UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA DE LA CONSTRUCCIÓN Y DE PROYECTOS DE INGENIERÍA CIVIL

MÁSTER UNIVERSITARIO EN INGENIERÍA DEL HORMIGÓN

TRABAJO FIN DE MÁSTER

CURSO ACADÉMICO 2020/2021

Aprendizaje profundo para la detección automática de fisuras de hormigón usando redes neuronales convolucionales

Autor: Julian Alberto Ortega Triana

Tutor: Ignacio Paya Zaforteza

Cotutor: José Antonio Lozano Galant

Valencia, septiembre de 2021

DEPARTAMENTO DE INGENIERÍA DE LA CONSTRUCCIÓN Y PROYECTOS DE INGENIERÍA CIVILUNIVERSITAT POLITÈCNICA DE VALÈNCIA

European Accreditation of Engineering Programmes EUR-ACE®

MUIH

RESUMEN

El envejecimiento de las infraestructuras hace de suma importancia el desarrollo de herramientas eficaces y rápidas en el área de patología estructural que permitan detectar potenciales patologías estructurales. Esto permite actuar rápidamente y evitar la degradación de la estructura y posteriores fallos.

En este contexto, este trabajo fin de máster (TFM) tiene como objetivo desarrollar una herramienta de *Deep Learning basado* en redes neuronales convolucionales para la detección automática de fisuras en superficies de hormigón.

De esta manera y para cumplir con el objetivo propuesto, se utilizó el *aprendizaje por transferencia*, una de las técnicas más importantes del *Deep Learning para* el aprendizaje automático en inteligencia artificial, que consiste en la modificación de algunos patrones ya entrenados de las redes neuronales, que reconocen una serie de características generales para reconocer otras más específicas.

En consecuencia, en este trabajo se comparó la efectividad, precisión y tiempo de procesamiento de tres arquitecturas de redes neuronales convolucionales (*GoogleNet, AlexNet y SqueezeNet*) las cuales fueron entrenadas con dos bases de datos (SDNET2018 y METU) para detectar específicamente fisuras de hormigón mediante la técnica del *fine tunnig*, con el fin de escoger solo una red que lograra completar la nueva tarea de clasificación de forma eficaz.

Así, los resultados muestran que con *GoogleNet* se clasifican las imágenes fisuradas y no fisuradas con una tasa de éxito del 93%. Además, de lograr detectar una grieta dentro de una región de interés delimitada y determinar el momento de aparición de una fisura analizando un video de rotura de una viga.

Por consiguiente y mediante este proceso, se generó una herramienta que es el punto de partida para la detección de fisuras y que con un mayor desarrollo en el futuro podrá clasificar automáticamente patologías en estructuras de mediante una red neural entrenada.

ABSTRACT

The aging of infrastructures makes the development of fast and effective measures, that will allow for the detection of potential structural pathologies, in the field of structural pathology, essential. This makes it possible to act fast, thus avoiding the degradation of the structure and any following malfunction.

In this context, this master's thesis aims at developing a deep learning tool based on convolutional neuronal networks for the automatic detection of fissures on concrete surfaces.

Thus, and with the aim of achieving the proposed objective, transfer learning was used, one of the most important deep learning techniques for artificial intelligence applied to automatic learning. It consists of the modification of some – already learnt-patterns of neuronal networks, recognizing certain general characteristics in order to then recognize some more specific ones.

Consequently, this work compared the efficacy, precision, and processing time of three convolutional neuronal networks architectures (GoogleNet, AlexNet y SqueezeNet), which were gathered from two databases (SDNET2018 y METU) to specifically find fissures in concrete through the fine-tuning technique, with the aim of choosing one network that could effectively complete the new classification task.

Thus, the results show that GoogleNet classifies fissured and non-fissured images with a 90% success rate. Moreover, it can detect a crack on a limited region of interest and determine the moment in which a fissure will appear by analyzing a video of the rupture of a rafter.

Consequently, through this process, a toll was generated, that will be the starting point for the detection of fissures, and that, with further development, in the future will be able to automatically classify structural pathologies through a trained neuronal network.

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

1 INTRODUCCION.

1.1 PLANTEMIENTO DEL PROBLEMA.

En muchas infraestructuras de hormigón aparecen fisuras a lo largo de su vida útil que pueden poner de manifiesto y potenciar posibles patologías estructurales. Por ello, la detección de estas fisuras es vital en los trabajos de inspección. En la mayoría de los casos, la inspección es visual y realizada por seres humanos. Esta inspección manual es laboriosa, costosa y requiere mucho tiempo. También tiene una confiabilidad, objetividad y reproducibilidad limitada, debido a la experiencia, habilidad y conocimientos necesarios y al error humano causado por la falta de atención y la fatiga.

La detección automática de grietas utilizando inteligencia artificial es una opción complementaria a la inspección humana visual. Para ello, se emplean algoritmos como el *Machine Learning* (ML) y el *Deep Learning (*DL) que han recibido gran atención en la última década.

El ML es un método que utiliza algoritmos matemáticos que permiten que la maquinas aprendan para resolver problemas mediante la identificación, la clasificación o la predicción. Los algoritmos identifican patrones y aprenden de datos introducidos y luego usan esta información para sacar conclusiones de nuevos datos.

El DL es una evolución del ML que utiliza algoritmos distintos. Mientras que el ML usa arboles de decisión o algoritmos de regresión, el DL usa redes neuronales convolucionales (CNN-*Convolutional neural network*) que funcionan de forma muy parecida a las conexiones neuronales del cerebro humano. Este método tiene un rendimiento excelente en el procesado de imágenes a gran escala y ha mostrado excelentes resultados cuando se han aplicado a la clasificación de objetos (Valueva et al., 2020)

En la *Figura 1* se muestra el grafico de interrelación de las diferentes técnicas computacionales, siendo la inteligencia artificial la técnica que permite a las

computadoras imitar la inteligencia humana mientras que el ML y DL son subconjuntos de esta.

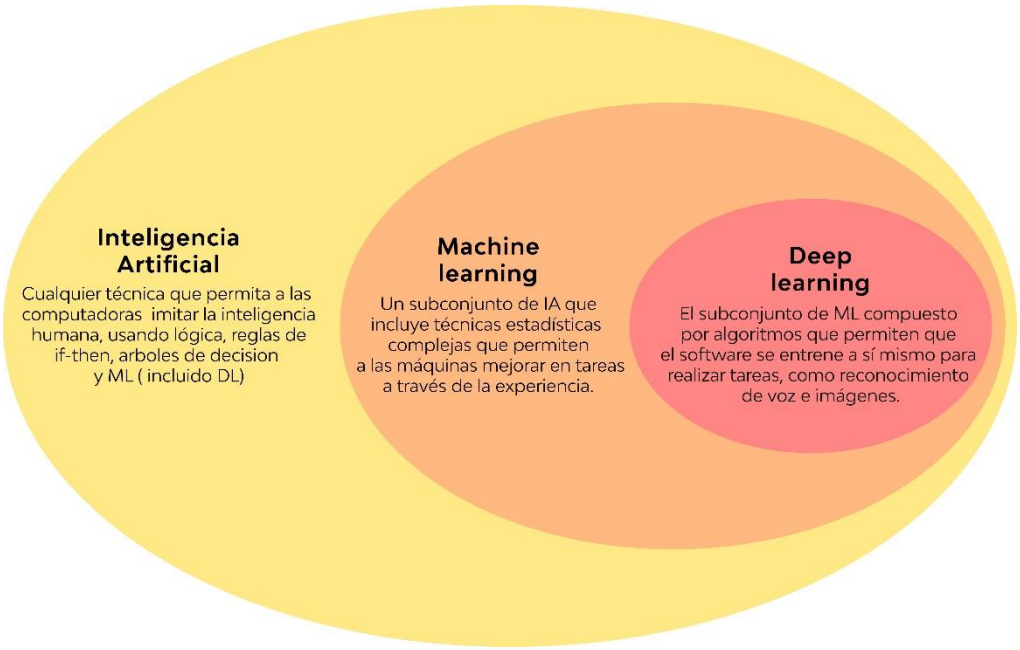


Figura 1. Gráfico interrelación de diferentes técnicas computacionales

Las CNN tienen una característica que les da importantes ventajas en el ámbito del reconocimiento y la clasificación: pueden extraer automáticamente características importantes de las imágenes tales como su enfoque/desenfoque, la ubicación de bordes, los cambios de brillo de la imagen, entre otras. La figura 2 muestra un esquema general de como identifica una CNN una especie animal. Para ello, las CNN contienen varias capas ocultas especializadas y con una jerarquía: las primeras capas pueden detectar líneas, curvas… y estas capas se especializan para reconocer formas complejas como un rostro o una silueta de un animal. Una técnica especialmente interesante dentro del DL es el *Transfer Learning* (TL), que se caracteriza por emplear modelos de CNN cuyas capas ya están entrenadas y usarlas para clasificar otro tipo de imágenes. Por ello, el TL muy útil cuando solo se dispone de una pequeña cantidad de datos, por sus ventajas en términos de tiempo y precisión.

Las redes reciben una serie de valores de entrada y cada una de estas entradas llega a un nodo llamado neurona (círculos de la *Figura* 2). Las neuronas de la red están a su

vez agrupadas en capas (bloques verticales de la *Figura* 2) que forman la red neuronal. Cada una de las neuronas de la red posee a su vez un peso, un valor numérico, con el que modifica la entrada recibida. Los nuevos valores obtenidos salen de las neuronas y continúan su camino por la red hasta cumplir con la tarea de clasificación.

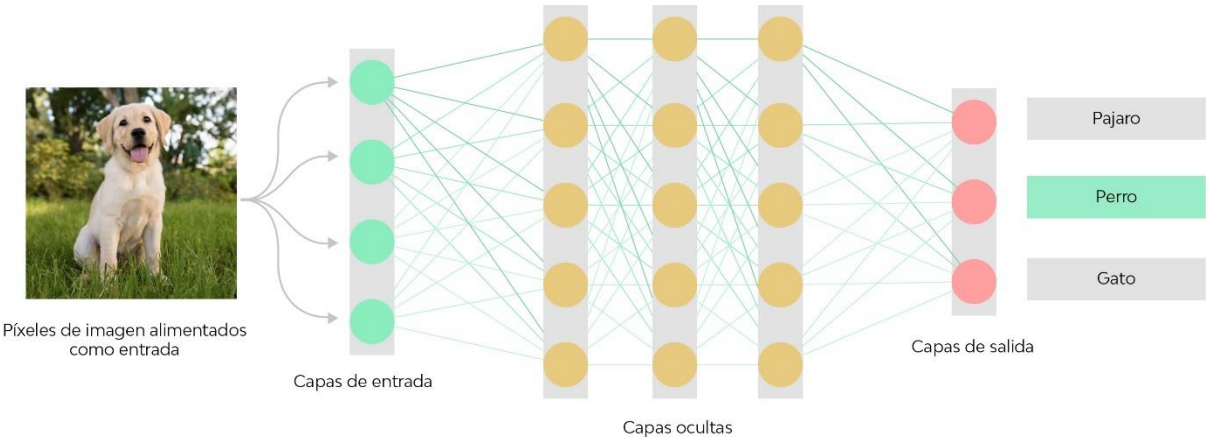


Figura 2. Funcionamiento de una red neuronal.

En este contexto, este trabajo final de máster (TFM) emplea tres arquitecturas de redes neuronales convolucionales (CNN), *AlexNet, SqueezeNet y GoogleNet,* para identificar fisuras en superficies de hormigón. Para ello, las arquitecturas se entrenaron con dos bases de datos con imágenes de superficies fisuradas de estructuras de hormigón. Ajustando los pesos de las bases de datos y calibrando la cantidad de imágenes durante la etapa de entrenamiento, se logró llegar a una precisión que se considera óptima para la tarea de clasificación. Además, se ha conseguido determinar la distancia a la que debe tomarse una fotografía en un proceso de inspección estructural.

1.2 OBJETIVOS, CONTRIBUCIONES Y ESTRUCTURA DEL DOCUMENTO

Los objetivos del presente Trabajo Fin de Máster son:

Objetivo General.

Desarrollar una herramienta basada en redes neuronales convolucionales para la detección automática de fisuras de hormigón.

Objetivos específicos.

- Obtener un banco de imágenes de fisuras en superficies de hormigón que permita disponer de conjuntos de datos para entrenar las redes neuronales convolucionales.

- Entrenar los modelos de redes neuronales convolucionales con el banco de datos obtenido.

- Analizar los resultados de entrenamiento de las redes neuronales y elegir la más adecuada.

- Validar la precisión de la herramienta con una imagen de una fisura de un ensayo de rotura de una vigueta de hormigón.

Para conseguir estos objetivos, este trabajo se estructura de la forma siguiente. El estado actual del conocimiento en el tema de estudio se presenta en el capítulo 2. Seguidamente, el capítulo 3 describe la metodología y bases de datos de imágenes empleadas, presenta las arquitecturas de redes neuronales analizadas y proporciona los detalles de su entrenamiento. Finalmente, el capítulo 4 muestra la aplicación de estas redes a la detección de fisuras en un ensayo de flexión de una vigueta pretensada y el capítulo 5 contiene las conclusiones del trabajo y las futuras líneas de investigación.

2 ESTADO DEL ARTE

2.1 Metodología de búsqueda

La pregunta de investigación de este estudio es cómo los algoritmos de inteligencia artificial (IA) han aportado avances a la ingeniería civil en la detección de fallos en el ámbito de la monitorización de estructuras (*Structural Health Monitoring* en inglés).

Para responder a esta pregunta se ha realizado una búsqueda y selección de artículos en bases de datos científicas y académicas con las características siguientes:

- La bases de datos *Web of Science, Scopus, Proquest y Ebsco* fueron utilizadas para la búsqueda y selección de los artículos. Se establece un periodo de búsqueda entre los años 2000 y 2021 ya que no existe información anterior a ese periodo.

- Se han utilizado como palabras claves los términos "*Crack Detection", "Bridge inspection", "artificial intelligence", "pathologies", "Neural networks", "artificial intelligence in civil and structural engineering", "Deep Learning structural engineering"* con los operadores booleanos "and" y "or" y restringiendo la búsqueda a la inteligencia artificial aplicada a la ingeniería estructural.

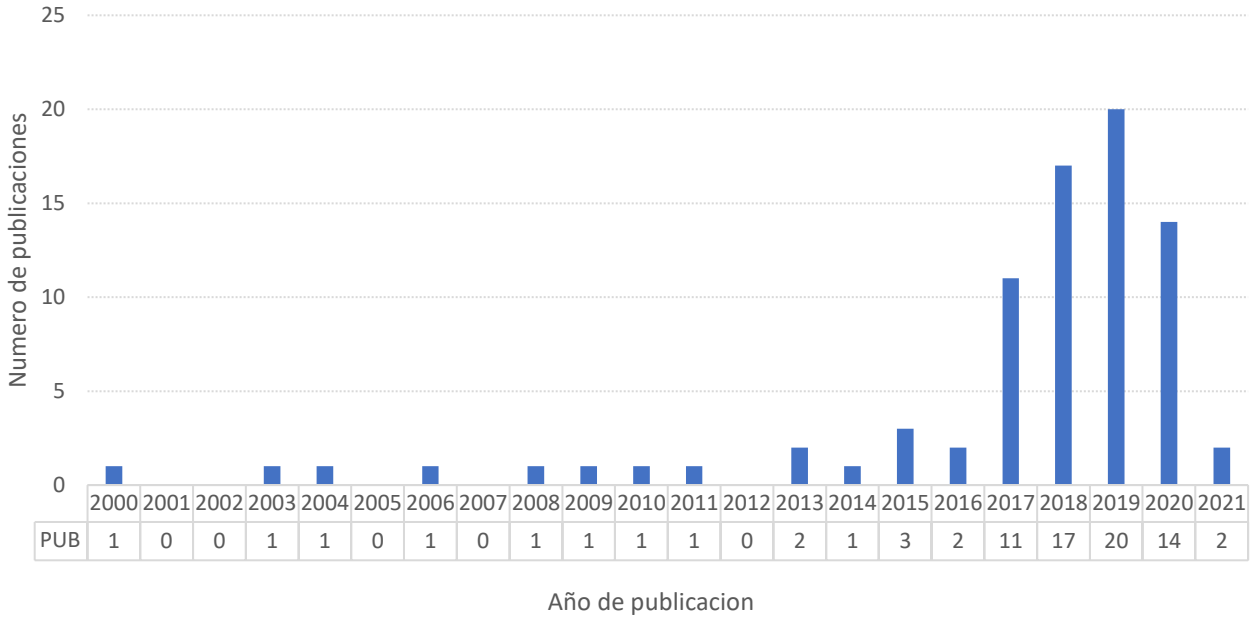
- Se seleccionaron los artículos usando como criterio solo artículos académicos con proceso de revisión por pares y en inglés. De esta forma se obtuvo un conjunto inicial de 39 artículos.

- Se revisaron y analizaron las referencias de los artículos seleccionados anteriormente utilizando los filtros ya mencionados para la selección del primer conjunto de artículos.

La técnica de búsqueda utilizada anteriormente está basada en la empleada por (Navarro et al., 2019) (Navarro et al., 2019) y ha proporcionado como resultado final un conjunto de 80 artículos.

2.2 Revisión de la información

La Figura 3 muestra la evolución temporal del número de artículos publicados en el tema analizado desde el año 2000. En la primera década, el número de publicaciones fue reducido (un 8,75% del total). Sin embargo, un 86 % de las publicaciones analizadas son posteriores al año 2014, lo que revela que el tema es de gran actualidad.



	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
PUB	1	0	0	1	1	0	1	0	1	1	1	1	0	2	1	3	2	11	17	20	14	2

Figura 3. Distribución de publicaciones por año (2000-2021)

La *Figura* 4 muestra las tipologías estructurales consideradas en las publicaciones seleccionadas. Analizando las mismas, se puede indicar que:

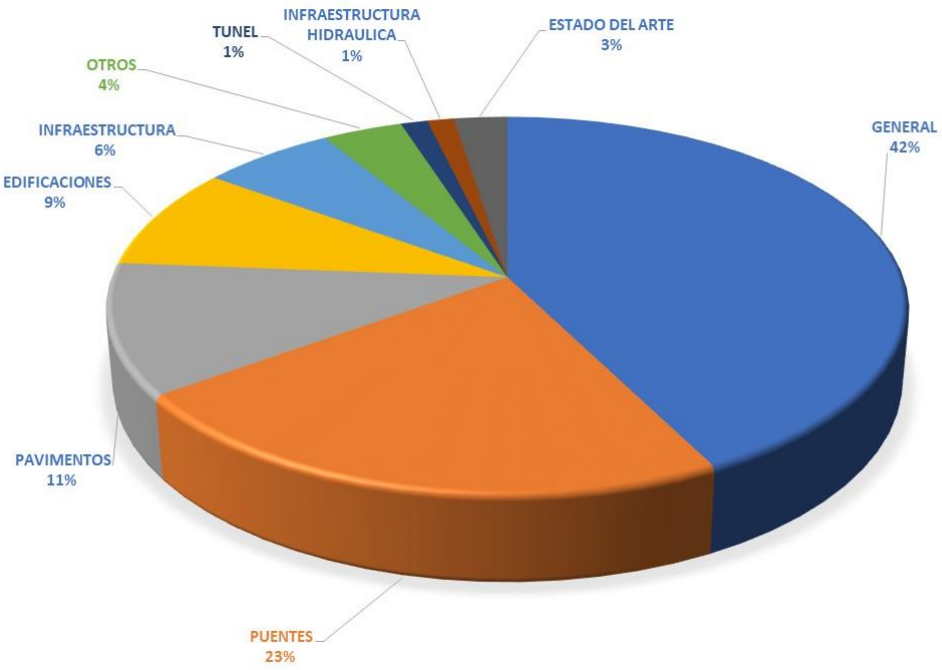


Figura 4. Gráfico de tortas – aplicaciones de DL en tipologías estructurales

i) Un 42.5% de los artículos analizan aspectos generales, no asociados a una tipología estructural concreta. Se han catalogado en este grupo los artículos donde se han usado patrones de fisuras en superficies metálicas y de hormigón. Estos artículos emplean para el entrenamiento y validación de los algoritmos bases de datos de acceso libre (RESNET2018, METU CAMPUS BUILDING) o generadas específicamente para ese fin.

ii) Un 23% de los artículos están relacionados con aplicaciones a los puentes. Actualmente existe una creciente demanda en la monitorización e inspección del estado de estas estructuras ya que muchos puentes fueron construidos hace más de 50 años y presentan signos de desgaste por efectos ambientales y de cargas (Vu et al., 2019). Estos estudios utilizaron

imágenes tomadas in situ por medio de inspección visual, vehículos aéreos no tripulados, bases de datos de tableros o vigas de puentes de hormigón y daños por fatiga, fisuras y corrosión en puentes metálicos.

iii) Un 11% de los documentos muestran un interés en la monitorización del estado de los pavimentos y en especial en el desgaste que es uno de los factores que afectan la funcionalidad de las carreteras. Por estas razones, la detección automática de grietas en pavimentos se ha convertido en un tema importante en la investigación del transporte, y específicamente los métodos basados en visión por computadora atraen cada vez más atención debido a su bajo costo, buena estabilidad y facilidad de operación.

iv) El 9% de los artículos se centran en edificaciones y están relacionados con el desarrollo de herramientas eficaces y rápidas aplicables al ámbito de la patología estructural.

v) Un 6% de los artículos se enfocan en la infraestructura en general, abarcando desde componentes de puentes de hormigón hasta muros de estructuras hidráulicas. Según (Nguyen et al., 2020), muchos de estos componentes, han sufrido diversas condiciones geológicas, de carga y ambientales que causan grietas que influyen en la calidad de las operaciones. Por tanto, la evaluación del estado de las infraestructuras existentes es una tarea importante para garantizar un pronto mantenimiento.

vi) El 4% de los artículos se enfocan en el desarrollo de las redes neuronales para la detección de grietas, bordes y demás características de las fisuras, pero no en superficies de hormigón o acero, sino en otro tipo de elementos (cerámicos por ej.) o en la detección de bordes y grietas generadas en

elementos estructurales por softwares de elementos finitos. Se ha definido esta categoría como “otros”.

vii) Un 1% de los artículos está dedicado a túneles, en este se presenta un mecanismo para detección de fisuras proponiendo una forma de agilizar las operaciones de recolección de datos por medio de algoritmos de inteligencia artificial. En condiciones normales la operación demanda una gran cantidad de recursos.

viii) El 1% es enfocado a estructuras hidráulicas. Se usa una red convolucional (CNN) con transferencia de aprendizaje (DL) y fotos tomadas en el sitio con una cámara de alta definición, para detección de daños en la estructura.

ix) Un 3% de los artículos se enfocan en determinar y trazar la evolución de los algoritmos de inteligencia artificial que se han utilizado para la clasificación de imágenes.

La revisión de los artículos muestra un crecimiento en la última década del uso de métodos de inteligencia artificial aplicados al campo de la ingeniería civil. Además, se observa que, dentro de los muchos métodos de las IA, el ML y el DL son los que más se han adaptado y utilizado en la monitorización estructural, detección de daños, optimización, modelado de propiedades del hormigón, identificación estructural, etc.

El estado del arte muestra que los algoritmos de ML se han ocupado de problemas como el desarrollo de herramientas de gestión para la seguridad estructural y la identificación de daños, optimización, evaluación de desempeño, confiabilidad estructural e identificación de parámetros estructurales. Dentro de todas las aplicaciones mencionadas, el ámbito de la monitorización estructural y modelación de propiedades del hormigón es el que ha recibido mayor atención en los últimos diez

años. No obstante, la revisión realizada muestra que las arquitecturas de DL son las más utilizadas para la monitorización estructural e identificación de daños. Estos algoritmos son un método nuevo y se clasifican como técnicas de monitorización estructural basadas en la visión, en las que se toma un conjunto de imágenes capturadas en varios estados de la estructura estudiada para extraer propiedades de las imágenes como la posición de fisuras, aprender características de estas propiedades y, en base a este aprendizaje, desarrollar modelos predictivos. Todos los estudios sugieren que los métodos del DL son herramientas efectivas para la monitorización de salud estructural y que su uso para futuras investigaciones es muy prometedor.

Según lo descrito anteriormente, la *Figura* 5 muestra las aplicaciones actuales y emergentes de ambos métodos.

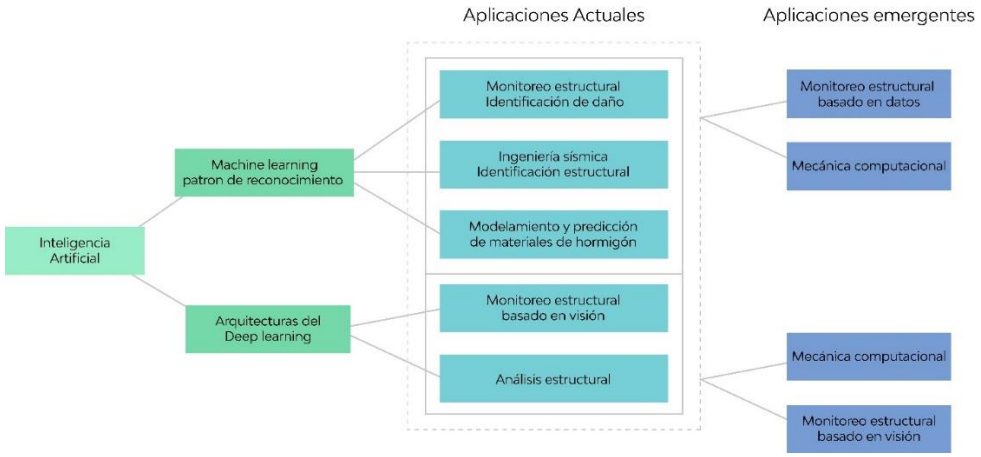


Figura 5. Aplicaciones del ML y DL en la ingeniería estructural.

Uno de los grandes desafíos dentro del desarrollo de algoritmos de reconocimiento visual ha sido la segmentación semántica, que es la capacidad de segmentar una imagen desconocida en diferentes partes y objetos (por ejemplo, playa, océano, sol, perro, nadador). Además, la segmentación es incluso más profunda que el reconocimiento de objetos porque el reconocimiento no es necesario para la

segmentación. Específicamente, los seres humanos pueden realizar la segmentación de imágenes sin siquiera saber cuáles son los objetos. Realizar la segmentación sin conocer la identidad exacta de todos los objetos en la escena es una parte importante de nuestro proceso de comprensión visual que puede brindarnos un modelo poderoso para comprender el mundo y también puede usarse para mejorar o aumentar las técnicas de visión por computadora existentes.

Como conclusión importante tras la revisión de la documentación investigada y analizada, se puede decir que el uso de algoritmos de inteligencia artificial aplicados a la ingeniería civil está en aumento, por lo que desarrollar herramientas de automatización de procesos en la monitorización y patología estructural basadas en aprendizaje profundo, sugiere una línea de investigación importante.

Además, la aplicación de campos emergentes como la mencionada anteriormente segmentación semántica en algoritmos de redes neuronales aplicadas a ingeniería estructural, vislumbra un vasto campo de aplicación mejorando la precisión de dichos algoritmos y creando la posibilidad de automatizar la clasificación de patologías estructurales.

3 METODOLOGIA

Dentro de este capítulo, se van a especificar las técnicas que van a ser empleadas para obtener la información que se necesita para la elaboración del presente trabajo final de máster.

La metodología empleada comprende los pasos siguientes:

- Recopilación y preparación de las bases de datos de imágenes de fisuras en superficies de hormigón.
- Definición y explicación de los algoritmos de IA empleados. Para el desarrollo de este trabajo se utilizarán arquitecturas de redes neuronales convolucionales CNN.
- Explicación del proceso de convolución de una capa convolucional de una CNN.
- Definición de los componentes de las capas de las CNN.
- Explicación del *Transfer Learning* y descripción de las tres arquitecturas a utilizar (*AlexNet, SqueezeNet y GoogleNet)*.
- Definición y aplicación del *Fine-tuning* para el proceso de entrenamiento.

3.1 Recopilación y preparación de las bases de datos

En los siguientes apartados se explicará el proceso de tratamiento de las bases de datos siguiendo los siguientes pasos:

- Indicar las bases de datos de imágenes de fisuras utilizadas y que imágenes de estas bases de datos fueron utilizadas.
- Aplicar el *Data Augmentation* para aumentar la cantidad de imágenes y ajustar el tamaño de entrada según la red neuronal a utilizar.
- Dividir las imágenes de cada base de datos en grupos para entrenar, validar y probar las redes neuronales.

Con el desarrollo de las redes neuronales y los modelos pre entrenados algunos conjuntos de datos se han vuelto indispensables para el entrenamiento de reconocimientos de objetos visuales por medio de inteligencia artificial como *ImageNet* o *PASCAL VOC* (Gao & Mosalam, 2018). A diferencia de PASCAL VOC, ImageNet contiene más de un millón de imágenes relacionadas con la ingeniería estructural, como el archivo en línea de ingeniería sísmica de NISEE, que incluye imágenes estructurales del estado anterior y posterior a sacudidas sísmicas. Sin embargo, estos bancos de datos no contienen imágenes enfocadas en el entrenamiento de fisuras. Por eso, en este TFM se contó con dos bases de datos especialmente diseñadas para este fin: *SDNET2018* y *CONCRETE METU CAMPUS BUILDING DATASET*.

SDNET2018 (Marc Maguire (2018), Utah *State University doi:* //doi.org/10.15142/T3TD19), es un conjunto de datos de imágenes anotadas para entrenamiento, validación y evaluación comparativa de algoritmos de detección de grietas basados en inteligencia artificial en elementos hormigón. Contiene más de 56.000 imágenes de tableros de puentes de hormigón, muros y pavimentos. Las imágenes están etiquetadas en dos grupos: *positive* para imágenes con grietas y *negative* para imágenes sin grietas. Todas las imágenes tienen un tamaño de 227x227 pixeles con canales RGB.

La base de datos *CONCRETE METU CAMPUS BUILDING DATASET* (Özgenel, Çağlar Fırat (2019), "*Concrete Crack Images for Classification*", Mendeley Data, V2, doi: 10.17632/5y9wdsg2zt.2) es un conjunto de datos que contiene imágenes de grietas de hormigón procedentes de varios edificios del campus del Middle *East Technical University* (METU). El conjunto de imágenes está etiquetado como *negative* para superficies sin grietas y *positive* para superficies agrietadas. Cada clase tiene 20.000 imágenes, por lo que la base contiene 40.000 imágenes de 227 x 227 píxeles con canales RGB.

En este TFM se han utilizado imágenes de tableros (11.595 imágenes negativas y 2.025 imágenes positivas) y muros (3.851 imágenes negativas y 11.595 imágenes positivas) de la base SDNET2018 y las 40.000 imágenes de la base METU, ver Tabla 1.

Etiquetas	**Sub carpetas**			**Total**	**Base de datos**
	Entrenamiento	**Validación**	**Prueba**		
Negative	6957	2899	1739	11595	SDNET2018-TABLEROS
Positive	1215	506	304	2025	
Negative	8572	3572	2143	14287	SDNET2018-MUROS
Positive	2311	963	578	3851	
Negative	12000	5000	3000	20000	METU
Positive	12000	5000	3000	20000	
TOTAL	**43055**	**17940**	**10764**		

Tabla 1. Bases de datos empleadas.

A continuación, y a modo de ejemplo, se muestran las imágenes fisuradas y no fisuradas de cada uno de los bancos de imágenes utilizados en este trabajo. La Figura 6 corresponde a la base de datos de puentes de SDNET2018 y la Figura 7 a la de muros de hormigón de SDNET2018, mientras que la *Figura* 8 corresponde a la base de datos de la universidad METU.

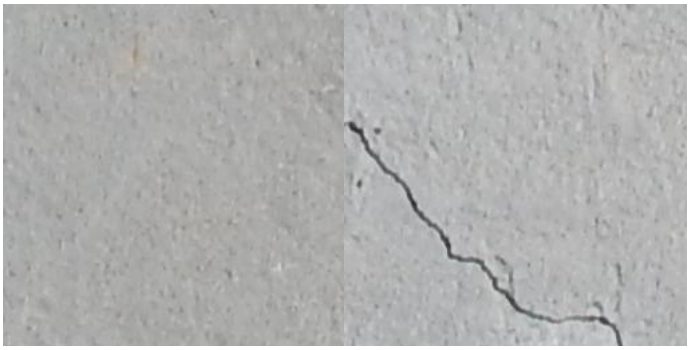


Figura 6. Imágenes No fisurado/fisurado - SDNET2018 (*Tableros de puentes*).

Figura 7. Imágenes No fisurado/fisurado - SDNET2018 (Muros de hormigón).

Figura 8. Imágenes No fisurado/fisurado - METU

Después de seleccionar las imágenes que se utilizarán de las bases de datos SDNET2018 y METU, se dispondrá a realizar el aumento y la agrupación de estas para el proceso de entrenamiento de las redes neuronales.

3.1.1 Data Augmentation

El *data augmentation* es el proceso de aumentar la cantidad de imágenes disponibles transformando imágenes originales para conseguir variantes de una única foto. Este proceso es importante, debido a que el entrenamiento de una red neuronal convolucional requiere de muchas muestras.

Además, el *data augmentation* es uno de los métodos más utilizados para reducir el problema de sobreajuste. Se denomina sobreajuste al hecho de hacer un modelo tan ajustado a los datos de entrenamiento que se impide la generalización de los datos de prueba y la consecuente obtención de patrones generales que sean extrapolables a nuevos datos. Algo similar ocurre a los seres humanos: el sobreajuste se produce si se aprenden las cosas de memoria sin entender el concepto.

Para aumentar el número de imágenes positivas de la base de datos se aplicaron rotaciones de 45º, 90º, 180º y espejo vertical como muestra la Figura 9 , lo que permitió multiplicar por cuatro el número de imágenes disponibles.

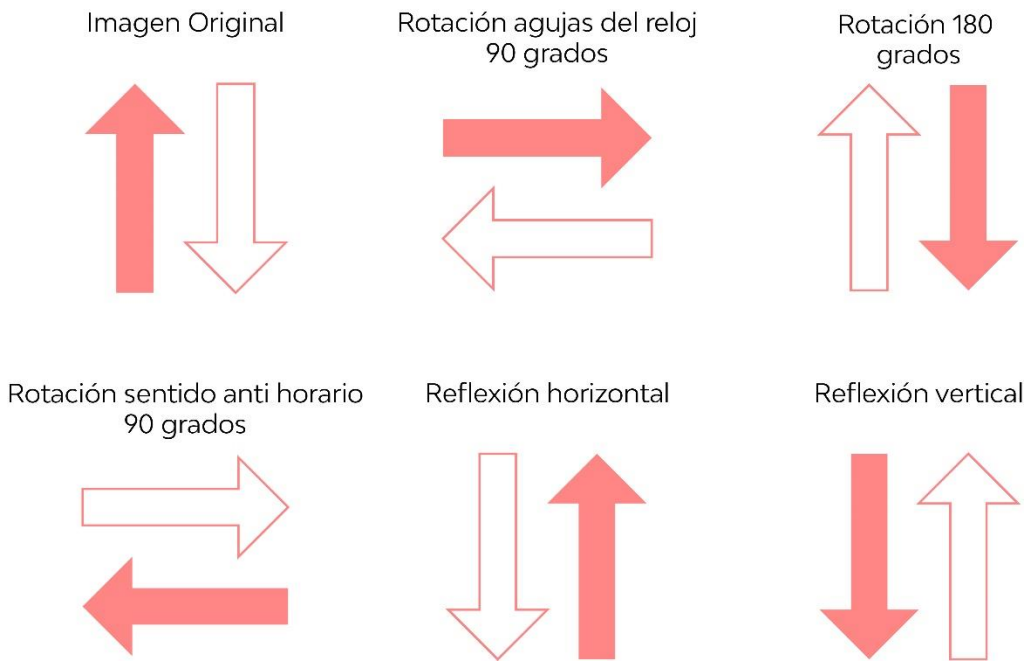


Figura 9. Ejemplo de incremento del número de imágenes

Finalmente, algunas de las CNNs empleadas en este TFM requerían imágenes de entrada de tamaño 224x224x3 y 227x227x3 pixeles, que eran diferentes de los tamaños de las imágenes de las bases de datos. Por ello, se utilizó un *augmentation* para cambiar automáticamente el tamaño de las imágenes de entrada, empleando para ello el código de MATLAB de la Figura 10.

```
% Image Augmentation
% imageAugmenter = imageDataAugmenter( ...
%     'RandXReflection',true, ...
%     'RandYReflection', true);
% augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
%     'DataAugmentation',imageAugmenter);
%
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain);
augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation);
augimdsTest = augmentedImageDatastore(inputSize(1:2),imdsTest);
```

Figura 10. Código MATLAB de *augmentation* para el ajuste del tamaño de las imágenes de entrada.

3.1.2 Agrupación de datos.

La división de los datos de entrada en tres grupos (entrenamiento, validación y prueba) es crucial en la creación del algoritmo de DL. El grupo de entrenamiento se usa para que la red neuronal aprenda patrones sobre los datos. El grupo de validación se utiliza para buscar cuáles son los mejores hiperparámetros de la red neuronal para la resolución de un problema (los hiperparámetros son características de las CNN como el número de capas, número de nodos, funciones de activación, etc). Finalmente, el grupo de prueba se emplea para comprobar la calidad la red neuronal después de entrenarla y encontrar los mejores hiperparámetros.

Los grupos normalmente deben ser divididos aleatoriamente al inicio de cada proyecto, una partición muy frecuente es una proporción de 60:20:20 (60 % para entrenamiento, 20 % para validación y 20% para prueba). No obstante, estos valores generalmente dependen de la naturaleza del proyecto y del tamaño de la base de datos, empleándose en este TFM una distribución de 60:25:15 (véase la Tabla 2)

Etiquetas	Grupos			Total
	Entrenamiento	Validación	Prueba	
Negative	27529	11471	6882	45882
Positive	28533	11889	7133	47555

Tabla 2. Preparación de los grupos de imágenes.

3.2 Definición y explicación de los algoritmos de IA: Machine Learning y Deep Learning.

El *Machine Learning* (ML) o aprendizaje automático es uno de los mayores campos de las inteligencias artificiales que analiza información extrayendo y aprendiendo de ésta para después tomar decisiones por cuenta propia, sin necesidad de una programación previa y explícita. Una subclase del ML es el aprendizaje profundo o *Deep Learning (*DL) que se compone de redes neuronales convolucionales (CNN). Estas redes tienen más de una capa oculta y funcionan de forma muy parecida a las conexiones neuronales del cerebro humano. En una arquitectura de este tipo, aumentar el número de capas da como resultado una red más profunda. Las redes con más profundidad tienen más neuronas que pueden mejorar las predicciones en conjuntos de datos más complicados (Salehi & Burgueño, 2018).

Para el desarrollo de este trabajo se utilizará algoritmos de DL que se componen de redes neuronales convolucionales. A continuación, en la *Figura* 11 se muestra la arquitectura de una red neuronal convolucional típica para una tarea de reconocimiento de imágenes. La CNN tiene una imagen de entrada y pasa por las diversas capas de la red hasta llegar a la clasificación de salida. Estas redes comprenden tres tipos de capas: capas convolucionales (*Convolution*), de agrupación *o pooling* y capas completamente conectadas o *fully connected* y cada una cuenta con

su función de activación ReLU. En el apartado 3.6 se explican qué son las funciones de activación.

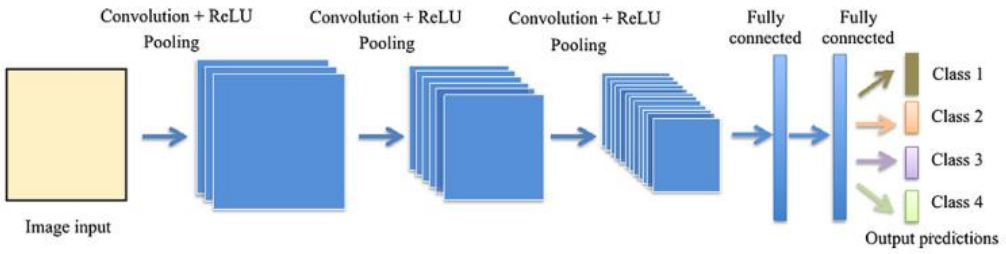


Figura 11. Esquema de arquitectura típico de una red neuronal convolucional (Salehi & Burgueño, 2018)

Las CNN en una etapa inicial del proceso alternan entre las capas convolucionales y las *pooling,* y para la etapa final trabajan con las capas completamente conectadas con un clasificador no lineal, que toma una decisión de clasificación basándose en los valores de las características de las capas anteriores. (ReLU). (Salehi & Burgueño, 2018).

La *Figura* 12 muestra la arquitectura de la red convolucional llamada VGG-16, que es utilizada para reconocimiento de imágenes. En la figura se muestran los componentes de las redes neuronales mencionados anteriormente. La red recibe una imagen de entrada con tamaño 224x224x3 pixeles (3 se refiere a que es una imagen a color, tiene los 3 canales RGB) y cada vez que avanza por cada una de las capas, la imagen convoluciona reduciendo su tamaño mientras la red va extrayendo y almacenando características. Las ultimas capas con las funciones de activación en conjunto con las capas completamente conectadas son las encargadas de la tarea de clasificación.

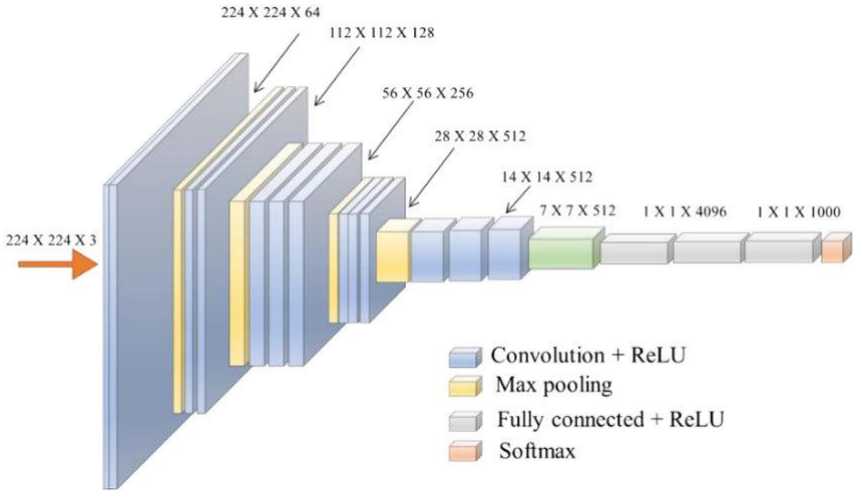


Figura 12. Arquitectura de la VGG-16 (Pandiyan et al., 2019)

3.3 Capa convolucional (Convolutional layer).

Para poder entender el funcionamiento de las redes neuronales convolucionales es necesario conocer de qué están compuestas las capas convolucionales y cómo funcionan.

La capa convolucional es un método efectivo de extracción de características. Las capas convolucionales reciben como entrada una imagen de tamaño variable convertida en pixeles, a la cual se le aplica un filtro que devuelve un mapa con las características de la información de entrada. Con esta operación se reduce el tamaño de los parámetros de la información inicial. Este proceso da como resultado un píxel en la imagen de salida.

Las neuronas son las unidades básicas de una red neuronal. En las CNN, cada neurona de una capa está conectada con cada neurona de la capa siguiente, la fórmula 1 muestra la operación del píxel de salida de la convolución.

$$y_i = \sum_i x_i * w_{ij} + b_i \quad (1)$$

Donde x_i representa la entrada de cada canal, w_{ij} los pesos de cada filtro que son los que controlan la fuerza de conexión entre dos neuronas, b_i representa el *bias* o sesgo que es un peso adicional que se le da a la imagen de salida y siempre tiene en valor de 1. El *bias* garantiza que incluso cuando todas las entradas son ceros, todavía habrá una activación en la neurona e y_i representa el píxel de salida.

Dentro de las capas convolucionales existen hiperparámetros que son parámetros que ya están configurados antes del modelo y son propios del algoritmo. Dentro de las convoluciones hay tres que son importantes: *kernels, paddings* y los *stride*.

3.3.1 Kernels o filtros

Los filtros o *kernels*, son matrices pequeñas que se utilizan al hacer una convolución con la imagen. Se utilizan para detectar bordes, desenfoques, enfoques, etc. Algunos de los filtros existentes se muestran en la *Figura* 13.

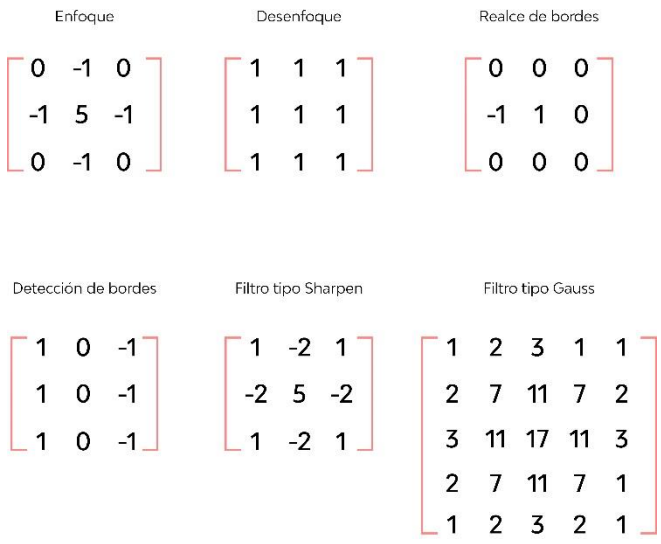


Figura 13. Ejemplos de filtros más utilizados

La *Figura* 14, muestra el proceso inicial de la convolución. Donde una imagen de entrada que tiene un tamaño de 6x6x1 pixeles, (1 se refiere al número de canales de color, en este caso asumimos una imagen en escala de grises). Las imágenes entran como matrices y cada numero de la matriz es un píxel. En el ejemplo de la figura 14 se aplica un filtro 3 X 3 (también llamado *kernel,* 3 X 3 es el tamaño del *kernel*). Los números en el filtro sugieren que es un detector de borde (ver figura 13) y extraerá características de borde de la entrada original. El filtro convolucionará con la imagen de entrada y generará una imagen de salida.

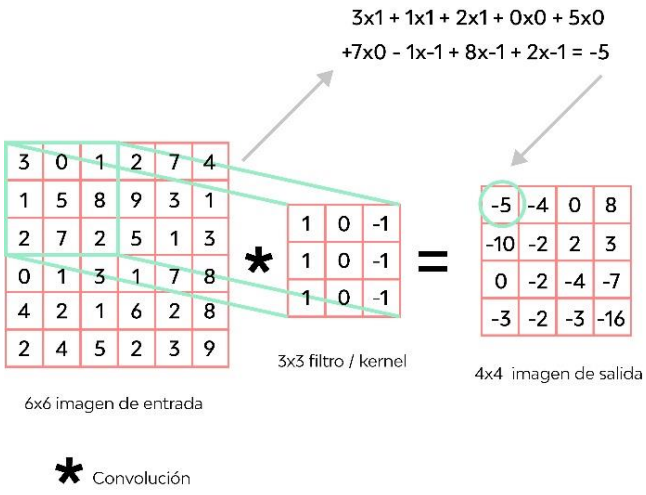


Figura 14. Proceso de convolución

Seguidamente se muestra el código de *MATLAB* y la aplicación de este a las imágenes de algunos de los filtros utilizados en las capas convolucionales de las redes de este trabajo (figuras 15,16, 17 y 18).

```
% 1. Sharpen Filter
Isharpen = imsharpen(I, 'Amount', 2, 'Radius', 16);
figure; montage({I,Isharpen},[])
title('Original Image Vs Sharpen Image');


% 2. Gaussian Blur
Iblur = imgaussfilt(I, 8);
figure; montage({I,Iblur},[])
title('Original Image Vs Blured Image');

% 3. Edge Filter
Iedge = I - Iblur;
Idiffuse = rgb2lab(Iedge);
max_luminosity = 100;
L = Idiffuse(:,:,1)/max_luminosity;
Idiffuse(:,:,1) = imadjust(L)*max_luminosity;
Iedge = lab2rgb(Idiffuse);
figure; montage({I,Iedge},[])
title('Original Image Vs Edge Image');

% Igray = rgb2gray(I);
% Iedge1 = edge(Igray,'Canny', []);
% Iedge2 = edge(Igray,'Prewitt', 0.1);
% Iedge3 = edge(Igray,'Sobel', 0.1);
% figure; montage({Igray,Iedge1,Iedge2,Iedge3}, [], 'Size', [2 2])
% title('Original Image Vs Canny Vs Prewitt Vs Sobel');
```

Figura 15. Código MATLAB de algunos filtros

Figura 16. Filtro *Sharpen* – Vigueta T12.2 laboratorio de hormigón de la universidad Politécnica de valencia (UPV)

Figura 17. Filtro *Blured* - Vigueta T12.2 laboratorio de hormigón UPV

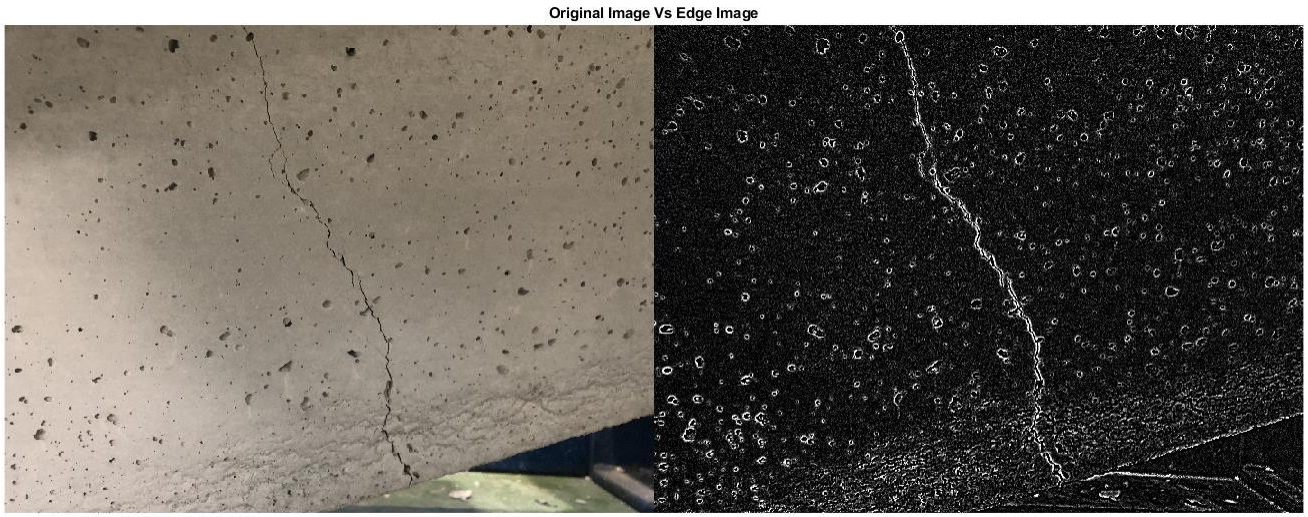


Figura 18. Filtro detección de bordes - Vigueta T12.2 laboratorio de hormigón UPV

3.3.2 Paddings

El segundo hiperparámetro son los *paddings* o rellenos y su función es la de adicionar pixeles en los bordes de la imagen de entrada antes de que se aplique la convolución mencionada anteriormente. Esto se hace porque, a medida que aumenta el número de capas convolucionales en una red neuronal, una imagen puede reducirse a un tamaño muy pequeño (como de 1 X 1 pixeles), lo que supone una pérdida importante de información del centro de la imagen.

En la *Figura* 19 se muestra un ejemplo de una convolución, donde a una imagen de tamaño original de 6x6x1pixeles (1 se refiere al número de canales de color, en este caso es una imagen en escala de grises) se le añade un relleno de ceros y se aplica un filtro de 3x3 que extraerá características de la imagen. El filtro convolucionará con la imagen de entrada y generará una imagen nueva.

Como se puede ver la imagen de 6x6 mantiene su tamaño después de la convolución debido a los rellenos aplicados a estas. Esto con el fin de que la imagen a medida que avanza en las capas convolucionales no pierda información importante en el centro de la imagen.

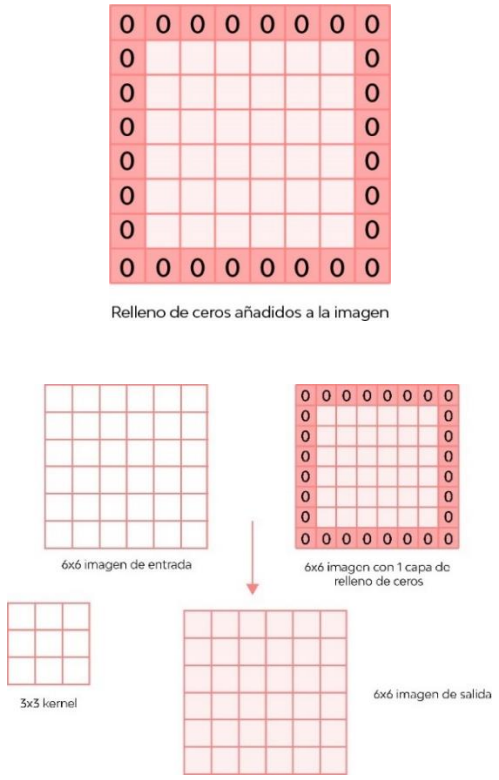


Figura 19. Proceso de convolución aplicando rellenos (Padding)

A continuación, se muestra un ejemplo de un relleno de ceros en los bordes de la imagen el cual se representa por un borde negro alrededor de la imagen

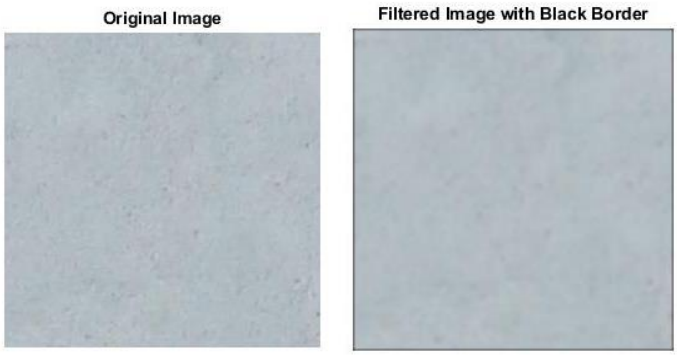


Figura 20. Imagen con borde negro de relleno (Uncracked wall SDNET2018).

3.3.3 Strides

Por último, tenemos los *strides* o zancadas que son los pasos en el que se desplazan los pixeles sobre la imagen de entrada. La orientación de las zancadas depende de la configuración de cada arquitectura, en algunos casos primero es horizontal y luego vertical, pero en otras arquitecturas es, al contrario, es la forma que tiene la arquitectura de leer los pixeles de la imagen. En la figura 21 se muestra de ejemplo una zancada de (1,1). Los pixeles se mueven de 1 en 1 en la convolución con un movimiento de izquierda a derecha y de arriba abajo. Como se observa en el ejemplo, el primer paso de la convolución es la operación de la matriz roja con el volumen de entrada, generando un píxel rojo en la imagen de salida. Luego avanza un píxel a la derecha (matriz verde), esta opera con la imagen de entrada y genera un segundo píxel en la imagen de salida (píxel verde).

Figura 21. Proceso de las zancadas (strides)

En la Figura 22 se muestra el proceso de la convolución. Se tiene una imagen de entrada de tamaño (7x7x3 pixeles, el 3 se refiere a que es una imagen de color con los canales RGB, rojo, verde y azul) a la que se le aplica un filtro *kernel* de tamaño

(3x3x3), este filtro opera con la imagen de entrada moviéndose las zancadas que se fijen, en este caso son de 1 en 1 (movimientos de izquierda a derecha y de arriba abajo), por último, el bias hace que los 3 canales se junten en una sola matriz generando la imagen de salida.

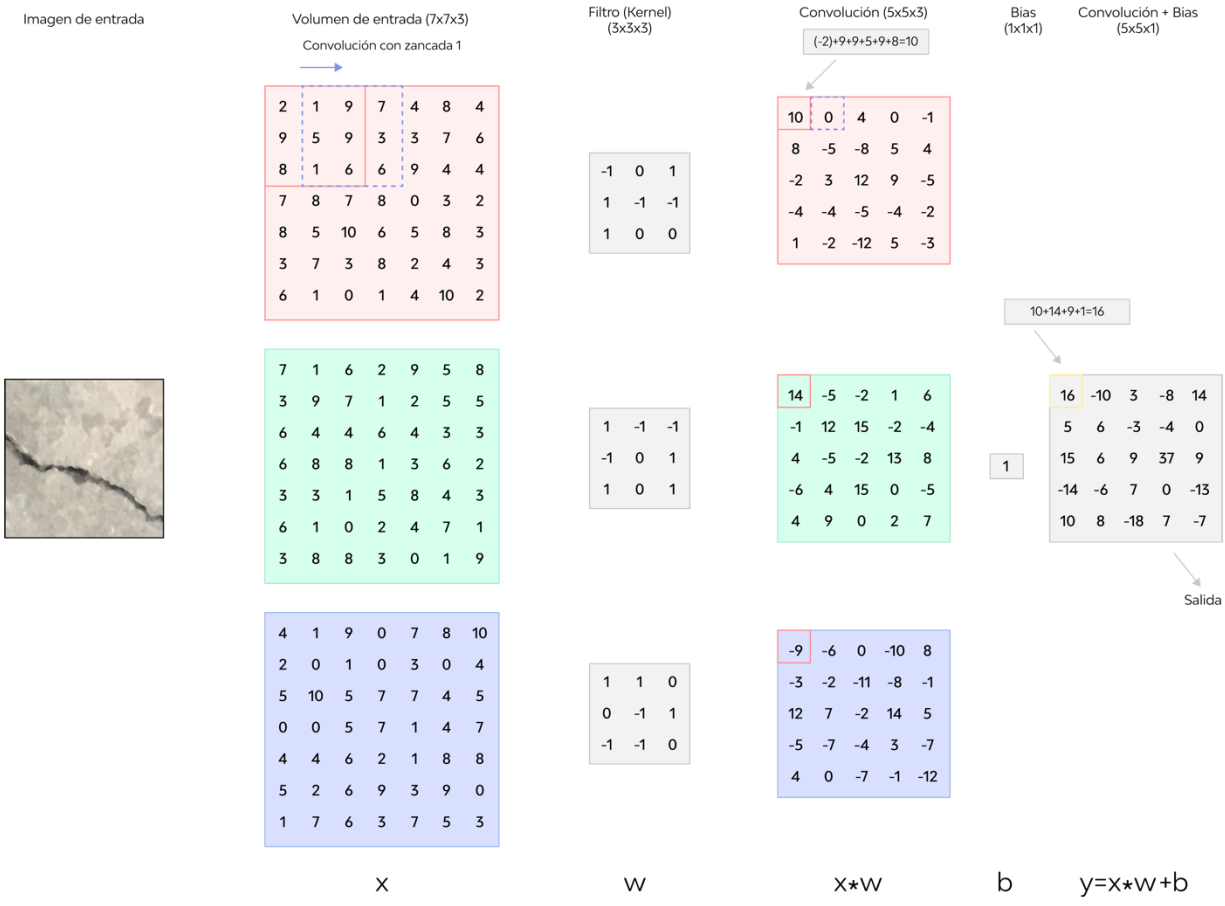


Figura 22. Proceso de convolución.

Después de explicar los componentes de las capas de convolución, que son las encargadas de obtener las características de las imágenes de entrada, se procede a explicar las siguientes capas presentes en las CNN.

3.4 Capas de agrupación (Pooling layer)

Las capas de agrupación o *max pooling* son nuevas capas que se agregan después de la convolución y se utilizan para reducir las dimensiones bajando los pixeles de la imagen obtenida de las capas de convolución del paso anterior.

En las capas de agrupación era común utilizar capas de agrupación promedio para propagar el valor medio de los datos de entrada. Sin embargo, en los últimos modelos se utiliza la agrupación máxima o el *max pooling* el cual propaga el valor máximo dentro de un campo receptivo para la siguiente capa (Rawat & Wang, 2017).

En el ejemplo de la *Figura* 23 se considera un filtro 2X2 y una zancada de 2 (a diferencia del ejemplo anterior figura 21, este se mueve dos pixeles). Primero, se toma la región 2X2 de la parte superior izquierda de la imagen de salida del proceso de convolución anterior. Aplicando la agrupación máxima, se calcula el valor máximo de cada valor del bloque 2X2. Este valor se almacena y se utiliza para completar la salida completa de la operación de agrupación máxima.

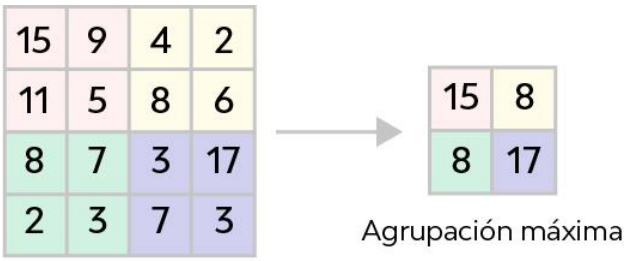


Figura 23. Agrupación máxima

Los tiempos de cálculo no se ven afectados cuando la red se enfrenta a una imagen de mayor dimensión porque la agrupación reduce la resolución de salida de las capas

convolucionales. Estas no pierden las propiedades importantes de la imagen, esta extrae la información más dominante manteniendo el proceso de entrenamiento eficaz.

3.5 *Capas completamente conectadas (Fully connected layers)*

Las capas completamente conectadas (*fully conected*) involucran pesos, sesgos y neuronas, donde cada píxel se considera una neurona separada que se conecta con neuronas de otra capa. Es la capa final de la red y es utilizada como un clasificador de imágenes entre diferentes categorías parta el entrenamiento. En la *Figura* 24 se muestra la estructura general de las capas completamente conectadas. Cada circulo se corresponde a una neurona y las neuronas se encuentran agrupadas en capas y conectadas entre ellas.

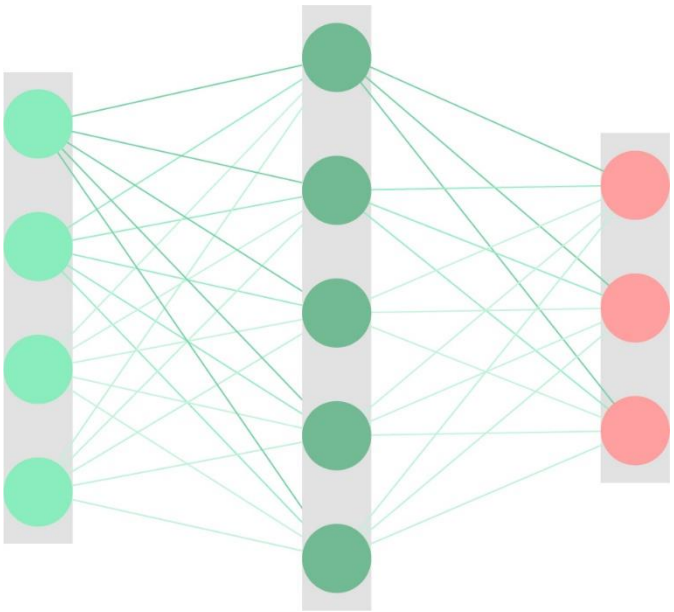


Figura 24. Capas completamente conectadas

3.6 Funciones de activación

Son las que transmiten la información generada por la combinación lineal de los procesos anteriores, es decir son la manera de transmitir la información por las conexiones de salida. La información puede transmitirse sin modificaciones, función identidad, o bien puede no transmitirse. Cada una de las capas que conforman la red neuronal tienen una función de activación que permitirá reconstruir o predecir. Como el objetivo es que la red neuronal sea capaz de resolver problemas cada vez más complejos, las funciones de activación generalmente harán que los modelos sean no lineales.

3.6.1 Función ReLU

ReLU es una función de activación de los nodos ocultos presentes en las redes para producir resultados más deseables. como se ve en el esquema de la figura 11 cada una de las capas tienen la función de activación que permite la reconstrucción y la predicción de la información.

ReLU es la función más usada en el mundo del DL, sobre todo en la clasificación de imágenes. Su fórmula matemática es la siguiente:

$$f(x) = \max(0, x) \qquad (2)$$

Dado un valor x, si dicho valor es menor que 0, conviértelo a 0, y si es mayor, devuelve el mismo valor. Esto en los algoritmos de DL quiere decir que siempre que una neurona genere una salida negativa, se desactiva dicha neurona.

La *Figura* 25 representa la gráfica de la función ReLU.

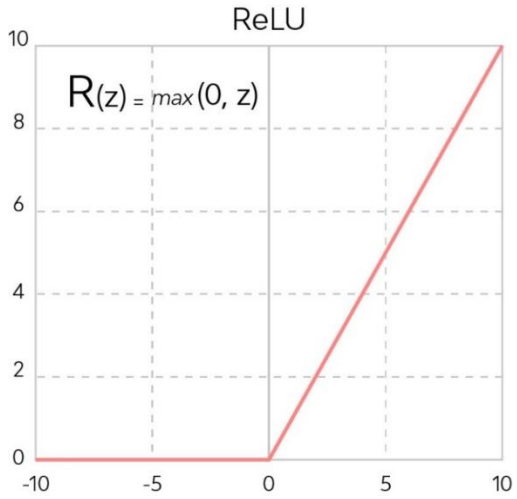


Figura 25. función ReLU

Una de las grandes ventajas de la función ReLU es su simplicidad y por tanto rapidez computacional, agilizando los procesos de entrenamiento y de predicción de las redes neuronales.

3.6.2 *Función Softmax*

En las arquitecturas de muchas redes neuronales también se encuentran activadores como el *Softmax* que al igual que el ReLU es una función de activación, pero para tareas múltiples. Esta función se encuentra generalmente en la última capa, ayuda a que los modelos tengan la capacidad de poder manejar datos no lineales y es necesaria para predecir las clases de entrada de las imágenes.

Sirve para cuando se tienen múltiples clases y se quiere extraer la probabilidad de pertenecer a las distintas clases posibles. Por ello, la salida de una capa *softmax* representa la probabilidad de reconocimiento de objetos. La ecuación 3 representa la función *softmax,* que normaliza el resultado entre 0 y 1.

$$p(y^i = n|x^i; w) = \frac{1}{\sum_{j=1}^{n} e^{w_j^T x(i)}} \begin{bmatrix} e^{w_1^T x(i)} \\ e^{w_1^T x(i)} \\ \vdots \\ e^{w_n^T x(i)} \end{bmatrix} \quad (3)$$

Donde *x* representa la información de entrada, *n* es el número de categorías, *w* los parámetros de peso y $i = 1 \ldots . m, w_n^T x(i)$ son los datos de entrada de la capa softmax, la *Figura* 26 muestra la gráfica de la función Softmax.

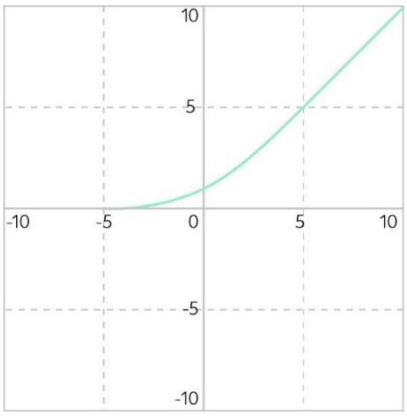


Figura 26. Función Sofmax

En la Figura 27 se muestra el diagrama de flujo para la predicción de fisuras usando la arquitectura de las CNN vistas en los apartados anteriores. El proceso comienza con la construcción de una base de datos de imágenes que se divide en grupos de imágenes para el entrenamiento, la prueba y la validación, tal y como se explicó en el apartado 3.1.2. Seguidamente se entrena la CNN, esta realiza una búsqueda exhaustiva y por último clasifica el resultado.

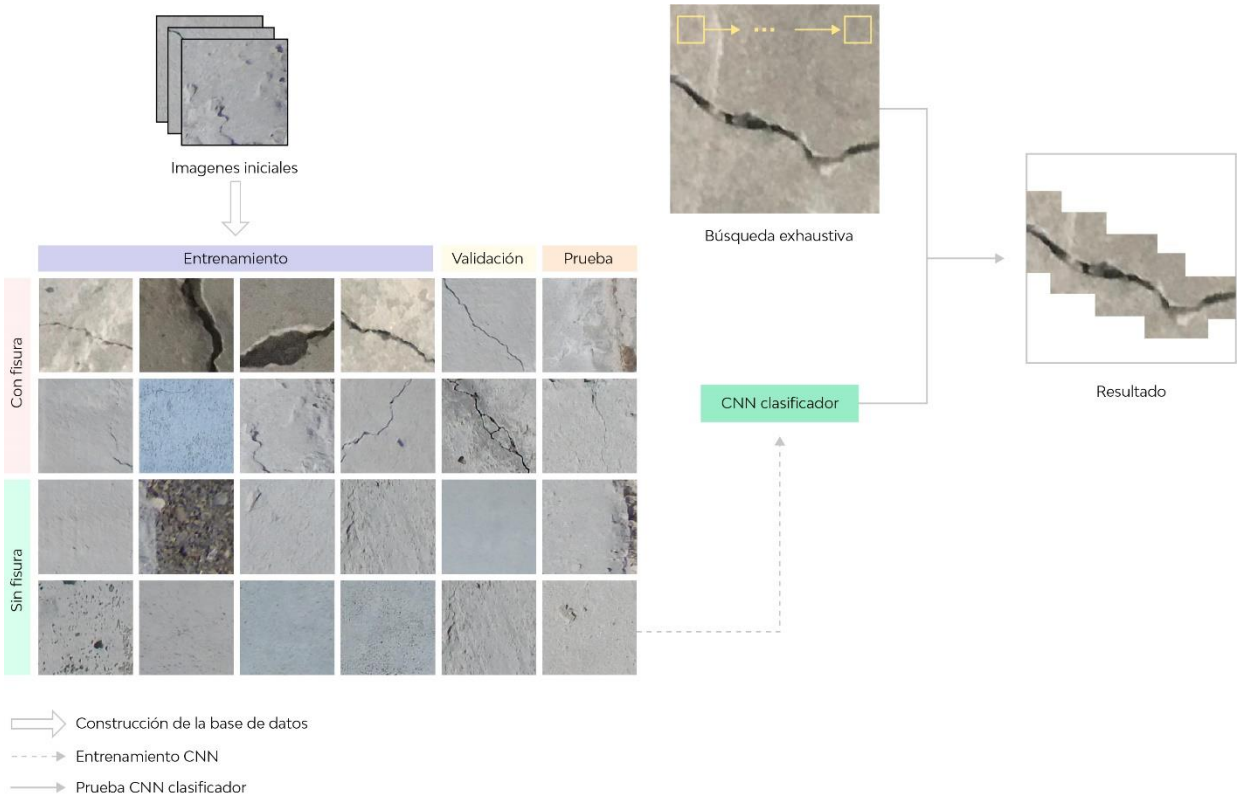


Figura 27. Diagrama de flujo para la detección de fisuras usando una CNN.

3.7 Transfer-learning

El *Transfer Learning* (TL) ofrece enormes ventajas en términos de tiempo y precisión. Siendo el TL una técnica del DL, que consiste en utilizar como base modelos de CNN que ya tienen sus capas entrenadas para clasificar otro tipo de imágenes. Esta técnica es especialmente útil cuando no se cuenta con una base de datos grande, porque la recolección de imágenes puede ser una tarea dispendiosa y costosa.

La definición general de la transferencia de aprendizaje se explica en la Figura 28. (Pan & Yang, 2010). Dado un dominio de origen (A) con su tarea de aprendizaje (A) y un dominio de destino (B) con su tarea de destino (B), el objetivo del TL es ayudar a mejorar la predicción en el aprendizaje de la tarea de destino (B) utilizando el conocimiento del dominio de origen (A) y el destino de origen (A).

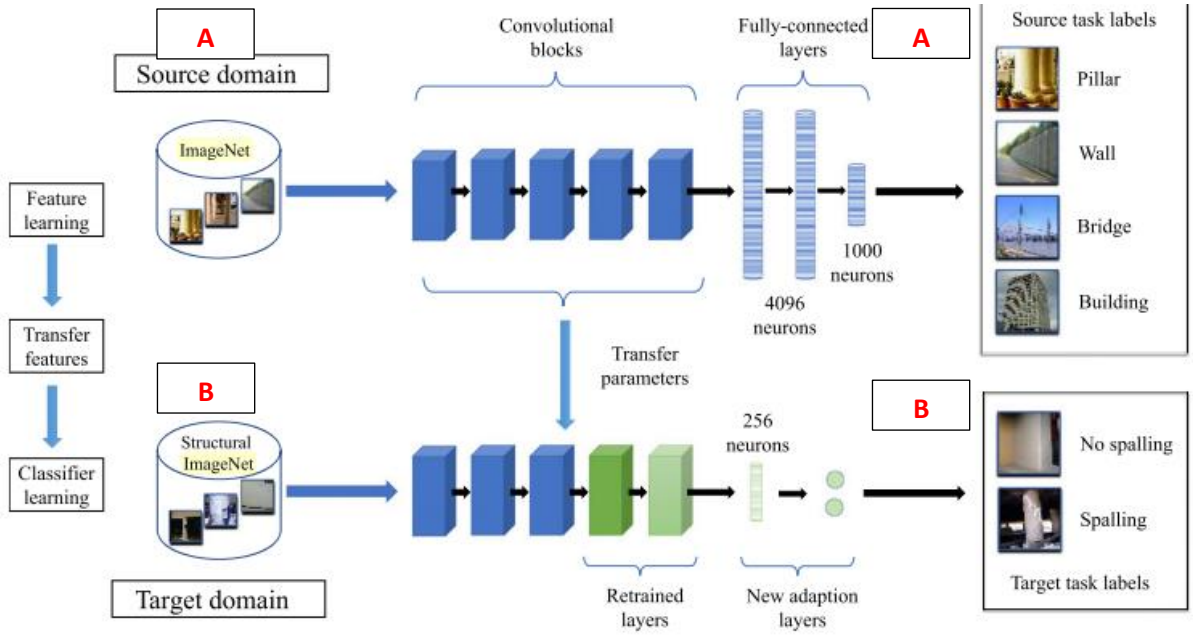


Figura 28. Diagrama del Transfer Learning (Gao & Mosalam, 2018)

La profundidad de las redes neuronales, entendida como la cantidad de capas que tienen las redes, está directamente relacionada con el número de parámetros de entrenamiento y demanda una gran cantidad de tiempo e información. Por ello, el empleo de modelos pre entrenados minimiza el tiempo de proceso de las tareas al transferir conocimiento previo obtenido de un problema similar.

Las redes que son precisas cuando se emplean con las imágenes de la base de datos *ImageNet* también suelen ser precisas cuando se aplican a otros conjuntos de datos de imágenes utilizando el TL. Esta generalización es posible porque las redes han aprendido a extraer características poderosas e informativas de imágenes naturales que se generalizan a otros conjuntos de imágenes similares. Sin embargo, la alta precisión en *ImageNet* no siempre se transfiere directamente a otras tareas, por lo que es una buena idea probar varias redes (Beale et al., 2020)

En este TFM se utilizan tres arquitecturas de redes neuronales preentrenadas (*AlexNet, GoogleNet y SqueezeNet*). El *Helpcenter* de Matlab indica que las características importantes a la hora de elegir una CNN son: la precisión, la velocidad, el tamaño de la red y el espacio que utilizan en el disco. La elección de una red es generalmente un equilibrio entre estas características. La Figura 29 ilustra esta idea pues compara la precisión de la validación de la base de datos de *ImageNet* con el tiempo necesario para hacer una predicción utilizando la red. El tamaño del marcador en la figura 29 es proporcional a la memoria que gasta la red en el disco.

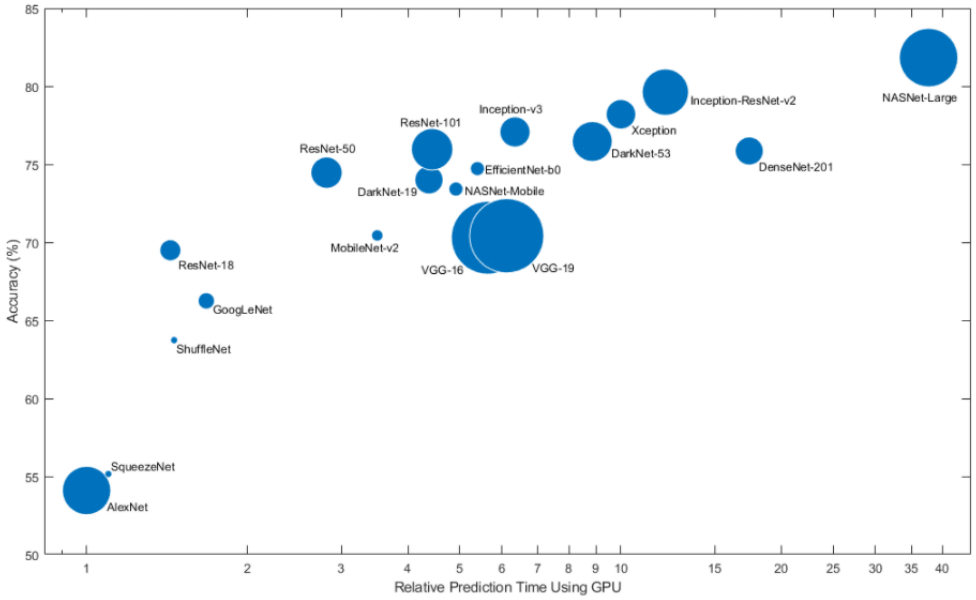


Figura 29. Indicador de velocidad relativa vs precisión de las redes neuronales (Matlab Helpcenter) https://es.mathworks.com)

3.7.1 Arquitectura AlexNet

Es uno de los primeros modelos existentes del DL y fue desarrollado para la clasificación de objetos en imágenes. Fue la arquitectura ganadora de la competición de *ImageNet* en el 2012, competencia que se realizó desde el 2010 hasta el 2017 en

la que las tareas de desafío utilizan subconjuntos de la base de datos de ImageNet. Su arquitectura tiene 5 capas convolucionales con *max pooling*, tres capas completamente conectadas y la función de activación *softmax,* en total la red cuenta con 25 capas. Esta red se ha ajustado para detección de fisuras tal como se hizo en los trabajos de (H. Kim et al., 2017)y (B. Kim & Cho, 2018)

La arquitectura comprende cinco capas de convolución (C1-C5), tres capas de agrupación máxima (MP1-MP3), siete capas de no linealidad que utilizan la función de unidad lineal rectificada (ReLU) (ReLU1-ReLU7), dos capas de normalización (Norm1 — Norm2), tres capas completamente conectadas (FC1 — FC3), dos capas de abandono (DP1 — DP2), una capa softmax (SM) y una capa de clasificación (CL). Cada capa se aplica a la imagen mediante la operación de convolución. En las Figs.30 y 31 se muestra la arquitectura de AlexNet junto con su número y tamaño de filtro correspondiente. Los valores del kernel se determinan iterativamente a través del entrenamiento, pero el tamaño, el número y el paso de los kernels están predeterminados. Las capas de no linealidad operan sobre el resultado de cada capa de convolución a través de la comparación de elementos. La función ReLU utilizada para la no linealidad se define como el valor máximo de cero.

La capa de no linealidad, una capa de agrupación máxima introduce un representante para un conjunto de píxeles vecinos tomando su valor máximo. Las capas de agrupación máxima son esenciales para reducir el tiempo de cálculo y los problemas de sobreajuste en la CNN” (Dorafshan et al., 2018)

En la *Figura* 30 se muestra la configuración de las capas de la arquitectura de AlexNet.

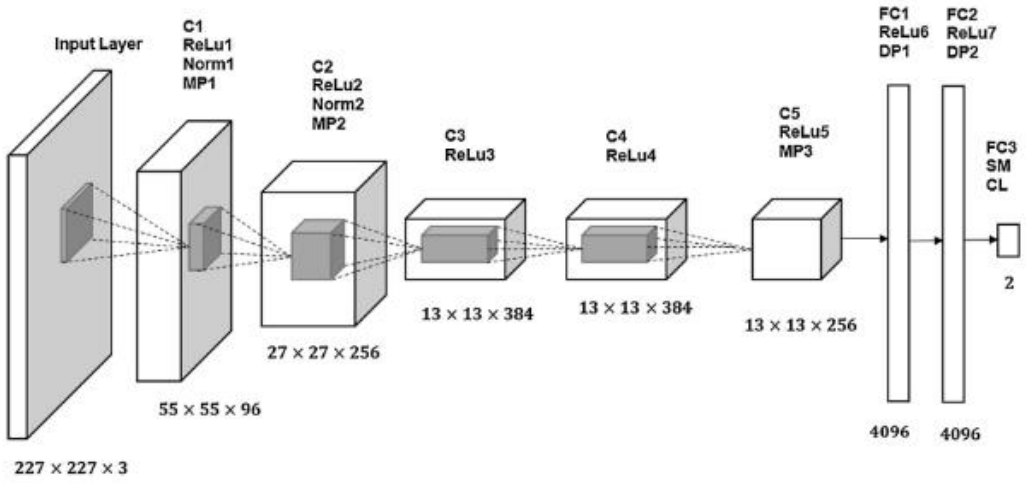


Figura 30. Arquitectura AlexNet (Dorafshan et al., 2018)

3.7.2 Arquitectura GoogleNet

GoogleNet es una red neuronal convolucional que tiene 22 capas de profundidad. En el *Toolbox* de MATLAB se puede cargar una versión pre entrenada de la red con la base de datos *ImageNet.* La red entrenada en *ImageNet* clasifica las imágenes en 1000 categorías de objetos. Estas redes han aprendido diferentes representaciones de características para una amplia gama de imágenes. Las redes previamente entrenadas tienen un tamaño de entrada de imagen de 224 por 224 pixeles. En las figuras 31 y 32 se puede ver los pesos y tipos de cada capa y por último el esquema general de todas sus capas.

Todas las convoluciones, incluidas las que se encuentran dentro de los módulos, utilizan activación lineal rectificada (ReLU). El tamaño del campo receptivo en nuestra red es 224 × 224 tomando canales de color RGB con sustracción media. "# 3 × 3 reduce" y "# 5 × 5 reduce" representa el número de filtros 1 × 1 en la capa de reducción utilizada antes de las convoluciones 3 × 3 y 5 × 5. Se puede ver el número de filtros 1 × 1 en la capa de proyección después de la agrupación máxima incorporada en la

columna del *pool project*. Todas estas capas de reducción / proyección utilizan también activación lineal rectificada (ver figura 31).

La red se diseñó teniendo en cuenta la eficiencia computacional y la practicidad, de modo que la inferencia se pueda ejecutar en dispositivos individuales, incluso aquellos con recursos computacionales limitados, especialmente con poca huella de memoria. La red tiene 22 capas de profundidad cuando se cuentan solo las capas con parámetros (o 27 capas si también contamos la agrupación). El número total de capas (bloques de construcción independientes) que se utilizan para la construcción de la red es de aproximadamente 100. Sin embargo, este número depende del sistema de infraestructura de aprendizaje automático utilizado. (Szegedy et al., 2015)

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Figura 31. Arquitectura de GoogleNet (Szegedy et al., 2015)

Figura 32. Arquitectura de GoogleNet (Szegedy et al., 2015)

3.7.3 Arquitectura SqueezeNet

SqueezeNet es una red neuronal convolucional que tiene 18 capas de profundidad y un total de 68 capas. En el *Toolbox* de MATLAB se puede cargar una versión pre entrenada de la red con la base de datos *ImageNet*. La red entrenada en *ImageNet* clasifica las imágenes en 1000 categorías de objetos. Estas redes han aprendido diferentes representaciones de características para una amplia gama de imágenes. Las redes previamente entrenadas tienen un tamaño de entrada de imagen de 227 por 227 pixeles. En las figuras 34 y 35 se puede ver los pesos de cada capa y por último el esquema general de todas sus capas.

La arquitectura de SqueezeNet se compone de capas de "compresión (*squeeze*) " y "expansión (*expand*)". Una capa convolucional comprimida tiene solo filtros 1 × 1. Estos se introducen en una capa de expansión que tiene una mezcla de filtros de convolución de 1 × 1 y 3 × 3. Esto se muestra a continuación.

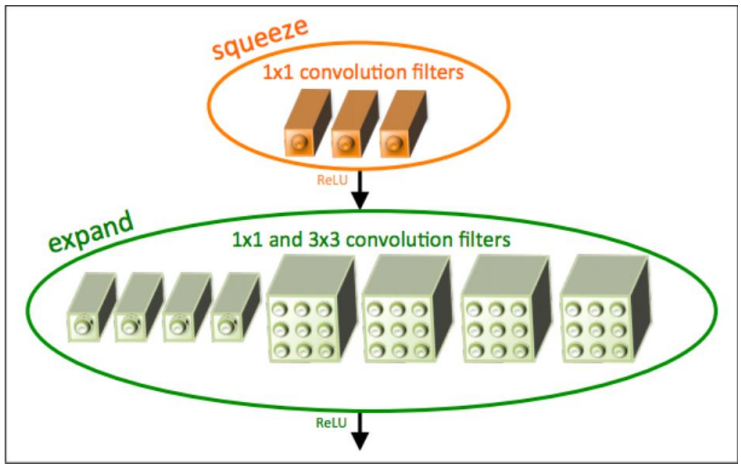


Figura 33. Organización de los filtros convolucionales de SqueezeNet (Iandola et al., 2016)

Los autores de la arquitectura de SqueezeNet utilizan el término “*fire module*” para describir una capa de compresión y una de expansión juntas.

La imagen de entrada se envía primero a una capa convolucional independiente. A esta capa le siguen 8 "módulos fire " que se denominan "fire2-9”. La siguiente imagen muestra el proceso de SqueezeNet.

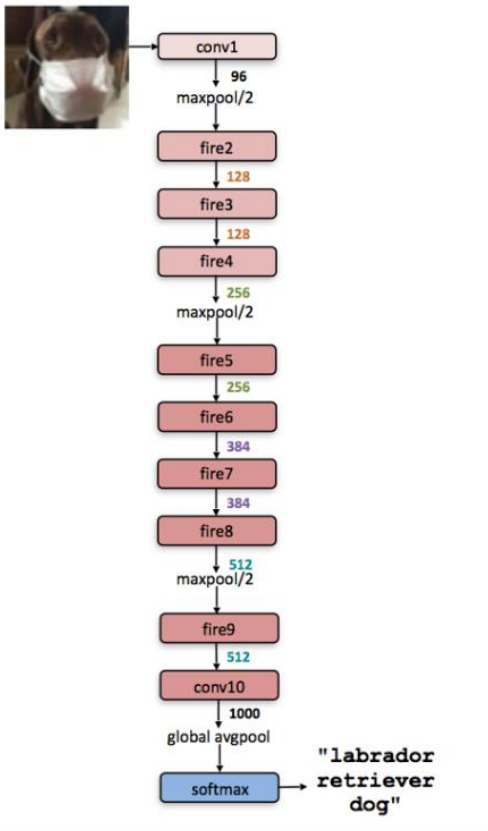


Figura 34. Vista de la arquitectura de SqueezeNet (Iandola et al., 2016)

SqueezeNet comienza con una capa de convolución independiente (conv1), seguida de 8 módulos *Fire (fire2-9)*, y termina con una capa de convolución final (conv10). Se aumenta gradualmente el número de filtros por módulo *fire* desde el principio hasta el final de la red. SqueezeNet realiza un agrupamiento máximo con un paso de 2 después de las capas conv1, fire4, fire8 y conv10". (Iandola et al., 2016)

layer name/type	output size	filter size / stride (if not a fire layer)	depth	s_{1x1} (#1x1 squeeze)	e_{1x1} (#1x1 expand)	e_{3x3} (#3x3 expand)	s_{1x1} sparsity	e_{1x1} sparsity	e_{3x3} sparsity	# bits	#parameter before pruning	#parameter after pruning
input image	224x224x3										-	-
conv1	111x111x96	7x7/2 (x96)	1				100% (7x7)			6bit	14,208	14,208
maxpool1	55x55x96	3x3/2	0									
fire2	55x55x128		2	16	64	64	100%	100%	**33%**	6bit	11,920	5,746
fire3	55x55x128		2	16	64	64	100%	100%	**33%**	6bit	12,432	6,258
fire4	55x55x256		2	32	128	128	100%	100%	**33%**	6bit	45,344	20,646
maxpool4	27x27x256	3x3/2	0									
fire5	27x27x256		2	32	128	128	100%	100%	**33%**	6bit	49,440	24,742
fire6	27x27x384		2	48	192	192	100%	**50%**	**33%**	6bit	104,880	44,700
fire7	27x27x384		2	48	192	192	**50%**	100%	**33%**	6bit	111,024	46,236
fire8	27x27x512		2	64	256	256	100%	**50%**	**33%**	6bit	188,992	77,581
maxpool8	13x12x512	3x3/2	0									
fire9	13x13x512		2	64	256	256	**50%**	100%	**30%**	6bit	197,184	77,581
conv10	13x13x1000	1x1/1 (x1000)	1				**20%** (3x3)			6bit	513,000	103,400
avgpool10	1x1x1000	13x13/1	0									
	activations			parameters			compression info				1,248,424 (total)	**421,098** (total)

Figura 35. Dentro de la arquitectura de SqueeezeNet (Iandola et al., 2016)

3.8 Fine-tuning

Una mayor afinación o *fine-tuning* es una forma eficiente de ajustar la red existente a una nueva red con tareas de clasificación. El *fine-tuning* (FT) se puede llevar a cabo con un bajo rendimiento computacional y con relativamente pocos datos. Cuando se trabajan redes neuronales convolucionales con capas en serie y con capas completamente conectadas para la salida, es común que algunas capas convolucionales se “congelen” durante el entrenamiento y las ultimas capas completamente conectadas se reemplacen.

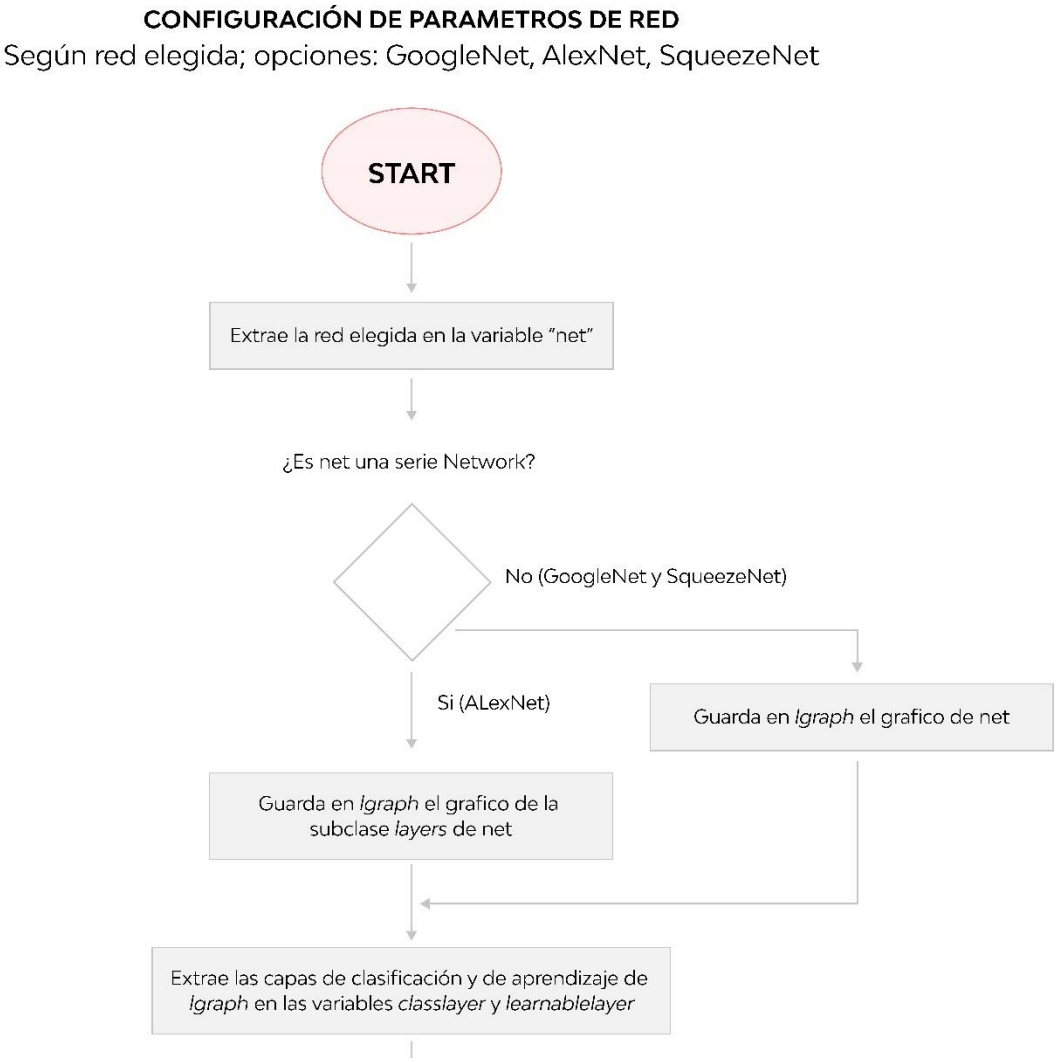
Los pesos de las primeras capas se “congelan” estableciendo los parámetros de aprendizaje de esas capas en “Cero”, es decir esas capas no pueden aprender nada durante el entrenamiento. Debido a que no es necesario calcular los gradientes de las capas congeladas, hacer esto en las capas iniciales puede acelerar significativamente el entrenamiento de la red. Si el nuevo conjunto de datos es pequeño, congelar las capas también puede evitar que esas capas se sobreajusten al nuevo conjunto de datos.

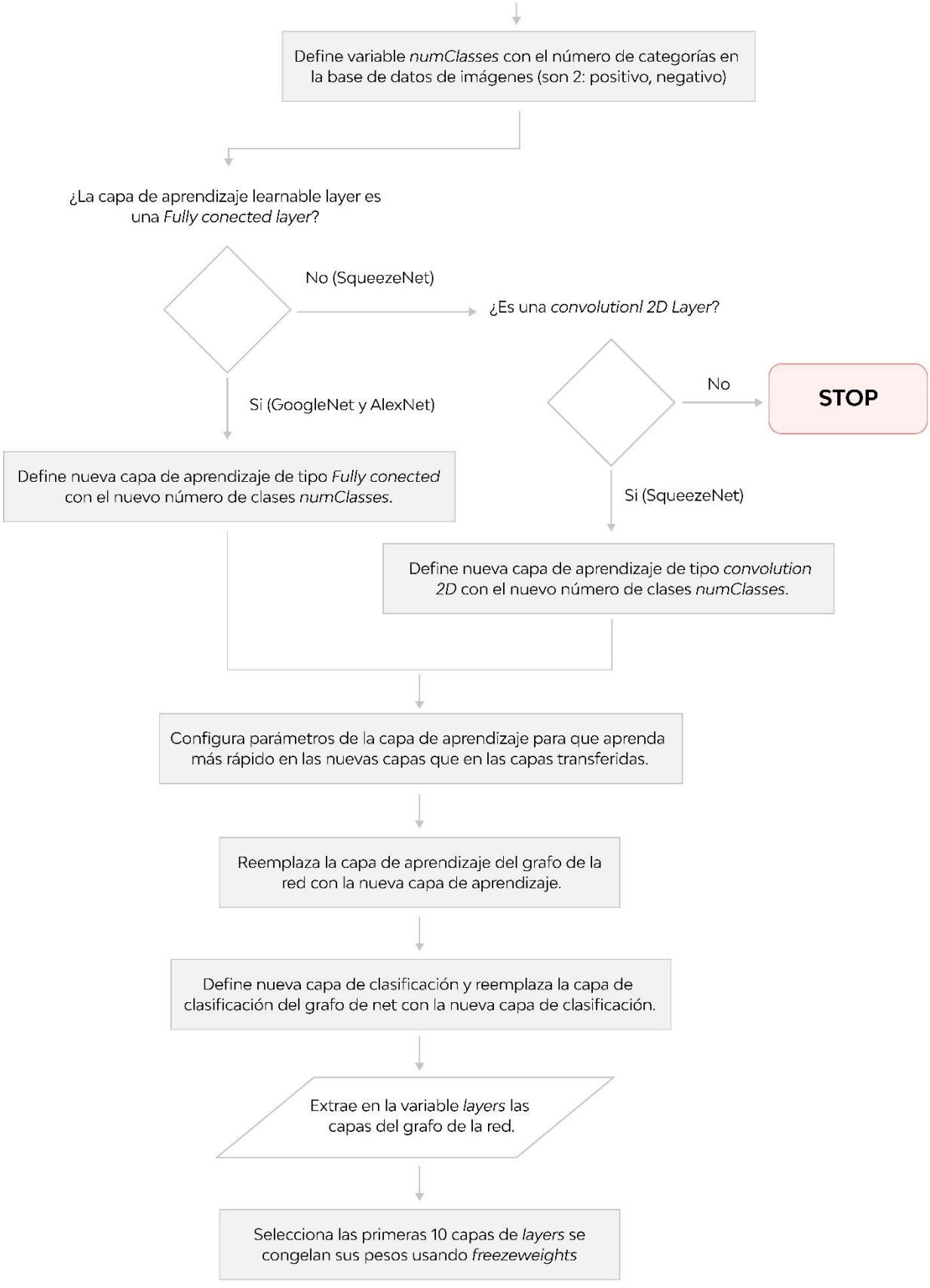
Se utilizó la función de apoyo *freezeWeights* en el código de MATLAB para establecer las tasas de aprendizaje en cero en las primeras 10 capas de las redes. Posteriormente se utilizó la función de *createLgraphUsingConnections* para volver a conectar todas las capas en el orden original. El nuevo gráfico de capa contiene las mismas capas, pero con las tasas de aprendizaje de las capas anteriores establecidas en cero.

Las capas, “*loss3-classifier*” y “*ouput*” en las redes, contienen información sobre cómo combinar las características que la red extrae en probabilidades de clase, un valor de pérdida y etiquetas de predicción. Para reentrenar una red pre entrenada para clasificar nuevas imágenes, estas capas se reemplazan por nuevas capas adaptadas al nuevo conjunto de datos.

La última capa con pesos que se pueden entrenar es una capa completamente conectada. Se reemplaza la capa completamente conectada de la red por una nueva capa completamente conectada con el número de salidas igual al número de clases del nuevo conjunto de datos, en este trabajo serian solo 2 clases, imágenes con fisuras y sin fisuras (Positivas y negativas).

En algunas redes, como *SqueezeNet*, la última capa que se puede aprender es una capa convolucional de 1 por 1. En este caso, se reemplaza la capa convolucional con una nueva capa convolucional con el número de filtros igual al número de clases. La *Figura* 36 muestra el diagrama de flujo de congelamiento de capas y del reemplazo de las ultimas capas de todas las redes.




Define variable *numClasses* con el número de categorías en la base de datos de imágenes (son 2: positivo, negativo)
¿La capa de aprendizaje learnable layer es una *Fully conected layer*?
No (SqueezeNet)
¿Es una *convolutionl 2D Layer*?
No
STOP
Si (GoogleNet y AlexNet)
Si (SqueezeNet)
Define nueva capa de aprendizaje de tipo *Fully conected* con el nuevo número de clases *numClasses*.
Define nueva capa de aprendizaje de tipo *convolution 2D* con el nuevo número de clases *numClasses*.
Configura parámetros de la capa de aprendizaje para que aprenda más rápido en las nuevas capas que en las capas transferidas.
Reemplaza la capa de aprendizaje del grafo de la red con la nueva capa de aprendizaje.
Define nueva capa de clasificación y reemplaza la capa de clasificación del grafo de net con la nueva capa de clasificación.
Extrae en la variable *layers* las capas del grafo de la red.
Selecciona las primeras 10 capas de *layers* se congelan sus pesos usando *freezeweights*

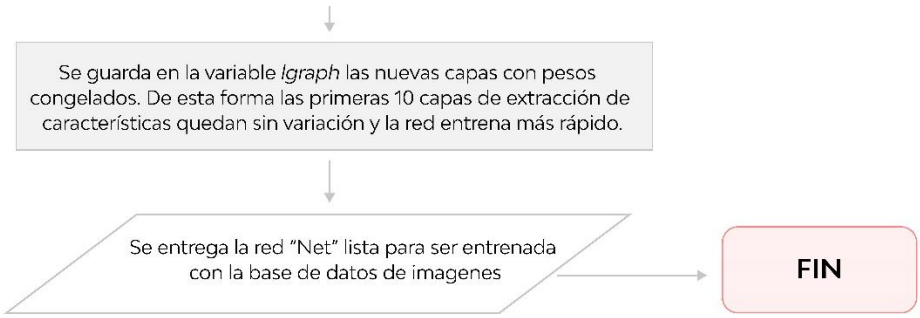


Figura 36. Diagrama de flujo de *fine tunning*.

La primera capa la red pre entrenada ha aprendido filtros para capturar características de reconocimiento de regiones, gradientes y de detección de borde. Estas características "primitivas" son luego procesadas por capas de red más profundas que combinan las características iniciales para formar características de imagen de mayor nivel. Estas características de nivel superior son más adecuadas para tareas de reconocimiento, porque combinan todas las características primitivas en una representación de imagen más rica.(Donahue et al., n.d.).

La *Figura* 37 muestra el modelo jerárquico que visualiza las sucesivas fases en el proceso de entrenamiento. Desde el nivel bajo, el nivel medio hasta el nivel alto, las características de cada capa (conjunto de neuronas) se muestran en diferentes componentes.

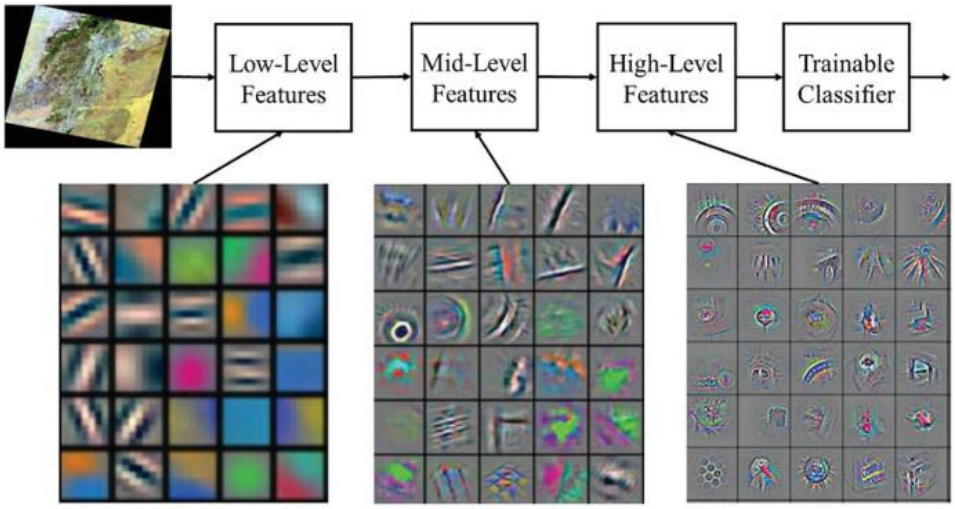


Figura 37. Jerarquía del proceso de entrenamiento de CNN (Traore et al., 2018)

Los pesos de la primera capa convolucional congeladas de las 3 redes se muestran a continuación. (figuras 38, 39 y 40).

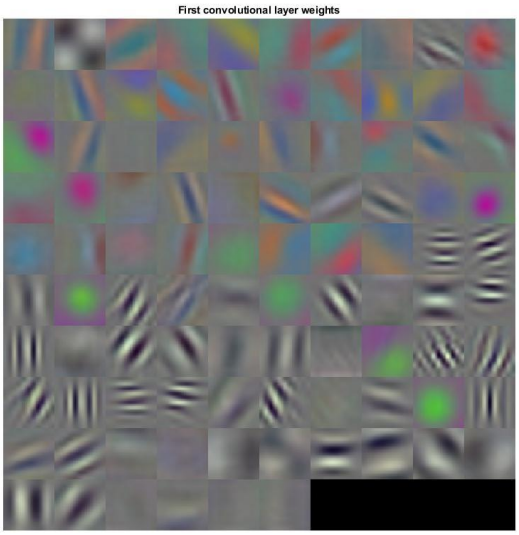


Figura 38. Pesos de la primera capa convolucional AlexNet

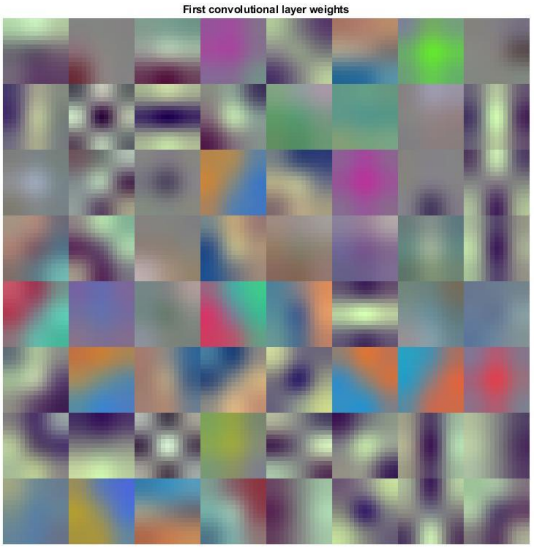


Figura 39. Pesos de la primera capa convolucional SqueezeNet

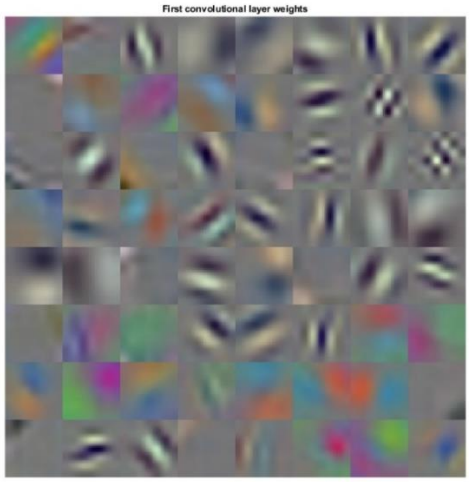


Figura 40. Pesos de la primera capa convolucional GoogleNet

En la Figura 41, se muestra el esquema del congelamiento de los primeros bloques de las redes y la modificación de las ultimas capas de clasificación. En este esquema a la CNN ingresa la base de datos de entrenamiento, que pasa por las capas congeladas; estas mismas extraen la información inicial de características como enfoque, patrones, detección de bordes, etc. Al pasar a las capas de reentrenamiento la red empieza a aprender y extraer características propias de las imágenes de la base

de datos. Por último, en las últimas neuronas se modifican el número de salidas con la cantidad de tareas de clasificación habiéndose adoptado en este trabajo dos grupos, positivos y negativos.

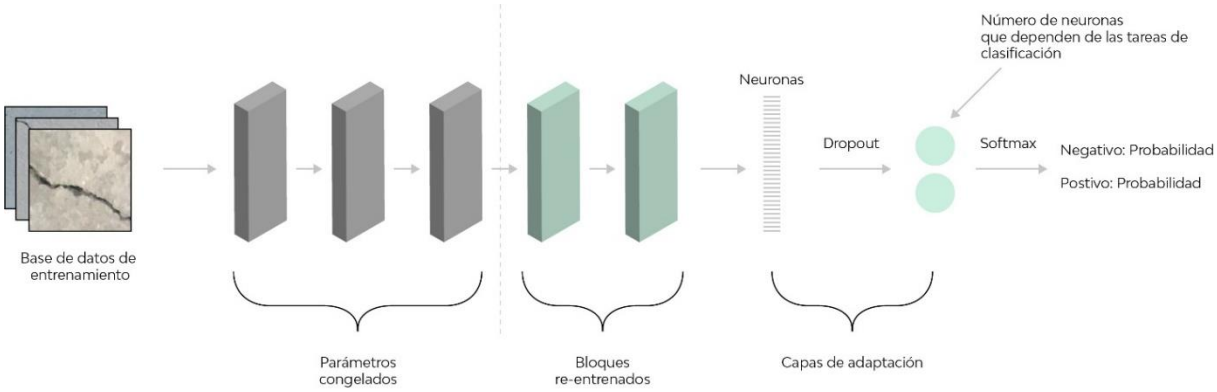


Figura 41. Esquema del fine-tuning en una CNN.

4 EXPERIMENTOS Y RESULTADOS

En este apartado se mostrarán los resultados del proceso de entrenamiento de las tres redes neuronales elegidas en el apartado anterior y se escogerá la CNN con mejores resultados para posteriormente probar la herramienta en la etapa final.

4.1 Configuración del entorno del software

El experimento se realizó en un ordenador con un Intel® core i7-8750H CPU @ 2.20GHz, 16 GB de memoria aleatoria y una GPU NVIDIA GTX 1050 Ti con 4 GB de memoria y sistema operativa Windows 10. Se utilizaron los modelos pre-entrenados del Toolbox de MATLAB para entrenar los parámetros de red para las dos categorías de detección. El software utilizó la aceleración del GPU.

En entrenamiento de redes profundas convolucionales tiene un importante coste computacional que se puede reducir con el entrenamiento en paralelo a través de través de CPUs multinúcleo, GPUs de alto rendimiento y clústeres con múltiples CPUs y GPUs. El uso de GPU u opciones paralelas requiere del empleo de la Parallel Computing Toolbox de Matlab. (Beale et al., 2020)

En la Tabla 3 se muestra el hardware y las consideraciones de memoria del ordenador al igual que el software empleado para el proceso de entrenamiento de las CNN.

Hardware de aprendizaje profundo y Consideraciones de memoria	**Recomendaciones**	**Productos requeridos**
GPU	De forma predeterminada, la función trainNetwork usa una GPU si está disponible. Requiere una GPU compatible dispositivo.	*MATLAB *Deep Learning Toolbox *Parallel Computing Toolbox

Tabla 3. Productos y recomendaciones para entrenamiento (Beale et al., 2020)

4.2 Entrenamiento y detección de fisuras

En este apartado se entrenará la red con las bases de datos que se prepararon en el apartado 3.1.

Durante el entrenamiento de la red neuronal, el número de imágenes utilizadas en cada actualización se llama tamaño de lote o *batch size*, cada actualización completa de un *batch size* se llama iteración, y cada actualización completa de toda la base de datos se llama *epoch* o época.

A continuación (Figura 42), se muestra el conjunto de opciones con las cuales se configura la etapa de entrenamiento de las redes neuronales:

```
options =

  TrainingOptionsSGDM with properties:

                     Momentum: 0.9000
            InitialLearnRate: 3.0000e-04
           LearnRateSchedule: 'none'
         LearnRateDropFactor: 0.1000
         LearnRateDropPeriod: 10
            L2Regularization: 1.0000e-04
    GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                  MaxEpochs: 8
              MiniBatchSize: 64
                    Verbose: 0
            VerboseFrequency: 50
              ValidationData: [1×1 augmentedImageDatastore]
         ValidationFrequency: 69
          ValidationPatience: Inf
                     Shuffle: 'every-epoch'
              CheckpointPath: ''
       ExecutionEnvironment: 'auto'
                 WorkerLoad: []
                  OutputFcn: []
                      Plots: 'training-progress'
             SequenceLength: 'longest'
       SequencePaddingValue: 0
    SequencePaddingDirection: 'right'
        DispatchInBackground: 0
     ResetInputNormalization: 1
```

Figura 42. Configuración etapa de entrenamiento de las redes neuronales.

Se estableció el número de épocas de entrenamiento en 8, un valor que se determinó empíricamente ya que durante el entrenamiento se podía observar que después de la época 6 el proceso se estabilizaba. Se estableció la tasa de aprendizaje en 0.003, un valor pequeño para ralentizar el aprendizaje en las capas de transferencia, y el tamaño del lote en 64, estos valores fueron determinados empíricamente y por medio de recomendaciones de redes ya entrenadas.

Después del entrenamiento se puede observar el porcentaje de precisión de la red neuronal. La gráfica de la función de pérdida representa la diferencia entre un valor predicho y el valor correcto, durante el procedimiento, la función de pérdida se minimiza continuamente actualizando repetidamente todos los parámetros de la red (figura 43 y 44)

La entropía cruzada (Figura 44) caracteriza la distancia entre la salida real (probabilidad) y la salida esperada (probabilidad). Es decir, cuanto menor es el valor de la entropía cruzada, más cercanas están las dos distribuciones de probabilidad. Cuanto mayor es la entropía cruzada, mayor es la distancia entre dos distribuciones de probabilidad y más diferentes son las dos distribuciones de probabilidad. (Feng et al., 2019) .

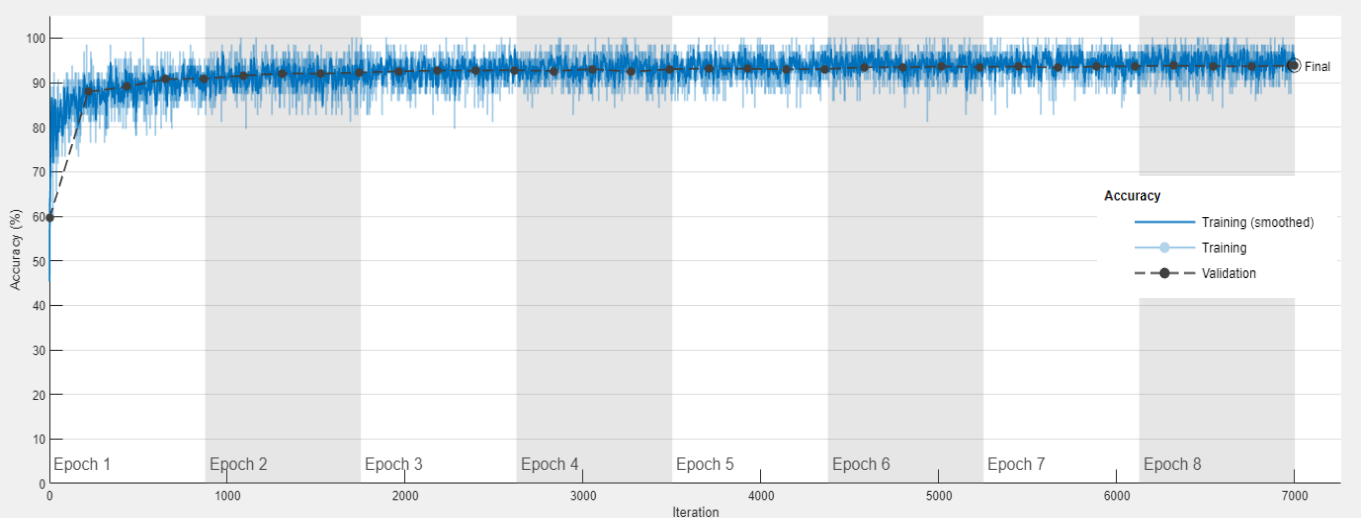


Figura 43. Ejemplo de evolución de la precisión del conjunto de datos de validación con cada época

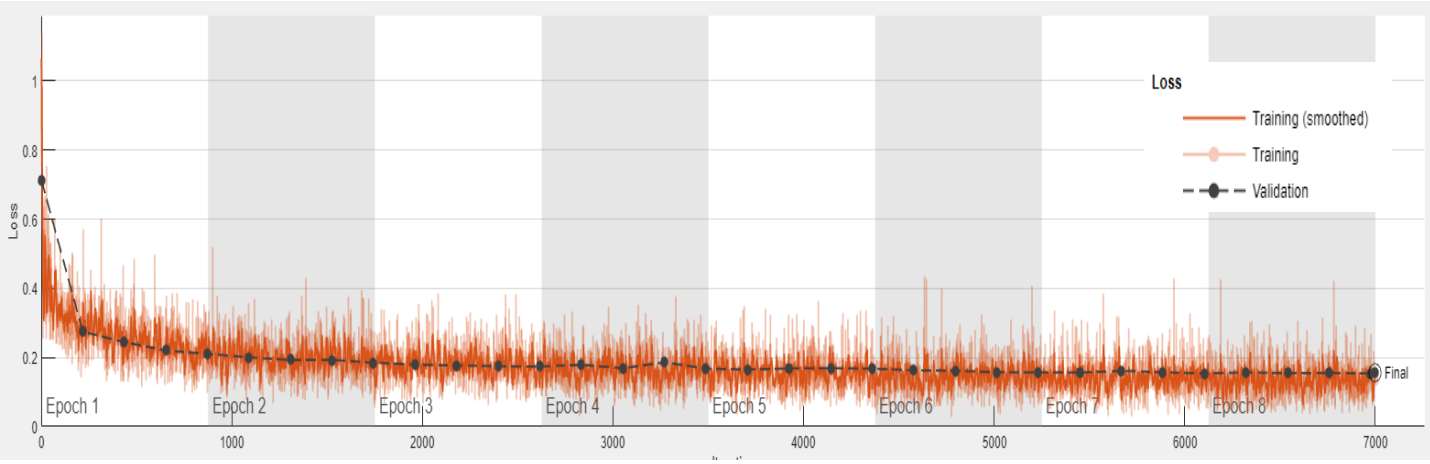


Figura 44. Pérdida de entropía cruzada del conjunto de datos en cada época

Se han empleado los siguientes indicadores para medir la precisión de la red neuronal en la detección de fisuras:

1) La tasa de verdaderos positivos (TPR) cuya fórmula se presenta en la ecuación (4), siendo (TP) un verdadero positivo, que sucede cuando la red identifica una fisura y el resultado de la predicción también es positivo. Y un falso negativo (FN) representa los resultados en que la etiqueta de la imagen es verdadera pero el resultado de la predicción es negativo.

$$TPR = \frac{TP}{TP + FN} \qquad (4)$$

2) La tasa de verdaderos negativos (TNR) cuya fórmula se presenta en la ecuación (5), los falsos positivos (FP) representa los resultados en que la etiqueta de la imagen es falsa pero el resultado de predicción es positivo. Y un verdadero negativo (TN) representa los resultados en que la etiqueta de la imagen es negativa y el resultado de la predicción es negativo.

$$TNR = \frac{TN}{TN + FP} \qquad (5)$$

3) La precisión (ACC), cuya fórmula se presenta en la ecuación (6).

$$ACC = \frac{TP + TN}{TP + FN + TN + FP} \qquad (6)$$

A continuación, se analizará los datos de entrenamiento de las tres arquitecturas mencionadas anteriormente. En un primer entrenamiento para las tres arquitecturas se usaron las imágenes de tableros de la base de datos SDNET2018 y, debido a que la proporción de distribución de la base de datos es 60:25:15, se toman 8172 imágenes para el entrenamiento y 3405 para la validación. Después del entrenamiento, se utilizan las 2043 imágenes restantes para probar la precisión del método de detección de fisuras, ver Tabla 4.

Etiquetas	**Sub carpetas**			**Total**	**BASE DE DATOS**
	Entrenamiento	**validación**	**Prueba**		
Negative	6957	2899	1739	11595	SDNET2018-TABLEROS
Positive	1215	506	304	2025	
TOTAL	**8172**	**3405**	**2043**	**13620**	

Tabla 4. Bases de datos entrenamiento 1.

Para cada proceso se analizan los indicadores mencionados anteriormente, la velocidad de entrenamiento de la red y la matriz de confusión, que permite visualizar el desempeño del algoritmo durante el aprendizaje.

La Figura 45 se muestra el resumen del proceso de entrenamiento de la arquitectura AlexNet. En la parte superior de la gráfica se muestra la precisión en la etapa de

entrenamiento y en la parte inferior la función de pérdida. A mano derecha de la gráfica está el resumen de las iteraciones, las épocas y el tiempo que ha tardado en entrenar. Para este primer entrenamiento se ha tomado un tiempo de 5 minutos y 29 segundos.

La gráfica de entropía cruzada de la parte inferior cambia significativamente y tienen una tendencia decreciente, mostrando que la red gradualmente converge y tiende a ser estable.

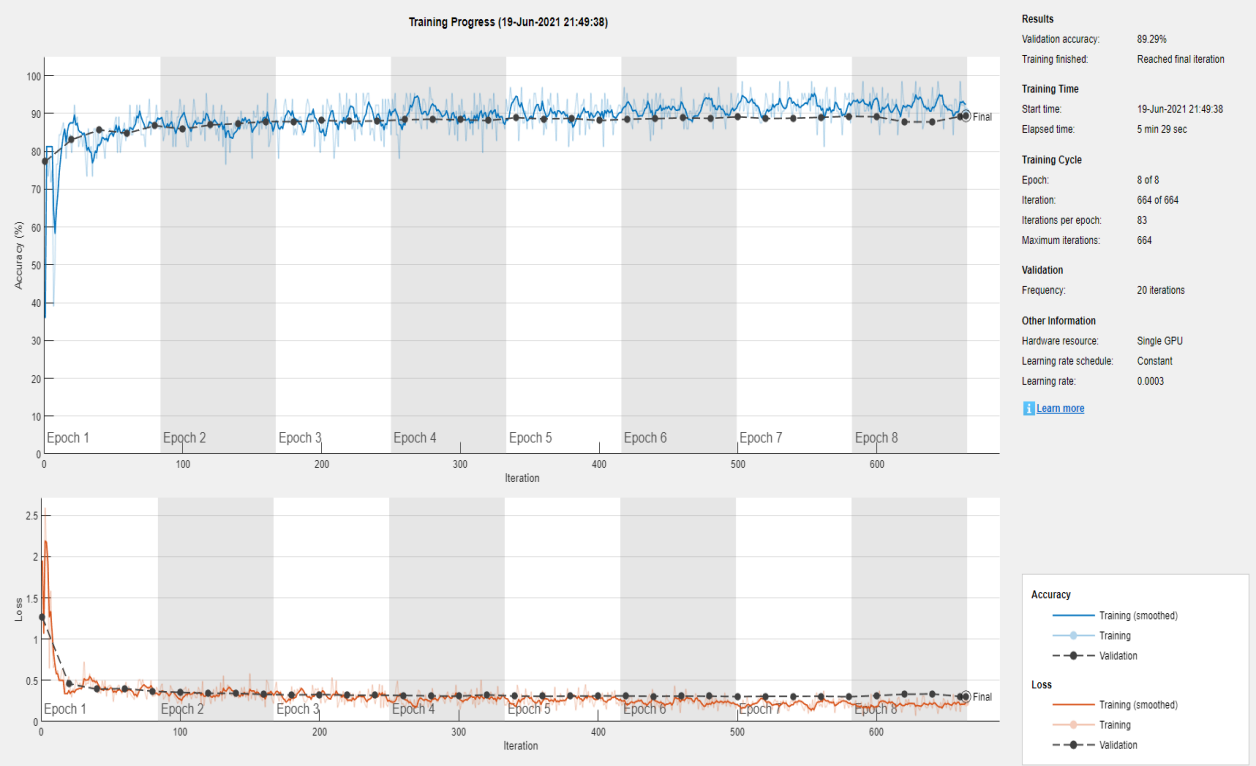


Figura 45. AlexNet proceso de entrenamiento precisión de validación 89.29%

A continuación, se muestran los indicadores del entrenamiento de AlexNet. Una tasa de verdaderos positivos de 73.7%. una tasa de verdaderos negativos de 96% y una precisión del 90.6%.

$$TPR = 0.737$$

$$TNR = 0.96$$

$$ACC = 0.906$$

La figura 46 muestra imágenes aleatorias de la fase de entrenamiento con los porcentajes alcanzados de precisión de la predicción de AlexNet. Aunque la clasificación sea binaria (negativo o positivo) la predicción de la red es en ponderados, después de una predicción por encima de 50% la clasificación se considera positiva o negativa dependiendo de la etiqueta.

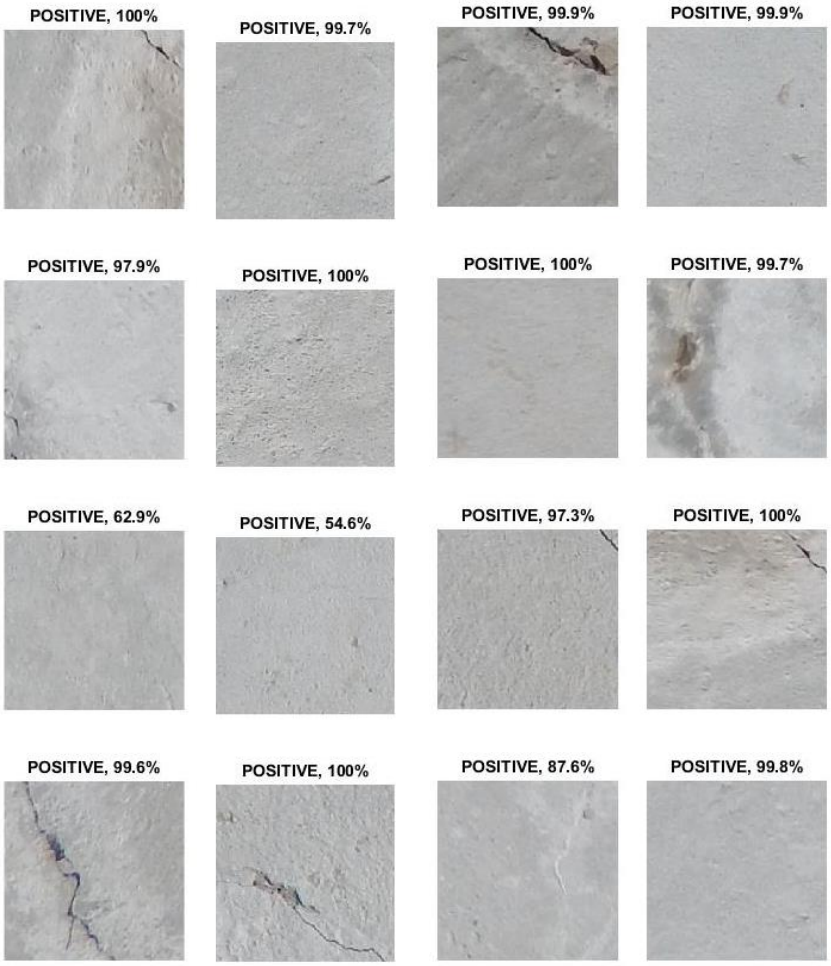


Figura 46. Porcentajes clasificación de las imágenes AlexNet

En la *Figura* 47 se muestra la matriz de confusión de ALEXNET, allí tenemos *el target Class* que es la etiqueta original de las imágenes (verdadera). El *Output Class* es la etiqueta que predice la red neuronal. La matriz de confusión muestra cuántos negativos detecta la red como negativos (993), cuantos negativos detecta como positivos (83), cuántos positivos detecta como negativo (43) y cuántos positivos detecta como positivo (221)

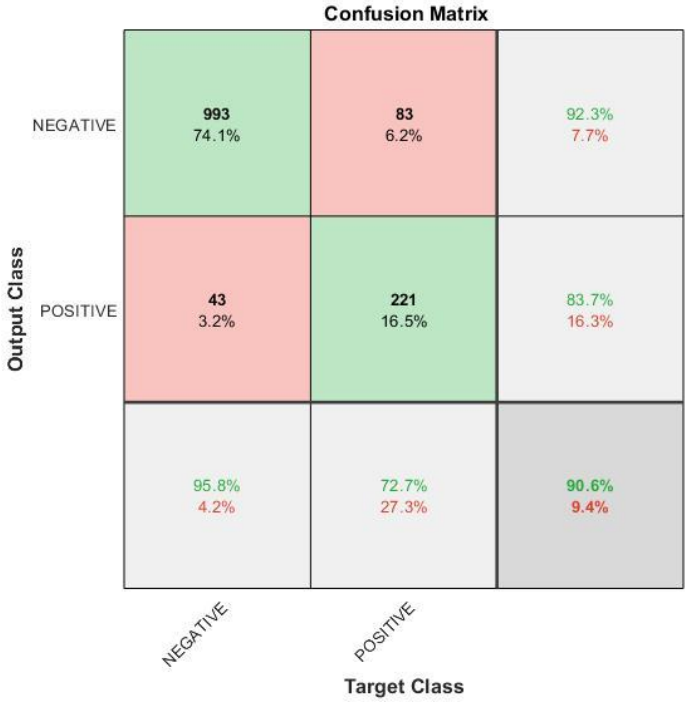


Figura 47. Matriz de confusión AlexNet primer entrenamiento

En la *Figura* 48 se muestra el resumen del proceso de entrenamiento de la arquitectura SqueezeNet. En la parte superior de la gráfica se muestra la precisión en la etapa de entrenamiento y en la parte inferior la función de pérdida. A mano derecha de la gráfica está el resumen de las iteraciones, las épocas y el tiempo que ha tardado en entrenar. Para este primer entrenamiento la red tardó 8 minutos y 6 segundos.

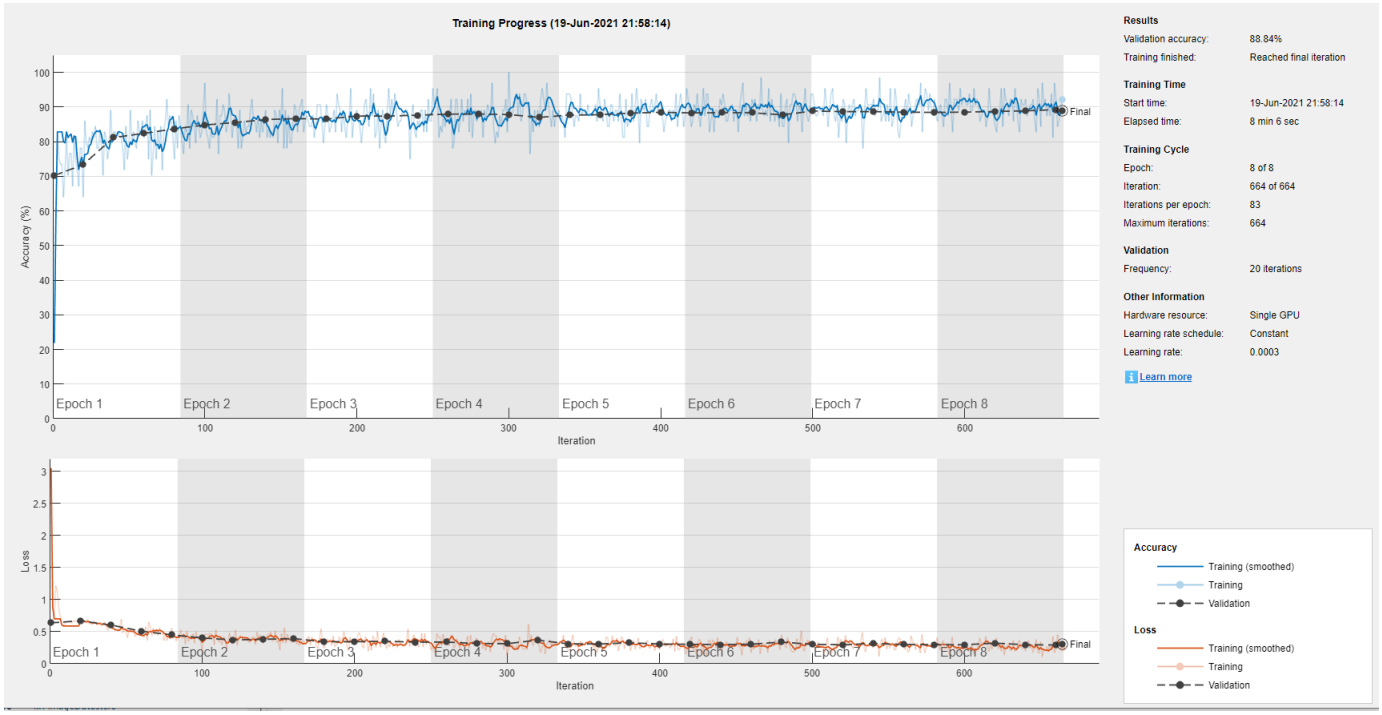


Figura 48. SqueezeNet proceso de entrenamiento precisión de validación 88.84%

A continuación, se muestran los indicadores del entrenamiento de SqueezeNet. Una tasa de verdaderos positivos de 65.1%. una tasa de verdaderos negativos de 94.3% y una precisión del 87.7%.

$$TPR = 0.651$$

$$TNR = 0.943$$

$$ACC = 0.877$$

La Figura 49 muestra imágenes aleatorias de la fase de entrenamiento con los porcentajes alcanzados de precisión de la predicción de SqueezeNet.

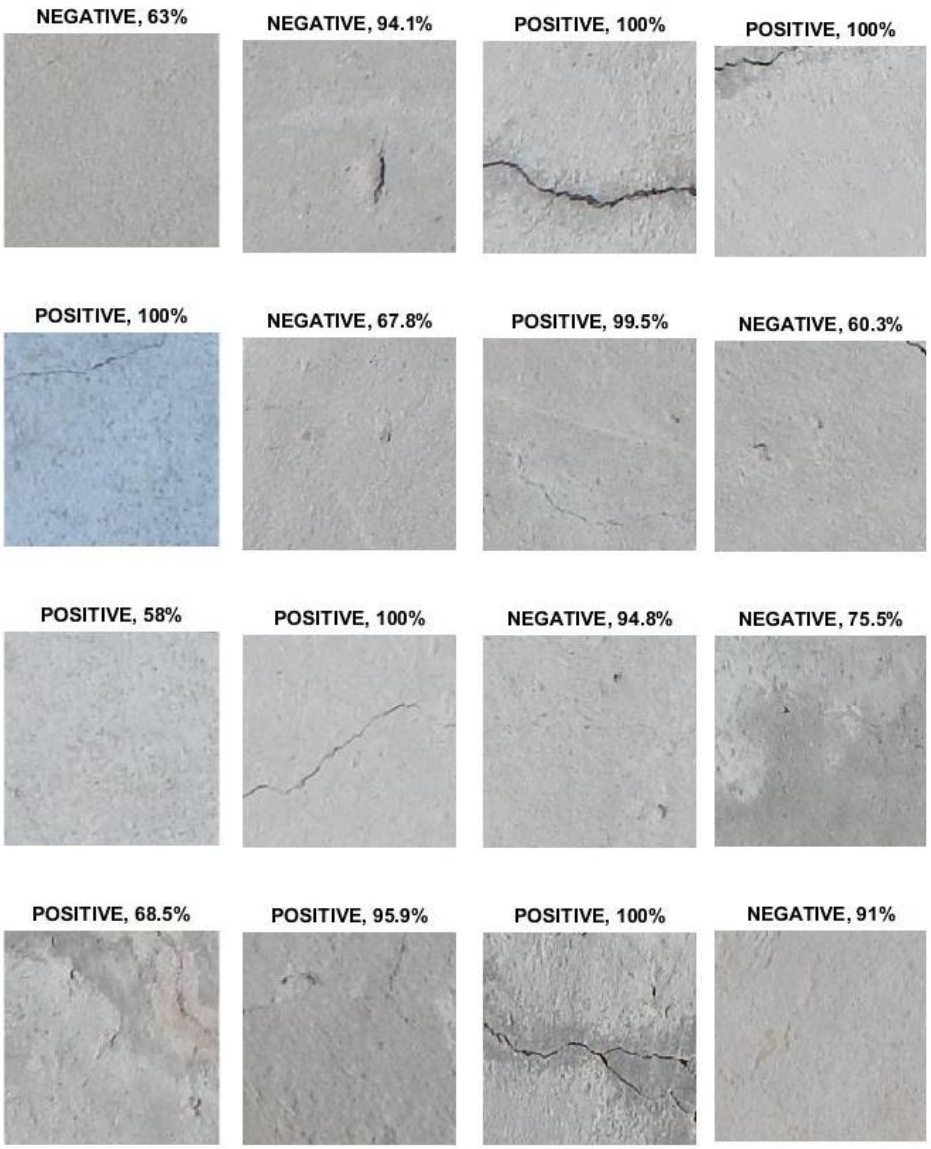


Figura 49. Porcentajes clasificación de las imágenes SqueezeNet.

En la *Figura* 50 se muestra la matriz de confusión para la arquitectura SqueezeNet. Detectando (977) negativos como negativos, (106) negativos detectados como positivos, (59) positivos detectados como negativos y (198) positivos detectados como positivos.

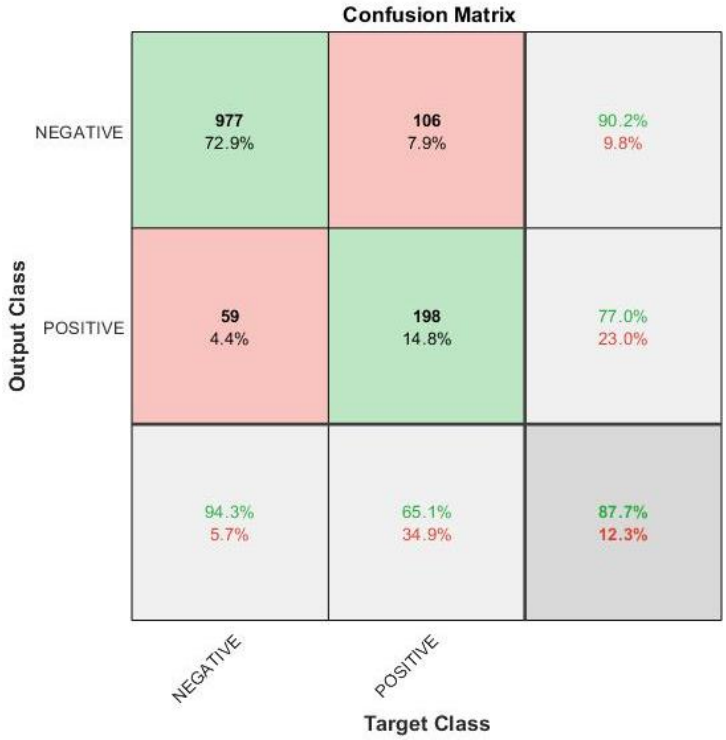


Figura 50. Matriz de confusión SqueezeNet primer entrenamiento

En la *Figura* 51 se muestra el resumen del proceso de entrenamiento GoogleNet. En la parte superior de la gráfica se muestra la precisión en la etapa de entrenamiento y en la parte inferior la función de pérdida. A mano derecha de la gráfica está el resumen de las iteraciones, las épocas y el tiempo que ha tardado en entrenar. Para este primer entrenamiento la red tardó 17 minutos y 47 segundos.

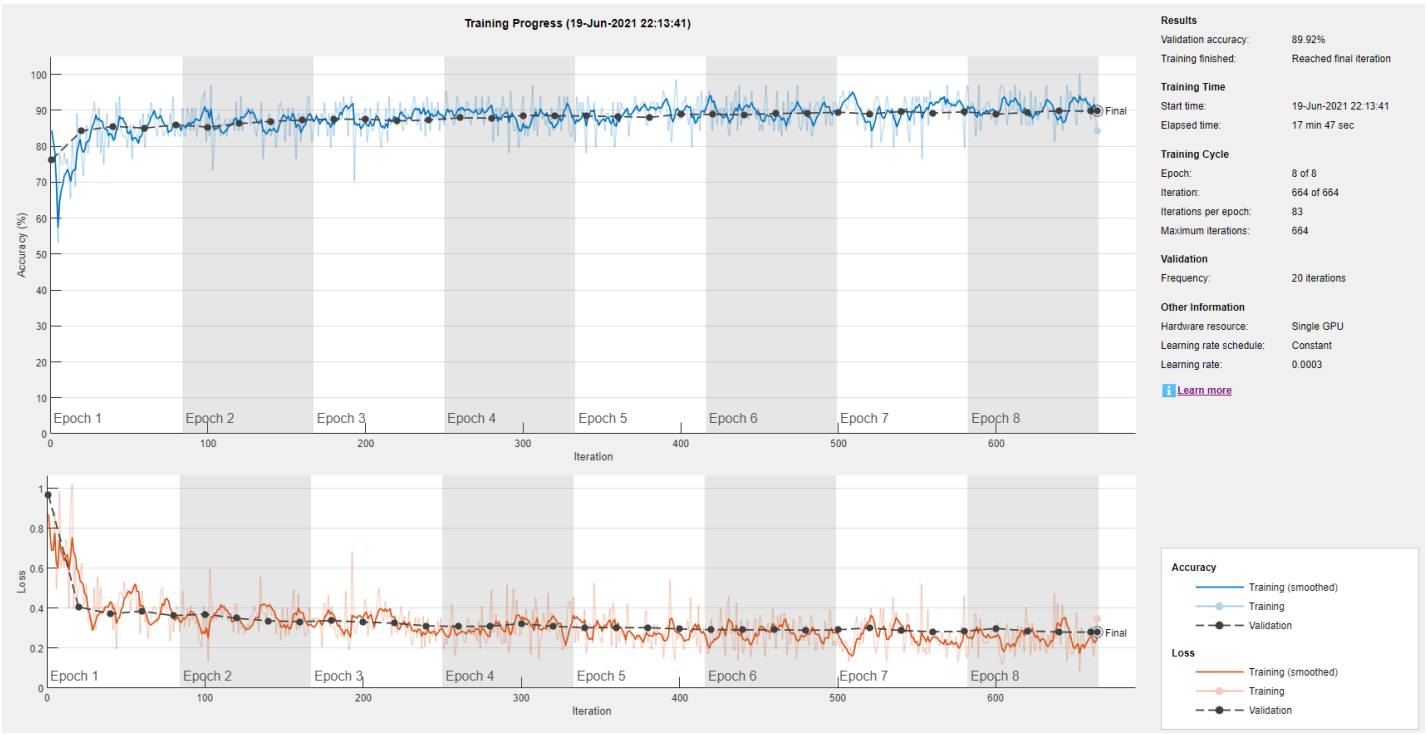


Figura 51. GoogleNet proceso de entrenamiento precisión de validación 89.92%

A continuación, se muestran los indicadores del entrenamiento de GoogleNet. Una tasa de verdaderos positivos de 64.1%. una tasa de verdaderos negativos de 97.5% y una precisión del 89.9%.

$$TPR = 0.641$$

$$TNR = 0.975$$

$$ACC = 0.899$$

La *Figura* 52 muestra imágenes aleatorias de la fase de entrenamiento con los porcentajes alcanzados de precisión de la predicción de la arquitectura GoogleNet.

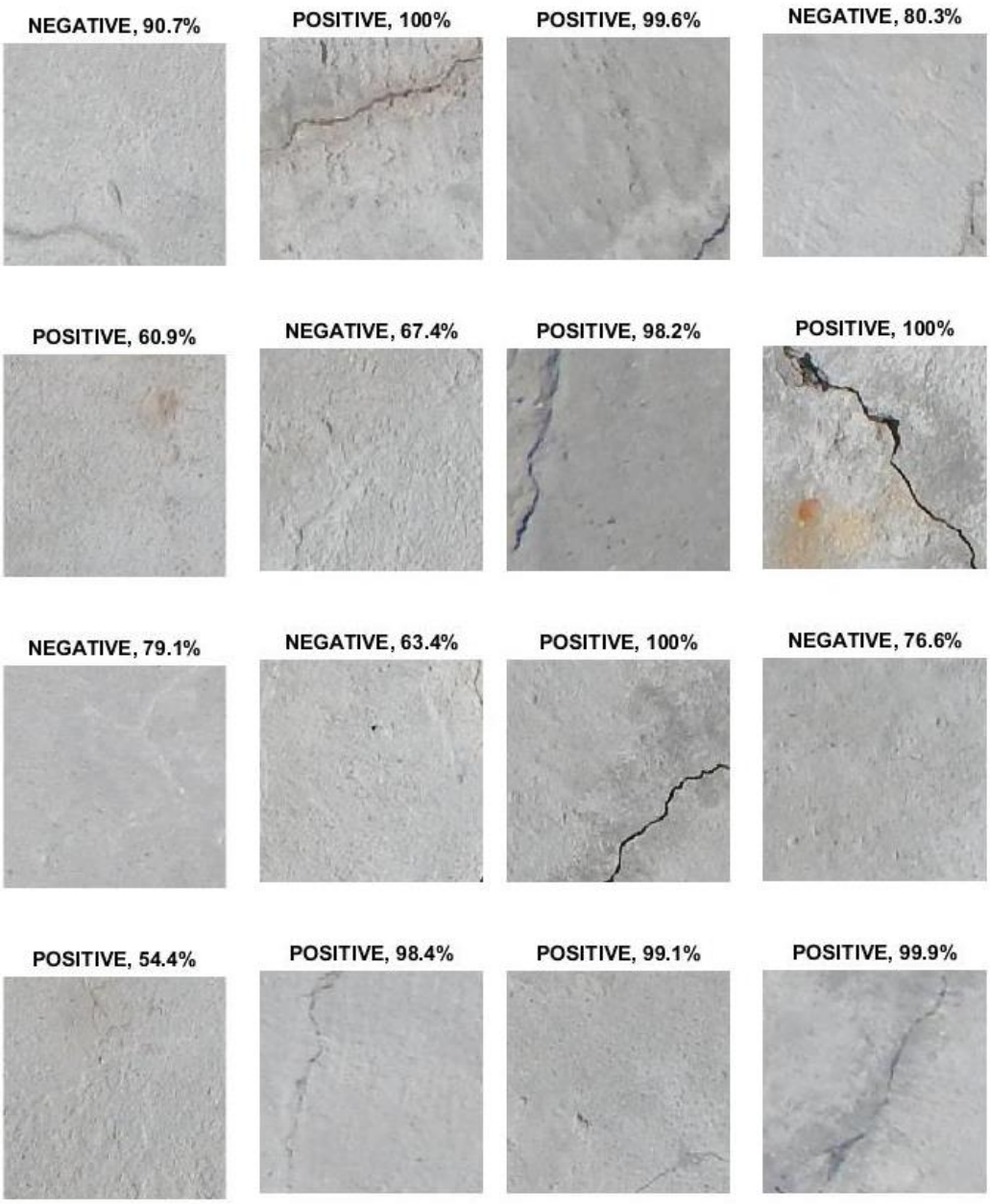


Figura 52. Porcentajes clasificación de las imágenes GoogleNet

En la *Figura* 53 se muestra la matriz de confusión para la arquitectura GoogleNet. Detectando (1010) negativos como negativos, (109) negativos detectados como

positivos, (26) positivos detectados como negativos y (195) positivos detectados como positivos.

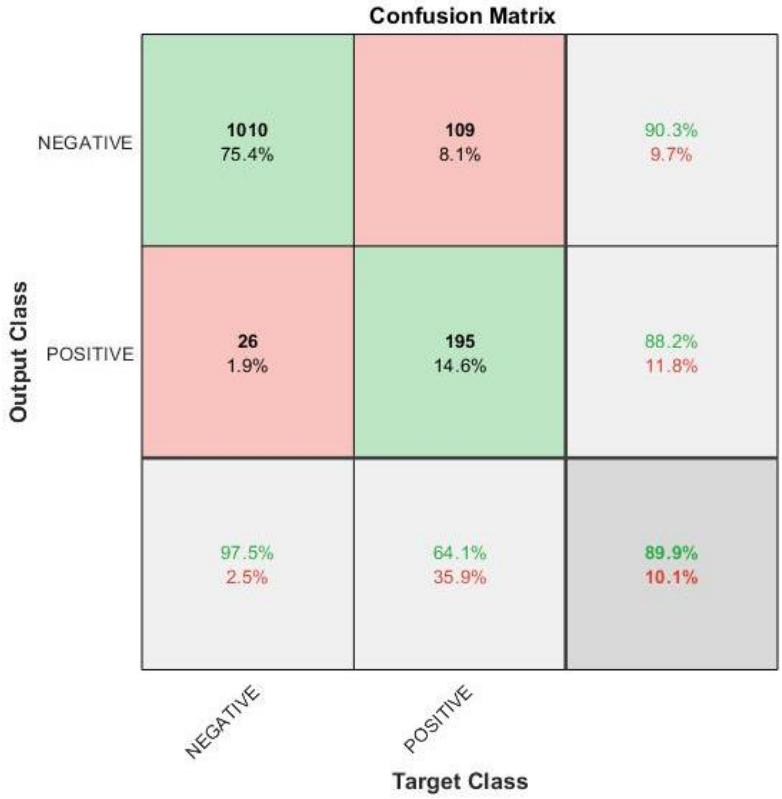


Figura 53. Matriz de confusión GoogleNet primer entrenamiento

Como se observa en la Tabla 5 y en la *Figura* 54, el porcentaje global de las redes neuronales es aceptable, con una precisión del más del 87%, pero todas las redes se encuentran sobreentrenando los resultados hacia la etiqueta de negativos y no está detectando correctamente las imágenes que presentan fisuras. Como se observa en las figuras 48,51 y 54, existen imágenes que la red ha catalogado como positivas y su etiqueta real es negativa, y otras que se detectan como negativas y su verdadera etiqueta es positiva.

ARQUITECTURA	PRECISION POSITIVOS (%)	PRECISION NEGATIVOS (%)	PRECISION TOTAL (%)	TIEMPO PROCESAMIENTO
ALEXNET	72.7%	95.8%	90.6%	5 min 29 seg
SQUEEZENET	65.1%	94.3%	87.7%	8 min 6 seg
GOOGLENET	64.1%	97.5%	89.9%	17 min 47 seg

Tabla 5. Precisión de detección de las arquitecturas entrenamiento 1.

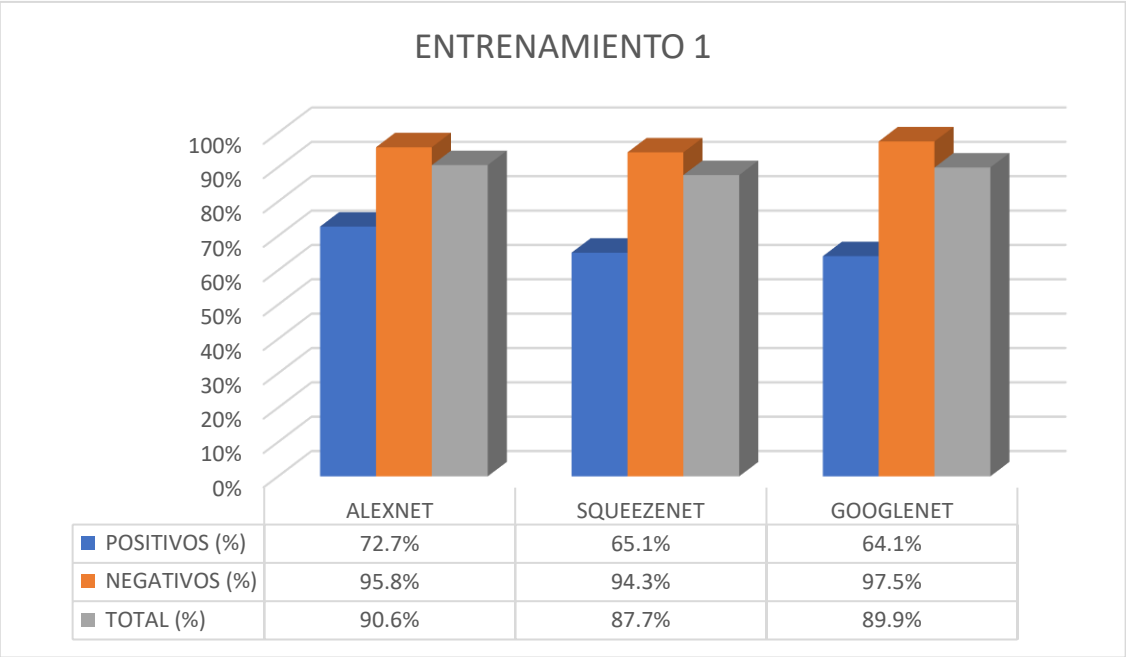


Figura 54. Gráfico columnas agrupadas entrenamiento 1

Por ello se realiza un segundo entrenamiento aumentando la base de datos añadiendo las imágenes de muros de la base de datos SDNET2018. Según la proporción de agrupación 60:25:15 indicada en el apartado 3.1.2., se toman 19055 imágenes para el entrenamiento y 7940 para la validación. Después del entrenamiento, se utilizan las 4764 imágenes restantes para probar la precisión del método de detección de fisuras, (véase la Tabla 6).

Etiquetas	Sub carpetas			Total	Base de datos
	Entrenamiento	validación	Prueba		
Negative	6957	2899	1739	11595	
Positive	1215	506	304	2025	SDNET2018-TABLEROS
Negative	8572	3572	2143	14287	
Positive	2311	963	578	3851	SDNET2018-MUROS
TOTAL	**19055**	**7940**	**4764**		

Tabla 6. Bases de datos empleadas en el entrenamiento 2.

En la Figura 55 se muestra el resumen del proceso del segundo entrenamiento de la arquitectura AlexNet. Para este segundo entrenamiento el tiempo de procesamiento aumento debido a que la nueva base de datos contenía muchas más imágenes, el tiempo para AlexNet fue de 19 minutos con 21 segundos.

Figura 55. AlexNet proceso de entrenamiento precisión de validación 87.34%

A continuación, se muestran los indicadores del entrenamiento de AlexNet. Una tasa de verdaderos positivos de 86.1%. una tasa de verdaderos negativos de 90.8% y una precisión del 88.4%.

$$TPR = 0.861$$

$$TNR = 0.908$$

$$ACC = 0.884$$

La figura 56 muestra imágenes aleatorias de la fase de entrenamiento con los porcentajes alcanzados de precisión de la predicción de AlexNet en el segundo entrenamiento.

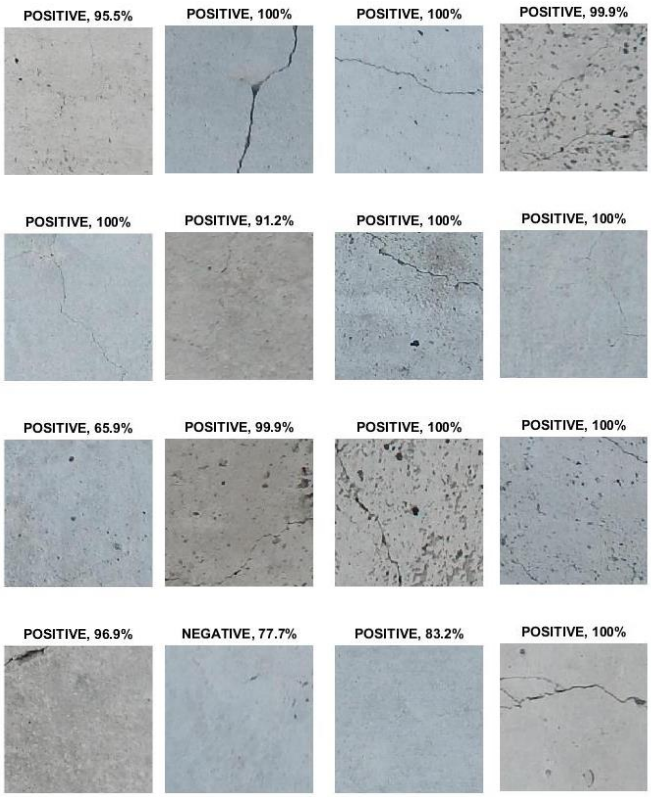


Figura 56. Porcentajes clasificación de las imágenes AlexNet

En la Figura 57 se muestra la matriz de confusión para la arquitectura AlexNet. Detectando (1945) negativos como negativos, (321) negativos detectados como positivos, (198) positivos detectados como negativos y (1990) positivos detectados como positivos.

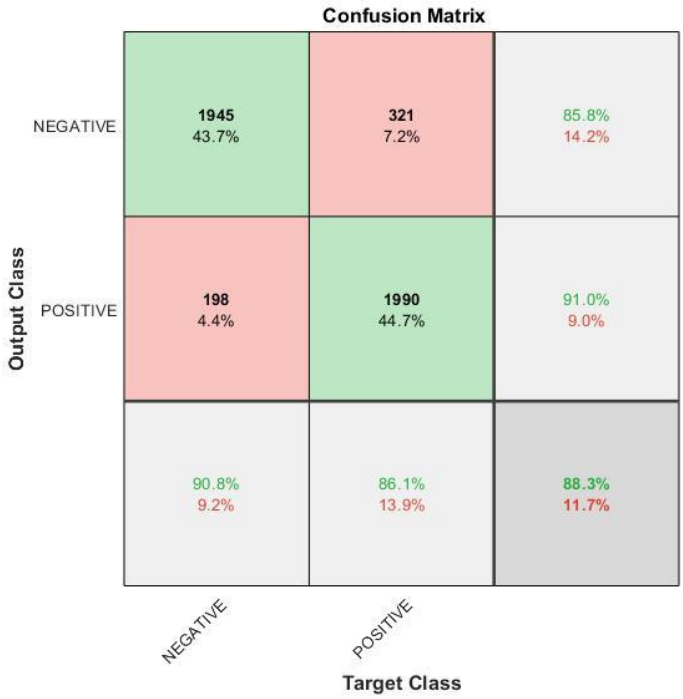


Figura 57. Matriz de confusión AlexNet segundo entrenamiento

En la Figura 58 se muestra el resumen del proceso del segundo entrenamiento de la arquitectura SqueezeNet. Con un tiempo de procesamiento de 29 minutos con 8 segundos.

Figura 58. SqueezeNet proceso de entrenamiento precisión de validación 89.22%

A continuación, se muestran los indicadores del entrenamiento de SqueezeNet. Una tasa de verdaderos positivos de 89.0%. una tasa de verdaderos negativos de 89.5% y una precisión del 88.6%.

$$TPR = 0.890$$

$$TNR = 0.895$$

$$ACC = 0.886$$

La *Figura* 59 muestra imágenes aleatorias de la fase de entrenamiento con los porcentajes alcanzados de precisión de la predicción de SqueezeNet en el segundo entrenamiento.

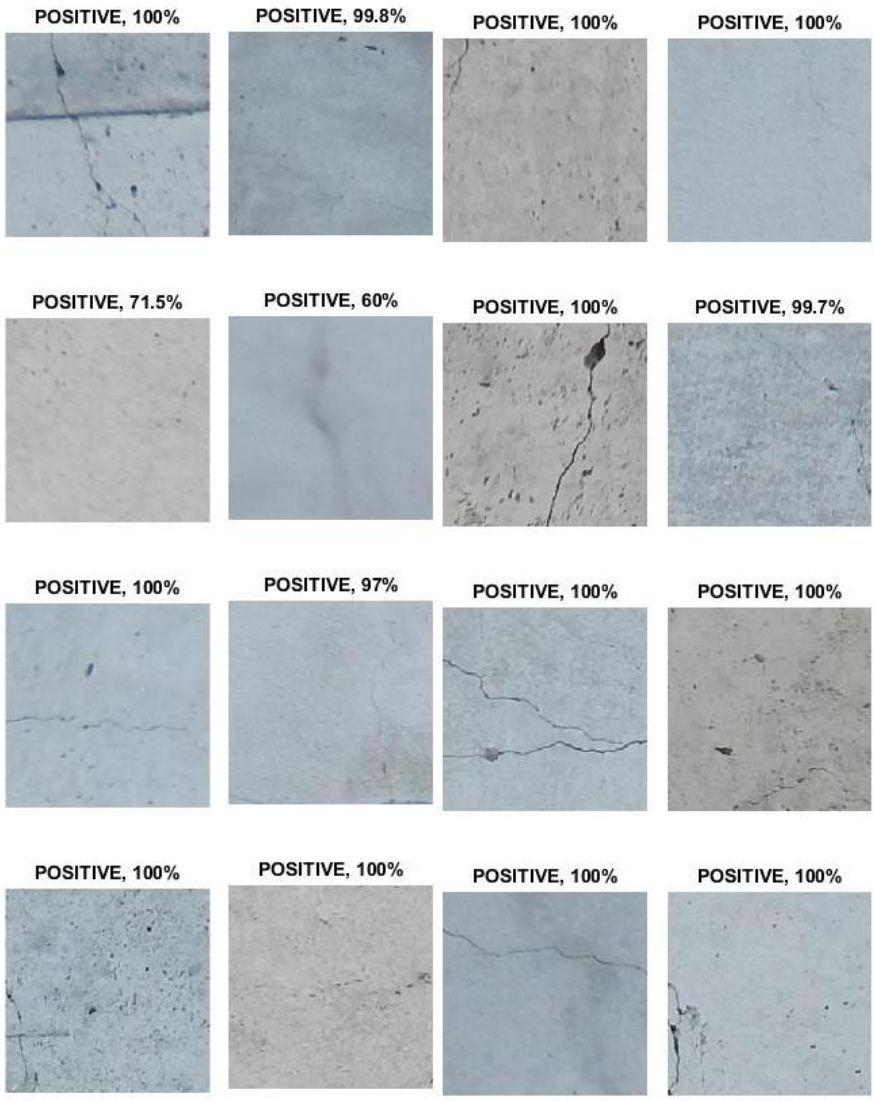


Figura 59. Porcentajes clasificación de las imágenes SqueezeNet

En la Figura 60 *se* muestra la matriz de confusión para la arquitectura SqueezeNet en el segundo entrenamiento. Detectando (1898) negativos como negativos, (243) negativos detectados como positivos, (245) positivos detectados como negativos y (2068) positivos detectados como positivos.

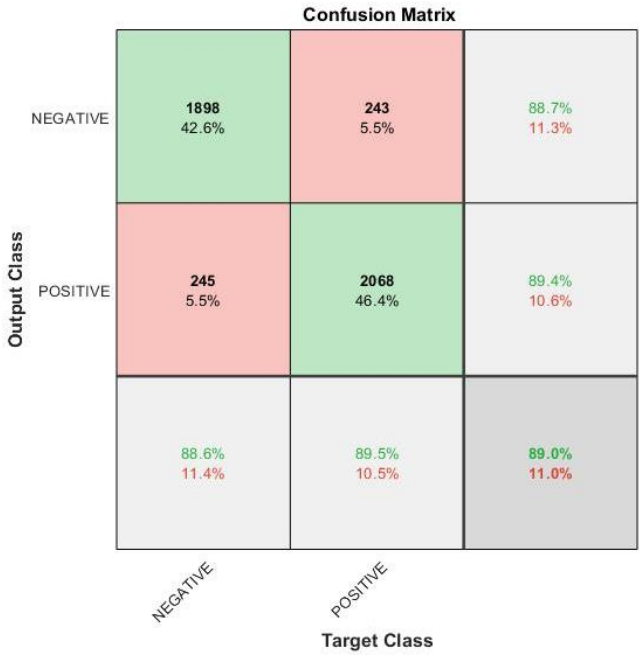


Figura 60. Matriz de confusión SqueezeNet segundo entrenamiento

En la Figura 61 se muestra el resumen del proceso del segundo entrenamiento de la arquitectura GoogleNet. Con un tiempo de procesamiento de 71 min con 41 segundos.

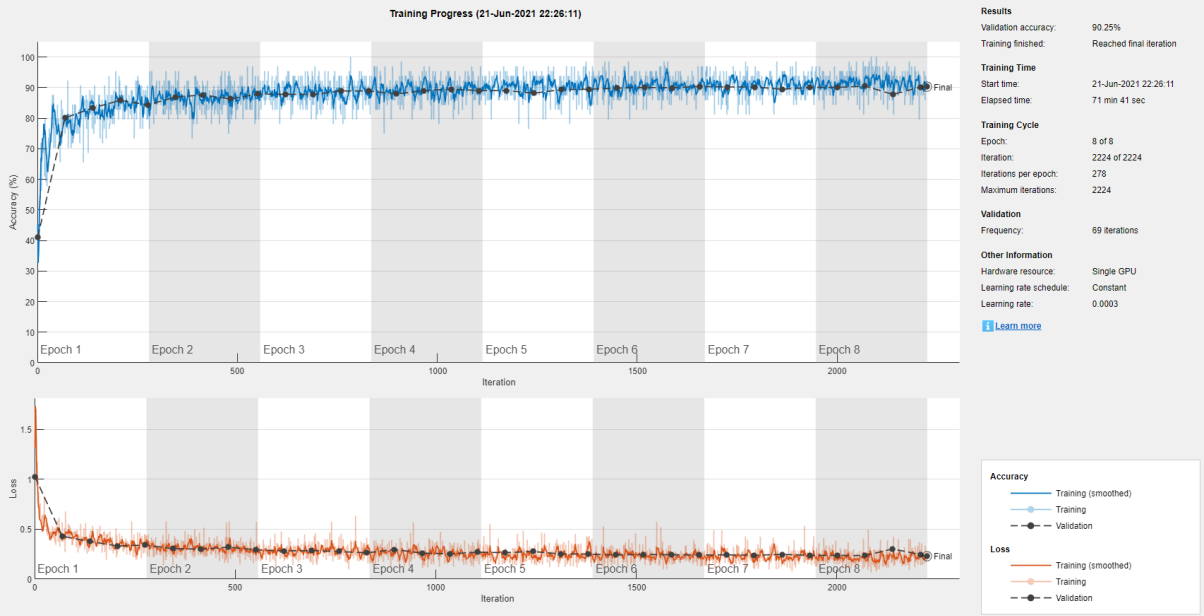


Figura 61. GoogleNet proceso de entrenamiento precisión de validación 90.25%

A continuación, se muestran los indicadores del entrenamiento de GoogleNet. Una tasa de verdaderos positivos de 88.7%. una tasa de verdaderos negativos de 92.4% y una precisión del 90.5%.

$$TPR = 0.887$$

$$TNR = 0.924$$

$$ACC = 0.905$$

La Figura 62 muestra imágenes aleatorias de la fase de entrenamiento con los porcentajes alcanzados de precisión de la predicción de GoogleNet en el segundo entrenamiento.

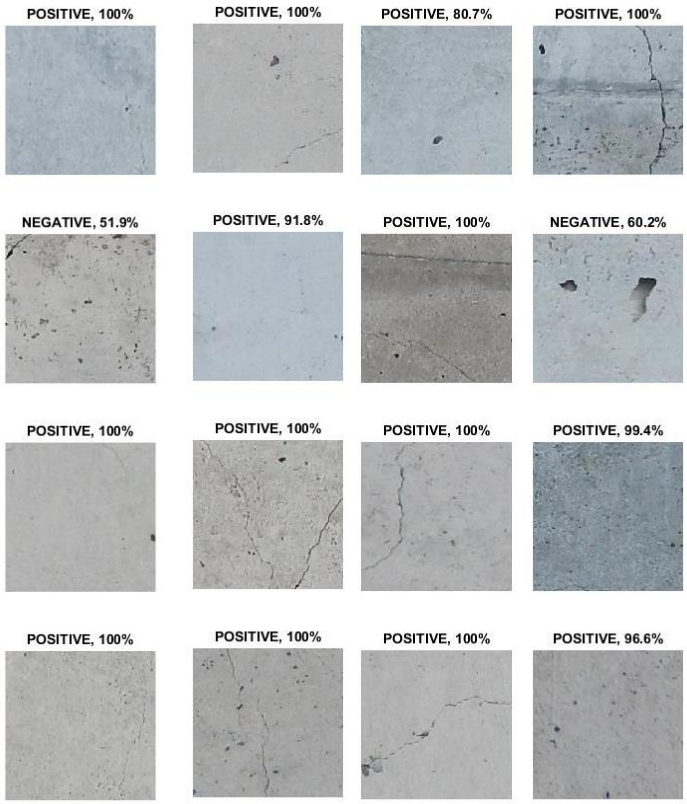


Figura 62. Porcentajes clasificación de las imágenes GoogleNet

En la Figura 63 se muestra la matriz de confusión para la arquitectura GoogleNet en el segundo entrenamiento. Detectando (1981) negativos como negativos, (262) negativos detectados como positivos, (162) positivos detectados como negativos y (2049) positivos detectados como positivos.

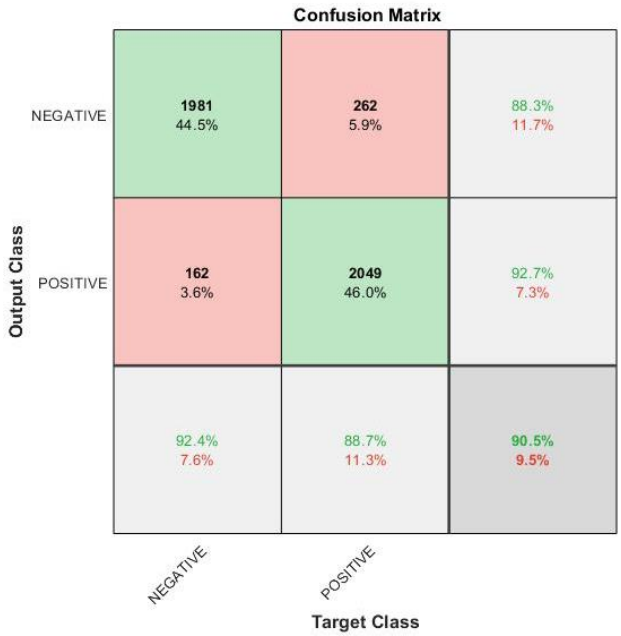


Figura 63. Matriz de confusión GoogleNet segundo entrenamiento

Con el data set de SDNET2018 de muros incluidos en el segundo entrenamiento, se ve en la Tabla 7 y la Figura 64 como la precisión de las redes en general disminuyeron para las dos primeras redes con respecto al primer entrenamiento. Pero los valores de predicción de fisuras positivos aumentaron considerablemente y están casi iguales para ambas etiquetas (negativa y positiva). Pero los resultados de precisión de las redes no son muy convincentes y aun predicen con cierto error la clasificación de las imágenes.

ARQUITECTURA	PRECISION POSITIVOS (%)	PRECISION NEGATIVOS (%)	PRECISION TOTAL (%)	TIEMPO PROCESAMIENTO
ALEXNET	86.1%	90.8%	88.3%	19 min 21 seg
SQUEEZENET	89.5%	88.6%	87.7%	29 min 8 seg
GOOGLENET	88.7%	92.4%	90.5%	71 min 41 seg

Tabla 7. precisión de detección de las arquitecturas entrenamiento 2

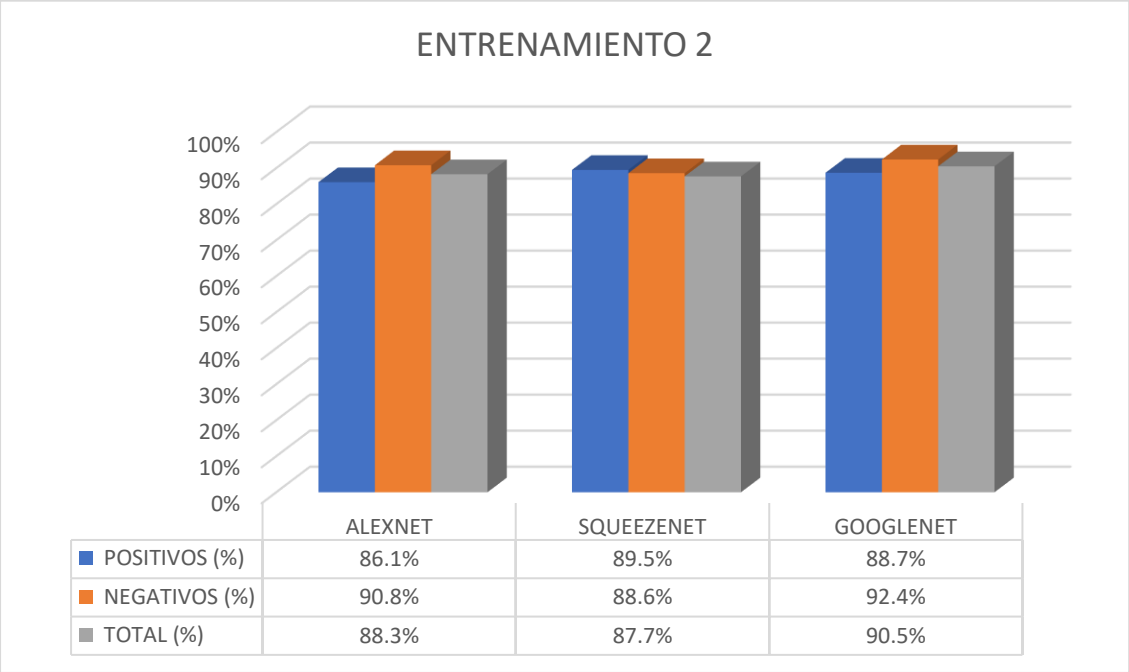


Figura 64. Gráfico columnas agrupadas entrenamiento 2

Como se observa en la tabla Tabla 6. En proporción, las imágenes de etiqueta negativa tienen mayor cantidad que las positivas, siendo estas un 85.13% (11595 imágenes) del total de la base de datos para tableros y un 78.77% (14287 imágenes) de la base de muros, ocasionando como se refleja en los dos entrenamientos anteriores un sobreajuste hacia la predicción de las etiquetas negativas, no entrenando bien las CNN.

Etiquetas	Sub carpetas			Total	Base de datos
	Entrenamiento	validación	Prueba		
Negative	6957	2899	1739	11595	
Positive	1215	506	304	2025	SDNET2018-TABLEROS
Negative	8572	3572	2143	14287	
Positive	2311	963	578	3851	SDNET2018-MUROS
TOTAL	**19055**	**7940**	**4764**		

Tabla 6. Bases de datos empleadas en el entrenamiento 2.

Es evidente que en el entrenamiento 2, donde la base de datos de muros tiene un porcentaje mayor de imágenes positivas, la precisión aumentó considerablemente. Con base en esto, para poder obtener mejores resultados en general de la precisión de las redes, fue necesario que ambas etiquetas tuvieran un peso similar y así evitar el sobre entrenamiento, para ello se utilizó el *augmentation* descrito en el apartado 3.2 y se sumó la base de datos METU.

Para un tercer entrenamiento se utilizan las imágenes y las proporciones de la tabla 1.

Etiquetas	Grupos			Total
	Entrenamiento	Validación	Prueba	
Negative	27529	11471	6882	45882
Positive	28533	11889	7133	47555

Tabla 1. Preparación de los grupos de imágenes.

En la figura 65 se muestra el resumen del proceso del tercer entrenamiento de la arquitectura AlexNet. Con un tiempo de procesamiento de 53 minutos y 50 segundos.

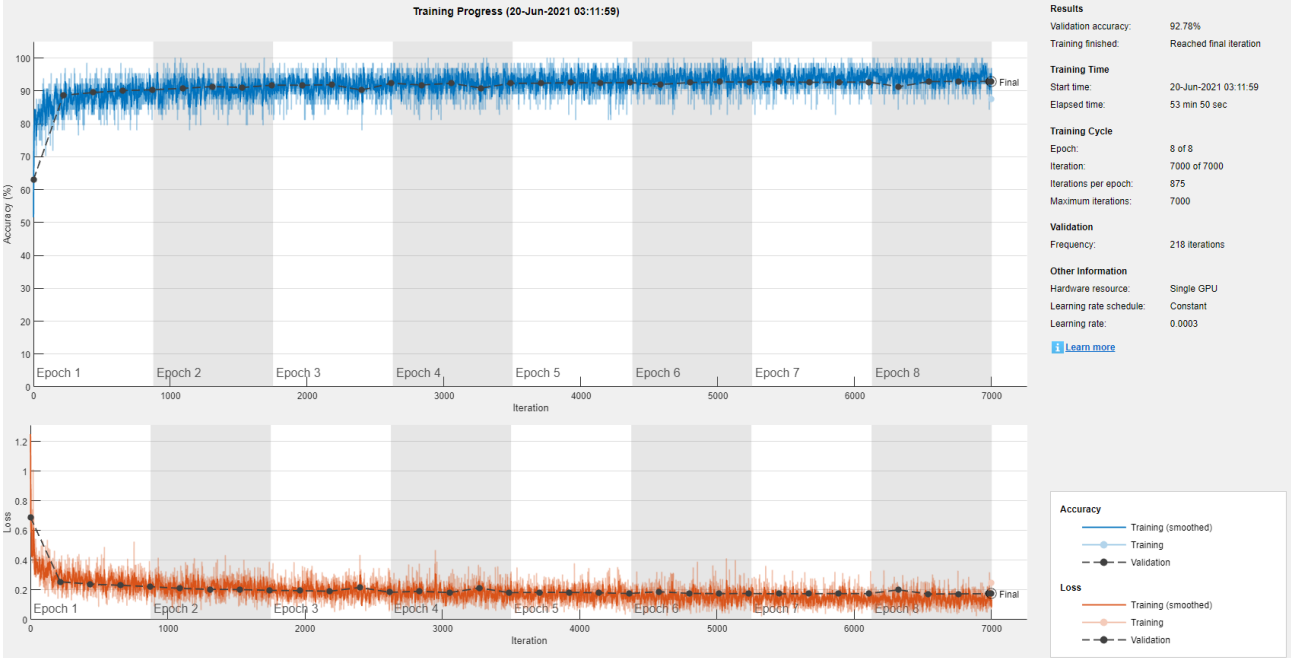


Figura 65. AlexNet proceso de entrenamiento precisión de validación 92.78%

A continuación, se muestran los indicadores del entrenamiento de AlexNet. Una tasa de verdaderos positivos de 92.6%. una tasa de verdaderos negativos de 92.5% y una precisión del 92.5%.

$$TPR = 0.926$$

$$TNR = 0.925$$

$$ACC = 0.925$$

La Figura 66 muestra imágenes aleatorias de la fase de entrenamiento con los porcentajes alcanzados de precisión de la predicción de AlexNet en el tercer entrenamiento.

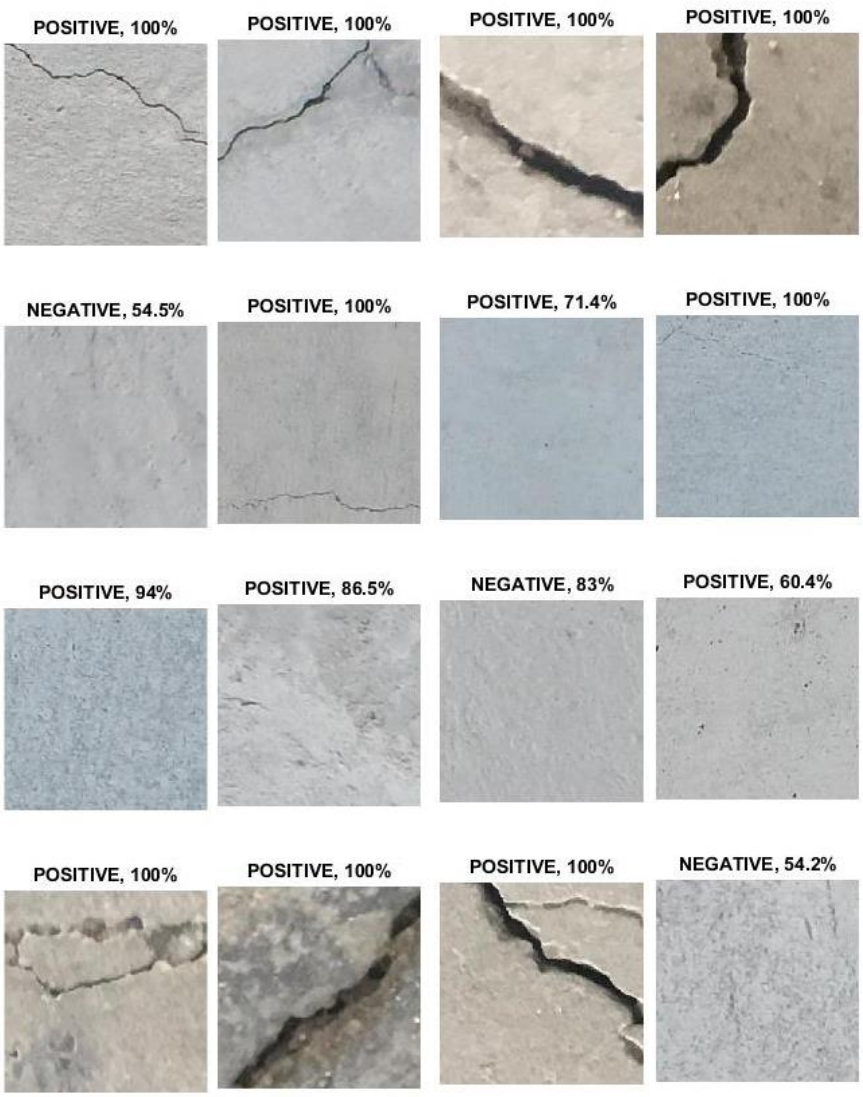


Figura 66. Porcentajes clasificación de las imágenes AlexNet

En la Figura 67 muestra la matriz de confusión para la arquitectura AlexNet en el tercer entrenamiento. Detectando (6366) negativos como negativos, (529) negativos detectados como positivos, (516) positivos detectados como negativos y (6604) positivos detectados como positivos.

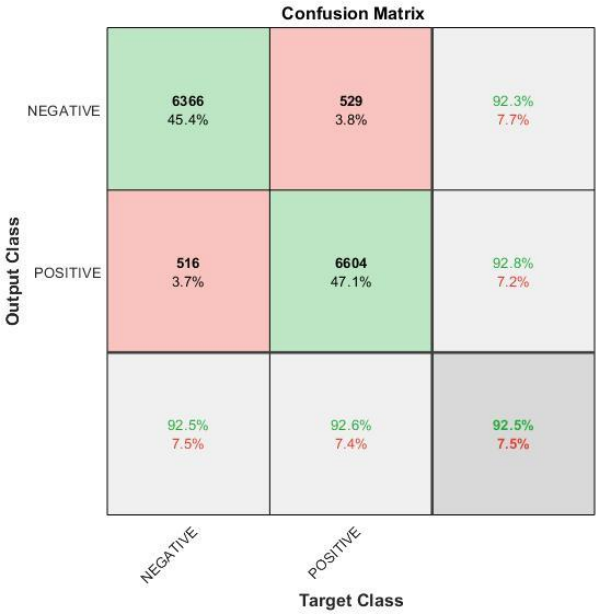


Figura 67. Matriz de confusión AlexNet tercer entrenamiento

A continuación, se muestra el resumen del proceso del tercer entrenamiento de SqueezeNet. Con un tiempo de procesamiento de 81 minutos 3 segundos.

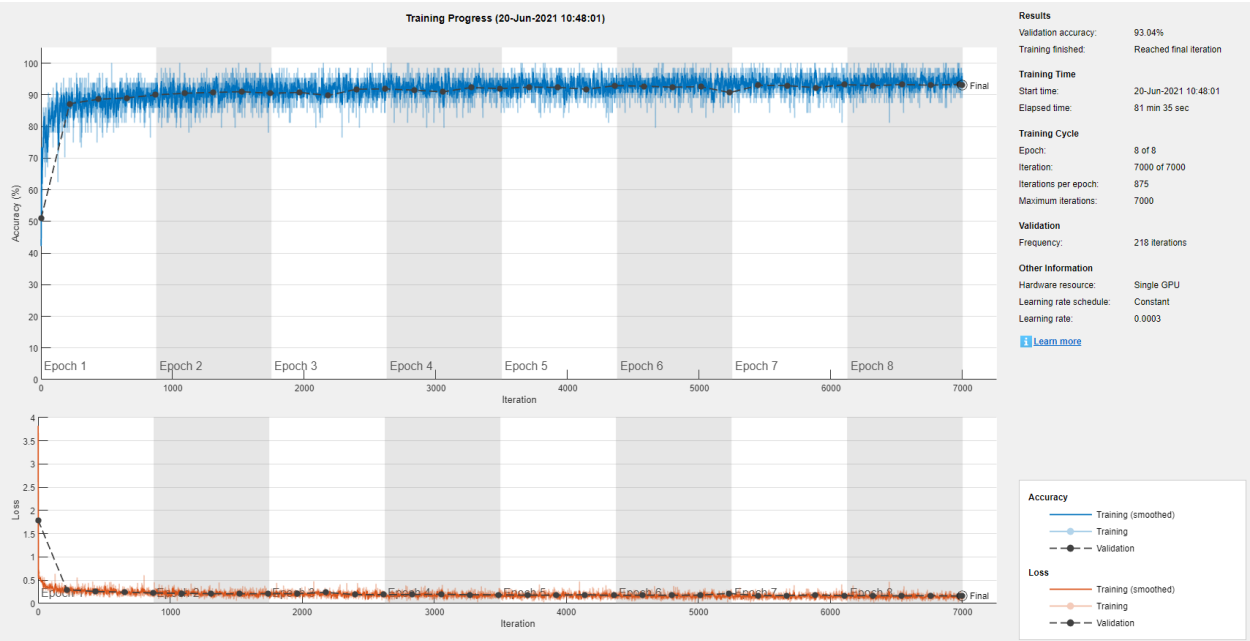


Figura 68. SqueezeNet proceso de entrenamiento precisión de validación 93.04%

Los indicadores para el tercer entrenamiento de SqueezeNet son los siguientes, una tasa de verdaderos positivos de 89.7%. una tasa de verdaderos negativos de 97.2% y una precisión del 93.2%.

$$TPR = 0.897$$

$$TNR = 0.972$$

$$ACC = 0.932$$

La Figura 69 muestra imágenes aleatorias de la fase de entrenamiento con los porcentajes alcanzados de precisión de la predicción de SqueezeNet en el tercer entrenamiento.

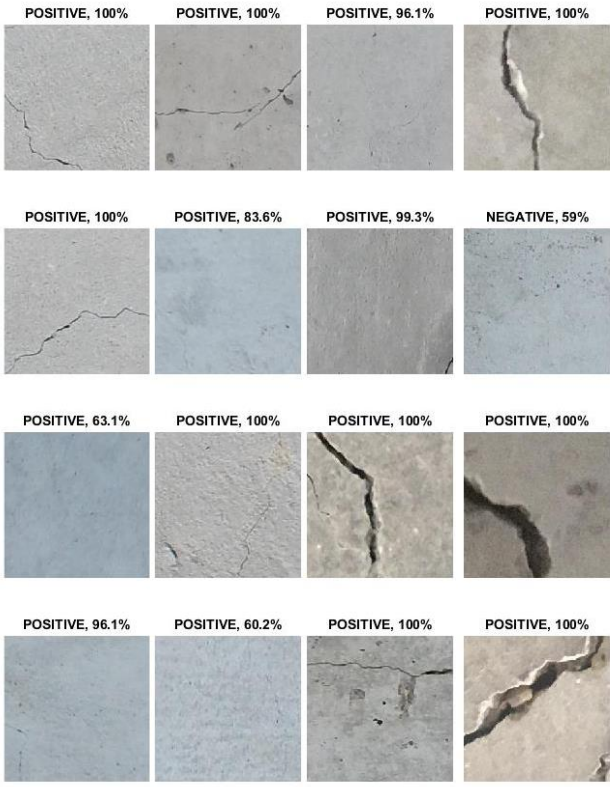


Figura 69. Porcentajes clasificación de las imágenes SqueezeNet

En la *Figura* 70 muestra la matriz de confusión para la arquitectura SqueezeNet en el tercer entrenamiento. Detectando (6656) negativos como negativos, (733) negativos detectados como positivos, (226) positivos detectados como negativos y (6400) positivos detectados como positivos.

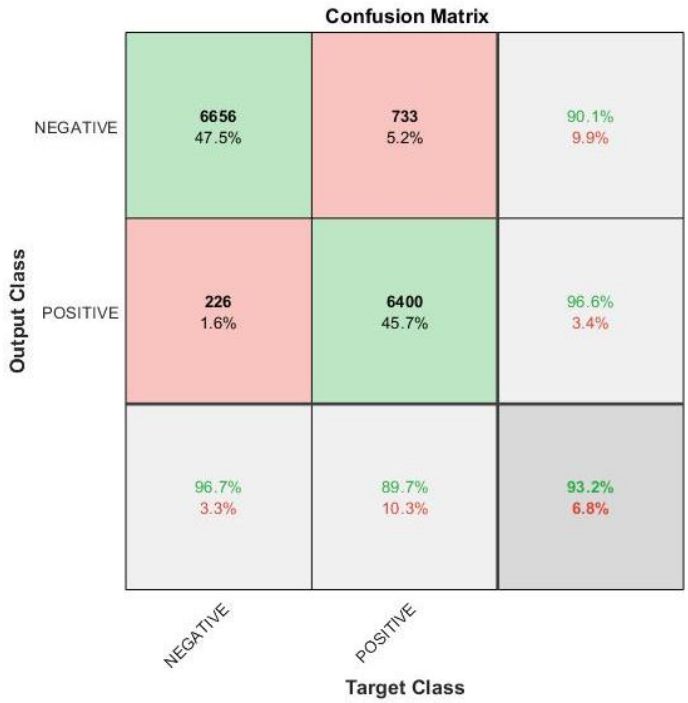


Figura 70. Matriz de confusión SqueezeNet tercer entrenamiento

En la Figura 71 se muestra el resumen del proceso del tercer entrenamiento de la arquitectura GoogleNet. Con un tiempo de 180 minutos con 38 segundos.

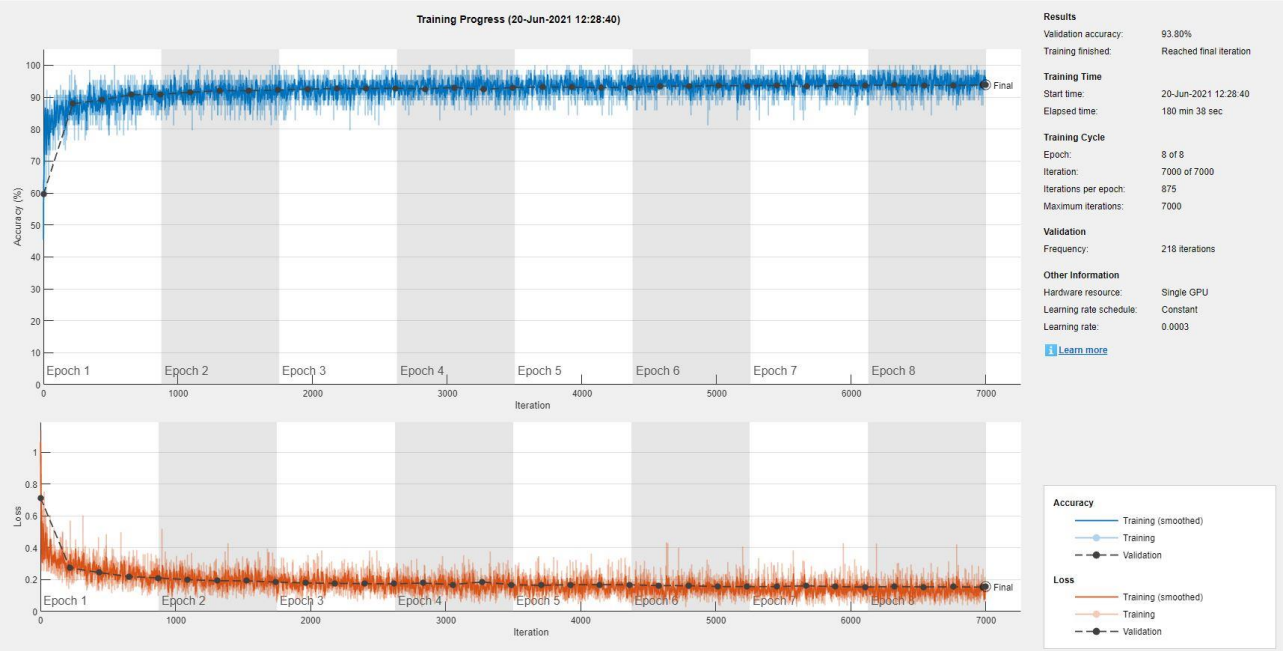


Figura 71. GoogleNet proceso de entrenamiento precisión de validación 93.80%

Los indicadores para el tercer entrenamiento de GoogleNet son los siguientes, una tasa de verdaderos positivos de 90.5%. una tasa de verdaderos negativos de 97.2% y una precisión del 93.8%.

$$TPR = 0.905$$

$$TNR = 0.972$$

$$ACC = 0.938$$

En la siguiente figura, se muestran imágenes aleatorias de la clasificación de las fisuras en el tercer entrenamiento de GoogleNet.

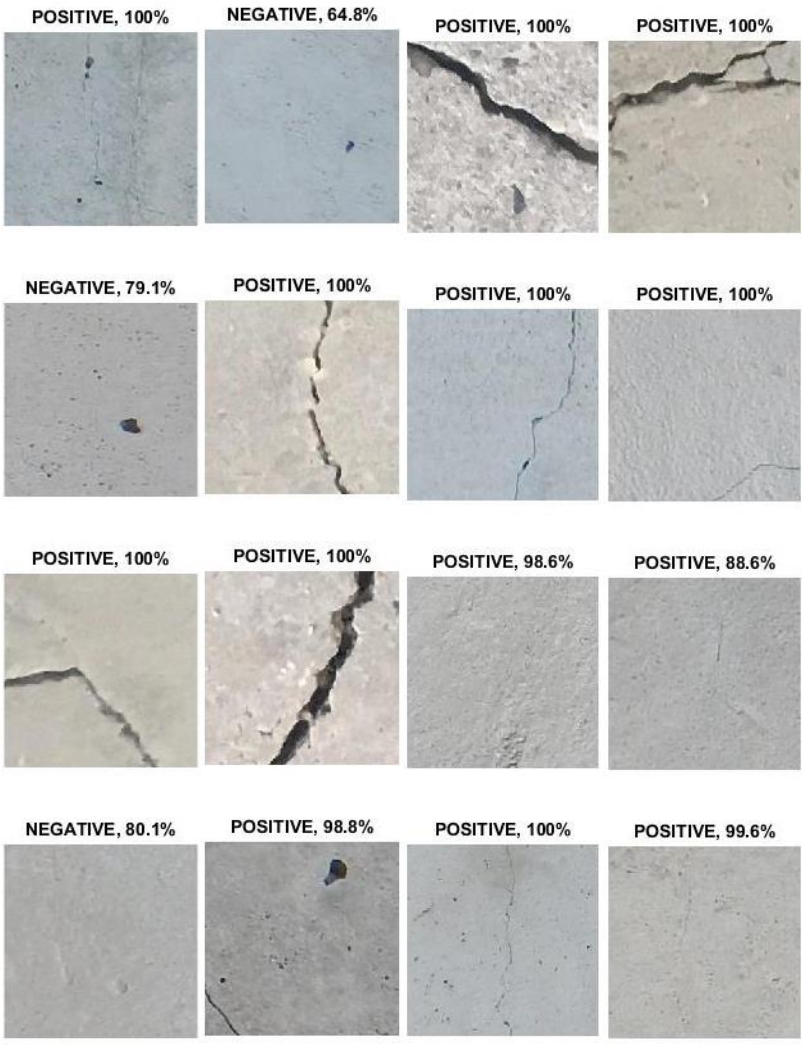


Figura 72. Porcentajes clasificación de las imágenes GoogleNet

En la *Figura* 73 muestra la matriz de confusión para la arquitectura GoogleNet en el tercer entrenamiento. Detectando (6687) negativos como negativos, (678) negativos detectados como positivos, (195) positivos detectados como negativos y (6455) positivos detectados como positivos.

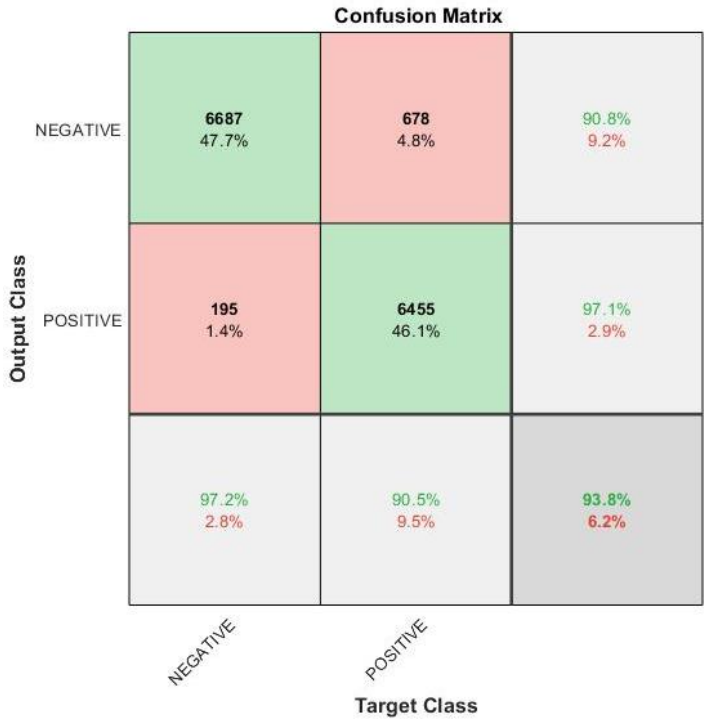


Figura 73. Matriz de confusión GoogleNet tercer entrenamiento

El tercer entrenamiento muestra un aumento en los indicadores de las tres redes, ver Tabla 8 y Figura 74, AlexNet es la red neuronal con un menor tiempo de procesamiento (53 min con 30 segundo) y donde la precisión de positivos, negativos y total no presentan mucha variación entre sí. Pero es la red que ocupa mayor espacio en disco duro, 9 veces más que las otras, el tamaño de la red es un factor importante a la hora de desarrollar una herramienta informática.

ARQUITECTURA	PRECISION POSITIVOS (%)	PRECISION NEGATIVOS (%)	PRECISION TOTAL (%)	TIEMPO PROCESAMIENTO
ALEXNET	92.6%	92.5%	92.5%	53 min 50 seg
SQUEEZENET	89.7%	96.7%	93.2%	81 min 35 seg
GOOGLENET	90.5%	97.2%	93.8%	180 min 38 seg

Tabla 8. Precisión de detección de las arquitecturas entrenamiento 3

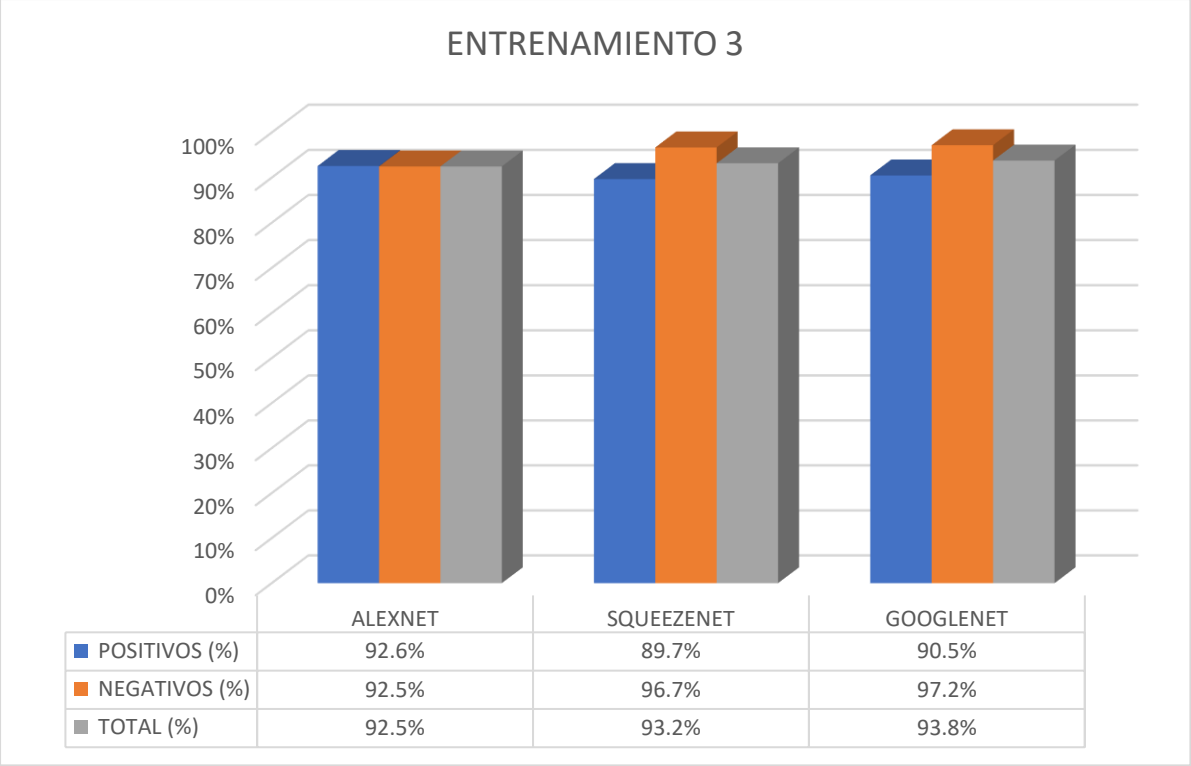


Figura 74. Gráfico columnas agrupadas entrenamiento 3

La red con la mejor precisión general de todo el entrenamiento es GoogleNet con 93.8% y ocupa 27 MB (Tabla 9) de espacio en el disco lo cual hace una herramienta fácil de procesar en cualquier aplicación. Además, GoogleNet es la arquitectura que cuenta con la mayor cantidad de capas profundas que a lo largo de todo el proceso de clasificación extraen y almacenan más características que en un futuro puedan ser utilizadas para mejorar la herramienta.

Red	Profundidad	Tamaño	Tamaño imagen de entrada
SqueezeNet	18	5.2 MB	227x227
GoogleNet	22	27 MB	224x224
AlexNet	8	227 MB	227x227

Tabla 9. Propiedades generales de las redes entrenadas

4.3 Prueba de la herramienta

Se ha decidido utilizar la arquitectura de GoogleNet para la validación de la herramienta de detección de fisuras. Se dispone a probar la herramienta con fisuras de una vigueta de hormigón llevada a rotura en el laboratorio de hormigón de la Universidad Politécnica de Valencia (Figura 75).

Figura 75. Rotura Vigueta T12.2- laboratorio de hormigón UPV

Se tomó una serie de fotografías y videos de la carga hasta la rotura a flexión de una vigueta T12.2 de 3 metros de longitud con la sección transversal y armado como se muestra en la Figura 76. La carga es aplicada por un gato hidráulico que se divide en dos cargas puntuales sobre la vigueta a L/3, estos puntos de aplicación dividen la vigueta en tres tramos iguales (Figura 77). Se utilizó un pórtico autoportante con un gato hidráulico de 50 TN para luces de 1, 2, y 3 metros.

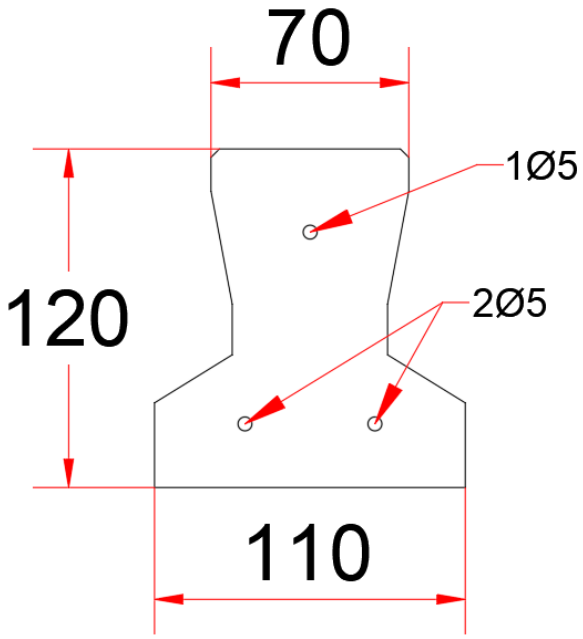


Figura 76. Sección vigueta T12.2 (mm)

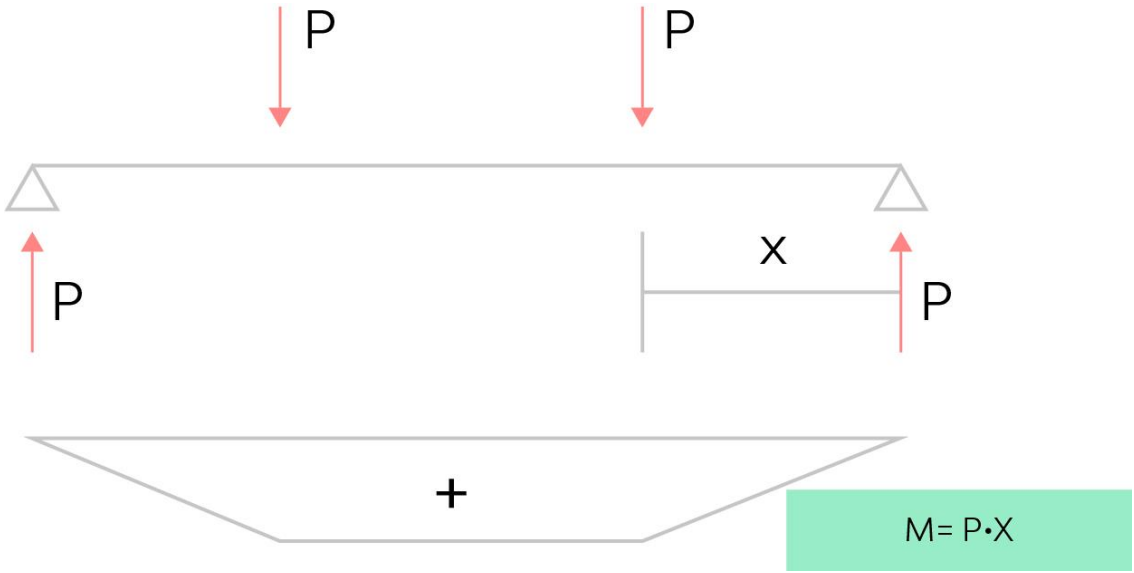


Figura 77. Método de carga y ley de momentos flectores en la vigueta

En la Figura 78 se muestra la gráfica de carga aplicada por el gato hidráulico vs deflexión de la vigueta. La vigueta rompe aproximadamente con una carga de 1.4 toneladas y una flecha de 100mm.

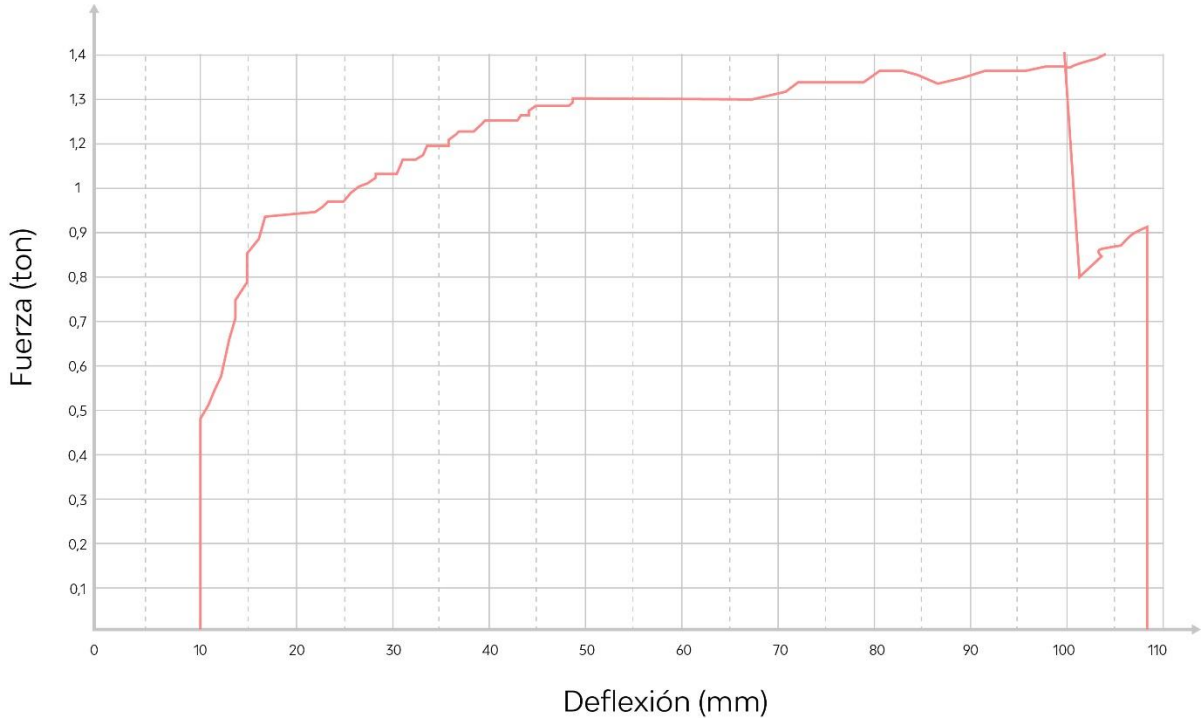


Figura 78. Grafica rotura (Carga aplicada Vs Deformación)

De las figuras 79 a 86 se muestra la secuencia de imágenes de apertura de la fisura hasta el momento del fallo. Estas imágenes se introducirán en la herramienta de detección de fisuras para determinar cuándo la grieta comienza a ser detectada, identificando el porcentaje de región de interés que debe tener la imagen para su clasificación como positiva.

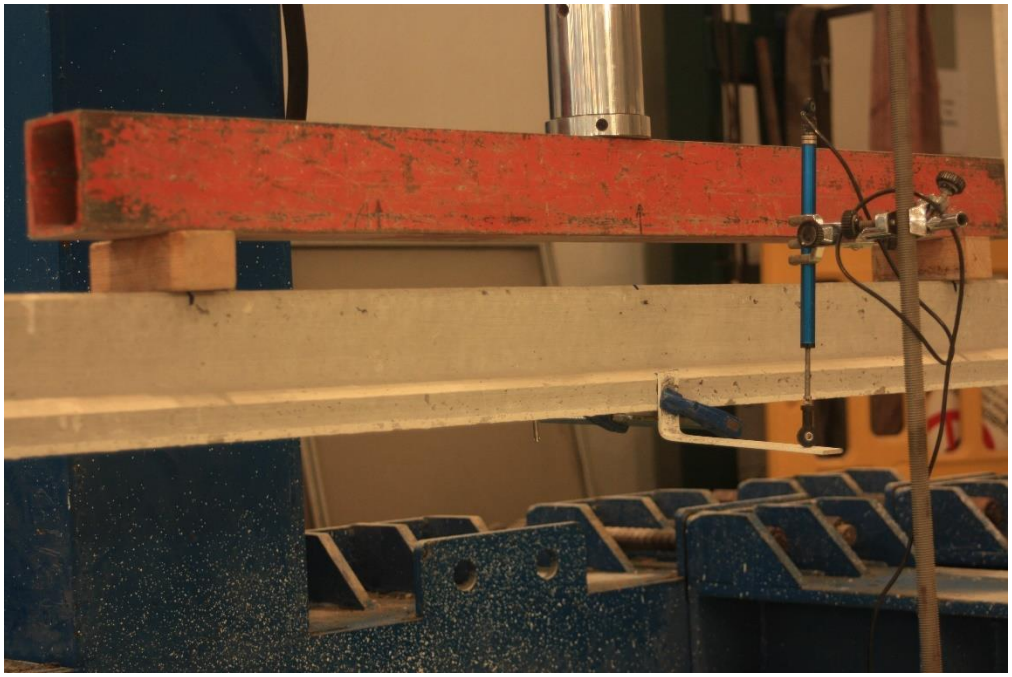


Figura 79. Estado inicial Vigueta T12.2 (Imagen A)

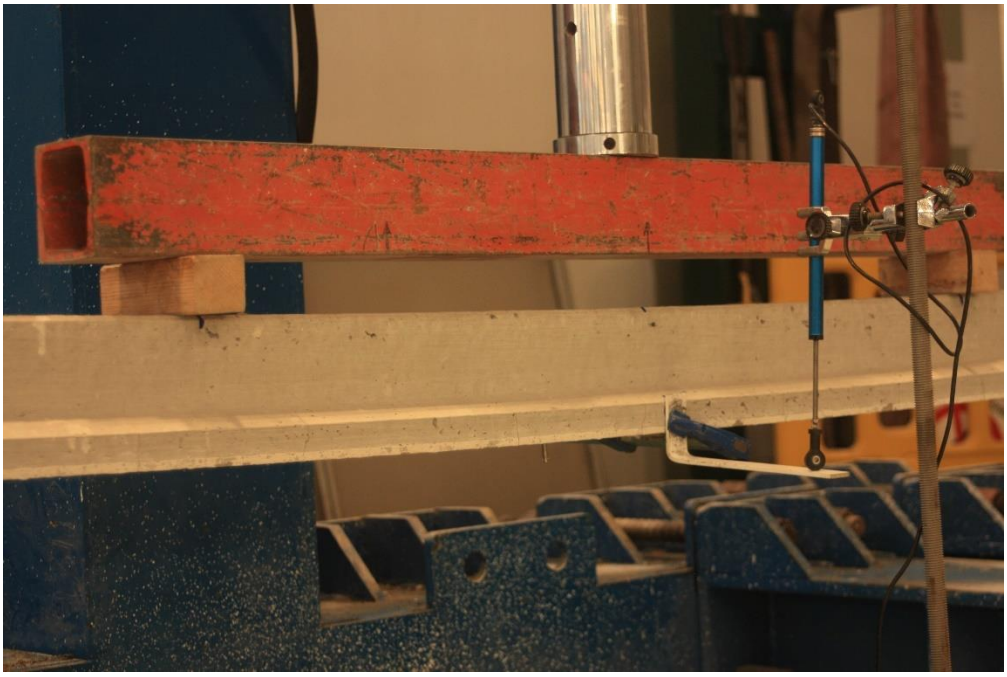


Figura 80. Aparición de la primer fisura Vigueta T12.2 (Imagen B)

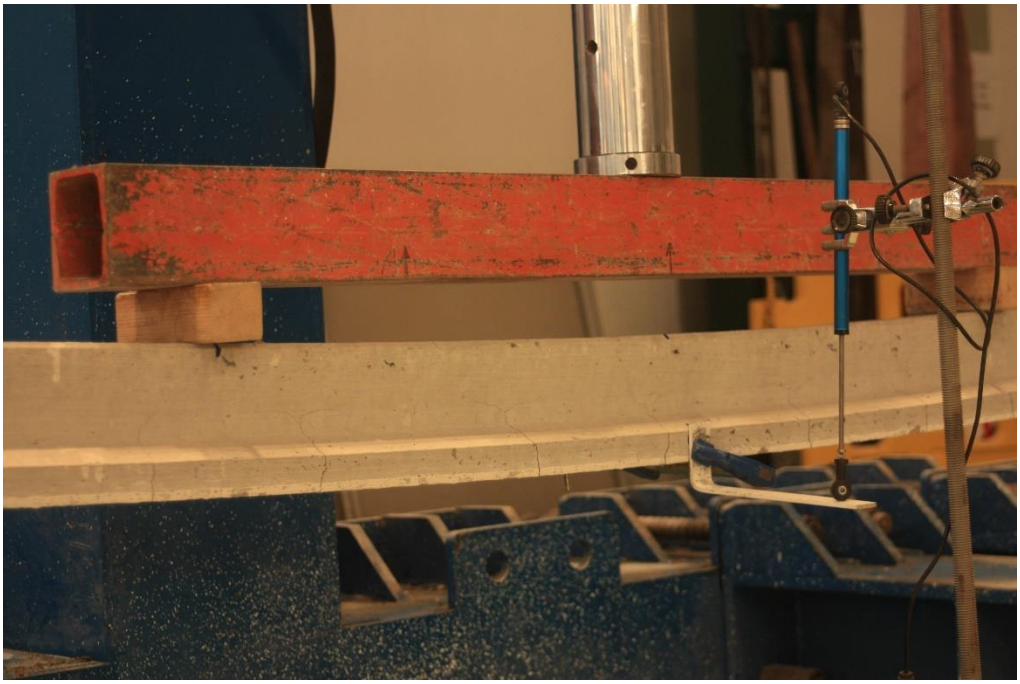


Figura 81. Aumento de flecha y apertura de fisura Vigueta T12.2 (Imagen C)

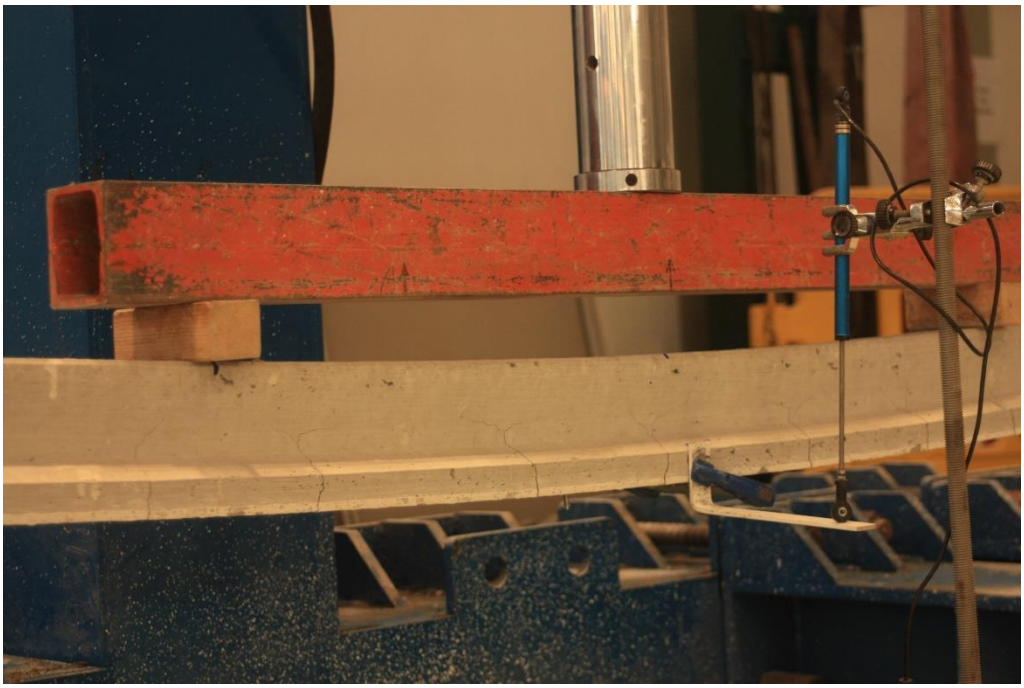


Figura 82. Aumento de flecha y apertura de fisura Vigueta T12.2 (Imagen D)

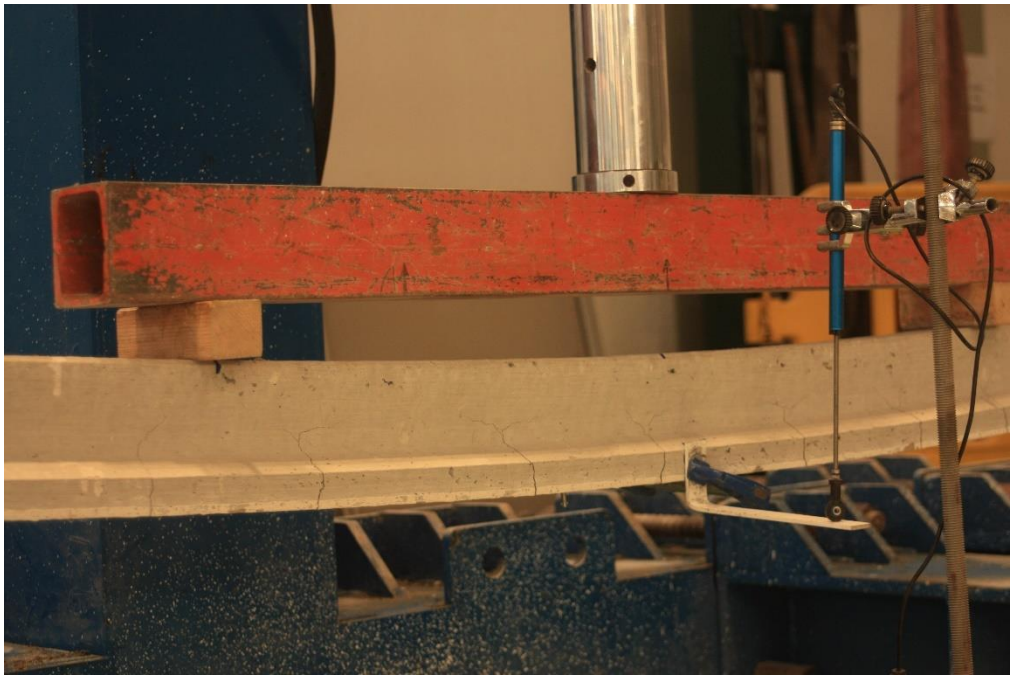


Figura 83. Aumento de flecha y apertura de fisura Vigueta T12.2 (Imagen E)

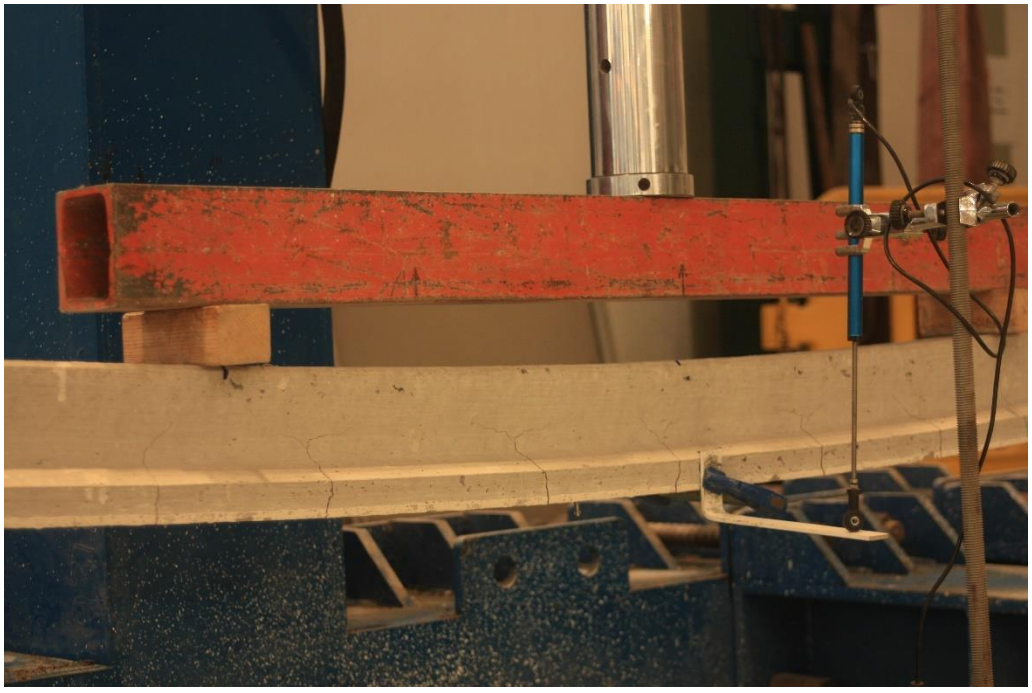


Figura 84. Aumento de flecha y apertura de fisura Vigueta T12.2 (Imagen F)

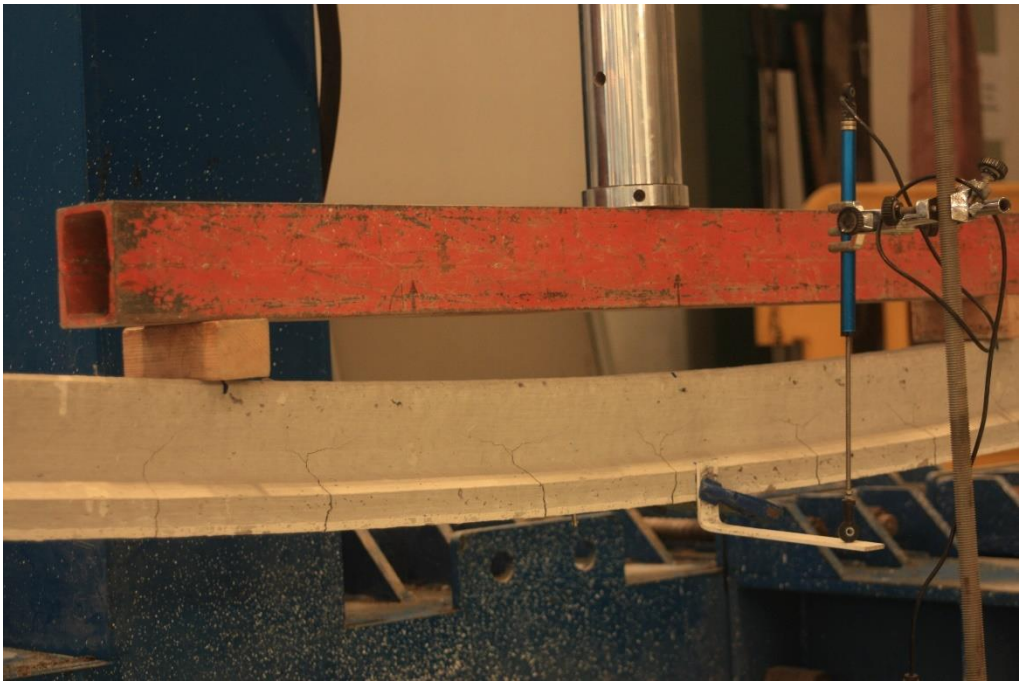


Figura 85. Aumento de flecha y apertura de fisura Vigueta T12.2 (Imagen G)

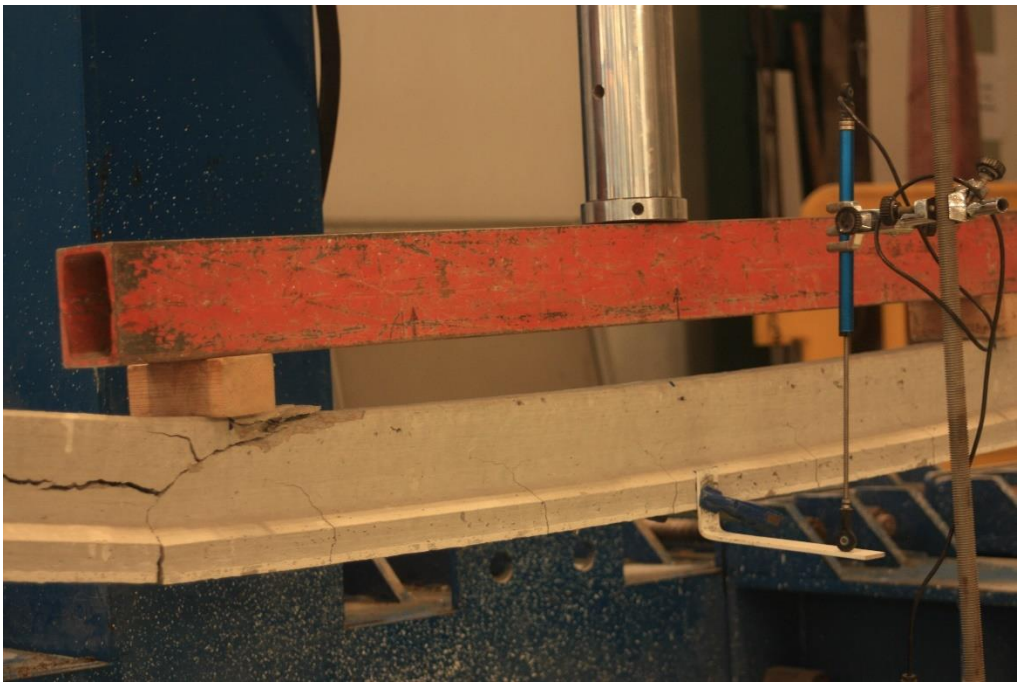


Figura 86. Rotura Vigueta T12.2 (Imagen H)

Se hace una máscara de recorte en cada fotografía delimitando la región de interés, en el caso de las imágenes introducidas se aplica un fondo negro segmentando la vigueta para que en la detección no se generen lecturas erróneas (*Figura* 87). Se hacen acercamientos a la imagen identificando los porcentajes de las zonas de interés.

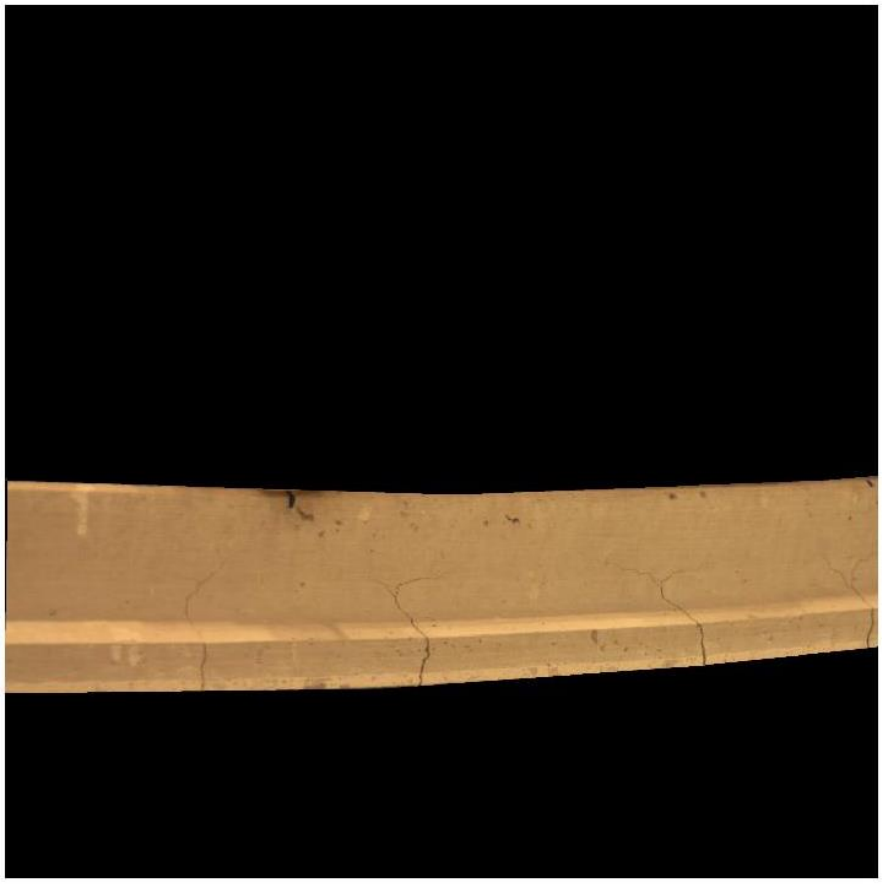


Figura 87. Mascara aplicada a la imagen G de la Vigueta T12.2

Ahora es necesario identificar en qué momento de la consecución de imágenes se empiezan a detectar como positivas las grietas, teniendo en cuenta el porcentaje de acercamiento a la región de interés, es decir, qué tamaño deben tener las grietas para que el algoritmo detecte un positivo. En la Figura 88 se muestran los porcentajes de acercamiento a la fisura, en este caso se toma la imagen G del ensayo.

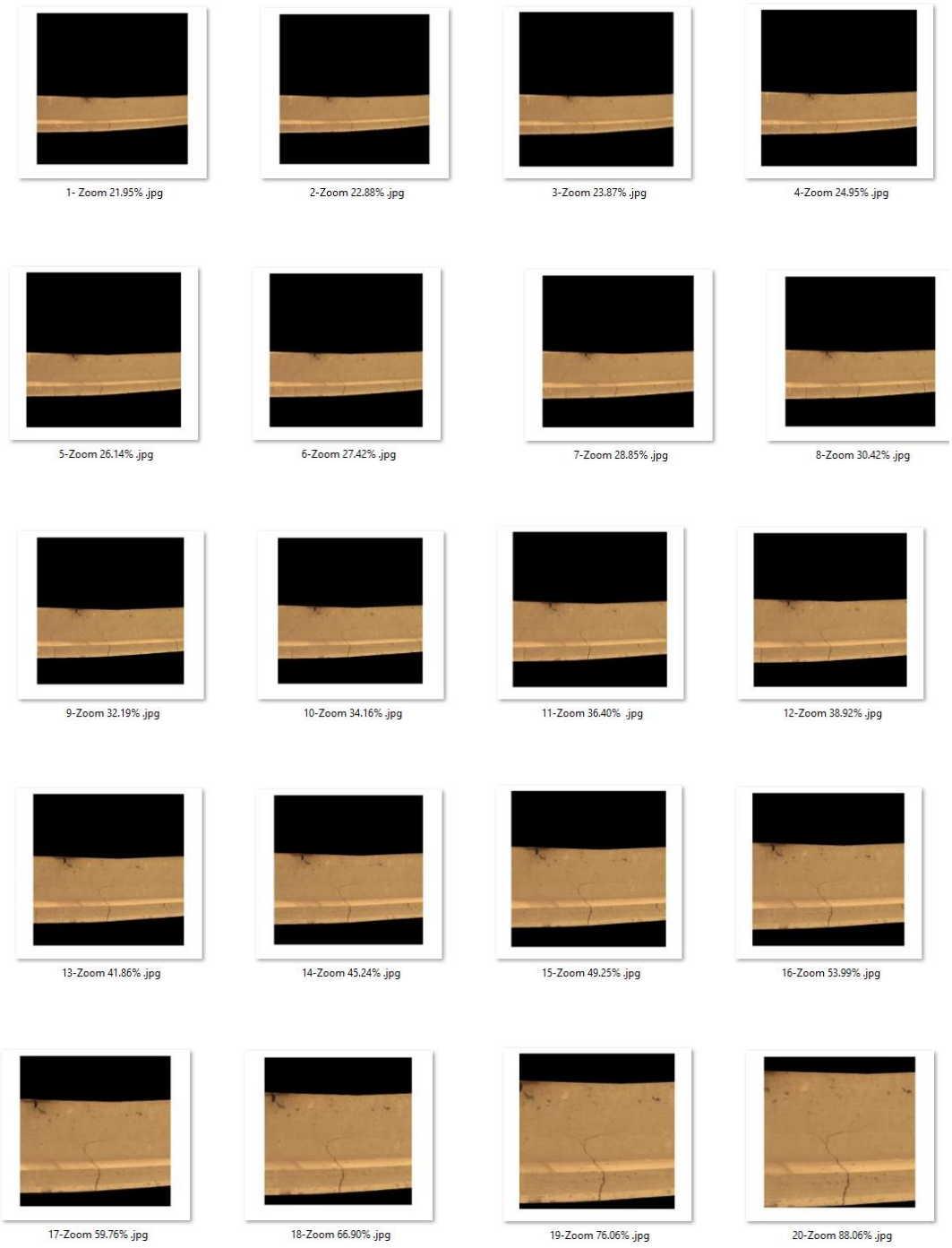


Figura 88. Porcentajes de la región de interés de la imagen G de la Vigueta T12.2

Las figuras 89 al 95 representan en el eje horizontal (X) el porcentaje de acercamiento de la imagen que se encuentra relacionado con el ancho de la fisura y en el eje vertical (Y) la precisión de que detecte como positiva la fisura.

A medida que la fisura se hace más grande, la herramienta comienza a detectar la fisura y a clasificarla como positiva. A partir de la figura Figura 91 la predicción es correcta y con un acercamiento del 60% a la imagen su precisión es mayor al 80% (figuras *Figura* 89 a 95)

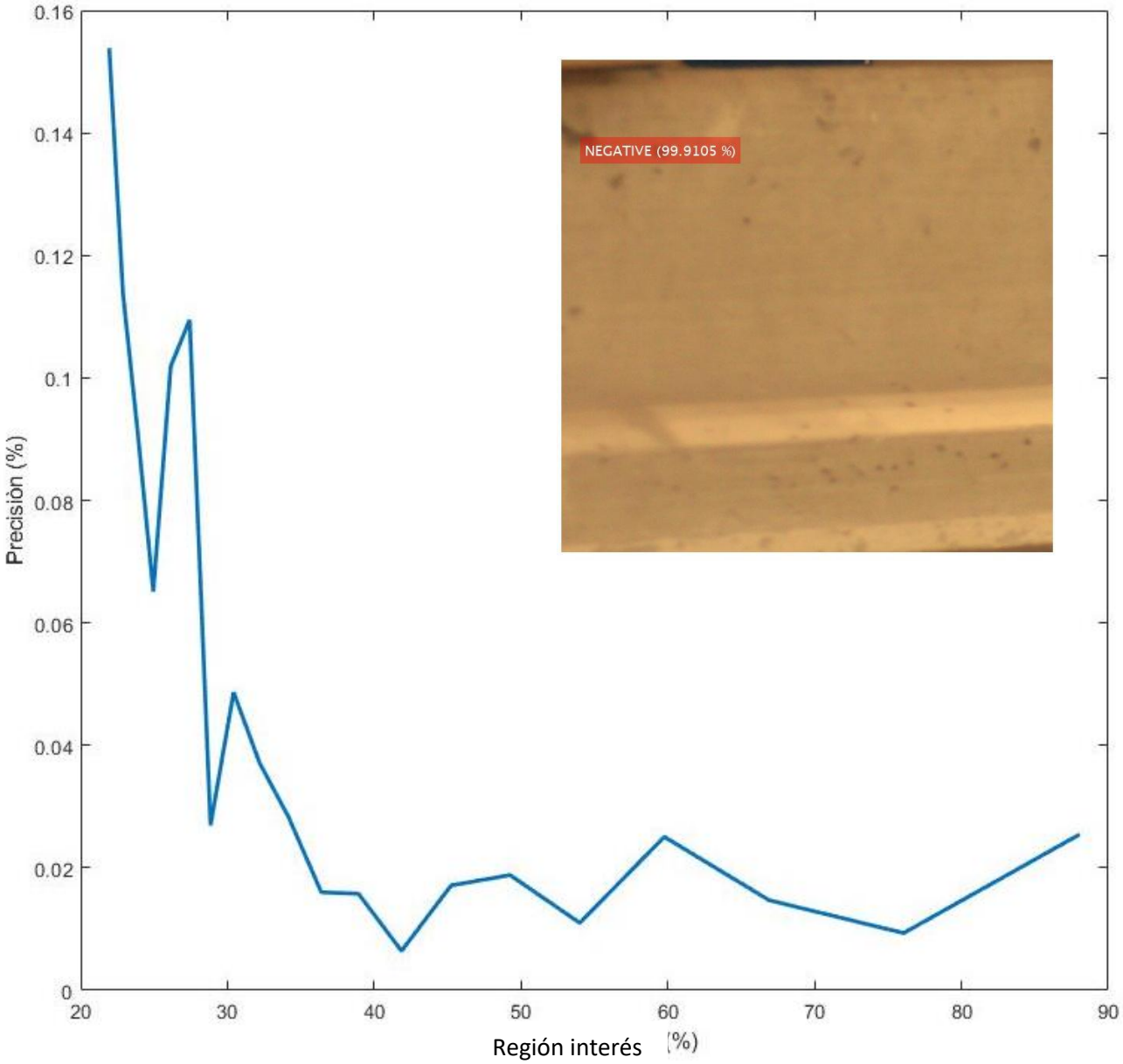


Figura 89. Clasificación de la imagen 1 clasificación negativa

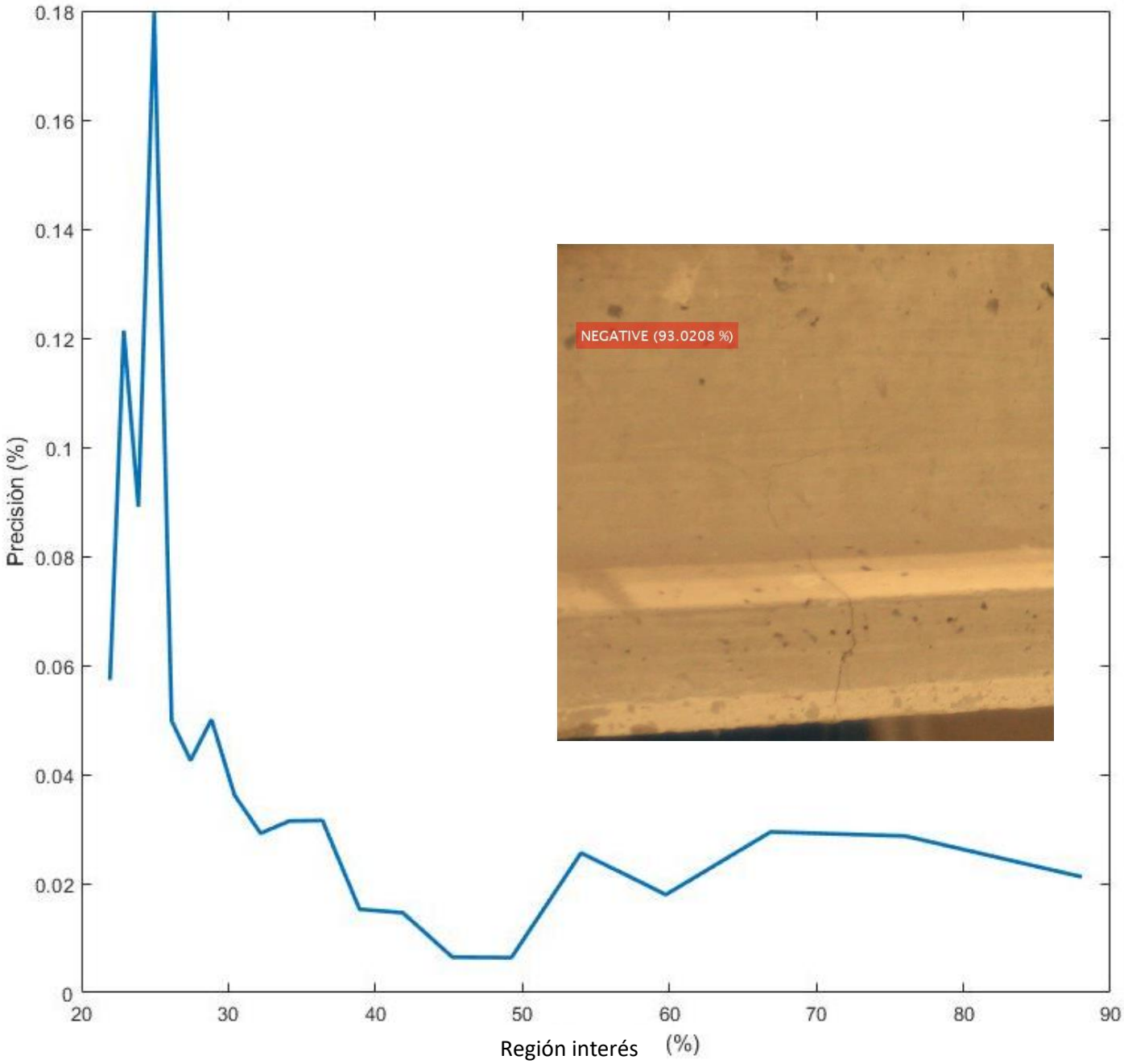


Figura 90. Clasificación de la imagen 2 clasificación negativa

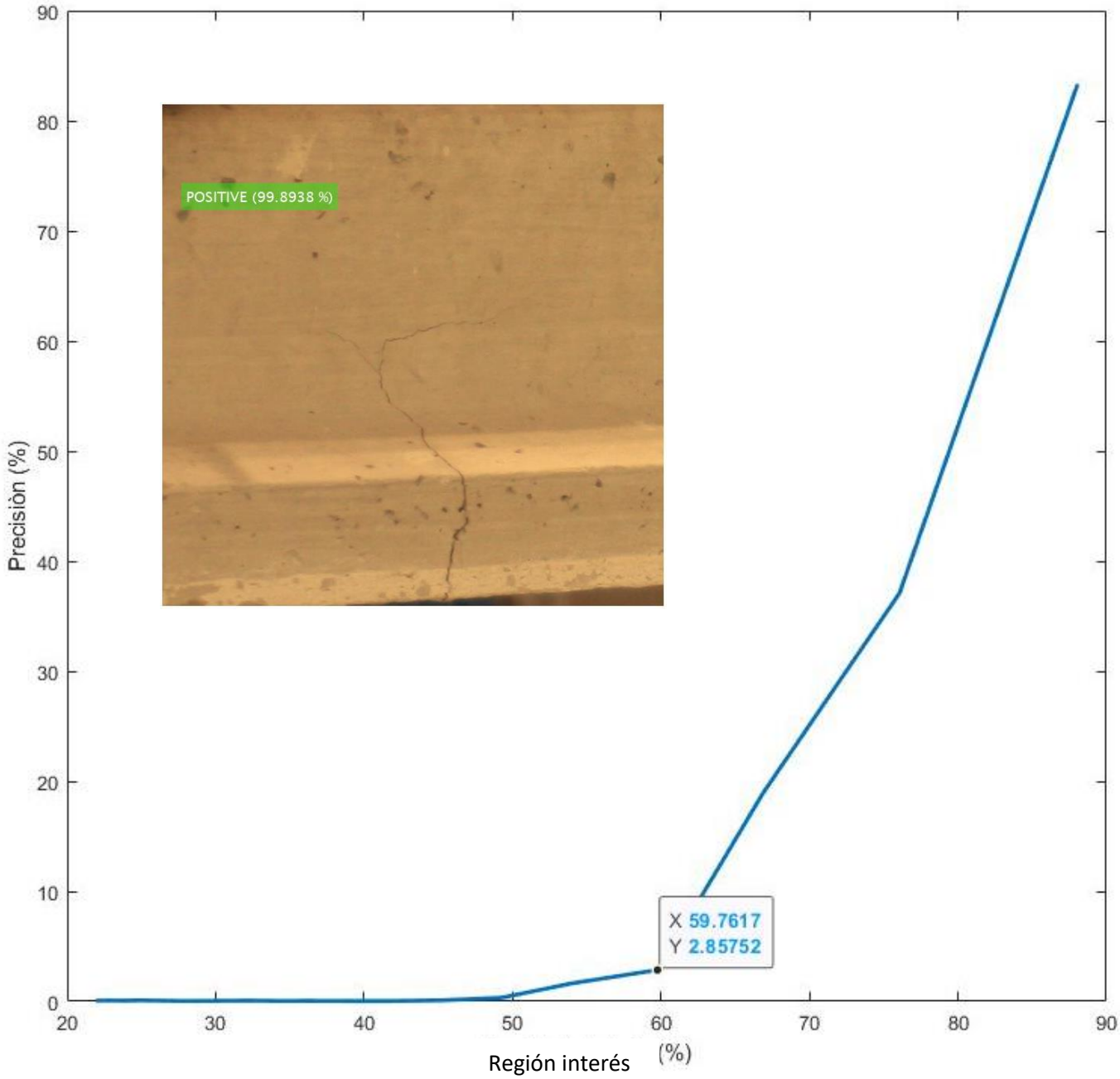


Figura 91. Clasificación de la imagen 3 clasificación positiva.

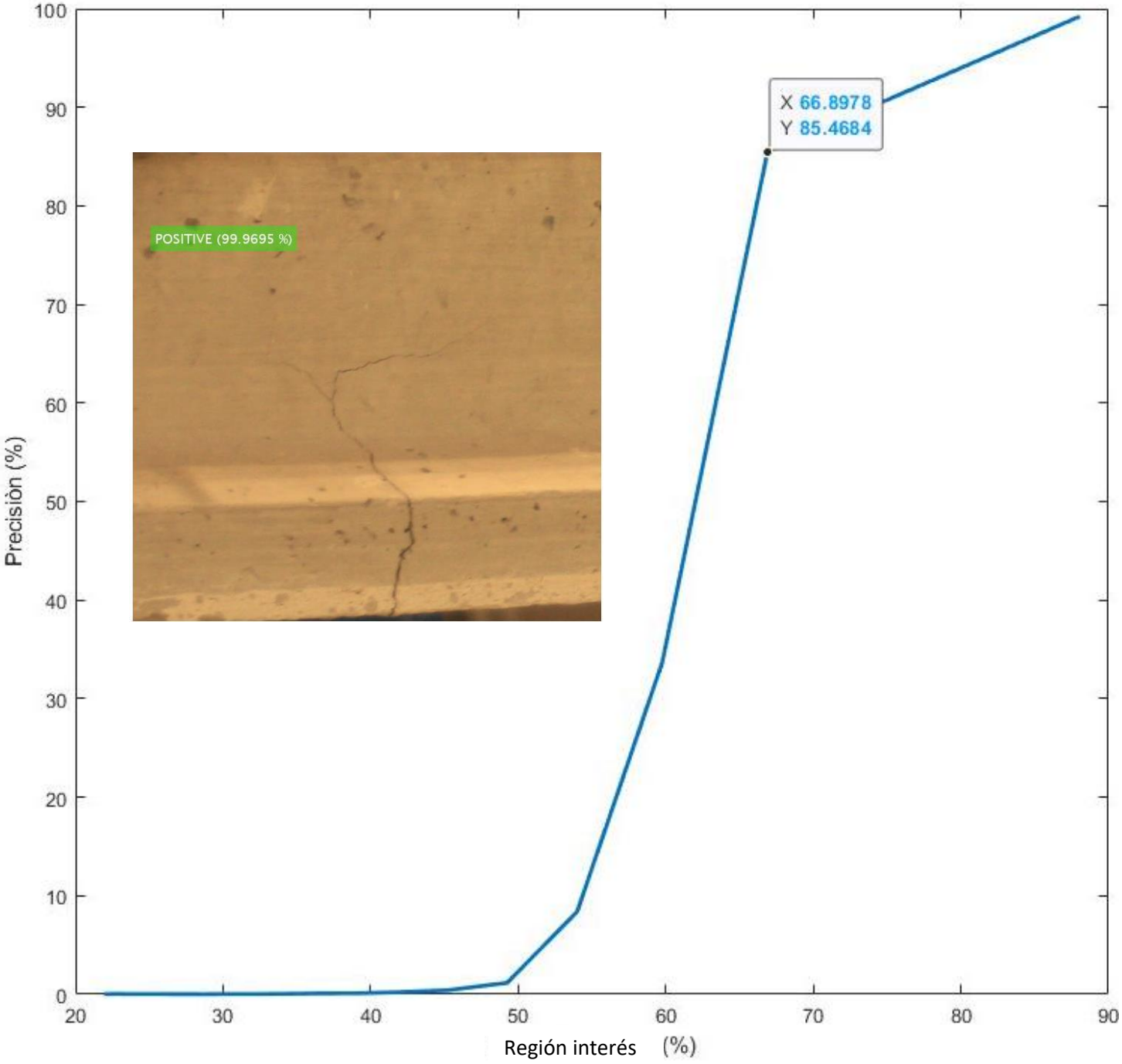


Figura 92. Clasificación de la imagen 4 clasificación positiva.

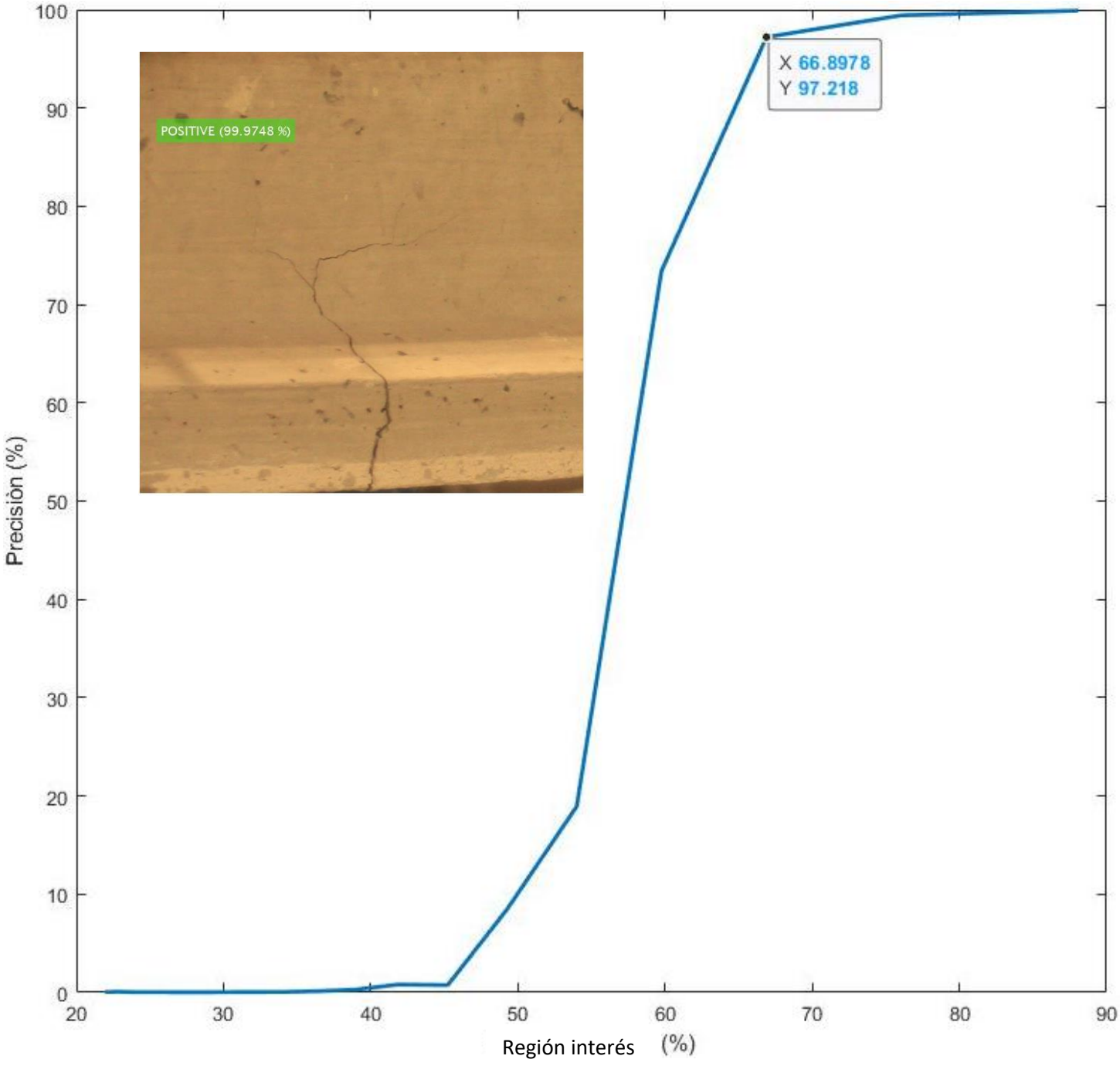


Figura 93. Clasificación de la imagen 5 clasificación positiva.

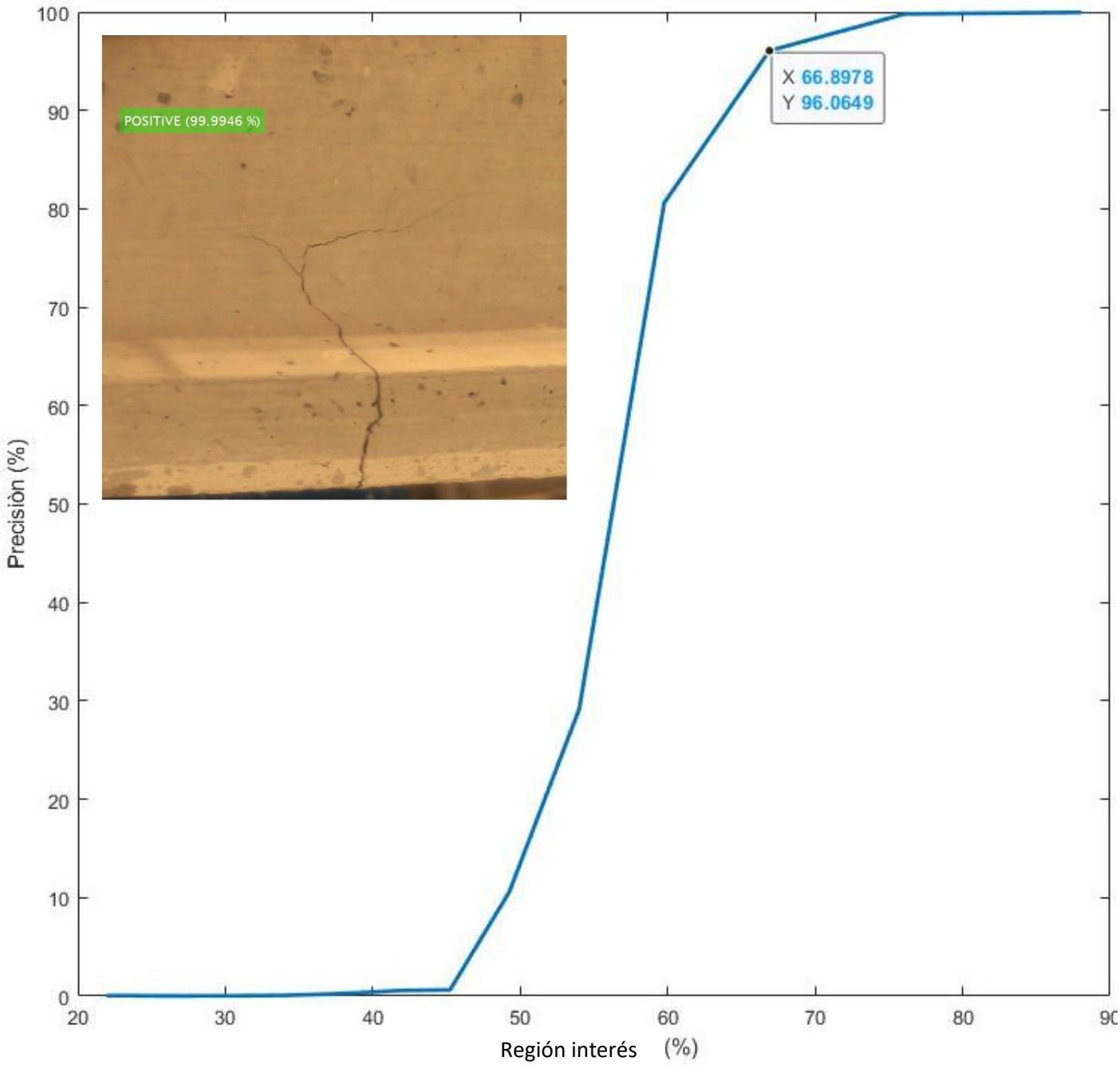


Figura 94. Clasificación de la imagen 6 clasificación positiva.

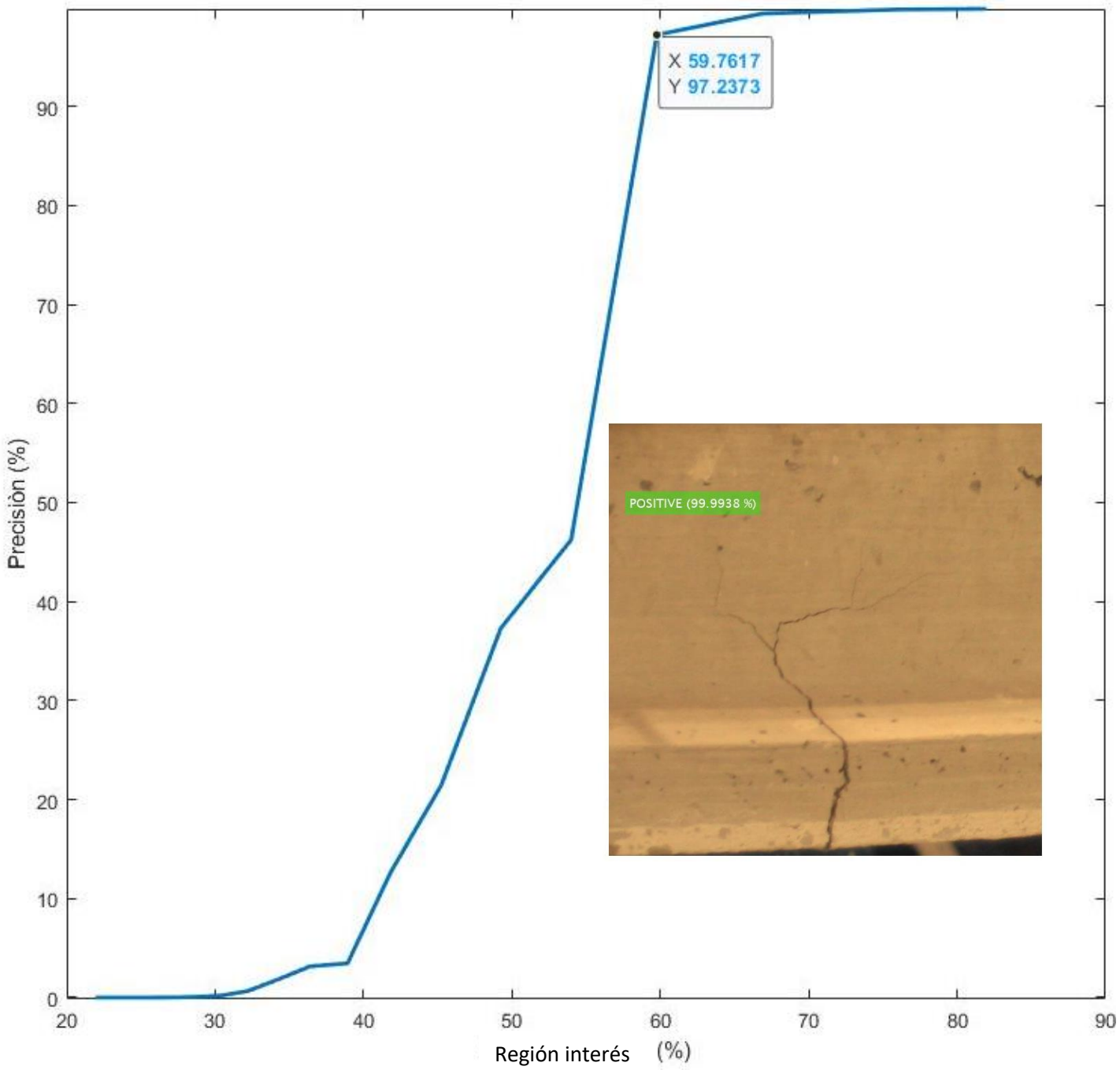


Figura 95. Clasificación de la imagen 7 clasificación positiva.

5 CONCLUSIONES

En este trabajo se ha propuesto un método de detección de fisuras utilizando tres redes neuronales convolucionales (CNN) diferentes, AlexNet, SqueezeNet y GoogleNet. Las CNN se han entrenado empleando bases de datos SDNET 2018 y METU de imágenes de fisuras en superficies de hormigón, que contaban con dos etiquetas: “positiva” para imágenes fisuradas y “negativa” para imágenes sin fisuras.

Las bases de datos eran relativamente pequeñas, por lo que el Transfer Learning (TL) fue especialmente adecuado tratar los datos. Por lo tanto, se aplicó esta técnica para detectar daños de fisuras en las superficies de hormigón. Se hicieron tres combinaciones de las bases de datos en la etapa de entrenamiento. La arquitectura que logró una mayor precisión de la prueba final fue GoogleNet con un 93,8% del conjunto de datos de prueba.

De los trabajos realizados se pueden extraer las siguientes conclusiones:

- Las redes neuronales convolucionales (CNN) pueden detectar fisuras con una precisión de más del 90%.

- La red neuronal con mayor precisión y menor gasto de memoria de disco duro fue GoogleNet. Es la arquitectura que cuenta con la mayor cantidad de capas profundas que a lo largo de todo el proceso de clasificación extraen y almacenan más características que en un futuro puedan ser utilizadas para mejorar la herramienta.

- La herramienta desarrollada puede identificar fisuras en estructuras sometidas a un proceso de carga hasta la rotura. El análisis de las capacidades de las CNNs para detectar las fisuras ha proporcionado una serie de parámetros que se deben de tener en cuenta a la hora tomar las fotografías de las fisuras como,

por ejemplo, la distancia a la que debe estar el objetivo de la cámara y la apertura mínima de fisura que permite detectarla.

Las técnicas actuales de IA no pueden considerarse totalmente automatizadas, y la percepción humana no es fácil de replicar mediante algoritmos de DL. Dichas capacidades deben abordarse en estudios futuros, incluida la importancia del daño con respecto al tipo de componentes estructurales, materiales, ubicaciones y otras condiciones ambientales.

En un futuro, deben investigarse con profundidad los aspectos siguientes:

- La naturaleza de los daños, así como su importancia, pueden diferir de un componente estructural a otro cuando se considera el contexto estructural global. Por lo tanto, se deben diseñar enfoques jerárquicos integrales en los que el reconocimiento de componentes estructurales se incluya como un primer paso esencial antes de usar datos de imagen.

- Las bases de datos de sistemas estructurales y otros componentes son limitadas para propósitos de la monitorización estructural, por lo cual es necesario estudiar y entrenar modelos cuando surgen nuevas condiciones, como texturas, juntas, luz, medio ambiente, contaminaciones. Los problemas relacionados con el medio ambiente no se pueden simular perfectamente a través de modelos numéricos generalizados; por lo tanto, una mejor base de datos solo se puede formar adquiriendo datos del mundo real.

- La naturaleza de los daños, así como su importancia, pueden diferir de un componente estructural a otro cuando se considera el contexto estructural global. Por lo tanto, se deben diseñar enfoques jerárquicos integrales en los

que el reconocimiento de componentes estructurales se incluya como un primer paso esencial antes de usar las bases de datos.

- Las redes neuronales facilitan la tarea de identificación de daños automatizando el proceso y alcanzando niveles muy aceptables de precisión. Sin embargo, en algunos casos, los inspectores no tienen acceso a todas las partes de las estructuras para adquirir datos de imágenes. Esta inspección es una tarea que demanda tiempo en la recolección de datos, el uso de drones se perfila como una excelente herramienta que minimiza la necesidad de trabajo físico además de ahorrar tiempo, ser rentables, seguras, disponibles y precisas.

- De esta forma se podría contar en un futuro con una herramienta de clasificación de fisuras que proporcionara una información detallada que incluyera, por ejemplo, la dirección de la fisura, su ancho y longitud, e incluso el patrón de fisuración de un elemento estructural, lo que p permitiría identificar el tipo de patología que sufre una estructura.

6 BIBLIOGRAFIA

Beale, M. H., Hagan, M. T., & Demuth, H. B. (2020). *Deep Learning Toolbox ™ User ' s Guide How to Contact MathWorks*.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., Darrell, T., Eecs, T., & Edu, B. (n.d.). *DeCAF : A Deep Convolutional Activation Feature for Generic Visual Recognition*.

Dorafshan, S., Thomas, R. J., & Maguire, M. (2018). Comparison of deep convolutional neural networks and edge detectors for image-based crack detection in concrete. *Construction and Building Materials*, *186*, 1031–1045. https://doi.org/10.1016/j.conbuildmat.2018.08.011

Feng, C., Zhang, H., Wang, S., Li, Y., Wang, H., & Yan, F. (2019). *Structural Damage Detection using Deep Convolutional Neural Network and Transfer Learning*. *23*, 4493–4502. https://doi.org/10.1007/s12205-019-0437-z

Gao, Y., & Mosalam, K. M. (2018). Deep Transfer Learning for Image-Based Structural Damage Recognition. *Computer-Aided Civil and Infrastructure Engineering*, *33*(9), 748–768. https://doi.org/10.1111/mice.12363

Iandola, F. N., Han, S., & Dally, W. J. (2016). *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and textless1MB model size*. *February*.

Kim, B., & Cho, S. (2018). Automated vision-based detection of cracks on concrete surfaces using a Deep Learning technique. *Sensors (Switzerland)*, *18*(10). https://doi.org/10.3390/s18103452

Kim, H., Ahn, E., Cho, S., Shin, M., & Sim, S. H. (2017). Comparative analysis of image binarization methods for crack identification in concrete structures. *Cement and Concrete Research*, *99*(March), 53–61. https://doi.org/10.1016/j.cemconres.2017.04.018

Navarro, I. J., Yepes, V., & Martí, J. v. (2019). A Review of Multicriteria Assessment Techniques Applied to Sustainable Infrastructure Design. *Advances in Civil Engineering*, *2019*. https://doi.org/10.1155/2019/6134803

Nguyen, C., Kawamura, K., & Nakamura, H. (2020). *Automatic Crack Detection for Concrete Infrastructures Using Image Processing and Deep Learning* . https://doi.org/10.1088/1757-899X/829/1/012027

Pan, S. J., & Yang, Q. (2010). *A Survey on Transfer Learning*. *22*(10).

Pandiyan, V., Murugan, P., Tjahjowidodo, T., Caesarendra, W., Manyar, O. M., & Then, D. J. H. (2019). In-process virtual verification of weld seam removal in robotic abrasive belt grinding process using Deep Learning . *Robotics and Computer-Integrated Manufacturing*, *57*(October), 477–487. https://doi.org/10.1016/j.rcim.2019.01.006

Rawat, W., & Wang, Z. (2017). *Deep Convolutional Neural Networks for Image Classification : A Comprehensive Review Deep Convolutional Neural Networks for Image Classification : A Comprehensive Review*. *October*. https://doi.org/10.1162/NECO

Salehi, H., & Burgueño, R. (2018). Emerging artificial intelligence methods in structural engineering. *Engineering Structures*, *171*(November 2017), 170–189. https://doi.org/10.1016/j.engstruct.2018.05.084

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *07-12-June*(June), 1–9. https://doi.org/10.1109/CVPR.2015.7298594

Traore, B. B., Kamsu-Foguem, B., & Tangara, F. (2018). Deep convolution neural network for image recognition. *Ecological Informatics*, *48*, 257–268. https://doi.org/10.1016/j.ecoinf.2018.10.002

Valueva, M. v, Nagornov, N. N., Lyakhov, P. A., Valuev, G. v, & Chervyakov, N. I. (2020). ScienceDirect Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Mathematics and Computers in Simulation*, *177*, 232–243. https://doi.org/10.1016/j.matcom.2020.04.031

Vu, C., Sekiya, H., Hirano, S., Okatani, T., & Miki, C. (2019). Automation in Construction A vision-based method for crack detection in gusset plate welded joints of steel bridges using deep convolutional neural networks. *Automation in Construction*, *102*(July 2018), 217–229. https://doi.org/10.1016/j.autcon.2019.02.013

7 GLOSARIO

Augmentation: Proceso para aumentar el número de imágenes del conjunto de dato.

Algoritmo paralelo: Algoritmo que puede ser ejecutado por partes en el mismo instante de tiempo por varias unidades de procesamiento, para finalmente unir todas las partes y obtener el resultado correcto

Batch: Conjunto de datos de tamaño finito (Lote).

Bias: Valor de sesgo que se añade a la red neuronal.

Convolutional Neural Network (CNN): Tipo de redes neuronales profundas mayormente utilizadas en la visión artificial o visión por computador.

Deep Learning (DL): Conjunto de algoritmos de aprendizaje automático que emula el aprendizaje humano con el fin de obtener ciertos conocimientos.

Epoch: Iteración completa de aprendizaje de todo el conjunto de datos en fase de entrenamiento.

Fine-Tuning (FT): técnica que toma valores de una red neuronal ya entrenada y los utiliza como inicio para hacer un nuevo modelo que se está entrenando con datos similares.

ImageNet : Conjunto de datos que se utiliza para evaluar algoritmos de clasificación, localización y reconocimiento de imágenes.

Kernels: Filtros que se aplican a una imagen para extraer características importantes o patrones de esta….

Machine Learning (ML): Rama de la inteligencia artificial que desarrolla algoritmos para que las maquias aprendan de forma automática.

Max Pooling: Agrupación que hace un filtro que reduce el tamaño de la imagen, agrupando varios pixeles en uno, ayudando a reducir la carga computacional para el próximo proceso.

Neural Network (NN): Modelo computacional que pretende simular la estructura de una red neuronal biológica a través de neuronas artificiales conectadas. Los datos de entrada atraviesan la red neuronal efectuando diferentes operaciones y finalmente se obtienen unos valores de salida.

Transfer Learning (TL): Técnica que centra en almacenar el conocimiento adquirido en la resolución de un problema para aplicarse a un problema diferente pero similar.