Universidad Internacional de La Rioja (UNIR)

ESIT

Máster Universitario en Inteligencia Artificial

Herramienta de diagnóstico previa a la realización de pintura mural con drones

Trabajo Fin de Máster

Presentado por: Balza de Vallejo Julián, Asier

Director/a: Galindo Diez, Emma

Ciudad: Vitoria-Gasteiz
Fecha: 2021

Resumen

Una de las labores más importantes previas a la realización de cualquier intervención pictórica sobre una pared, es comprobar el estado de su superficie. Cuando es además esta área realmente extensa e inaccesible directamente para el ojo humano, cobra sentido pensar en la automatización de dicha tarea.

El principal fenómeno que describe la situación de la superficie de una fachada o edificio es el desconchado o delaminado, que debe ser identificado y cuantificado para valorar y planificar su reparación previa a una posible labor de mantenimiento o intervención pictórica también automatizadas. En este aspecto, se fija la atención sobre el dron como vehículo, elemento extractor de la información necesaria para la tarea descrita: obtención de fotografía, imagen. Y es a partir de esa información obtenida, donde entra en juego el uso de la Inteligencia Artificial, concretamente el área de visión por computación con el objetivo de diseñar un sistema capaz de identificar el fenómeno descrito, las áreas de desconchado o descascarillado.

En la actualidad, el fenómeno de identificación y clasificación de las grietas como indicadores del estado de salud de las distintas obras de ingeniería civil, ha sido abordado mediante distintos enfoques en el ámbito de la visión por computación. Concretamente han dado muy buenos resultados modelos del ámbito del aprendizaje profundo, en especial el uso de redes neuronales convolucionales, y es dicha línea la que ha inspirado este trabajo.

Así, tras confeccionar inicialmente un *data set* propio constituido por 6452 imágenes, se han implementado y con ello entrenado tres modelos de redes neuronales convolucionales a partir de las arquitecturas XCeption, VGG16 y ResNet50, consiguiendo valores de ratio de éxito entre 82% y 90%.

A partir del modelo que ha ofrecido mayor valor de exactitud, se ha confeccionado una herramienta que, tras proporcionarle imágenes de edificios tomadas por un dron, devuelve un mapa visual sobre la imagen original donde se identifican las zonas que el modelo ha clasificado como positivas, contenedoras de desconchados o descascarillados. La aplicación confeccionada, constituye así una herramienta de diagnóstico visual y es el punto de partida para la evaluación de la magnitud del fenómeno buscado, así como de la elaboración de un mapa preciso de intervención automatizada.

Palabras Clave: diagnóstico de superficies, desconchado, visión por computación, redes neuronales convolucionales, drones.

Abstract

Delamination is the principal phenomenon that may reveal the surface´s health on a wall, and drone vehicles are a good choice for being able to read this info from any building, getting the necessary images.

Nowadays identification and classification of cracks on civil engineering structures as diagnosis of architectural condition has been researched and treated from different points of view around computer vision, on deep learning specifically, using convolutional neural networks. This working line has inspired this research.

First, an appropriate data set concerning delamination has been built, containing 6452 images. Deep learning models have been trained on it, getting from 82% to 90% accuracy score.

Taking the model that outperforms the others on classification task, the following tool has been built: entering pictures of a wall taken by a drone, it returns a visual map over the original picture, showing on red the frames identified as containing delamination.

Keywords: surface diagnosis, delamination, computer vision, convolutional neural networks, drone.

Índice de contenidos

1. Introducción....1
1.1 Motivación....1
1.2 Planteamiento del trabajo....3
1.3 Estructura de la memoria....3
2. Contexto y estado del arte....4
2.1 Detección de defectos en superficies....4
2.2 Técnicas de aprendizaje profundo....6
2.2.1 Redes Neuronales Artificiales....7
2.2.2 Redes Neuronales Convolucionales....9
2.3 Uso actual de CNNs para detección de defectos en superficies....11
3. Objetivos y metodología de trabajo....15
3.1. Objetivo general....15
3.2. Objetivos específicos....15
3.3. Metodología del trabajo....16
4. Identificación de requisitos....18
4.1. Construcción del *data set*....18
4.2. Clasificador DCNN....20
4.2.1 Entorno de Trabajo y *frameworks*....21
4.2.2 Etapas del desarrollo para la implementación de los modelos....22
4.3. Flujo de trabajo general....25
4.4. Toma de imágenes mediante dron....25
5. Descripción de la herramienta software desarrollada....27
5.1. Implementación de modelos para el clasificador DCNN....27
5.1.1 Desarrollo de código para ejecución....27
5.1.2 Arquitecturas DCNN....32
5.1.3 Entrenamientos....34

5.1.4 Elección del mejor modelo 35
5.2 Desarrollo de la herramienta....37
6. Evaluación....39
6.1 Evaluación de los modelos clasificadores....39
6.2 Aplicación de la herramienta desarrollada 41
6.2.1 Mapas de Salida....41
7. Conclusiones y trabajo futuro 44
7.1. Conclusiones 44
7.2. Líneas de trabajo futuro 45
8. Bibliografía 47
Anexos....51
Anexo: Artículo de investigación 51

Índice de tablas

Tabla 1 Valores de función de pérdida y exactitud obtenidos para los distintos modelos y parámetros empleados, para los conjuntos de validación y test.

.....41

Tabla 1 Valores medios de la función de pérdida y 'accuracy' obtenidos a partir de los valores de los conjuntos de test, para los modelos entrenados.

.....42

Índice de figuras

Fig. 1. *Fotografía de un dron de inspección. (Fuente: https://www.applus.com/global/en/what-we-do/solutions/uav-drone-inspection-services)* 1

Fig. 2. *Izq.: Dron realizando un mural con aerosol (Fuente: https://newatlas.com/ufo-drones-graffiti-painting/60423/) Dcha.: Mural de grandes dimensiones realizado en un edificio de Rusia. (Fuente: https://www.mk.ru/mosobl/2019/09/01/spalnyy-rayon-podmoskovya-prevratilsya-v-muzey-stritarta.html)*..... 2

Fig. 3 *Izq.: Imagen de una grieta perteneciente al data set "Concrete Crack Images for Classification". (Fuente: https://data.mendeley.com/datasets/5y9wdsg2zt/1) Dcha.: Imagen de un desconchado en una pared (Fuente: Internet)*..... 5

Fig. 4 *Esquema de una neurona artificial basada en la idea original de McCulloch y Pitts. Fuente: https://www.researchgate.net/publication/323465059/figure/fig2/AS:599207769554946@1519873673906/McCulloch-Pitts-computational-model-of-a-neuron.png*..... 7

Fig. 5 *Izda.: Ilustración de un filtro de dimensiones 3x3 actuando sobre volumen de de entrada. Fuente: https://upload.wikimedia.org/wikipedia/commons/9/95/Convolutional_Neural_Network_with_Color_Image_Filter.gif. Licencia Creative Commons. Dcha: Ilustración del deslizamiento del filtro a través del volumen de entrada. Fuente: https://miro.medium.com/max/652/0*_BbDhaxqSZP3Yomf.* 10

Fig. 6 *Muestras de imágenes de la clase positiva del data set confeccionado por el autpr,"Delaminations2021"; la fila de arriba está compuesta por imágenes completas (mapas) y las de la fila inferior son el resultado del fragmentado o corte*..... 19

Fig. 7 *Imágenes, de arriba abajo: Dron antes de tomar vuelo; Detalle de una de las fachadas escaneada; Imagen global de una de las fachadas escaneadas*..... 26

Fig. 8 *Extracto del código empleado en la parte de pre-procesamiento y data augmentation*..... 29

Fig. 9 *Muestra de 25 imágenes del conjunto de entrenamiento, etiquetadas automáticamente como positivas (1) y negativas (0)*..... 30

Fig. 10 *Preparación de los conjuntos de test "Test 2" e "Internet", para la evaluación posterior de los modelos*... 31

Fig. 11 *Ejemplo de aplicación del método compile(), todavía sin aplicar a un modelo concreto*..... 32

Fig. 12 *Esquema de una red VGG16. (Fuente: https://towardsdatascience.com/a-demonstration-of-transfer-learning-of-vgg-convolutional-neural-network-pre-trained-model-with-c9f5b8b1ab0a)*..... 33

Fig. 13 *Esquema de la red ResNet50(Fuente: https://machinelearningknowledge.ai/keras-implementation-of-resnet-50-architecture-from-scratch/)*..... 34

Fig. 14 *Extracto de código donde se declara el número de epochs, los objetos 'callback' y la implementación del método fit() para inicio del entrenamiento*..... 35

Fig. 15 *Arriba: Extracto de la salida generado durante el entrenamiento del modelo basado en la red ResNet50, con caso de overfitting. Abajo: Gráfica que muestra los valores de exactitud obtenidos durante el entrenamiento correspondiente al extracto anterior*..... 36

Fig. 16 *Ejemplo del uso del método evaluate(), para un mismo modelo, empleando en cada caso un conjunto de test*..... 36

Fig. 17 *Extracto de Código donde se observa la extracción del modelo clasificador encapsulado en el fichero json, la carga de los pesos generados durante el entrenamiento seleccionado y compilación del modelo dejándolo preparado para predecir*..... 37

Fig. 18 *Imagen que muestra el código contenido en la función diag_map_gen_2*……..38

Fig. 19 *Mapas de entrada (dcha.) y de salida (izq.) tras emplear la herramienta desarrollada*…..……41

Fig. 20 *Muestra de salida de la información relativa a las clasificaciones realizadas por la función diag_map_gen_2, correspondiente a la primera imagen presentada, realizada por el dron*……..*42*

Fig. 21 *Ejemplo de uso de la herramienta para un mapa confeccionado a partir de cuatro imágenes tomadas por dron. Mapa de entrada (dcha.) y mapa de salida (izda.)*……..42

Fig. 22 *Imágenes correspondientes a la salida (arriba) y entrada (abajo) de la herramienta, para fachada de un edificio compueta por 12 imágenes tomadas con dron*…..……43

Fig. 23 *Elementos que han dado lugar a falsos positivos, de izda. a dcha.: Trapo colgado, cables y persiana*….44

Fig. 24 *Muestra, en las propiedades internas de una de las imágenes proporcionadas por el dron, de las coordenadas gps y altura en el instante de realizar la toma*…..……46

1. Introducción

1.1 Motivación

Desde que el ser humano comenzó a desarrollar las primeras herramientas, y después con éstas, las primeras máquinas, comenzó también, de alguna manera la búsqueda de la automatización de las posibles tareas que éste podía realizar. La evolución del conocimiento, dando lugar a máquinas mecánicas cada vez más sofisticadas, alimentadas después eléctricamente y con capacidad, más adelante, para computar electrónicamente, han llevado ese paradigma de la automatización a niveles que con anterioridad apenas podrían haber sido imaginados. Y es entonces cuando esa evolución de la computación y el cálculo automatizado, tras ser etiquetado como inteligente, continúa filtrándose por cada poro del tejido de todos los campos de conocimiento de la actualidad.

Uno de esos campos, donde confluyen tanto la tecnología como medio físico, junto con el uso de la inteligencia computacional, son los vehículos aéreos no tripulados, drones o UAVs (del inglés, *Unmanned Aerial Vehicles*). Son el vehículo ideal para transportar de manera ágil, y a cualquier zona de difícil acceso, un conjunto de sensores que recogen y transmiten datos, que más tarde un sistema inteligente puede interpretar y calcular respuestas y acciones que suponen un triunfo de la automatización (Carrio et al., 2017) nuevamente, en la tarea aplicada.

Fig. 1 *Fotografía de un dron de inspección. (Fuente: https://www.applus.com/global/en/what-we-do/solutions/uav-drone-inspection-services)*

Una de las labores que el ser humano lleva realizando desde épocas muy tempranas y que lo ha acompañado hasta la actualidad es la realización de pinturas murales, desde las primeras intervenciones en cuevas hasta los más extensos murales que decoran hoy día grandes

edificaciones. Y es también una tarea en la que la automatización hace algún tiempo que hizo su incursión.

En 2019, en Turín, bajo el nombre de UFO-Urban Flying Opera, se llevó a cabo un proyecto de automatización de pintura mural, con una extensión de 150 m^2, empleando aerosoles accionados por cuatro drones, controlados por un sistema desarrollado por Tsuru Robotics. Esta demostración, con objetivo fundamentalmente artístico pone en evidencia cómo este sector está siendo también alcanzado por la tecnología mencionada (Vempati et al., 2018).

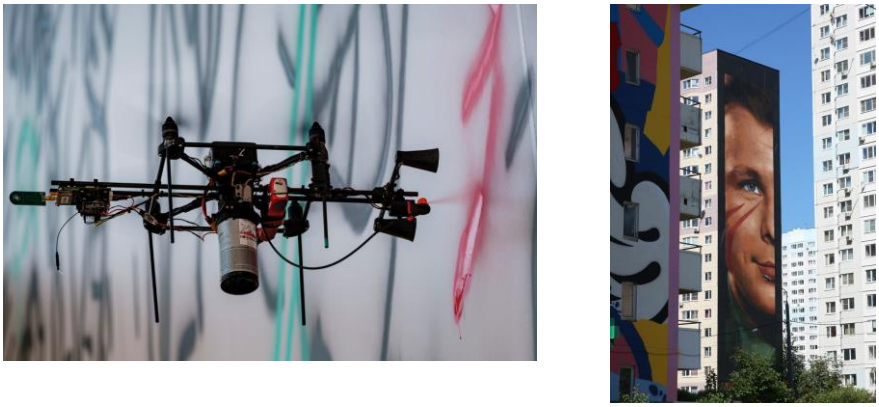

Fig. 2 *Izq.: Dron realizando un mural con aerosol (Fuente: https://newatlas.com/ufo-drones-graffiti-painting/60423/) Dcha.: Mural de grandes dimensiones realizado en un edificio de Rusia. (Fuente: https://www.mk.ru/mosobl/2019/09/01/spalnyy-rayon-podmoskovya-prevratilsya-v-muzey-stritarta.html)*

La realización de pinturas murales a gran escala, tanto con carácter artístico, como de rehabilitación o preventivo son ya labores alcanzadas por el proceso de automatización pudiendo evitar la construcción de andamiajes, ahorrando tiempos de ejecución (Pereira & Pereira, 2015)y costes, y evitando accidentes, en definitiva optimizando la tarea.

Es en este contexto de la realización autónoma de labores de pintura mural, mantenimiento o prevención, donde se torna importante e indispensable una **labor previa: la inspección de la superficie que va a ser intervenida**. La exposición a los fenómenos propios de la intemperie de los materiales que conforman la fachada de un edificio, pueden deteriorarla evidenciándose esto como levantamientos, grietas, descascarillados o desconchados. **Identificar y localizar** estos puntos es fundamental para un correcto desarrollo de la pintura mural, evitando a corto plazo un daño inminente y muy costoso *a posteriori* sobre el trabajo realizado.

1.2 Planteamiento del trabajo

La inspección para la identificación y localización de las posibles zonas deterioradas y obtener un diagnóstico del estado de una superficie, es una labor que se ha hecho hasta ahora visualmente, mediante intervención humana. Esto supone tener que chequear cada metro cuadrado, teniendo que acceder al mismo, con el coste que ello supone en infraestructura (andamios, elevadoras), riesgo y tiempo.

Poder automatizar, acelerar esa tarea con un coste reducido despliega un horizonte de posibles soluciones. Y es ahí donde la extracción de imágenes facilitadas por drones, clasificadas por un sistema cognitivo artificial e interpretadas computacionalmente, constituye una herramienta potente e interesante. Recoger el conjunto de imágenes, identificar los puntos de deterioro y confeccionar un mapa de diagnóstico, tanto visual para interpretación humana, así como un mapa de actuación para la posible intervención/reparación automática, perpetrada también por drones específicos, es una labor muy útil.

1.3 Estructura de la memoria

La distribución que sigue el resto de los contenidos es la siguiente:

En el capítulo 2 se plantea el entorno del problema descrito y se presenta la situación actual de soluciones similares, desarrollos y técnicas existentes que de una manera u otra lo abordan.

A continuación, en el capítulo 3 se presentan los objetivos generales y específicos de este trabajo, para continuar con una descripción de la metodología empleada para la consecución de éstos. En el capítulo 4 se identifican las necesidades y especificaciones que dan lugar a la herramienta de software desarrollada para introducir después en el capítulo 5 las fases de desarrollo y detalles de ésta.

Prosigue en el capítulo 6 la evaluación de la aplicación de la herramienta, donde se presentan los resultados obtenidos tanto en el entrenamiento de clasificadores como en el uso de la herramienta. Como cierre se presentan en el capítulo 7 las conclusiones a las que se ha llegado, y varias posibles líneas de trabajo futuro. Finalmente, en el capítulo 8 se incluye la bibliografía en la que se ha apoyado todo este trabajo.

2. Contexto y estado del arte

Es interesante, en este punto, enfatizar la idea del dron como vehículo, en sentido literal y figurado, tanto para la actuación sobre la superficie, así como para la extracción de datos a partir de ésta. Es precisamente, esa actuación, intervención pictórica descrita al inicio, la que necesita previamente esa extracción de información, de la obtención de datos procedentes del espacio a ser intervenido.

Una vez obtenida la información requerida, el primer punto de atención reside en la identificación del fenómeno buscado, como lo es en este caso el descascarillado o desperfecto de zonas, lo que se analiza en la subsección “Detección de defectos en superficies”.

2.1 Detección de defectos en superficies

Se centra en este punto la mirada sobre las diferentes técnicas, en el ámbito de la inteligencia artificial, que han sido y están siendo aplicadas a la detección de fenómenos, en este caso entendidos como defectos en superficies. El concepto es muy amplio y abarca campos tan diversos como la detección de faltas en vigas de acero, superficies y estructuras metálicas, inspección de paneles solares y de placas electrónicas (PCB), o defectos de fábrica en textiles por nombrar sólo algunos de ellos.

Sin embargo, el foco que abarca la atención de este trabajo es de las superficies estructurales arquitectónicas, constituidas en definitiva por materiales de construcción como el hormigón, cemento, ladrillos, yeso, y diversos revestimientos como fijadores, pinturas y barnices, en última instancia.

Uno de los fenómenos más estudiados en estas estructuras, y perceptible en las superficies es el de las grietas, que son potenciales indicadores de su estado de salud (Phung et al., 2017), (Cha et al., 2018), (Kim et al., 2020). Elementos de la ingeniería civil como puentes (Adhikari et al., 2014), carreteras (Maeda et al., 2018) y edificios, donde la identificación de este fenómeno lleva décadas siendo una tarea presencial, es desde hace tiempo foco de atención para su automatización, desde múltiples enfoques, dando lugar al uso de diversas técnicas y tecnologías involucradas (Gopalakrishnan et al., 2018).

Otro de los fenómenos susceptibles de ocurrir sobre las paredes, y de gran interés pero mucho menos tratado, es el de los desconchados o delaminaciones (Shin et al., 2020). Se trata de levantamientos del revestimiento o enlucido en determinadas zonas de la superficie, también indicadores del estado de una construcción, pero en un sentido menos estructural y mayoritariamente relacionado con las reacciones producidas entre y sobre los materiales de

Fig. 3 *Izq.: Imagen de una grieta perteneciente al data set "Concrete Crack Images for Classification". (Fuente: https://data.mendeley.com/datasets/5y9wdsg2zt/1) Dcha.: Imagen de un desconchado en una pared (Fuente: Internet)*

cobertura. El motivo principal de que se dé este fenómeno es la pérdida de elasticidad de las capas más exteriores. Entre los factores detonantes, se encuentra el uso de productos de baja calidad, el asentamiento de la propia estructura, la aplicación del revestimiento en condiciones de humedad, la aparición de humedad, y por supuesto la exposición a las condiciones climatológicas de la intemperie (bajas temperaturas, sol, lluvia), que consiguen mermar las propiedades de los materiales inicialmente aplicados.

En este sentido, y acorde al propósito de este trabajo el conjunto de técnicas de detección contempladas se encuentran englobadas bajo la etiqueta de no-destructivas, son considerados métodos de evaluación no invasivos, ya que no interfieren ni deterioran el objeto a explorar (Mohan & Poobal, 2018), (Maniat et al., 2021).

A la hora de abordar la tarea inicial de la extracción de información, son diversos los fundamentos relativos a los sensores empleados, abarcando desde infrarrojos, imagen térmica, ultrasonidos, láser y radiografía (Mohan & Poobal, 2018), vibraciones (Azimi et al., 2020). Y por supuesto la Luz Visible, en definitiva, la extracción de imágenes mediante cámara digital, tanto fotografía (Carrio et al., 2017), como *frames* de vídeo (Bhowmick et al., 2020). Es en la toma de imágenes fotográficas, donde reside el interés para el desarrollo de este trabajo.

La identificación de fenómenos como los descritos, ha sido una tarea comúnmente tratada desde el denominado campo de la visión por computación. Un posible enfoque son las técnicas de procesamiento de imagen y algoritmos propios de la visión por computación. Suponen una fuente muy diversa de abordaje del problema planteado pudiendo identificar el

uso de diferentes técnicas como *wavelet transform*, *minimal path selection*, *edge detection* e *intensity thresholding* (Phung et al., 2017) . Pero requieren en general, una severa etapa de pre-procesado ya que son muy susceptibles a las condiciones en las que se realiza la toma de imágenes (Kim et al., 2020). Otra forma de lograr la detección o identificación de fenómenos en superficies es el empleo de **técnicas de aprendizaje profundo**, *Deep Learning* (DL), que se introducen en la siguiente sub-sección.

2.2 Técnicas de aprendizaje profundo

Hoy en día el aprendizaje automático, es una de las ramas más fervientes del campo de la Inteligencia Artificial. La ejecución de algoritmos en computadoras, que pueden realizar generalizaciones a partir de un conjunto de datos proporcionado está en constante expansión y estudio en cualquier área aplicable. Si además de la extracción de patrones a partir de la información disponible, se consigue que los sistemas aprendan propiamente representaciones adecuadas, a partir de los datos en bruto para la solución de ciertos problemas, esto constituye propiamente una subárea denominada *representation learning*, o aprendizaje de representaciones. Y es precisamente dentro de ese marco donde se sitúa el concepto del aprendizaje profundo, que parte del desarrollo de sistemas cognitivos dotados de gran capacidad de abstracción, artífices de la formulación de representaciones complejas a partir de representaciones más simples, y que está ofreciendo muy buenos resultados en muchas de las áreas de investigación actual global.

Todo este desarrollo que se está introduciendo tiene su origen en la síntesis de los sistemas basados en redes neuronales artificiales, constituidas a partir del concepto o idea matemática de la neurona artificial, que se expone a continuación.

2.2.1 Redes Neuronales Artificiales

Una neurona artificial, es un artilugio matemático, de manifiesta bioinspiración, que intenta replicar el comportamiento de las células homónimas, presentes en los seres vivos dotados de sistema nervioso.

El funcionamiento de dicha unidad viene articulado por una función matemática denominada función de activación o *non-linearity*, y que proporciona un resultado de salida, tras la aplicación sobre la suma de un término denominado *bias* y una serie de entradas o *inputs* multiplicadas cada una de ellas por un factor correspondiente, llamado peso y caracterizado por la letra w, del inglés *weight*. Esta idea, fue inicialmente desarrollada por W. S. McCulloch y W. Pitts (McCulloch & Pitts, 1943).

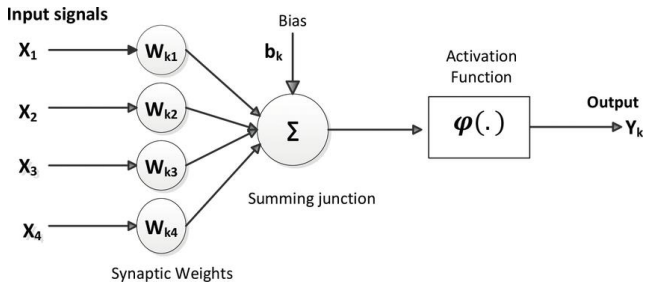

Fig. 4 *Esquema de una neurona artificial basada en la idea original de McCulloch y Pitts. Fuente: https://www.researchgate.net/publication/323465059/figure/fig2/AS:599207769554946@1519873673906/McCulloch-Pitts-computational-model-of-a-neuron.png*

La unidad descrita, actuaría como un nodo en una red, tras disponer un conjunto interconectado: las entradas a un nodo proceden de otros nodos o neuronas, de la misma manera que las salidas de una unidad desembocan en otras unidades. Este conglomerado, recibe el nombre de red neuronal artificial. Las unidades o neuronas que reciben directamente información del exterior, que se pretende inyectar a la red, constituyen lo que se denomina **capa de entrada**. A continuación, puede haber un número variable de capas, a su vez con un número variable de neuronas cada una de ellas, constituyendo lo que se denomina como capas intermedias o **capas ocultas**, *hidden layers,* en inglés. La ultima capa, que proporciona el resultado que aspira a ser interpretado, constituye la **capa de salida**. El conjunto de capas, número de neuronas en cada una de ellas, así como las conexiones entre ellas y las funciones de activación de cada nodo, integran el concepto de **arquitectura de la red neuronal**, y que engloba de forma general la complejidad de la red, así como su capacidad para resolver los

problemas planteados. Y es precisamente, esa capacidad la que hace que el conjunto se considere como un sistema, cognitivo, artificial.

Observando el sistema descrito en conjunto, se puede entender éste como una función matemática no lineal, y que, dado un conjunto de valores de entrada, ésta proporciona un conjunto de valores de salida, que dependen del conjunto de pesos (w_i) y *bias* (b_i), denominados **parámetros de la red**, de cada una de las neuronas. Así, en el contexto del aprendizaje supervisado, donde se dispone de una serie de datos etiquetados, se pretende buscar el conjunto de valores para los parámetros de la red, que den como salida la respuesta esperada. La diferencia entre la función ideal, que viene dada por el conjunto de datos disponible, y la salida obtenida por la red para un conjunto determinado de valores de los parámetros es lo que se denomina **función de pérdida** o *loss function,* en inglés y que se desea minimizar. Esto convierte la situación planteada en un problema de optimización y las acciones realizadas para llevar a cabo la obtención de los mejores valores para los parámetros de la red recibe el nombre de **entrenamiento**.

Un entrenamiento de una red neuronal se desarrolla en una serie de ciclos o *epochs*, en los que la red contempla el conjunto de datos disponible para el entrenamiento, y es en general un proceso computacionalmente costoso. A su vez, cada ciclo se compone de una serie de pasos o *steps,* donde se toma en cada uno un subconjunto de datos llamado *batch*, o *mini-batch* para el que se ejecutan dos fases: *forward pass* y *backward pass*. Al realizar el *fordward pass*, para un determinado conjunto de parámetros, se calcula la salida de la red, y se determina el valor de la función de pérdida. Durante el *backward pass*, se propaga el valor del error hacia atrás, con la intención de modificar los valores de los pesos y los *bias*, de manera que en la siguiente fase el error obtenido sea menor. La determinación sobre cuánto se debe modificar cada peso viene determinada por la aplicación de la técnica conocida como ***gradient descent*** en inglés, o descenso del gradiente. Consiste a grandes rasgos en propagar hacia atrás por la red, el valor de la derivada o gradiente de la función de activación de cada neurona, mediante la aplicación de la regla de la cadena, del cálculo diferencial. Así se consigue avanzar hacia el valor del conjunto de parámetros que minimizan la función de pérdida. Los algoritmos que se encargan de realizar dicha tarea reciben el nombre de algoritmos de retropropagación o ***backpropagation***, en inglés y las diferentes técnicas empleadas se conocen como **optimizadores**.

El aumento de datos disponibles, así como la capacidad de computo de los ordenadores, junto con el desarrollo de las técnicas de entrenamiento, y más recientemente los entornos de trabajo que simplifican la programación necesaria para determinar las arquitecturas de los sistemas cognitivos artificiales y llevar a cabo los entrenamientos, han contribuido a la

evolución y desarrollo de estos sistemas permitiendo la aplicación en muy diversos entornos, de una manera relativamente rápida y al alcance de cada vez más usuarios.

Existen, por su arquitectura, muy diversos tipos de redes neuronales, siendo en este punto las redes neuronales convolucionales, las que requieren el interés pare el desarrollo de este trabajo.

2.2.2 Redes Neuronales Convolucionales

Volviendo sobre la idea, de que cada variable de entrada a una red neuronal constituye un aspecto o *feature*, en el contexto de la visión por computación donde la entrada de información son fundamentalmente imágenes, mapas de pixeles, el número de pesos por cada neurona puede llegar a ser intratable computacionalmente, además de dotar al sistema de un exceso de susceptibilidad de aprendizaje debido al elevado número de parámetros empleados. Como solución a este problema se desarrolló el concepto de las redes neuronales convolucionales, que son capaces de tratar con colecciones de entradas como imágenes a color, sin emplear tan elevado número de parámetros, y pudiendo trabajar con aspectos espaciales de dichas entradas. Al hablar de imagen a color como dato de entrada, se está suponiendo un volumen de valores de entrada, 3 capas consecutivas, una por cada componente de color, como es el caso del sistema RGB (Red, Green Blue) y cada capa integrada por una matriz cuyas dimensiones son los pixeles que forman el plano espacial de la imagen. Los valores de cada uno de esos pixeles están entre 0 y 255, indicando esto la intensidad de luz relativa a cada color.

El nombre proviene del tipo principal de capa que las compone, la capa convolucional o *convolutional layer,* en inglés. Este tipo de capa se fundamenta en la presencia filtros, un volumen de parámetros cuyas dimensiones del plano espacial se eligen, y cuya profundidad debe coincidir con la profundidad del volumen de datos de entrada a la capa convolucional. Ese filtro, se va a deslizar por todo el espacio planar de los datos de entrada, con un determinado paso o avance de pixeles, también elegido. En cada posición, se va a realizar la operación que da nombre a la capa, una convolución, que consiste en obtener un único resultado a partir de la suma de cada uno de los valores del volumen de entrada, multiplicados por cada uno de los pesos, parámetros del filtro, espacialmente correspondientes. Así se obtiene un único valor para cada una de las posiciones que el filtro recorre, obteniendo finalmente lo que se denomina mapa de características, *feature map* en inglés. Se obtiene una nueva capa o *feature map* por cada filtro que integra una capa convolucional. Se obtiene como salida un volumen de datos cuyas dimensiones espaciales dependen del tamaño del

filtro y paso elegidos, y cuya profundidad viene determinada por el numero de filtros empleados. El conjunto de parámetros que definen la capa son por tanto los pesos (y un *bias*) que componen cada filtro, evidenciando de este modo, la reducción en el número de parámetros necesarios que ofrece esta variante de red neuronal.

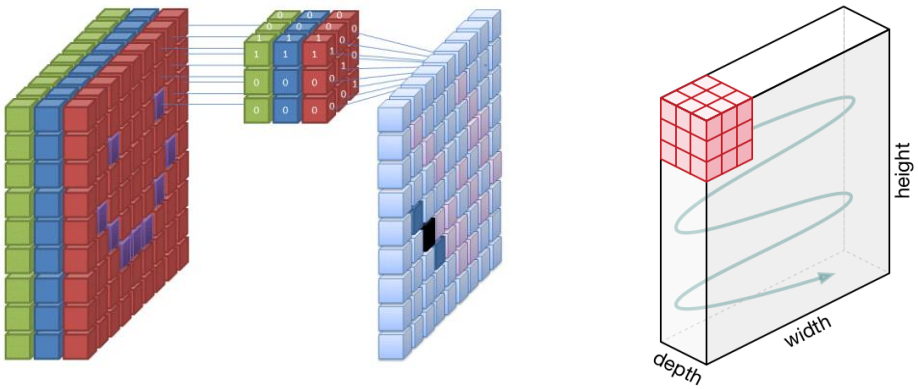

Fig. 5 *Izda.: Ilustración de un filtro de dimensiones 3x3 actuando sobre volumen de de entrada. Fuente: https://upload.wikimedia.org/wikipedia/commons/9/95/Convolutional_Neural_Network_with_Color_Image_Filter.gif. Licencia Creative Commons. Dcha: Ilustración del deslizamiento del filtro a través del volumen de entrada. Fuente: https://miro.medium.com/max/652/0*_BbDhaxqSZP3Yomf.*

Además de las capas convolucionales, es habitual emplear capas Max Pooling. Estas capas constan de un único filtro, que no tiene además parámetros y que realiza únicamente la operación de “máximo”, es decir, en cada punto de operación el único valor que se conserva es el máximo absorbido por el filtro. Esto hace que dichas capas no añadan parámetros a la red, y además reducen la dimensionalidad espacial del volumen de datos de entrada, simplificando el flujo de información que atraviesa la red. Se suelen colocar intercaladas con las capas convolucionales.

Otro tipo de capas, que se encuentran en las arquitecturas típicas son las denominadas *fully connected*. Son capas como las descritas en la sección Redes neuronales Artificiales, integradas por un número determinado de neuronas y suelen componer el final o salida de las arquitecturas previas, proporcionando la parte que “decide” o “clasifica” a partir de lo extraído en el conjunto de capas previas, cuando se emplea la red para tales fines. De hecho, el numero de neuronas en la última capa o capa de salida, se corresponde con el numero de clases a predecir. Empleando una función de activación con valores de salida entre 0 y 1, se

interpreta el valor de salida de cada neurona de la capa de salida, para una determinada entrada o imagen, como la probabilidad de pertenencia a la clase que representa.

Desde que se considerara, en 1998 el primer caso práctico de red convolucional, aplicada al reconocimiento de dígitos manuscritos en el correo postal (LeCun et al., 1998), la evolución tecnológica mencionada en cuanto a disposición de datos y capacidad de computación de los sistemas artificiales ha propiciado su uso en múltiples campos y en especial en el campo de la visión por computación, en tareas como la clasificación de imágenes, detección de objetos y segmentación (Zheng et al., 2020). La capacidad para resolver lo mejor posible, cada situación planteada reside en la arquitectura de la propia CNN, proporcionando especialmente desde 2012 (Krizhevsky et al., 2017), un conjunto de soluciones, arquitecturas en definitiva, con ejemplos como AlexNet, GoogLeNet, VGG, o ResNet.

Matizar, como cierre de las dos subsecciones anteriores, que el adjetivo “profundo”, adjudicado a la parte mencionada del aprendizaje automático, proviene de la adición de más y más capas a los sistemas cognitivos descritos, otorgándoles complejidad como consecuencia de dicha “profundidad”, proporcionando esto mayor capacidad para resolución de problemas o tareas más complejas.

2.3 Uso actual de CNNs para detección de defectos en superficies

Dado que el concepto de detección de defectos en superficies es amplísimo, se centra la atención en el campo de la inspección, monitorización y diagnóstico en torno a la infraestructura civil. Uno de los primeros ejemplos de uso de redes convolucionales para tal fin es (Cha et al., 2017) demostrando efectividad y mejora frente a las anteriormente empleadas técnicas de procesamiento de imagen, gracias a la capacidad que otorga el cálculo automatizado de indicadores de presencia de daño durante el entrenamiento de las redes (Cha et al., 2018).

Otra de las primeras contribuciones, aunque no tan específica de la detección de defectos en superficies, si no de manera más genérica en el ámbito de la detección de objetos en imágenes, se encuentra en (Girshick et al., 2014), donde apoyándose en dicho paradigma a través del empleo de la división de las imágenes de entrada en regiones, da lugar al uso de las denominadas R-CNN (*Region based* CNNs). Éstas, tras dividir el espacio en las mencionadas regiones, hacen uso de una CNN para la extracción de un vector de

características de longitud fija y finalmente se emplea un clasificador, como máquinas de vector de soporte, para clasificar cada región y finalmente mostrar, la reconstrucción de los objetos (categorías) encontrados en la imagen. Sin embargo, un problema que presentan es el alto coste de tiempo empleado en la detección sobre cada entrada. Como mejora, se plantea en (Cha et al., 2018) en un ámbito ya específico de la detección de defectos en superficies estructurales, el uso de Faster R-CNNs, que permiten una clasificación global casi a tiempo real, con una precisión del 87,8%.

En línea con la automatización de la monitorización de daño estructural en edificios, debido al envejecimiento, está el ejemplo de (Bang et al., 2021) que empleando luz laser y cámaras de profundidad, consigue detectar grietas, delaminaciones y exposición de acero, empleando un también Faster R-CNNs. También en este contexto, es interesante la utilización que hace (Zheng et al., 2020) de las redes FCN (*Fully Convolutional Networks*), RCNN y RFCN (Richer Fully Convolutional Networks), componiendo un algoritmo inteligente que ofrece buenos resultados en aspectos relativos a la detección de faltas en la superficie de los edificios, puentes, presas, etc.,.

Relativo al mantenimiento de las estructuras de hormigón, está el ejemplo de aplicación de (Li & Zhao, 2019) que entrenando la arquitectura AlexNet, con 60.000 imágenes de grietas, y tras integrarla en un dispositivo smartphone, consigue en la práctica detectar dicho fenómeno con una exactitud del 95%.

No obstante, no siempre es fácil conseguir *data sets* tan amplios, conjuntos de ejemplos de fenómenos concretos etiquetados, que permitan entrenar modelos desde “0”, por ello un tipo de solución común consiste en emplear modelos de aprendizaje profundo, pre-entrenados (Gopalakrishnan et al., 2018). Estos modelos se han entrenado previamente con *data sets* mucho mayores, de manera que los parámetros obtenidos engloban reconocimiento de características específicas procedentes de imágenes, que resultan útiles a la hora de reutilizarlos con nuevos *data sets*, más sencillos y específicos, permitiendo entrenar únicamente la parte final del modelo global, es decir la parte relativa a la clasificación de las entradas. Esta técnica recibe el nombre de ***transfer learning***. Como ejemplo del uso de ésta técnica, a partir de imágenes tomadas por drones para la detección de defectos en superficies de infraestructuras, con una obtención del 90% de exactitud, se presenta (Gopalakrishnan et al., 2018).

Otro ejemplo de uso, concretamente en el ámbito de la evaluación del pavimento, es decir del estado de las carreteras, se encuentra en (Maniat et al., 2021) donde empleando imágenes de Google Street View, demuestran su aplicación efectiva frente a sofisticadas soluciones de ámbito comercial.

Otra visión interesante del uso de CNNs, volviendo al ámbito de monitorización del estado de edificios se encuentra en (Liu et al., 2019), donde se propone una arquitectura concreta, denominada DeepCrack, definida como una arquitectura de *aprendizaje profundo jerárquico* cuyo objetivo es no solo la detección si no la segmentación de las grietas detectadas. En esta misma línea, (Bhowmick et al., 2020) presentan el uso de la arquitectura U-Net, demostrando su efectividad en la consecución de la tarea en cuestión, a partir de vídeos tomados por drones. La obtención de la segmentación de las grietas detectadas se presenta además como un paso intermedio decisivo hacia la obtención de las propiedades geométricas (longitud, anchura, orientación) de las grietas. Un claro ejemplo de la utilidad de la obtención de estas propiedades se presenta en (Z. Zhu et al., 2011), donde se emplea para conocer la magnitud de las mismas originadas en infraestructuras localizadas en zonas sometidas a terremotos. Otro ejemplo de método interesante de obtención de las características geométricas basado en técnicas de procesamiento de imagen, además del empleo de redes convolucionales para la detección de grietas, se presenta en (Kim et al., 2020).

Como ejemplo de uso de red convolucional, jerárquica e híbrida, por el uso de un módulo de binarización para la obtención de un mapa de características de salida, se presenta (Q. Zhu et al., 2021), que es además una evolución de la mencionada red DeepCrack.

Además de la detección de grietas, entendiéndola como una clasificación binaria, el ejemplo de CMDNet muestra la posible clasificación de las imágenes procedentes del estado de infraestructuras en 5 clases diferentes: “intacta”, “grieta”, “exposición del acero”, “levantamiento” o “delaminación” y “goteo” o “fuga”. Este tipo de uso de red convolucional se caracteriza por emplear “ramas” de redes de atención, y su arquitectura está basada en la red VGG16, que se analiza más adelante. Consigue además una exactitud del 98,9% (Shin et al., 2020).

Finalmente, realizando una interpretación global de los aspectos más relevantes encontrados en común en la literatura que conforma el estado del arte se ha decidido destacar las siguientes cuestiones:

-Todos los artículos consultados están centrados en las labores de diagnóstico y mantenimiento de obras de ingeniería civil, pero ninguna con un enfoque destinado a la intervención pictórica posterior.

-De los ejemplos encontrados solo uno (Shin et al., 2020) analiza conjuntamente a otros fenómenos el tema de las delaminaciones en fachadas.

-Todas las implementaciones de CNNs emplean un tamaño de imagen de entrada próximo a 227x227 píxeles, por lo que el análisis de imágenes de entrada mayores se ve resuelto por el

uso de una ventana deslizante o descomposición de la imagen de entrada en sub-imágenes de dicho tamaño.

-El entrenamiento de una determinada arquitectura basada en redes convolucionales se atisba como clave para la sintetización de un clasificador que pueda discernir con la mayor exactitud posible zonas con delaminados, de zonas intactas. Para ello, es necesario contar con un gran conjunto de datos, imágenes del fenómeno o fenómenos a analizar, para poder extraer la cantidad de información necesaria, para desarrollar con éxito la tarea. En este sentido, además del volumen inicial destinado al entrenamiento, y para evitar consecuencias como el *overfitting* o el exceso de aprendizaje del conjunto de datos de entrenamiento, existen técnicas de ampliación de datos o *data augmentation*, que permiten insertar variabilidad al modelo y dotarlo de mayor capacidad de inferencia. Y como medida recurrida para casos en los que el *data set* de entrenamiento no es muy amplio, el uso del *transfer learning* se muestra efectivo y permite reducir tiempos de entrenamiento.

3. Objetivos y metodología de trabajo

Se exponen a continuación, el objetivo global que se persigue mediante el desarrollo de este trabajo, así como los objetivos específicos que marcan hitos concretos para lograrlo. Finalmente quedan reflejados los pasos a seguir en el plan descrito en el apartado metodología.

3.1. Objetivo general

Se proyecta completar un programa que, recogiendo un conjunto de fotografías de una fachada de edificio tomadas por un dron, pueda identificar correctamente las zonas donde haya deterioro superficial (delaminaciones y grietas), devolviendo un mapa para su localización y cuantificación, como herramienta de ayuda a las labores previas a tareas de mantenimiento o intervención pictórica sobre la superficie estudiada.

3.2. Objetivos específicos

Como objetivos específicos para lograr la meta descrita, se plantean los siguientes puntos:

- Crear un *data set* específico de delaminaciones con casos positivos y negativos.
- Seleccionar una arquitectura de CNN propicia para entrenarla con el *data set* creado y poder identificar los casos positivos que son objeto de estudio.
- Implementar el programa que gestione la entrada de datos y la salida con la información deseada.
- Testar la herramienta diseñada, con fotografías extraídas por al menos un dron real, sobre uno o varios edificios seleccionados.

3.3. Metodología del trabajo

En primer lugar, y como punto de partida del proyecto se va a confeccionar un *data set* específico de delaminaciones, levantamientos y desconchados, ya que no ha sido posible encontrar ninguno en la red global. Para ello, se van a extraer manualmente, con cámara digital, fotografías de paredes evidenciadas con tales fenómenos, obteniendo cada vez tanto el caso positivo como el negativo. Se va a realizar lo mismo con imágenes que no estén sujetas a derechos de autor como resultado de la búsqueda de términos descriptivos en la red, tanto en castellano como en inglés. Se pretende completar un conjunto de aproximadamente 2000 elementos, de los cuales la distribución entre positivos y negativos sea del 50%. Se determinará la resolución de los elementos del conjunto de datos, relevante para el siguiente punto.

Una vez obtenido este material, es el momento de comenzar a implementar una red neuronal convolucional, capaz de reconocer los casos positivos en imágenes susceptibles de contener zonas con delaminaciones. Las arquitecturas descritas anteriormente para fines similares son los modelos en los que dicha red neuronal se va a basar. Se trata por tanto de acceder a ejemplos implementados para comprender cómo han sido sintetizados y su posible usabilidad para el proyecto. A partir de ahí es el momento de desarrollar el código propio necesario para implementar los modelos y habilitar sus respectivos entrenamientos.

Una vez elegido un primer conjunto de modelos, éstos se van a entrenar con el conjunto de datos construido, empleando modificaciones, transformaciones en orientación, contraste y lateralización, sobre el conjunto de datos inicial, aprovechando las ventajas del *data augmentation* (Shin et al., 2020) con el objetivo de reducir el posible *overfitting* debido a que dicho conjunto podría no resultar suficiente debido a su extensión (Cha et al., 2018). Se empleará la mayor parte del *data set* confeccionado para el entrenamiento, dejando una parte para verificación y unos pocos ejemplos como evaluación final, test.

El hecho de completar unos primeros entrenamientos, con un conjunto de parámetros o hiperparámetros inicial, será un indicador de que los modelos, computacionalmente se han implementado correctamente y los resultados obtenidos apuntarán el nivel de efectividad a la hora de clasificar a partir del conjunto de datos proporcionado.

Tras interpretar los primeros resultados obtenidos, se trata a través del refinamiento del código inicial planteado, y la exploración de distintos valores de hiperparámetros, de encontrar el resultado que ofrezca mayor exactitud y menor función de pérdida. El modelo que mejores

resultados ofrezca del conjunto inicial explorado constituirá el motor de clasificación de la herramienta que se va a desarrollar.

La funcionalidad y utilidad del modelo construido se va a probar con imágenes varias de una fachada tomadas con un dron real. El resultado debe ofrecer información al humano acerca de la localización sobre la superficie de los puntos conflictivos.

Es en este punto, donde según los resultados obtenidos en el desarrollo hasta el momento, se deba elegir una estrategia para el flujo de trabajo de la herramienta de software:

-Confección inicial del mosaico a partir de las fotos tomadas, e inspección virtual, escaneado y clasificando sección a sección.

-Etiquetado inicial de cada una de las imágenes tomadas y posterior confección del mapa visual, con las imágenes ya clasificadas.

Finalmente, se realizará una valoración basada en una inspección visual sobre la efectividad de los resultados ofrecidos por la herramienta. La intención final es que el resultado sea extrapolable a áreas muchísimo mayores y con zonas de difícil acceso para su evaluación visual.

4. Identificación de requisitos

En base a los distintos aspectos de los requerimientos de la herramienta de software, se detallan las necesidades correspondientes a la creación del *data set*, implementación de un modelo clasificador DCNN, entornos de trabajo y toma de imágenes reales con dron, en los siguientes apartados.

4.1. Construcción del *data set*

Tal como se indica más arriba, uno de los puntos fundamentales del desarrollo de este trabajo, consiste en la creación de un conjunto de imágenes de entrenamiento específico del tipo de deterioro descrito.

Para ello, se han tomado 661 fotografías con una cámara Nikon Coolpix S3000, con 5 megapíxeles de resolución (2592x1944), de las que un 60% muestran un caso positivo del defecto frente al 40% que representan el negativo. A este conjunto inicial, se le ha añadido otro conjunto de imágenes tomadas con diversos terminales (smartphones) componiendo un total de 162 nuevas imágenes con la relación entre casos positivos y negativos al 65% y 35% respectivamente. A partir de este muestrario inicial, y conociendo que la resolución o dimensiones de entrada típicos a un sistema clasificador candidato se encuentran entre (224x224) y (227x227) píxeles, se ha considerado cada imagen como un mapa mayor, donde aparecen distintos y variados aspectos del fenómeno a estudiar, el desconchado o delaminado en cuestión.

Para ello se han desarrollado dos funciones, que han permitido obtener un conjunto de imágenes a partir de una mayor, así como una función capaz de renombrar todos los ficheros (imágenes) contenidas en una carpeta, para mayor orden y control del proceso. Estas funciones se describen más adelante en el capítulo 5.

Tras estudiar el tamaño (área) de píxeles que representa propiamente el fenómeno frente al área completa de las imágenes originales, se ha decidido hacer cortes o fragmentos de 648x648 píxeles. Así tras realizar el fragmentado masivo, tanto en las imágenes de casos positivos como en las de negativos, ambos conjuntos han alcanzado un numero de elementos muy superior. No obstante, al realizar esta acción, fragmentos de casos inicialmente positivos resultan en la obtención de algunos casos negativos, en función de la distribución inicial del defecto. Es por ello por lo que se ha revisado cada uno de los nuevos casos obtenidos, inspeccionando y reclasificando manualmente, y desechando en algún caso también si el

contenido de la imagen no mostraba claramente un caso positivo o negativo. De esta manera, se ha conseguido sintetizar un **conjunto final de 6452 imágenes, compuesto por 3226 casos positivos y los mismos negativos, revisados individualmente**. Se ha decidido nombrar a este conjunto "Delaminations2021".

Las imágenes originales completas o mapas referidos anteriormente también forman parte de ese *data set*, de manera que el sistema clasificador no solo aprenda partes del fenómeno, si no una visión global del mismo.

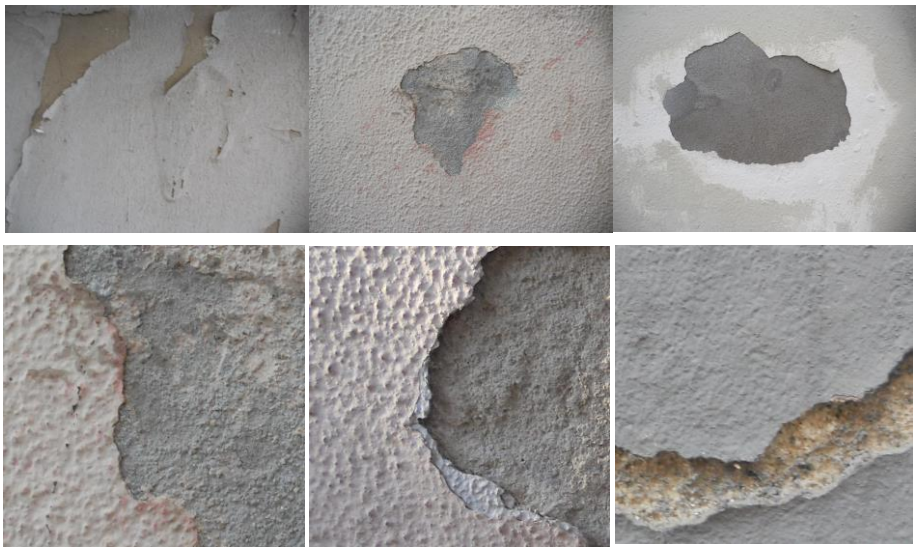

Fig. 6 *Muestras de imágenes de la clase positiva del data set confeccionado por el autpr,"Delaminations2021"; la fila de arriba está compuesta por imágenes completas (mapas) y las de la fila inferior son el resultado del fragmentado o corte.*

Además, se ha sintetizado un segundo *data set* de tamaño muy inferior, a partir de 5 imágenes (consideradas mapas) que no han entrado a formar parte del conjunto anterior, con el propósito de emplearlo como **conjunto de test**. Más adelante se hará referencia a él como "Test 2" y está compuesto, a través del método de fragmentación empleado antes, por 130 imágenes pertenecientes a ambos casos. Por otro lado, se ha sintetizado otro conjunto de test como resultado de hacer una búsqueda general de términos de designación del defecto en cuestión en internet, a partir de imágenes sin copyright. Este otro conjunto se ha generado como el anterior y está integrado por 430 imágenes. En este caso, las imágenes al tener muy diversos orígenes y condiciones (iluminación, ángulo, etc.,) pretenden proporcionar una medida adicional a la validación y al test del clasificador, para evaluar la capacidad de extrapolar, o inferir conocimiento del sistema, ante el fenómeno en nuevas condiciones.

4.2. Clasificador DCNN

Se precisa de un modelo clasificador, para la tarea de etiquetar las imágenes entrantes, como contenedoras de delaminados (positivas) o no contenedoras (negativas). Tal como se ha explicado en el apartado 2.1 la implementación de un modelo basado en redes convolucionales profundas constituye en primera instancia una buena aproximación para la resolución del problema. Uno de los aspectos que clave que focaliza una zona concreta del espectro de soluciones es el tamaño del data set con el que se va a entrenar el modelo. Para conjuntos del orden de 40K o 60K imágenes se han implementado redes profundas con 8 (Cha & Choi, 2017) y 12 (Kim et al., 2020) capas convolucionales, con resultados por encima del 95% de acierto. Sin embargo, para conjuntos de imágenes de un orden de magnitud inferior, se señala en la literatura como una buena solución la implementación de modelos bajo el paraguas del *transfer learning*.

En este sentido, una gran parte de los ejemplos consultados en cuanto a la tarea de clasificación de grietas, son adaptaciones de modelos complejos que han surgido a raíz de la competición *The ImageNet Large Scale Visual Recognition Challenge* (ILSVRC), basada en el hito en el campo de los *data sets,* ImageNet (Russakovsky et al., 2015), (Krizhevsky et al., 2017). Estos modelos, están disponibles, son instanciables desde la librería Keras, tanto la arquitectura como los pesos de los modelos pre-entrenados con el mencionado conjunto de datos ImageNet. La idea es considerar que las capas convolucionales ya entrenadas, en ese menester de clasificar entre 1000 clases diferentes, han adquirido una capacidad de extracción de información general suficiente, para que después un conjunto de capas “fully connected” sean entrenadas con el *data set* requerido y el modelo completo sea capaz de clasificar correctamente.

Implementar este tipo de red, requiere además una exploración del entorno de programación más adecuado, así como la realización de un conjunto de pasos previos y posteriores a la implementación de la arquitectura y el entrenamiento del modelo en sí.

4.2.1 Entorno de Trabajo y *frameworks*

Sobre el entorno de trabajo, se parte del uso de notebooks de Jupyter, albergando código Python, desde Anaconda. Esto permite ir probando y comprendiendo la envergadura del objetivo deseado, sobre todo desde el punto de vista computacional. Se contempla como siguiente paso, trabajar el entorno en Google Colab, ya que este permite, al ejecutarse en remoto, hacer uso de las GPU pudiendo acelerar esto el proceso de entrenamiento, siendo éste una actividad paralelizable.

Conviene recordar en este momento, cómo a nivel de definición, el conjunto de operaciones necesarias para implementar y entrenar correctamente una red neuronal profunda presenta muy alta complejidad. Controlar el amplísimo flujo de datos desde la perspectiva de la programación más básica es una vía dificultosa. Es por ello que, en los últimos años han ido creciendo opciones de software que lo facilitan, dando lugar a ecosistemas propicios para el desarrollo de redes más simplificado y dinámico.

La idea general que ha permitido esto es la concepción de una red neuronal como un grafo de computación: grafo dirigido con operaciones definidas en sus nodos y aristas que los conectan y que permiten el fujo de datos a través de los mismos. Así un *framework* ofrece un conjunto de estructuras abstractas creando un paradigma que permite, a través de la escritura de código sencillo, definir y entrenar diversas redes. La constante evolución de estos *frameworks*, ha llevado a garantizar los entrenamientos de las estructuras implementadas de manera optimizada, permitiendo hacer un uso muy eficiente de los recursos disponibles, empleando internamente librerías numéricas específicas.

TensorFlow, constituye un ejemplo muy notable de conjunto de librerías o *framework*, y que ha sido además desarrollado por Google. El nombre procede de entender los datos como tensores (*Tensor*) que fluyen (*Flow*) por el mencionado grafo de computación. Es de código abierto y se ha empleado en este trabajo a través de su API para Python.

Para un desarrollo, todavía más ágil en lo que a la definición de redes neuronales profundas y complejas se refiere, se ha empleado Keras, denominada como librería de alto nivel. A través de su API en Python, simplifica las instrucciones necesarias para poner en marcha modelos complejos. Esta librería corre sobre TensorFlow constituyendo esto el principal entorno empleado para el desarrollo del clasificador DCNN de este trabajo.

4.2.2 Etapas del desarrollo para la implementación de los modelos

Se señalan a continuación todos los puntos de trabajo que han requerido especial detenimiento y comprensión para la correcta implementación de los distintos modelos en el avance hacia la consecución del objetivo global.

Pre-procesamiento

Por un lado, las imágenes procedentes del *data set* elegido para entrenar, deben disponerse para poder ser pasadas al modelo, con el formato y estructura de archivos que éste requiera. Se trata de una fase de pre-procesado, donde además los datos procedentes de cada imagen deben ser estandarizados.

Data Augmentation

Se trata de una técnica muy popularizada de regularización, y altamente recomendada para un problema como el planteado en este trabajo. Es una medida que pretende reducir en la medida de lo posible el exceso de ajuste del modelo, especialmente en este caso, debido a una cantidad relativamente baja de datos de entrenamiento. Durante el entrenamiento, a la hora de ir tomando los subconjuntos de imágenes, se realiza una serie de operaciones o ligeras modificaciones de los datos originales, que los alteran a nivel de píxel, pero mantienen en esencia la clase a la que pertenecen. Esto aumenta la varianza o variabilidad de los datos de entrenamiento favoreciendo que el modelo no se sobreajuste, y no pierda así capacidad de generalización.

Validación Cruzada (cross-validation)

Un punto muy relevante a tener en cuenta, en general, en los modelos de aprendizaje automático es el empleo de métodos de validación cruzada, *cross-validation* (CV). A la hora de separar entre los conjuntos de entrenamiento y validación o test, la distribución de clases, en los nuevos subconjuntos puede no estar balanceada, de manera que dé lugar a un entrenamiento pobre para alguna de las clases, o viceversa, un modelo excesivamente entrenado en cierta clase, y además esa distribución desbalanceada puede producir, de manera aditiva a través del conjunto de validación o test, unos resultados de evaluación del modelo, falsos, clasificando excesivamente bien alguna de las clases o no evaluando

realmente la clasificación de cierta clase. Para obtener unos resultados de evaluación lo más fiable posible, se recomienda emplear métodos de validación cruzada, como *K-fold*, básicamente tomar "K" particiones diferentes de subconjuntos.

En el caso que atañe a este trabajo, inicialmente, las clases están balanceadas y además están separadas estructuralmente en diferentes archivos. De manera que al hacer la partición o *split* del conjunto de datos, el modelo realiza una partición completamente balanceada, a nivel de número de elementos de cada clase. No obstante, los datos de entrenamiento, en cuanto a la varianza que pueden aportar distintos repartos de subconjuntos, sí que es un punto importante a tener en cuenta, es decir, garantizar o conocer el impacto que puede tener una determinada partición, en lo que a variabilidad o varianza en los datos de aprendizaje se refiere.

Entrenamiento del modelo

Esta parte, aunque propiamente la realizan de manera automática todo el conjunto de algoritmos, funciones y métodos implementados, requiere de sumo cuidado en la elección de los valores del conjunto de hiperparámetros ya que puede desembocar en que la red no aprenda correctamente y no alcance valores satisfactorios eficientemente, o incluso que ni los alcance.

La propia arquitectura de la red neuronal junto con los parámetros modificables de las diferentes capas que las componen, suponen una parte importante del conjunto de hiperparámetros.

Los parámetros fundamentales propios del entrenamiento, son el optimizador, la función de pérdida y la métrica de evaluación. El optimizador, que es el algoritmo encargado de llevar a cabo la retropopagación del error obtenido a través de la red, contiene además la elección de uno de los parámetros más críticos del entrenamiento: el ***learning rate (lr)*****.** Es una magnitud que expresa la velocidad de aprendizaje de la red, la magnitud del avance al realizar la minimización de la función de coste o pérdida (loss), y que puede desembocar, si su valor es excesivamente alto, en *overshooting,* es decir que la función de coste aumente en lugar de reducirse; o puede resultar en entrenamientos excesivamente lentos, si su valor es muy bajo.

Por otro lado, recordar que la función de pérdida representa la formulación matemática del error o diferencia entre los resultados ideales procedentes de los datos etiquetados y los resultados obtenidos para los datos de entrada a través del conjunto de operaciones que representa la red, y cuyo objetivo fundamental del sistema es minimizar su valor.

Por último, la métrica de evaluación es la formalización del criterio de evaluación del rendimiento de la red, necesaria para comprender el alcance de la misma y poder tomarla como criterio de ordenación entre las distintas variantes y modelos implementados.

Una vez seleccionados todos estos parámetros, el desarrollo del entrenamiento va a tener lugar de manera cíclica a través del número de *epochs* seleccionado. Recordar que en cada ciclo, la red habrá tenido en cuenta todos los ejemplos del conjunto de datos de entrenamiento, agrupados en los mencionados subconjuntos o *batches*. En cada ciclo, se puede comprobar el valor de la función de pérdida y de la métrica de evaluación elegidas. Estos valores pueden además emplearse como criterios de parada del entrenamiento, o de almacenamiento del conjunto de parámetros de la red alcanzados mediante el empleo de los recursos denominados ***callbacks***, que se verán más adelante.

Evaluación del modelo

Además de emplear un conjunto de validación, es conveniente contar con uno o varios conjuntos de test, que permitan ir evaluando el comportamiento del modelo entrenado, así como implementar funciones que permitan visualizar la evolución de los valores de la métrica de evaluación en función de los *epochs* del entrenamiento.

4.3. Flujo de trabajo general

De cara al desarrollo de la herramienta, se evidencia el siguiente flujo de trabajo a realizar:

1.- Elección del modelo que haya mostrado el mejor resultado tras la validación y con los conjuntos de test preparados.

2.-Desarrollo de las distintas partes de la herramienta:

A) Entrada de un mapa o fotografía de gran tamaño.

B) Segmentación en cuadrícula o *frames* de dicho mapa y disposición de cada unidad para poder ser procesada.

C) Clasificación de cada uno de los elementos de la cuadrícula.

D) Coloreado de cada una de las clasificaciones en función del resultado

F) Salida de una copia del mapa inicial insertado, con la clasificación coloreada y un conjunto de datos de información sobre el resultado obtenido.

4.4. Toma de imágenes mediante dron

Ya que el propósito final de la herramienta es clasificar imágenes a partir de las fotografías tomadas por un dron, de una fachada real, se han tomado imágenes con un modelo DJI Air 2S, en vuelo manual, sobre dos fachadas de edificios, que visualmente revelan la presencia de descascarillados. Se ha realizado un barrido manual, a distancia constante al edificio tomando fotografías progresivamente. El material recogido, conforma la información necesaria en cuanto a imágenes para probar la herramienta, además de proporcionar en los metadatos de cada imagen, donde se encuentran las coordenadas GPS y altura, del momento en el que se toma cada fotografía.

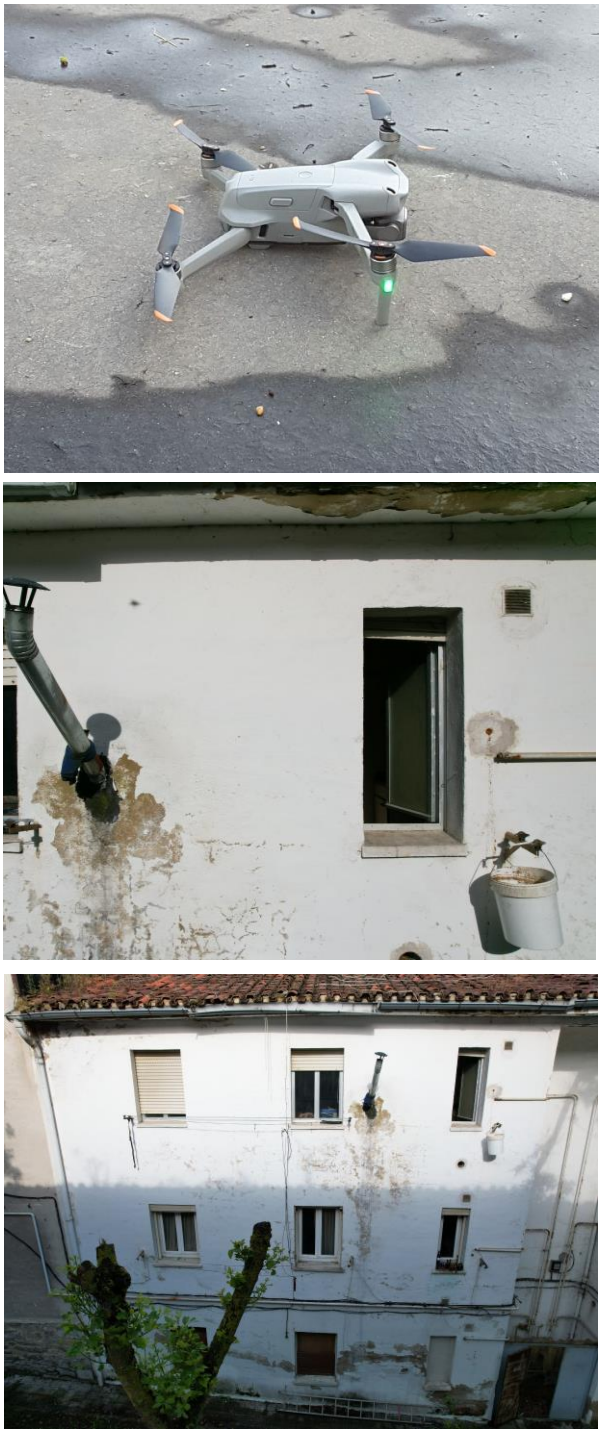

Fig. 7 *Imágenes, de arriba a abajo: Dron antes de tomar vuelo; Detalle de una de las fachadas escaneada; Imagen global de una de las fachadas escaneadas.*

5. Descripción de la herramienta software desarrollada

En este capítulo, se describen de manera precisa y técnica, las distintas labores realizadas durante todo el proceso de sintetizado y evaluación de la herramienta desarrollada para el diagnóstico de fachadas previa a la intervención pictórica. Está dividido en dos grandes bloques o subcapítulos, en los que se desglosa una primera parte acerca de la implementación y elección del modelo clasificador basado en redes convolucionales profundas, y una segunda parte de configuración del flujo de información propio de la herramienta, que la hacen funcional y evaluable.

5.1. Implementación de modelos para el clasificador DCNN

A partir de la información consultada, mostrada en la sección dedicada al estado del arte, en lo que al problema central de este trabajo se refiere, se ha decidido implementar tres modelos diferentes: XCeption, VGG16 y ResNet50. En los siguientes subapartados se describen los pasos realizados para implementar y entrenar cada uno de los modelos.

5.1.1 Desarrollo de código para ejecución

Tal como se ha redactado, el tándem de librerías empleadas para el desarrollo del código necesario ha sido TensorFlow y Keras. Tras indagar en ejemplos más simples e inspirándose en ejemplos propuestos en ambas documentaciones correspondientes, (https://keras.io/api/ , https://www.tensorflow.org/api_docs/python/tf) se ha creado un guión para notebook común como punto de partida donde se aborda el pre-procesamiento de los datos, *data augmentation*, implementación de la arquitectura específica del modelo, compilación del modelo y otras funciones relativas al entrenamiento, y ejecución del entrenamiento y evaluación, estas dos últimas se comentan en profundidad en las dos últimas secciones respectivamente.

En el extracto de código de la figura 8, se observa cómo se ha empleado la clase ImageDataGenerator() para crear los objetos que contienen las diferentes acciones de *data augmentation* (rotación, brillo, recortado, volteo) para el posterior conjunto de entrenamiento y de reescalado (rescale=1.0/255) para ambos conjuntos de entrenamiento y validación. El

motivo de realizar esta operación es el siguiente: las imágenes que constituyen los datos de entrada, son *arrays* de tres dimensiones: disposición planar de píxeles (altura y anchura) y profundidad, que está compuesta de tres capas “RGB”, donde en cada una se codifican los valores entre 0 y 255 relativos al rojo (Red), verde (Green) y azul (Blue) respectivamente. Con la operación de reescalado, lo que se está haciendo es trasladar cada uno de esos valores del rango descrito, al rango [0,1] para posibilitar los cálculos a través de la red neuronal.

En la celda siguiente, lo que se realiza es la aplicación del método flow_from_directory() sobre los objetos creados anteriormente, de manera que se obtienen, bajo los nombres train_dts y validation_dts , lo que se denomina como “Iteradores de Directorio”, que son fundamentalmente tuplas de las imágenes de entrada con sus etiquetas, designadas a partir de la estructura subyacente a la ruta asignada a través de la variable “filepath”. Se establece además el tamaño del *batch*, o subconjunto de datos de entrenamiento, fijándolo en 32 imágenes. El número de clases se infiere de la distribución de carpetas en las que los datos son almacenados, en este caso dos: ‘Positivos’ y ‘Negativos’.

In [ ]:

```
train_data_gen=ImageDataGenerator(
    rotation_range=90,
    brightness_range=[0.2,0.5],
    shear_range=0.2,
    fill_mode="reflect",
    horizontal_flip=True,
    vertical_flip=True,
    rescale=1.0/255.,
    validation_split=0.2
)
validation_data_gen  =  ImageDataGenerator(
    rescale = 1.0/255.,
    validation_split=0.2
)
```

In [ ]:

```
image_size = (224, 224)
batch_size = 32
file_path = "C:\\Users\\abalz\\Escritorio\\Asier\\Master IA\\Asignaturas\\Trabajo Fin Maste
save_file_path = "C:\\Users\\abalz\\Escritorio\\Asier\\Master IA\\Asignaturas\\Trabajo Fin

train_dts = train_data_gen.flow_from_directory(
    file_path,
    target_size=image_size,
    color_mode="rgb",
    classes=None,
    class_mode="binary",
    batch_size=batch_size,
    shuffle=True,
    seed=1000,
    save_to_dir=save_file_path, #probar
    save_prefix="DA_",
    save_format="jpg",
    follow_links=False,
    subset="training",
    interpolation="nearest",
)
validation_dts = validation_data_gen.flow_from_directory(
    file_path,
    target_size=image_size,
    color_mode="rgb",
    classes=None,
    class_mode="binary",
    batch_size=batch_size,
    shuffle=False,
    subset="validation",
    interpolation="nearest",
)
```

Fig. 8 *Extracto del código empleado en la parte de pre-procesamiento y data augmentation.*

En la figura 9, se muestra una representación de 25 imágenes, y cómo han sido etiquetadas automáticamente, designadas como 1 para la clase positiva, donde se puede comprobar visualmente el desconchado o descascarillado, y 0 para la clase negativa, donde las muestras no representan defecto.

```
In [6]:
plt.figure(figsize=(10, 10))
for images, labels in train_dts.take(1):
    for i in range(25):
        ax = plt.subplot(5, 5, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(int(labels[i]))
        plt.axis("off")
```

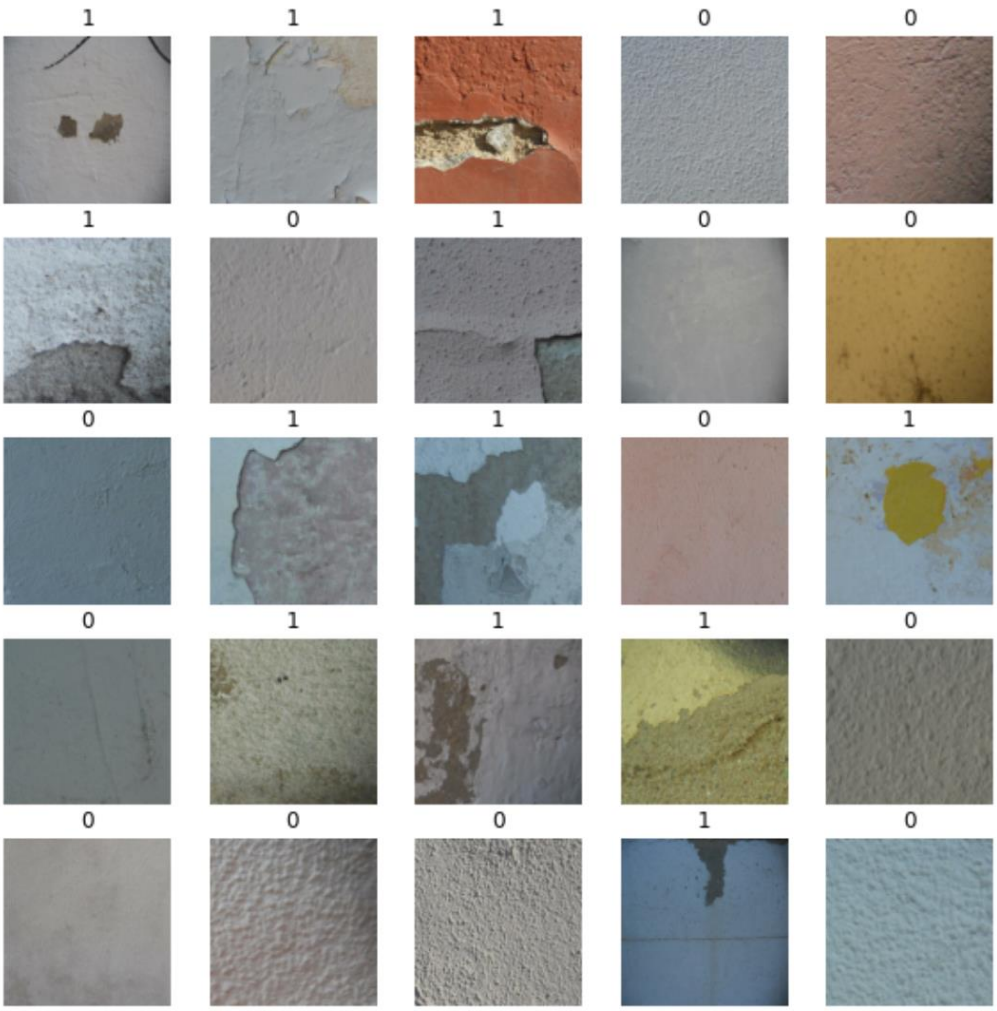

Fig. 9 *Muestra de 25 imágenes del conjunto de entrenamiento, etiquetadas automáticamente como positivas (1) y negativas (0).*

De manera análoga a los datos de entrenamiento, se preparan los conjuntos de test para su posterior evaluación, como se puede comprobar en la figura 10.

```
In [ ]:
test_2_path = "C:\\Users\\abalz\\Escritorio\\Asier\\Master IA\\Asignaturas\\T
test_2_gen = ImageDataGenerator() # considerar incluir  rescale=1./255.
test_2_gen = test_2_gen.flow_from_directory(test_2_path,
                                            target_size=(224,224),
                                            batch_size=32,
                                            shuffle=False,
                                            class_mode='binary')
```

```
In [ ]:
test_int_path = "C:\\Users\\abalz\\Escritorio\\Asier\\Master IA\\Asignaturas\
test_int_gen = ImageDataGenerator() # considerar incluir  rescale=1./255.
test_int_gen= test_int_gen.flow_from_directory(test_int_path,
                                               target_size=(224,224),
                                               batch_size=32,
                                               shuffle=False,
                                               class_mode='binary')
```

Fig. 10 *Preparación de los conjuntos de test "Test 2" e "Internet", para la evaluación posterior de los modelos.*

A continuación de los expuesto hasta ahora, vendría la implementación de la arquitectura propia de cada modelo, que se detalla en la siguiente sección.

Una vez creado el modelo, se procede a compilar éste haciendo para ello uso del método compile() y que es donde se eligen los parámetros como el optimizador, la función de pérdida (loss), y la métrica de evaluación.

A la hora de elegir el optimizador, para los modelos empleados en el desarrollo de la herramienta, se han empleado SGD (del inglés *Stochastic Gradient Descent* o descenso del gradiente estocástico) y Adam (procede de *adaptative moment estimation*) considerado como una evolución o mejora del algoritmo propio del descenso del gradiente y que en general se comporta bastante bien al ser implementado en tareas de aprendizaje profundo (Kingma & Ba, 2014).

En cuanto a la función de pérdida, son diversas las posibilidades ofrecidas por parte de las librerías empleadas para la implementación de los modelos, no obstante, para el tipo de problema de clasificación, binaria, que se desea resolver la más indicada se denomina ‘binary_crossentropy’, o también conocida como “log loss”.

Por último, como métrica de error o métrica de evaluación del modelo se ha decidido, en base a lo advertido en la literatura consultada, emplear la exactitud o ratio de éxito: ‘accuracy’. Es un indicador de lo “cerca” que está el conjunto de resultados obtenidos del resultado objetivo y cuya magnitud está comprendida entre 0 y 1.

```
In [ ]: .compile(
            optimizer=keras.optimizers.SGD(learning_rate=0.001),
            loss='binary_crossentropy',
            metrics='accuracy'
        )
```

Fig. 11 *Ejemplo de aplicación del método compile(), todavía sin aplicar a un modelo concreto.*

Los siguientes pasos relativos a los entrenamientos y evaluación de los modelos, se detallan en las dos últimas secciones respectivamente.

5.1.2 Arquitecturas DCNN

A continuación, se detallan las características que dan lugar a los tres modelos de arquitecturas de redes neuronales profundas empleadas en este trabajo.

XCeption

Xception es el nombre con el que se conoce a este tipo de arquitectura de red neuronal convolucional, profunda. Fue desarrollada por François Chollet, como una evolución de la red previa, Inception. Consiste en una estructura basada completamente en convoluciones separables en profundidad, desacoplables (Chollet, 2016).

El código original que permite crear todas las capas que conforman el modelo, ha sido tomado de la fuente https://keras.io/examples/vision/image_classification_from_scratch/. Tras implementar el modelo, a través del método summary(), se puede visualizar el conjunto de capas que compone el modelo, así como el número de parámetros total y generado en cada capa.

El modelo final implementado, consta de 2.773.913 parámetros entrenables. La capa final de salida es una capa densa, compuesta por una neurona y activación “sigmoid” para la determinación de la clase binaria de salida.

VGG16

Se trata de un arquitectura de red neuronal convolucional específica, desarrollada por un grupo de investigadores (Grupo de Geometría Visual, de ahí el nombre por sus siglas en inglés) de la universidad de Oxford (Simonyan & Zisserman, 2015), alrededor de la mencionada competición *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC).

Su estructura está compuesta fundamentalmente por 16 capas, administradas en 5 bloques de capas convolucionales seguidas de capas MaxPooling para la reducción de la dimensionalidad. Este tramo alberga 14.714.688 parámetros, que forman la parte pre-entrenada del modelo, mientras que contiene 6.455.809 entrenables, procedentes de las tres ultimas capas añadidas para la labor de clasificación. Se trata de dos capas “fully connected” de 256 y 128 neuronas, con activación “ReLU”, seguidas de la capa de salida con 1 neurona y activación “sigmoid” para la determinación de la clase binaria de salida.

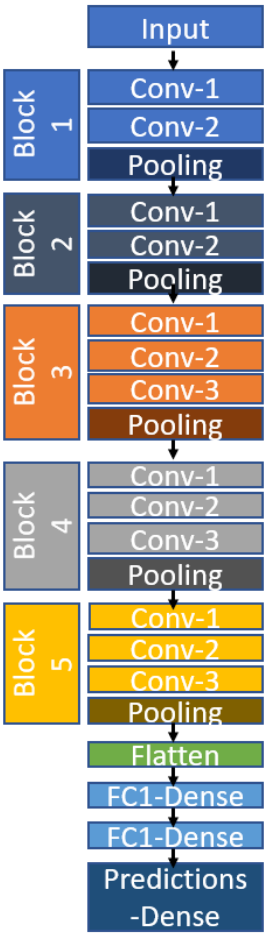

Fig. 12 *Esquema de una red VGG16. (Fuente: https://towardsdatascience.com/a-demonstration-of-transfer-learning-of-vgg-convolutional-neural-network-pre-trained-model-with-c9f5b8b1ab0a)*

ResNet50

Se trata de otro tipo de red que obtuvo muy buenos resultados en la competición ILSVRC, en 2015. Su nombre procede del término “residual networks”. Resulta de una combinación de bloques convolucionales y bloques identidad, además de las conexiones internas “skip”, donde reside el potencial de esta arquitectura.

Las funciones identity_block(), convolutional_block() y ResNet50() han sido implementadas a partir de la fuente https://machinelearningknowledge.ai/keras-implementation-of-resnet-50-architecture-from-scratch/.

Tras implementar el modelo, el método summary() revela que la red consta de 32.009.601 parámetros, de los cuales sólo algo menos de 9 millones van a ser entrenados, ya que el resto se congelan tras haberlos cargado del modelo pre-entrenado.

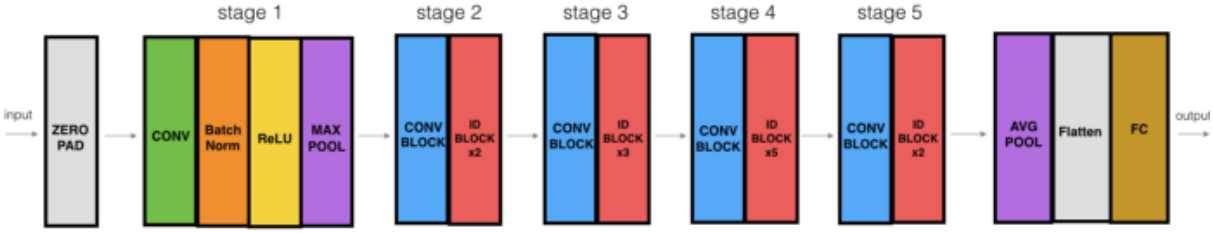

Fig. 13 *Esquema de la red ResNet50(Fuente: https://machinelearningknowledge.ai/keras-implementation-of-resnet-50-architecture-from-scratch/).*

5.1.3 Entrenamientos

Antes de comenzar con los entrenamientos, se ha tenido que determinar el uso de los mencionados *callbacks*, los objetos que permiten realizar acciones durante los entrenamientos, antes y después de cada *epoch*. Por un lado, se ha hecho uso de la clase ModelCheckpoint para poder guardar, a medida que avanza el entrenamiento, los valores de los pesos conseguidos hasta el momento en función de cierto criterio establecido para el valor de la métrica de evaluación. Por otro lado, se ha empleado la clase EarlyStopping para, en caso de no obtener mejora en las métricas designadas a lo largo de cierto número de *epochs*, para el entrenamiento, con el objetivo de no perder tiempo ni recursos de computación, ni llevar al modelo a *overfitting*. En este caso se ha contemplado establecer el límite de 20

epochs sin mejora, teniendo en cuenta que en general, se ha establecido un total de 50 *epochs* por entrenamiento.

```
In [38]:
epochs= 50
mc = keras.callbacks.ModelCheckpoint(
    '/content/drive/MyDrive/DCNN Models/XCeption/2G_XC_06.h5',
    monitor='val_accuracy',
    mode='max',
    verbose=1,
    patience=20)
es= keras.callbacks.EarlyStopping(
    monitor='val_accuracy',
    mode='max',
    verbose=1,
    patience=20)

In [39]:
History_6 = model_xc_3.fit(
    train_dts, validation_data=val_dts, epochs=epochs,verbose=1,callbacks=[mc,es])
```

Fig. 14 *Extracto de código donde se declara el número de epochs, los objetos 'callback' y la implementación del método fit() para inicio del entrenamiento.*

Para comenzar el entrenamiento, se ha empleado el método fit(), sobre el modelo creado, especificando las fuentes de datos de entrenamiento y validación, *epochs* y *callbacks*, tal como se observa en la figura 14.

5.1.4 Elección del mejor modelo

Una vez concluidos los entrenamientos, se ha empleado el método history[], sobre la variable que designa el entrenamiento, para junto con la librería matplotlib, representar la métrica de evaluación, *accuracy* y la función de pérdida del entrenamiento y validación, en cada *epoch* durante todo el entrenamiento, tal como se observa en la figura 15. Permite visualizar rápidamente la evolución de los mismos, comprobando si hay *overfitting*, o estancamiento en la mejora de la métrica de evaluación. En el caso mostrado en dicha figura, tanto en los valores observables del extracto, como la representación gráfica, muestran un ejemplo de *overfitting*, donde el valor de exactitud de entrenamiento ha alcanzado el máximo (1), y el valor correspondiente a la validación se estanca en 0.9860, durante un entrenamiento de un modelo basado en la red ResNet50.

```
Epoch 37/50
162/162 [==============================] - 32s 190ms/step - loss: 6.7718e-07 - accuracy: 1.0000 - val_loss: 0.1412 - val_
accuracy: 0.9860
Epoch 38/50
162/162 [==============================] - 32s 189ms/step - loss: 6.1601e-07 - accuracy: 1.0000 - val_loss: 0.1422 - val_
accuracy: 0.9860
Epoch 39/50
162/162 [==============================] - 32s 189ms/step - loss: 5.4525e-07 - accuracy: 1.0000 - val_loss: 0.1432 - val_
accuracy: 0.9860
Epoch 40/50
162/162 [==============================] - 32s 188ms/step - loss: 4.9895e-07 - accuracy: 1.0000 - val_loss: 0.1442 - val_
accuracy: 0.9860
Epoch 41/50
162/162 [==============================] - 32s 190ms/step - loss: 4.5591e-07 - accuracy: 1.0000 - val_loss: 0.1453 - val_
accuracy: 0.9860
Epoch 42/50
162/162 [==============================] - 32s 189ms/step - loss: 4.2646e-07 - accuracy: 1.0000 - val_loss: 0.1463 - val_
accuracy: 0.9860
Epoch 00042: early stopping
```

Fig. 15 *Arriba: Extracto de la salida generado durante el entrenamiento del modelo basado en la red ResNet50, con caso de overfitting. Abajo: Gráfica que muestra los valores de exactitud obtenidos durante el entrenamiento correspondiente al extracto anterior.*

Se ha empleado además el método evaluate() sobre el modelo, especificando el conjunto de datos con el que realizar el test. Este método devuelve los valores de la función de pérdida y la métrica de evaluación *accuracy* obtenidas tras la realización de la predicción sobre todos los datos del conjunto de test especificado.

```
In [29]: model_RN50.evaluate(test_2_gen, batch_size=32, verbose=2)
         5/5 - 5s - loss: 0.4064 - accuracy: 0.9077
Out[29]: [0.4063756763935089, 0.9076923131942749]

In [30]: model_RN50.evaluate(test_int_gen, batch_size=32, verbose=2)
         15/15 - 2s - loss: 0.4245 - accuracy: 0.9060
Out[30]: [0.42452678084373474, 0.9059829115867615]
```

Fig. 16 *Ejemplo del uso del método evaluate, para un mismo modelo, empleando en cada caso un conjunto de test.*

La elección del mejor modelo se va a realizar, empleando como único criterio el mayor valor de exactitud alcanzado, como resultado de realizar la media entre los valores de exactitud obtenidos para los conjuntos de test “Test 2” e “Internet” antes expuestos.

5.2 Desarrollo de la herramienta

El primer paso, que ha permitido el desarrollo de la herramienta ha sido encapsular el modelo entrenado, en un fichero “.json” junto con el fichero de pesos correspondiente en formato h5. De esta manera, se portabiliza el clasificador. Para su uso, basta con crear de nuevo el modelo, a partir de la designación de la ubicación del fichero que lo contiene, cargar los pesos y compilarlo.

```
json_file = open('C:\\Users\\abalz\\Escritorio\\Asier\\Master IA\\Asignaturas\\Trabajo
model_json_xc = json_file.read()
json_file.close()
model_xc = model_from_json(model_json_xc)
```

```
model_xc.load_weights("C:\\Users\\abalz\\Escritorio\\Asier\\Master IA\\Asignaturas\\Tr
```

```
opt = SGD(lr=1e-4, momentum=0.9)
model_xc.compile(loss="categorical_crossentropy", optimizer=opt,metrics=["accuracy"])
```

Fig. 17 *Extracto de Código donde se observa la extracción del modelo clasificador encapsulado en el fichero json, la carga de los pesos generados durante el entrenamiento seleccionado y compilación del modelo dejándolo preparado para predecir.*

El siguiente paso para completar el objetivo de la herramienta desarrollada, ha sido confeccionar una función, denominada diag_map_gen_2, a la que se le debe proporcionar la ruta de la imagen o mapa que se desea analizar, y un parámetro que designa el tamaño en píxeles la casilla o *frame* que va a ser individualmente clasificado por la función.

Internamente, la función, realiza una copia de la imagen de entrada, sobre la que más adelante marcará cada zona clasificada según su resultado obtenido. La imagen original se va recorriendo, en pasos del número de pixeles pasados como parámetro. Cada zona recorrida, se localiza mediante cuatro puntos, que sirven para confeccionar cada imagen que se va a pasar al analizador, a través del uso del método crop() perteneciente a la librería PIL. Este recorte, es convertido a un array y redimensionado, para hacerlo compatible con el modelo.

Una vez realizada la predicción para el recorte, empleando el método predict() sobre el modelo, se colorea, en caso de ser positivo el frame correspondiente en el mapa final, concretamente aumentndo la intensidad del canal rojo, para ser visualmente identificable.

Una vez realizadas todas las clasificaciones, la función devuelve el numero de clasificaciones realizadas, junto con el número de positivas y negativas, a modo informativo, así como la imagen, o mapa visual final coloreado.

El extracto de código de la figura 18 muestra la función desarrollada, donde se observan los puntos comentados.

```
def diag_map_gen_2(img_file_path,pxframe):

    img_map = Image.open(img_file_path)
    image_map_copy = img_map.copy()
    image_map_final_array = tf.keras.preprocessing.image.img_to_array(image_map_copy, data_format=None, dtype=None)

    w, h = img_map.size
    number = 0
    positivos = 0
    negativos = 0

    for xref in range(0, w, pxframe):
        for yref in range(0, h, pxframe):
            points = (xref, yref,
                      xref + pxframe if xref + pxframe <  w else w - 1,
                      yref + pxframe if yref + pxframe < h else h - 1)

            # recorte de imagen, conversion a array y redimensionado, para predecir con el modelo implementado
            img_to_clas = img_map.crop(points)
            img_to_clas_form = tf.keras.preprocessing.image.img_to_array(img_to_clas, data_format=None, dtype=None)
            img_to_clas_form_size = cv2.resize(img_to_clas_form, (224,224))
            pred = model_xc.predict(np.expand_dims(img_to_clas_form_size, axis=0))[0]



            # coloreado del frame si esta clasificado como positivo
            if pred<0.5:
                negativos+=1

            else:
                positivos+=1
                for pixw in range (points[0],points[2],1):
                    for pixh in range (points[1],points[3],1):
                        image_map_final_array[pixh, pixw, 0] = image_map_final_array[pixh, pixw, 0] + 50.0

            number+=1
            #columna

    print(number,'classified frames')
    print(positivos,'positives / ',negativos,'negatives')

    img_final_map = tf.keras.preprocessing.image.array_to_img(image_map_final_array)

    return img_final_map
```

Fig. 18 *Imagen que muestra el código contenido en la función diag_map_gen_2.*

6. Evaluación

En este capítulo se presentan los resultados obtenidos a lo largo de las distintas fases de desarrollo del proyecto. En primer lugar, se muestran los valores obtenidos para el conjunto de modelos y entrenamientos realizados, y en segundo lugar, se muestran los resultados obtenidos de la aplicación directa de la herramienta implementada.

6.1 Evaluación de los modelos clasificadores

En la siguiente tabla, se recogen los valores de la función de pérdida y la métrica de evaluación 'accuracy', para los conjuntos de validación y test realizados ("Test 2" e "Internet"), para cada una de las variantes de los tres modelos implementados:

Tabla 2 *Valores de función de pérdida y exactitud obtenidos para los distintos modelos y parámetros empleados, para los conjuntos de validación y test.*

Modelo	Datos entrenamiento	Conjunto de datos	*loss*	*accuracy*
XCeption	opt=Adam(lr=0.01)	Valid.	0.164	0.943
		Test 2	0.775	0.808
		Internet	0.401	0.863
	opt=Adam(lr=0.001)	Valid.	0.138	0.960
		Test 2	0.509	0.854
		Internet	0.418	0.853
VGG16	opt=SGD(lr=0.01)	Valid.	0.567	0.910
		Test 2	0.769	0.839
		Internet	0.537	0.817
	opt=SGD(lr=0.001)	Valid.	0.409	0.909
		Test 2	0.630	0.846
		Internet	0.515	0.801
ResNet50	opt=Adam(lr=0.01)	Valid.	0.146	0.986
		Test 2	0.879	0.938
		Internet	1.233	0.867
	opt=Adam(lr=0.001)	Valid.	0.124	0.972
		Test 2	0.406	0.907
		Internet	0.424	0.906

En general, se observa que los valores obtenidos para el conjunto de validación son superiores a los obtenidos tras evaluar los conjuntos de test. Esto se debe, especialmente en el conjunto “Internet”, a que está compuesto por una serie de datos con mayor varianza, en cuanto a la composición y formato original de cada imagen.

En el caso de la red basada en la arquitectura XCeption, se han observado mejores resultados al establecer el *learning rate* del optimizador Adam, en 10^{-3}. Se obtiene una puntuación más equilibrada en exactitud y mas baja en la función de pérdida.

Para la red VGG16, se observa una tendencia similar, en magnitud empleada para el *learning rate*, pero el optimizador que mejor resultado ha dado ha sido SGD.

Finalmente, la red con arquitectura ResNet50 ha obtenido mejores resultados con el optimizador Adam, y el valor del *learning rate* 10^{-3}. Los valores de la exactitud no distan del resto de modelos, pero son equilibrados para ambos conjuntos de test y se obtiene además un valor más bajo que en el resto de modelos para la función de pérdida.

Como criterio de elección del modelo, tras realizar la media de los valores de ratio de éxito obtenidos para los conjuntos de test aplicados, se ha tomado el que alcanza mayor valor que es el modelo basado en la arquitectura ResNet50.

Tabla 3 *Valores medios de la función de pérdida y 'accuracy' obtenidos a partir de los valores de los conjuntos de test, para los modelos entrenados.*

Modelo	Datos entrenamiento	loss (media)	accuracy (media)
XCeption	opt=Adam(lr=0.01)	0.588	0.836
	opt=Adam(lr=0.001)	0.464	0.854
VGG16	opt=SGD(lr=0.01)	0.653	0.828
	opt=SGD(lr=0.001)	0.573	0.824
ResNet50	opt=Adam(lr=0.01)	1.056	0.903
	opt=Adam(lr=0.001)	0.415	**0.907**

6.2 Aplicación de la herramienta desarrollada

Tras confeccionar el código necesario para llevar a cabo los objetivos de la herramienta, las imágenes obtenidas a modo de mapa visual, para distintos entornos son mostradas para su interpretación.

6.2.1 Mapas de Salida

En esta primera pareja de imágenes de la figura 19, se observa a la derecha la imagen original y una vez aplicada la herramienta, a la izquierda, el resultado devuelto por la misma.

Fig. 19 *Mapas de entrada (dcha.) y de salida (izq.) tras emplear la herramienta desarrollada.*

Resaltados en rojo, se observan los *frames* clasificados como positivos, es decir que contienen desconchado o delaminado, donde de manera masiva, se observa que corresponden con las zonas más afectadas por desconchados y descascarillados. El resto de *frames*, que se observan más azulados o verdosos, han sido clasificados como negativos. En este caso, la imagen de arriba corresponde a una única imagen, tomada por el dron.

Al ejecutar la función que permite obtener el mapa de salida, se obtiene además la información que revela el número total de clasificaciones realizadas y la distribución de casos positivos y negativos, tal como se ve en la figura 20.

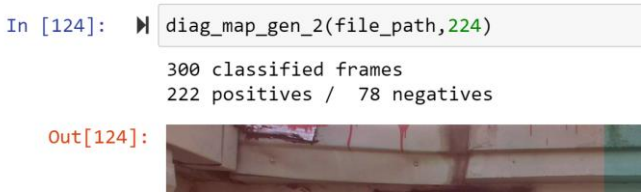

Fig. 20 *Muestra de salida de la información relativa a las clasificaciones realizadas por la función diag_map_gen_2, correspondiente a la primera imagen presentada, realizada por el dron.*

Un segundo ejemplo, lo compone el par de la figura 21, esta vez cada imagen está compuesta, montada manualmente a partir de 4 imágenes tomadas en puntos equiespaciados sobre la superficie retratada. De nuevo se observa, como las zonas afectadas están mayoritariamente cubiertas por *frames* coloreados, mientras que las zonas consideradas libres, mantienen el tono verde/azulado.

Se observa como especialmente en la zona de la ventana hay varios falsos positivos, zonas que el modelo ha considerado como contenedoras de delaminado, sin serlo realmente. Enfatizar, que la tarea de clasificación ha sido realizada a partir de la confección previa del mapa manual constituido por las cuatro imágenes proporcionadas por el dron y no en cada fotografía independiente.

Fig. 21 *Ejemplo de uso de la herramienta para un mapa confeccionado a partir de cuatro imágenes tomadas por dron. Mapa de entrada (dcha.) y mapa de salida (izda.)*

Por último, se presenta este tercer par de mapas, compuestos manualmente a partir de una secuencia de 12 fotografías de una fachada tomadas por el dron. De nuevo, se observa a

grandes rasgos cómo las zonas donde se manifiesta el fenómeno se encuentran compuestas fundamentalmente por casillas clasificadas correctamente como positivas.

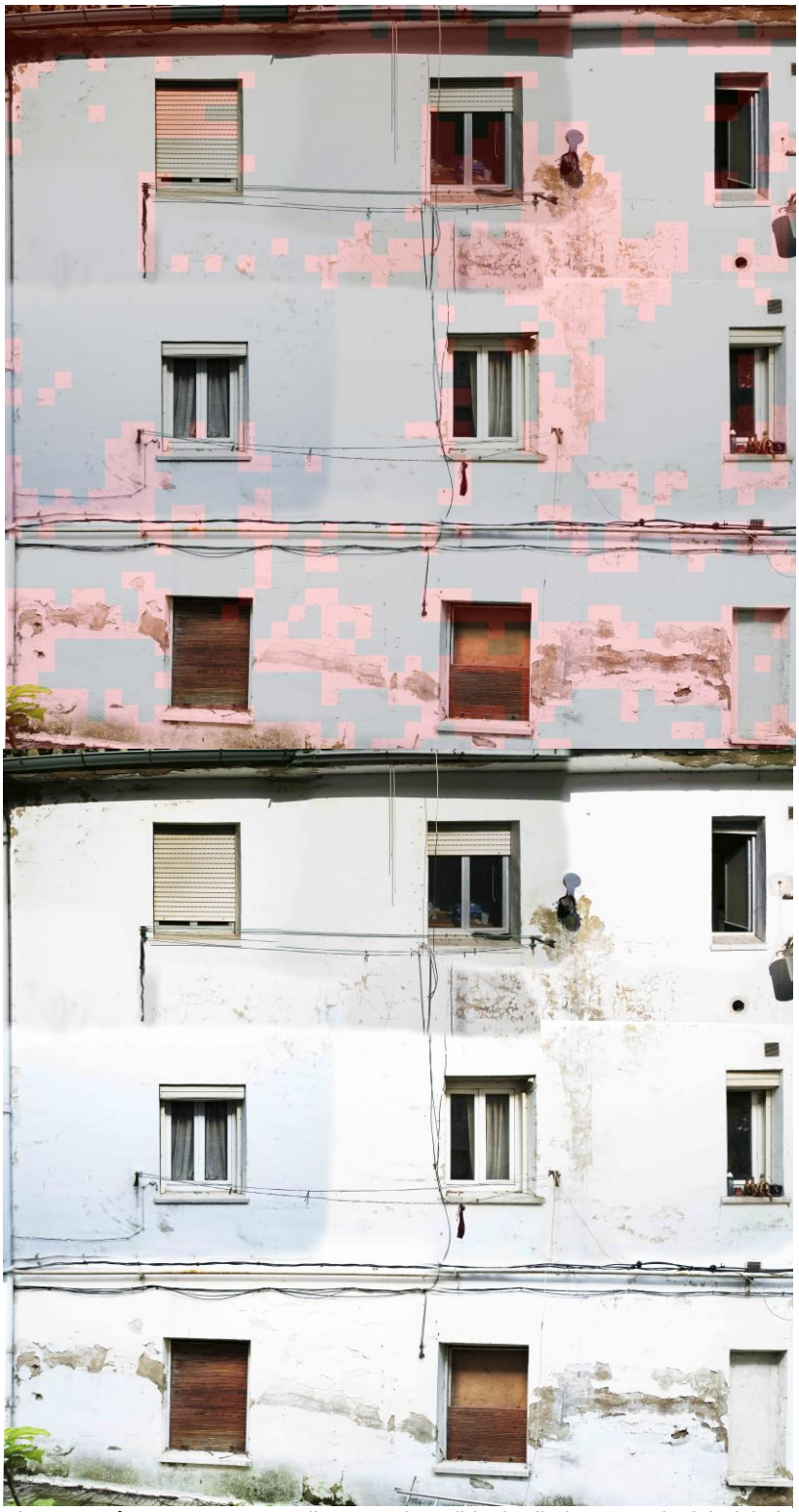

Fig. 20 *Imágenes correspondientes a la salida (arriba) y entrada (abajo) de la herramienta, para fachada de un edificio compueta por 12 imágenes tomadas con dron.*

De nuevo, y en este caso de manera mas acusada, se observan zonas con falsos positivos, debidos a elementos, no contemplados antes por el sistema como los mostrados en la figura 23.

Fig. 21 *Elementos que han dado lugar a falsos positivos, de izda. a dcha.: Trapo colgado, cables y persiana.*

7. Conclusiones y trabajo futuro

Una vez presentada y expuesta toda la labor realizada, esta sección concentra las conclusiones a las que se ha llegado tras la realización el trabajo experimental, así como la exploración o apunte de las líneas de trabajo futuro, que sitúan y conectan el aporte realizado con diversas mejoras y posibles usos.

7.1. Conclusiones

-Se ha confeccionado un *data set* (Delaminations2021), con 6452 imágenes que muestran el fenómeno del delaminado, desconchado o descascarillado sobre superficies de edificios.

-Se han entrenado tres modelos diferentes de redes neuronales convolucionales profundas con el *data set* creado, siendo capaces de identificar el desperfecto buscado con exactitudes entre el 82%.

-Los resultados obtenidos por las redes entrenadas empleando la técnica del *tranfer learning*, y los obtenidos por la red entrenada por completo son equiparables, no se observa un dominio en cuanto a resultados de una técnica sobre otra, para el conjunto de datos de entrenamiento empleado.

-A partir de los modelos entrenados, se ha confeccionado una herramienta, que permite localizar las zonas afectadas por el fenómeno buscado a partir de una imagen proporcionada.

-La herramienta creada, además de proporcionar un mapa visual para el humano, constituye una primera etapa para la intervención automatizada sobre fachadas, proporcionando el diagnóstico necesario, previo a una intervención pictórica.

-Los mapas de diagnóstico obtenidos, donde el sistema es capaz de identificar mayoritariamente las zonas afectadas de una fachada a partir de las imágenes proporcionadas por un dron, ponen de manifiesto que se ha alcanzado el objetivo global marcado para este trabajo.

7.2. Líneas de trabajo futuro

Se identifican varios puntos que pueden ser abordados como mejora directa de la herramienta desarrollada y métodos empleados:

-Búsqueda más profunda de modelos a implementar, así como tuneo de hiperparámetros más extenso, para la consecución de un modelo más exacto.

-Empleo de métodos de cosido de imágenes, propios del ámbito de percepción computacional, para elaborar mapas a partir del conjunto de imágenes tomadas por un dron de manera automatizada y no manual.

-Hacer uso de las coordenadas de localización proporcionadas en cada imagen del dron, para elaborar un **mapa de localización de precisión de las zonas afectadas**, para las futuras labores de intervención con drones.

Fig. 22 *Muestra, en las propiedades internas de una de las imágenes proporcionadas por el dron, de las coordenadas gps y altura en el instante de realizar la toma.*

-Empleo de técnicas propias de la Inteligencia Artificial, y concretamente del área de aprendizaje profundo, para diseñar sistemas de control, que guíen al dron automáticamente durante la toma de imágenes para la aplicación de la herramienta en una superficie real, extensa.

-A partir de la proporción de una magnitud dimensional sobre las fotografías tomadas por el dron, podría calcularse la dimensión real total de las zonas afectadas, proporcionando esto una información muy valiosa para el cálculo de los costes de la futura intervención necesaria.

8. Bibliografía

Adhikari, R. S., Moselhi, O., & Bagchi, A. (2014). Image-based retrieval of concrete crack properties for bridge inspection. *Automation in Construction*, *39*, 180–194. https://doi.org/10.1016/j.autcon.2013.06.011

Azimi, M., Eslamlou, A. D., & Pekcan, G. (2020). Data-Driven Structural Health Monitoring and Damage Detection through Deep Learning: State-of-the-Art Review. *Sensors 2020, Vol. 20, Page 2778*, *20*(10), 2778. https://doi.org/10.3390/S20102778

Bang, H., Min, J., & Jeon, H. (2021). Deep learning-based concrete surface damage monitoring method using structured lights and depth camera. *Sensors*, *21*(8), 2759. https://doi.org/10.3390/s21082759

Bhowmick, S., Nagarajaiah, S., & Veeraraghavan, A. (2020). Vision and deep learning-based algorithms to detect and quantify cracks on concrete surfaces from UAV videos. *Sensors (Switzerland)*, *20*(21), 1–19. https://doi.org/10.3390/s20216299

Carrio, A., Sampedro, C., Rodriguez-Ramos, A., & Campoy, P. (2017). A review of deep learning methods and applications for unmanned aerial vehicles. In *Journal of Sensors* (Vol. 2017). Hindawi Limited. https://doi.org/10.1155/2017/3296874

Cha, Y. J., & Choi, W. (2017). Vision-based concrete crack detection using a convolutional neural network. *Conference Proceedings of the Society for Experimental Mechanics Series*, *2 Part F2*, 71–73. https://doi.org/10.1007/978-3-319-54777-0_9

Cha, Y. J., Choi, W., & Büyüköztürk, O. (2017). Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks. *Computer-Aided Civil and Infrastructure Engineering*, *32*(5), 361–378. https://doi.org/10.1111/mice.12263

Cha, Y. J., Choi, W., Suh, G., Mahmoudkhani, S., & Büyüköztürk, O. (2018). Autonomous Structural Visual Inspection Using Region-Based Deep Learning for Detecting Multiple Damage Types. *Computer-Aided Civil and Infrastructure Engineering*, *33*(9), 731–747. https://doi.org/10.1111/mice.12334

Chollet, F. (2016). Xception: Deep Learning with Depthwise Separable Convolutions. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, *2017-January*, 1800–1807. https://arxiv.org/abs/1610.02357v3

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 580–587. https://doi.org/10.1109/CVPR.2014.81

Gopalakrishnan, K., Gholami, H., Vidyadharan, A., Choudhary, A., & Agrawal, A. (2018). CRACK DAMAGE DETECTION IN UNMANNED AERIAL VEHICLE IMAGES OF CIVIL INFRASTRUCTURE USING PRE-TRAINED DEEP LEARNING MODEL. *INTERNATIONAL JOURNAL FOR TRAFFIC AND TRANSPORT ENGINEERING*, *8*(1), 1–14. https://doi.org/10.7708/ijtte.2018.8(1).01

Kim, J. J., Kim, A. R., & Lee, S. W. (2020). Artificial neural network-based automated crack detection and analysis for the inspection of concrete structures. *Applied Sciences (Switzerland)*, *10*(22), 1–13. https://doi.org/10.3390/app10228105

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. https://arxiv.org/abs/1412.6980v9

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. https://doi.org/10.1145/3065386

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2323. https://doi.org/10.1109/5.726791

Li, S., & Zhao, X. (2019). Image-Based Concrete Crack Detection Using Convolutional Neural Network and Exhaustive Search Technique. *Advances in Civil Engineering*, *2019*. https://doi.org/10.1155/2019/6520620

Liu, Y., Yao, J., Lu, X., Xie, R., & Li, L. (2019). DeepCrack: A deep hierarchical feature learning architecture for crack segmentation. *Neurocomputing*, *338*, 139–153. https://doi.org/10.1016/j.neucom.2019.01.036

Maeda, H., Sekimoto, Y., Seto, T., Kashiyama, T., & Omata, H. (2018). Road Damage Detection Using Deep Neural Networks with Images Captured Through a Smartphone. *Computer-Aided Civil and Infrastructure Engineering*, *33*(12), 1127–1141. https://doi.org/10.1111/mice.12387

Maniat, M., Camp, C. V., & Kashani, A. R. (2021). Deep learning-based visual crack detection using Google Street View images. *Neural Computing and Applications*, 1–18. https://doi.org/10.1007/s00521-021-06098-0

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics 1943 5:4*, *5*(4), 115–133. https://doi.org/10.1007/BF02478259

Mohan, A., & Poobal, S. (2018). Crack detection using image processing: A critical review and analysis. *Alexandria Engineering Journal*, *57*(2), 787–798. https://doi.org/10.1016/j.aej.2017.01.020

Pereira, F. C., & Pereira, C. E. (2015). Embedded image processing systems for automatic recognition of cracks using UAVs. *IFAC-PapersOnLine*, *28*(10), 16–21. https://doi.org/10.1016/j.ifacol.2015.08.101

Phung, M. D., Dinh, T. H., Hoang, V. T., & Ha, Q. P. (2017). Automatic crack detection in built infrastructure using unmanned aerial vehicles. *ISARC 2017 - Proceedings of the 34th International Symposium on Automation and Robotics in Construction*, 823–829. https://doi.org/10.22260/isarc2017/0115

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, *115*(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y

Shin, H. K., Ahn, Y. H., Lee, S. H., & Kim, H. Y. (2020). Automatic concrete damage recognition using multi-level attention convolutional neural network. *Materials*, *13*(23), 1–13. https://doi.org/10.3390/ma13235549

Simonyan, K., & Zisserman, A. (2015, September 4). Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. http://www.robots.ox.ac.uk/

Vempati, A. S., Kamel, M., Stilinovic, N., Zhang, Q., Reusser, D., Sa, I., Nieto, J., Siegwart, R., & Beardsley, P. (2018). PaintCopter: An autonomous UAV for spray painting on three-dimensional surfaces. *IEEE Robotics and Automation Letters*, *3*(4), 2862–2869. https://doi.org/10.1109/LRA.2018.2846278

Zheng, M., Lei, Z., & Zhang, K. (2020). Intelligent detection of building cracks based on deep learning. *Image and Vision Computing*, *103*, 103987. https://doi.org/10.1016/j.imavis.2020.103987

Zhu, Q., Dinh, T. H., Phung, M. D., & Ha, Q. P. (2021). Hierarchical Convolutional Neural Network With Feature Preservation and Autotuned Thresholding for Crack Detection. *IEEE Access*, *9*, 60201–60214. https://doi.org/10.1109/ACCESS.2021.3073921

Zhu, Z., German, S., & Brilakis, I. (2011). Visual retrieval of concrete crack properties for automated post-earthquake structural safety evaluation. *Automation in Construction*, *20*(7), 874–883. https://doi.org/10.1016/J.AUTCON.2011.03.004

Anexos

Anexo: Artículo de investigación

Herramienta de diagnóstico previa a la realización de pintura mural con drones

Balza de Vallejo Julián, Asier

Universidad Internacional de la Rioja, Logroño (España)

14/09/2021

Resumen

Una de las labores más importantes previas a la realización de cualquier intervención pictórica sobre una pared, es comprobar el estado de su superficie. El principal fenómeno que describe esta situación es el desconchado, cuya identificación y cuantificación son labores susceptibles de ser automatizadas. A través del dron como vehículo extractor de información, entra en juego el uso de la Inteligencia Artificial, dentro del área de visión por computación, haciendo uso de redes neuronales convolucionales con el objetivo de diseñar un sistema capaz de identificar el fenómeno descrito. A partir un data set específico propio constituido por 6452 imágenes, se han implementado y entrenado tres modelos basados en las arquitecturas XCeption, VGG16 y ResNet50, consiguiendo valores de ratio de éxito entre 82% y 90%. Se ha confeccionado una herramienta que, tras proporcionarles imágenes de edificios tomadas por un dron, devuelve un mapa visual, diagnóstico superficial sobre la imagen original donde se identifican las zonas que el modelo ha clasificado como positivas.

Palabras Clave

desconchado, diagnóstico de superficies, drones, redes neuronales convolucionales, visión por computación

I. Introducción

Desde que el ser humano comenzó a desarrollar las primeras herramientas, y después con éstas, las primeras máquinas, comenzó también, de alguna manera la búsqueda de la automatización de las posibles tareas que éste podía realizar. La evolución del conocimiento y la técnica, han llevado ese paradigma de la automatización a niveles que con anterioridad apenas podrían haber sido imaginados. Y es entonces cuando esa evolución en la computación y el cálculo automatizado, tras ser etiquetado como inteligente, continúa filtrándose por cada poro del tejido de todos los campos de conocimiento de la actualidad.

Uno de esos campos, donde confluyen tanto la tecnología como medio físico, junto con el uso de la inteligencia computacional, son los vehículos aéreos no tripulados, drones o UAVs (del inglés, *Unmanned Aerial Vehicles*). Son el vehículo ideal para transportar de manera ágil, y a cualquier zona de difícil acceso, un conjunto de sensores que recogen y transmiten datos, que más tarde un sistema inteligente puede interpretar y calcular respuestas y acciones que suponen un triunfo de la automatización (Carrio et al., 2017) nuevamente, en la tarea aplicada.

Una de las labores que el ser humano lleva realizando desde épocas muy tempranas y que lo ha acompañado hasta la actualidad es la realización de pinturas murales, desde las primeras intervenciones en cuevas hasta los más extensos murales que decoran hoy día grandes edificaciones. Y es también una tarea en la que la automatización hace algún tiempo que hizo su incursión.

En 2019, en Turín, bajo el nombre de UFO-Urban Flying Opera, se llevó a cabo un proyecto de automatización de pintura mural, con una extensión de 150 m^2, empleando aerosoles accionados por cuatro drones, controlados por un sistema desarrollado por Tsuru Robotics. Esta demostración, con objetivo fundamentalmente artístico pone en evidencia cómo este sector está siendo también alcanzado por la tecnología mencionada (Vempati et al., 2018).

La realización de pinturas murales a gran escala, tanto con carácter artístico, como de rehabilitación o preventivo son ya labores alcanzadas por el proceso de automatización pudiendo evitar la construcción de andamiajes, ahorrando tiempos de ejecución (Pereira & Pereira, 2015) y costes, y evitando accidentes, en definitiva optimizando la tarea.

Es en este contexto de la realización autónoma de labores de pintura mural, mantenimiento o prevención, donde se torna importante e indispensable una labor previa: la inspección de la superficie que va a ser intervenida. La exposición a los fenómenos propios de la intemperie de los materiales que conforman la fachada de un edificio, pueden deteriorarla evidenciándose esto como levantamientos, grietas, descascarillados o desconchados. Identificar y localizar estos puntos es fundamental para un correcto desarrollo de la pintura mural, evitando a corto plazo un daño inminente y muy costoso *a posteriori* sobre el trabajo realizado.

En la actualidad, el fenómeno de identificación y clasificación de las grietas como indicadores del estado de salud de las distintas obras de ingeniería civil, ha sido abordado mediante distintos enfoques en el ámbito de la visión por computación. Concretamente han dado muy buenos resultados modelos del ámbito del aprendizaje profundo, en especial el uso de redes neuronales convolucionales, y es dicha línea la que ha inspirado este trabajo.

Así, tras confeccionar inicialmente un *data set* propio

constituido por 6452 imágenes, se han implementado y con ello entrenado tres modelos de redes neuronales convolucionales a partir de las arquitecturas XCeption, VGG16 y ResNet50, consiguiendo valores de ratio de éxito entre 82% y 90%.

A partir del modelo que ha ofrecido mayor valor de exactitud, se ha confeccionado una herramienta que, tras proporcionarles imágenes de edificios tomadas por un dron, devuelve un mapa visual sobre la imagen original donde se identifican las zonas que el modelo ha clasificado como positivas, contenedoras de desconchados o descascarillados.

La aplicación confeccionada, constituye así una herramienta de diagnóstico visual y es el punto de partida para la evaluación de la magnitud del fenómeno buscado, así como de la elaboración de un mapa preciso de intervención automatizada.

II. Estado del Arte

Se centra en este punto la mirada sobre las diferentes técnicas, en el ámbito de la inteligencia artificial, que han sido y están siendo aplicadas a la detección de fenómenos, en este caso entendidos como defectos en superficies. El concepto es muy amplio y abarca campos tan diversos como la detección de faltas en vigas de acero, superficies y estructuras metálicas, inspección de paneles solares y de placas electrónicas (PCB), o defectos de fábrica en textiles por nombrar sólo algunos de ellos.

Sin embargo, el foco que abarca la atención de este artículo es el de las superficies estructurales arquitectónicas, obras de ingeniería civil, constituidas en definitiva por materiales de construcción como el hormigón, cemento, ladrillos, yeso, y diversos revestimientos como fijadores, pinturas y barnices, en última instancia.

Uno de los primeros ejemplos de uso de redes convolucionales para tal fin es (Cha et al., 2017) demostrando efectividad y mejora frente a las anteriormente empleadas técnicas de procesamiento de imagen, gracias a la capacidad que otorga el cálculo automatizado de indicadores de presencia de daño durante el entrenamiento de las redes (Cha et al., 2018).

Otra de las primeras contribuciones, aunque no tan específica de la detección de defectos en superficies, si no de manera más genérica en el ámbito de la detección de objetos en imágenes, se encuentra en (Girshick et al., 2014), donde apoyándose en dicho paradigma a través del empleo de la división de las imágenes de entrada en regiones, da lugar al uso de las denominadas R-CNN (Region based CNNs). Éstas, tras dividir el espacio en las mencionadas regiones, hacen uso de una CNN para la extracción de un vector de características de longitud fija y finalmente se emplea un clasificador, como máquinas de vector de soporte, para clasificar cada región y finalmente mostrar, la reconstrucción de los objetos (categorías) encontrados en la imagen. Sin embargo, un problema que presentan es el alto coste de tiempo empleado en la detección sobre cada entrada. Como mejora, se plantea en (Cha et al., 2018) en un ámbito ya específico de la detección de defectos en superficies estructurales, el uso de Faster R-CNNs, que permiten una clasificación global casi a tiempo real, con una precisión del 87,8%.

En línea con la automatización de la monitorización de daño estructural en edificios, debido al envejecimiento, está el ejemplo de (Bang et al., 2021) que empleando luz laser y cámaras de profundidad, consigue detectar grietas, delaminaciones y exposición de acero, empleando un también Faster R-CNNs. También en este contexto, es interesante la utilización que hace (Zheng et al., 2020) de las redes FCN (Fully Convolutional Networks), RCNN y RFCN (Richer Fully Convolutional Networks), componiendo un algoritmo inteligente que ofrece buenos resultados en aspectos relativos a la detección de faltas en la superficie de los edificios, puentes, presas, etc.,

Relativo al mantenimiento de las estructuras de hormigón, está el ejemplo de aplicación de (Li & Zhao, 2019) que entrenando la arquitectura AlexNet, con 60.000 imágenes de grietas, y tras integrarla en un dispositivo smartphone, consigue en la práctica detectar dicho fenómeno con una exactitud del 95%.

No obstante, no siempre es fácil conseguir *data sets* tan amplios, conjuntos de ejemplos de fenómenos concretos etiquetados, que permitan entrenar modelos desde "0", por ello un tipo de solución común consiste en emplear modelos de aprendizaje profundo, pre-entrenados (Gopalakrishnan et al., 2018). Estos modelos se han entrenado previamente con *data sets* mucho mayores, de manera que los parámetros obtenidos engloban reconocimiento de características específicas procedentes de imágenes, que resultan útiles a la hora de reutilizarlos con nuevos *data sets*, más sencillos y específicos, permitiendo entrenar únicamente la parte final del modelo global, es decir la parte relativa a la clasificación de las entradas. Esta técnica recibe el nombre de *transfer learning*. Como ejemplo del uso de ésta técnica, a partir de imágenes tomadas por drones para la detección de defectos en superficies de infraestructuras, con una obtención del 90% de exactitud, se presenta (Gopalakrishnan et al., 2018).

Otro ejemplo de uso, concretamente en el ámbito de la evaluación del pavimento, es decir del estado de las carreteras, se encuentra en (Maniat et al., 2021) donde empleando imágenes de Google Street View, demuestran su aplicación efectiva frente a sofisticadas soluciones de ámbito comercial.

Otra visión interesante del uso de CNNs, volviendo al ámbito de monitorización del estado de edificios se encuentra en (Liu et al., 2019), donde se propone una arquitectura concreta, denominada DeepCrack, definida como una arquitectura de aprendizaje profundo jerárquico cuyo objetivo es no solo la detección si no la segmentación de las grietas detectadas. En esta misma línea, (Bhowmick et al., 2020) presentan el uso de la arquitectura U-Net, demostrando su efectividad en la consecución de la tarea en cuestión, a partir de vídeos tomados por drones. La obtención de la segmentación de las grietas detectadas se presenta además como un paso intermedio decisivo hacia la obtención de las propiedades geométricas (longitud, anchura, orientación) de las grietas. Un claro ejemplo de la utilidad de la obtención de estas propiedades se presenta en (Z. Zhu et al., 2011), donde se emplea para conocer la magnitud de las mismas originadas en infraestructuras localizadas en zonas sometidas a terremotos. Otro ejemplo de método interesante de obtención de las características geométricas basado en técnicas de procesamiento de imagen, además del empleo de redes convolucionales para la detección de grietas, se presenta en (Kim et al., 2020).

Como ejemplo de uso de red convolucional, jerárquica e híbrida, por el uso de un módulo de binarización para la obtención de un mapa de características de salida, se presenta (Q. Zhu et al., 2021), que es además una evolución de la mencionada red DeepCrack.

Además de la detección de grietas, entendiéndola como una clasificación binaria, el ejemplo de CMDNet muestra la posible clasificación de las imágenes procedentes del estado de infraestructuras en 5 clases diferentes: "intacta", "grieta", "exposición del acero", "levantamiento" o "delaminación" y "goteo" o "fuga". Este tipo de uso de red convolucional se caracteriza por emplear "ramas" de redes de atención, y su arquitectura está basada en la red VGG16, que se analiza más adelante. Consigue además una exactitud del 98,9% (Shin et al., 2020).

III. Objetivos y Metodología

Se proyecta completar un programa que, recogiendo un conjunto de fotografías de una fachada de edificio tomadas por un dron, pueda identificar correctamente las zonas donde haya deterioro superficial (delaminaciones y grietas), devolviendo un mapa para su localización y cuantificación, como herramienta de ayuda a las labores previas a tareas de mantenimiento o intervención pictórica sobre la superficie estudiada.

Como objetivos específicos para lograr la meta descrita, se plantean los siguientes puntos:

- Crear un data set específico de delaminaciones con casos positivos y negativos.
- Seleccionar una arquitectura de CNN propicia para entrenarla con el data set creado y poder identificar los casos positivos que son objeto de estudio.
- Implementar el programa que gestione la entrada de datos y la salida con la información deseada.
- Testar la herramienta diseñada, con fotografías extraídas por al menos un dron real, sobre uno o varios edificios seleccionados.

En primer lugar, y como punto de partida del proyecto se va a confeccionar un data set específico de delaminaciones, levantamientos y desconchados, ya que no ha sido posible encontrar ninguno en la red global, extrayendo manualmente, con cámara digital, fotografías de paredes evidenciadas con tales fenómenos.

Una vez obtenido este material, es el momento de comenzar a implementar una red neuronal convolucional, capaz de reconocer los casos positivos en imágenes susceptibles de contener zonas con delaminaciones. Los ejemplos descritos anteriormente para fines similares son los modelos en los que dicha red neuronal se va a basar. Se trata por tanto de acceder a ejemplos implementados para comprender cómo han sido sintetizados y su posible usabilidad para el proyecto. A partir de ahí es el momento de desarrollar el código propio necesario para implementar los modelos y habilitar sus respectivos entrenamientos.

Una vez elegido un primer conjunto de modelos, éstos se van a entrenar con el conjunto de datos construido, empleando modificaciones, transformaciones en orientación, contraste y lateralización, sobre el conjunto de datos inicial, aprovechando las ventajas del *data augmentation* (Shin et al., 2020) con el objetivo de reducir el posible *overfitting* debido a que dicho conjunto podría no resultar suficiente debido a su extensión (Cha et al., 2018). Se empleará la mayor parte del *data set* confeccionado para el entrenamiento, dejando una parte para verificación y unos pocos ejemplos como evaluación final, test.

El hecho de completar unos primeros entrenamientos, con un conjunto de parámetros o hiperparámetros inicial, será un indicador de que los modelos, computacionalmente se han implementado correctamente y los resultados obtenidos apuntarán el nivel de efectividad a la hora de clasificar a partir del conjunto de datos proporcionado.

Tras interpretar los primeros resultados obtenidos, se trata a través del refinamiento del código inicial planteado, y la exploración de distintos valores de hiperparámetros, de encontrar el resultado que ofrezca mayor exactitud y menor función de pérdida. El modelo que mejores resultados ofrezca del conjunto inicial explorado constituirá el motor de clasificación de la herramienta que se va a desarrollar.

La funcionalidad y utilidad del modelo construido se va a probar con imágenes varias de una fachada tomadas con un dron real. El resultado debe ofrecer información al humano acerca de la localización sobre la superficie de los puntos conflictivos.

Finalmente, se realizará una valoración basada en una inspección visual sobre la efectividad de los resultados ofrecidos por la herramienta. La intención final es que el resultado sea extrapolable a áreas muchísimo mayores y con zonas de difícil acceso para su evaluación visual.

IV. Contribución

Creación del *data set*

El primer paso, ha consistido en la creación de un conjunto de imágenes de entrenamiento específico del tipo de deterioro descrito. Para ello, se han tomado 661 fotografías con una cámara Nikon Coolpix S3000, con 5 megapíxeles de resolución (2592x1944), de las que un 60% muestran un caso positivo del defecto frente al 40% que representan el negativo. A este conjunto inicial, se le ha añadido otro conjunto de imágenes tomadas con diversos terminales (smartphones) componiendo un total de 162 nuevas imágenes con la relación entre casos positivos y negativos al 65% y 35% respectivamente. A partir de este muestrario inicial, y conociendo que la resolución o dimensiones de entrada típicos a un sistema clasificador candidato se encuentran entre (224x224) y (227x227) píxeles, se ha considerado cada imagen como un mapa mayor, donde aparecen distintos y variados aspectos del fenómeno a estudiar, el desconchado o delaminado en cuestión.

Para ello se han desarrollado dos funciones, que han permitido obtener un conjunto de imágenes a partir de la fragmentación de una mayor, así como una función capaz de renombrar todos los ficheros (imágenes) contenidas en una carpeta, para mayor orden y control del proceso. Una vez aplicadas estas funciones se ha revisado cada uno de los nuevos casos obtenidos, inspeccionando y reclasificando manualmente, y desechando en algún caso también si el contenido de la imagen no mostraba claramente un caso positivo o negativo. De esta manera, se ha conseguido sintetizar un **conjunto final de 6452 imágenes, compuesto por 3226 casos positivos y los mismos negativos, revisados individualmente**. Se ha decidido nombrar a este conjunto "Delaminations2021".

Fig. 1 *Muestras de imágenes de la clase positiva del data set confeccionado por el autpr,"Delaminations2021"; la fila de arriba está compuesta por imágenes completas (mapas) y las de la fila inferior son el resultado del fragmentado o corte.*

Además, se ha sintetizado un segundo *data set* de tamaño muy inferior, con el propósito de emplearlo como **conjunto de test**. Más adelante se hará referencia a él como "Test 2" y está compuesto, a través del método de fragmentación empleado antes, por 130 imágenes pertenecientes a ambos casos. Por otro

lado, se ha sintetizado otro conjunto de test como resultado de hacer una búsqueda general de términos de designación del defecto en cuestión en internet, a partir de imágenes sin copyright. Este otro conjunto se ha generado como el anterior y está integrado por 430 imágenes. En este caso, las imágenes al tener muy diversos orígenes y condiciones (iluminación, ángulo, etc.,) pretenden proporcionar una medida adicional a la validación y al test del clasificador, para evaluar la capacidad de extrapolar, o inferir conocimiento del sistema, ante el fenómeno en nuevas condiciones.

Implementación de los modelos y entrenamientos

A partir de la información consultada, mostrada en la sección dedicada al estado del arte, en lo que al problema central de esta investigación se refiere, se ha decidido implementar tres modelos diferentes: XCeption, VGG16 y ResNet50. Para poder llevarlo a cabo, el tándem de librerías empleadas para el desarrollo del código necesario ha sido TensorFlow y Keras. Tras indagar en ejemplos más simples e inspirándose en ejemplos propuestos en ambas documentaciones correspondientes,(https://keras.io/api/https://www.tensorflow.org/api_docs/python/tf) se ha creado un guión para notebook común como punto de partida donde se aborda el pre-procesamiento de los datos, *data augmentation*, implementación de la arquitectura específica del modelo, compilación del modelo y otras funciones relativas al entrenamiento, y evaluación.

Para el pre-procesamiento de los datos se ha empleado la clase ImageDataGenerator() para crear los objetos que contienen las diferentes acciones de *data augmentation* (rotación, brillo, recortado, volteo) para el posterior conjunto de entrenamiento y de reescalado (rescale=1.0/255) para ambos conjuntos de entrenamiento y validación. El motivo de realizar esta operación es el siguiente: las imágenes que constituyen los datos de entrada son *arrays* de tres dimensiones: disposición planar de píxeles (altura y anchura) y profundidad, que está compuesta de tres capas “RGB”, donde en cada una se codifican los valores entre 0 y 255 relativos al rojo (Red), verde (Green) y azul (Blue) respectivamente. Con la operación de reescalado, lo que se está haciendo es trasladar cada uno de esos valores del rango descrito, al rango [0,1] para posibilitar los cálculos a través de la red neuronal. Despuéslo que se ha realizado es la aplicación del método flow_from_directory() sobre los objetos creados anteriormente, de manera que se obtienen, bajo los nombres train_dts y validation_dts , lo que se denomina como “Iteradores de Directorio”, que son fundamentalmente tuplas de las imágenes de entrada con sus etiquetas, designadas a partir de la estructura subyacente a la ruta asignada a través de la variable “filepath”. Se establece además el tamaño del *batch*, o subconjunto de datos de entrenamiento, fijándolo en 32 imágenes. El número de clases se infiere de la distribución de carpetas en las que los datos son almacenados, en este caso dos: ‘Positivos’ y ‘Negativos’.

En la figura 2, se muestra una representación de 25 imágenes, y cómo han sido etiquetadas automáticamente, designadas como 1 para la clase positiva, donde se puede comprobar visualmente el desconchado o descascarillado, y 0 para la clase negativa, donde las muestras no representan defecto. Y de manera análoga a los datos de entrenamiento, se preparan los conjuntos de test para su posterior evaluación.

A continuación de los expuesto hasta ahora, vendría la implementación de la arquitectura propia de cada modelo, empleando el tándem de librerías mencionadas.

Fig. 2 *Muestra de 25 imágenes del conjunto de entrenamiento, etiquetadas automáticamente como positivas (1) y negativas (0).*

Una vez creado el modelo, se procede a compilar éste haciendo para ello uso del método compile() y que es donde se eligen los parámetros como el optimizador, la función de pérdida (loss), y la métrica de evaluación. A la hora de elegir el optimizador, para los modelos empleados en el desarrollo de la herramienta, se han empleado SGD (del inglés *Stochastic Gradient Descent* o descenso del gradiente estocástico) y Adam (procede de *adaptative moment estimation*) considerado como una evolución o mejora del algoritmo propio del descenso del gradiente y que en general se comporta bastante bien al ser implementado en tareas de aprendizaje profundo (Kingma & Ba, 2014). En cuanto a la función de pérdida, son diversas las posibilidades ofrecidas por parte de las librerías empleadas para la implementación de los modelos, no obstante, para el tipo de problema de clasificación, binaria, que se desea resolver la más indicada se denomina ‘binary_crossentropy’, o también conocida como “log loss”. Por último, como métrica de error o métrica de evaluación del modelo se ha decidido, en base a lo advertido en la literatura consultada, emplear la exactitud o ratio de éxito: ‘accuracy’. Es un indicador de lo “cerca” que está el conjunto de resultados obtenidos del resultado objetivo y cuya magnitud está comprendida entre 0 y 1.

Antes de comenzar con los entrenamientos, se ha tenido que determinar el uso de los mencionados *callbacks*, los objetos que permiten realizar acciones durante los entrenamientos, antes y después de cada *epoch*. Por un lado, se ha hecho uso de la clase ModelCheckpoint para poder guardar, a medida que avanza el entrenamiento, los valores de los pesos conseguidos hasta el momento en función de cierto criterio establecido para el valor de la métrica de evaluación. Por otro lado, se ha empleado la clase EarlyStopping para, en caso de no obtener mejora en las métricas designadas a lo largo de cierto número de *epochs*, para el entrenamiento, con el objetivo de no perder tiempo ni recursos de computación, ni llevar al modelo a *overfitting*. En este caso se ha contemplado establecer el límite de 20 *epochs* sin mejora, teniendo en cuenta que en general, se ha establecido un total de 50 *epochs* por entrenamiento.

Para comenzar el entrenamiento, se ha empleado el método fit(), sobre el modelo creado, especificando las fuentes de datos de entrenamiento y validación, *epochs* y *callbacks*. Una vez concluidos los entrenamientos, se ha empleado el método history[], sobre la variable que designa el entrenamiento, para junto con la librería matplotlib, representar la métrica de evaluación, *accuracy* y la función de pérdida del entrenamiento y validación, en cada *epoch* durante todo el entrenamiento, tal como se observa en la figura 3. Permite visualizar rápidamente la evolución de los mismos, comprobando si hay *overfitting*, o

estancamiento en la mejora de la métrica de evaluación. En el caso mostrado en dicha figura, tanto en los valores observables del extracto, como la representación gráfica, muestran un ejemplo de *overfitting*, donde el valor de exactitud de entrenamiento ha alcanzado el máximo (1), y el valor correspondiente a la validación se estanca en 0.9860, durante un entrenamiento de un modelo basado en la red ResNet50.

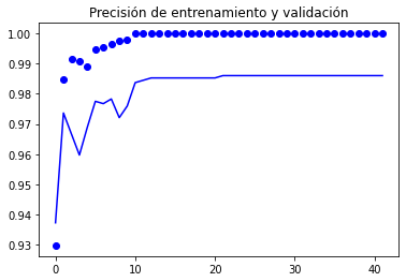

Fig. 3 *Gráfica que muestra los valores de exactitud obtenidos durante el entrenamiento correspondiente al extracto anterior.*

Se ha empleado además el método evaluate() sobre el modelo, especificando el conjunto de datos con el que realizar el test. Este método devuelve los valores de la función de pérdida y la métrica de evaluación *accuracy* obtenidas tras la realización de la predicción sobre todos los datos del conjunto de test especificado.

Confección de la herramienta de diagnóstico:

El primer paso, que ha permitido el desarrollo de la herramienta ha sido encapsular el modelo entrenado, en un fichero ".json" junto con el fichero de pesos correspondiente en formato h5. De esta manera, se portabiliza el clasificador. Para su uso, basta con crear de nuevo el modelo, a partir de la designación de la ubicación del fichero que lo contiene, cargar los pesos y compilarlo.

El siguiente paso para completar el objetivo de la herramienta desarrollada, ha sido confeccionar una función, denominada diag_map_gen_2, a la que se le debe proporcionar la ruta de la imagen o mapa que se desea analizar, y un parámetro que designa el tamaño en píxeles la casilla o *frame* que va a ser individualmente clasificado por la función.

Internamente, la función, realiza una copia de la imagen de entrada, sobre la que más adelante marcará cada zona clasificada según su resultado obtenido. La imagen original se va recorriendo, en pasos del número de pixeles pasados como parámetro. Cada zona recorrida, se localiza mediante cuatro puntos, que sirven para confeccionar cada imagen que se va a pasar al analizador, a través del uso del método crop() perteneciente a la librería PIL. Este recorte, es convertido a un array y redimensionado, para hacerlo compatible con el modelo.

Una vez realizada la predicción para el recorte, empleando el método predict() sobre el modelo, se colorea, en caso de ser positivo el *frame* correspondiente en el mapa final, concretamente aumentando la intensidad del canal rojo, para ser visualmente identificable. Una vez realizadas todas las clasificaciones, la función devuelve el número de clasificaciones realizadas, junto con el número de positivas y negativas, a modo informativo, así como la imagen, o mapa visual final coloreado.

Toma de imágenes con el dron

Ya que el propósito final de la herramienta es clasificar imágenes a partir de las fotografías tomadas por un dron, de una fachada real, se han tomado imágenes con un modelo DJI Air 2S, en vuelo manual, sobre dos fachadas de edificios, que visualmente revelan la presencia de descascarillados. Se ha realizado un barrido manual, a distancia constante al edificio tomando fotografías progresivamente. El material recogido, conforma la información necesaria en cuanto a imágenes para probar la herramienta, además de proporcionar en los metadatos de cada imagen, donde se encuentran las coordenadas GPS y altura, del momento en el que se toma cada fotografía.

Fig. 4 *Imágenes, de izda. a dcha.: Dron antes de tomar vuelo; Detalle de una de las fachadas escaneada; Imagen global de una de las fachadas escaneadas.*

V. Evaluación y Resultados

En primer lugar, se muestran los valores obtenidos para el conjunto de modelos y entrenamientos realizados, en segundo lugar, se muestran los resultados obtenidos de la aplicación directa de la herramienta implementada.

Evaluación de los modelos clasificadores

En la tabla I, se recogen los valores de la función de pérdida y la métrica de evaluación 'accuracy', para los conjuntos de validación y test realizados ("Test 2" e "Internet"), para cada una de las variantes de los tres modelos implementados.

En la tabla II, se presentan las medias de los valores de la función de pérdida y 'accuracy', éste último establecido como criterio para seleccionar el modelo que mejor resultado ha ofrecido.

Aplicación de la herramienta desarrollada

Tras aplicar la herramienta sobre las fotografías tomadas por el dron, las imágenes obtenidas a modo de mapa visual, para distintos entornos son mostradas en el apéndice I para su interpretación.

VI. Discusión

Modelos clasificadores

En general, se observa que los valores obtenidos para el conjunto de validación son superiores a los obtenidos tras evaluar

TABLA I

Modelo	*Datos entrenamiento*	*Conjunto de datos*	*loss*	*accuracy*
XCeption	opt=Adam(lr=0.01)	Valid.	0.164	0.943
		Test 2	0.775	0.808
		Internet	0.401	0.863
	opt=Adam(lr=0.001)	Valid.	0.138	0.960
		Test 2	0.509	0.854
		Internet	0.418	0.853
VGG16	opt=SGD(lr=0.01)	Valid.	0.567	0.910
		Test 2	0.769	0.839
		Internet	0.537	0.817
	opt=SGD(lr=0.001)	Valid.	0.409	0.909
		Test 2	0.630	0.846
		Internet	0.515	0.801
ResNet50	opt=Adam(lr=0.01)	Valid.	0.146	0.986
		Test 2	0.879	0.938
		Internet	1.233	0.867
	opt=Adam(lr=0.001)	Valid.	0.124	0.972
		Test 2	0.406	0.907
		Internet	0.424	0.906

TABLA II

Modelo	**Datos entrenamiento**	**loss (media)**	**accuracy (media)**
XCeption	opt=Adam(lr=0.01)	0.588	0.836
	opt=Adam(lr=0.001)	0.464	0.854
VGG16	opt=SGD(lr=0.01)	0.653	0.828
	opt=SGD(lr=0.001)	0.573	0.824
ResNet50	opt=Adam(lr=0.01)	1.056	0.903
	opt=Adam(lr=0.001)	0.415	0.907

los conjuntos de test. Esto se debe, especialmente en el conjunto "Internet", a que está compuesto por una serie de datos con mayor varianza, en cuanto a la composición y formato original de cada imagen.

En el caso de la red basada en la arquitectura XCeption, se han observado mejores resultados al establecer el *learning rate* del optimizador Adam, en 10^{-3}. Se obtiene una puntuación más equilibrada en exactitud y mas baja en la función de pérdida.

Para la red VGG16, se observa una tendencia similar, en magnitud empleada para el *learning rate*, pero el optimizador que mejor resultado ha dado ha sido SGD.

Finalmente, la red con arquitectura ResNet50 ha obtenido mejores resultados con el optimizador Adam, y el valor del *learning rate* 10^{-3}. Los valores de la exactitud no distan del resto de modelos, pero son equilibrados para ambos conjuntos de test y se obtiene además un valor más bajo que en el resto de modelos para la función de pérdida.

Como criterio de elección del modelo, tras realizar la media de los valores de ratio de éxito obtenidos para los conjuntos de test aplicados, se ha tomado el que alcanza mayor valor que es el modelo basado en la arquitectura ResNet50.

Mapas de Salida

En la primera imagen, en el apéndice I, figura 5, se observa a la izquierda la imagen original y una vez aplicada la herramienta, a la derecha, el resultado devuelto por la misma.

Resaltados en rojo, se observan los *frames* clasificados como positivos, es decir que contienen desconchado o delaminado, donde de manera masiva, se observa que corresponden con las zonas más afectadas por desconchados y descascarillados. El resto de *frames*, que se observan más azulados o verdosos, han sido clasificados como negativos. En este caso, la imagen de arriba corresponde a una única imagen, tomada por el dron.

Al ejecutar la función que permite obtener el mapa de salida, se obtiene además la información que revela el número total de clasificaciones realizadas y la distribución de casos positivos y negativos.

Un segundo ejemplo, lo compone el par de la figura 6, esta vez cada imagen está compuesta, montada manualmente a partir de 4 imágenes tomadas en puntos equiespaciados sobre la superficie retratada. De nuevo se observa, como las zonas afectadas están mayoritariamente cubiertas por *frames* coloreados, mientras que las zonas consideradas libres, mantienen el tono verde/azulado. Se observa como especialmente en la zona de la ventana hay varios falsos positivos, zonas que el modelo ha considerado como contenedoras de delaminado, sin serlo realmente.

Por último, se presenta este tercer par de mapas, compuestos manualmente a partir de una secuencia de 12 fotografías de una fachada tomadas por el dron. Se observa a grandes rasgos cómo las zonas donde se manifiesta el fenómeno se encuentran compuestas fundamentalmente por casillas clasificadas correctamente como positivas. En este caso y de manera más acusada, se observan zonas con falsos positivos, debidos a elementos, no contemplados antes por el sistema como los mostrados en la figura 7.

VII. CONCLUSIONES

Una vez presentada y expuesta toda la labor realizada, se exponen las conclusiones a las que se ha llegado tras la realización el trabajo experimental:

-Se ha confeccionado un *data set* (Delaminations2021), con

6452 imágenes que muestran el fenómeno del delaminado, desconchado o descascarillado sobre superficies de edificios.

-Se han entrenado tres modelos diferentes de redes neuronales convolucionales profundas con el data set creado, siendo capaces de identificar el desperfecto buscado con exactitudes entre el 82% y el 90%.

-Los resultados obtenidos por las redes entrenadas empleando la técnica del *tranfer learning*, y los obtenidos por la red entrenada por completo son equiparables, no se observa un dominio en cuanto a resultados de una técnica sobre otra, para el conjunto de datos de entrenamiento empleado.

-A partir de los modelos entrenados, se ha confeccionado una herramienta, que permite localizar las zonas afectadas por el fenómeno buscado a partir de una imagen proporcionada.

-La herramienta creada, además de proporcionar un mapa visual para el humano, constituye una primera etapa para la intervención automatizada sobre fachadas, proporcionando el diagnóstico necesario, previo a una intervención pictórica.

-Los mapas de diagnóstico obtenidos, donde el sistema es capaz de identificar mayoritariamente las zonas afectadas de una fachada a partir de las imágenes proporcionadas por un dron, ponen de manifiesto que se ha alcanzado el objetivo global marcado para éste trabajo.

Se identifican varios puntos que pueden ser abordados como mejora directa de la herramienta desarrollada y métodos empleados, situando y conectando el aporte realizado con diversas líneas de desarrollo y posibles usos:

-Búsqueda más profunda de modelos a implementar, así como tuneo de hiperparámetros más extenso, para la consecución de un modelo más exacto.

-Empleo de métodos de cosido de imágenes, propios del ámbito de percepción computacional, para elaborar mapas a partir del conjunto de imágenes tomadas por un dron de manera automatizada y no manual.

-Hacer uso de las coordenadas de localización proporcionadas en cada imagen del dron, para elaborar un **mapa de localización de precisión de las zonas afectadas**, para las futuras labores de intervención con drones.

-Empleo de técnicas propias de la Inteligencia Artificial, y concretamente del área de aprendizaje profundo, para diseñar sistemas de control, que guíen al dron automáticamente durante la toma de imágenes para la aplicación de la herramienta en una superficie real, extensa.

-A partir de la proporción de una magnitud dimensional sobre las fotografías tomadas por el dron, podría calcularse la dimensión real total de las zonas afectadas, proporcionando esto una información muy valiosa para el cálculo de los costes de la futura intervención necesaria.

APÉNDICES

Mapas de Salida

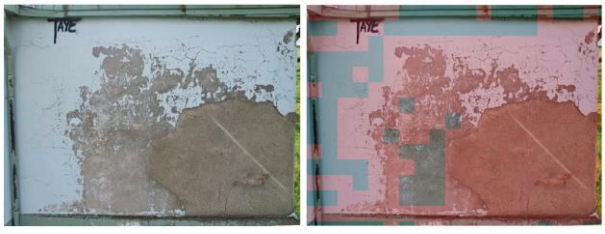

Fig. 5 *Izda.: Imagen original tomada por el dron. Dcha.: Imagen de salida de la herramienta.*

Fig. 6 *Imagen sintetizada manualmente a partir de 4 imágenes tomadas por el dron. Abajo: Resultado de insertar la imagen anterior a la herramienta desarrollada.*

Fig. 7 *Imagen de una fachada sintetizada manualmente a partir de 12 fotografías tomadas por el dron. Arriba: Resultado de la aplicación de la herramienta. Abajo: composición original.*

REFERENCIAS

Adhikari, R. S., Moselhi, O., & Bagchi, A. (2014). Image-based retrieval of concrete crack properties for bridge inspection. Automation in Construction, 39, 180–194. https://doi.org/10.1016/j.autcon.2013.06.011

Azimi, M., Eslamlou, A. D., & Pekcan, G. (2020). Data-Driven Structural Health Monitoring and Damage Detection through Deep Learning: State-of-the-Art Review. Sensors 2020, Vol. 20, Page 2778, 20(10), 2778. https://doi.org/10.3390/S20102778

Bang, H., Min, J., & Jeon, H. (2021). Deep learning-based concrete surface damage monitoring method using structured lights and depth camera. Sensors, 21(8), 2759. https://doi.org/10.3390/s21082759

Bhowmick, S., Nagarajaiah, S., & Veeraraghavan, A. (2020). Vision and deep learning-based algorithms to detect and quantify cracks on concrete surfaces from UAV videos. Sensors (Switzerland), 20(21), 1–19. https://doi.org/10.3390/s20216299

Carrio, A., Sampedro, C., Rodriguez-Ramos, A., & Campoy, P. (2017). A review of deep learning methods and applications for unmanned aerial vehicles. In Journal of Sensors (Vol. 2017). Hindawi Limited. https://doi.org/10.1155/2017/3296874

Cha, Y. J., & Choi, W. (2017). Vision-based concrete crack detection using a convolutional neural network. Conference Proceedings of the Society for Experimental Mechanics Series, 2 Part F2, 71–73. https://doi.org/10.1007/978-3-319-54777-0_9

Cha, Y. J., Choi, W., & Büyüköztürk, O. (2017). Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks. Computer-Aided Civil and Infrastructure Engineering, 32(5), 361–378. https://doi.org/10.1111/mice.12263

Cha, Y. J., Choi, W., Suh, G., Mahmoudkhani, S., & Büyüköztürk, O. (2018). Autonomous Structural Visual Inspection Using Region-Based Deep Learning for Detecting Multiple Damage Types. Computer-Aided Civil and Infrastructure Engineering, 33(9), 731–747. https://doi.org/10.1111/mice.12334

Chollet, F. (2016). Xception: Deep Learning with Depthwise Separable Convolutions. Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, 2017-January, 1800–1807. https://arxiv.org/abs/1610.02357v3

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 580–587. https://doi.org/10.1109/CVPR.2014.81

Gopalakrishnan, K., Gholami, H., Vidyadharan, A., Choudhary, A., & Agrawal, A. (2018). CRACK DAMAGE DETECTION IN UNMANNED AERIAL VEHICLE IMAGES OF CIVIL INFRASTRUCTURE USING PRE-TRAINED DEEP LEARNING MODEL. INTERNATIONAL JOURNAL FOR TRAFFIC AND TRANSPORT ENGINEERING, 8(1), 1–14. https://doi.org/10.7708/ijtte.2018.8(1).01

Kim, J. J., Kim, A. R., & Lee, S. W. (2020). Artificial neural network-based automated crack detection and analysis for the inspection of concrete structures. Applied Sciences (Switzerland), 10(22), 1–13. https://doi.org/10.3390/app10228105

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings. https://arxiv.org/abs/1412.6980v9

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 84–90. https://doi.org/10.1145/3065386

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278–2323. https://doi.org/10.1109/5.726791

Li, S., & Zhao, X. (2019). Image-Based Concrete Crack Detection Using Convolutional Neural Network and Exhaustive Search Technique. Advances in Civil Engineering, 2019. https://doi.org/10.1155/2019/6520620

Liu, Y., Yao, J., Lu, X., Xie, R., & Li, L. (2019). DeepCrack: A deep hierarchical feature learning architecture for crack segmentation. Neurocomputing, 338, 139–153. https://doi.org/10.1016/j.neucom.2019.01.036

Maeda, H., Sekimoto, Y., Seto, T., Kashiyama, T., & Omata, H. (2018). Road Damage Detection Using Deep Neural Networks with Images Captured Through a Smartphone. Computer-Aided Civil and Infrastructure Engineering, 33(12), 1127–1141. https://doi.org/10.1111/mice.12387

Maniat, M., Camp, C. V., & Kashani, A. R. (2021). Deep learning-based visual crack detection using Google Street View images. Neural Computing and Applications, 1–18. https://doi.org/10.1007/s00521-021-06098-0

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The Bulletin of Mathematical Biophysics 1943 5:4, 5(4), 115–133. https://doi.org/10.1007/BF02478259

Mohan, A., & Poobal, S. (2018). Crack detection using image processing: A critical review and analysis. Alexandria Engineering Journal, 57(2), 787–798. https://doi.org/10.1016/j.aej.2017.01.020

Pereira, F. C., & Pereira, C. E. (2015). Embedded image processing systems for automatic recognition of cracks using UAVs. IFAC-PapersOnLine, 28(10), 16–21. https://doi.org/10.1016/j.ifacol.2015.08.101

Phung, M. D., Dinh, T. H., Hoang, V. T., & Ha, Q. P. (2017). Automatic crack detection in built infrastructure using unmanned aerial vehicles. ISARC 2017 - Proceedings of the 34th International Symposium on Automation and Robotics in Construction, 823–829. https://doi.org/10.22260/isarc2017/0115

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision, 115(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y

Shin, H. K., Ahn, Y. H., Lee, S. H., & Kim, H. Y. (2020). Automatic concrete damage recognition using multi-level attention convolutional neural network. Materials, 13(23), 1–13. https://doi.org/10.3390/ma13235549

Simonyan, K., & Zisserman, A. (2015, September 4). Very deep convolutional networks for large-scale image recognition. 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings. http://www.robots.ox.ac.uk/

Vempati, A. S., Kamel, M., Stilinovic, N., Zhang, Q., Reusser, D., Sa, I., Nieto, J., Siegwart, R., & Beardsley, P. (2018). PaintCopter: An autonomous UAV for spray painting on three-dimensional surfaces. IEEE Robotics and Automation Letters, 3(4), 2862–2869. https://doi.org/10.1109/LRA.2018.2846278

Zheng, M., Lei, Z., & Zhang, K. (2020). Intelligent detection of building cracks based on deep learning. Image and Vision Computing, 103, 103987. https://doi.org/10.1016/j.imavis.2020.103987

Zhu, Q., Dinh, T. H., Phung, M. D., & Ha, Q. P. (2021). Hierarchical Convolutional Neural Network With Feature Preservation and Autotuned Thresholding for Crack Detection. IEEE Access, 9, 60201–60214. https://doi.org/10.1109/ACCESS.2021.3073921

Zhu, Z., German, S., & Brilakis, I. (2011). Visual retrieval of concrete crack properties for automated post-earthquake structural safety evaluation. Automation in Construction, 20(7), 874–883. https://doi.org/10.1016/J.AUTCON.2011.03.004