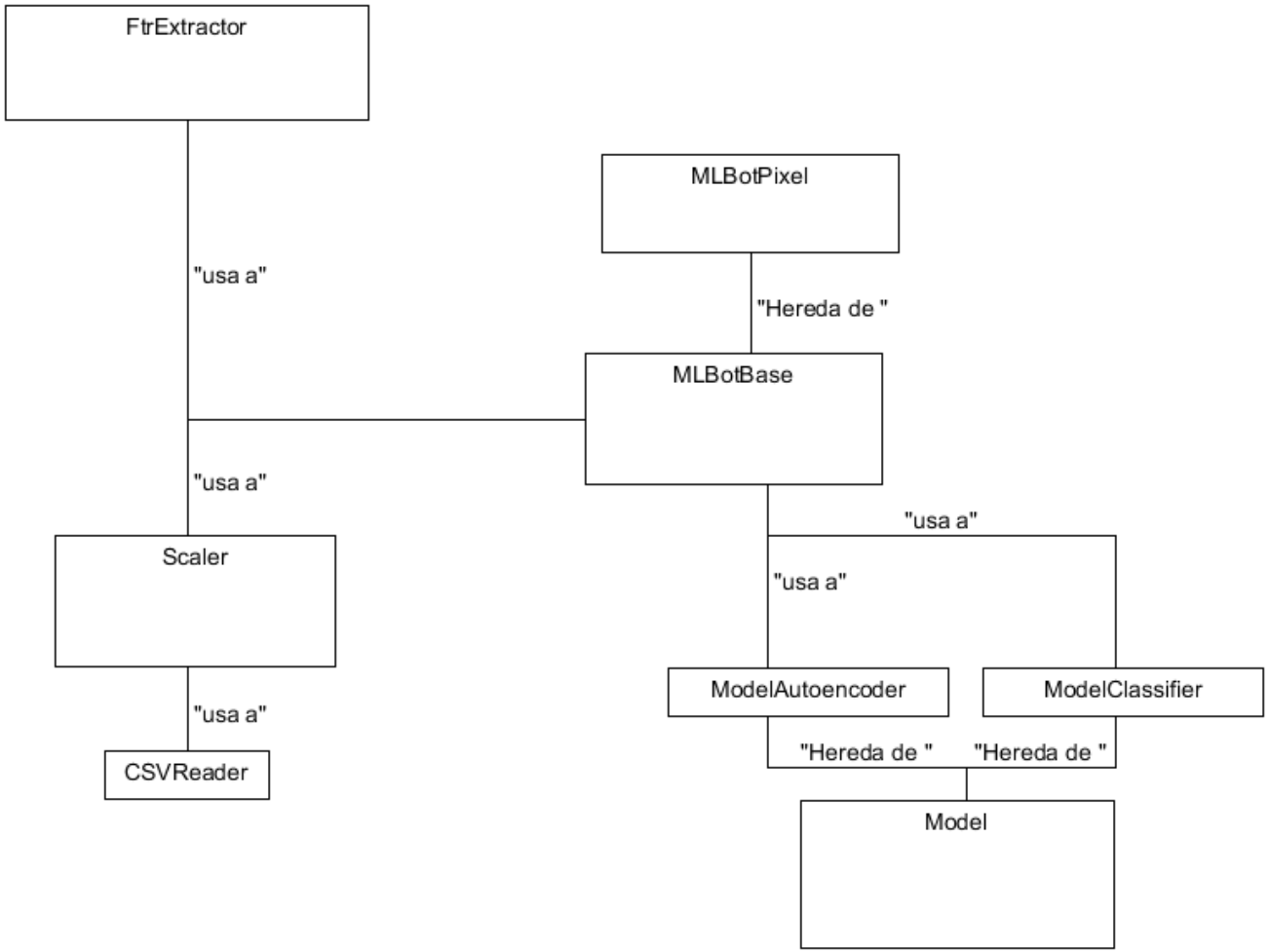


Figura 43: Diagrama de clases.

A.0.1. Clase CSVReader

La clase *CSVReader* se encarga de la lectura del archivo csv con los valores de la media y la desviación estándar de cada característica con el objetivo de realizar la normalización de los datos de entrada.

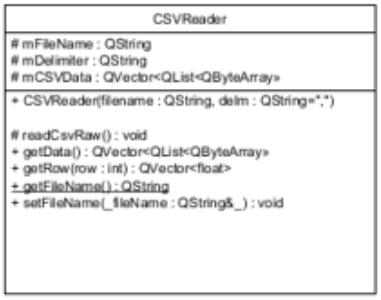


Figura 44: Clase CSVReader.

A.0.2. Clase Scaler

La clase *Scaler* se encarga de la normalización / desnormalización de los datos. Para su implementación se ha utilizado como referencia la clase *StandardScaler* disponible en scikit-learn [54].

A.0.3. Clase FtrExtractor

La clase *FtrExtractor* contiene los métodos necesarios para extraer y formatear las características de entrada del modelo, desde la selección de los puntos a procesar hasta la extracción del vector de entrada del modelo.
En este caso, los parámetros de configuración son escasos y opcionales, pero debido a que se trata de una clase de carácter general, se decide mantener el formato de las estructuras para facilitar las posibles ampliaciones que se

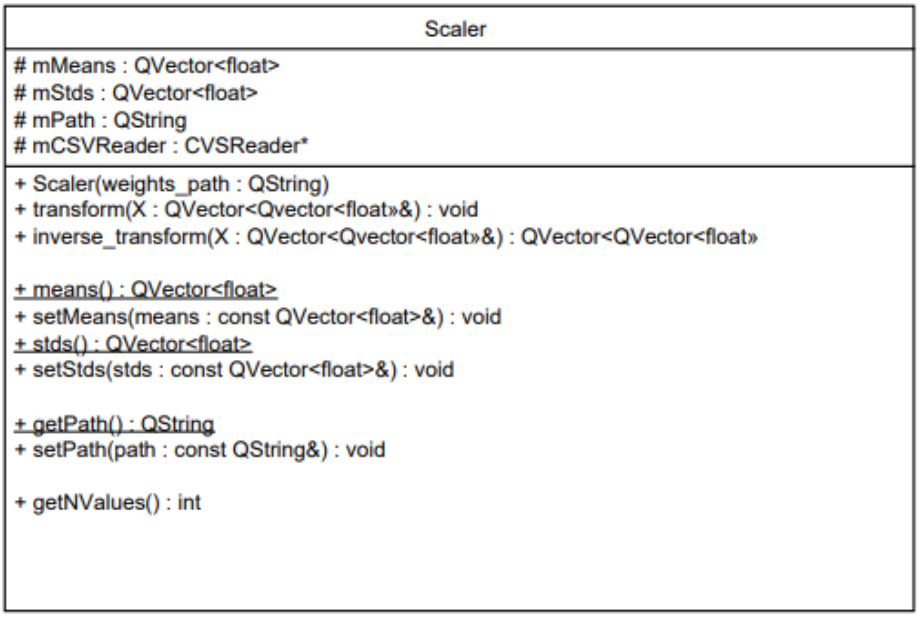


Figura 45: Clase Scaler.

realicen en el futuro.

- rootPath : Dirección donde se encuentran las imágenes de características.
- normalizeImg : Ofrece la posibilidad de normalizar cada imagen de forma independiente. Se utiliza en el caso de no disponer de la media y la desviación estándar de las características.

A.0.4. Clase Model

La clase Model se encarga de encapsular los métodos disponibles en la librería de Tensorflow. Debido a que los modelos de Tensorflow utilizan el tipo de dato "Tensor", se crean dos clases derivadas que se encargan de formatear la salida del modelo en función de las necesidades del problema. A día de hoy se han realizado implementaciones para modelos de tipo clasificador y de tipo

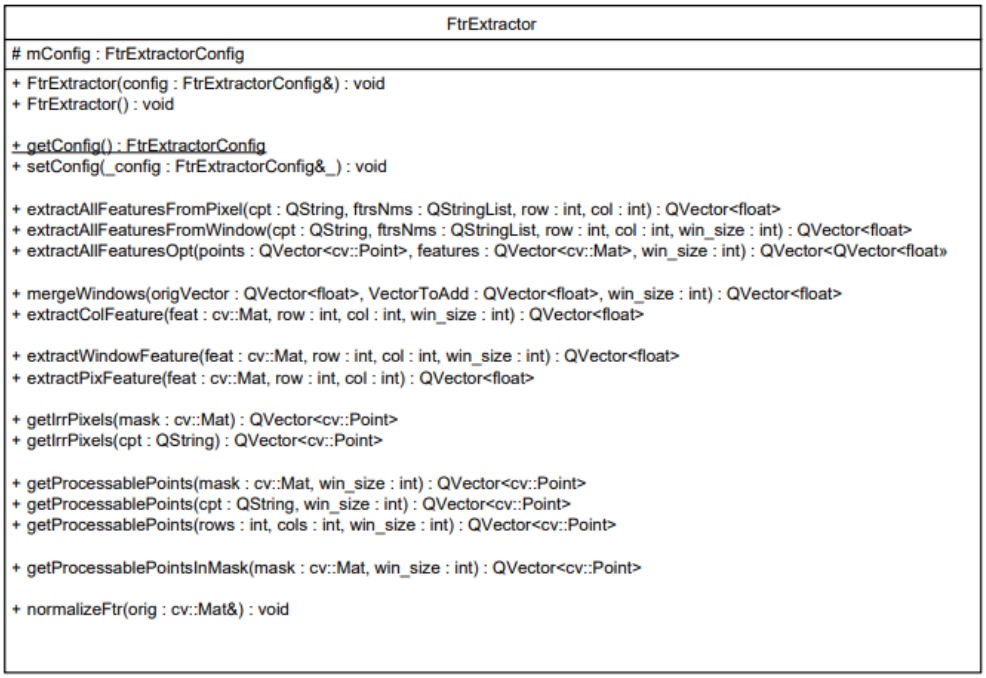


Figura 46: Clase FtrExtractor.

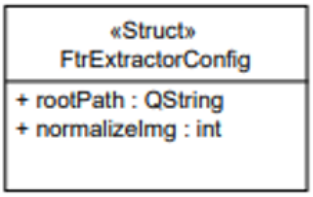


Figura 47: FtrExtractorConfig.

autoencoder (con capas densas). Para facilitar la selección del tipo de modelo desde la configuración, se enumeran los dos tipos de modelos disponibles.

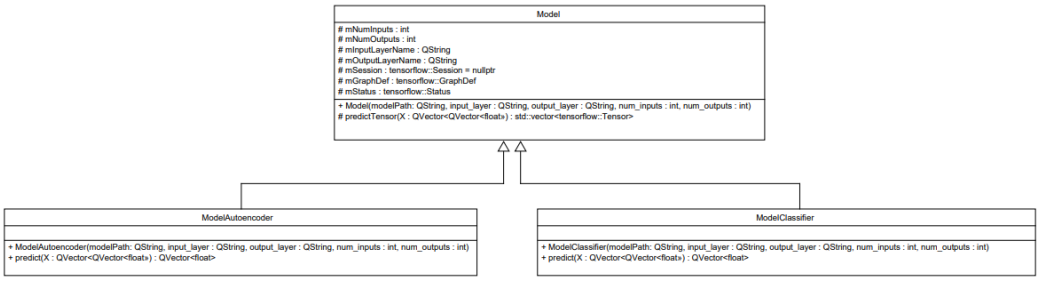


Figura 48: Clase Model

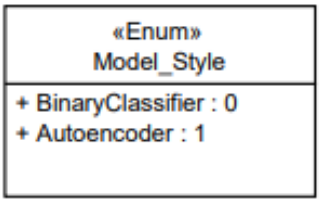


Figura 49: Model Style.

A.0.5. Clase MLBot

La clase MLBot es la que encapsula todo el proceso de ML. Se divide en una clase base "MLBotBase"que contiene las instancias de las clases mencionadas previamente y permite cargar la configuración de cada una de ellas. La clase MLBotPixel es una clase derivada que contiene las particularidades del análisis píxel a píxel. La configuración se guarda en una estructura MLBotConfig que contiene los siguientes parámetros:

- modelPath : Dirección del archivo .pb con los pesos del modelo.
- inputLayer : Nombre de lacapa de entrada del modelo.
- outputLayer : Nombre de la capa de salida del modelo.
- featureStyle : Tipo de extracción de características: por ventana, por píxel o por blob.
- modelStyle : Tipo de modelo : clasificador o autoencoder.
- featWindowSize : Tamaño de ventana. Solo necesario si featureStyle = FEATbyWINDOW.

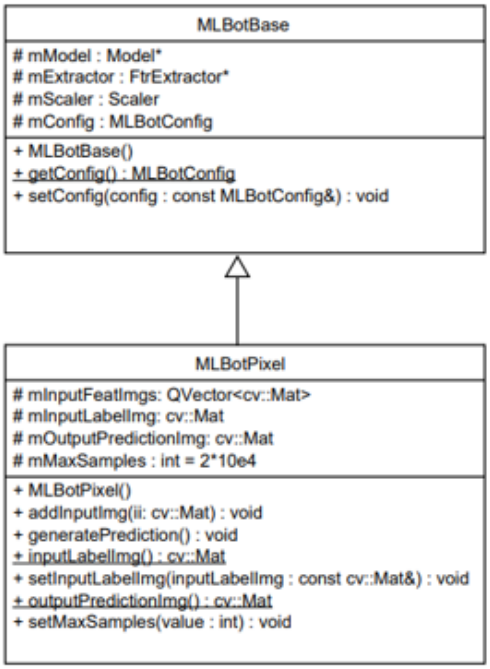


Figura 50: Clase MLBot.

- batchSize : Tamaño del lote.
- numInputs : Tamaño del vector de entrada.
- weigthsCSVPath : Dirección del csv con la media y la desviación estándar de las características.
- extractorConfig : Parámetros de configuración del extractor de características (ver Figura 47).

El método generatePrediction() es el encargado de lanzar el procesamiento de ML, cuyo procedimiento es el siguiente:

1. En función del parámetro modeStyle se genera una instancia de modelClassifier o modelAutoencoder.
2. Si se dispone de archivo csv de normalización se carga el directorio en el scaler. Si no se dispone de archivo de normalización, se normaliza cada imagen en mInputFeatImgs con un scaler del tipo MIN-MAX en

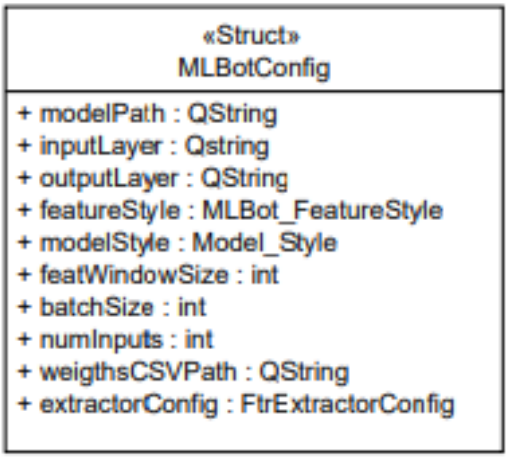


Figura 51: Estructura de configuración de MLBot.

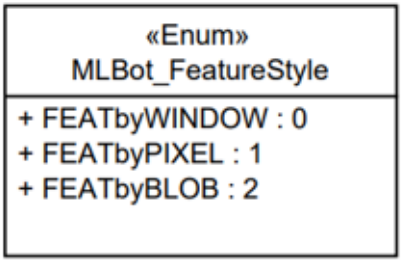


Figura 52: MLBot estilo de características.

el rango [-1,1].

3. Se extraen todos los puntos conflictivos. En este caso se procesan todos los puntos de la pinza.
4. Si el número de puntos a procesar es mayor que mMaxSamples, se realiza el procesamiento en lotes de mMaxSamples.
5. Se extraen las características y se normalizan en caso de que sea necesario.
6. Se ejecuta el modelo.
7. Se guardan las predicciones del modelo en la máscara mOutputPredictionImg.

B. Análisis cualitativo de métodos supervisados

El objetivo del análisis cualitativo de los resultados es confirmar que el modelo es capaz de destacar los defectos frente a las zonas normales de la pieza. A continuación se muestran imágenes de la salida en crudo de los modelos, es decir, sin aplicar postprocesamiento. De esta forma no solo se consigue evaluar la capacidad del modelo de detectar defectos sino también conocer las zonas donde falla y actuar en consecuencia. Como ya se ha comentado previamente, la mayoría de FPs aparecen en los bordes de las piezas por lo que no representan un gran problema, ya que se pueden eliminar restando la clasificación de referencia (ver Figura 23).

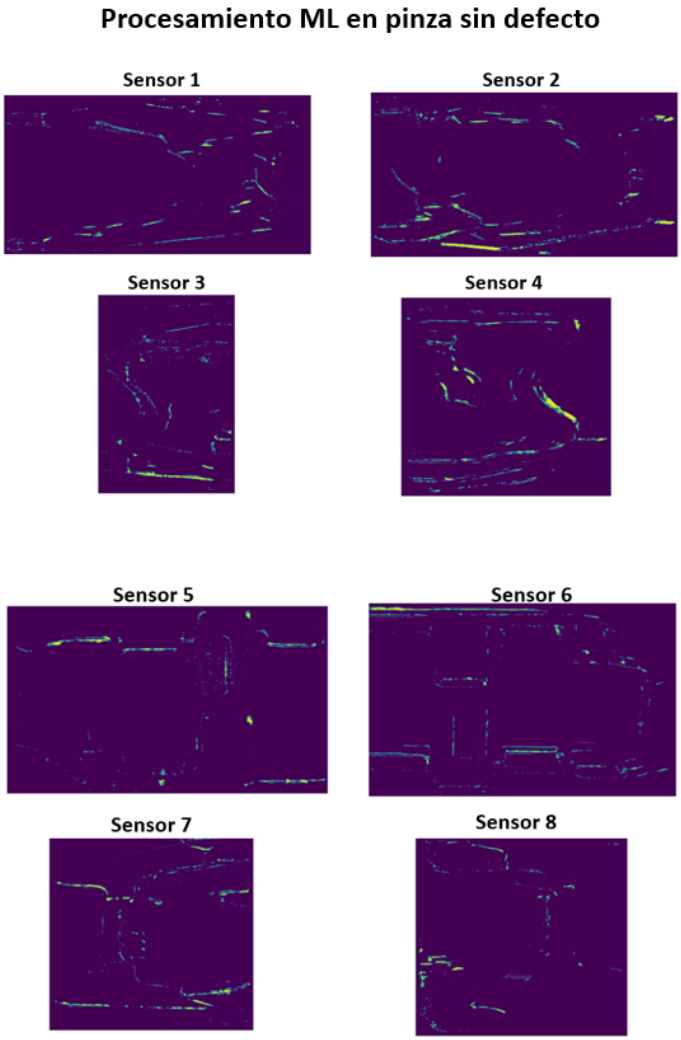


Figura 53: Ejemplo de procesamiento en crudo de los modelos de ML para una pinza OK.

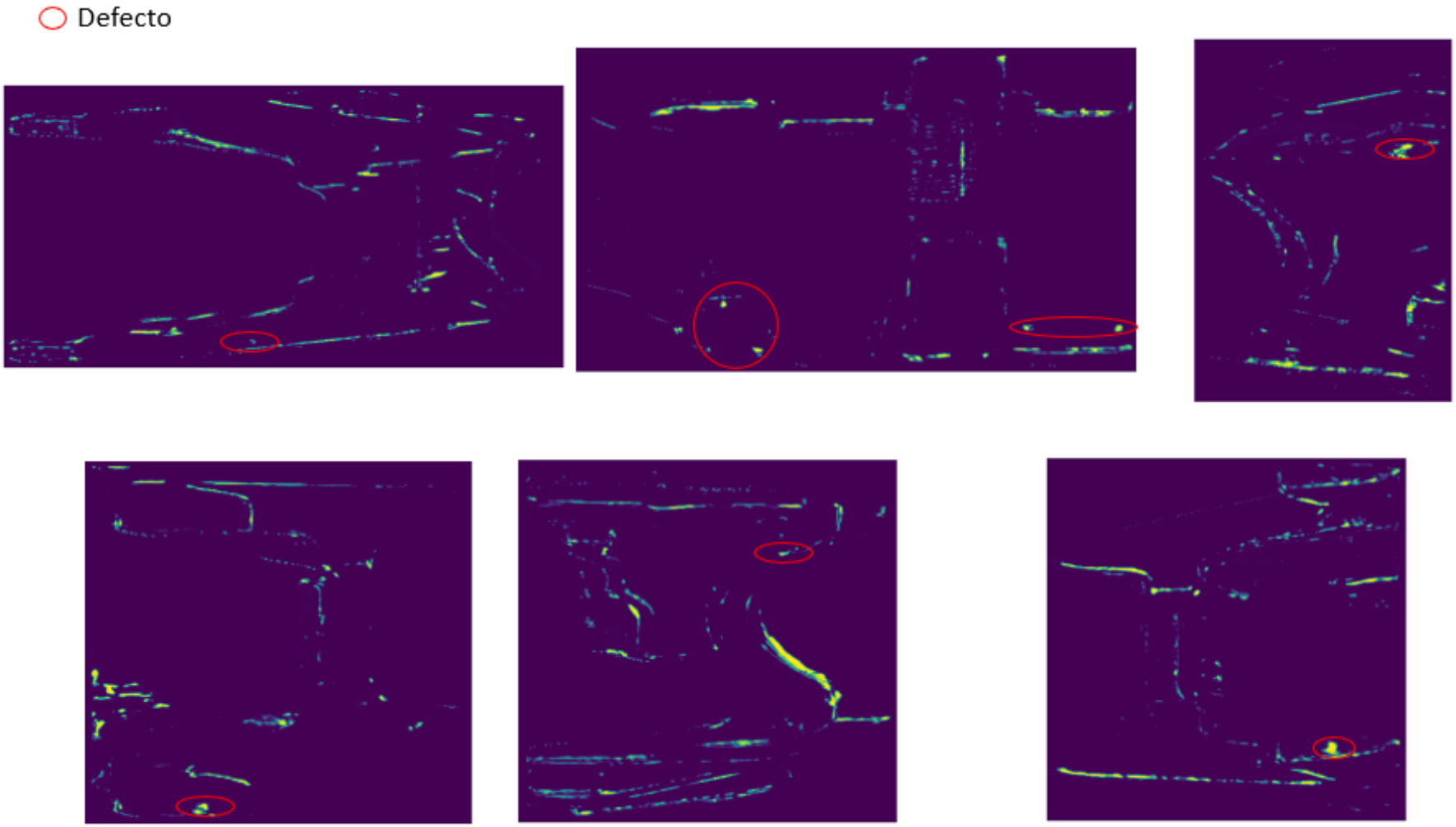


Figura 54: Ejemplo de procesamiento en crudo de los modelos de ML para pinzas con defectos.

C. Análisis cualitativo de métodos no supervisados

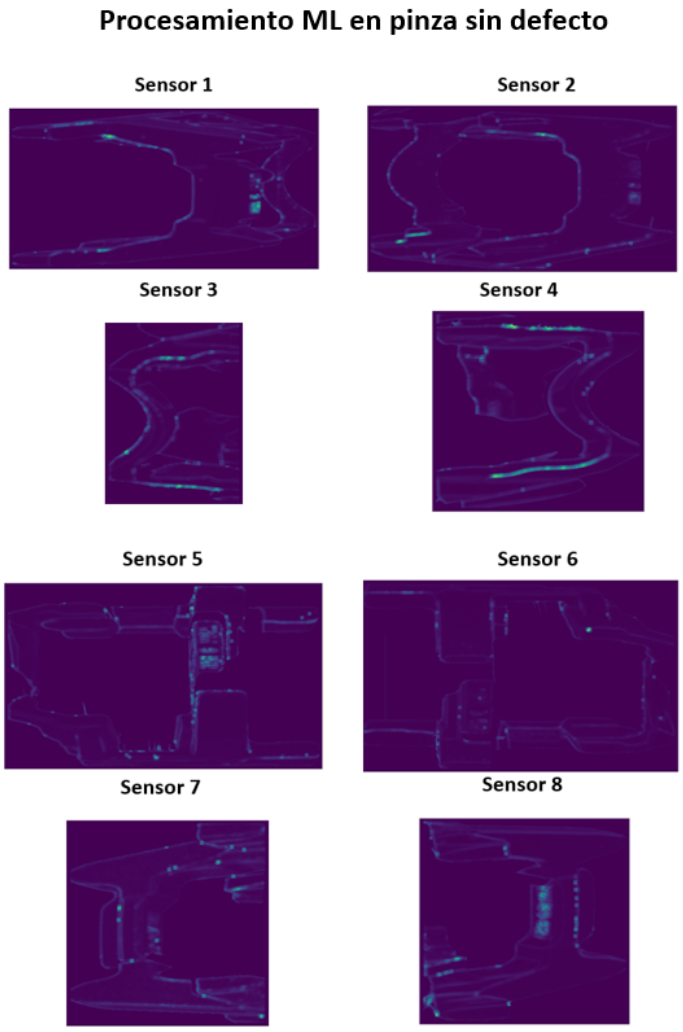


Figura 55: Ejemplo de procesamiento en crudo de los modelos de ML para una pinza OK. Se visualiza el residuo.

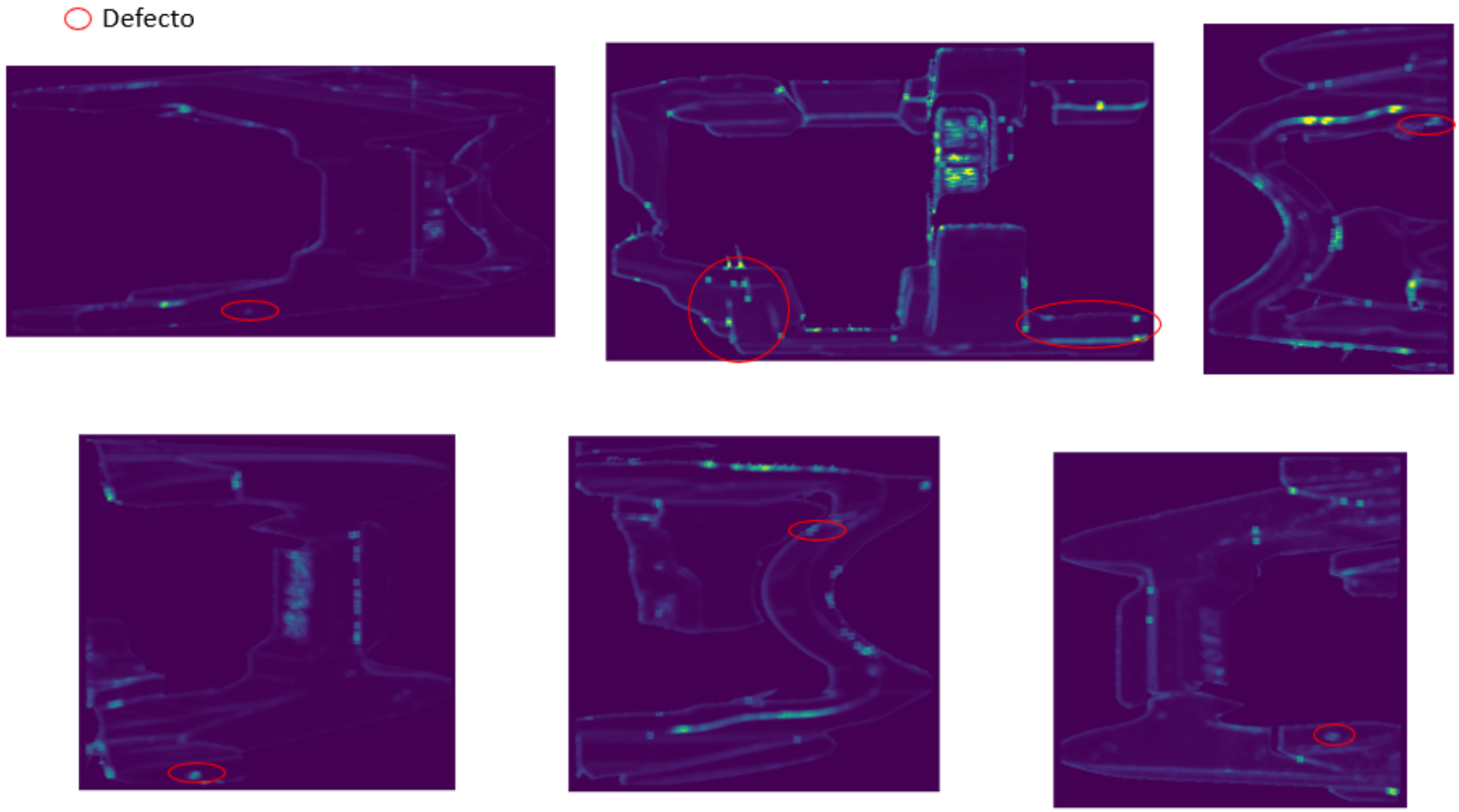


Figura 56: Ejemplo de procesamiento en crudo de los modelos de ML para pinzas con defectos. Se visualiza el residuo.

5. Referencias

[1] DSIplus. (2006) Desarrollo soluciones integrales plus. [Online]. Available: http://www.dsiplus.es/

[2] J. Marina Juárez, "Reconstrucción de perfiles submicrométricos con holografía conoscópica para la medición de rugosidad," Ph.D. dissertation, Universidad de Oviedo.

[3] J. Padmanabhan and M. Premkumar, "Machine learning in automatic speech recognition: A survey," *IETE Technical Review*, vol. 32, pp. 1–12, 02 2015.

[4] S. Manna, S. Ghildiyal, and K. Bhimani, "Face recognition from video using deep learning," in *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, 2020, pp. 1101–1106.

[5] R. Sharma, B. Kaushik, and N. Gondhi, "Character recognition using machine learning and deep learning - a survey," in *2020 International Conference on Emerging Smart Computing and Informatics (ESCI)*, 2020, pp. 341–345.

[6] T. Okuyama, T. Gonsalves, and J. Upadhay, "Autonomous driving system based on deep q learnig," in *2018 International Conference on Intelligent Autonomous Systems (ICoIAS)*, 2018, pp. 201–205.

[7] X.-Y. Wang and J. Bu, "A fast and robust image segmentation using fcm with spatial information," *Digital Signal Processing*, vol. 20, no. 4, pp. 1173 – 1182, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1051200409002437

[8] I. V. Systems. Industrial vision systems (ivisys). [Online]. Available: https://ivisys.com/

[9] J.-S. Chou and A. S. Telaga, "Real-time detection of anomalous power consumption," *Renewable and Sustainable Energy Reviews*, vol. 33, pp. 400 – 411, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364032114001142

[10] Z. Li, J. Li, Y. Wang, and K. Wang, "A deep learning approach for anomaly detection based on sae and lstm in mechanical equipment,"

The International Journal of Advanced Manufacturing Technology, vol. 103, 07 2019.

[11] M. Azimi and G. Pekcan, "Structural health monitoring using extremely-compressed data through deep learning," *Computer-Aided Civil and Infrastructure Engineering*, 11 2019.

[12] D. M. Hawkins, "The problem of overfitting," *Journal of chemical information and computer sciences*, vol. 44, no. 1, pp. 1–12, 2004.

[13] Python. [Online]. Available: https://www.python.org/

[14] Numpy. [Online]. Available: https://numpy.org/

[15] Pandas. [Online]. Available: https://pandas.pydata.org/

[16] Scikit learn. [Online]. Available: https://scikit-learn.org/

[17] Tensorflow. [Online]. Available: https://www.tensorflow.org/

[18] Keras. [Online]. Available: https://keras.io/

[19] Qt. [Online]. Available: https://www.qt.io/

[20] L. Chen, *Curse of Dimensionality*. Boston, MA: Springer US, 2009, pp. 545–546. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9_133

[21] R. Blomley, M. Weinmann, J. Leitloff, and B. Jutzi, "Shape distribution features for point cloud analysis-a geometric histogram approach on multiple scales," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 2, no. 3, p. 9, 2014.

[22] T. Dietterich, "Overfitting and undercomputing in machine learning," *ACM Comput. Surv.*, vol. 27, no. 3, p. 326–327, Sep. 1995. [Online]. Available: https://doi.org/10.1145/212094.212114

[23] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[24] O. Simeone, "A very brief introduction to machine learning with applications to communication systems," *IEEE Transactions on Cognitive Communications and Networking*, vol. 4, no. 4, pp. 648–664, 2018.

[25] J. E. Van Engelen and H. H. Hoos, "A survey on semi-supervised learning," *Machine Learning*, vol. 109, no. 2, pp. 373–440, 2020.

[26] Z. Ghahramani, "Unsupervised learning," in *Summer School on Machine Learning*. Springer, 2003, pp. 72–112.

[27] P. Y. Glorennec, "Reinforcement learning: An overview," in *Proceedings European Symposium on Intelligent Techniques (ESIT-00), Aachen, Germany*. Citeseer, 2000, pp. 14–15.

[28] W.-Y. Loh, "Classification and regression trees," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 14–23, 2011.

[29] Scikit-learn. (2020) Decision trees. [Online]. Available: https://scikit-learn.org/stable/modules/tree.html

[30] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, no. 3, pp. 199–222, 2004.

[31] Scikit-learn. (2019) Svm scikit-learn. [Online]. Available: https://scikit-learn.org/stable/modules/svm.html#id15

[32] R. HECHT-NIELSEN, "Iii.3 - theory of the backpropagation neural network**based on "nonindent" by robert hecht-nielsen, which appeared in proceedings of the international joint conference on neural networks 1, 593–611, june 1989. © 1989 ieee." in *Neural Networks for Perception*, H. Wechsler, Ed. Academic Press, 1992, pp. 65 – 93. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780127412528500108

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[34] Early stopping callback in keras api. [Online]. Available: https://keras.io/api/callbacks/early_stopping/

[35] Hyperas. [Online]. Available: https://github.com/maxpumperla/hyperas

[36] Dropout layer in keras api. [Online]. Available: https://keras.io/api/layers/regularization_layers/dropout/

[37] T. S. Madhulatha, "An overview on clustering methods," *arXiv preprint arXiv:1205.1117*, 2012.

[38] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE journal*, vol. 37, no. 2, pp. 233–243, 1991.

[39] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[40] Divergencia kullback-leibler. [Online]. Available: https://en.wikipedia.org/wiki/Kullback-Leibler_divergence

[41] Opencv: Eroding and dilating. [Online]. Available: https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html

[42] D. Singh and B. Singh, "Investigating the impact of data normalization on classification performance," *Applied Soft Computing*, vol. 97, p. 105524, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1568494619302947

[43] Hdf. [Online]. Available: https://www.hdfgroup.org/

[44] S. Picek, A. Heuser, A. Jovic, S. Bhasin, and F. Regazzoni, "The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 1, pp. 1–29, 2019.

[45] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[46] Z. H. Hoo, J. Candlish, and D. Teare, "What is an roc curve?" 2017.

[47] K. Boyd, K. H. Eng, and C. D. Page, "Area under the precision-recall curve: point estimates and confidence intervals," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2013, pp. 451–466.

[48] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: Association

for Computing Machinery, 2006, p. 233–240. [Online]. Available: https://doi.org/10.1145/1143844.1143874

[49] Tensorflow c++ api. [Online]. Available: https://www.tensorflow.org/api_docs/cc

[50] Accuracy class. [Online]. Available: https://keras.io/api/metrics/accuracy_metrics/#accuracy-class

[51] M. Chu, A. Wang, R. Gong, and M. Sha, "Strip steel surface defect recognition based on novel feature extraction and enhanced least squares twin support vector machine," *ISIJ international*, vol. 54, no. 7, pp. 1638–1645, 2014.

[52] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–6.

[53] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[54] Standard scaler scikit-learn. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html