PROCESAMIENTO DE IMÁGENES RGB PARA LA DETECCIÓN SEMIAUTOMÁTICA DE FISURAS EN ESTRUCTURAS DE HORMIGÓN.

RGB IMAGE PROCESSING FOR THE SEMIAUTOMATIC DETECTION OF CRACKS IN CONCRETE STRUCTURES.

TRABAJO FIN DE MÁSTER
CURSO 2022-2023

AUTOR
JORGE SANTIAGO CHAVEZ QUISHPE

DIRECTOR
YOLANDA GARCÍA RUIZ

COLABORADOR
CENTRO DE ESTUDIOS Y EXPERIMENTACIÓN DE OBRAS PÚBLICAS (CEDEX)

MÁSTER EN INTERNET DE LAS COSAS
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

PROCESAMIENTO DE IMÁGENES RGB PARA LA DETECCIÓN SEMIAUTOMÁTICA DE FISURAS EN ESTRUCTURAS DE HORMIGÓN.

RGB IMAGE PROCESSING FOR THE SEMIAUTOMATIC DETECTION OF CRACKS IN CONCRETE STRUCTURES

TRABAJO DE FIN DE MÁSTER EN INTERNET DE LAS COSAS
DEPARTAMENTO DE INFORMÁTICA

AUTOR
JORGE SANTIAGO CHAVEZ QUISHPE

DIRECTOR
YOLANDA GARCÍA RUIZ

COLABORADOR
CENTRO DE ESTUDIOS Y EXPERIMENTACIÓN DE OBRAS PÚBLICAS (CEDEX)

CONVOCATORIA: SEPTIEMBRE 2023
CALIFICACIÓN: 6

MÁSTER EN INTERNET DE LAS COSAS
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

29 DE SEPTIEMBRE DE 2023

DEDICATORIA

Con todo mi cariño dedico este trabajo a mis padres por su apoyo incondicional y haberme guiado a lo largo de mi carrera y de mi vida. A mis hermanos por ser mi fortaleza para seguir y por su ayuda en los buenos y malos momentos.

Jorge

AGRADECIMIENTOS

A mis padres por el gran esfuerzo que han hecho y apoyarme en todo. Siempre han sido mi ejemplo en la vida.

A toda mi familia que me han brindado su apoyo y por creer en mí.

A mi directora Yolanda García por guiarme, aconsejarme y darme su asesoramiento para lograr culminar este trabajo fin de máster, mostrando profesionalismo y gran interés en cada tutoría.

Al Centro de Estudios y Experimentación de Obras Públicas (CEDEX) por su aporte en el desarrollo del trabajo, en particular a Ismael Carpintero García por la oportunidad que nos ha brindado al abrirnos las puertas de la empresa y su predisposición al proyecto.

A los docentes por su entrega total a su vocación, por compartir sus conocimientos a lo largo del desarrollo del máster.

RESUMEN

Procesamiento de imágenes RBG para la detección semiautomática de fisuras en estructuras de hormigón.

El deterioro de las infraestructuras de hormigón da pie al desarrollo de herramientas eficaces y rápidas en el área de la construcción que permitan detectar potenciales patologías estructurales. Con el fin de actuar a tiempo evitando degradaciones en la estructura y posibles fallos en el futuro, el presente trabajo propone una metodología inicial para la detección y análisis de fisuras de manera semiautomática. Se utilizó una red neuronal convolucional ResNet para entrenar un modelo de detección de fisuras utilizando imágenes RGB. Luego, este modelo entrenado se aplicó a imágenes tomadas en campo por la empresa CEDEX para clasificar si contenían fisuras o no. Este enfoque demostró ser altamente efectivo para la detección de fisuras en condiciones del mundo real. Una vez etiquetadas las imágenes se calculó los valores de área y número de fisuras de cada imagen, estos valores fueron vinculados a una plataforma de IoT, ThingSpeak, que permite tener los datos en la nube, y acceder a ellos desde cualquier dispositivo vinculado a la cuenta. Además, se configuró la plataforma de tal manera que envía alertas al correo electrónico de los datos procesados. En resumen, a través de este proceso se ha desarrollado una herramienta inicial que se utiliza para la detección de fisuras en estructuras. Con un mayor desarrollo en el futuro, esta herramienta IoT tiene el potencial de evolucionar y ser capaz de clasificar automáticamente diversas patologías en las estructuras, lo que representa un avance significativo en la inspección y mantenimiento de infraestructuras.

Palabras clave

ResNet, hormigón, fisuras, clasificación, ThingSpeak

ABSTRACT

RGB image processing for the semiautomatic detection of cracks in concrete structures.

The deterioration of concrete infrastructures gives rise to the development of effective and fast tools in the construction area to detect potential structural pathologies. In order to act in time to avoid structural degradation and possible future failures, this paper proposes an initial methodology for semi-automatic crack detection and analysis. A ResNet convolutional neural network was used to train a crack detection model using RGB images. This trained model was then applied to images taken in the field by CEDEX to classify whether they contained cracks or not. This approach proved to be highly effective for crack detection in real-world conditions.Once the images were labelled, the area and number of cracks values were calculated for each image, these values were linked to an IoT platform, ThingSpeak, which allows the data to be held in the cloud, and accessed from any device linked to the account. In addition, the platform was configured in such a way that it sends email alerts of the processed data. In summary, through this process an initial tool has been developed that is used for crack detection in structures. With further development in the future, this IoT tool has the potential to evolve and be able to automatically classify various pathologies in structures, representing a significant advance in infrastructure inspection and maintenance.

Keywords

ResNet, concrete, cracking, classification, ThingSpeak

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1 - Introducción

En la actualidad, el hormigón es el material más utilizado para el desarrollo de cualquier infraestructura. La preparación técnica, desde el diseño hasta las etapas finales de construcción, debe cumplir las normativas y especificaciones propias del material, y junto con las modernas tecnologías se obtienen obras seguras, duraderas y estéticas. Sin embargo, estas estructuras en varias ocasiones se ven afectadas por grietas y fisuras (Toirac Corral, 2004).

Las estructuras de hormigón están sujetas a diversos factores que pueden conducir a la formación y propagación de fisuras (Toirac Corral, 2004), comprometiendo la integridad estructural, afectando la estética y provocando filtraciones de agua, lo que a su vez podría acelerar la degradación. Otros aspectos a tener en cuenta en la propagación de las fisuras son: la calidad del hormigón, el diseño estructural, las cargas aplicadas, la temperatura y humedad ambiental, así como los procesos de construcción y mantenimiento. Por dicha razón, surge la necesidad de desarrollar y/o validar métodos que permitan inspeccionar el estado de la estructura (Ercolani, Ortega, & Felix, 2015).

Colpari (2020) señala que la inteligencia artificial, en la actualidad, tiene un rol muy importante en la industria 4.0, donde se emplean sistemas inteligentes y tecnologías para crear una conexión entre los trabajos físicos y virtuales (Darko et al., 2020). Una parte de la inteligencia artificial es la visión computacional, la cual se enfoca en extraer información sobre una escena al analizar imágenes de dicha escena (Rosenfeld, 1998).

La detección automática de fisuras en el hormigón utilizando imágenes es un campo de investigación y desarrollo que combina la visión por computadora y la ingeniería civil (Mohan & Poobal, 2018). El objetivo es desarrollar algoritmos y sistemas que puedan identificar y analizar automáticamente las fisuras en estructuras de hormigón a partir de imágenes digitales, lo que puede ayudar a los ingenieros y profesionales de la construcción a evaluar el estado de las estructuras y tomar medidas preventivas o de reparación (Paglinawan et al., 2018).

En este contexto, este trabajo surge a raíz de la necesidad existente en la empresa CEDEX (Centro de estudios y experimentación de Obras Públicas), perteneciente al Ministerio de Transportes, Movilidad y Agenda Urbana, de aplicar metodologías innovadoras que permitan identificar automáticamente y comparar patrones de fisura en construcciones de hormigón a partir de imágenes RGB, el hormigón pasa a ser el material de estudio, al presentar una alta densidad de fisuras en su estructura.

1.1 Objetivos

1.1.1 Objetivo General

El objetivo principal del trabajo es generar un algoritmo clasificador de imágenes de estructuras de hormigón que presenten fisuras y obtener características de las mismas.

1.1.2 Objetivos específicos

Obtener y analizar bancos de imágenes de fisuras en superficies de hormigón para el entrenamiento y validación de una red neuronal convolucional.

Normalizar las imágenes RGB proporcionadas por la empresa para realizar la clasificación de las mismas.

Generar un algoritmo clasificador de imágenes con fisura y sin fisura mediante el uso de diferentes modelos de redes neuronales convolucionales para la clasificación de las imágenes proporcionadas por la empresa.

Aplicar técnicas de tratamiento de imágenes, como el uso de filtros, eliminación de ruido, detección de bordes, para lograr una mejor distinción entre las fisuras y el fondo en las imágenes

Extraer características de las fisuras mediante la aplicación de técnicas de segmentación.

1.2 Estructura de la memoria

La memoria de este trabajo se estructura de la siguiente forma:

En el Capítulo 2 se describe el estado del arte, que a su vez se subdivide en 4 secciones. En la primera sección se trata el tratamiento de imágenes RGB para la clasificación y extracción de valores métricos. La segunda sección aborda el tema de aprendizaje supervisado y su aplicación en el campo de tratamiento de imágenes. La tercera sección trata de las redes neuronales y su aplicación en la clasificación de imágenes. Finalmente, en la última sección, se mencionan algunos trabajos relacionados con este proyecto de investigación.

En el Capítulo 3 se presenta la metodología utilizada para el desarrollo del trabajo, se describe la red neuronal utilizada para la clasificación, las bases de datos con las que se realiza el entrenamiento de la red y también se describe el tratamiento que han tenido las imágenes para la extracción de sus valores numéricos y la carga de esta información en la nube.

En el Capítulo 4 se exponen los resultados obtenidos en el desarrollo del proyecto. Asimismo, la discusión en función de los objetivos planteados y de la información bibliográfica recolectada y analizada.

El Capítulo 5 se describen las conclusiones finales del trabajo y su alcance.

Por último, el Capítulo 6 se presenta un apartado en que se explican los futuros trabajos que se abren a partir del este proyecto.

1.3 Herramientas utilizadas y repositorio

Se recurrió a diversos repositorios académicos para la recopilación de artículos de revistas científicas, tales como: Scopus, Dialnet, Scielo, Web of Science, Depósito de Investigación Universidad de Sevilla – idUS, Portal de Revistas Científicas Complutenses, DOAJ (Directory of Open Access Journals), Directory of Open Access Repositories (OpenDOAR), Docta Complutense, CORE, IEEE Xplore, entre los más importantes. En la Tabla 1 se enlistan las palabras claves (español/inglés) utilizadas para la búsqueda de la información.

Palabras clave	
1. Image Processing	2. Hormigón
3. Inteligencia artificial	4. Estructuras
5. Redes neuronales	6. Crack Detection
7. Convolutional Neural Network	8. Clasificación de Imágenes
9. Fisura	10. Deep learning
11. Crack detection	12. Image recognition

Tabla 1 Palabras clave para la búsqueda de información

Las bases de datos de las imágenes utilizadas para el entrenamiento y validación de la red neuronal se pueden consultar a través de los siguientes links.

- Base de datos Kaggle: https://www.kaggle.com/datasets/arnavr10880/concrete-crack-images-for-classification.
- Base de datos SDNET2018: https://digitalcommons.usu.edu/all_datasets/48/

Por otro lado, las imágenes utilizadas para la extracción de la información de las fisuras, fueron proporcionas por CEDEX (Centro de estudios y experimentación de Obras Públicas), son imágenes RGB, en formato JPG, de 2048 x1536 pixeles, 24 bit de profundidad.

El hardware utilizado para el desarrollo del proyecto es un ordenador Intel® Core ™ i7-1065G7, memoria RAM de 16 GB y un sistema operativo de 64 bits con un procesador de x64. El principal software utilizado para el desarrollo del trabajo es MATLAB versión R2020. La Universidad Complutense de Madrid, cuenta con una licencia Total Academic Headcount (TAH) para MATLAB, Simulink y productos complementarios. Esta licencia tiene acceso a todas las toolbox y productos de Matlab. Matlab utiliza su propio lenguaje de programación (lenguaje M) que es un lenguaje de alto rendimiento para cálculos técnicos.

El código generado, así como las bases de datos utilizadas se pueden consultar en el repositorio:

https://github.com/JorTiag/DeteccionDeFisurasEnEstructurasDeHormigonConCNN.git

Capítulo 2 - Estado del arte

2.1 Redes neuronales artificiales y el aprendizaje supervisado

Las Redes Neuronales Artificiales (ANN) son sistemas de procesamiento de la información cuya estructura y funcionamiento están inspirados en las redes neuronales biológicas (Hilera & Martínez, 1995) y en las últimas décadas han desarrollado un gran aporte en la minería de datos, permitiendo la modelación de información a través de relaciones de forma inductiva por medio de algoritmos de aprendizaje.

Las ANN utilizan arquitecturas compuestas de transformaciones no lineales múltiples. El procesamiento de la información se desarrolla en muchos elementos simples denominados neuronas. El paso de las señales entre neuronas se realiza a través de las conexiones sinápticas. Estos enlaces tienen un peso denominado peso sináptico que pondera la señal transmitida. Finalmente, cada neurona aplica la función de activación o transferencia a las entradas para determinar la salida. Además, cabe mencionar que el tipo de procesamiento de la información es en paralelo. En el sentido de que muchas neuronas pueden estar funcionando al mismo tiempo (Palmer Pol & Montaño Moreno, 1999).

Una de las principales ventajas de las ANN es su orientación a la realización de distintas tareas relacionadas con la inteligencia artificial, divididas en dos grandes grupos: clasificación supervisada y la clasificación no supervisada. En clasificación no supervisada, o también denominado autoorganizado, no es necesario especificar o entrenar a la red para saber cuál es la respuesta correcta y se permite realizar sus propias asociaciones. Su función principal es identificar patrones, regularidades o categorías dentro de los datos que se le proporcionan como entrada. Se utiliza para tareas autoasociativas y para agrupar datos basados en similitudes. (Palmer Pol & Montaño Moreno, 1999).

Por contrario, una clasificación supervisada entrena a una red con datos de entrada consistentes en un patrón y respuesta esperada, esto permite obtener un porcentaje de aciertos y errores, debido a que el resultado obtenido por la red puede ser comparado con el resultado esperado (real), permitiendo ajustes incrementales del modelo hasta que se acerque más a la respuesta esperada (Águila Martínez, 2017) . Este tipo de aprendizaje es muy útil para la clasificación de patrones y para la aproximación de funciones. En el proceso de entrenamiento, los pesos se ajustan iterativamente para minimizar el error entre la salida de la red y la salida

deseada del usuario, utilizando los valores del conjunto de entrenamiento. (Bonilla & Puertas, 1997).

El aprendizaje supervisado utiliza un conjunto de datos de entrenamiento para crear un modelo predictivo. Entre los métodos más comunes de este proceso se incluyen: el clasificador lineal, el clasificador basado en árboles de decisión, la máquina de vector soporte (SVM), las redes neuronales convolucional (CNN) y los métodos basados en ensamble, como el bosque aleatorio y el aumento de gradiente (Varela-Arregoces & Campbells, 2011).

Las CNN son Redes Neuronales Artificiales con capas ocultas especializadas que forman una jerarquía. Comienzan detectando propiedades simples y se especializan gradualmente en capas más profundas para reconocer detalles específicos en imágenes, permitiendo la generalización. (Artola Moreno, 2019).

2.2 Aplicaciones de las redes neuronales

Las ANN pueden producir resultados aceptables en aplicaciones donde las entradas son incompletas o contienen ruido. Esto se debe a su capacidad para aprender patrones y adaptarse a la información disponible, lo que les permite manejar datos imperfectos y aun así proporcionar resultados útiles. Las aplicaciones de las ANN se extienden a una amplia variedad de campos, incluyendo la biología, medicina, militar, manufacturación, finanzas, psicología y psiquiatría, empresas, medio ambiente (McCord Nelson & Illingworth, 1991), entre otros. Respecto a las áreas de aplicación se encuentran las siguientes: análisis y procesado de señales, reconocimiento y clasificación de imágenes, robótica, filtrado de ruido, obtención de modelos de retina, predicción de rendimiento académico, monitorización de cirugías entre otros (Olabe, 1998).

La clasificación de imágenes se refiere a la tarea de asignar una etiqueta o categoría a una imagen dada. El rendimiento de una red neuronal está fuertemente influenciado por la calidad de las imágenes de entrenamiento. Los algoritmos clasificadores construyen un modelo a partir de un conjunto de imágenes categorizadas, y luego este modelo se utiliza para clasificar nuevas imágenes (Núñez Sánchez-Agustino, 2016). Entre los desafíos y las limitaciones se encuentran el sobreajuste del modelo, la necesidad de grandes conjuntos de datos de entrenamiento etiquetados, la falta de generalización en dominios desconocidos y la interoperabilidad del modelo.

2.3 Tratamiento de imágenes

Se realizan operaciones de pre-procesamiento en el sistema para mejorar los resultados de la red neuronal, tanto en el módulo de segmentación de caracteres como en la carga del conjunto de imágenes de entrenamiento.

Primero, se recomienda que las imágenes introducidas en CNN tengan dimensiones superiores a 187 píxeles y que los píxeles abarquen todo el rango dinámico. Para cumplir con estas especificaciones las imágenes ingresan a una etapa de redimensionamiento mediante el acolchado a cero, que consiste en añadir ceros al perímetro de la imagen original hasta alcanzar las dimensiones de 224x224 píxeles, seguido de un escalado de píxeles al rango de 0 a 255 y, si es necesario, de la replicación de capas, donde se consigue que la imagen tenga tres capas (RGB), quedando una nueva matriz con dimensiones de 224x224x3. En el caso de las imágenes que originalmente tenían una sola capa, se replica la misma capa en todos los canales (Pérez-Careta et al., 2002).

Además, existen procesos que permiten corregir las distorsiones, uno de ellos es el realce de imágenes. Este proceso se puede realizar pixel por pixel o por operaciones de procesamiento en un grupo de pixeles. Estas operaciones permiten realzar las características de brillo y contraste de una imagen, reducir su contenido de ruido, o agudizar o intensificar detalles presentes en ella.

El filtrado espacial es un método para mejorar o cambiar una imagen mediante un proceso llamado convolución espacial. Este proceso se desplaza a través de la imagen de entrada pixel por pixel y determina el valor de brillo en la imagen de salida utilizando información de los pixeles circundantes. La convolución espacial permite filtrar la imagen en función de la actividad de frecuencia espacial en esa área, lo que mejora o modifica la imagen en base a su contenido de frecuencia espacial. (Aldalur & Santamaría, 2002).

La binarización es un proceso de simplificación de una imagen en escala de grises basado en el análisis de su histograma. A veces, en lugar de una simple división en blanco y negro, se divide el histograma en varias partes, lo que resulta en una imagen con múltiples niveles de grises en lugar de solo dos (Gallo Sánchez, 2013).

La segmentación de las imágenes determina la geometría de las diversas estructuras que componen la imagen, en donde, las regiones de interés (ROI) deben ser uniformes y homogéneas en alguna característica en concreto, deben tener la menor cantidad de ruido posible y deben diferir de las regiones adyacentes.

2.4 Trabajos relacionados

En la búsqueda bibliográfica se recopilo información acerca de los métodos relacionados con la inteligencia artificial y el tratamiento de imágenes. Existen diferentes algoritmos y métodos para la recolección de información (imágenes) que permiten detectar fisuras en estructuras de hormigón.

El estudio de Paglinawan et al. (2019) se enfoca en el análisis de una sola fisura en un muro de hormigón, abarcando la captura de la imagen, procesamiento y medición del espesor de la fisura. Utilizan 20 imágenes, 10 tomadas a diferentes distancias de la fisura y 10 de zonas no fisuradas. El proceso incluye escala de grises, umbral adaptativo, filtro de mediana, morfología de imagen, contorno ajustado y morfoesquelético para detectar la fisura. La clasificación muestra un 10% de error y un 20% de falsos positivos debido a sombras y defectos en la pintura. El espesor óptimo se encuentra a 58 cm con un error menor al 10%.

Por otro lado, Prasanna et al., (2016) emplean un robot con cámaras y un Ground-Penetrating Radar para inspeccionar puentes. Utilizan un clasificador STRUM que consta de tres fases: detección de líneas robustas, cálculo de características en un espacio calibrado y un clasificador de aprendizaje automático. Las imágenes se categorizan manualmente para un entrenamiento sólido y mejores resultados.

El principal logro de esta investigación es la representación del aspecto de la fisura como un vector de diversas características calibrado en el espacio, considerando intensidad, gradientes y escala espacial. Esto lleva al algoritmo STRUM a lograr una precisión superior al 90% en la detección de fisuras.

Por otro lado, Cha et al. (2017) utiliza el método de aprendizaje profundo con el uso de CNN, para el entrenamiento de la red, se utilizó una base de datos con 332 imágenes de las cuales 277 se clasificaron en el grupo de entrenamiento y se dividieron en 2 grupos: imágenes con fisura e imágenes sin fisura y las 55 restantes se designaron para la etapa de testeo.

La metodología utiliza MatConvNet para una red neuronal profunda con 8 capas, incluyendo capas convolucionales, de agrupamiento y una capa final que determina la presencia de fisuras en las imágenes (Colpari, 2020). Se generó un clasificador de fisuras con un total de 4000 imágenes de entrenamiento, y la CNN aprendió características mediante la actualización de pesos. Los resultados indican una precisión de más del 97.95% en las 55 imágenes de prueba (Cha et al., 2017).

Silvia y Lucena (2018) al igual que, Cha et al. (2017) utilizan un método de aprendizaje profundo para la clasificación, sin embargo este proceso involucraba 4 fases: 1) crear un conjunto de datos de imagen clasificada de referencia, 2) establecer el punto de referencia para el sistema de inteligencia artificial, 3) implementar el enfoque de transferencia de aprendizaje y 4) realizar los experimentos de entrenamiento.

En la primera fase se utilizó un total de 851 imágenes de 756x756 pixeles, estas imágenes se recortaron a una resolución de 256x256 pixeles, obteniendo un total de 3500 muestras, se clasificaron en 2 grupos: fisuradas y no fisuradas. Este conjunto de datos se dividió en datos de entrenamiento y prueba en una proporción de 80/20.

Para establecer el punto de referencia del sistema se utilizó la red neural convolucional VGG16, que es una CNN que consta de 16 capas de profundidad, de las cuales 13 son convolucionales y 3 están completamente conectadas. El tamaño de la muestra era relativamente pequeño, lo que da lugar a la fase 3 del proceso, la transferencia de aprendizaje utilizando un estudio experimental para evaluar la influencia de parámetros de entrenamiento como la tasa de aprendizaje, el número de neuronas (o nodos) en la última capa completamente conectada y el tamaño del conjunto de datos de entrenamiento en la precisión del modelo de clasificación. De los valores propuestos por los autores determinaron que el mejor presentaba un resultado de 92.27%, y determinando que la variable más importante es el tamaño del set de datos.

Capítulo 3 - Metodología

3.1 ResNet

Las redes neuronales convolucionales (CNN) han revolucionado el campo de la visión por computadora en los últimos años, logrando avances significativos en tareas como el reconocimiento de objetos, la clasificación de imágenes y la detección de objetos.

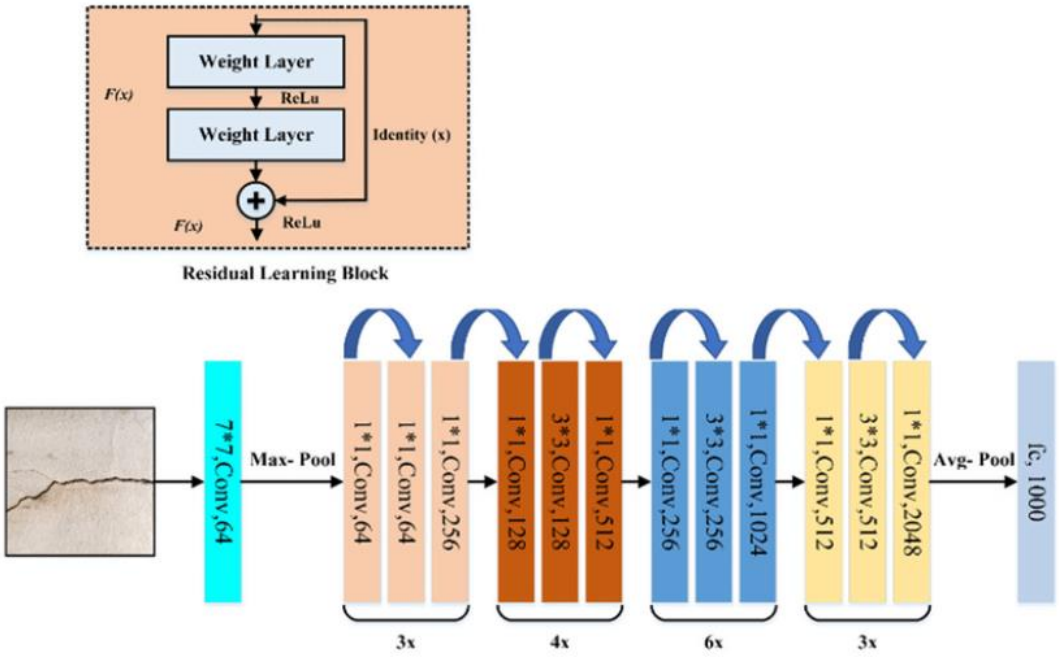
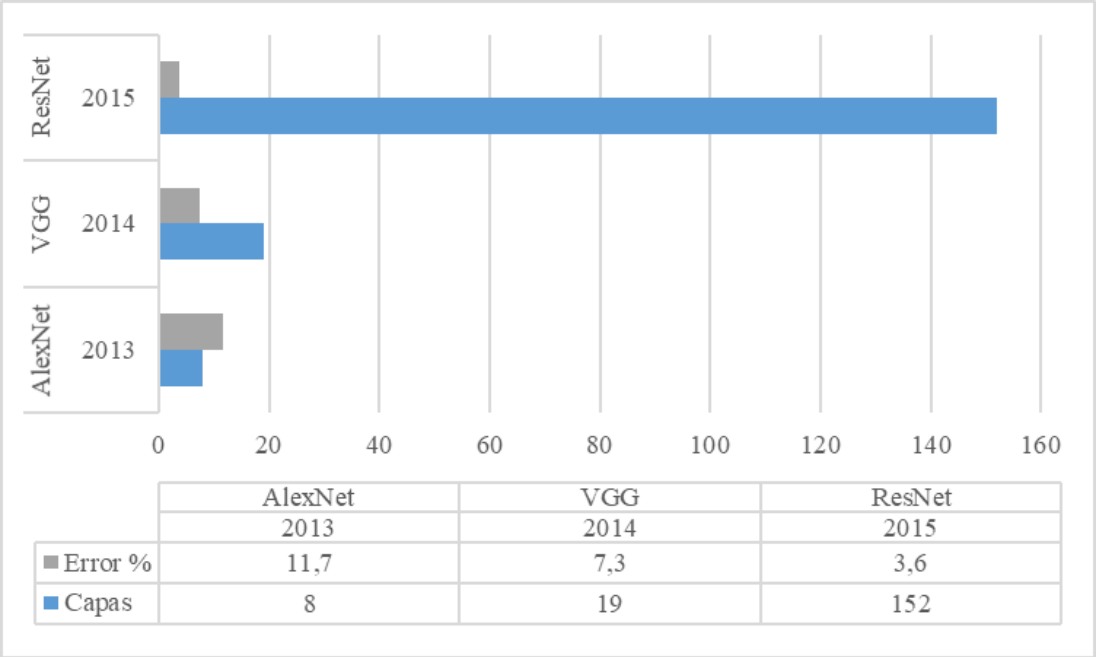


Figura 1 La arquitectura del modelo ResNet-50 presentada por Ali et al. (2021).

En 2015, los investigadores de Microsoft Research introdujeron una arquitectura innovadora llamada ResNet (He et al., 2015), abordando el problema de la degradación del rendimiento en las CNN (Figura 1). ResNet introdujo el concepto de conexiones residuales, que permiten que las capas de una red aprendan las diferencias residuales en lugar de aprender representaciones completas. Este redujo la tasa de error hasta el 3.57% gracias a la implementación de bloques residuales y 152 capas, tal como se muestra en la Figura 2 (Simonyan & Zisserman, 2014). Esto facilita el entrenamiento de redes más profundas sin sufrir degradación en el rendimiento y ha llevado a mejoras significativas en el campo del aprendizaje profundo.

El paradigma ResNet tuvo un impacto significativo al permitir el entrenamiento exitoso de redes neuronales profundas con más de 100 capas, superando con éxito el problema del Desvanecimiento de Gradiente (Rivera, 2019).



	AlexNet	VGG	ResNet
	2013	2014	2015
Error %	11,7	7,3	3,6
Capas	8	19	152

Figura 2 Redes relevantes en la Competencia ImageNet. Fuente: elaboración propia.

3.1.1 Fundamentos teóricos de ResNet:

ResNet-50 es una red neuronal convolucional con 50 capas de profundidad. El primer elemento de la red es la capa de entrada de la imagen (Figura 3), como se especifica, el tamaño de las imágenes es de 224 x 224 en RGB (3 canales de color). En la Figura 4 se presenta la arquitectura de CNN e información detallada sobre sus capas.

```
                    Name: 'input_1'
              InputSize: [224 224 3]

Hyperparameters
       DataAugmentation: 'none'
          Normalization: 'zerocenter'
 NormalizationDimension: 'auto'
                   Mean: [224×224×3 single]
```

Figura 3 Propiedades de la primera capa. Fuente: elaboración propia.

Además, en la arquitectura de ResNet-50 se describen las características clave de la arquitectura de ResNet, incluyendo las capas residuales y las conexiones skip.

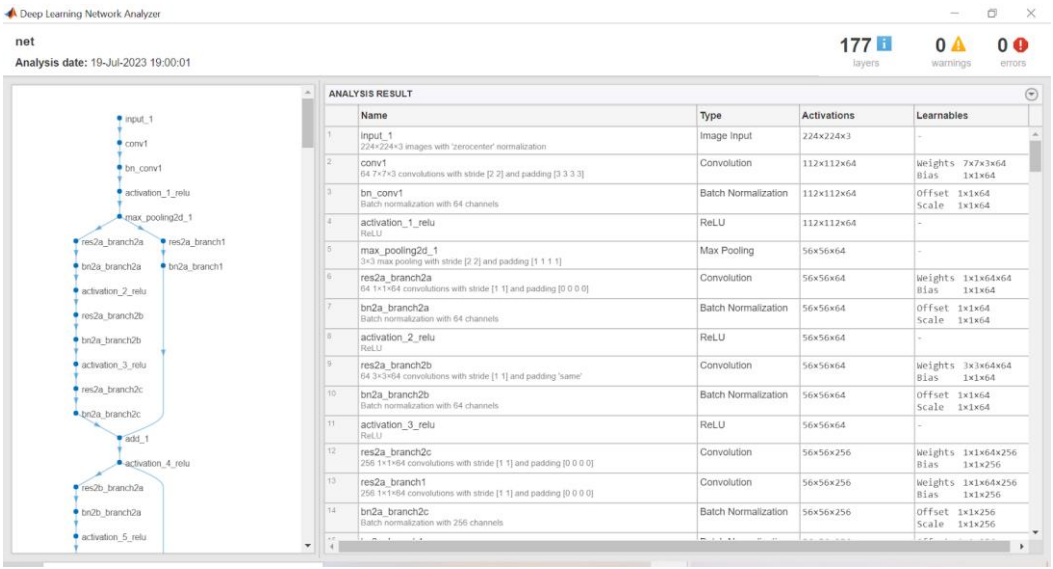


Figura 4 Arquitectura de ResNet. Fuente: elaboración propia.

Todas las convoluciones, incluidas las que se encuentran dentro de los módulos, utilizan activación lineal rectificada (ReLU) definida por: $f(x) = máx(o, x)$, donde x es la entrada de la neurona. Este tipo de capa se puede utilizar múltiples veces en una misma red de neuronas artificiales.

La función de activación, que ajusta los pesos de una neurona en función de su resultado, es ampliamente utilizada en el aprendizaje profundo para mejorar la precisión y velocidad del entrenamiento. Su propósito es agregar no linealidad sin modificar los campos receptivos de la capa convolucional (Bonilla Carrión, 2020).

El proceso de convolución utiliza la matriz de Kernel, para ir operando matemáticamente (producto escalar) los pixeles cercanos generando una nueva matriz de salida, Figura 5, que crea una nueva capa de neuronas ocultas. Al utilizar imágenes RGB la matriz kernel realmente sería de 7x7x3, es decir: un filtro con 3 kernels de 3×3; luego los 3 filtros se suman (y se le suma una unidad bias) y conformarán 1 salida (cómo si fuera 1 solo canal).

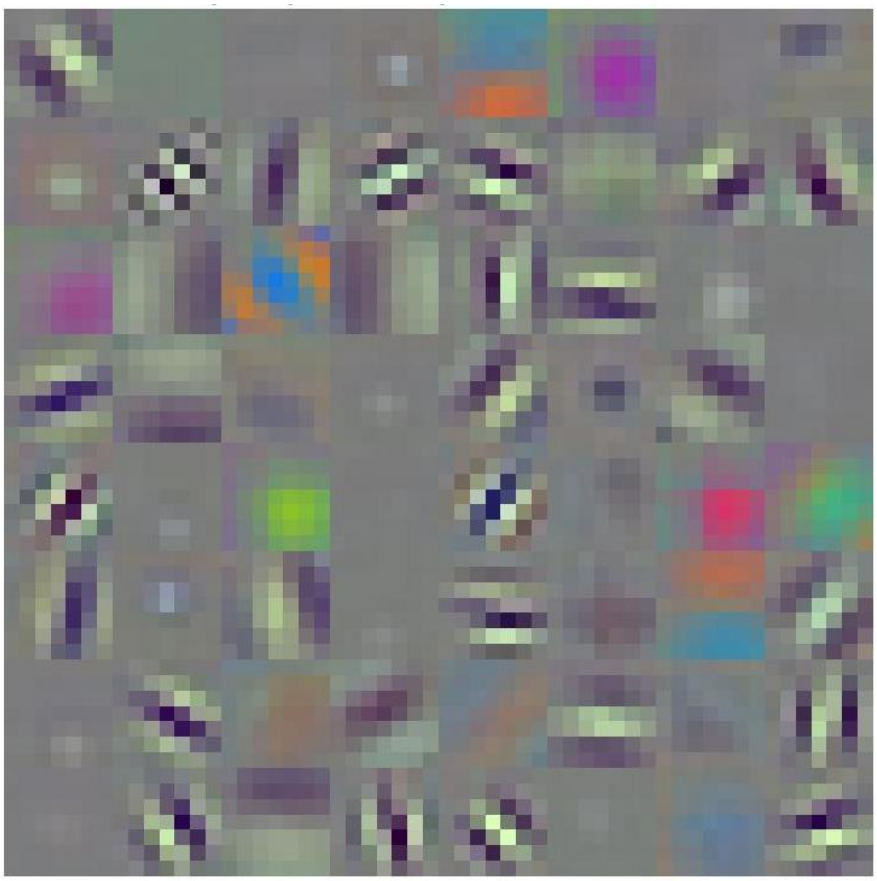


Figura 5 Capa de pesos de la primera convolución de ResNet. Fuente: elaboración propia.

3.2 Obtención y tratamiento del Dataset

En el presente proyecto de investigación se utilizaron 2 bases de datos: *Concrete Crack Images for Classification* (Tabla 2) y *A concrete crack image dataset for machine learning applications* (Tabla 3) para el entrenamiento y validación de la red neuronal.

Nombre	**Fuente**	**Descripción**
Concrete Crack Images for Classification	Kaggle (Özgenel, 2019)	Se dividen en 2 clases: negativas y positivas (a fisuras) Cada clase tiene 20000 imágenes de 227 x 227 píxeles con canales RGB. (Figura 6)

Tabla 2 Características de Kaggle dataset

Figura 6 Ejemplo de la categoría "Negativa"(izq) y "Positiva"(der). Fuente: base de datos Kaggle (Özgenel, 2019)

Nombre	Fuente	Descripción
A concrete crack image dataset for machine learning applications	Utah State University (Maguire et al.,2018)	Contiene más de 56,000 imágenes de cubiertas, paredes y pavimentos de puentes de concreto Fisura y No Fisura, de 256 x 256 píxeles en formato JPG. (Figura 7)

Tabla 3 Características de SDNET2018

Figura 7 Ejemplo de la categoría "Negativa"(izq) y "Positiva"(der) Fuente: base de datos SDNET (Maguire et. al.,2018).

Gao y Mosalam (2018) destacan que el uso de redes neuronales y modelos pre-entrenados ha hecho que ciertos conjuntos de datos, como ImageNet, sean esenciales para el entrenamiento en el reconocimiento de objetos visuales mediante inteligencia artificial. Sin embargo, notan que estos datos no incluyen imágenes específicas para entrenar la detección de fisuras. Para resolver esta limitación, abordan el problema entrenando la red utilizando las bases de datos mencionadas anteriormente.

Para el entrenamiento de la red neuronal, es necesario definir 2 grupos en las bases de datos: entrenamiento y testeo. Estos se dividen en una proporción de 70/30. Es importante que el conjunto de datos de testeo tenga un volumen suficiente como para generar resultados estadísticamente significativos, y a la vez, que sea representativo del conjunto de datos global. Hay que tener en cuenta que para la validación de la red entrenada se utilizara las imágenes proporcionas por la empresa (Figura 8).

Figura 8 Estructuras de hormigón con fisuras (CEDEX,2023)

En la Tabla 4, se muestra el número de imágenes utilizadas para cada paso del entrenamiento y testeo de cada categoría analizada en el proyecto.

Dataset	Entrenamiento		Testeo		Total
	Con Fisura	Sin Fisura	Con Fisura	Sin Fisura	
Kaggle	14000	14000	6000	6000	40000
SDNET	5940	5940	2544	2544	16968
Combinación	19940	19940	8544	8544	56968

Tabla 4 Base de datos

3.3 Entrenamiento de la ResNet

Para el entrenamiento de la red neuronal hay que tener en cuenta ciertas consideraciones: es necesario que el número de elementos de las 2 categorías (Figura 9) sea el mismo, en este caso el número de imágenes "Con Fisuras" obligatoriamente debe ser el mismo que el número de imágenes "Sin Fisura", de no ser así, se observaría un sobreajuste en el entrenamiento para la categoría que tenga mayor cantidad de datos y una baja exactitud en el testeo en la categoría contraria.

El código utilizado para el entrenamiento de la red se muestra en el Apéndice A. Una vez entrenada la red, se hace una validación del proceso con los datos de testeo. Finalmente, se obtiene una matriz de confusión y el valor de exactitud de la clasificación.

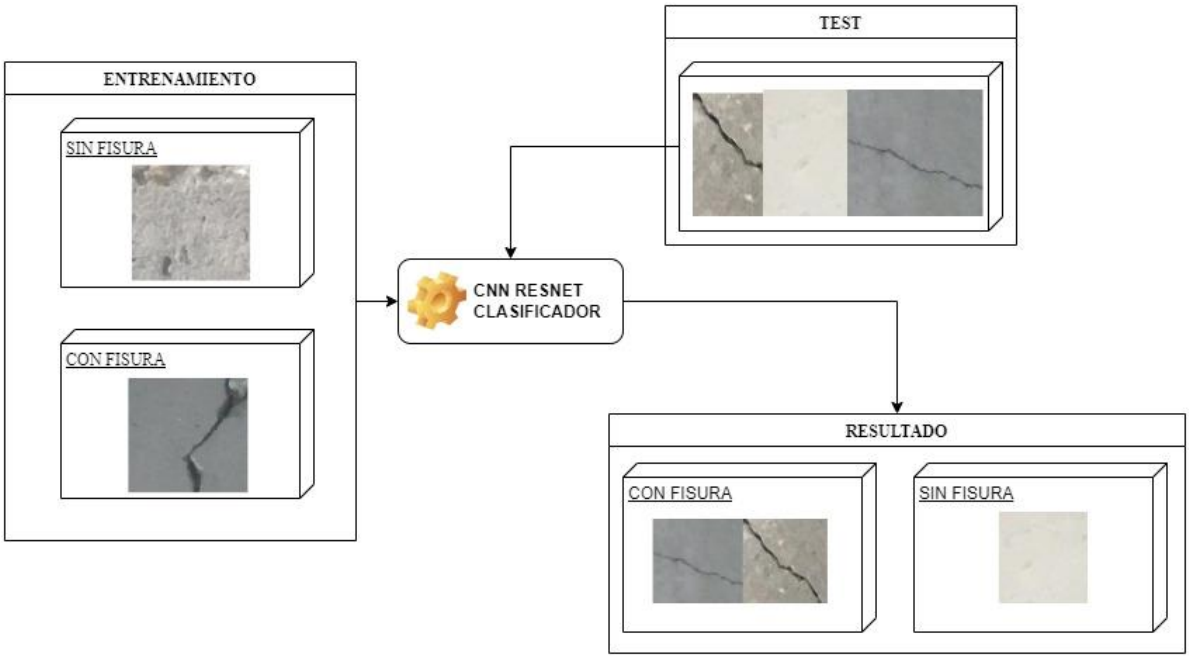


Figura 9 Funcionamiento de la red. Fuente: elaboración propia.

3.4 Tratamiento de las imágenes

Las imágenes utilizadas para la clasificación son las proporcionas por la empresa CEDEX. Además, para esta sección se utilizarán únicamente las imágenes con extensión JPG, clasificadas previamente como “Con Fisura”, de las que se extrae información métrica.

Cabe mencionar que CEDEX proporcionó varias fotografías de fisuras en estructuras de hormigón para la realización del proyecto, sin embargo, en necesario que las imágenes cumplan ciertos requisitos para que su clasificación sea la correcta.

1. Es importante que la imagen sea lo más “limpia” posible. Este es un aspecto principal, debido a que, si la imagen tiene demasiados objetos, la red no la clasificará correctamente. Además, lo ideal sería que la fisura no tenga un aspecto similar al fondo, si esto sucede los valores de los pixeles serán parecidos y afectara en el cálculo de los valores métricos.
2. Matlab a través de su toolbox permite leer y escribir archivos de extensión BMP, GIF, JPEG, PNG y TIFF, por lo tanto, las imágenes que ingresen a la clasificación deben ser en los formatos antes mencionados.
3. El tamaño mínimo de la entrada de imagen de la CNN es de 224 x 224 pixeles.
4. Las variaciones del punto de vista influyen en la captura de la imagen. Este punto toma más relevancia en el análisis multi-temporal de una fisura, debido a que se analiza el crecimiento de una fisura. Es indispensable que esta imagen se capture de la misma forma siempre, si la cámara cambia de posición es muy posible tener como resultado dos imágenes completamente diferentes.
5. Otro aspecto es la iluminación. Una imagen puede variar dependiendo de la hora del día que se la capture. Es evidente que una imagen que se tome en el día va a ser más clara que una que se capture en la noche. No hay que olvidar que el análisis se realiza dependiendo de los valores de los pixeles de la imagen. Además, hay que tener en cuenta la sombra que se genera por la posición del sol.
6. La distancia que exista entre cámara y estructura debe ser siempre la misma, evitando problemas con la variación de las escalas.

Finalmente, se ha seleccionado 4 imágenes, que se muestra en la Tabla 5, que cumplen con los requisitos antes enumerados.

Nº de Imagen	Imagen	Etiqueta
Imagen 1		Con Fisura
Imagen 2		Sin Fisura
Imagen 3		Con Fisura
Imagen 4		Con Fisura

Tabla 5 Imágenes a clasificar.

3.4.1 Filtro de imágenes:

Aplicar un filtro a una imagen es una de las principales maneras de trabajar con ella para: suavizar la imagen, con la que se pretenden reducir la cantidad de variaciones de intensidad entre píxeles vecinos. Otra de las funciones principales de este tratamiento es la eliminación de ruido para tratar aquellos píxeles cuyo nivel de intensidad es muy distinto al de sus vecinos. Realzar y detectar bordes también es un proceso de interés, debido a que permite destacar los

bordes de los objetos contenidos en una imagen detectando los píxeles donde se produce un cambio brusco de intensidad (Sada Pezonaga & Sanz Delgado, 2015).

El filtrado espacial es una técnica comúnmente utilizada para eliminar el ruido de una imagen. Consiste en aplicar un filtro a través de una ventana deslizante en la imagen, donde cada píxel se reemplaza por una combinación lineal de los píxeles vecinos. Los filtros más utilizados incluyen el de media, mediano y Gauss. Estos filtros son eficaces para reducir el ruido impulsivo y el ruido aleatorio, preservando al mismo tiempo los detalles importantes de la imagen.

Entonces, dada una imagen $f(i,j)$, el procedimiento consiste en generar una nueva imagen $g(i,j)$ cuya intensidad para cada píxel se obtiene promediando los valores de intensidad de los píxeles $f(i,j)$ incluidos en un entorno de vecindad predefinido.

En Matlab la función que me permite realizar un filtro de la media será: imfilter (Carela, 2019). Sin embargo, previamente se configura mediante el filtro *fspecial,* esta función crea filtros bidimensionales del tipo especificado por type, que entre las sintaxis disponibles se encuentra "log" que devuelve un filtro laplaciano-gaussiano rotacionalmente simétrico de tamaño *hsize* con desviación estándar *sigma*.

$$h = fspecial('log', hsize, sigma)$$

El tamaño del filtro es especificado como entero positivo o vector de 2 elementos de enteros positivos, en la función "*log*", el tamaño del filtro por defecto es $[5\ 5]$. El valor predeterminado de sigma es de 0.5, debe ser un numero positivo.

Entonces, siguiendo con la estructura de "*imfilter"* se define por: $B = imfilter(A,h)$, donde A es la imagen y h es el filtro antes mencionado (máscara). Es importante definir correctamente el tamaño de la máscara teniendo en cuenta que cuanto más grande sea la máscara se consigue una mayor reducción del ruido, pero a cambio los borde se difuminan más.

En la Figura 10 se presenta la imagen filtrada una vez definido los parámetros adecuados para el proyecto.

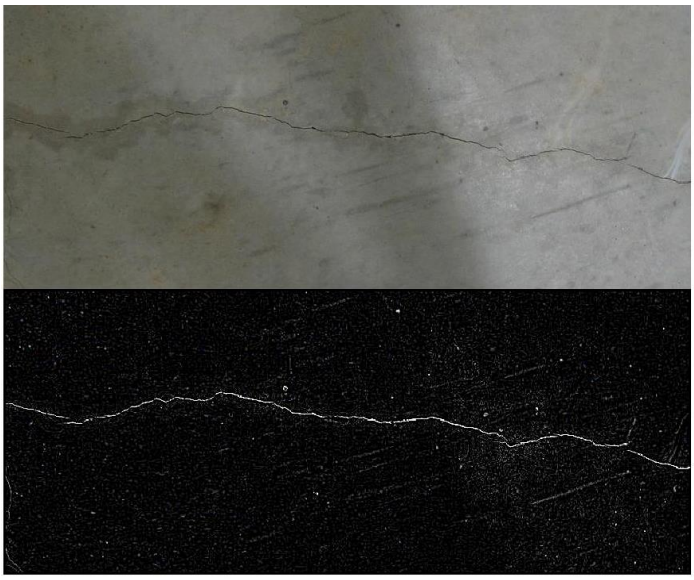


Figura 10 Imagen RGB vs imagen filtrada. Fuente: elaboración propia.

3.4.2 Eliminación del ruido

A menudo las imágenes están afectadas por ruido, lo que puede afectar negativamente la precisión de la clasificación o cálculo de variables métricas. Algunos de los ruidos comúnmente encontrados son: el gaussiano, en el que todos los píxeles que componen la imagen cambian su valor en base a una distribución normal o gaussiana. El efecto de este ruido es que en general se generan valores aleatorios que variarán respecto del valor original del píxel.

Sal y pimienta, es un ruido impulsivo, donde los valores que toma el píxel son muy altos o muy bajos. El efecto final de este ruido es que ciertos píxeles de forma aleatoria cambian a el valor máximo (sal=blanco) o el mínimo (pimienta=negro), tal como se muestra en la Figura 11.

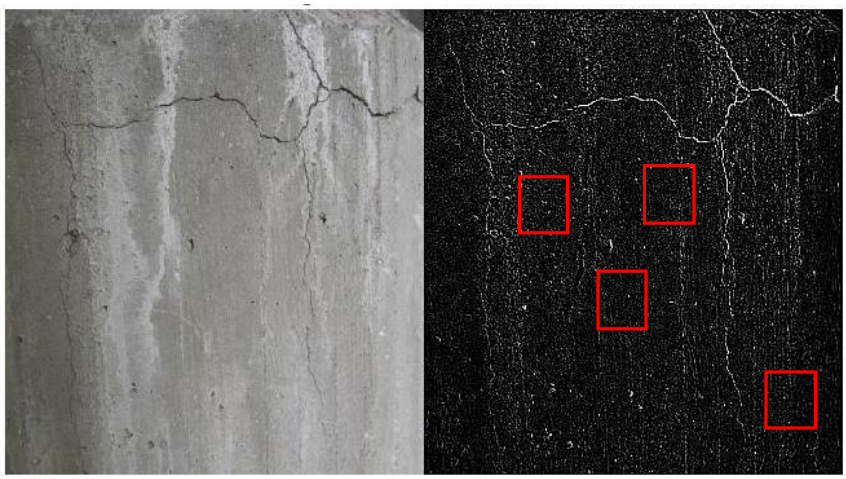


Figura 11 Imagen con ruido. Fuente: elaboración propia.

En Matlab, mediante la aplicación de la función “imnoise” se puede simular los efectos de algunos de los problemas enumerados anteriormente. Por ejemplo, filtrado lineal permite eliminar ruidos como los promediadores o los gaussianos (Figura 12), en el que cada píxel se ajusta a la media de los píxeles de su entorno, se reducen las variaciones locales causadas por el grano.

Otra de las funciones es “wiener2” que filtra la imagen en escala de grises I utilizando un filtro de Wiener adaptativo de paso bajo a nivel de píxel basado en la media local y la varianza de un entorno local de cada píxel (Lim, 1990). Definido por:

$$J = wiener2(I, [m\ n], noise)$$

Donde I es la imagen de entrada y $[m\ n]$ es el tamaño del entorno especificado como vector de 2 elementos, donde m es el número de filas y n es el número de columnas (MATLAB, 2010).

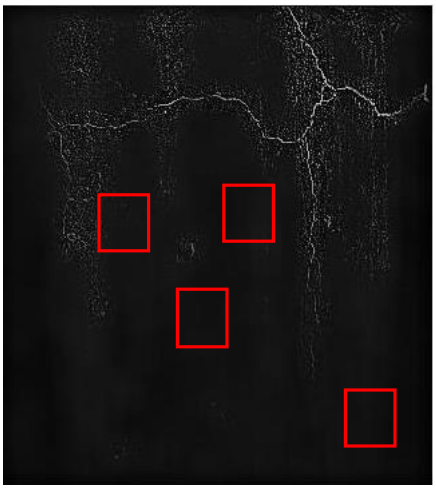


Figura 12 Eliminación del ruido. Fuente: elaboración propia.

3.4.3 Binarización de imágenes

Las imágenes binarias son matrices de pixeles con solo dos valores, 0 o 1, representando objetos (1) y el fondo (0). La función "im2bw" convierte una imagen en escala de grises a binaria, asignando 1 (blanco) a los pixeles con una luminancia mayor que un nivel dado y 0 (negro) a los demás. Se realiza primero la conversión a escala de grises utilizando ind2gray o rgb2gray y luego a escala de grises binaria usando umbrales especificados.

Una vez binarizada la imagen se aplican procesos de post tratamiento, en el cual se hace un relleno de gaps y la eliminación de pequeñas áreas con el fin de tener una imagen los más limpia posible, para que cuando se obtengan los valores métricos sean lo más cercano a la realidad.

3.4.4 Segmentación y etiquetado.

Es el proceso en donde se separa la imagen en diferentes regiones, dependiendo de la similitud de los pixeles, de esa manera se distingue la fisura (objeto de interés) del resto de la imagen. En otras palabras, para la segmentación se diferencia únicamente dos clases: la primera, en torno al objeto de interés, mientras que la segunda corresponde al fondo de la imagen (Carela, 2019).

Este paso es necesario previo a la obtención de las variables métricas de la imagen. La función "bwlabel" permite etiquetar los componentes conectados de la imagen.

El etiquetado de objetos consiste en recorrer una imagen binaria con el objetivo de detectar objetos diferentes en la imagen. El valor de la etiqueta depende de la vecindad definida, y como se ha explicado previamente, dicha vecindad puede ser 4-vecinos u 8-vecinos, tal como se indica en la Tabla 6.

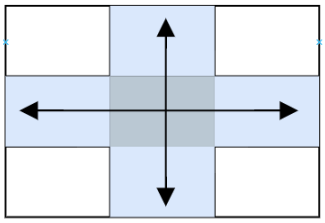
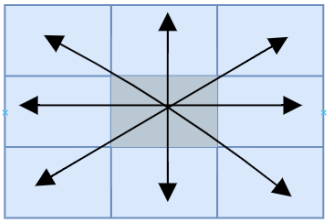
Número de Píxeles	Descripción	Ejemplo
4	Se consideran conectados si comparten bordes.	
8	Se consideran conectados si comparten bordes o vértices	

Tabla 6 Conectividad de píxeles.

3.4.5 Obtención de los valores métricos

Una vez realizado la segmentación y el etiquetado se obtienen la propiedades de la imagen con la función "*regionprops* ". Si específica "*all*" *regionprops* calcula todas las mediciones de la forma y del valor de los píxeles (MATLAB, 2010).

Como medición de forma se tiene la propiedad "Area" da el número real de píxeles en la región, devuelto como un escalar cuyo valor corresponde aproximadamente al número total de

pixeles activos de la imagen. Mientras que en la medición de valor de los píxeles se obtienen "PixelValues" que devuelve el número de píxeles en la región, como un vector de *p* por 1, donde *p* es el número de píxeles de la región. Cada elemento del vector contiene el valor de un píxel de la región. Para transformar el área en pixeles a área en unidades métricas es necesario conocer el área del pixel en unidades métricas y multiplicar con el número total de pixeles. (Fiter, 2012).

3.5 Internet de las cosas (IoT)

El IoT es una arquitectura emergente basada en la Internet global que facilita el intercambio de bienes y servicios entre redes de la cadena de suministro y que tiene un impacto importante en la seguridad y privacidad de los actores involucrados (Weber, 2010). Esto representa la próxima evolución de Internet, que será un enorme salto en su capacidad para reunir, analizar y distribuir datos que se pueden convertir en información y conocimiento (Evans, 2011).

En la actualidad se aplica Internet de las Cosas en áreas de salud, construcciones, tráfico vehicular, agricultura, educación, visión artificial, conservación del ambiente, meteorología, entre otros (Zanella et al., 2014)

3.5.1 Implementación en Thingspeak

ThingSpeak es un servicio de plataforma de análisis de IoT de MathWorks, los creadores de Matlab y Simulink. Permite agregar, analizar y visualizar datos en tiempo real en la nube desde dispotivos IoT conectados, como se observa en la Figura 13. Es compatible con dispositivos como Arduino, Raspberry Pi y el Módulo WiFi ESP8266, entre otros. Se puede ejecutar código Matlab en ThingSpeak para realizar análisis y procesamiento de datos en tiempo real, y es posible crear sistemas de IoT sin configurar servidores ni software web (Olivar Ruiz, 2021).

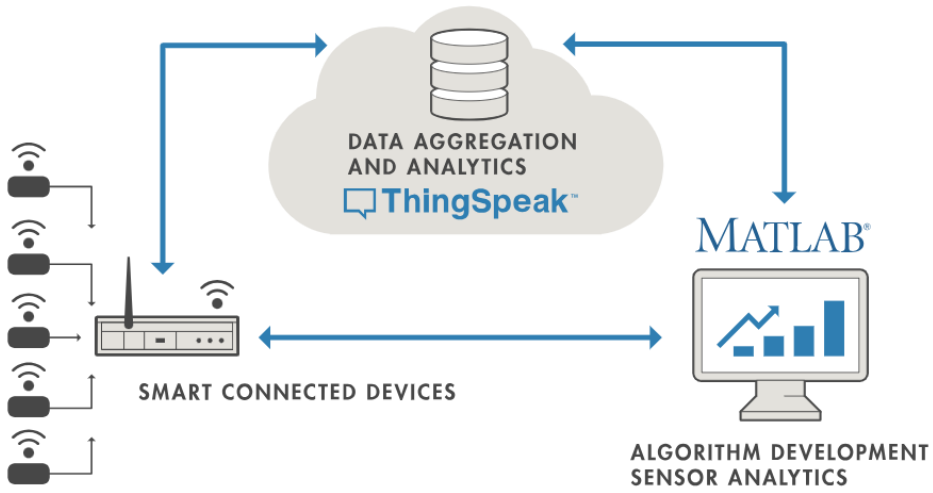


Figura 13 Funcionamiento de ThingSpeak (MathWorks, 2020)

El principal servicio que ofrece Thigspeak es una serie de servidores a los que se puede acceder para consultar información que les es transmitida desde una fuente distinta. Permite la creación de canales con un identificador único y una serie de claves de lectura y escritura.

Es posible enviar datos a ThingSpeak desde los dispositivos mediante API MQTT o REST. Se puede acceder a visualizaciones instantáneas de sus datos en directo desde cualquier navegador web conectado a Internet. Una de sus principales ventajas es su capacidad de ejecutar código en MATLAB para realizar análisis en línea y procesar datos a medida que ingresan.

Los datos que se envían desde el dispositivo a la plataforma se almacenen en canales, cada canal almacena hasta 8 campos de datos, y se pueden crear tantos canales como sea necesario. Cada canal está formado por los siguientes campos: 8 campos para almacenamiento de datos, por ejemplo, datos enviados por un sensor, 3 campos para el almacenamiento de información, para guardar latitud, longitud y elevación y 1 campo “estado”, para describir la información almacenada en el canal.

ThingSpeak se usa a menudo para la creación de prototipos y sistemas IoT de prueba de concepto que requieren análisis. Con Matlab integrado en la plataforma, es posible realizar calibraciones, análisis, transformación de datos e incluso crear gráficos personalizados para visualizar dichos datos, aunque por defecto ofrece ya diferentes posibilidades para ello. También cuenta con una serie de indicadores de alarma o displays numéricos (Olivar Ruiz, 2021).

La transferencia de los datos a la nube desde el ordenador se realiza a través de esta aplicación. Además, se crea un entorno que permite el análisis y la visualización de los datos, así como la creación de una alerta que indique cuando el proceso de clasificación se ha terminado y envié los resultados por correo electrónico.

3.5.1.1 Creación del canal

Para el presente trabajo se crea un canal, denominado "Detección de Fisuras" (Figura 14) con 5 campos: el primero se refiere a si la imagen detecta una fisura asignando 1 (con fisura) y 0 (sin fisura), de este campo depende el valor de los demás, si la imagen no presenta fisura, los valores en los siguientes campos serán nulos, el segundo campo contiene el número de fisuras en la imagen, el campo 3 tiene el área de las fisuras, el campo 4 tiene el número de imágenes con fisura y el ultimo campo detalla el número de imágenes sin fisura.

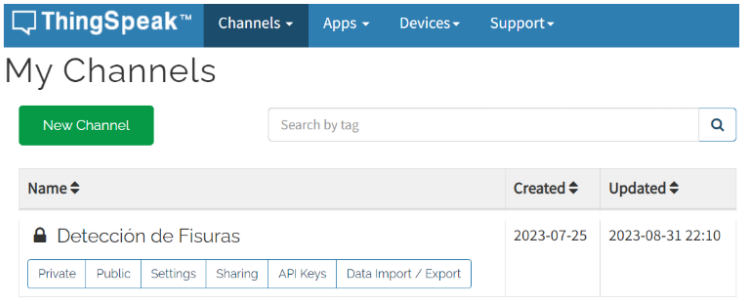


Figura 14 Creación del canal: Detección de Fisuras. Fuente: elaboración propia.

El canal "Detección de Fisuras" es en el que se van a registrar los datos que se obtengan del análisis de las imágenes de las estructuras de hormigón en la Figura 15 se muestra la configuración del canal.

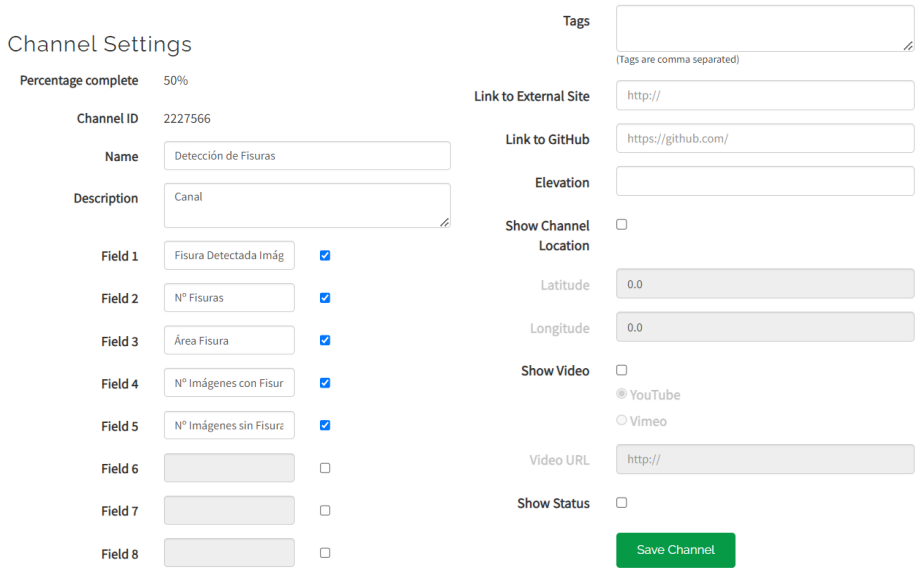


Figura 15 Configuración del canal "Detección de Fisuras" en ThingSpeak. Fuente: elaboración propia.

Una vez creado el canal, la propia plataforma genera de manera automática el identificador del canal (Chanel ID), y las claves (API Keys) del canal. Esta información es esencial para conectar con los procesos que se realizan en Matlab, es decir, de esta manera se carga la información en la nube del procesamiento y clasificación de las imágenes.

```
%% Datos del canal:
ChannelIDFisuras = 2227566;
readAPIKeyFisuras = '9U91DTUCZJ2LGNQG';
writeAPIKeyFisuras = '2EPJ3ZEPKYRNPCIF';

%% Inicio captura de datos durante el tiempo de espera.
FisuraDetectada = 0;
NumeroFisuras = 0;
AreaFisura = 0;
NumImgFisura = 0;
NumImgSFisura = 0;
thingSpeakWrite(ChannelIDFisuras,[FisuraDetectada,NumeroFisuras,AreaFisura,
```

Figura 16 Carga de datos en ThingSpeak. Fuente: elaboración propia.

En la Figura 16 se observa la conexión directa del canal con el código. Además, es importante mencionar que el tiempo mínimo de envió de datos en la plataforma es de 15 segundos.

3.5.1.2 Análisis de datos para el envió de notificaciones

ThingSpeak permite enviar notificaciones o alertas a través de mensajes al correo electrónico. Para acceder a esta función es necesario crear una aplicación utilizando MATLAB Analysis (Figura 17). Aquí se analizan los datos para el envío de los datos al correo electrónico.

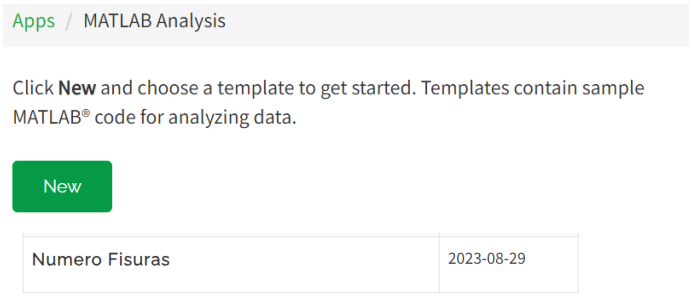


Figura 17 Creación del MATLAB Analysis. Fuente: elaboración propia.

Para establecer el momento especifico en el que se realizará la clasificación y análisis de las imágenes se utiliza la aplicación TimeControl.

Este proceso se ha configurado en frecuencia recurrente para que la clasificación y el análisis de las imágenes se ejecute semanalmente los días lunes a las 8 am. Estos parámetros se especifican en la Figura 18. Una vez creada la alerta se enlaza al MATLAB Analysis.

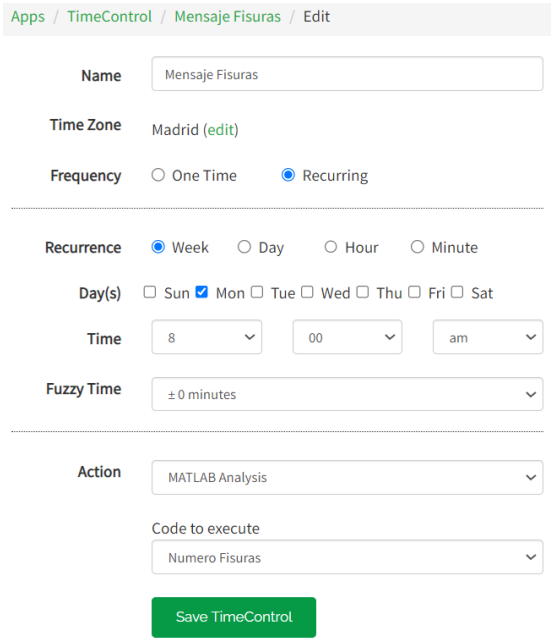


Figura 18 configuración de parámetros de TimeControl. Fuente: elaboración propia.

En el código presentado en el Apéndice B se detalla los parámetros de configuración establecidos en el MATLAB Analysis. Los campos que se utilizan es el 4 y 5, se establece el asunto del correo, en este caso “Número Imágenes con Fisura”. Se utiliza una sentencia try-catch para manejar posibles errores al enviar un mensaje. Si no se producen errores, el mensaje se envía utilizando un URL (AlertUrl) y una clave de alerta (alertApikey) proporcionados por la plataforma IoT, lo que permite el acceso a la página de correo electrónico.

Capítulo 4 - Resultados

4.1 Base de datos

Para la elección del dataset utilizado para el entrenamiento de la red se basó el criterio en la matriz de confusión y el valor de la exactitud obtenido en cada proceso, dado como resultado los valores que se muestran en la Tabla 7. La matriz de confusión es una matriz bidimensional de tamaño $n\ x\ n$ donde las filas representan las categorías test obtenidas de la cobertura real y las columnas están dadas por las clases obtenidas de la clasificación; con el objetivo de identificar el grado de exactitud global y entre categorías, a través del cálculo de los errores de omisión (píxeles que el usuario asignó y no coinciden con la asignación resultante del clasificador) y de comisión (píxeles que el clasificador considera pertenecen a una clase y no fue considerado por el usuario) (Chuvieco, 1995).

SDNET2018			
	Con Fisura	**Sin Fisura**	Total
Con Fisura	3291	2648	5939
Sin Fisura	208	5731	5939
Total	3499	8379	11878
Exactitud	94%	68%	

KaggleDataset			
	Con Fisura	**Sin Fisura**	Total
Con Fisura	13990	10	14000
Sin Fisura	13	13987	14000
Total	14003	13997	28000
Exactitud	99,9%	99,9%	

Tabla 7 Matriz de confusión

Al ser 2 bases de datos elaboradas con el objetivo del entrenamiento de una red en la detección de fisuras es predecible tener valores muy buenos en el test. Sin embargo, el entrenamiento con SDNET2018 presenta valores de omisión muy altos, esto se debe a que las imágenes son de diferentes materiales, además cuentan con varios ruidos en las fotos, diferentes objetos que dificultan la visualización de la fisura, caso contrario al de KaggleDataset, que cuenta con imágenes únicamente de hormigón y muy centradas en la fisura de la estructura.

Al combinar estas dos bases de datos se logra mejorar los resultados de SDNET2018 (Tabla 8)

KaggleSDNET			
	Con Fisura	**Sin Fisura**	Total
Con Fisura	17188	2751	19939
Sin Fisura	448	19491	19939
Total	17636	22242	39878
Exactitud	86%	98%	

Tabla 8 Matriz de confusión bases combinadas

Como se menciona en la metodología, las imágenes de utilizadas para clasificar y analizar las características de la fisura fueron proporcionadas por la empresa CEDEX. Estas imágenes tomadas en campo dieron los resultados de la Tabla 9.

SDNET2018			
	Con Fisura	**Sin Fisura**	Total
Con Fisura	2	1	3
Sin Fisura	0	1	1
Total	2	2	4
Exactitud	50%		

KaggleDataset			
	Con Fisura	**Sin Fisura**	Total
Con Fisura	3	0	3
Sin Fisura	0	1	1
Total	3	1	4
Exactitud	100%		

KaggleSDNET			
	Con Fisura	**Sin Fisura**	Total
Con Fisura	4	0	4
Sin Fisura	0	0	0
Total	4	0	4
Exactitud	75%		

Tabla 9 Porcentaje de clasificación de imágenes proporcionadas por la empresa. Fuente: elaboración propia.

La red entrenada con la base de datos Kaggle (Özgenel, 2019) clasifica las imágenes con una exactitud del 100%. Por lo tanto, esta es la red que se implementa en el código del proyecto.

Aunque la empresa proporcionó varias imágenes relacionadas con el proyecto, la mayoría de ella no cumplían los requisitos mencionados en el punto 3.4. Sin embargo, estas imágenes sirven para ejemplificar los errores más comunes en la captura de las fotografías Figura 19.

Problemas en la iluminación de la imagen

Dificultad para diferenciar entre fisura y fondo

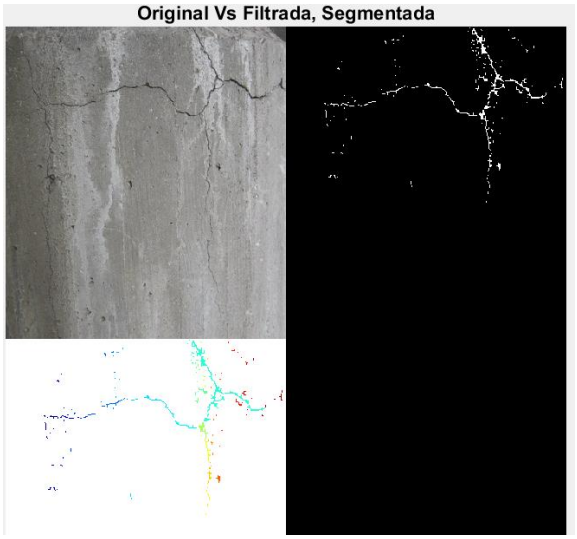
Muchos elementos en la imagen

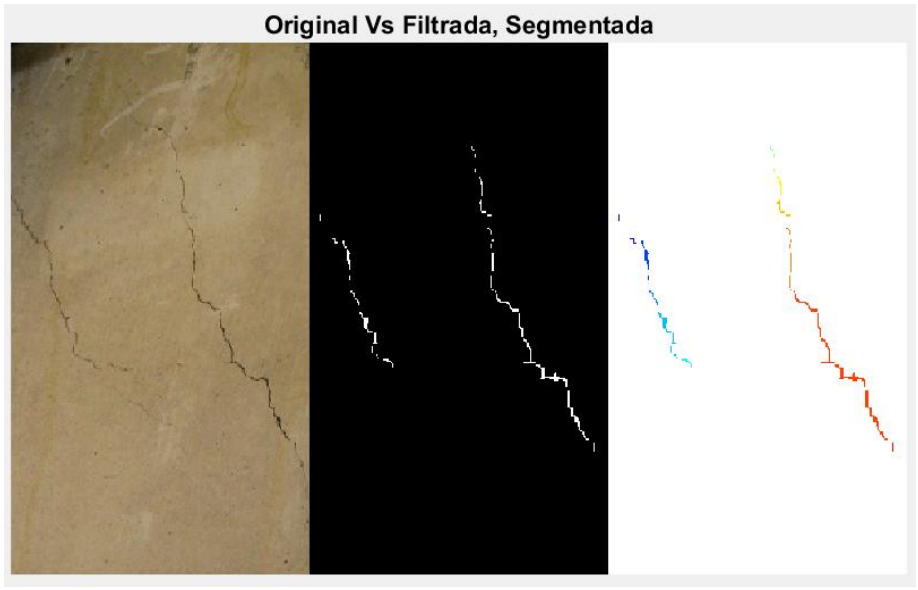
Distorsión de la imagen

Figura 19 Imágenes con problemas y errores de captura. Fuente: elaboración propia.

4.2 Detección de la fisura.

La imagen final que se obtiene después del proceso de filtrado, binarizado, segmentación y etiquetado se presenta en la Figura 20.





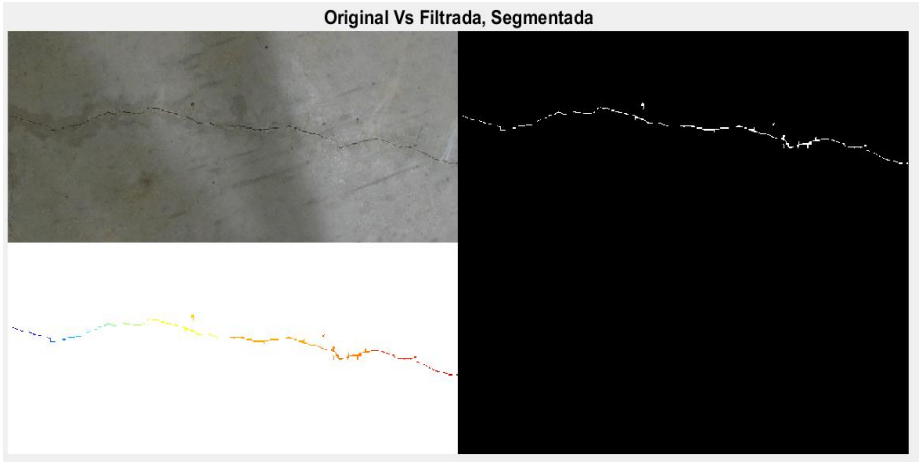


Figura 20 Imágenes procesadas. Fuente: elaboración propia.

Las imágenes se visualizan una vez que el código en Matlab es ejecutado. Las imágenes se van mostrando en orden, se imprime la imagen original, seguido de la imagen binarizado sin ruido y la imagen con las fisuras segmentadas.

El resultado del proceso de eliminación de ruido es aceptable, se logra una visualización de la fisura (en color blanco) distinguible del fondo de la imagen. Para la segmentación de las fisuras se pintan de colores distintos cada uno de los trozos de las misma.

Es importante mencionar que en algunos tramos las fisuras pierden continuidad, esto es debido a que, en estas zonas, los valores de los pixeles de las fisuras son semejantes a los del fondo, generándose una pérdida de la información. Otra posible causa se produce en la eliminación del ruido, en este proceso se eliminan pequeños trozos de la fisura.

4.3 Visualización de los datos en el canal ThingSpeak

Cabe mencionar que el canal puede gestionarse de manera privada y pública, para visualizar los datos sobre las fisuras enviadas desde el código ejecutado en MATLAB. En el canal Detección de Fisuras se añaden visualizadores (pestaña Add Visualizations) que permiten representar los datos obtenidos de manera gráfica.

En la Figura 21 se presentan el número de imágenes que se clasifican como "Con Fisura" y la fecha en la que se obtuvo la clasificación, además se incluye un indicador de color verde que indica que se ha detectado una fisura en alguna de las imágenes.

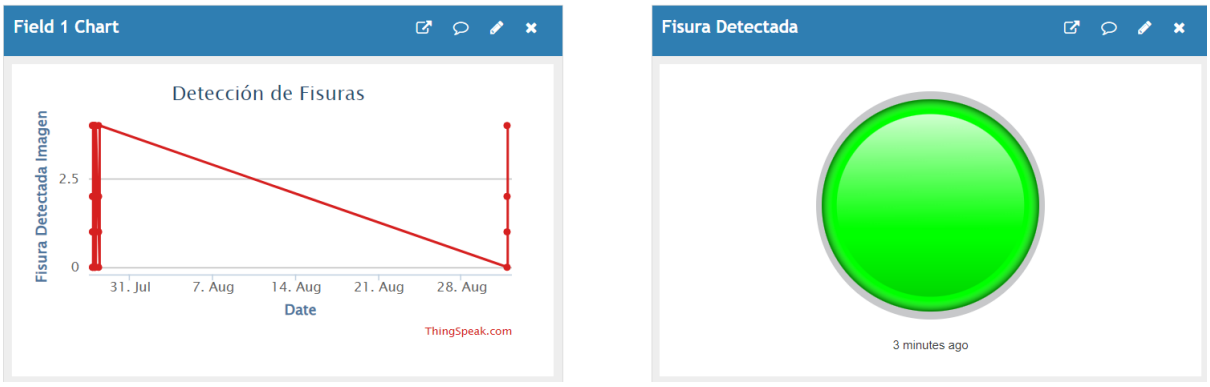


Figura 21 Gráfico e indicador del canal "Detección de Fisuras". Fuente: elaboración propia.

En la Figura 22 se grafica el número de fisuras y el área que se calculan de cada una de las imágenes que se han etiquetado anteriormente como imagen "Con Fisura", estos mismos datos se almacenan en un documento Excel que puede descargarse desde el canal de

ThingSpeak. Además, el código en Matlab incorpora la generación de un documento en el que se almacenan los datos del proceso, los valores obtenido (ver la Tabla 10). Además, la plataforma almacena los datos en la nube cada vez que se ejecuta el código, permitiendo tener un registro histórico de las imágenes analizadas. Estos valores se presentan en la Tabla 11.

Nº de imagen	Nº de fisura (píxeles)	Área de fisura (píxeles)
1	114	4934
2	0	0
3	14	710
4	22	133

Tabla 10 Datos exportados de la ejecución del código.

Fecha	ID	Nº de imagen	Nº de fisuras	Área	Nº imágenes con fisura	Nº de imágenes sin fisura
2023-08-31T22:06:43+02:00	1	0	0	0		
2023-08-31T22:07:01+02:00	2	1	114	4934	1	0
2023-08-31T22:07:18+02:00	3	3	14	710	2	1
2023-08-31T22:07:35+02:00	4	4	22	1331	3	1
2023-08-31T22:10:38+02:00	5	0	0	0		
2023-08-31T22:10:56+02:00	6	1	114	4934	1	0
2023-08-31T22:11:13+02:00	7	3	14	710	2	1
2023-08-31T22:11:30+02:00	8	4	22	1331	3	1
2023-09-04T03:07:40+02:00	9	0	0	0	0	0
2023-09-04T03:07:58+02:00	10	1	114	4934	1	0
2023-09-04T03:08:15+02:00	11	3	14	710	2	1
2023-09-04T03:08:32+02:00	12	4	22	1331	3	1

Tabla 11 Valores descargados de ThingSpeak

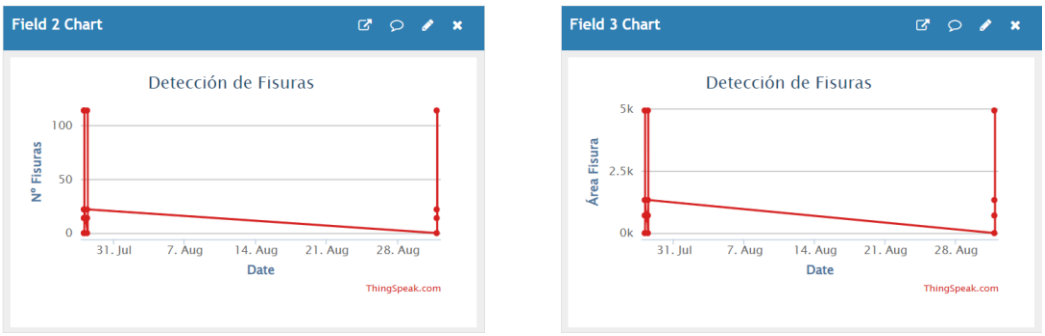


Figura 22 Gráfico de número de fisura y área de fisuras. Fuente: elaboración propia.

Finalmente se ha añadido 2 gráficas en las que se muestra el número total de imágenes con fisura e imágenes sin fisura (Figura 23). Estas gráficas permiten controlar el número total de imágenes que han entrado en la clasificación y de qué manera se han etiquetado en el proceso.

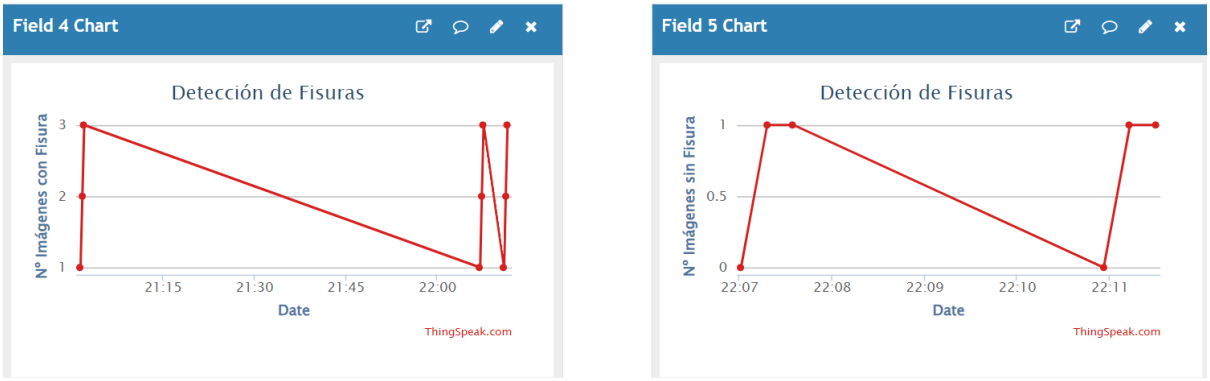


Figura 23 Número de imágenes con fisura y sin fisura. Fuente: elaboración propia.

4.4 Envió del email desde ThingSpeak

En este punto del trabajo se obtiene el resultado final de todo el proceso, en donde en el correo especificado se recibirá un mensaje de alerta desde la plataforma de ThingSpeak. Tal como se visualiza en la Figura 24 los datos que se reciben en el correo son el número total de imágenes clasificadas "Con Fisura" y el número total de imágenes clasificadas "Sin Fisura" que en este caso corresponden al campo 4 y 5 respectivamente.

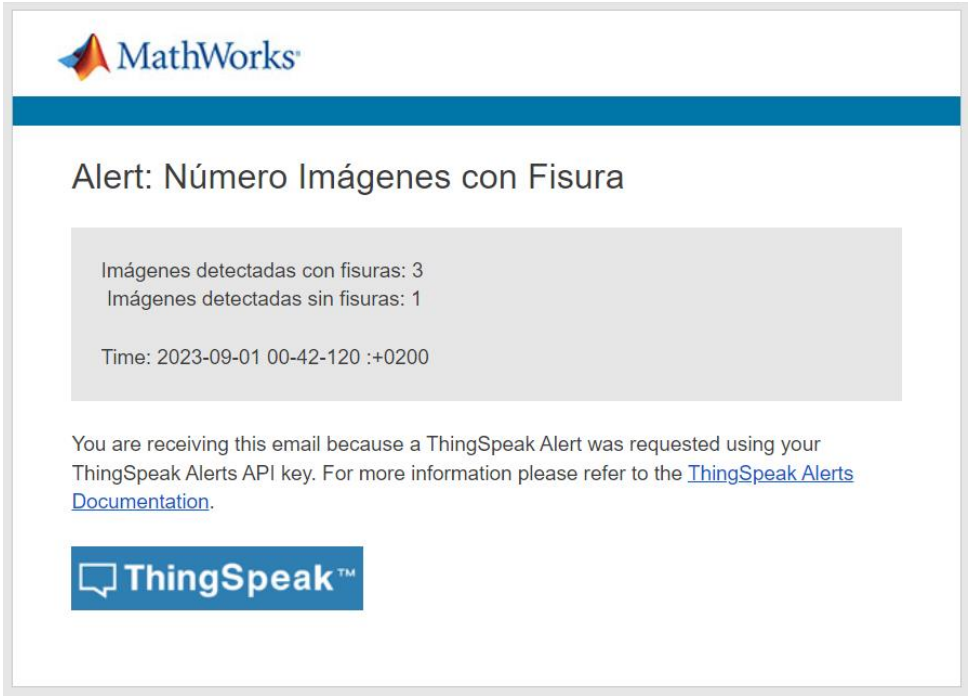


Figura 24 - Email de alerta recibido al correo electrónico. Fuente: elaboración propia.

Capítulo 5 - Conclusiones

La recopilación y análisis de información bibliográfica juega un papel importante en la realización de proyectos de IoT. Es esencial comprender algunos conceptos antes de empezar con el desarrollo del trabajo, tener en cuenta las nuevas metodologías, herramientas, códigos y procesos que se implementan en el entorno del IoT y que actualmente están en el mercado.

Es importante disponer de un conjunto de imágenes de entrenamiento amplio y representativo del problema para obtener resultados satisfactorios con una red neuronal convolucional.

Es evidente que la cantidad de las imágenes utilizadas para la clasificación no representa un número considerable respecto a la muestra inicial de 134 imágenes. La mayoría de las imágenes no se ajustan a los parámetros especificados en el apartado 3.4. La resolución espacial y espectral de las imágenes son factores determinantes en la clasificación semiautomática de fisuras.

La CNN ResNet obtuvo muy buenos resultados en su fase de entrenamiento y validación llegando al 99% utilizando la base de datos *Concrete Crack Images for Classification*.

Respecto al tratamiento de las imágenes, los resultados obtenidos fueron satisfactorios, permitiendo obtener datos sobre el número de fisuras y el área en píxeles. Para determinar el grado de exactitud en el cálculo de los valores, se requiere contar con información de campo, referido esto a la medición real de las fisuras. Esto permitiría comparar los valores obtenidos en la segmentación versus los valores medidos de las fisuras.

Matlab a través de ThingSpeak incluye herramientas que permiten visualizar los datos de las imágenes; el número de fisura, el área calculada en píxeles y el número total de imágenes clasificadas en cada categoría enviados desde el dispositivo, procesarlos y generar alertas.

ThingSpeak que es capaz de contribuir con servicios en la rama de la construcción con el fin de ayudar a los ingenieros y profesionales de la construcción a evaluar el estado de las estructuras y tomar medidas preventivas o de reparación.

Capítulo 6 - Trabajo a futuro

Teniendo en cuenta que el mayor de los problemas se presentó al clasificar las imágenes tomadas en campo, es importante establece requisitos mínimos para la captura de la información.

Dado que el sistema propuesto transmite información mediante wifi, se espera una conexión estable a internet, de igual manera la instrumentación para la medición en campo. Paglinawan et al. (2019) presenta un equipamiento para la recolección de datos, donde menciona que utilizó un Raspberry Pi 3, que se encargó de realizar el tratamiento de imágenes. La fuente de alimentación es una batería externa portable; al Raspberry Pi 3 se le conectó un sensor ultrasónico y una cámara, las mismas se encargaron de medir la distancia a la cámara y tomar la imagen respectivamente.

Las especificaciones de la cámara son: el módulo de cámara v2 tiene un sensor Sony IMX219 de 8 megapíxeles, se puede utilizar para tomar videos de alta definición, así como fotografías, es compatible con los modos de video 1080p30, 720p60 y VGA90. El sensor de distancia es: un sensor ultrasónico de distancia HC-SR04 que proporciona una funcionalidad de medición sin contacto de 2 cm a 400 cm con una precisión de rango que puede alcanzar hasta 3 mm, solo hay cuatro pines en el HC-SR04: VCC (alimentación), activación, eco (recepción) y GND (conexión a tierra).

En función del software/plataforma se podría utilizar otro que ofrezca mayores ventajas, actualmente existen en el mercado programas diseñados exclusivamente para implementar aplicaciones IoT, por lo que ya cuentan con los aspectos necesarios para obtener una aplicación más profesional, aunque por un precio más elevado.

Por otro lado, se obtendrían mejores resultados si se cumplen los requisitos establecidos en el apartado 3.4, la resolución espacial y espectral de las imágenes son factores determinantes en la investigación de clasificación automática de fisuras.

Se puede mejorar la calidad de resultados, comparado una imagen con otra, para evaluar si la fisura ha cambiado o ha evolucionado. Para esto es necesario tener un registro histórico de las fisuras y analizarlas temporalmente, teniendo un intervalo de tiempo, dependiendo de la estructura y el uso.

Chapter 7 Introduction

Concrete is currently the most widely used material for the development of any infrastructure. The technical preparation, from the design to the final stages of construction, must comply with the regulations and specifications specific to the material, and together with modern technologies, safe, durable and aesthetic works are obtained. However, these structures are often affected by cracks and fissures (Toirac Corral, 2004).

Concrete structures are subject to various factors that can lead to the formation and propagation of cracks (Toirac Corral, 2004), compromising structural integrity, affecting aesthetics and causing water seepage, which in turn could accelerate degradation. Other aspects to be taken into account in the propagation of cracks are: the quality of the concrete, the structural design, the applied loads, the ambient temperature and humidity, as well as the construction and maintenance processes. For this reason, there is a need to develop and/or validate methods to inspect the condition of the structure (Ercolani, Ortega, & Felix, 2015).

Colpari (2020) points out that artificial intelligence currently plays a very important role in Industry 4.0, where intelligent systems and technologies are used to create a connection between physical and virtual jobs (Darko et al., 2020). One part of artificial intelligence is computer vision, which focuses on extracting information about a scene by analysing images of that scene (Rosenfeld, 1998)

Automatic detection of cracks in concrete using images is a field of research and development that combines computer vision and civil engineering (Mohan & Poobal, 2018). The aim is to develop algorithms and systems that can automatically identify and analyse cracks in concrete structures from digital images, which can help engineers and construction professionals to assess the condition of structures and take preventive or repair measures (Paglinawan et al., 2018).

In this context, this work arises from the existing need in the company CEDEX (Centro de estudios y experimentación de Obras Públicas), belonging to the Ministry of Transport, Mobility and Urban Agenda, to apply innovative methodologies that allow to automatically identify and compare crack patterns in concrete constructions from RGB images, concrete becomes the material of study, as it presents a high density of cracks in its structure.

Chapter 8 Conclusions and future work

The collection and analysis of bibliographic information plays an important role in the realisation of IoT projects. It is essential to understand some concepts before starting with the development of the work, taking into account the new methodologies, tools, codes and processes that are implemented in the IoT environment and that are currently on the market.

It is important to have a large and representative set of training images of the problem in order to obtain satisfactory results with a convolutional neural network.

It is evident that the number of images used for classification does not represent a considerable number compared to the initial sample of 134 images. Most of the images do not conform to the parameters specified in section 3.4. The spatial and spectral resolution of the images are determining factors in the semi-automatic crack classification.

CNN ResNet obtained very good results in its training and validation phase, reaching 99% using the Concrete Crack Images for Classification database.

With regard to image processing, the results obtained were satisfactory, allowing data to be obtained on the number of cracks and the area. To determine the degree of accuracy in the calculation of the values, it is necessary to have information from the field, referring to the actual measurement of the cracks. This would allow a comparison of the values obtained in the segmentation versus the measured values of the cracks.

Matlab through ThingSpeak includes tools to visualise the image data; the crack number, the calculated area in pixels and the total number of images classified in each category sent from the device, process them and generate alerts.

ThingSpeak is able to contribute to services in the construction branch in order to provide a large number of applications, from taking and capturing photos to processing and sending the information.

This project gives rise to future IoT research in the area of construction, it is a less expensive, efficient, effective and more complete form of a system for detecting and monitoring cracks in concrete structures.

Future work

Bearing in mind that the biggest problem arose when classifying the images taken in the field, it is important to establish minimum requirements for capturing the information.

Since the proposed system transmits information via wifi, a stable internet connection is expected, as is the instrumentation for field measurement. Paglinawan et al. (2019) presents equipment for data collection, where he mentions that he used a Raspberry Pi 3, which was responsible for performing the image processing. The power source is a portable external battery; an ultrasonic sensor and a camera were connected to the Raspberry Pi 3, which were in charge of measuring the distance to the camera and taking the image respectively.

The camera specifications are: the v2 camera module has an 8 megapixel Sony IMX219 sensor, it can be used to take high definition videos as well as photos, it supports 1080p30, 720p60 and VGA90 video modes. The distance sensor is: an HC-SR04 ultrasonic distance sensor that provides non-contact measurement functionality from 2cm to 400cm with a range accuracy that can reach up to 3mm, there are only four pins on the HC-SR04: VCC (power), trigger, echo (receive) and GND (ground).

Depending on the software/platform, another one could be used that offers greater advantages; there are currently programmes on the market designed exclusively to implement IoT applications, so they already have the necessary aspects to obtain a more professional application, albeit at a higher price.

On the other hand, better results would be obtained if the requirements established in section 3.4 are met; the spatial and spectral resolution of the images are determining factors in automatic crack classification research.

The quality of results can be improved by comparing one image with another to assess whether the crack has changed or evolved. For this it is necessary to have a historical record of the cracks and to analyse them temporally, having a time interval, depending on the structure and the use.

Bibliografía

Águila Martínez, J. (2017). *Aprendizaje supervisado en conjunto de datos no balanceados con redes neuronales artificiales.* Barcelona: Universidad Oberta de Catalunya.

Aldalur, B., & Santamaría, M. (2002). Realce de imágenes: filtrado espacial. *Revista de teledetección, 17*(2), 31-42.

Ali, L., Alnajjar, F. S., Gochoo, M., Jassmi, H. A., Khan, W., & Serhan, M. A. (2021). Performance Evaluation of Deep CNN-Based Crack Detection and Localization Techniques for Concrete Structures. *Sensors*, 1-22.

Artola Moreno, Á. (2019). *Clasificación de imágenes usando redes neuronales convolucionales en Python.* Universidad de Sevilla: Dpto. de Teoría de la Señal y Comunicaciones.

Bonilla Carrión, C. (2020). *Redes convolucionales.* Sevilla: Universidad de Sevilla.

Bonilla, M., & Puertas, R. (1997). *Análisis de las redes neuronales: aplicación a problemas de predicción y clasificación.* Valencia: Servei de Publicacions: Universitat de València.

Calderón, L. S., & Bairán, J. (2020). Detección y medición semiautomática de fisuras en elementos de hormigón en fotos digitales usando procesamiento de imágenes. *Hormigon y acero*, 21-27.

Carela, P. A. (2019). *Tratamiento digital de imágenes en matlab para la detección de fallas.* Cataluña: Universidad Politecnica de Cataluña.

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

Chuvieco, E. (1995). *Fundamentos de teledetección espacial.* Rialp.

Colpari, M. H. (2020). *Uso de inteligencia artificial para la detección automatizada de fisuras en estructuras de hormigón armado.* Santiago: Escuela de Construcción Civil.

Dung, C. V., & Anh, L. D. (2019). Autonomous concrete crack detection using deep fully convolutional neural network. *Automation in Construction*, 52-58.

Ercolani, G. D., Ortega, N. F., & Felix, D. H. (2015). Detección de fisuras en vigas de hormigón pretensado. *Biotecnología, Nanotecnología,Bioingeniería y Materiales*, 90-97.

Evans, D. (2011). Internet de las cosas. Cómo la próxima evolución de Internet lo cambia todo. *Cisco Internet Bussiness Solutions Group-IBSG, 1*(11), 4-11.

Fiter, E. L. (2012). *Descripción, comparación y ejemplos de uso de las funciones de la toolbox de procesado digital de imágenes de MATLAB.* Madrid: Universidad Politécnica de Madrid.

Gallo Sánchez, M. B. (2013). *Segmentación digital de imágenes médicas para el mejoramiento del diagnóstico de anomalías en los Centros Radiológicos de la Ciudad de Ambato.* Ambato: Universidad Técnica de Ambato.

Gao, Y., & Mosalam, K. M. (2018). Deep Transfer Learning for Image-Based Structural Damage Recognition. *Computer-Aided Civil and Infrastructure Engineering*, 748-768. doi:https://doi.org/10.1111/mice.12363

He, K., Zhang, X., Ren, S., & J. Sun. (2015). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition.*, 770-778.

Hilera, J., & Martínez, V. (1995). *Redes neuronales artificiales: Fundamentos, modelos y aplicaciones.* Madrid: Ra-Ma.

Hoshyar, A. N., Samali, B., Liyanapathirana, R., Houshyar, A. N., & Yu, Y. (2019). Structural damage detection and localization using a hybrid method and artificial intelligence

techniques. *Structural Health Monitoring*, 150-169. doi:https://doi.org/10.1177/1475921719887768

Lim, J. S. (1990). Two-Dimensional Signal and Image Processing. *Englewood Cliffs, NJ, Prentice Hall*, 548.

Maguire, M., Dorafshan, S., & Thomas, R. J. (17 de Mayo de 2018). *SDNET2018: A concrete crack image dataset for machine learning applications*. Obtenido de Utah State University: https://digitalcommons.usu.edu/all_datasets/48/

MathWorks. (2020). *ThingSpeak*. Obtenido de ThingSpeak for IoT Projects: https://es.mathworks.com/products/thingspeak.html

MATLAB. (2010). Image Processing and Computer Vision. *version 9.9 (R2020b)*. Massachusetts: United States. Obtenido de version 9.9 (R2020b).

McCord Nelson, M., & Illingworth, W. (1991). *A practical guide to neural nets.* Massachusetts: Addison-Wesley.

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

Núñez Sánchez-Agustino, F. J. (2016). *Diseño de un sistema de reconocimiento automático de matrículas de vehículos mediante una red neuronal convolucional.* Barcelona: Universitat Oberta de Catalunya.

Olabe, X. B. (1998). *Redes neuronales artificiales y sus aplicaciones .* Bilbao: Publicaciones de la Escuela de Ingenieros.

Olivar Ruiz, P. (2021). *Diseño de un medidor iot doméstico.* Valladolid: universidad de valladolid.

Özgenel, Ç. F. (2019). Concrete Crack Images for Classification. *Mendeley Data*. doi: 10.17632/5y9wdsg2zt.2

Paglinawan, A. C., Cruz, F. R., Casi, N. D., Ingatan, P. A., Karganilla, A. B., & Moster, G. V. (2019). Crack Detection Using Multiple Image Processing for Unmanned Aerial Monitoring of Concrete Structure. *IEEE Region 10 Annual International Conference*, 2534-2538.

Palmer Pol, A., & Montaño Moreno, J. J. (1999). ¿Qué son las redes neuronales artificiales? Aplicaciones realizadas en el ámbito de las adicciones. *Adicciones, 11*(3), 243-255.

Pérez-Careta, E., Guzmán-Sepúlveda, J. R., Lozano-García, J. M., Torres-Cisneros, M., & Guzman-Cabrera, R. (2022). Clasificación de imágenes médicas mediante aprendizaje automático. *Ciencia de los ordenadores*, 35-38. doi:https://doi.org/10.6036/10117

Prasanna, P., Dana, K. J., Gucunski, N., Basily, B. B., La, H. M., Lim, R. S., & Parvardeh, H. (2016). Automated Crack Detection on Concrete Bridges. *IEEE Transactions on Automation Science and Engineering*, 591-599. doi:https://doi.org/10.1109/TASE.2014.2354314

Rivera, M. (Agosto de 2019). *La Red Residual (Residual Network, ResNet)*. Obtenido de http://personal.cimat.mx:8181/~mrivera/cursos/aprendizaje_profundo/resnet/resnet.html

Rosenfeld, A. B. (1998). Computer Vision: Basic Principles. *Proceedings of the IEEE*, 863-868.

Sada Pezonaga, S., & Sanz Delgado, J. A. (2015). *Tratamiento de imágenes con ruido impulsivo mediante reglas difusas y algoritmos genéticos.* Pamplona: Universidad Pública de Navarra.

Silva, W. R., & Lucena, D. S. (2018). Concrete Cracks Detection Based on Deep Learning Image Classification. *Proceedings*, 489. doi:https://doi.org/10.3390/ICEM18-05387

Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, 1-14.

Toirac Corral, J. (2004). Patología de la construcción grietas y fisuras en obras de hormigón ; origen y prevención. *Ciencia y Sociedad*, 72-114.

Varela-Arregoces, E., & Campbells, E. (2011). Redes Neuronales Artificiales: una revisión del estado del arte, aplicaciones y tendencias futuras. *Investigación y Desarrollo en TIC*, 18-27.

Vieco, J. (5 de Enero de 2018). *Cleverpy.* Obtenido de Red convolucional con PyTorch: https://cleverpy.com/2018/01/05/red-convolucional-pytorch/

Weber, R. H. (2010). Internet of Things - New Security and Privacy Challenges. *Computer Law & Security Review*, 23-30.

Zanella, A., Bui, N., Castellani, A., & Vangelista, L. Z. (2014). Internet of Things for Smart Cities. *IEEE INTERNET OF THINGS JOURNAL*, 22-32.

APÉNDICES

Apéndice A - Código en Matlab

Código Modelo CNN

```
%Lectura de las carpetas.
%outputFolder=fullfile('SDNET2018');
rootFolder=fullfile('KaggleDataset');

%Distincion de categorias.
categorias={'ConFisura','SinFisura'};

IMDS=imageDatastore(fullfile(rootFolder,categorias),'LabeLSource','foldernames');

%Conteo de archivos de cada categoria ('Con Fisura, Sin Fisura').
tbl=countEachLabel(IMDS);

%Averigua que categoria contiene el menor número de archivos.
minSetCount=min(tbl{:,2})

%Iguala el numero de archivos en cada categoria (carpeta), tomando como
%referencia el menor numero.
IMDS=splitEachLabel(IMDS,minSetCount,'randomize');
countEachLabel(IMDS);

%Leer las imagenes y etiquetar.
confisura=find(IMDS.Labels=='ConFisura',1);
sinfisura=find(IMDS.Labels=='SinFisura',1);

%Visualizacion de las imagenes.
figure
subplot(2,2,1);
imshow(readimage(IMDS,confisura));
title('Con fisura')
subplot(2,2,2);
imshow(readimage(IMDS,sinfisura));
title('Sin fisura')

%figure; montage({confisura,sinfisura},[])
%title('Con fisura Vs Sin fisura')

% Carga de ResNet-50
% Asignamos una variable (net) para almacenar la red.
net=resnet50();
figure
plot(net)
title('Arquitectura de ResNet-50')
%Dimenciona la figura de la arquitectura ResNet-50.
set(gca,'YLim',[150 170])

%Propiedades de la primera capa de entrada, avergua que tipo de entrada
%acepta.
```

```
net.Layers (1)
%Propiedades de la ultima capa (cantidad de clases de la red neuronal).
net.Layers(end)

%Otra forma de obtener la cantidad de clases de la red neuronal.
numel(net.Layers(end).ClassNames)

%Caracteristicas de la arquitectura ResNet-50.
analyzeNetwork(net)

%division para el entrenamiento y el test.
[trainingSet,testSet]=splitEachLabel(IMDS,0.7,'randomize');

%Obtener el tamaño requerido de la imagen de entrada.
imageSize=net.Layers(1).InputSize;

augmentedTrainingSet=augmentedImageDatastore(imageSize,trainingSet,'ColorPreprocessing','gray2r
gb');
augmentedTestSet=augmentedImageDatastore(imageSize,testSet,'ColorPreprocessing','gray2rgb');

wl=net.Layers(2).Weights;
wl=mat2gray(wl);

%Primera convolución.
figure
montage(wl)
title('Capa de pesos de la primera convolución')

%Entrenamiento.
featureLayer = 'fc1000';
trainingFeatures = activations(net,augmentedTrainingSet, featureLayer, 'MiniBatchSize', 32,
'OutputAs', 'columns');

trainingLables = trainingSet.Labels;
classifier = fitcecoc (trainingFeatures, trainingLables,'Learner', 'Linear', 'Coding',
'onevsall','ObservationsIn','columns');

%Validación (test).
testFeatures = activations(net,augmentedTestSet, featureLayer, 'MiniBatchSize', 32, 'OutputAs',
'columns');

predictLabels=predict(classifier,testFeatures,'ObservationsIn','columns');

%Matriz de confusión.
testLables=testSet.Labels
confusionmat(testLables,predictLabels);

%Exactitud
accuracy = sum(predictLabels == testLables)/numel(testLables)
```

Código clasificación de la imagen y datos de la fisura

```
%% Datos del canal:
ChannelIDFisuras = 2227566;
readAPIKeyFisuras = '9U91DTUCZJ2LGNQG';
```

```
writeAPIKeyFisuras = '2EPJ3ZEPKYRNPCIF';
%% Inicio captura de datos durante el tiempo de espera.
FisuraDetectada = 0;
NumeroFisuras = 0;
AreaFisura = 0;
NumImgFisura = 0;
NumImgSFisura = 0;
thingSpeakWrite(ChannelIDFisuras,[FisuraDetectada,NumeroFisuras,AreaFisura, NumImgFisura,NumI
mgSFisura],'Writekey',writeAPIKeyFisuras);
% Hay que esperar 15 segundos para dar tiempo a la escritura.
pause(2); %Solo se ademiten esperas de 2 segundos
pause(2); pause(2); pause(2); pause(2); pause(2); pause(2); pause(2);
%%Obtener ruta actual.
ruta=pwd;

%%Agregar carpetas y subcarpetas donde estan imagenes.
addpath(genpath('ModeloKaggle'));

%Lectura de las carpetas.
outputFolde=fullfile('FotosAnalisisImagen');
rootFolde=fullfile(outputFolde,'EdificacionObraCivil');

%Distinción de categorias.
categoria={'ArchivoSantaCruzdeTenerife','CARResudenciaBlume','CasaCulturaTenerife','CasonRetiro',
'CorreaCubiertaZaragoza','FrayBernardino','MuseoAduanaMalaga','MuseoRomanticismo','ParquedeBo
mberos','Almonte','RioEspana','Toledo','Valmojado'};

IMD=imageDatastore(fullfile(rootFolde,categoria),'LabeLSource','foldernames');

%conteo de archivos
tbl=countEachLabel(IMD)
%minCount=min(tbl{:,2})
%maxCount=max(tbl{:,2})
TotalFotos=sum(tbl{:,2})

%%Colocarnos en la carpeta donde están las imagenes
cd HormigonPruebas

%%El tipo de archivo que acepta es jpg.
%dirImagen1 = dir('**/*.jpg'); %Busca todos los archivos en la carpeta y subcarpeta
dirImagen1 = dir('*.jpg'); %Busca todos los archivos en la carpeta
numimagenes1= length(dirImagen1);
datos = cell(1, numimagenes1);

ContImg=0;
DatosCategoria='Categoria';
c=0;
ConF=0;
s=0;
A=0;
AreaT=0;
NumF = 0;
NumFisT = 0;

%% Detección y analisis de las imagenes con fisura y sin fisura.
for k = 1:numimagenes1
```

```
%%%Para imagenes colocar imread.
datos{1,k} = imread(dirImagen1(k).name);
ContImg=[ContImg;k];
Foto=k;
%figure(k);
%imshow(datos{1,k});

%%Ejecuta el modelo, clasifica las imagenes.
ds=augmentedImageDatastore(imageSize,datos{1,k},'ColorPreprocessing','gray2rgb');
imageFeatures = activations(net,ds, featureLayer, 'MiniBatchSize', 32, 'OutputAs', 'columns');
label=predict(classifier,imageFeatures,'ObservationsIn','columns');
sprintf('La imagen %d pertenece a la categoria %s',Foto,label)

DatosCategoria=[DatosCategoria;label];

%Conteo de imagenes ConFisura y Sin fisura.
if label == 'ConFisura'
c=c+1;
ConF=[ConF;k];
%Filtro imfilter.
h = fspecial('log',7,0.4);
I2 = imfilter(datos{1,k},h);
I2 = rgb2gray(I2); %función rgb2gray
figure (k); montage({datos{1,k},I2},[])
%title('Original Vs Filtro imfilter')
%title('Original Vs Escala de Grises')
%Filtro para suavizar el ruido.
I2 = wiener2(I2, [50 50]);
%figure (k); montage({datos{1,k},I2},[])
%title('Original Vs Filtro wiener2')

%Binarizacion de la imagen.
I2=im2bw(I2,0.40);
%figure (k); montage({datos{1,k},I2},[])
%title('Original Vs Función im2bw')
% Relleno de gaps.
se=strel('disk',3);
I2=imclose(I2,se);
%figure (k); montage({datos{1,k},I2},[])
%title('Original Vs Relleno de Gaps')
%Eliminacion de ruido (Elimina las partes que tengan mas de 130 megapixeles)
I2=bwareaopen(I2,6);
%figure (k); montage({datos{1,k},I2},[])
%title('Original Vs Filtro bwareaopen')

% Segmentacion.
F = bwlabel(I2);
%Propiedades de la fisura (datos de la segmentacion).
PropF = regionprops(F, 'all');
%Numero de fisuras de la imagen.
NumF = max(max(F));
NumFisT = [NumFisT;NumF];
disp('Número de Fisuras:'), disp(NumF)
%Area de las fisuras.
A = sum([PropF.Area])
AreaT = [AreaT;A];
```

```
%figure (k); montage({datos{1,k},label2rgb(F)},[])
%title('Original Vs Segmentación')
% Visualización de la imagen original, filtrada y segmentada.
figure (k); montage({datos{1,k},I2,label2rgb(F)},[])
title('Original Vs Filtrada, Segmentada')
%% Captura de datos de cada imagen.
FisuraDetectada = k;
NumeroFisuras = NumF;
AreaFisura = A;
NumImgFisura = c;
NumImgSFisura = s;
thingSpeakWrite(ChannelIDFisuras,[FisuraDetectada,NumeroFisuras,AreaFisura,NumImgFisura,Numl
mgSFisura],'Writekey',writeAPIKeyFisuras);
% Hay que esperar 15 segundos para dar tiempo a la escritura.
pause(2); %Sólo se ademiten esperas de 2 segundos
pause(2); pause(2); pause(2); pause(2); pause(2); pause(2); pause(2);
else
s=s+1;
end
end

%Tablas de datos.
TablaCategorias=table(ContImg,DatosCategoria);
TablaConteo=table(c,s);
TablaPropiedadesFisura=table(ConF,NumFisT,AreaT);

%%Regresar a ruta principal.
cd (ruta);

%%Exportar datos a exel
%xlswrite('DatosCategorias',ContImg,'Datos','B4');
%writetable(TablaCategorias,'DatosCategorias.xls','Sheet',2,'Range','B2');
%writetable(TablaConteo,'DatosCategorias.xls','Sheet',2,'Range','E2');
%writetable(TablaPropiedadesFisura,'DatosFisura.xls','Sheet',1,'Range','B2');
```

Apéndice B - Código en MATLAB Analysis

```
% Para leer de su propio canal, cambie el channelID valor.
ChannelIDFisuras = 2227566;
readAPIKeyFisuras = '9U91DTUCZJ2LGNQG';

% Todas las claves API de alertas comienzan con TAK.
alertApiKey = 'TAK1r68DeOqtySmTI83';

% Establecer la URL y el encabezado.
% El servicio de alertas requiere una ThingSpeak-Alerts-API-Key encabezamiento.
% Usar weboptions para establecer el encabezado.
alertUrl = "https://api.thingspeak.com/alerts/send";
options = weboptions("HeaderFields", ["ThingSpeak-Alerts-API-Key", alertApiKey ]);

% Establecer el asunto del correo electrónico.
```

```
alertSubject = sprintf("Número Imágenes con Fisura");

% Lea los datos recientes usando thingSpeakRead.
DatosFisura = thingSpeakRead(ChannelIDFisuras,'Fields',[1],'ReadKey',readAPIKeyFisuras);
DatosConFisura = thingSpeakRead(ChannelIDFisuras,'Fields',[4],'ReadKey',readAPIKeyFisuras);
DatosSinFisura = thingSpeakRead(ChannelIDFisuras,'Fields',[5],'ReadKey',readAPIKeyFisuras);

% Usa los datos mas recientes del canal para configurar el mensaje.
if (DatosFisura>0)
alertBody = sprintf('Imágenes detectadas con fisuras: %d\n Imágenes detectadas sin fisuras: %d\n',
DatosConFisura,DatosSinFisura );
% Envia la solicitud de alerta.
try
webwrite(alertUrl , "body", alertBody, "subject", alertSubject, options);

catch someException
fprintf("Failed to send alert: %s\n", someException.message);
end
end
```