Universidad Autónoma de Sinaloa

Facultad de Informática Culiacán

Facultad de Ciencias de la Tierra y el Espacio

Posgrado en Ciencias de la Información

IMPLEMENTACIÓN DE ALGORITMOS GENÉTICOS PARA LA OPTIMIZACIÓN DE HIPERPARÁMETROS EN MODELOS DE CLASIFICACIÓN PARA LA DETECCIÓN DE GRIETAS EN PUENTES DE CONCRETO REFORZADO

TESIS

Como requisito para obtener el grado de

Maestro en Ciencias de la Información

Presenta:

Eliel Avilez Valenzuela

Directores:

Dr. Arturo Yee Rendón

Dr. José Ramón Gaxiola Camacho

Culiacán, Sinaloa. Enero de 2025

Agradecimientos

Agradezco a mis asesores, el Dr. Arturo Yee Rendón y el Dr. José Ramón Gaxiola Camacho. Gracias por sus enseñanzas, su guianza, y el siempre mostrarse disponibles para atender el proyecto como es debido.

A mis profesores del posgrado por ayudar a mi formación.

A mi esposa Dalia por siempre apoyarme incondicionalmente, tener confianza en mí, y paciencia durante el posgrado.

A mi madre por siempre alentarme a estudiar más allá de la licenciatura.

A la Universidad Autónoma de Sinaloa por permitirme estudiar el Posgrado en Ciencias de la Información en la Facultad de Informática Culiacán, y al Consejo Nacional de Humanidades, Ciencias y Tecnologías por otorgarme el beneficio de ser becado (CVU: 1224395) para cursar mis estudios de posgrado.

Por último, al Dios creador del universo, que otorga sabiduría y gracia al que se lo pide.

Índice general

Índice de figuras

Índice de tablas

Resumen

Los modelos predictivos de Inteligencia Artificial (IA) basados en redes neuronales artificiales son ampliamente utilizados para resolver problemas de clasificación y regresión en las distintas áreas de la ciencia. Por ejemplo, en la Física se generan modelos que permiten clasificar tipos de eventos producidos por las colisiones de partículas, determinar enfermedades oculares, como la retinopatía diabética, el glaucoma, entre otras. En Biología, se generan modelos que permiten clasificar las diferentes especies de plantas, identificación de patógenos, predecir el crecimiento de poblaciones de especies, entre otros. En la Ingeniería civil, se generan modelos para identificar grietas en concreto, clasificar diferentes tipos de suelo, estimar la vida útil de estructuras civiles, entre otros. Estos modelos son sensibles a los valores de sus parámetros, hiperparámetros estructurales, y a sus hiperparámetros de entrenamiento. Por lo tanto, es crucial optimizar estos elementos para asegurar el correcto funcionamiento del modelo y lograr la tarea deseada, minimizando el margen de error en la predicción. La determinación de los valores de estos elementos suele realizarse mediante experimentación, a veces utilizando un enfoque de prueba y error, o incluso adoptando valores predeterminados (por defecto). Sin embargo, el uso de valores predeterminados puede, en muchas ocasiones, no ser suficiente para generar modelos adecuados.

Existe un problema especialmente con los hiperparámetros de la red neuronal, estos elementos no pueden ser aprendidos por el modelo en su entrenamiento. Por esta razón, en este trabajo se propone el uso de técnicas metaheurísticas, particularmente evolutivas como mecanismo de búsqueda para explorar de manera eficiente el espacio de posibilidades y encontrar valores apropiados de hiperparámetros de la red neuronal, para así llegar a un modelo predictivo que tenga un buen desempeño con estos valores encontrados.

Las estructuras civiles, como carreteras, puentes, presas, túneles y pavimentos, frecuentemente enfrentan elevadas tensiones físicas debido a diversos factores, tales como terremotos, explosiones, o simplemente el uso cotidiano. Estos eventos pueden ocasionar desde el colapso total de la estructura hasta daños físicos, comúnmente manifestados en forma de grietas. La detección de grietas, en imágenes digitales, supone un desafío debido a su forma irregular, que no tiene

una estructura o tamaños específicos. En este trabajo de investigación, se propone, como caso de estudio, generar modelos predictivos optimizados con nuestra metodología, para la detección temprana de grietas en estructuras de concreto reforzado, y así poder estimar la condición estructural que guardan diversas estructuras de ingeniería civil.

Los resultados obtenidos en el presente trabajo de investigación demuestran que el uso de algoritmos genéticos permite optimizar hiperparámetros de redes neuronales artificiales de manera efectiva. Gracias al uso de este método de búsqueda, se pudieron encontrar configuraciones adecuadas de hiperparámetros, lo que permitió mejorar significativamente el desempeño de los modelos. Utilizando la Puntuación $F1$ como métrica de desempeño, se logró una mejora notable en los modelos de clasificación para la detección de grietas en el concreto, elevándola de 0.76 a 0.90. Esto ha permitido generar modelos más efectivos para la inspección de obras civiles.

Abstract

Artificial Intelligence (AI) predictive models based on artificial neural networks are widely used to solve classification and regression problems in different scientific fields. For example, in Physics, several models are used for the classification of particle collision events to detect optical diseases like diabetic retinopathy, glaucoma, among other things. In Biology, some these models are used to classify different type of plants, to identify pathogens, to predict population growth of certain species, etc. In Civil Engineering, the above-mentioned models are commonly used for different applications as concrete crack detection, classification of soil types, estimation of service life of structures, among other applications. In general, these models are sensitive to many factors as parameters configuration, structural hyperparameters, and the training hyperparameters. Hence, the above-documented elements must be adjusted for the model to work properly and get the expected results reducing the error of the prediction. These values are established by testing, often by trial and error, and even in some cases, default values are assigned to them, sometimes this may prevent the generation of proper-working models.

Within this frame of reference, there is a particular problem related to hyperparameters of the net, these elements can't be learned by the model in the training phase. Therefore, in this research, a metaheuristic approach is proposed, particularly an evolutionary one to be used as search method to efficiently explore the space of possibilities and appropriate values for the hyperparameters of the neural network, with the objective to create a model with a good performance based on the already found values.

Infrastructure like roads, bridges, dams, tunnels, and pavements, frequently face structural challenges due to several factors as earthquakes, blasts, or simply the daily usage. These events could cause the total collapse of the structure, or physical damages usually manifested with concrete cracks. In this sense, crack detection with digital images is a challenge, due to its irregular shape, without a defined size or structure. In this research work, it is proposed to generate a predictive model based on a novel and unique methodology for early crack detection in reinforced concrete structures, and thus, estimate the structural condition of several engineering structures.

The obtained results in this research study, show that genetic algorithms allow to optimize neural networks hyperparameters succesfuly. Due to this search method usage, there were newfound hyperparameter configurations that led to a F1 Score improvement in classification models for concrete crack detection, F1 Score went from 0.76 to 0.90, allowing the generation of better models for infrastructure monitoring.

Capítulo 1

Introducción

En la actualidad, para tratar problemas de clasificación y regresión se utilizan modelos basados en redes neuronales artificiales (RNA) los cuales cuentan con una gran popularidad, entre ellos se encuentran las técnicas de aprendizaje de máquina convencional o aprendizaje profundo. Por ejemplo, los modelos basados en aprendizaje profundo se ha utilizado para clasificar objetos en imágenes digitales de distintas especies de plantas, detección de enfermedades a través de imágenes médicas, clasificar y/o detectar los objetos en las imágenes de concreto para detectar grietas en obras civiles (Mohan y Poobal, 2018), entre otras aplicaciones.

1.1 Contexto del problema

Las RNA son muy utilizadas en la disciplina del aprendizaje de máquina para resolver diversos problemas. El aprendizaje de máquina es una subárea de la inteligencia artificial, la cual tiene como objetivo modelar matemáticamente la relación que existe entre un conjunto de características que describen a un objeto y una variable objetivo que actúa como etiqueta para determinar la pertenencia del objeto a una clase o grupo específico. Esta tarea se conoce como clasificación, y los modelos basados en RNA han demostrado ser muy útiles a través de los años. El aprendizaje de máquina engloba el aprendizaje de máquina convencional y al aprendizaje profundo.

El desempeño de los modelos de aprendizaje de máquina depende de los valores de sus parámetros (Kaveh y Mesgari, 2023). Existen en la literatura varias técnicas y enfoques para

optimizar los valores de los parámetros de las RNA, como: pesos, y sesgos. Los pesos son los valores que determinan la contribución relativa de la entrada con la salida de una neurona, lo que permite determinar la importancia de cada conexión entre dos neuronas. Los sesgos son aquellos valores que permiten al modelo aprender no solo la pendiente de una función activación, sino también trasladar o desplazar el punto de intersección de la función generada, lo que ayuda a adaptar el modelo a los datos de entrada. En la actualidad, existe una gran cantidad de algoritmos para optimizar los parámetros de la red como pesos y sesgos. Estos algoritmos realizan actualizaciones de los valores de los pesos y sesgos durante la fase de entrenamiento de la RNA, de modo que estos valores son ajustados para poder disminuir el error en las predicciones del modelo (Desai, 2020). A esto se le conoce como "inducción", o "entrenar el modelo".

No obstante, en contraste con los parámetros de una red neuronal artificial (RNA), los hiperparámetros de entrenamiento y estructurales de la red no son aspectos que el modelo puede aprender durante la etapa de inducción. Los hiperparámetros de entrenamiento afectan directamente el proceso de inducción del modelo, algunos de ellos son: la tasa de aprendizaje, el tamaño de lotes de datos, valores y tasas para técnicas de regularización, etc. También, existen hiperparámetros estructurales de la red, estos definen la arquitectura y distribución estructural de la red neuronal, algunos ejemplos son: la cantidad de capas, las conexiones entre capas, los tipos de capas, así como la cantidad y forma de los filtros redes neuronales convolucionales (RNC). Estos elementos son fundamentales para la configuración y el rendimiento de las redes neuronales (Khan, 2018). La ausencia de una metodología común definida en la asignación de valores para los hiperparámetros y la arquitectura de la RNA ha llevado a que, en general, la comunidad científica asigne estos valores utilizando convenciones establecidas, basándose en experiencias previas, o utilizando valores conocidos que se hayan utilizado en otros modelos (Chung y Shin, 2020). Esto añade un factor de arbitrariedad en la asignación de valores para hiperparámetros y estructura de la red, que puede resultar en la generación de un modelo que no sea adecuado para el caso de estudio que se esté estudiando.

Explorar todas las combinaciones posibles de valores para hiperparámetros y estructuras de una RNA no es viable debido a la gran cantidad de combinaciones que existen, lo que resultaría en un proceso tardado. Sin embargo, se puede recurrir a técnicas de búsqueda eficientes para encontrar los valores adecuados para el modelo que se desee construir (Fujino y col., 2017).

Los algoritmos genéticos (AG) son técnicas de búsqueda pertenecientes a la rama del cómputo evolutivo, el funcionamiento de estos algoritmos se basa en la teoría de la evolución de las especies, utilizando el mecanismo de adaptación y supervivencia de los individuos más aptos (Katoch y col., 2021).

Una de las ventajas más notables de los algoritmos evolutivos es que pueden encontrar soluciones buenas en grandes espacios de búsqueda en relativamente poco tiempo, esto es conveniente porque permite evaluar diversas combinaciones sin alargar significativamente el tiempo de experimentación, dando como resultado la obtención de buenas soluciones que faciliten generar modelos basados en RNA que tengan un desempeño correcto para el caso de estudio (Aszemi y Dominic, 2019). Con una metodología basada en la generación de modelos a través de optimización de hiperparámetros y arquitectura de RNA, se pueden asignar valores de forma más objetiva a estos componentes del modelo, eliminando así la selección de valores de manera arbitraria.

Con la capacidad de generar modelos de clasificación a la medida dependiendo del problema a través del uso de algoritmos genéticos, se pueden estudiar casos de estudio relevantes en la sociedad como lo es la evaluación de la condición estructural de obras civiles, edificaciones en las cuales los humanos desarrollamos prácticamente la totalidad de nuestras actividades, y nuestra seguridad depende de la condición de dichas estructuras, ya que un edificio o un puente en malas condiciones puede presentar fallas y poner en riesgo la seguridad de los usuarios.

En el área de Ingeniería Civil, el estado del arte en investigación y desarrollo se enfoca en aplicaciones que buscan mejorar diversos aspectos de la vida cotidiana y la infraestructura urbana. Por ejemplo, una tendencia es la utilización de modelos predictivos para la detección de grietas en el concreto en diversas obras civiles (Gibb y col., 2018), como edificios, puentes, túneles, pavimento, etc. Y con esto, se tiene la capacidad de evaluar el nivel de deterioro que podría presentar una estructura. Las obras civiles ya mencionadas suelen ser sometidas a esfuerzos considerables, causados por múltiples factores, como: desastres naturales, accidentes, uso diario, entre otros. En muchas ocasiones estos esfuerzos pueden provocar un colapso si el daño es muy grave, o bien pueden formarse grietas en el concreto (Munawar y col., 2021).

Las grietas en una estructura civil como un puente, pueden disminuir la capacidad de carga del puente, causar discontinuidades en la superficie, propiciar la corrosión de los componentes

internos del puente como el acero que refuerza la esctructura, ya que las grietas pueden permitir el paso de agua y diversas sustancias al interior de la estructura, mitigando así las capacidades que tiene el puente de soportar esfuerzo físico, y por consiguiente poniendo en riesgo la integridad del mismo y de sus usuarios (Liu y col., 2023). Por lo tanto, es fundamental tener la capacidad de detectar grietas en obras civiles, ya que siendo detectadas en una etapa temprana de su formación, pueden ser evaluadas y se pueden aplicar medidas preventivas o correctivas según sea el caso aplicable, de esta forma se pueden evitar catástrofes como un colapso de la estructura, el cual conllevaría pérdidas monetarias significativas, un largo tiempo de reparación y reestructuración, y potenciales daños a la integridad de las personas que sean usuarios de dicha estructura (Mohan y Poobal, 2018).

Los métodos tradicionales de inspección y detección de grietas en el concreto están estrechamente ligados a la subjetividad humana, ya que en muchas ocasiones las inspecciones son realizadas manualmente por personas, implicando que se debe tomar en cuenta el nivel de experiencia de los inspectores, las condiciones físicas y mentales del inspector en ese momento, climatología, entre otras características (Zhang y col., 2021). Además, estos métodos de inspección manuales, suelen tomar largos periodos de tiempo, son costosos, y pueden poner en riesgo la integridad de los inspectores, ya que muchas veces la revisión de la condición estructural de una obra civil requiere acceder a zonas de gran altura o difícil acceso. Debido a todas estas circunstancias, es viable y de gran ayuda contar con herramientas altamente eficientes como los modelos predictivos entrenados para poder detectar grietas en superficies de concreto, para así tener un elemento computacional que no esté sujeto a la subjetividad humana, además que puede ahorrar costos monetarios, tiempo, y por supuesto evita poner en riesgo a un inspector humano al inspeccionar obras civiles.

El principal objetivo de este trabajo de investigación es proponer una metodología de optimización de hiperparámetros utilizando un método de búsqueda poblacional, como los algoritmos genéticos. Este enfoque permite encontrar combinaciones óptimas de hiperparámetros a través de una búsqueda guiada, generando modelos de clasificación basados en redes neuronales artificiales. El algoritmo genético busca maximizar la puntuación $F1$ de las soluciones encontradas para así poder mejorar el desempeño de los modelos. El caso de estudio se centra en la clasificación de objetos de interés en imágenes, específicamente en la detección de grietas en concreto

reforzado.

1.2 Definición formal del problema

El encontrar los valores de hiperparámetros para una red neuronal para maximizar su puntuación $F1$ se puede expresar formalmente como un problema de optimización. Un problema de optimización puede ser descrito como: encontrar los valores de n variables $[x_1, x_2, ..., x_n]$, contenidas en el vector $\vec{x}$, que optimizan la función objetivo $f(\vec{x})$. El problema de optimización puede ser expresado como:

$$\text{Encontrar } \vec{x} = [x_1, x_2, ..., x_n] \text{ que maximice } f(\vec{x}) \tag{1.1}$$

En este caso particular, el problema de optimización consiste en encontrar los valores de los hiperparámetros de una red neuronal artificial por medio de algoritmos genéticos para mejorar su desempeño en una tarea de clasificación binaria, maximizando el valor de su puntuación $F1$. El vector que contiene los valores posibles de los hiperparámetros puede ser expresado como:

$$\vec{x} = [x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8] \text{ sujeto a:} \tag{1.2}$$

- $x_1 \in \{0, 1, 2, 3\}$.

  Tal que x_1 representa un método de inicialización de parámetros, donde:

  - $x_1 = 0$, denota utilizar inicialización *LeCun Uniform*.
  - $x_1 = 1$, denota utilizar inicialización *LeCun Normal*.
  - $x_1 = 2$, denota utilizar inicialización *Glorot Uniform*.
  - $x_1 = 3$, denota utilizar inicialización *Glorot Normal*.

- $x_2 \in \{512, 1024, 2048, 4096\}$.

  Tal que x_2 representa la cantidad de neuronas en la primera capa oculta.

- $x_3 \in \{512, 1024, 2048, 4096\}$.

  Tal que x_3 representa la cantidad de neuronas en la segunda capa oculta.

- $x_4 \in \{1, 2, 3, 4, 5\}$.

  Tal que x_4 representa la función de activación, donde:

  - $x_4 = 1$, denota la utilización de la función de activación Sigmoide.
  - $x_4 = 2$, denota la utilización de la función de activación *ReLU*.
  - $x_4 = 3$, denota la utilización de la función de activación *Tanh*.
  - $x_4 = 4$, denota la utilización de la función de activación *SELU*.
  - $x_4 = 5$, denota la utilización de la función de activación *Tansig*.

- $x_5 \in \{0 \cup [0.5, 0.9]\}$.

  Tal que x_5 representa la tasa de difusión.

- $x_6 \in [10^{-5}, 10^{-2}]$.

  Tal que x_6 representa la tasa de aprendizaje.

- $x_7 \in \{8, 16, 32, 64\}$.

  Tal que x_7 representa el tamaño de lotes de entrenamiento.

- $x_8 \in \{1, 2, 3, 4\}$.

  Tal que x_8 representa el algoritmo de optimización de parámetros, donde:

  - $x_8 = 1$, denota la utilización del algoritmo de optimización *SGD*.
  - $x_8 = 2$, denota la utilización del algoritmo de optimización *AdaDelta*.
  - $x_8 = 3$, denota la utilización del algoritmo de optimización *RMSProp*.
  - $x_8 = 4$, denota la utilización del algoritmo de optimización *ADAM*.

Se pretende encontrar los valores de $\vec{x}$ utilizando un modelo evolutivo, tal que maximice la puntuación $F1$ de clasificación. Esto permitirá la correcta creación de modelos predictivos a ser utilizados en la detección de grietas en puentes de concreto reforzado.

1.3 Objetivos

1.3.1 Objetivo general

Implementar una metodología de optimización de hiperparámetros para un modelo de detección de grietas en concreto reforzado en imágenes, basado en RNC con técnicas de algoritmos genéticos.

1.3.2 Objetivos específicos

- Realizar un estudio exhaustivo del estado del arte.
- Recopilar y procesar el conjunto de datos de imágenes con grietas en superficies de concreto.
- Generar propuesta metodológica.
- Aplicar la metodología generada en la detección de grietas en puentes de concreto reforzado.

1.4 Hipótesis

La utilización de algoritmos genéticos como técnicas de optimización de hiperparámetros para un modelo de clasificación basado en redes neuronales artificiales, permitirá mejorar el desempeño del modelo base de la arquitectura *VGG-16*, incrementando la puntuación $F1$ por 10 %.

Los capítulos subsecuentes se dispondrán de la siguiente manera: en el Capítulo 2, se presenta el marco teórico para este trabajo de tesis, así como un estudio del estado del arte sobre las distintas técnicas que existen para elección de hiperparámetros de una RNA, el funcionamiento de los algoritmos genéticos, y también se incluirán distintas téncicas que existen en la literatura para detección de grietas en concreto. En el Capítulo 3, se presenta el esquema metodológico basado en cómputo evolutivo propuesto para la optimización de hiperparámetros de la RNA para el modelo de clasificación. En el Capítulo 4, se presentan los resultados obtenidos en la evaluación de los modelos de clasificación generados por la metodología, así como su interpretación y

discusión. En el Capítulo 5, se relatan las conclusiones del trabajo, así como posibles líneas de investigación futuras.

Capítulo 2

Conceptos básicos, antecedentes, y estado del arte

En este capítulo, se presentan los fundamentos teóricos de las técnicas utilizadas en el presente trabajo de tesis, así como el trabajo relacionado en el estado del arte que abona sustento al mismo.

La inteligencia artificial (IA) es una rama de las Ciencias de la Computación que consiste en la automatización de tareas complejas por medio de técnicas matemáticas y algorítmicas cuando una metodología de programación tradicional puede no ser viable. La IA ha sido de gran ayuda a lo largo de las últimas décadas, contribuyendo al crecimiento de otras ramas de la ciencia facilitando la resolución de problemas que de otra forma serían intratables. Algunos ejemplos son: algoritmos para realizar búsquedas, reconocimiento de patrones, problemas de visión por computadora, coches autónomos, entre otros.

El aprendizaje de máquina por otro lado, es una disciplina que forma parte de la inteligencia artificial e incluye tanto el aprendizaje de máquina convencional como el aprendizaje profundo. El aprendizaje de máquina convencional tiene como objetivo modelar matemáticamente la relación que existe entre un conjunto de características que describen a un objeto y una variable objetivo, que actúa como etiqueta para indicar la pertenencia del objeto a una clase o grupo específico. Este tipo de tarea se conoce como clasificación. Además, en esta área también se abordan problemas de regresión, que consisten en predecir un valor real a partir de un conjunto de características o datos históricos. Por otro lado, el aprendizaje profundo utiliza distintas

capas sucesivas para procesar los datos, permitiendo jerarquías de abstracción y representación de datos para así poder extraer características representativas de los datos a procesar.

Las redes neuronales artificiales (RNA) son técnicas de aprendizaje de máquina convencional que pueden realizar tareas de clasificación y regresión. Estas redes pueden trabajar con un conjunto de características extraídas previamente o aprender automáticamente características relevantes a partir de datos crudos. Actualmente, las técnicas de aprendizaje profundo han sido ampliamente utilizadas para resolver problemas complejos en la clasificación y/o detección de objetos en imágenes digitales. Particularmente, las redes neuronales convoluciones (RNC) son técnicas de aprendizaje profundo que se derivan de las RNA. Las RNC tienen la capacidad de extraer las características de manera automática sin intervención humana, esto se logra a través de múltiples capas de representación y abstracción de datos.

Por otro lado, las técnicas de búsqueda metaheurísticas son ampliamente utilizadas para resolver problemas de optimización. En este trabajo se describen las técnicas de búsqueda evolutivas, las cuales son inspiradas en la evolución biológica. Un ejemplo de estas técnicas son los algoritmos genéticos.

Se presentan también las técnicas de búsqueda metaheurísticas que se utilizarán en este trabajo, como lo son los algoritmos genéticos (AG). Los AG son una técnica de búsqueda de cómputo evolutivo. Los AG basan su funcionamiento en el principio de la supervivencia del más apto de la teoría de la evolución de Charles Darwin. Asimismo, se tratarán las distintas propuestas para optimizar RNA en el estado del arte, así como también las distintas técnicas propuestas para la detección de grietas en el concreto de estructuras civiles.

2.1 Redes neuronales artificiales

Las redes neuronales artificiales (o simplemente llamadas redes neuronales) son técnicas de aprendizaje de máquina, una subárea de la inteligencia artificial que consiste en modelar matemáticamente la relación entre un conjunto de variables predictoras, con una variable objetivo (Khan, 2018). Cuando la variable objetivo toma valores categóricos o discretos, estamos hablando de un caso de clasificación. Por otro lado, si la variable toma valores continuos, estamos hablando de un caso de regresión. Las RNA son abstracciones matemáticas que modelan el

funcionamiento de las neuronas biológicas con las cuales los seres vivos procesan y aprenden la información de su entorno (Abdi y col., 1999). Visto desde el punto de vista computacional, las RNA son modelos computacionales compuestos por múltiples nodos (neuronas) interconectados entre sí, cada conexión entre dos neuronas tiene asociado un peso, el cual se multiplica por el valor de entrada de la neurona asociada a esa conexión. También, se realiza la aplicación de funciones de activación a las salidas de las neuronas, permitiendo así múltiples transformaciones de naturaleza no lineal dentro del modelo, añadiendo flexibilidad al modelo resultante de una RNA.

2.1.1 Neurona

Las neuronas son las unidades básicas que conforman una RNA. Cada neurona representa a una función matemática de clasificación (en un problema de clasificación), siendo los parámetros de esta los valores de entrada, los pesos de las conexiones, y el sesgo. Dichos parámetros de la neurona son todos valores reales. La representación matemática de una neurona se describe en la Ecuación 2.1. Cabe mencionar que esta operación no se realiza en las neuronas de entrada de la red, solo en las subsecuentes.

$$y = f(\sum_{i=1}^{n} x_i w_i + b) \tag{2.1}$$

Cada neurona puede tener múltiples valores de entrada, cada valor de entrada x_i lleva asociado un peso w_i, y la entrada de la neurona, es la suma ponderada de sus valores de entrada con sus pesos. A este valor se le suma el valor del sesgo b, el cual es un término que brinda la flexibilidad a la función de activación de cada neurona, permitiéndole ajustarse y modelar relaciones complejas entre las características de entrada y las salidas deseadas. Esto posibilita que la red aprenda funciones más sofisticadas y no lineales, ya que no está limitada a pasar únicamente por el origen, ampliando así su capacidad de representación y aprendizaje. Después, el valor resultante se utiliza como entrada para una función de activación f, obteniendo así el valor de salida de la neurona, y.

Se puede encontrar una representación gráfica de la neurona y sus componentes en la Figura 2.1.

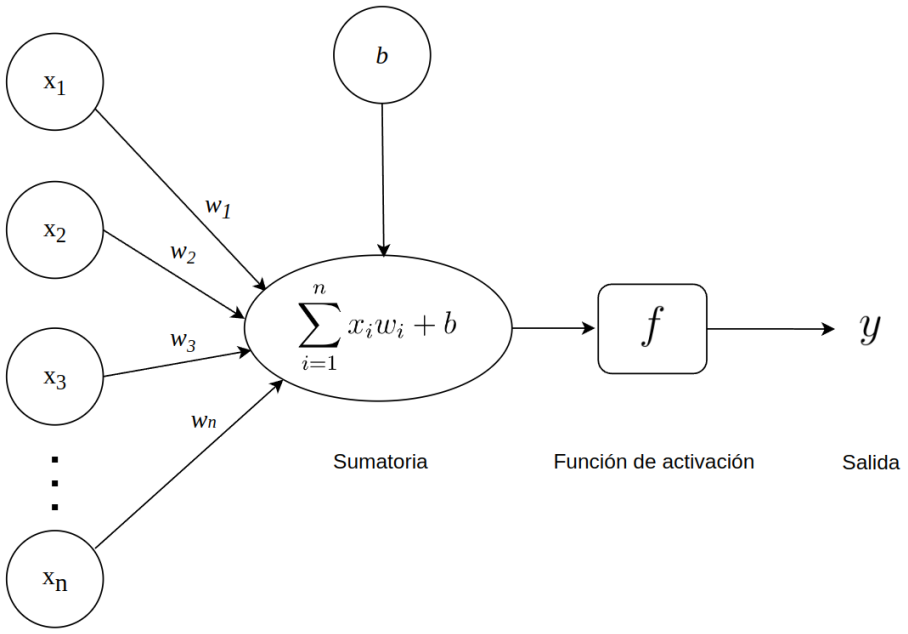


Figura 2.1: Representación gráfica de una neurona.

Un conjunto de estas unidades interconectadas entre sí, forman una red neuronal artificial, las cuales pueden ser dispuestas en distintas formas para lograr distintos propósitos.

2.1.2 Estructura

La estructura de RNA más utilizada para problemas de clasificación y regresión son en su mayoría redes multicapa totalmente conectadas (también llamadas redes de capas densas). Las capas totalmente conectadas, son capas consecutivas de neuronas, donde cada neurona j de la capa k, está conectada con todas las neuronas de la capa $k+1$, transmitiendo así los valores de salida de cada neurona a otras neuronas.

Las capas de la red se subdividen en tres grupos:

- Capa de entrada: es la capa que recibe los valores de entrada de la red neuronal.
- Capas ocultas: son las capas que se encuentran en medio de la capa de entrada y la capa de salida.
- Capa de salida: es la capa que se encuentra al final de la red neuronal, y su salida, es la salida de la red neuronal.

Se puede apreciar en la Figura 2.2 una posible disposición de una red neuronal multicapa.

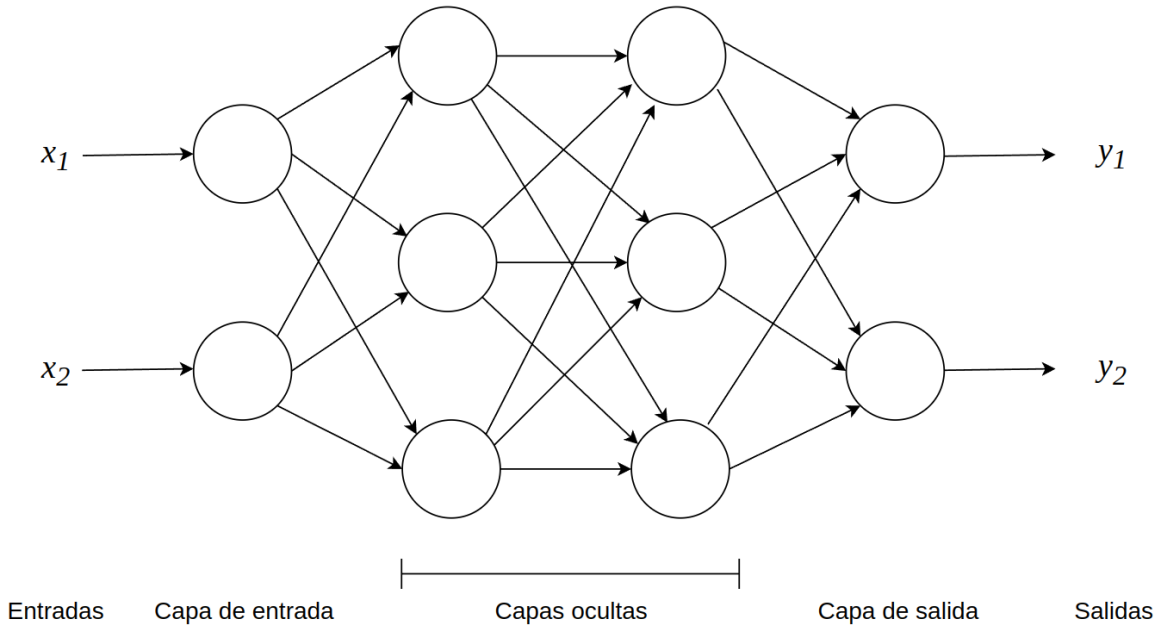


Figura 2.2: Representación gráfica de una red neuronal multicapa.

2.1.3 Entrenamiento

El entrenamiento de una RNA consiste en ajustar (ajustar es un término utilizado para hacer referencia al proceso de modificar u optimizar) los valores de los pesos w y los sesgos b de la red, esto con el objetivo de aumentar la exactitud del modelo y disminuir el error (minimizando una función de pérdida). El ajuste de estos parámetros se realiza con base en los valores de salida producidos al hacer pasar datos a través de la red neuronal. El primer paso es ingresar datos por la capa de entrada. Después, cada neurona transmitirá su valor de salida a las neuronas con las que exista una conexión en la siguiente capa, y así sucesivamente hasta llegar a la capa de salida. La salida obtenida se compara con la salida esperada, y se hace el cálculo del error utilizando una función de error (también llamada función de pérdida). Una vez obteniendo el valor de error, se ejecuta un proceso llamado retropropagación, el cual propaga el error desde la capa de salida hasta la capa de entrada de la red. El proceso de retropropagación altera los valores de los pesos de la red de acuerdo a la contribución de cada uno al error global. La

contribución de cada peso al error se puede obtener calculando el gradiente de la función de error evaluada con los valores de entrada, de esta forma se busca minimizar el error cambiando el valor de los pesos para mover el valor de error en dirección negativa del gradiente. A este proceso se le conoce también como optimización.
Durante la retropropagación, se pueden utilizar distintos algoritmos optimizadores para el ajuste de los valores de los pesos y sesgos, algunos de los optimizadores más utilizados son los siguientes:

- *SGD*: el Descenso Estocástico de Gradiente (*Stochastic Gradient Descent* por su término en inglés) es un algoritmo que calcula el gradiente de la función de error con respecto a los pesos de la red cuando se hace pasar un lote de datos (conjunto de instancias) aleatorio a través de la red. El hecho de tener el valor del gradiente permite ajustar los pesos en la dirección negativa de este, ya que, este valor nos indica la dirección en la que la función crece más rápidamente. Si se desea minimizar la función, podemos movernos en dirección opuesta al gradiente. Por lo tanto, el gradiente es fundamental para reducir así el valor de la función de error.

- *AdaGrad*: el Gradiente Adaptativo (*Adaptive Gradient* por su término en inglés) es un optimizador que funciona bajo el mismo principio que SGD a la hora de actualizar los valores de los pesos de la red en cuanto al movimiento en dirección del gradiente negativo. Sin embargo, actualiza los parámetros con el uso de una tasa de aprendizaje adaptativa para cada uno de los pesos (i.e., actualiza cada peso a una tasa propia, a diferencia de SGD que los actualiza a la misma tasa), esto es posible ya que se guarda un historial de gradientes para cada peso. Esto puede resultar útil cuando las contribuciones de los pesos al error son muy dispersas.

- *ADAM*: Estimación de momento adaptativo (*Adaptive Moment Estimation* por su término en inglés), es un algoritmo que agrega ciertas características a sus predecesores. El historial de gradientes para cada peso se limita a guardar solo una fracción de su histórico, utiliza los últimos dos momentos del gradiente para calcular el próximo paso, y mantiene estabilidad en las actualizaciones de los pesos.

Existe un gran número de algoritmos optimizadores, sin embargo para la optimización de los parámetros de una red neuronal generalmente se opta por métodos basados en cálculo de

gradientes como los ya expuestos, debido a que se facilita el cómputo de los valores de los pesos con el algoritmo de retropropagación. El algoritmo optimizador de pesos de una red neuronal también es parte de los hiperparámetros de una RNA, en la Subsección 2.1.4 se habla de los hiperparámetros con más detalle.

2.1.4 Hiperparámetros

Los hiperparámetros de una RNA son aquellas características de la red que no son entrenables, es decir, los valores de los hiperparámetros deben ser seleccionados en la etapa de diseño del modelo, antes del entrenamiento. Los hiperparámetros en una red neuronal tienen un gran impacto en el desempeño del modelo generado, ya que ciertos conjuntos de valores posibles pudieran generar un modelo con un desempeño pobre (si hacemos referencia a la métrica de tasa de aciertos, significa que el modelo no está logrando clasificar correctamente una cantidad significativa de instancias) para algún caso de estudio en específico, es por ello que es relevante ajustar los valores de los hiperparámetros de manera que se pueda mejorar el desempeño del modelo lo más posible, es decir, es relevante la optimización de hiperparámetros en las redes neuronales artificiales.

Es común detectar en el estado del arte que no se haga uso de una metodología estructurada para la selección de hiperparámetros. Por ejemplo, en ocasiones a los hiperparámetros se les asignan valores por defecto o convención. Además, los ajustes de los hiperparámetros se suelen realizar de manera manual en la etapa de experimentación (Aszemi y Dominic, 2019), dependiendo del desempeño del modelo generado se hacen los ajustes correspondientes para mitigar el error (i.e., a prueba y error). Ajustar manualmente los hiperparámetros puede no ser factible debido a que el rango de valores que se debe explorar puede ser demasiado extenso, superando los recursos disponibles en términos de capacidad de cómputo y tiempo.

Otra forma en la que se eligen los valores de los hiperparámetros, es con base en experiencias pasadas de los desarrolladores en distintos trabajos de investigación. Los ya mencionados criterios de selección de valores para hiperparámetros de una RNA, conllevan incertidumbre y subjetividad, ya que al no realizar una exploración del espacio de búsqueda para los posibles valores de hiperparámetros pueden existir configuraciones inexploradas que podrían generar modelos con un desempeño favorable para el caso de estudio que se esté tratando en determina-

do momento. Esto genera una necesidad en la literatura por utilizar metodologías estructuradas de selección de valores para los hiperparámetros de las redes neuronales, y así generar modelos con mejores capacidades para los distintos trabajos en la ciencia.

En la literatura se encuentran otras estrategias para seleccionar los valores de los hiperparámetros, algunas de las más utilizadas se mencionan a continuación:

- Búsqueda en rejilla: *Grid Search* por su término en inglés. Consiste en tomar un rango de valores finitos para ciertos hiperparámetros en forma de rejilla, y evaluar distintas combinaciones de estos, que representarían las intersecciones de estos distintos valores.

  La búsqueda en rejilla puede ser útil para optimizar los hiperparámetros bajo ciertos escenarios. Sin embargo, no siempre es la mejor opción, ya que cuando la cantidad de hiperparámetros a optimizar es grande, la cantidad de evaluaciones necesarias crece exponencialmente (Zöller y Huber, 2021). Además, frecuentemente es necesario acotar manualmente el espacio de búsqueda para encontrar valores de hiperparámetros óptimos con búsqueda en rejilla.

- Búsqueda aleatoria: *Random Search* por su término en inglés. El método de búsqueda aleatoria consiste en elegir configuraciones candidatas de valores de hiperparámetros del espacio de búsqueda de manera aleatoria.

  La búsqueda aleatoria se introdujo para superar algunas desventajas de la búsqueda en rejilla. El método ofrece una simplicidad teórica y de implementación similar a la búsqueda en rejilla, sin embargo, de acuerdo a (Bergstra y Bengio, 2012) la búsqueda aleatoria suele ser más efectiva y utiliza menos recursos que la búsqueda en rejilla, ya que la manera en la que se explora el espacio de búsqueda no se limita a la exploración exhaustiva de los valores posibles para los hiperparámetros. Búsqueda en rejilla y búsqueda aleatoria, padecen de la misma desventaja: sus evaluaciones de funciones son independientes de pruebas pasadas, por lo tanto pueden sobre-explorar un área del espacio de búsqueda donde el modelo tiene un comportamiento deficiente, y pueden sub-explorar un área del espacio de búsqueda donde el modelo tiene un comportamiento positivo (Yang y Shami, 2020). Este problema puede suponer un desperdicio de recursos tanto en tiempo, en cómputo, y existe la posibilidad de que no se logre generar un modelo con el desempeño deseado.

- Optimización Bayesiana: este método puede solventar la antes discutida debilidad de la búsqueda en rejilla y búsqueda aleatoria. La optimización Bayesiana determina el siguiente punto de evaluación de valores de hiperparámetros basado en evaluaciones anteriores (Snoek y col., 2012).

  La decisión de evaluación de solución es posible ya que el algoritmo construye un modelo probabilístico para la función objetivo del problema, nombrado "modelo surrogado", el cual supone una distribución de probabilidad para los valores de los hiperparámetros de los cuales depende la función objetivo. Actualizando el modelo surrogado, el algoritmo es capaz de explorar y explotar el espacio de búsqueda a necesidad de acuerdo a las evaluaciones pasadas de la función objetivo, haciendo así más probable el encontrar un óptimo global para la función. Aunque este método puede dirigir mejor la búsqueda en comparación con búsqueda de rejilla o búsqueda aleatoria, se debe designar cuidadosamente el balance entre exploración y explotación del espacio de búsqueda, de lo contrario la búsqueda puede estancarse en un mínimo local (Yang y Shami, 2020).

- Optimización basada en población: *Population-based Optimization* por su término en inglés. Los algoritmos de optimización basados en población consisten en generar una población de individuos y actualizarlos en cada generación (iteración) de acuerdo a los criterios del algoritmo.

  Los individuos generados por el algoritmo son evaluados en cada generación hasta encontrar un individuo óptimo o bien cumplir una condición de terminación, como ejecutar una cantidad determinada de iteraciones. Dentro de la rama de los métodos basados en población, se encuentran los algoritmos genéticos. Como una ventaja importante, los algoritmos genéticos ofrecen la capacidad de encontrar soluciones en espacios de búsqueda grandes en relativamente poco tiempo, se ha demostrado que este tipo de algoritmos tiene buenas capacidades de optimización para redes neuronales de varios tipos (Jaderberg y col., 2017). Además, tienen la flexibilidad necesaria para poder optimizar hiperparámetros numéricos continuos, y categóricos, lo cual resulta de especial ayuda al optimizar los hiperparámetros de una red neuronal ya que existen hiperparámetros de distintos tipos, como reales, enteros, continuos, categóricos, etc. (Yang y Shami, 2020)

La optimización de hiperparámetros no tiene un método definitivo, ya que existen ventajas y desventajas en cada uno de ellos. Sin embargo, para la optimización de hiperparámetros en RNA se considera beneficioso el uso de técnicas basadas en población así como lo son los algoritmos genéticos, esto debido a su flexibilidad y capacidad de encontrar soluciones de forma rápida en espacios de búsqueda grandes con varias dimensiones (Yang y Shami, 2020). Las ventajas mencionadas propician la generación de modelos con un buen desempeño, a veces incluso superior a aquellos modelos cuyos hiperparámetros son ajustados por otras metodologías (Itano y col., 2018), (Rattanavorragant y Jewajinda, 2019), (Alibrahim y Ludwig, 2021).

Las redes neuronales tienen distintos hiperparámetros que se pueden optimizar, algunos de ellos se presentan a continuación:

Función de activación

Las redes neuronales se podrían definir como una combinación de transformaciones lineales, cada una de estas producida por una función de regresión lineal generada en cada neurona. Muchos problemas computacionales no pueden resolverse con modelos lineales, esta es una de las razones por las que se introducen funciones de activación, muchas de ellas no lineales para así poder realizar transformaciones no lineales utilizando la salida de la suma ponderada de las neuronas, finalmente así dando flexibilidad al modelo matemático generado en la red neuronal, permitiendo generar modelos más complejos. Algunas de las funciones de activación más utilizadas en el estado del arte son las siguientes:

- Función escalón: permite establecer un umbral de activación discreto para la neurona, permitiendo únicamente dos valores posibles de salida, 0 o 1. La función escalón se define como se muestra en la Ecuación 2.2.

$$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases} \tag{2.2}$$

- Función Sigmoide: esta función es frecuentemente utilizada para mantener los valores de salida de las neuronas entre 0 y 1. Una de las ventajas de la función Sigmoide es que es continua, diferenciable, y su derivada se puede expresar en términos de sí misma, lo

cual es de especial ayuda al computar el gradiente. La función Sigmoide se define como se muestra en la Ecuación 2.3.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

- Función $ReLU$: la función de unidad lineal rectificada ($ReLU$ por sus siglas en inglés) toma el valor de salida de la transformación lineal siempre y cuando sea mayor a 0, en el caso contrario su valor será 0. Dado su funcionamiento, las neuronas que tengan valores negativos de salida en su suma ponderada, permanecerán desactivadas. La función $ReLU$ se define como se muestra en la Ecuación 2.4.

$$f(x) = max(0, x) \tag{2.4}$$

El efecto de la función de activación en la red neuronal es importante para poder generar modelos con buen desempeño, ya que las prestaciones de cada una pueden ser de ayuda para determinadas aplicaciones específicas. Por ejemplo, algunos autores señalan que la función $ReLU$ es más eficiente que otras en ciertos escenarios, debido a que no todas las neuronas se activan a la vez, lo cual permite una mayor robustez a los cambios pequeños en los valores de entrada (Apicella y col., 2021).

Tasa de aprendizaje

La tasa de aprendizaje (*Learning Rate* por su término en inglés) se puede definir como la magnitud de actualización de los pesos en dirección del gradiente de la función objetivo durante determinada iteración de entrenamiento. La elección de un valor de tasa de aprendizaje para el entrenamiento del modelo es muy importante para tratar de alcanzar el valor mínimo de la función de pérdida (error). Al ajustar los pesos de la red en dirección del gradiente, una tasa de aprendizaje alta implicaría un ajuste de gran magnitud para los pesos, lo cual pudiera hacer que el valor de error no pueda descender a un mínimo de la función debido a que el paso del ajuste es demasiado amplio. Por otro lado, si la tasa de aprendizaje es muy baja, la longitud del avance del valor del error en dirección del gradiente va a ser baja, por consiguiente el algoritmo podría

tomar demasiado tiempo en llegar a un mínimo, o bien no poder alcanzarlo nunca (Bengio, 2012).

En la literatura existen múltiples políticas de actualización para la tasa de aprendizaje, como el tener un valor estático de tasa de aprendizaje, ajustarlo de forma manual en la experimentación dependiendo del resultado del modelo, o bien algún algoritmo que cambie el valor de la tasa de aprendizaje de manera decreciente o siguiendo alguna función designada (Wu y col., 2019).

Tamaño de lotes

El tamaño de lotes (*Batch Size* por su término en inglés) indica la cantidad de instancias que se procesarán a través de la red neuronal durante el entrenamiento en cada iteración, con el fin de ajustar los parámetros en la dirección del gradiente de la función objetivo. Por ejemplo, si se tiene un conjunto de datos de $1,000$ instancias, se podría dividir en lotes de 100 datos, de manera que el conjunto de datos estaría dividido en 10 lotes de 100 instancias. Siguiendo el ejemplo del conjunto de datos mencionado, al entrenar el modelo los datos ingresarían a la red neuronal lote por lote, después de cada lote de datos se calcula el error del modelo, y se ajustan los pesos de la red neuronal para mitigar el error (Hoffer y col., 2017).

La elección de un valor de tamaño de lotes tiene un impacto directo en la capacidad de generalizar del modelo, es por ello que es importante elegir un valor adecuado para el problema a resolver. Uno de los efectos más relevantes del tamaño de lotes que se ha detectado sobre el entrenamiento del modelo, es la disminución de la tasa de aciertos del modelo si el tamaño de lotes es muy grande, y una mejor capacidad de generalización si el tamaño de lotes es menor (Sekhari y col., 2021). Sin embargo, tener un tamaño de lote grande acelera la velocidad del entrenamiento, mientras que en el caso contrario, cuando el tamaño de lote es pequeño, la fase de entrenamiento puede prolongarse considerablemente.

Tasa de difusión

La tasa de difusión (*Dropout* por su término en inglés) es una medida de regularización para redes neuronales. La tasa de difusión ayuda a prevenir el sobreajuste en la fase de entrenamiento.

El sobreajuste en los modelos predictivos se presenta cuando el modelo tiene un buen desempeño solamente con el conjunto de datos de entrenamiento, pero muestra un mal desempeño con nuevos datos desconocidos. El sobreajuste puede suceder debido a una sobreparametrización del modelo, es decir, tener una cantidad muy alta de parámetros con respecto a la cantidad de observaciones en el conjunto de datos de entrenamiento.

La tasa de difusión consiste en desactivar con una probabilidad p las neuronas de la red durante el entrenamiento en cada iteración. De este modo, existirá una menor cantidad de pesos en la red neuronal que contribuyan al error de clasificación, y por consiguiente una disminución de parámetros que se pueda reflejar en una mejor capacidad de generalización del modelo, previniendo así el sobreajuste (Srivastava y col., 2014).

2.2 Redes neuronales convolucionales

Las redes neuronales convolucionales (RNC) son técnicas de aprendizaje profundo derivadas de las redes neuronales artificiales. Las RNC tienen la capacidad de extraer las características de un conjunto de datos de manera automática sin intervención humana, esto se logra mediante la aplicación de operaciones de convolución. Estas operaciones convolucionales consisten en deslizar filtros digitales pequeños (kernels) sobre la imagen de entrada y realizar multiplicaciones de elementos para resaltar patrones específicos, utiliza también una función de activación para agregar no-linealidad al resultado de la convolución. La convolución es lo que caracteriza a este tipo de redes neuronales, además, los datos pasan a través de múltiples capas de representación y abstracción de datos, que permiten la transformación de los datos de entrada a vectores numéricos de características.

Las primeras capas de una RNC extraen características locales o de bajo nivel, para ser subsecuentemente combinadas en las siguientes capas, consiguiendo así representaciones más complejas de alto nivel (Salman y col., 2013).

Uno de los primeros trabajos de investigación sobre las RNC fue propuesto por (LeCun y col., 1989). Los autores desarrollaron una arquitectura de RNC para reconocimiento de códigos postales escritos a mano, dando origen al concepto de "convolución" en el área de estudio de las redes neuronales. Años más tarde, en el estudio de LeCun y col., 1998 se utilizó una nueva

versión de RNC para el reconocimiento de caracteres numéricos escritos a mano, y se comparó con otros esquemas de clasificación. La nueva arquitectura de RNC propuesta se nombró *LeNet-5*, y se volvió la arquitectura de RNC más famosa en su época.

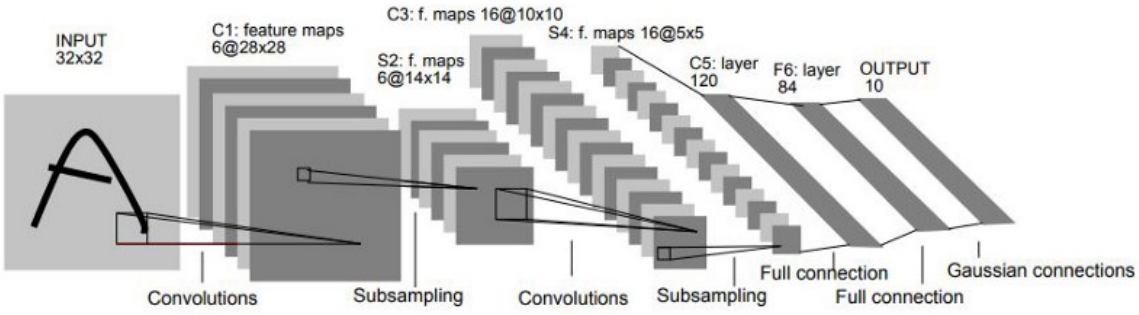


Figura 2.3: Arquitectura *LeNet-5* (LeCun y col., 1998).

Gracias a la capacidad de las RNC para extraer de manera efectiva las características de conjuntos de imagénes digitales, se volvieron ampliamente utilizadas en la comunidad científica para tratar problemas de visión por computadora, detección y clasificación de objetos, entre otras. Sin embargo, las arquitecturas de RNC tienden a estar compuestas de un gran número de capas, y el aumentar la cantidad de capas en una red neuronal puede provocar distintos problemas, como estancar el valor objetivo en un óptimo local, sobreajuste, explosión y desvanecimiento de gradiente, etc (Li y col., 2021). Además, la profundidad de este tipo de redes neuronales puede llevar consigo un gran costo computacional, debido a la gran cantidad de operaciones que debe realizar para procesar la información.

Para solventar algunos de estos problemas, las RNC implementan distintas técnicas para reducir la carga de procesamiento del algoritmo, como conexiones locales entre neuronas, y capas de agrupamiento para reducir la dimensionalidad de los datos sin perder información relevante.

Las RNC están conformadas por los siguiente componente principales:

1. Capas convolucionales: estas capas permiten la extracción de características y patrones mediante operaciones de convolucion con filtros (kernels). El resultado de la operación de convolución se hace pasar por una función de activación, agregando así flexibilidad

al modelo permitiendo ajustarse mejor a los datos de entrada, una de las funciones más utilizadas en las capas convolucionales es la función de activación $ReLU$.

2. Capas de agrupación: estas capas permiten reducir la dimensionalidad de las características, preservando las características más importantes y disminuyendo la cantidad de parámetros en la red. El método de agrupación más común es el de agrupamiento por valor máximo, donde se toma el valor máximo de una ventana específica de la entrada.

3. Capas totalmente conectadas: estas capas están compuestas de varias neuronas, cada una conectada con todas las neuronas de la capa siguiente, y sirven para realizar la tarea de clasificación o regresión. Las capas completamente conectadas son las que conforman las RNA tradicionales.

En una RNC, un bloque convolucional es una unidad modular que combina una o más capas convolucionales con capas de agrupamiento (*pooling*, por su término en inglés). Este bloque es fundamental para la extracción de características en la RNC. Por otro lado, las capas totalmente conectadas actúan como el clasificador final.

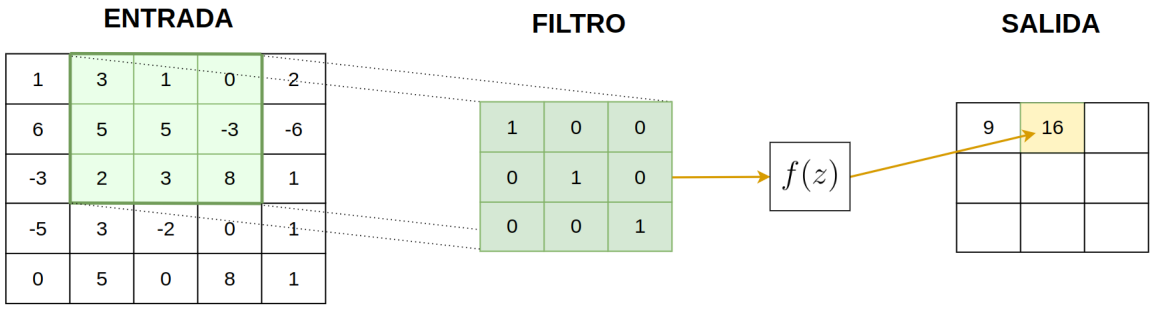
A lo largo de los años se han propuesto distintas arquitecturas de redes neuronales convolucionales por varios grupos de investigación en el mundo, las arquitecturas varían muchos de sus aspectos estructurales, como el tipo y cantidad de capas a utilizar, tamaño y cantidad de filtros en capas convolucionales, entre otras. Esta variedad de arquitecturas ha proporcionado flexibilidad a la comunidad científica para poder tratar distintos problemas y adaptar la estructura de una RNC a su conveniencia.

2.2.1 Capa convolucional

Las capas convolucionales realizan las operaciones principales dentro de una RNC para extraer las características de los datos de entrada. Las capas convolucionales están compuestas por diversos filtros (kernels), los cuales convolucionan con la matriz de datos de entrada (hablando de una convolución en 2D) para así obtener características representativas que el filtro es capaz de extraer. Los filtros son matrices de números discretos, estos números son aprendidos por la RNC durante el entrenamiento realizando la operación de convolución. La convolución consiste

en deslizar los filtros a través de la matriz de entrada de manera horizontal y de manera vertical. Mientras el filtro se mueve a través de la matriz de entrada, se realiza la multiplicación de todos los valores superpuestos del filtro con los datos de entrada, y posteriormente se suman los valores para obtener un solo valor escalar, este valor escalar es la entrada para una función de activación determinada, y el resultado de esta función formará parte de la salida.

En la Figura 2.4 se ilustra la operación de convolución con paso de uno, es decir, el filtro se desliza celda por celda para realizar la operación de convolución en cada uno de los pasos. Dependiendo la estrategia se pueden dar pasos de más de una celda para realizar la operación.



Siendo $f(z) = max(0, z)$

$$z[1,2] = (3*1) + (1*0) + (0*0) + (5*0) + (5*1) + (-3*0) + (1*0) + (3*0) + (8*1) = 16$$

$$f(z[1,2]) = 16$$

Figura 2.4: Ejemplo de operación de convolución.

Realizando la operación de convolución en cada una de las posiciones posibles de la entrada, resulta en un mapa de activación que abstrae las características de la entrada extraídas por el filtro.

Un aspecto importante a resaltar, es que una capa convolucional puede estar compuesta por n filtros, todos ellos convolucionarán con la entrada y producirán una salida cuya cantidad de canales será n.

2.2.2 Capa de agrupamiento

Debido a que las RNC suelen estar compuestas de un gran número de capas, el costo computacional crece considerablemente para realizar la cantidad de operaciones necesarias para procesar los datos de entrada. Las RNC cuentan con un tipo de capas para reducir la dimensión de los datos procesados, estas capas se denominan de agrupamiento (*pooling* por su término en inglés). Las capas de agrupamiento, de manera similar a las capas convolucionales, implementan una ventana deslizante que toma valores de la entrada y realiza con ellos una operación para reducir la cantidad de datos manteniendo la información relevante de las capas anteriores. La ventana deslizante suele tener pasos de desplazamiento mayores a uno, es decir, la ventana deslizante se traslada dos celdas o más en cada movimiento, de esta manera se puede garantizar una reducción en tamaño de la matriz de entrada.

Un ejemplo del uso de la capa de agrupamiento, es el agrupamiento por valor máximo (*max pooling* por su término en inglés), este tipo de agrupamiento consiste en tomar el valor máximo de la zona de la matriz de entrada en la cual se encuentre superpuesta la ventana deslizante. Un ejemplo del agrupamiento por valor máximo se puede apreciar en la Figura 2.5.

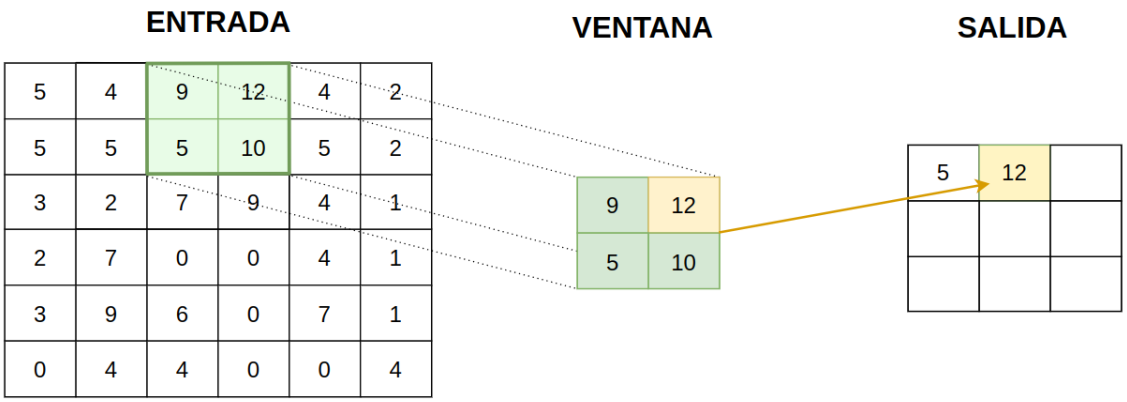


Figura 2.5: Ejemplo de agrupamiento por valor máximo.

2.2.3 Capas completamente conectadas

Las capas completamente conectadas (también llamadas capas densas) son las capas de neuronas que componen a las RNA convencionales mencionadas en la Subsección 2.1.2. Las RNC suelen

tener capas densas al final de su estructura para ejecutar la tarea de clasificación.

Durante el proceso de extracción de características realizado en las capas convolucionales y de agrupación, los datos de entrada de la RNC se transforman en coeficientes que abstraen las características representativas de las instancias de entrada, de manera que dichos coeficientes usualmente terminan contenidos en arreglos n-dimensionales al final del extractor de características. Una vez obtenidos los arreglos n-dimensionales de características, pasan por una fase de aplanado que consiste en convertir el arreglo n-dimensional en un vector unidimensional para utilizar las características extraídas como entrada de las capas completamente conectadas.

Conceptualmente, las RNC cuentan con un extractor de características compuesto de una serie de bloques convolucionales, cada uno compuesto de capas convolucionales y de agrupamiento, y después del extractor de características se encuentra un bloque clasificador compuesto por capas completamente conectadas.

2.2.4 *VGG-16*

Las arquitecturas de RNCs se refieren a las diversas propuestas de diseño especifico de RNCs que se han desarrollado para resolver diferentes problemas de clasificación de objetos en imágenes digitales. Estas arquitecturas están compuestas por diferentes disposiciones de las capas de convolución, capas de agrupación, y capas densas (completamente conectas), entre otras posibles capas especializadas.

Una de las arquitecturas de RNC más utilizadas en la última década es *VGG-16*, fue propuesta en el trabajo de (Simonyan y Zisserman, 2014), siendo esta una propuesta para participar en el reto *ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC-2014)*, dicho reto consiste en tratar un problema de clasificación multiclase (1000 clases) del conjunto de datos *ImageNet*, el cual contiene más de $1,000,000$ de imágenes.

Las principales aportaciones de *VGG-16* que los autores señalan, son la experimentación con distintas profundidades de la RNC, llegando hasta 19 capas, además de utilizar filtros más pequeños (3×3) de lo que se acostumbraba en el estado del arte, también, los autores señalan obtener buenos resultados en contraste con los modelos más recientes de la literatura en ese año.

La arquitectura *VGG-16* funciona con imágenes de tamaño $224 \times 224 \times 3$ como entrada, es

decir, 224 pixeles de alto, 224 pixeles de ancho, y tres canales de color de profundidad, siendo estos rojo, verde, y azul (RGB). La estructura de *VGG-16* se puede dividir en dos partes principales: el extractor de características, y el bloque clasificador. El extractor de características es el encargado de extraer las características de las imágenes de entrada a través de las operaciones de convolución y agrupamiento. En contraste, dentro del bloque clasificador se ubican las capas densas (totalmente conectadas), las cuales se emplean para llevar a cabo la clasificación.

El extractor de características consiste en una secuencia de bloques convolucionales, cada uno de estos bloques compuestos por capas convolucionales, y capas de agrupamiento por valor máximo. Se tienen cinco conjuntos de capas, los primeros dos conjuntos se componen de dos capas convolucionales seguidas de una capa de agrupamiento por valor máximo, los últimos tres conjuntos están formados por tres capas convolucionales, agregando al final de cada bloque una capa de agrupamiento por valor máximo.

En el primer bloque convolucional del extractor de características cuentan con 64 filtros, las del segundo bloque tienen 128 filtros, las capas del tercer bloque tienen 256 filtros, por último, las capas convolucionales de los últimos dos bloques de capas convolucionales cuentan con 512 filtros cada una. Conforme se procesan los datos a través del extractor de características, la matriz de entrada reduce sus dimensiones de alto y ancho debido a las capas de agrupamiento, y aumenta sus dimensiones en profundidad por la cantidad de filtros de las capas convolucionales. Suponiendo el caso de una sola imagen de entrada de tamaño $224 \times 224 \times 3$, al ser procesada por el extractor de características, al final resulta en un tensor de tamaño $7 \times 7 \times 512$, esto implica una reducción considerable en cantidad de datos y conlleva a un menor costo computacional al momento de realizar la tarea de clasificación.

El bloque clasificador de *VGG-16* se compone de dos capas ocultas completamente conectadas y una capa de salida. Como entrada al bloque de clasificación se tiene el tensor de salida del extractor de características de tamaño $7 \times 7 \times 512$, sin embargo, el tensor debe pasar por una fase de aplanado para poder ser usado como entrada para la red neuronal del bloque de clasificación, obteniendo así un arreglo unidimensional de $25,088$ valores. Por último, la salida del bloque clasificador es un vector de probabilidades que indica la confianza de la red en cada clase de salida.

En la Figura 2.6 se presenta el esquema de la arquitectura *VGG-16*.

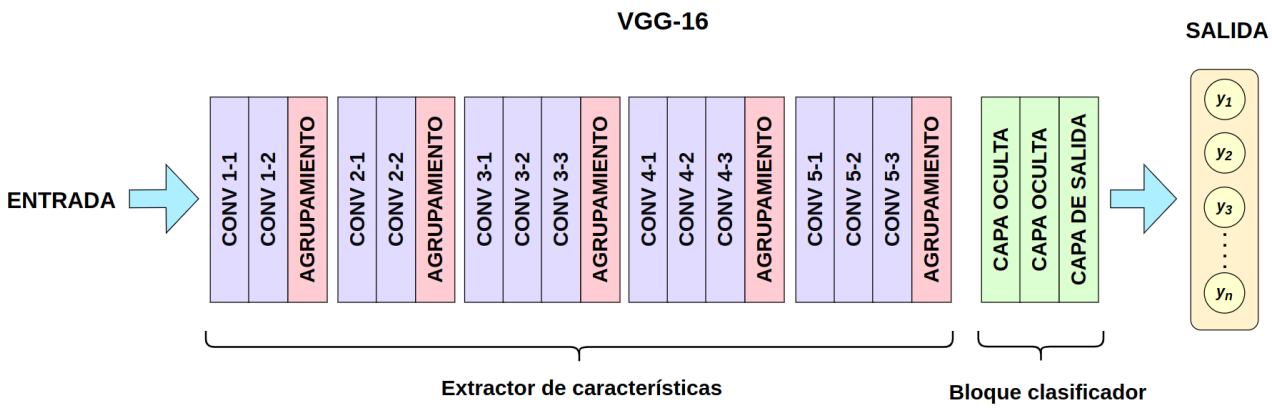


Figura 2.6: Arquitectura VGG-16 (Simonyan y Zisserman, 2014).

2.2.5 Aprendizaje por transferencia

El aprendizaje por transferencia es una técnica de aprendizaje de máquina que consiste en utilizar el conocimiento que adquirió un modelo al ser entrenado para alguna tarea específica, para tratar otra tarea similar (M. Hussain y col., 2019). Utilizando la técnica de aprendizaje por transferencia, es posible entrenar un modelo basado en RNC con un conjunto de datos X, y utilizar otro conjunto de datos Y para realizar extracción de características, de esta forma, se hace uso de los patrones aprendidos con el conjunto X, permitiendo que estos puedan ser extrapolados y se puedan extraer características con el conjunto Y, esto sin haber entrenado al modelo explícitamente con el conjunto Y. Un modelo que ya fue entrenado con un conjunto de datos X y posteriormente es utilizado para extracción de características de un conjunto de datos distinto, se le puede nombrar modelo pre-entrenado.

Una de las ventajas más prominentes del aprendizaje por transferencia, es el hecho de que no es requerido entrenar de nuevo la RNC en cuestión, que esto generalmente tiende a ser un proceso costoso tanto en tiempo y en recursos de cómputo. Por ejemplo (Simonyan y Zisserman, 2014) reportan que para entrenar una sola RNC de arquitectura *VGG* con el conjunto de datos *ImageNet* puede tomar entre dos y tres semanas. Además, es posible utilizar el conocimiento del modelo pre-entrenado para caracterizar conjuntos de datos limitados, es decir, debido a que entrenar una RNC requiere una gran cantidad de datos, gracias al aprendizaje por transferencia

se pueden caracterizar conjuntos de datos con pocas instancias partiendo de una RNC preentrenada.

En general, hay dos estrategias principales en el aprendizaje por transferencia: extractor de características y ajuste fino (Gao y Mosalam, 2018).

- Extractor de características: partiendo de una RNC preentrenada, se utilizan los bloques convolucionales para extraer características de un conjunto de datos, sin volver a entrenar los bloques. Después, las características extraídas son usadas para entrenar una RNA. La principal ventaja del uso de este modo, es el ahorro de recursos en el entrenamiento y extracción de características, ya que solo se extraen una vez, y solo se entrenarían RNAs con ellas, y entrenar una RNA es menos complejo que una RNC, computacionalmente hablando.

- Ajuste fino: al igual que el modo extractor de características, se toma una RNC preentrenada como punto de partida, sin embargo, a diferencia de este modo, el extractor de características en este caso sí se reentrena. En ocasiones solo se reentrenan las últimas capas convolucionales del modelo, para así poder mantener las características de bajo nivel aprendidas con anterioridad, y en otras situaciones, se reentrena todo el extractor de características. Sea cual sea el enfoque en el entrenamiento, la tasa de aprendizaje durante el entrenamiento debe ser más baja de lo que sería al entrenar la RNC sin utilizar aprendizaje por transferencia, esto para realizar alteraciones pequeñas a los coeficientes del extractor de características y no perder las representaciones y patrones aprendidos durante el primer entrenamiento (Gao y Mosalam, 2018).

2.2.6 Agrupamiento global por promedio

La salida de las capas del extractor de características es un arreglo multidimensional (tensor) de coeficientes que es el resultado de caracterizar las imágenes de entrada, también se les nombra características. Después, las características de salida son vectorizadas (o aplanadas) para ser usadas como entradas para la RNA clasificadora. Una representación gráfica de aplanado de tensor se muestra en la Figura 2.7. La dimensión de profundidad de los tensores se les llama mapas de características.

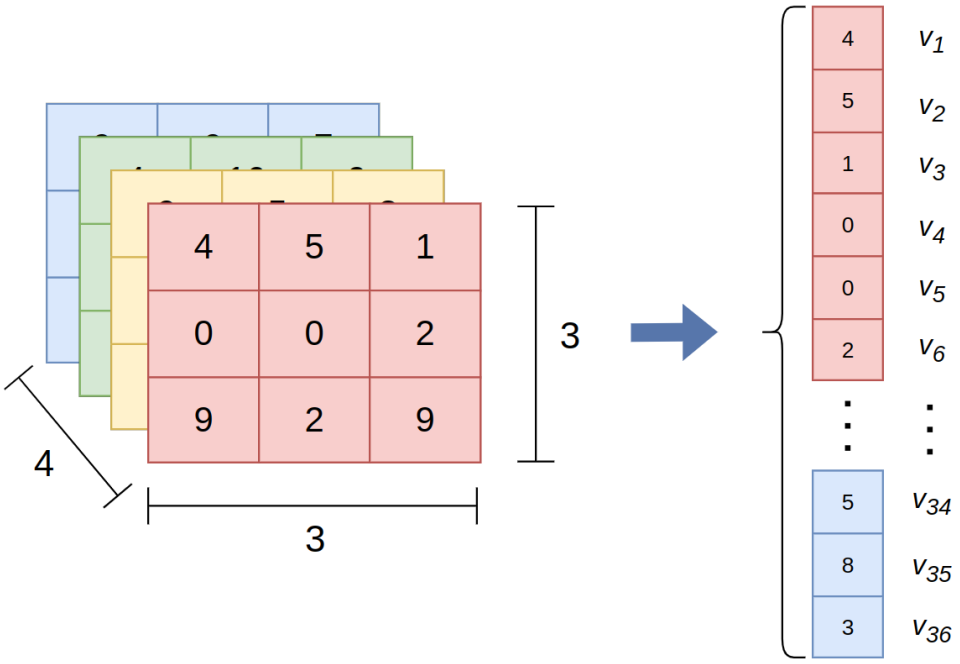


Figura 2.7: Tensor de dimensiones $3 \times 3 \times 4$ (izquierda) siendo aplanado, resultando en un vector de 36 componentes (derecha).

En ocasiones los vectores de características extraídos resultan ser de gran tamaño, esto genera una mayor cantidad de parámetros en la RNA y puede llevar al modelo generado a producir una función matemática que modele al conjunto de entrenamiento y no sea capaz de generalizar el conocimiento correctamente (sobreajuste). Es por esto que se ha optado por crear estrategias que disminuyan la cantidad de parámetros en los modelos para así evitar el sobreajuste, una de ellas es el agrupamiento global por promedio.

El agrupamiento global por promedio (*GAP*, por sus siglas en inglés) es una técnica que permite simplificar el tensor de salida de las capas convolucionales de una RNC para su subsecuente entrada de las capas completamente conectadas de una RNA. El *GAP* consiste en calcular el promedio de cada mapa de características que compone el tensor, así obteniendo solo un valor representativo de cada mapa (Lin y col., 2013). Esto permite una reducción considerable de coeficientes de entrada, permitiendo disminuir la cantidad de parámetros sin reducir el alcance en las características extraídas. En la Figura 2.8 se ilustra el efecto de aplicar *GAP* a un tensor.

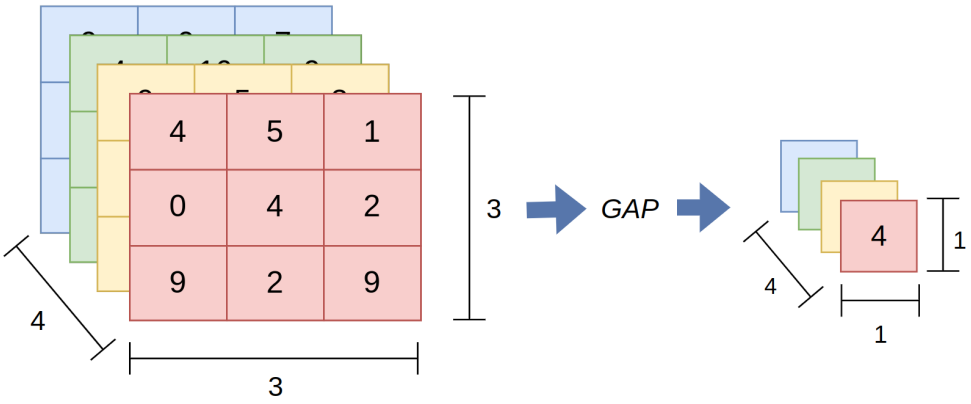


Figura 2.8: Aplicando GAP a tensor de dimensiones $3 \times 3 \times 4$ (izquierda) resultando en un tensor de dimensiones $1 \times 1 \times 4$ (derecha).

2.3 Algoritmos genéticos

Frecuentemente en la ciencia se presentan problemas cuyo tratamiento requiere realizar una búsqueda de distintas combinaciones de valores de variables para poder encontrar "soluciones aceptables" o soluciones potenciales que cumplan ciertos criterios, en ocasiones la cantidad de posibilidades es muy extensa, por lo que se requiere un método de exploración eficiente y versátil. Los algoritmos genéticos son técnicas de búsqueda que pueden suplir las necesidades anteriormente mencionadas. Los algoritmos genéticos son técnicas metaheurísticas inspiradas en la evolución biológica, modelan el principio de la supervivencia del más apto propuesto en la teoría de evolución de Charles Darwin (Michalewicz y Schoenauer, 1996).

El principio básico de funcionamiento de los algoritmos genéticos se basa en operar con una población de n individuos distintos (soluciones) y simula el proceso de reproducción biológica para generar nuevas soluciones. Dentro de los mecanismos del algoritmo, se hace uso de técnicas de selección para decidir cuales son las mejores soluciones y enfocar el desarrollo de la búsqueda utilizando dichas soluciones, esto aumenta las posibilidades de encontrar mejores individuos conforme el algoritmo avanza (Miguel A., 2017).

Los individuos de los algoritmos genéticos (también llamados *cromosomas*) generalmente se representan como una cadena de caracteres codificada en binario, aunque también pueden ser

utilizados números enteros, números reales, o cualquier codificación que sea de utilidad para el problema a resolver. Cada uno de los elementos que componen la cadena de caracteres de un *cromosoma* se le nombra *gen*, y a los distintos valores que puede tomar un *gen*, se les llama *alelos*.

2.3.1 Operadores genéticos

El algoritmo genético clásico se compone de distintos operadores que actúan sobre la población para así generar nuevos individuos. Se utiliza una función para evaluar la aptitud de cada *cromosoma* (*fitness* por su término en inglés), con el operador de selección se seleccionan *cromosomas* de acuerdo a su aptitud para después realizar una cruza, la cual combina los genes de los individuos "padres" seleccionados dando lugar a nuevos individuos "hijos". El operador de mutación modifica de manera aleatoria las soluciones generadas para fomentar la exploración, y por último, se establece la política de reemplazo para elegir a los individuos que pasarán a la siguiente generación. A continuación se describen más a detalle los operadores del algoritmo genético.

Selección

El operador de selección determina la participación de los individuos en el proceso de cruza. Los métodos de selección más utilizados se describen a continuación:

- Selección por ruleta: se selecciona un individuo para la cruza de manera aleatoria con una probabilidad proporcional a su aptitud. Es decir, los *cromosomas* con mayor aptitud, son más propensos a ser elegidos, tratando de mantener *genes* de un elemento apto (Jebari, Madiafi y col., 2013). Una posible desventaja es que si hay individuos dominantes, pueden ganar siempre el sorteo por selección de ruleta, esto podría estancar la exploración del algoritmo.

- Selección por torneo: la selección por torneo consiste en seleccionar aleatoriamente k individuos de la población compuesta de n *cromosomas*, el proceso se realiza n veces. Este mecanismo de selección prueba las comparaciones posibles entre combinaciones de

individuos, pudiendo seleccionar a los *cromosomas* más aptos y así mantener mejores soluciones (Katoch y col., 2021).

- Selección por *ranking*: la selección por *ranking* es una variante del método de selección por ruleta, con el objetivo de evitar una convergencia prematura por la existencia de algún individuo dominante en la población. Se realiza un *ranking* de *cromosomas* de mejor a peor de acuerdo a su medida de aptitud, después se les asignan probabilidades proporcionales a su *ranking*, no a su medida de aptitud; por último, se realiza la selección como dicta el proceso de selección por ruleta (Goldberg y Deb, 1991).

Cruza

La operación de cruza realiza la recombinación de *genes* de dos *cromosomas* padres para generar un nuevo individuo hijo. Algunos de los mecanismos más conocios de cruza se listan a continuación:

- Cruza de un punto: en la cruza de un punto, se elige un punto de corte para los *cromosomas* padre para obtener dos subsecuencias de *genes*, las cuales se intercambian entre padres, generando nuevas soluciones hijo (Katoch y col., 2021).

- Cruza de k-puntos: este método es muy similar a la cruza de un punto, su diferencia reside en que se tienen múltiples puntos de corte en el *cromosoma*, recombinando las subcadenas que surgen de cada padre para generar nuevos hijos.

- Cruza uniforme: la cruza uniforme tiene un enfoque en cada *gen* de los padres, ya que cada *gen* en la posición m del *cromosoma* hijo, tiene un 0.5 de probabilidad de ser heredado tanto del padre 1 como del padre 2. De esta manera el *cromosoma* hijo obtiene el 50 % del material genético de cada uno de sus padres (Soon y col., 2013).

- Cruza por emparejamiento parcial: se generan dos puntos de corte para los individuos padres, aquellos *genes* que se ubiquen entre los dos puntos de corte forman parte automáticamente de dos nuevos *cromosomas* hijos, además, esta subcadena es mapeada de un padre al otro, *gen* por *gen*. La información restante de los padres es transferida a los hijos nuevos. En caso de que al transferir la información restante de los padres se presente

un *alelo* que ya existe en otra posición del *cromosoma* hijo, se utiliza el valor de *gen* que se pueda mapear a través de los valores de la sección extraída en la primera etapa de la cruza (A. Hussain y col., 2017). Este tipo de procedimiento de cruza es descrito como uno de los que brinda mejores resultados (Katoch y col., 2021).

Mutación

El operador de mutación ayuda a mantener la diversidad en la población de soluciones, permitiendo la exploración del espacio de búsqueda haciendo cambios a los *cromosomas* hijos en cada generación. Algunos de los métodos más conocidos de mutación son los siguientes:

- Mutación por desplazamiento: la cadena que conforma al *cromosoma* es desplazada dentro del mismo cromosoma, esto cambiará la disposición de los genes del individuo. El punto de desplazamiento es elegido aleatoriamente (Katoch y col., 2021).
- Mutación por inversión: dentro de la cadena de un individuo, se selecciona una subcadena, se invierten sus valores de *genes*, y se reposiciona en alguna posición aleatoria del *cromosoma* (Katoch y col., 2021).
- Mutación por mezcla: este método de mutación es muy similar a la mutación por inversión, ya que se toma una subsecuencia del *cromosoma* y sus *alelos* son mezclados en un orden aleatorio (Katoch y col., 2021).

En el Algoritmo 1 se presenta el pseudocódigo de un algoritmo genético clásico.

Algoritmo 1 Algoritmo genético clásico

Entrada: Tamaño de población: N,
Cantidad máxima de iteraciones: G,
Función de aptitud: f

Salida: Mejor individuo: Y_m

```
Begin
  Inicializar población inicial de n cromosomas: Y_i (i = 1, 2, 3, ..., n)
  Inicializar contador de iteraciones: t
  Calcular fitness para cada individuo
  while (t ≤ G)
     Seleccionar cromosomas para cruza
     Aplicar cruza
     Aplicar mutación
     Aplicar política de reemplazo para siguiente generación
     Incrementar t+1
  end while
  retornar mejor solución: Y_m
End
```

2.4 Estado del arte y trabajos relacionados

En el estado del arte se han trabajado principalmente dos vertientes de estudio para la detección de grietas en el concreto: técnicas basadas en procesamiento de imágenes, y técnicas basadas en aprendizaje de máquina, principalmente aprendizaje profundo. Es evidente como en el estado del arte a través de la década 2010-2020 se incrementó de forma notable la cantidad de implementaciones basadas en aprendizaje de máquina convencional y aprendizaje profundo para este caso de estudio (Zhang y col., 2021), (Munawar y col., 2021), (Liu y col., 2023), (Gibb y col., 2018), esto llevó consigo una transición en la ciencia de las técnicas de procesamiento de imágenes a aprendizaje de máquina, obteniendo mejores resultados con este último mejorando en gran manera la capacidad de los investigadores para monitorear la condición de diversas obras civiles de concreto.

2.4.1 Técnicas basadas en procesamiento de imágenes

Las técnicas de procesamiento de imágenes se enfocan en la aplicación de técnicas de preprocesamiento de imágenes, filtros digitales, transformaciones de dominios basados en intensidades de los pixeles de las imágenes, entre otros. Estas técnicas se emplean para poder detectar grietas en superficies de concreto bajo diversas condiciones, de manera que el sistema computacional pueda hacer una separación entre las grietas y el resto de la imagen.

La aplicación de filtros digitales y umbralización de imágenes han sido utilizados en numerosos trabajos dando resultados favorables para la tarea de detección de grietas. Una metodología para detección de grietas basada en procesamiento de imágenes fue propuesta por (Talab y col., 2016). La metodología consiste en hacer una transformación a escala de grises de la imagen, para posteriormente aplicar el filtro de *Sobel* a la imagen y detectar los bordes que producen las imperfecciones de las grietas en la imagen; después, se ejecutan más etapas de filtrado para eliminar ruido residual, para por último obtener una imagen binarizada con las grietas delineadas, esto por el método de Otsu.

Otra técnica frecuentemente utilizado es el filtro de Gabor. El filtro de Gabor es una técnica para filtrar imágenes en su dominio espacial y de frecuencia, siendo capaz de detectar texturas, bordes, objetos, etc. Su funcionamiento es similar a un filtro pasabandas. Un ejemplo de esto

fue propuesto por (Salman y col., 2013), en este trabajo se propone el uso del filtro de Gabor para la detección de grietas en concreto utilizando un banco de filtros de Gabor con diversas orientaciones, pudiendo detectar así grietas en distintas orientaciones, para finalmente hacer una fusión de las detecciones de los distintos filtros de Gabor, obteniendo así la detección de la grieta.

A inicios de la década de 2010 era común el uso de técnicas de procesamiento de imagénes para detección de grietas en el concreto, sin embargo, con el rápido crecimiento del campo de la Inteligencia Artificial en la ciencia, el uso de técnicas basadas en aprendizaje de máquina convencional y aprendizaje profundo se ha incrementado considerablemente en los últimos 12 años (Munawar y col., 2021). Las técnicas de aprendizaje de máquina proveen la capacidad de automatizar la detección y clasificación de objetos para así poder aprovechar el potencial de la IA y poder implementar nuevas metodologías de detección de grietas en el concreto.

Para poder contrastar las diferencias entre las técnicas de procesamiento de imágenes con las de aprendizaje profundo, (Dorafshan, Thomas y Maguire, 2018) realizaron un estudio comparativo donde implementan diferentes filtros de detección de bordes para detectar grietas en imágenes. Se utilizaron filtros de dominio espacial, como: *Sobel*, *Prewitt*, *Roberts*, *Laplaciano de Gaussiano*; también filtros de dominio de frecuencia: *Butterworth*, y *Gaussiano*. Los filtros de dominio espacial funcionan haciendo pasar un kernel deslizante a través de la imagen, el kernel cuenta con coeficientes estratégicamente seleccionados para poder resaltar bordes orientados de manera vertical, y horizontal. Por su parte, los filtros que trabajan en el dominio de la frecuencia, requieren una transformación de las imágenes para obtener sus componentes frecuenciales, después se miden las distancias entre los pixeles de la imagen al centro del origen de la frecuencia detectada, y así detectar grietas extendidas en distintas direcciones de la imagen. Por otro lado, para comparar se utilizó la arquitectura de RNC *AlexNet*, la cual cuenta con 5 capas de convolución, 3 capas de agregación por valor máximo, y 3 capas de neuronas totalmente conectadas como clasificador.

Los resultados en (Dorafshan, Thomas y Maguire, 2018) muestran que el acercamiento de aprendizaje profundo con *AlexNet* resultó ser superior en todos los aspectos, obteniendo una mayor tasa de aciertos, una mayor puntuación $F1$, menores tiempos de cómputo (en predicciones), dando así un claro ejemplo de la utilidad del aprendizaje profundo en el monitoreo de

obras civiles.

Tabla 2.1: Resumen de trabajos relacionados basados en técnicas de procesamiento de imágenes.

Autores	**Conjunto de datos**	**Técnica principal**	**Preprocesamiento**	**Métricas**
Talab y col., 2016	Recolectado por autores	Método de *Otsu*	Conversión a escala de grises, Filtro de *Sobel*	Distribución de grises
Salman y col., 2013	Recolectado por autores	Filtro de *Gabor*	-	Precisión: 0.95, Recuperación: 0.78
Dorafshan, Thomas y Maguire, 2018	Recolectado por autores	Filtros de: *Sobel, Prewitt, Roberts, Laplaciano de Gaussiano, Butterworth, Gaussiano*. RNC: AlexNet	-	Sensibilidad: 0.86, Especificidad: 0.99, Puntuación $F1$: 0.89

2.4.2 Técnicas basadas en aprendizaje de máquina

En 2014, (Prasanna y col., 2014) propusieron una metodología con varias etapas que implementaban técnicas de aprendizaje de máquina convencional para detección de grietas en concreto. El primer paso de su metodología, consiste en la localización de segmentos de grietas en imágenes por bloques ajustando líneas con medidas estadísticas de intensidad de pixeles. En segundo lugar, se extraen características en las zonas donde se detectaron las líneas para obtener características basadas en intensidades, en gradiente, y espacio; con esto se obtienen "vectores ajustados de multi-características" para poder ingresarlos como datos a los algoritmos de clasificación. Como última etapa, utilizan algoritmos como *SVM, Adaboost*, y *Bosque aleatorio* para clasificar las características. Los autores reportaron un 0.90 de tasa de aciertos con esta metodología.

Siguiendo con la tendencia, se han utilizado distintas arquitecturas de RNC como *U-Net* en (Escalona y col., 2019), la cual fue concebida como una RNC para segmentación de imágenes biomédicas compuesta de 23 capas convolucionales en total. Sin embargo, esta arquitectura fue utilizada para la detección de grietas en concreto. *U-Net* consta de dos caminos convolucionales,

uno que contrae los mapas de características como una RNC común, y otro que expande las características concatenando mapas de características. Se implementaron dos variantes menos profundas de *U-Net* (11 y 7 capas convolucionales) para poder comparar los resultados de modelos menos complejos que el original. Los mejores resultados se obtuvieron de *U-Net* con 11 capas convolucionales (*U-Net-B*), obteniendo una puntuación $F1$ de 0.95 en una de sus configuraciones experimentales. Como variante extra, se utilizó la arquitectura *VGG-16* pre-entrenada con el conjunto de datos *ImageNet*, sin embargo, esta última tuvo una puntuación $F1$ de 0.89.

Existen variantes de extracción de características de imágenes de superficies de concreto utilizando técnicas de transformación y distintos tipos de RNC, como la propuesta RNC unidimensional, llamada así porque su entrada se dispone como un vector unidimensional de valores. Por ejemplo, en (Zhang y col., 2021) proponen una metodología cuyo primer paso es transformación de imágenes a dominio de la frecuencia a través de la transformada rápida de Fourier (*FFT* por sus siglas en inglés), esto para disminuir el uso de recursos computacionales a la hora de entrenar los modelos de clasificación, estos vectores de componentes frecuenciales se utilizan como entrada a la RNC unidimensional. Finalmente, utilizando una red de memoria a corto plazo se realiza la clasificación en tiempo real de las imágenes, alcanzando una tasa de aciertos de 0.99, en contraste con una metodología similar que utiliza una RNC convencional que alcanzó 0.97. Cabe señalar que los autores en (Zhang y col., 2021) declaran que la arquitectura final de la RNC utilizada fue obtenida a través de un extensivo proceso de experimentación de prueba y error. Este largo proceso puede ser mitigado con algún método de búsqueda que ayude a definir los hiperparámetros estructurales de la RNC.

Otras propuestas para detección de grietas en el concreto utilizan arquitecturas de RNC poco complejas, alcanzando altas tasas de aciertos. En (Cha y col., 2017) se propone una arquitectura que consta de solo cuatro capas convolucionales, dos de agrupamiento, y una capa completamente conectada. En comparación con otras arquitecturas reconocidas, esta cuenta con un menor número de parámetros debido a sus reducidas dimensiones. El conjunto de datos usado fue recopilado por los autores del trabajo, contiene más de 40,000 imágenes. La tasa de aciertos que reportan los autores es de 0.97.

En otros trabajos, utilizan el aprendizaje por transferencia con ajuste fino (Gao y Mosalam,

2018) para obtener buenos resultados de métricas sin tener largos tiempos de entrenamiento. Los autores utilizaron la arquitectura *VGG-16* pre-entrenada con el conjunto de datos *ImageNet*, para detectar distintos tipos de daño en superficies de concreto, como grietas y resquebrajamiento. La RNC fue ajustada finamente re-entrenando los últimos dos bloques convolucionales. El conjunto de datos fue recolectado y nombrado por los autores como *Structural ImageNet*. La tasa de aciertos más alta reportada es de 0.945.

Tabla 2.2: Resumen de trabajos relacionados basados en técnicas de aprendizaje de máquina.

Autores	**Conjunto de datos**	**Técnica principal**	**Preprocesamiento**	**Métricas**
Prasanna y col., 2014	Recolectado por autores	*SVM, Adaboost, Bosque aleatorio*	Detección de líneas, *Pirámide Laplaciana*	Tasa de aciertos: 0.91, sensibilidad 0.74, precisión: 0.97
Escalona y col., 2019	*CFD, AigleRN*	RNC: U-Net-B	-	Precisión: 0.97, Recuperación: 0.94, Puntuación $F1$: 0.95
Zhang y col., 2021	*SDNET2018*	RNC personalizada	*FFT*, filtro pasa altas	Tasa de aciertos: 0.99
Cha y col., 2017	Recolectado por autores	RNC personalizada	-	Tasa de aciertos: 0.98
Gao y Mosalam, 2018	Recolectado por autores (*Structural ImageNet*)	*VGG-16*	-	Tasa de aciertos: 0.945

2.4.3 Optimización de RNA con algoritmos genéticos

Los algoritmos genéticos han sido ampliamente utilizados para optimización de algoritmos en el estado del arte, gracias a su flexibilidad y capacidad de encontrar soluciones satisfactorias en los distintos problemas que se presentan en la ciencia. Existen múltiples trabajos de investigación donde se presentan algoritmos genéticos optimizando parámetros o hiperparámetros de redes neuronales, ayudando a mejorar su desempeño. Un ejemplo claro es el trabajo realizado por (Yan, 2010) donde utilizó un algoritmo genético que establece restricciones de edad de los individuos de la población para aplicar el operador de cruza, restricciones de parentezco consanguíneo, entre otras; tratando de imitar la mecánica de la reproducción de la especie humana.

El autor aplica este esquema para optimizar los pesos iniciales de una red neuronal artificial, utilizado sobre un caso de estudio de clasificación de patrones en código grey.

El mismo año, (Zhao y col., 2010) optimizaron los pesos iniciales de una red neuronal con un algoritmo genético clásico, esto aplicado al caso de estudio de detección de números *Renminbi* (*RMB*), los cuales son números identificadores únicos que se encuentran en la moneda de circulación en China.

Otros trabajos hacen aportes de optimización de hiperparámetros de RNC con algoritmos genéticos. En el trabajo de (Gibb y col., 2018) se propone un esquema de optimización de hiperparámetros estructurales para una RNC con un algoritmo genético, el cual explorará el espacio de búsqueda de hiperparámetros estructurales, estos hiperparámetros son: cantidad de capas de convolución, cantidad de capas de agrupamiento por valor máximo, tamaño de filtros convolucionales, y cantidad de filtros convolucionales. Se reporta en los resultados que el mejor individuo del algoritmo genético obtuvo una tasa de aciertos de 0.89. Los autores resaltan la importancia de utilizar un algoritmo genético para explorar el espacio de búsqueda de hiperparámetros, ya que para encontrar el mejor individuo solo se construyeron 300 modelos, en contraste con los $16,000$ que se tendrían que ejecutar en una búsqueda exhaustiva para el trabajo propuesto en (Gibb y col., 2018).

Por otro lado, en (Chaiyasarn y col., 2021) proponen una metodología basada en aprendizaje de máquina para clasificación de imagenes de superficies de concreto. Las imágenes son obtenidas con un Vehículo Aéreo No Tripulado (VANT) y una cámara DSLR. Después de almacenar las imágenes captadas, se utiliza una RNC para extracción de características; su arquitectura consta de tres bloques, cada uno compuesto por una capa de convolución y una capa de agrupación por valor máximo. Una vez extraídas las características, se probaron distintos algoritmos clasificadores, como: *SVM*, RNA, *Bosque aleatorio*, y RNA optimizada por algoritmos genéticos. La optimización de la RNA por algoritmos genéticos se implementó sobre los pesos iniciales de la red, es decir, cada individuo del algoritmo genético es un modelo cuyos pesos iniciales serán definidos por el algoritmo genético. Los resultados indican que el método de clasificación con *SVM* obtuvo una mejor tasa de aciertos que el resto de los algoritmos de clasificación con 0.94, y 0.91 para la RNA optimizada por algoritmos genéticos, cuyo clasificador fue el segundo lugar en tasa de aciertos. Los autores en (Chaiyasarn y col., 2021) resaltan la importancia de utilizar

RNC para extraer características de imágenes ya que es una herramienta con la aptitud necesaria para lograr representaciones abstractas suficientes para caracterizar imágenes de manera efectiva y automática. También hacen énfasis en integrar clasificadores modificados al bloque extractor de carácterísticas del método para así obtener resultados deseados.

Un ejemplo de optimización de hiperparámetros de una red neuronal artificial se describe en el trabajo de (Itano y col., 2018). En este trabajo se comparan los resultados obtenidos en diversos conjuntos de datos con la optimización, y sin la optimización evolutiva. El enfoque principal fue en optimizar hiperparámetros estructurales, de entrenamiento, e hiperparámetros de regularización para mejorar el desempeño del modelo generado. Los investigadores reportan resultados superiores a metodologías similares, y al uso del modelo sin optimización por algoritmos genéticos.

Es notorio como los algoritmos genéticos tienen la capacidad de optimizar modelos con múltiples combinaciones posibles como lo son las redes neuronales, dada su versatilidad para poder tratar problemas complejos y obtener buenos resultados.

Tabla 2.3: Resumen de trabajos relacionados basados en RNA optimizadas con AG.

Autores	**Conjunto de datos**	**Técnica principal**	**Preprocesamiento**	**Métricas**
Yan, 2010	Recolectado por autores	RNC optimizada por AG (pesos iniciales)	-	Error promedio cuadrático: 1×10^{-5}
Chaiyasarn y col., 2021	Recolectado por autores	RNC, RNA optimizada por AG (pesos iniciales)	-	Tasa de aciertos: 0.90
Zhao y col., 2010	Recolectado por autores	RNC, RNA optimizada por AG (pesos iniciales)	-	Tasa de reconocimiento: 0.95
Gibb y col., 2018	Recolectado por autores	RNC optimizada por AG (cantidad de capas de convolución y agrupamiento, tamaño de filtros, cantidad de filtros)	-	Tasa de aciertos: 0.94
Itano y col., 2018	*Breast Cancer Wisconsin Diagnostic (BCWD), Ionosphere (Iono), Connectionist Bench - Sonar, Mines vs. Rocks (Sonar), Heart Disease (Heart), Iris*	RNA optimizada por AG (hiperparámetros y regularización)	-	Tasa de aciertos: 0.99

Capítulo 3

Metodología

En este capítulo se presenta la metodología propuesta para optimización de hiperparámetros en modelos de clasificación con algoritmos genéticos para el caso de estudio de detección de grietas en puentes de concreto reforzado.

El conjunto de datos utilizado es el *SDNET2018*, que contiene 56,092 imágenes de superficies de estructuras civiles como puentes, pavimento y muros.

Una vez obtenido el conjunto de datos, se utiliza el extractor de características pre-entrenado de la arquitectura VGG-16 (aprendizaje por transferencia) para realizar la caracterización de las imágenes, es decir, extraer las características de las imágenes y abstraerlas en arreglos numéricos para poder utilizarlos como entrada para una RNA clasificadora. Posteriormente, se implementa el modelo base de la red neuronal clasificadora VGG-16 para evaluar su desempeño con la caracterización realizada y el conjunto de datos.

Luego, se optimizan los hiperparámetros de la red neuronal clasificadora utilizando algoritmos genéticos, lo que permite probar múltiples configuraciones de hiperparámetros en busca de un conjunto de valores que mejore el desempeño del modelo. Finalmente, se evalúa y compara el desempeño de los modelos generados. El esquema metodológico se presenta en la Figura 3.1.

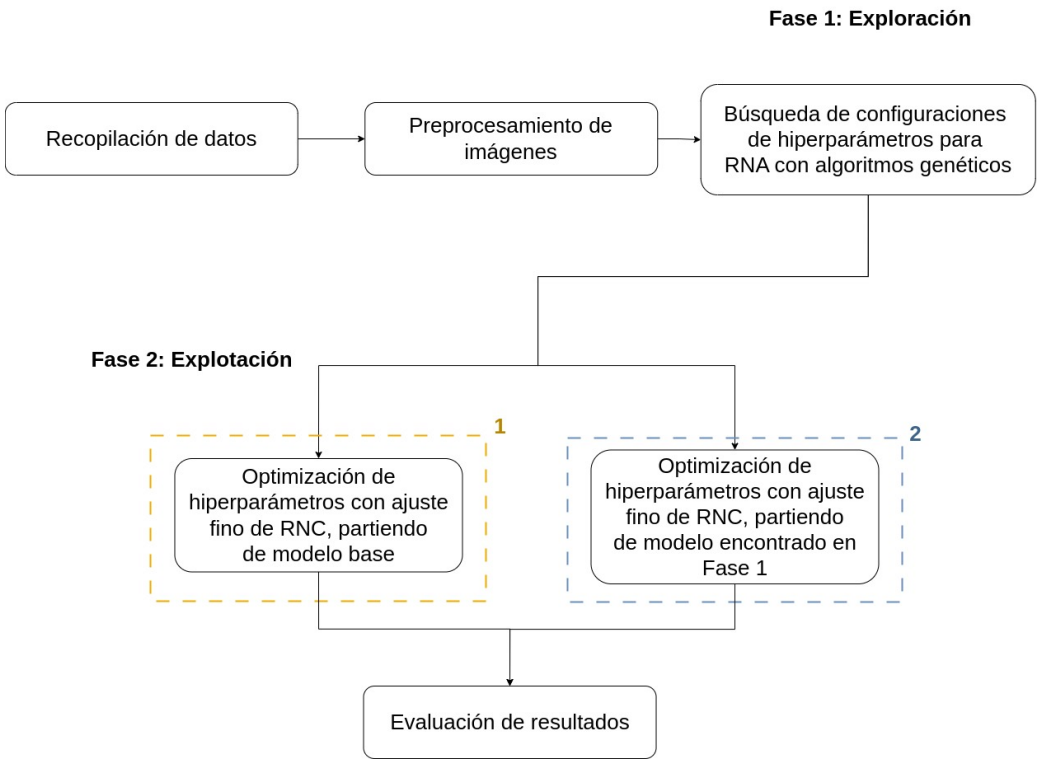


Figura 3.1: Diagrama de flujo de la metodología propuesta.

3.1 Conjunto de datos

El conjunto de datos utilizado es *SDNET2018*, el cual consiste de $56,092$ imágenes RGB de 256×256 píxeles (Dorafshan, Thomas y col., 2018). Todas estas imágenes son de superficies de estructuras civiles de concreto reforzado como: puentes, pavimento, y muros. En el conjunto de datos se encuentran en total $8,484$ imágenes con superficies de concreto con grietas, y $47,608$ imágenes de superficies de concreto sin grietas. En la Tabla 3.1 se muestra la distribución de imágenes y clases dentro del conjunto de datos.

Tabla 3.1: Desglose de conjunto de datos *SDNET2018* por cantidad de imágenes.

Conjunto de datos	**Subdivisiones**	**Clases**
SDNET2018: 56,092	Puentes: 13,620	Positivas: 2,025 Negativas: 11,595
	Pavimento: 24,334	Positivas: 2,608 Negativas: 21,726
	Muros: 18,140	Positivas: 3,851 Negativas: 14,287

El conjunto de datos *SDNET2018* es uno de los conjuntos de imágenes de superficies con grietas en concreto más grandes en la literatura científica, fue recopilado por investigadores de la Universidad Estatal de Utah, y se ha utilizado en diversos trabajos de investigación, como (Dorafshan, Thomas y Maguire, 2018), (Dorafshan y Maguire, 2017), (Dorafshan, Thomas, Coopmans y col., 2018), entre otros.

El conjunto consiste en imágenes con distintos tipos de superficies y grietas, incluyendo grietas finas, grietas gruesas, manchas de pintura, sombras, obstrucciones de escombro o plantas, texturas lisas, texturas rugosas, etc. Estas características de las imágenes permitirán al modelo poder aprender a detectar la presencia de grietas aún cuando se encuentra ruido u obstrucciones en el ambiente.

Las imágenes de puentes fueron tomadas de tableros de puentes que el equipo de investigación conserva y prueba en el Laboratorio de Sistemas, Materiales y Salud Estructural (*SMASH*, por sus siglas en inglés). Las imágenes de pavimento y muros, fueron capturadas en distintas estructuras del campus de la Universidad Estatal de Utah. En la Figura 3.2 se muestran algunas imágenes de muestra del conjunto de datos.

Figura 3.2: Imágenes de muestra del conjunto de datos *SDNET2018*.

3.2 Extracción de características de imágenes

Las RNC son capaces de realizar la caracterización de imágenes de manera automática. En este estudio, se utilizaron los bloques convolucionales de la arquitectura *VGG-16* para extraer características del conjunto de datos. Es importante destacar que el extractor de características empleado está preentrenado con un conjunto de datos a gran escala llamado *ImageNet*.

Para aprovechar el conocimiento adquirido por *VGG-16* con *ImageNet*, se utilizó el aprendizaje por transferencia en modo extractor de características. Esto permite extrapolar los patrones aprendidos previamente por el modelo a un nuevo conjunto de datos, como *SDNET2018*. En otras palabras, el conocimiento aprendido por *VGG-16* con *ImageNet* puede aplicarse para caracterizar *SDNET2018*, ahorrando así recursos, ya que no es necesario entrenar una RNC desde cero. Además, dado que *ImageNet* cuenta con más de 1,000,000 de imágenes y 1,000 clases de objetos variados, la capacidad de extracción de características de *VGG-16* es altamente gene-

ralizable. Esto permite enfocar los esfuerzos en mejorar la RNA clasificadora, optimizando así el proceso y recursos, principalmente tiempo.

En la Figura 3.3 se muestra el diagrama de la etapa de extracción de características.

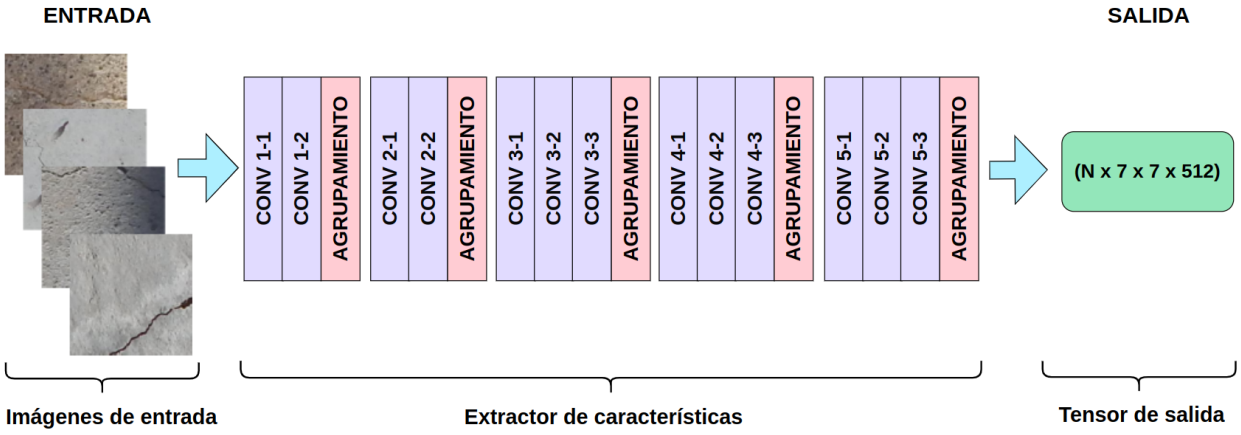


Figura 3.3: Extracción de características de imágenes con los bloques convolucionales de la arquitectura *VGG-16*. La salida es un tensor de 4 dimensiones, siendo estas <*Instancias*, *Alto*, *Ancho*, *Canales*>.

3.3 Generación y evaluación de modelo de clasificación base (*VGG-16*)

Se generó un modelo clasificador basado en la arquitectura *VGG-16* como caso base para evaluar su comportamiento con el conjunto de datos. Este conjunto incluye todas sus subdivisiones: puentes, muros y pavimento, tanto en situaciones de datos balanceados como desbalanceados en las distintas clases.

Para el procesamiento, se utilizaron los tensores de características obtenidos del extractor de características de *VGG-16* como entradas para el modelo. Antes de ingresar los tensores a la red neuronal, se les somete a una fase de aplanado, transformando el tensor de cuatro dimensiones en un arreglo unidimensional. Este arreglo actúa como una capa de entrada pasiva para la red neuronal.

Luego, los datos se procesan a través de dos capas ocultas, cuyas neuronas utilizan la función

de activación *ReLU*. Finalmente, en la capa de salida, se emplean dos neuronas (una para cada clase) con la función de activación *softmax*, que proporciona la probabilidad de que la observación pertenezca a cada una de las clases (Simonyan y Zisserman, 2014).

Como hiperparámetros del modelo, se utilizaron técnicas de regularización para evitar el sobreajuste, como tasa de difusión de 0.5 en las dos primeras capas ocultas, regularización L_2 con el valor de 5×10^{-4}. La tasa de aprendizaje inicia con el valor de 10^{-2} y decrece por un factor de 10 cuando la tasa de aciertos del conjunto de datos de validación no mejora (Simonyan y Zisserman, 2014). El modelo fue entrenado por 100 épocas.

Una vez entrenado el modelo, se evalúa su desempeño utilizando predicciones sobre un conjunto de datos de prueba. Cada predicción puede clasificarse como correcta o incorrecta. Las clasificaciones correctas e incorrectas se pueden expresar de la siguiente manera:

- VP: verdadero positivo. Representa a una observación del conjunto de datos que pertenece a la clase positiva, y el modelo la clasificó como perteneciente a la clase positiva.
- FP: falso positivo. Representa a una observación del conjunto de datos que pertenece a la clase negativa, y el modelo la clasificó como perteneciente a la clase positiva.
- VN: verdadero negativo. Representa a una observación del conjunto de datos que pertenece a la clase negativa, y el modelo la clasificó como perteneciente a la clase negativa.
- FN: falso negativo. Representa a una observación del conjunto de datos que pertenece a la clase positiva, y el modelo la clasificó como perteneciente a la clase negativa.

Posteriormente, se evalúa el desempeño del modelo con las métricas presentadas a continuación.

- Tasa de aciertos: Cantidad de clasificaciones correctas con respecto a la cantidad total de observaciones.

$$TasaAciertos = \frac{VP + VN}{VP + VN + FP + FN} \tag{3.1}$$

- Precisión: Cantidad de observaciones clasificadas correctamente como positivas con respecto a la cantidad total de observaciones que se clasificaron como positivas.

$$Prec = \frac{VP}{VP + FP} \tag{3.2}$$

- Recuperación: Cantidad de observaciones clasificadas correctamente como positivas con respecto a la cantidad total de observaciones que pertenecen a la clase positiva.

$$Rec = \frac{VP}{VP + FN} \tag{3.3}$$

- Puntuación $F1$: Métrica que combina Precisión y Recuperación para obtener una media de estas dos métricas.

$$F1 = \frac{2(Prec * Rec)}{Prec + Rec} \tag{3.4}$$

El flujo seguido para la evaluación del modelo se muestra en la Figura 3.4.

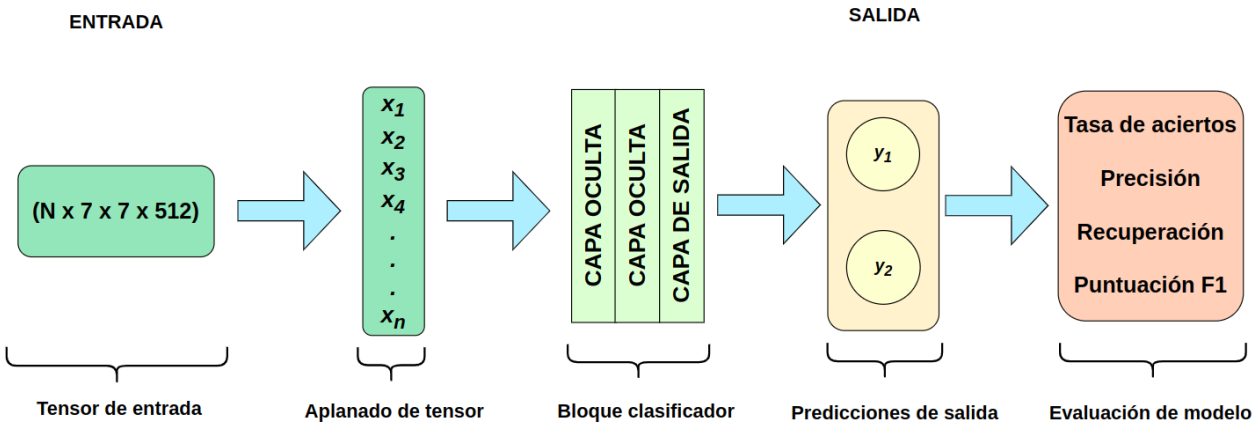


Figura 3.4: Evaluación de bloque clasificador de la arquitectura *VGG-16*.

3.4 Esquema de optimización de hiperparámetros

Partiendo de la red neuronal clasificadora de la arquitectura *VGG-16*, se utilizó un algoritmo genético que optimice sus hiperparámetros para obtener distintas configuraciones y encontrar un modelo que tenga un mejor comportamiento que el modelo base, así mejorando la detección de grietas en imágenes de superficies de concreto.

Cada individuo del algoritmo genético está compuesto por genes que abarcan aspectos como la inicialización de los parámetros, hiperparámetros estructurales de la red neuronal, e hiperparámetros que competen a la fase de entrenamiento del modelo.

En la Tabla 3.2 se presenta el modelo de individuo propuesto para la optimización del modelo de clasificación.

Tabla 3.2: Componentes del individuo en el algoritmo genético y sus valores permitidos.

Gen	**Hiperparámetros**	**Valores permitidos**
1	Inicialización de parámetros	0 - LeCun Normal 1 - LeCun Uniform 2 - Glorot Normal 3 - Glorot Uniform
2	Cantidad de neuronas en primera capa oculta	$\{512, 1024, 2048, 4096\}$
3	Cantidad de neuronas en segunda capa oculta	$\{512, 1024, 2048, 4096\}$
4	Función de activación	0 - Sigmoide 1 - *ReLU* 2 - *Tanh* 3 - *SELU* 4 - *Tansig*
5	Tasa de difusión	$\{0, 0.5, 0.6, 0.7, 0.8, 0.9\}$
6	Tasa de aprendizaje	$\{0.1, 0.01, 0.001, 0.0001\}$
7	Tamaño de lotes	$\{8, 16, 32, 64\}$
8	Algoritmo de optimización de parámetros	0 - *SGD* 1 - *AdaDelta* 2 - *RMSProp* 3 - *ADAM*

Cada cromosoma de la población representa una configuración específica de la red neuronal. Para evaluar la aptitud de un individuo, se entrenará el modelo resultante de la configuración de sus genes. Posteriormente, se evaluará el modelo y se utilizará su puntuación $F1$ de la clase positiva como medida de aptitud, ya que esta métrica es un indicador de la capacidad de detección de grietas del modelo.

En la Figura 3.5 se presenta un ejemplo de un individuo de la población del algoritmo genético.

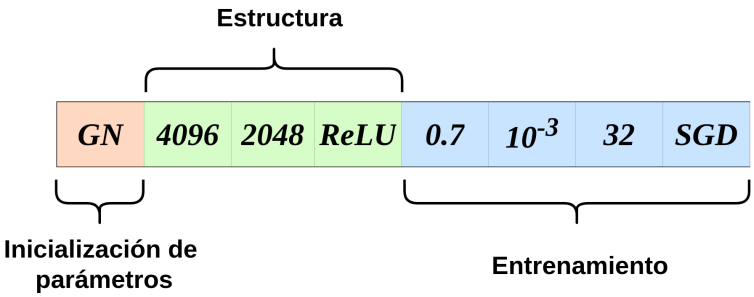


Figura 3.5: Ejemplo de individuo de la población del algoritmo genético. En el *cromosoma* se pueden detectar las tres divisiones principales, que son: inicialización de parámetros (*Gen* 1, naranja), hiperparámetros estructurales (*Genes* 2-4, verde), e hiperparámetros de entrenamiento (*Genes* 5-8, azul).

Dado que para evaluar un individuo se necesita entrenar el modelo correspondiente a su configuración, el algoritmo genético constará de pocas generaciones y pocos cromosomas. Esto se debe a que un número elevado de generaciones o individuos implicaría tiempos de procesamiento prolongados para los equipos de cómputo. En trabajos como (Fujino y col., 2017), (Zhao y col., 2010), y (Gibb y col., 2018) se pude observar como se sigue este principio utilizando poblaciones y cantidad de generaciones no mayores a 30.

Para optimizar los hiperparámetros de la red neuronal, se hace uso de un algoritmo genético clásico cuyas especificaciones se presentan en la Tabla 3.3.

La optimización de hiperparámetros se llevó a cabo en dos fases:

- Fase 1, exploración: consiste en encontrar distintas configuraciones de hiperparámetros a través de varias ejecuciones del AG, la configuración que genere el mejor modelo de clasificación será utilizado para la Fase 2.

- Fase 2, explotación: los hiperparámetros del mejor modelo obtenido en la Fase 1, se usarán para optimizar la RNC con ajuste fino, y posterior a eso realizar la búsqueda de nuevos modelos de clasificación con AG a partir del modelo finamente ajustado.

Tabla 3.3: Operadores y parámetros de algoritmo genético.

Parámetro	**Valor**
Tamaño de población	20
Cantidad de generaciones	50
Inicialización de la población	Aleatoria
Medida de aptitud	Puntuación $F1$
Método de selección	Torneo binario
Tipo de cruza	Cruza de un punto
Tasa de cruza	1.0
Tipo de mutación	Variación aleatoria de gen
Tasa de mutación	0.1
Política de reemplazo	Mantener los 10 hijos generados en la generación actual y mantener los 10 mejores padres para la siguiente generación

3.5 Fase 1, exploración

La fase de exploración consiste en utilizar un AG clásico para explorar el espacio de búsqueda y encontrar una configuración de hiperparámetros para una RNA que tenga un mejor desempeño y/o menor complejidad que el clasificador original de *VGG-16*. Para obtener dichos modelos, el AG se ejecuta 3 veces, y se guarda el mejor modelo de cada ejecución, estos serán nombrados **Modelo A**, **Modelo B**, y **Modelo C**. Para entrenar los modelos generados por el algoritmo genético, se utilizan las características extraídas por los bloques convolucionales de *VGG-16*, que fue empleado en modo extractor de características, tal como se describe en la Sección 3.2.

Posteriormente, se utiliza la configuración de hiperparámetros del mejor modelo obtenido en la Fase 1 como punto de partida para generar nuevos modelos de clasificación a partir de un ajuste fino de la RNC en la Fase 2.

Una representación de los modelos generados se muestra en la Figura 3.6.

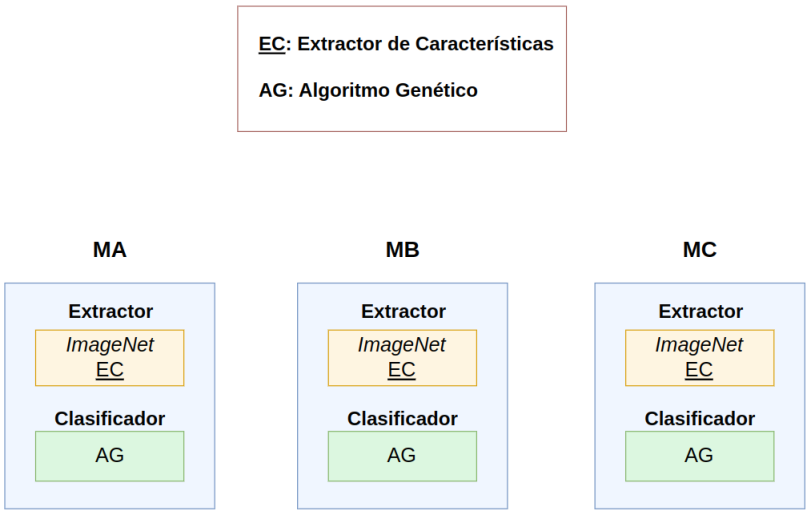


Figura 3.6: Fase 1, exploración.

3.6 Fase 2, explotación

La fase de explotación consiste en ejecutar un flujo de generación de modelos a partir del mejor modelo encontrado por un algoritmo genético en la Fase 1. Esta fase se divide en dos flujos experimentales: Flujo 1 y Flujo 2. Ambos flujos son prácticamente idénticos, con la única diferencia de que el primer modelo del Flujo 1 se genera a partir del clasificador *VGG-16*, mientras que el primer modelo del Flujo 2 se genera utilizando el mejor modelo encontrado en la Fase 1. La finalidad de tener ambos flujos es contrastar el desempeño de los modelos.

Al inicio de cada flujo se realiza un ajuste fino entrenando la base convolucional de *VGG-16* con el clasificador correspondiente al flujo. Este ajuste fino permite adecuar los filtros del extractor de características para caracterizar las imágenes del concreto de manera más efectiva.

Después de entrenar el nuevo modelo con el ajuste fino, los bloques convolucionales finamente ajustados se utilizaron como extractor de características para ejecutar de nuevo el AG y encontrar nuevas configuraciones de hiperparámetros y generar dos nuevos modelos. Uno de los dos nuevos modelos utiliza agrupación global por promedio, para disminuir la dimensionalidad

de la entrada a la RNA clasificadora, y así disminuir la complejidad del modelo. En la Figura 3.7 se muestra una representación de los modelos correspondientes a la Fase 2.

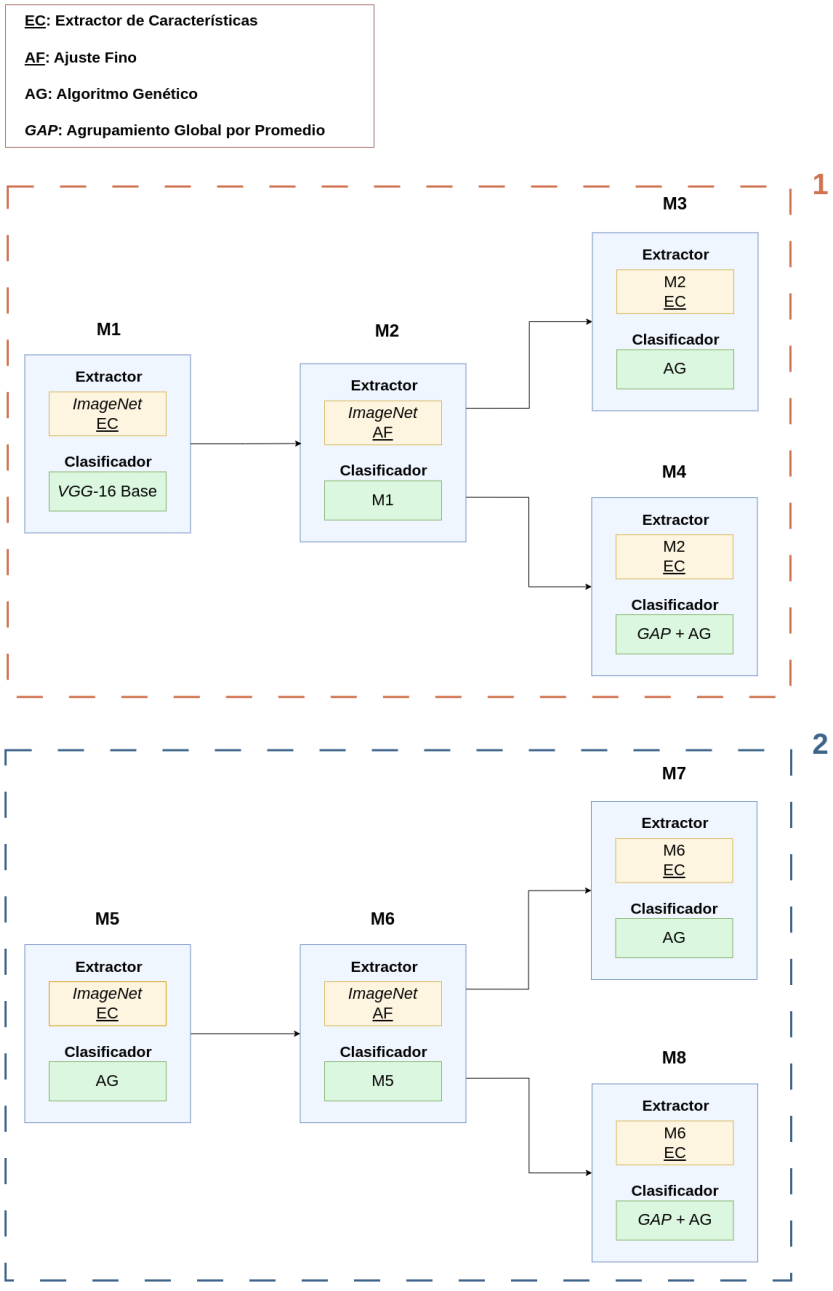


Figura 3.7: Fase 2, explotación.

Capítulo 4

Experimentación y resultados

En este capítulo se exponen los resultados de los experimentos realizados en este trabajo de investigación. Se presentan las evaluaciones de las distintas configuraciones experimentales utilizadas, comenzando con la arquitectura de red neuronal artificial original de *VGG-16*, designada como el caso base. Además, se incluyen los resultados de la evaluación de los distintos modelos de clasificación generados por los algoritmos genéticos (AGs).

El AG clásico sirve como primer punto de comparación para el caso base, siendo el caso más básico de AG para resolver el problema de optimización de hiperparámetros. La evaluación de los modelos se realiza de acuerdo a las métricas descritas en el Capítulo 3, que también proporciona las especificaciones del AG clásico utilizado.

4.1 Ambiente computacional

Los experimentos fueron realizados utilizando una laptop modelo Acer Predator Helios 300 2022 con el sistema operativo Ubuntu 22.04, un procesador Intel Core i7-12700H de 20 núcleos a 2.3 GHz, 16 GB de RAM, 1 TB de almacenamiento SSD interfaz M2, y una unidad de procesamiento de gráficos (*GPU*) NVIDIA RTX 3060 con 6 GB de RAM.

Para el entrenamiento de los modelos de clasificación, se utilizó código en Python 3.6.7, en conjunto con la biblioteca Keras 2.6.0 con TensorFlow 2.6.2 como *back-end*.

4.2 Evaluación de modelo de clasificación: *VGG-16* con aprendizaje por transferencia

La arquitectura del clasificador *VGG-16* se utilizó como modelo base para contar con un punto de partida. Se emplearon cuatro configuraciones experimentales, cada una compuesta por distintos subconjuntos de imágenes del conjunto de datos original (*SDNET2018*). Algunas de estas configuraciones presentan clases balanceadas, mientras que otras tienen clases desbalanceadas.

El balance de clases implica que en el conjunto de datos hay la misma cantidad de instancias para cada clase (positiva y negativa). En contraste, en conjuntos de datos con clases desbalanceadas, la cantidad de ejemplos para cada clase es diferente. A continuación, se desglosan las configuraciones experimentales evaluadas con el modelo de clasificación en su estado base:

1. Imágenes de puentes con desbalance de datos ($2,025$ pertenecientes a la clase positiva y $11,595$ pertenecientes a la clase negativa).

2. Imágenes de puentes con balance de datos ($2,000$ pertenecientes a la clase positiva y $2,000$ pertenecientes a la clase negativa).

3. Imágenes de puentes, muros, y pavimento con desbalance de datos ($2,500$ pertenecientes a la clase positiva y $13,599$ pertenecientes a la clase negativa).

4. Imágenes de puentes, muros, y pavimento con balance de datos ($6,000$ pertenecientes a la clase positiva y $6,000$ pertenecientes a la clase negativa).

En algunas de las distintas configuraciones experimentales se incluyen imágenes de grietas en distintas estructuras de concreto además de puentes, así pudiendo evaluar el comportamiento del modelo con mayor variabilidad en las imágenes del conjunto de datos, debido a que las grietas formadas en pavimento y muros son visualmente distintas a las grietas en puentes. Esta variabilidad permitirá evaluar el comportamiento del modelo con grietas en concreto en general.

El modelo de clasificación del caso base fue entrenado con los siguientes hiperparámetros:

- Épocas: 200.

- Tamaño de lote: 32.

- Algoritmo de optimización: (SGD).
- Función de pérdida: Entropía cruzada binaria.
- Tasa de aprendizaje adaptativa: 0.01 inicial, y reducir en factor de $\frac{1}{10}$ si cada cinco épocas no mejora su tasa de aciertos de validación.

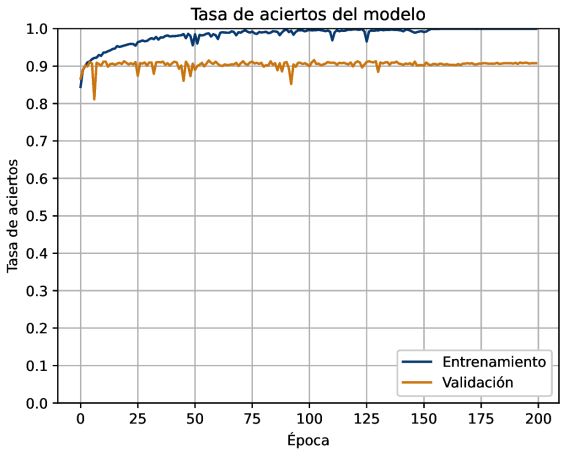
En las Tablas 4.1 y 4.2 se muestran los resultados de las distintas configuraciones experimentales con el modelo en el caso base.

Tabla 4.1: Métricas de evaluación para configuraciones experimentales con el modelo del caso base (Conjuntos de datos con desbalance de clases).

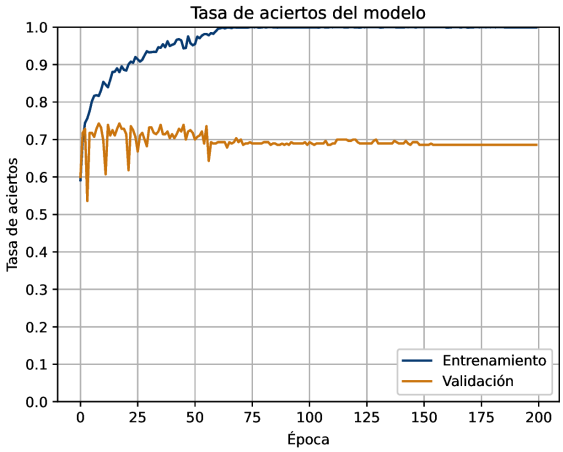
Clase	**Modelo configuración 1**			**Modelo configuración 3**		
	Precisión	**Recuperación**	***F*1**	**Precisión**	**Recuperación**	***F*1**
Grieta	0.78	0.47	0.59	0.78	0.49	0.60
No grieta	0.91	0.97	0.94	0.91	0.97	0.94

Tabla 4.2: Métricas de evaluación para configuraciones experimentales con el modelo del caso base (Conjuntos de datos con balance de clases).

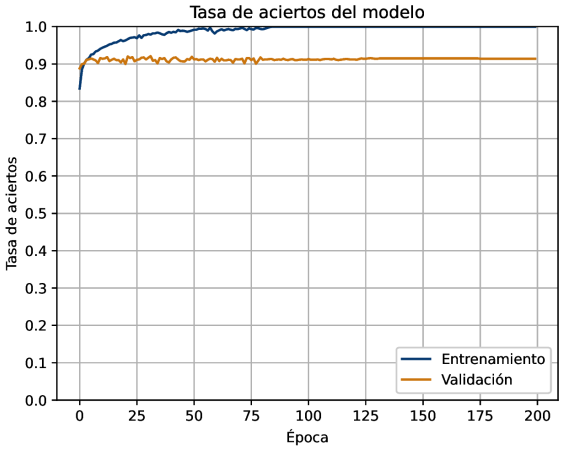
Clase	**Modelo configuración 2**			**Modelo configuración 4**		
	Precisión	**Recuperación**	***F*1**	**Precisión**	**Recuperación**	***F*1**
Grieta	0.81	0.71	0.76	0.82	0.75	0.78
No grieta	0.72	0.82	0.77	0.76	0.83	0.79



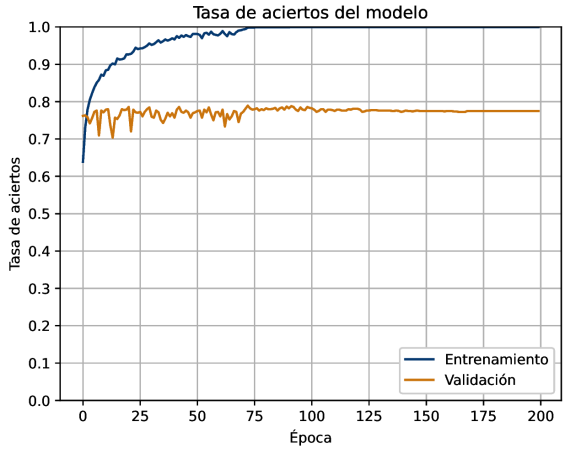
(a) Configuración 1.



(b) Configuración 2.



(c) Configuración 3.



(d) Configuración 4.

Figura 4.1: Curvas de entrenamiento caso base *VGG-16* con distintas configuraciones de datos. (a) Imágenes de puentes con desbalance de datos. (b) Imágenes de puentes con balance de datos. (c) Imágenes de puentes, muros, y pavimento con desbalance de datos. (d) Imágenes de puentes, muros, y pavimento con balance de datos.

Los resultados del modelo base con las configuraciones experimentales indican que el modelo aprendió las características extraídas de los bloques convolucionales de *VGG-16*. Concretamente, las configuraciones 2 y 4 donde se cuenta con clases balanceadas, el modelo presentó una capacidad similar de clasificación tanto para la clase positiva como para la negativa, obtenien-

do una Puntuación $F1$ máxima de 0.79 para la clase negativa y 0.77 para la clase positiva. Sin embargo, las curvas de entrenamiento correspondientes a estas configuraciones, que están representadas en las Figuras 4.1 (b) y (d), presentan un sobreajuste en el entrenamiento, ya que existe una diferencia entre los valores de tasa de aciertos para los conjuntos de datos de entrenamiento y validación.

En contraste con estos resultados, los experimentos con desbalance de clases mostraron un sesgo notable hacia la clase mayoritaria (clase negativa). Esto se debe a que, en las configuraciones, hay al menos cinco veces más imágenes de superficies de concreto sin grietas que de superficies con grietas. Este sesgo se evidencia en las tasas de recuperación para la clase positiva en las configuraciones experimentales presentadas en la Tabla 4.1, siendo 0.49 la máxima tasa de recuperación de la clase positiva entre las dos configuraciones. Es decir, el modelo falla al clasificar como positivas aquellas instancias que realmente pertenecen a la clase positiva, lo que reduce la Puntuación $F1$ del modelo, alcanzando un máximo de 0.60 para las clases positivas.

En las curvas de entrenamiento correspondientes a estas configuraciones, representadas en las Figuras 4.1 (a) y (c), se observa un mejor ajuste en comparación con las configuraciones con balance de datos. Sin embargo, es importante señalar que este ajuste aparente del modelo se debe a la alta tasa de aciertos en la clase negativa del conjunto de datos. Por lo tanto, el desempeño de estas configuraciones debe ser evaluado por sus métricas específicas por clase, ya que las curvas de entrenamiento pueden proporcionar información errónea sobre las capacidades reales de los modelos.

En las secciones siguientes se le nombrará modelo o configuración base a la configuración experimental 2 de la presente sección, que corresponde a la configuración que fue probada con imágenes de puentes con balance de datos.

4.3 Fase 1, exploración: Optimización de hiperparámetros con propuesta metodológica

En esta sección se presentan los resultados obtenidos del proceso de optimización de hiperparámetros para modelos de clasificación a través de un AG clásico. El flujo del AG clásico se

muestran en la Figura 4.2. Los individuos del AG representarán distintas configuraciones de hiperparámetros, de manera que se puede generar un modelo de clasificación distinto para cada combinación distinta de hiperparámetros.

Las especificaciones del AG clásico utilizado se describen en la Tabla 4.3.

Tabla 4.3: Operadores y parámetros de AG.

Parámetro	**Valor**
Tamaño de población	20
Cantidad de generaciones	50
Inicialización de la población	Aleatoria
Medida de aptitud	Puntuación $F1$
Método de selección	Torneo binario
Tipo de cruza	Cruza de un punto
Tasa de cruza	1.0
Tipo de mutación	Variación aleatoria de gen
Tasa de mutación	0.1
Política de reemplazo	Mantener los 10 hijos generados en la generación actual y mantener los 10 mejores padres para la siguiente generación

El AG generó diversos modelos de clasificación con distintas configuraciones de hiperparámetros, permitiendo así obtener desempeños equiparables y superiores la red neuronal clasificadora base de *VGG-16*. Se realizaron tres ejecuciones distintas del AG, obteniendo tres modelos que fueron los mejores en sus respectivas poblaciones.

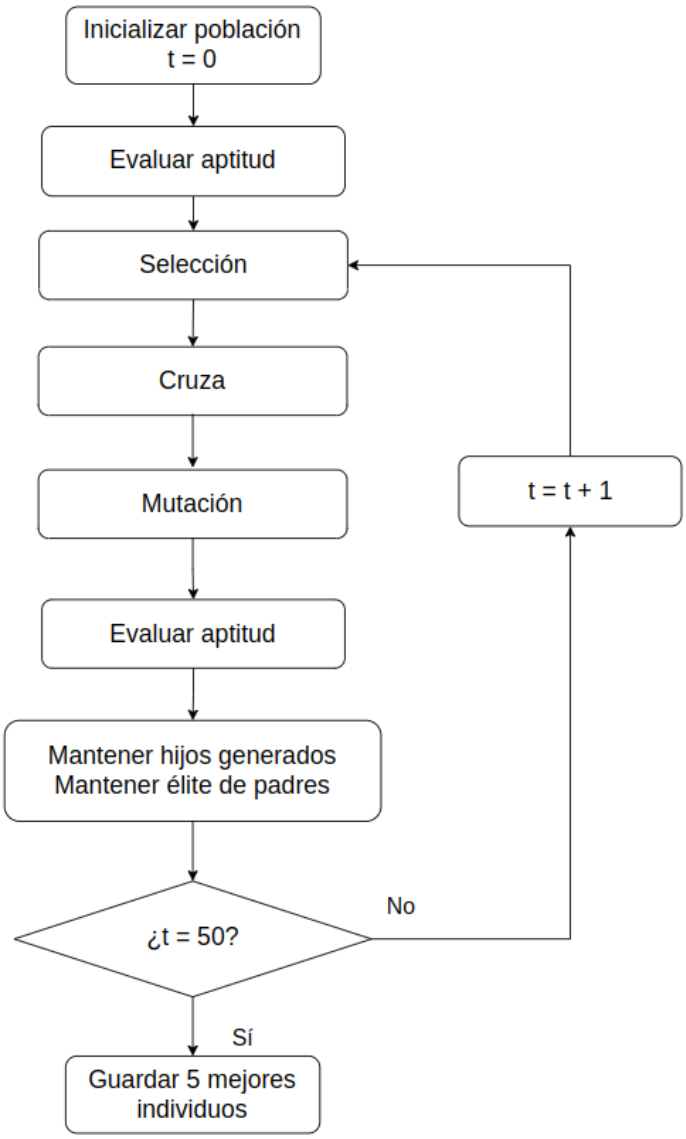


Figura 4.2: Diagrama de flujo AG clásico.

Los modelos generados durante la ejecución del AG fueron entrenados con las siguientes especificaciones no variables:

- Épocas: 200.
- Función de pérdida: Entropía cruzada binaria.

Para el entrenamiento se utilizó un conjunto de imágenes de puentes con clases balanceadas con $2,000$ instancias por cada clase.

Las distintas configuraciones de hiperparámetros obtenidas por los mejores individuos del AG se presentan en la Tabla 4.4. El mejor modelo generado por la ejecución 1, es llamado **Modelo A**, el mejor modelo generado por la ejecución 2, es nombrado **Modelo B**, y por último el mejor modelo generado por la ejecución 3 es el **Modelo C**.

Tabla 4.4: Configuraciones de hiperparámetros de individuos con la más alta aptitud generados por AG.

Gen	**Hiperparámetro**	**Modelo A**	**Modelo B**	**Modelo C**
1	Inicialización de parámetros	LeCun Normal	Glorot Normal	LeCun Normal
2	Cantidad de neuronas en primera capa oculta	1024	512	4096
3	Cantidad de neuronas en segunda capa oculta	1024	1024	1024
4	Función de activación	Sigmoide	*SeLU*	*ReLU*
5	Tasa de difusión	0	0.5	0.7
6	Tasa de aprendizaje	0.001	0.001	0.001
7	Tamaño de lotes	16	32	16
8	Optimizador	AdaDelta	AdaDelta	SGD

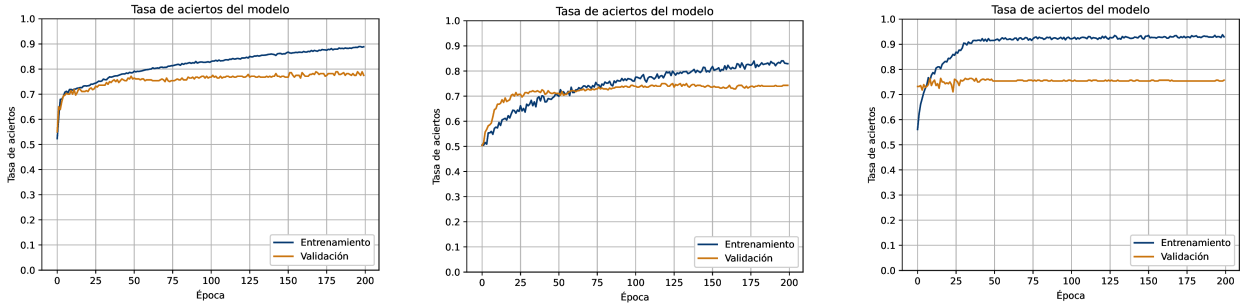
En la Tabla 4.5 se muestran los resultados de los modelos con la más alta Puntuación $F1$ de cada una de las tres ejecuciones, además de incluír el modelo clasificador *VGG-16* base, mismo de la configuración experimental 2 presentado en la Subsección 4.2.

Tabla 4.5: Métricas de evaluación para modelos generados con AG clásico con aptitud más alta a través de distintas ejecuciones.

Clase	**Modelo A**			**Modelo B**			**Modelo C**			***VGG-16* base**		
	Prec	***Rec***	***F1***	***Prec***	***Rec***	***F1***	***Prec***	***Rec***	***F1***	***Prec***	***Rec***	***F1***
Grieta	0.82	0.74	0.78	0.84	0.74	0.79	0.82	0.75	0.78	0.81	0.71	0.76
No grieta	0.77	0.84	0.80	0.76	0.85	0.80	0.75	0.82	0.78	0.72	0.82	0.77

En la Figura 4.3 se presentan las curvas de entrenamiento correspondientes a los modelos con la puntuación $F1$ más alta generados por las distintas ejecuciones del AG.

De acuerdo a los resultados de la Tabla 4.5, el modelo base de *VGG-16* no tiene el puntaje más alto en ninguna de las métricas de evaluación consideradas para la experimentación. Se puede concluir que el AG pudo mejorar el desempeño del caso base.



(a) Modelo A. (b) Modelo B. (c) Modelo C.

Figura 4.3: Curvas de entrenamiento de modelos generados por AG con aptitud más alta de cada ejecución.

En la Tabla 4.5 se puede apreciar que el **Modelo B** tiene las mejores puntuaciones $F1$ de todos los modelos, tanto para la clase positiva como la negativa. Estas puntuaciones del Modelo B indican que tuvo el mejor desempeño general con respecto a los demás. También, en la Figura 4.3 (b), que es la correspondiente al **Modelo B**, es claro que se tiene un mejor ajuste de entrenamiento, debido a que la diferencia entre la tasa de aciertos para los conjuntos de entrenamiento y validación en las últimas épocas se encuentra al rededor de 0.1, en contraste

con la Figura 4.3 (a) y (c) que presentan un mayor sobreajuste, presentando diferencias entre 0.15 y 0.2.

Otro aspecto relevante a tomar en cuenta en estos experimentos, son las mejoras obtenidas en términos de tiempo de procesamiento y recursos computacionales utilizados. Como se señaló en la Tabla 4.4, los modelos generados en los AGs tienen sustancialmente menos neuronas que el modelo base *VGG-16*, esto implica una reducción significativa de cantidad de parámetros entrenables. La reducción de parámetros impacta directamente al entrenamiento de los modelos, reduciendo el tiempo que es necesario para entrenarlos, y reduciendo también los recursos necesarios para ello. Sin embargo, es importante señalar que el **Modelo C** no tuvo mejoras en cuanto al tiempo de entrenamiento. En la tabla 4.6 se puede observar la comparativa de cantidad de parámetros y tiempo de entrenamiento.

Tabla 4.6: Comparación de modelos generados por AG y *VGG-16* base, respecto a cantidad de parámetros y tiempo de entrenamiento.

Modelo	**Cantidad de parámetros**	**Tiempo de entrenamiento (s)**
Modelo A	24,742,784	188
Modelo B	**13,372,930**	**94**
Modelo C	106,961,922	960
VGG-16 base	119,554,050	400

Dados los datos anteriores, podemos concluir que el **Modelo B** es el mejor modelo generado en las ejecuciones del AG. Esto debido a que presenta mejores resultados generales en sus métricas de puntuación $F1$, se visualiza también un mejor ajuste a los datos del entrenamiento, sugiriendo así una mejor capacidad de generalizar el conocimiento adquirido durante el entrenamiento.

Por último, el **Modelo B** fue también el modelo más simple en términos de cantidad de parámetros. Mediante las estrategias de exploración y explotación del algoritmo genético, se pudo descubrir un modelo más simple, que se entrena con mayor rapidez y presenta mejoras leves en las métricas en comparación con el clasificador base de la arquitectura *VGG-16*. El **Modelo B** será utilizado en las siguientes secciones para el desarrollo de nuevos modelos.

4.4 Fase 2, explotación: Optimización de modelos obtenidos en la Fase 1

La optimización de hiperparámetros mediante un algoritmo genético en la Fase 1 permitió la creación de un modelo de clasificación que mejora el rendimiento y las características del modelo base *VGG-16*. Este modelo, conocido como **Modelo B** en la Sección 4.3, se utiliza en la Fase 2 de la experimentación. En esta sección, se detalla el proceso para desarrollar nuevos modelos de clasificación, que incluye el ajuste fino del extractor de características de *VGG-16* y la implementación de un algoritmo genético para generar clasificadores adicionales.

En la Figura 4.4 se muestran los dos flujos seguidos para la generación de modelos de clasificación. Ambos flujos están compuestos por cuatro modelos distintos, cada modelo consiste en un extractor de características tomado de la arquitectura *VGG-16*, y una RNA como clasificador. Los bloques convolucionales de cada modelo pueden ser utilizados en modo extractor de características (EC) o en modo ajuste fino (AF).

Además, en los modelos finales de cada flujo, M4 y M8, se propone el uso de agrupación global por promedio con el objetivo de simplificar las características de entrada al bloque clasificador, así reduciendo la cantidad de parámetros del modelo y por consiguiente disminuyendo la complejidad computacional y temporal.

La diferencia principal entre los flujos, reside en que el Flujo 1 toma como punto de partida un modelo (M1) cuyo clasificador es el clasificador base de la arquitectura *VGG-16*, mientras que el primer modelo del Flujo 2 (M5) es un clasificador encontrado por un AG en la Fase 1 de la experimentación: el **Modelo B**. Utilizando ambos flujos, es posible comparar el desempeño obtenido en los modelos finales, contrastando así las métricas de un modelo generado que parte del clasificador base de *VGG-16* y otro modelo que tiene como punto de partida un clasificador generado por un AG.

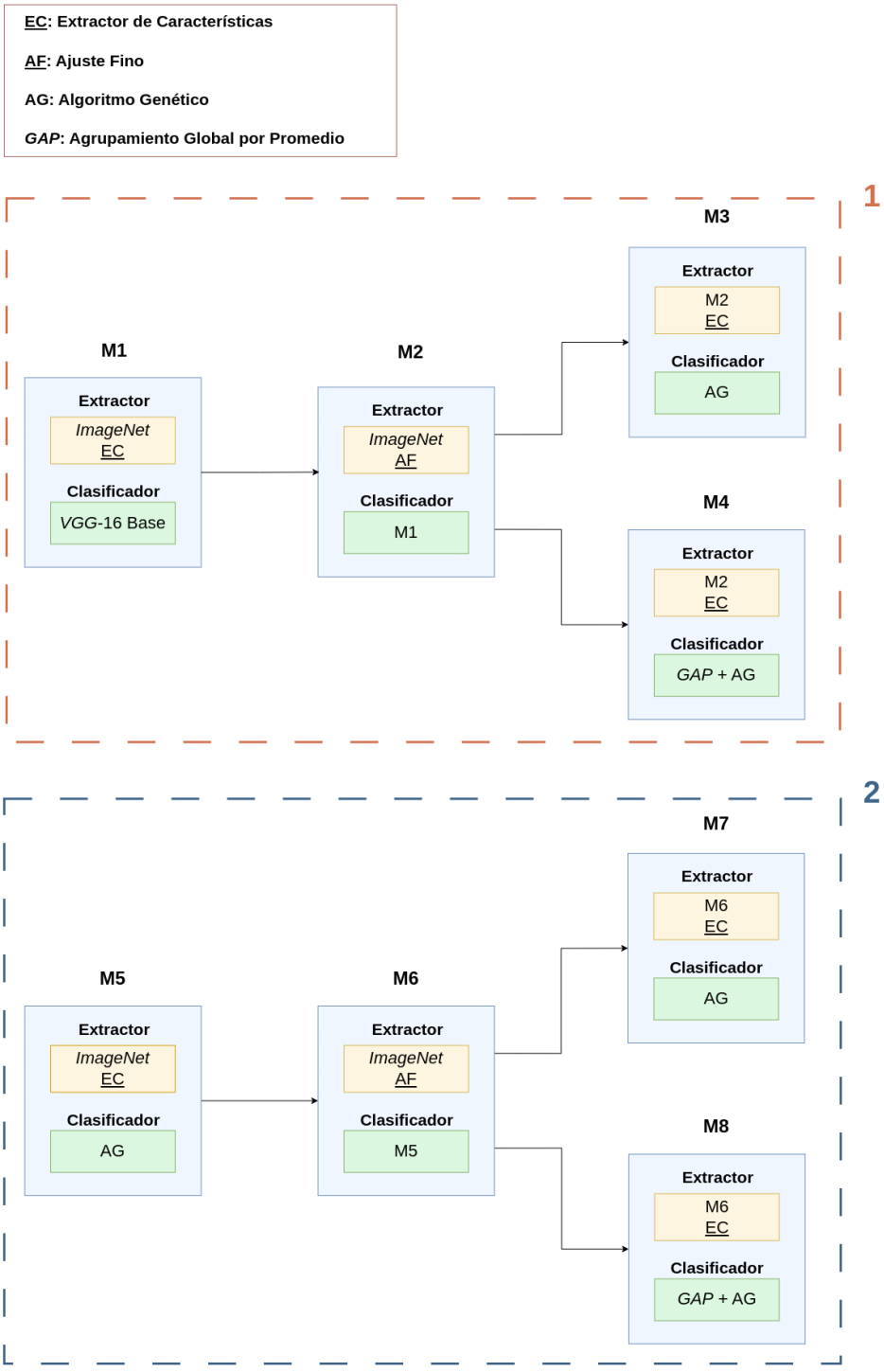


Figura 4.4: Diagrama a bloques de modelos generados. Flujo 1: Generación de modelos partiendo de caso base (naranja), y Flujo 2: Generación de modelos partiendo de Modelo B (azul).

4.4.1 Flujo 1

El primer modelo del Flujo 1, M1, está compuesto por un extractor de características preentrenado con el conjunto de datos *ImageNet* en modo extractor de características, y el clasificador es igual a la RNA del modelo base *VGG-16*. M1 es considerado el caso base de la experimentación, es decir, M1 es equivalente a la arquitectura original *VGG-16*.

En el siguiente modelo, M2, se mantiene el clasificador de M1, pero el extractor ahora es configurado en modo ajuste fino partiendo de los pesos de *ImageNet*. El ajuste fino en M2 se realiza entrenando todas las capas convolucionales del extractor, esto permitirá a los bloques convolucionales mejorar su capacidad de extracción de características para imágenes de superficies de concreto con grietas. Es por ello que M2 presenta una mejora en sus métricas con respecto a M1.

Posterior a la generación de M2, se tiene ahora un extractor de características mejorado por ajuste fino, este mismo extractor se utiliza en modo extractor de características para generar M3 y M4. Estos dos modelos siguientes además de contar con un extractor finamente ajustado, utilizan clasificadores encontrados por AG. En particular, el clasificador de M4 es obtenido a través de un AG con la aplicación de agrupación global por promedio para sus características de entrada. Es importante notar que los entrenamientos para obtener M3 y M4 son distintos debido a la simplificación de características de entrada en M4.

En la Tabla 4.7 se describen las configuraciones de hiperparámetros de los clasificadores correspondientes a los modelos del Flujo 1. A pesar de que M2 utiliza la misma configuración de clasificador que M1, es importante señalar que para aumentar la efectividad del ajuste fino, se optó por utilizar una tasa de aprendizaje y un tamaño de lotes menor a los usados en M1, estos cambios son necesarios para poder hacer ajustes menores y más precisos en los pesos de las capas convolucionales de M2, y así no perder las representaciones útiles ya aprendidas por el extractor de *VGG-16* con el conjunto de datos *ImageNet*.

Tabla 4.7: Configuraciones de hiperparámetros de clasificadores para modelos en Flujo 1.

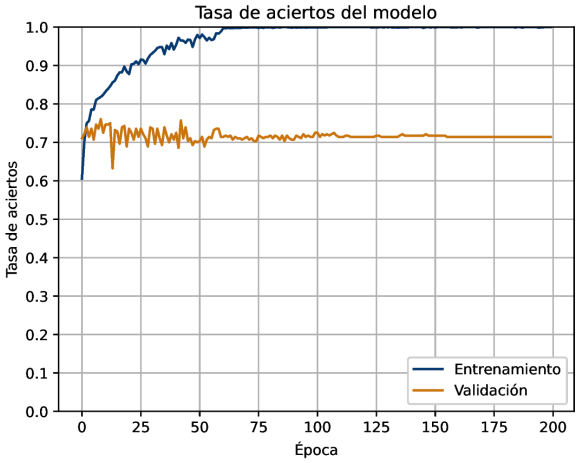
Gen	**Hiperparámetro**	**M1**	**M2**	**M3**	**M4**
1	Inicialización de parámetros	Glorot Normal	Glorot Normal	Glorot Normal	LeCun Uniform
2	Cantidad de neuronas en primera capa oculta	4096	4096	512	1024
3	Cantidad de neuronas en segunda capa oculta	4096	4096	1024	512
4	Función de activación	*ReLU*	*ReLU*	*SeLU*	*ReLU*
5	Tasa de difusión	0.5	0.5	0.5	0.6
6	Tasa de aprendizaje	0.001	0.0001	0.001	0.0001
7	Tamaño de lotes	32	8	32	64
8	Optimizador	*SGD*	*SGD*	*AdaDelta*	*RMSProp*

En la Tabla 4.8 se presentan los valores obtenidos en las métricas de evaluación para el Flujo 1.

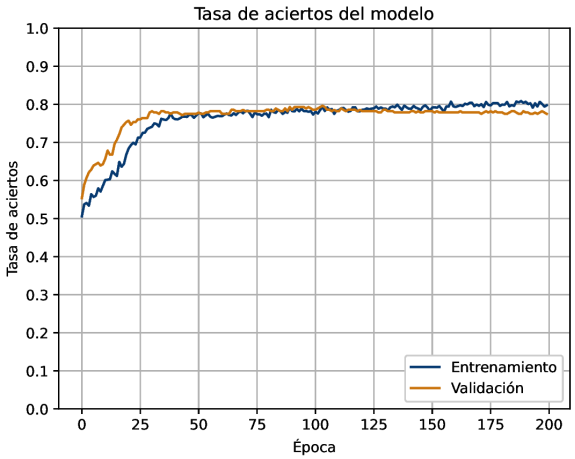
Tabla 4.8: Métricas de evaluación para modelos generados en Flujo 1 de Fase 2.

Clase	**M1**			**M2**			**M3**			**M4**		
	Prec	***Rec***	***F1***	***Prec***	***Rec***	***F1***	***Prec***	***Rec***	***F1***	***Prec***	***Rec***	***F1***
Grieta	0.81	0.71	0.76	0.85	0.71	0.77	0.87	0.77	0.82	0.83	0.84	0.83
No grieta	0.72	0.82	0.77	0.74	0.87	0.80	0.78	0.87	0.83	0.82	0.80	0.81

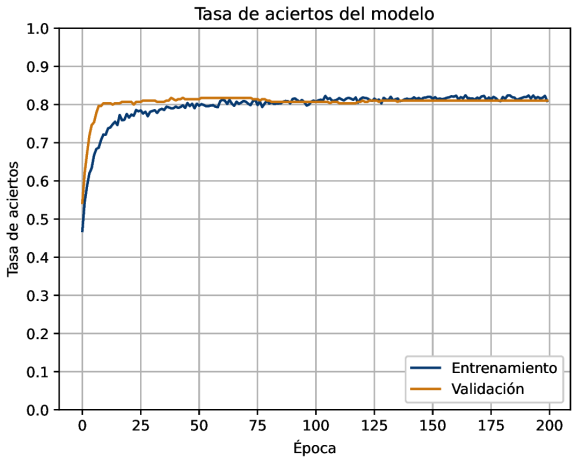
En la Figura 4.5 se pueden apreciar las curvas de entrenamiento de los modelos correspondientes al Flujo 1.



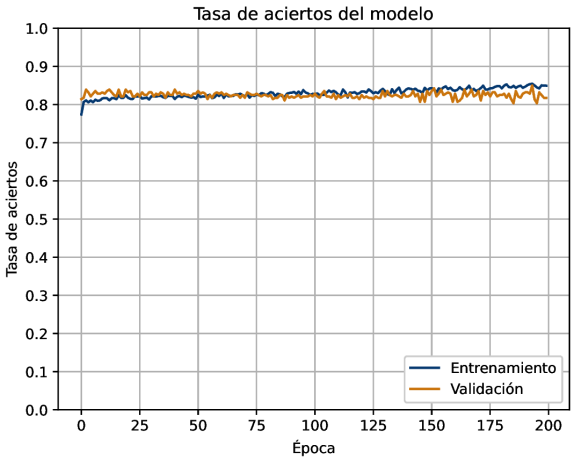
(a) M1.



(b) M2.



(c) M3.



(d) M4.

Figura 4.5: Curvas de entrenamiento de modelos en el Flujo 1 de la Fase 2.

De acuerdo con las métricas de desempeño y las curvas de aprendizaje obtenidas, se puede apreciar como el modelo base, M1, presenta un sobreajuste donde la curva de validación y entrenamiento tienen una diferencia cercana a 0.3 en tasa de aciertos en su punto de convergencia, esto puede deberse a la gran cantidad de parámetros del clasificador. Además, la puntuación $F1$ de la clase grieta de M1 fue la menor de los cuatro modelos con 0.76.

A partir del modelo M2 se comienza a ver una mejoría notable en cuanto al ajuste de los datos, ya que existe un sobreajuste mínimo en las curvas de entrenamiento, y la puntuación

$F1$ aumentó a 0.77. Cabe señalar que esto fue gracias a que también mejoró su capacidad para detectar imágenes sin grietas y aumentó la precisión del modelo. M2 cuenta con una mejor capacidad predictiva debido a el ajuste fino en el extractor de características, esto permite un mejor ajuste de datos y un aumento en las puntuaciones de sus métricas.

Los modelos M3 y M4, cuyos clasificadores fueron encontrados mediante un algoritmo genético (AG), mostraron ligeras mejoras en sus curvas de entrenamiento y aumentos significativos en las métricas en comparación con M2 y M1. M3 y M4 lograron puntuaciones F1 de 0.82 y 0.83, respectivamente. Además, las curvas de aprendizaje de M3 y M4 presentan un sobreajuste prácticamente nulo, gracias al ajuste fino del extractor en M2 y al uso de clasificadores eficientes proporcionados por el AG.

Un aspecto importante a destacar es que M4 es el modelo más equilibrado en términos de precisión y recuperación, con una precisión de 0.83 y una recuperación de 0.84, mostrando una diferencia de solo 0.01 entre estas métricas. En contraste, otros modelos presentan diferencias de hasta 0.10 entre precisión y recuperación. Este equilibrio sugiere que M4 tiene una mejor capacidad para distinguir entre instancias de distintas clases en comparación con los otros modelos.

En conclusión, M4 es el mejor modelo del Flujo 1, lo que indica que el ajuste fino combinado con un AG para explorar diferentes configuraciones de hiperparámetros y la aplicación de agrupación global por promedio en las características resultan en modelos superiores a los basados únicamente en *VGG-16*.

4.4.2 Flujo 2

El primer modelo del Flujo 2, M5, está compuesto por un extractor de características preentrenado con el conjunto de datos *ImageNet* en modo extractor de características, y el clasificador corresponde al modelo generado en la Fase 1, denominado **Modelo B** en la Sección 4.3. El modelo M5 es el punto de partida de este flujo, es análogo al modelo M1 en el Flujo 1, con la diferencia de que M5 no parte del modelo de RNA base de la arquitectura original *VGG-16*.

En el siguiente modelo, M6, se mantiene el clasificador de M5, pero el extractor ahora es configurado en modo ajuste fino partiendo de los pesos de *ImageNet*. El ajuste fino en M6 se realiza entrenando todas las capas convolucionales del extractor, esto permitirá a los

bloques convolucionales mejorar su capacidad de extracción de características para imágenes de superficies de concreto con grietas, al igual que se hizo en M2 en el Flujo 1.

Una vez generado el modelo M6, se tiene ahora un extractor de características mejorado por ajuste fino, este mismo extractor se utiliza en modo extractor de características para generar los modelos M7 y M8. Estos dos modelos siguientes además de contar con un extractor finamente ajustado, utilizan clasificadores encontrados por AG. En particular, el clasificador del modelo M8 es obtenido a través de un AG con la aplicación de agrupación global por promedio para sus características de entrada. Es importante notar que los entrenamientos para obtener los modelos M7 y M8 son distintos debido a la simplificación de características de entrada en M8.

En la Tabla 4.10 se presentan las configuraciones de hiperparámetros de los clasificadores correspondientes a los modelos del Flujo 2. A pesar de que el modelo M6 utiliza la misma configuración de clasificador que el modelo M5, es importante señalar que para aumentar la efectividad del ajuste fino, se optó por utilizar una tasa de aprendizaje y un tamaño de lotes menor a los usados en M5, estos cambios son necesarios para poder hacer ajustes menores y más precisos en los pesos de las capas convolucionales de M6, y así no perder las representaciones útiles ya aprendidas por el extractor de *VGG-16* con el conjunto de datos *ImageNet*.

En la Tabla 4.9 se presentan los valores obtenidos en las métricas de evaluación para el Flujo 2.

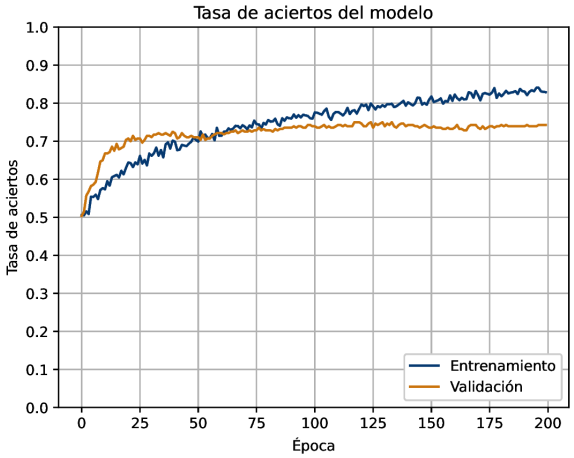
Tabla 4.9: Métricas de evaluación para modelos generados en Flujo 2 de Fase 2.

Clase	**M5**			**M6**			**M7**			**M8**		
	Prec	***Rec***	***F1***	***Prec***	***Rec***	***F1***	***Prec***	***Rec***	***F1***	***Prec***	***Rec***	***F1***
Grieta	0.84	0.74	0.79	0.83	0.80	0.82	0.91	0.90	0.90	0.92	0.88	0.90
No grieta	0.76	0.85	0.80	0.80	0.83	0.81	0.89	0.90	0.90	0.89	0.92	0.90

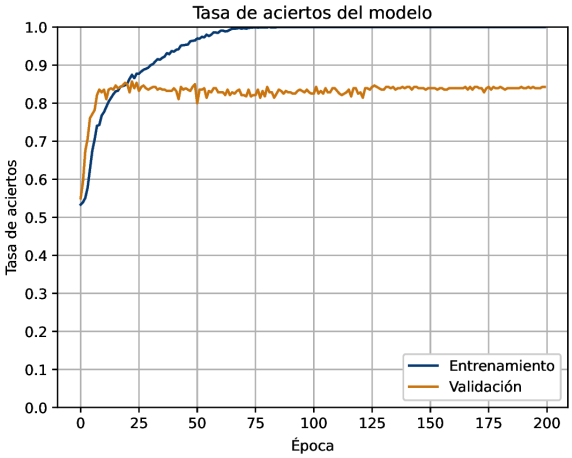
Tabla 4.10: Configuraciones de hiperparámetros de clasificadores para modelos en Flujo 1.

Gen	**Hiperparámetro**	**M5**	**M6**	**M7**	**M8**
1	Inicialización de parámetros	Glorot Normal	Glorot Normal	LeCun Normal	LeCun Uniform
2	Cantidad de neuronas en primera capa oculta	512	512	1024	2048
3	Cantidad de neuronas en segunda capa oculta	1024	1024	512	512
4	Función de activación	*SeLU*	*SeLU*	*SeLU*	*SeLU*
5	Tasa de difusión	0.5	0.5	0.5	0.5
6	Tasa de aprendizaje	0.001	0.0001	0.001	0.0001
7	Tamaño de lotes	32	8	32	64
8	Optimizador	*AdaDelta*	*AdaDelta*	*AdaDelta*	*RMSProp*

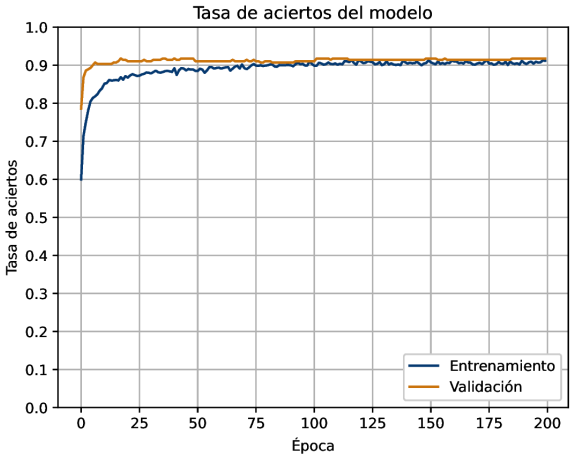
En la Figura 4.6 se pueden apreciar las curvas de entrenamiento de los modelos correspondientes al Flujo 2.



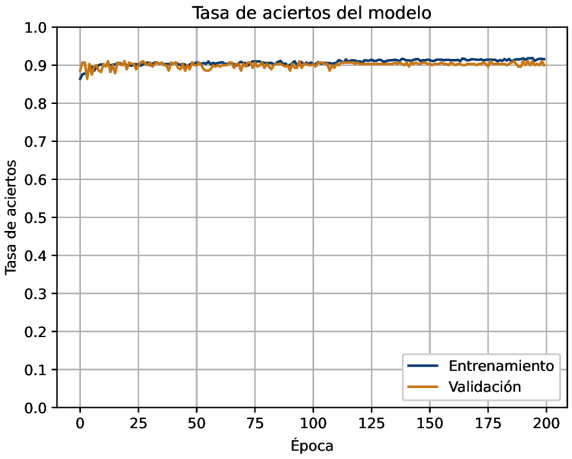
(a) M5.



(b) M6.



(c) M7.



(d) M8.

Figura 4.6: Curvas de entrenamiento de modelos en el Flujo 2 de la Fase 2.

Observando las métricas y curvas de aprendizaje obtenidas, se puede apreciar como el primer modelo del presente flujo, M5, presenta un sobreajuste no muy pronunciado, donde la curva de validación y entrenamiento tienen una diferencia cercana a 0.1 en tasa de aciertos en la última época de entrenamiento. Además, la puntuación $F1$ de la clase grieta del modelo M5 fue la menor de los cuatro modelos con 0.79.

A partir del modelo M6 se puede notar un aumento en las métricas de evaluación, la puntuación $F1$ aumentó a 0.82. Sin embargo, el modelo tiene un sobreajuste más marcado en comparación con M5, pero no tan grave como el de M1.

Los modelos cuyos clasificadores fueron encontrados por AG, que son M7 y M8, mostraron mejoras notables en cuanto a sus curvas de entrenamiento y aumentos sustanciales en cuanto a métricas con respecto a los modelos M6 y M5. M7 y M8 presentaron unas puntuaciones $F1$ de 0.90. Además, en las curvas de aprendizaje de ambos modelos muestran un sobreajuste prácticamente nulo, lo cual se debe al ajuste fino del extractor realizado a partir del modelo M6 y al uso de un clasificador adecuado y eficiente proporcionado por el algoritmo genético (AG). Un punto importante a notar, es que a diferencia del Flujo 1, la precisión y recuperación de los modelos fue aumentando de manera equilibrada a partir del ajuste fino del modelo M5. En conclusión, el modelo M8 es el mejor modelo del Flujo 2, a pesar de que su puntuación $F1$ es igual a la de M7, M8 cuenta con muchos menos parámetros debido a la agrupación global por promedio aplicada para reducir sus características de entrada, lo que resulta en una menor complejidad y un tiempo de entrenamiento más corto para el modelo.

4.5 Discusión de resultados: Flujo 1 y Flujo 2

Los modelos finales de ambos flujos, M4 y M8, de la Fase 2 de la experimentación presentan resultados muy diferentes, esto se debe al modelo utilizado para realizar ajuste fino de la RNC en cada flujo.

Un punto de contraste importante, es el desempeño obtenido por los modelos M2 y M6, ambos generados con ajuste fino. En la Tabla 4.11 se presentan los resultados en las métricas de estos dos modelos. Es relevante destacar que M2 y M6 son los modelos que se generan a partir de ajuste fino del extractor de características de la RNC, pero con diferentes clasificadores: M2 utiliza el clasificador base de *VGG-16*, mientras que M6 emplea un clasificador optimizado mediante un AG.

Tabla 4.11: Métricas de evaluación para M2 y M6.

Clase	M2			M6		
	Prec	***Rec***	***F1***	***Prec***	***Rec***	***F1***
Grieta	0.85	0.71	0.77	0.83	0.80	0.82
No grieta	0.74	0.87	0.80	0.80	0.83	0.81

En principio, era de esperar que el modelo M6 mostrara una mejora respecto al modelo M2, dado que el clasificador de M6 había tenido un desempeño superior al de M2 en etapas anteriores. La puntuación $F1$ de M6 supera en 0.05 a la de M2 en la clase positiva, gracias a una mejora en la recuperación de 0.09. Aunque la mejora en la puntuación $F1$ no es muy pronunciada, M6 representa un avance significativo en las métricas para los modelos derivados de él.

Este efecto se hace más evidente en los mejores modelos de cada flujo, es decir M4 y M8. En la Tabla 4.12 se presentan las métricas de dichos modelos para poder comparar el impacto del ajuste fino ejecutado con un AG.

Tabla 4.12: Métricas de evaluación para M4 y M8.

Clase	M4			M8		
	Prec	***Rec***	***F1***	***Prec***	***Rec***	***F1***
Grieta	0.83	0.84	0.83	0.92	0.88	0.90
No grieta	0.82	0.80	0.81	0.89	0.92	0.90

Las puntuación $F1$ del modelo M8 es mayor que la del modelo M4 por 0.07, esto es una diferencia significativa tomando en cuenta que se trata de los mejores modelos generados en cada uno de los flujos experimentales, mostrando que el modelo M8 fue superior al final del flujo.

Además, una comparación importante dentro de la experimentación es el desempeño del mejor modelo generado, M8, con el modelo base del proyecto, M1. De esta forma se hace evidente la gran mejoría que se tuvo al aplicar la metodología de optimización de hiperparámetros con AG. Los resultados se presentan en la Tabla 4.13.

Tabla 4.13: Métricas de evaluación para M1 y M8.

Clase	M1			M8		
	Prec	***Rec***	***F1***	***Prec***	***Rec***	***F1***
Grieta	0.81	0.71	0.76	0.92	0.88	0.90
No grieta	0.72	0.82	0.77	0.89	0.92	0.90

Se puede observar como la puntuación $F1$ en la clase positiva de M8 es superior por 0.24 con respecto a la de M1, esto indica que el modelo tiene una mejor capacidad de detección de grietas que M1, esto indica que aplicando esta metodología de optimización de hiperparámetros con AG habilita a la comunidad científica para poder generar modelos con buen desempeño para detección de grietas en superficies de puentes de concreto reforzado.

4.6 Prueba de modelo con imágenes de puente Hidalgo

Para validar la capacidad de detección de grietas del modelo M8, se emplearon imágenes diferentes a las del conjunto de datos *SDNET2018*. Esto permitió verificar que el modelo puede identificar grietas en concreto sin importar las variaciones en el entorno, como texturas, colores, manchas o iluminación. Las imágenes empleadas provienen de las vigas inferiores del puente Miguel Hidalgo, situado en la ciudad de Culiacán, Sinaloa.

Figura 4.7: Imagen de muestra: Superficie de concreto con grieta en puente Miguel Hidalgo.

Para el procesamiento se tomaron regiones de interés de 224 × 224 pixeles de las imágenes originales para ingresarlas al modelo, ya que 224 × 224 es la resolución de entrada para la RNC. Algunas imágenes de regiones de interés se presentan en la Figura 4.8.

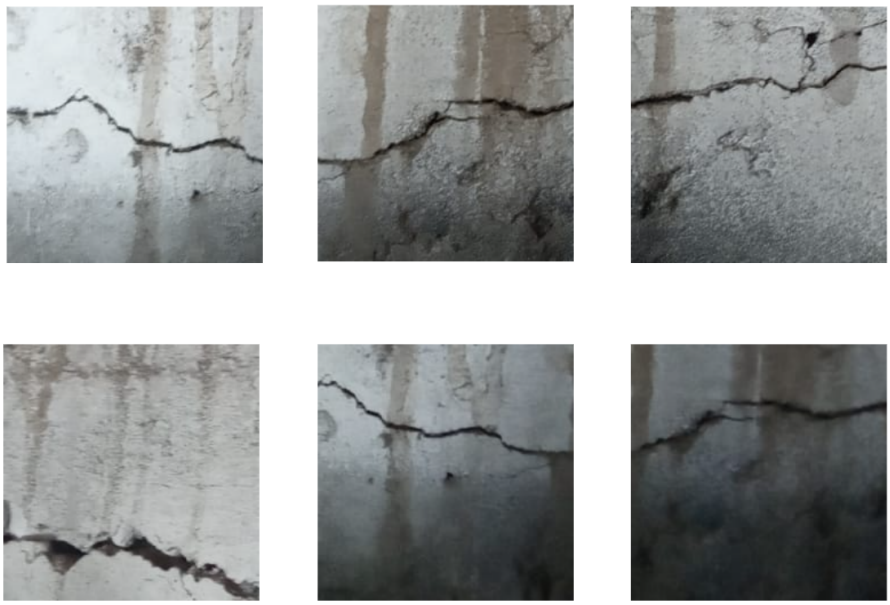


Figura 4.8: Regiones de interés con grieta en puente Miguel Hidalgo.

Una vez obtenidas las predicciones se calculan sus métricas de evaluación, las cuales se presentan en la Tabla 4.14.

Tabla 4.14: Métricas de evaluación para M8 usando imágenes del puente Miguel Hidalgo como imágenes de prueba.

Clase	**M8**		
	Prec	***Rec***	***F1***
Grieta	0.91	0.87	0.89
No grieta	0.86	0.90	0.88

El modelo M8 demostró ser fiable para detectar grietas a pesar de usar un conjunto de datos distinto al utilizado en el entrenamiento. En la prueba ejecutada, el modelo obtuvo una puntuación $F1$ de 0.89, lo que es esperado dado su desempeño en la sección anterior.

Capítulo 5

Conclusiones

En el presente trabajo de tesis se propuso una metodología de optimización de hiperparámetros para RNA basada en técnicas metaheurísticas de cómputo evolutivo, más concretamente AG. La metodología propuesta fue aplicada al caso de estudio de detección de grietas en puentes de concreto reforzado a partir de imágenes digitales. Para esto, se utilizó el conjunto de datos *SDNET2018* el cual se encuentra disponible en internet. La detección de grietas es un problema complejo ya que no hay una forma ni tamaño predeterminados para estas, por lo que es necesario el uso de técnicas de aprendizaje profundo como la arquitectura (RNC) *VGG-16*.

La metodología presentada en este trabajo aborda el problema de la poca (o nula) optimización que se suele aplicar a los hiperparámetros estructurales y de entrenamiento en modelos de clasificación basados en RNA. Dado que los hiperparámetros impactan en gran medida el desempeño de dichos modelos, es crucial optimizarlos mediante alguna metodología computacional. Además, el espacio de búsqueda de configuraciones para hiperparámetros en este estudio asciende a 122,880 posibilidades. Este problema es tratado utilizando un AG para encontrar distintas configuraciones de hiperparámetros, generando así múltiples modelos y logrando buenas soluciones para obtener los mejores resultados posibles en las métricas de evaluación de dichos modelos.

El aporte principal de este estudio es el diseño de dos etapas de experimentación: una etapa de exploración, donde se busca una configuración adecuada de hiperparámetros mediante AG para un modelo de clasificación, con el objetivo de mejorar el rendimiento del modelo original *VGG-16*; y una etapa de explotación, en la que, partiendo del modelo optimizado en la primera

etapa, se realiza un ajuste fino de la RNC para mejorar el extractor de características, y luego se vuelve a utilizar un algoritmo genético (AG) para encontrar un nuevo modelo que tenga un mejor desempeño que el del modelo de la etapa de exploración.

Al ejecutar la etapa de exploración, se encontró mediante AG una configuración de hiperparámetros, a partir de la cual se generó un modelo que alcanzó una puntuación $F1$ de 0.79 en contraste con 0.76 del modelo *VGG-16* base. Además, el modelo encontrado mitigó el sobreajuste del modelo original. Desde este punto, es evidente que el uso de técnicas de búsqueda como AG supone una mejora en la puntuación $F1$ de los modelos.

En la fase de explotación, a partir del modelo encontrado en la etapa de exploración, se realizó un ajuste fino del extractor de características para mejorar el proceso. Con este ajuste fino, la RNC mejoró sustancialmente su capacidad para caracterízar imágenes, lo que también elevó el desempeño de los modelos clasificadores, mejorando la puntuación $F1$ hasta 0.82. Por último, se llevó a cabo otra búsqueda de hiperparámetros con AG utilizando el extractor de características ajustado finamente, obteniendo así un modelo con una puntuación $F1$ de 0.90, el cual fue el mejor de todos los generados durante la experimentación.

Para la validación del modelo final, se utilizaron imágenes distintas a las de *SDNET2018* como conjunto de prueba. Las imágenes utilizadas corresponden a una estructura civil local con agrietamiento en sus superficies: el puente Miguel Hidalgo, ubicado en la ciudad de Culiacán, Sinaloa. El mejor modelo generado obtuvo una puntuación $F1$ de 0.89 con estas nuevas imágenes, lo que significa que el modelo mantiene sus capacidades de detección de grietas independientemente del escenario, ya sea con variaciones estructurales, de textura, de iluminación, de color, entre otros factores.

Estas etapas experimentales demuestran cómo el uso de un algoritmo de búsqueda, como lo son los AG, ayuda a explorar de manera eficiente el espacio de búsqueda, permitiendo encontrar soluciones viables con buen desempeño en un tiempo razonable. Además, en lo que a hiperparámetros de RNA se refiere, el AG demostró encontrar combinaciones de hiperparámetros adecuadas para generar modelos que tuvieron mucho mejor desempeño, y menor complejidad en contraste con utilizar el modelo original de *VGG-16*.

Líneas de investigación futuras

Como trabajo a futuro, se propone realizar modificaciones en los hiperparámetros al proceso de ajuste fino del flujo de generación de modelos en la etapa de explotación de la metodología. Por ejemplo, se sugiere utilizar siempre *SGD* como optimizador y tamaño de lote menor o igual a ocho para realizar el ajuste fino. Estas modificaciones podrían mejorar aún más el extractor de características de la RNC, mejorando así el proceso de caracterización, permitiendo así la generación de modelos potencialmente mejores que los ya encontrados.

Además, se propone explorar variantes del AG donde se utilicen versiones distintas al AG clásico. Cambiar las estrategias evolutivas puede beneficiar a la búsqueda de nuevas soluciones no exploradas, permitiendo así generar nuevos y mejores modelos.

Bibliografía

Abdi, H., Valentin, D., & Edelman, B. (1999). *Neural networks*. Sage.

Alibrahim, H., & Ludwig, S. A. (2021). Hyperparameter optimization: Comparing genetic algorithm against grid search and bayesian optimization. *2021 IEEE Congress on Evolutionary Computation (CEC)*, 1551-1559.

Apicella, A., Donnarumma, F., Isgrò, F., & Prevete, R. (2021). A survey on modern trainable activation functions. *Neural Networks*, *138*, 14-32.

Aszemi, N. M., & Dominic, P. D. D. (2019). Hyperparameter Optimization in Convolutional Neural Network using Genetic Algorithms. *International Journal of Advanced Computer Science and Applications*. https://api.semanticscholar.org/CorpusID:197679951

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. En *Neural Networks: Tricks of the Trade: Second Edition* (pp. 437-478). Springer.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research*, *13*(2).

Cha, Y.-J., Choi, W., & Büyüköztürk, O. (2017). Deep learning-based crack damage detection using convolutional neural networks. *Computer-Aided Civil and Infrastructure Engineering*, *32*(5), 361-378.

Chaiyasarn, K., Buatik, A., & Likitlersuang, S. (2021). Concrete crack detection and 3D mapping by integrated convolutional neural networks architecture. *Advances in Structural Engineering*, *24*(7), 1480-1494.

Chung, H., & Shin, K.-s. (2020). Genetic algorithm-optimized multi-channel convolutional neural network for stock market prediction. *Neural Computing and Applications*, *32*, 7897-7914.

Desai, C. (2020). Comparative analysis of optimizers in deep neural networks. *International Journal of Innovative Science and Research Technology*, *5*(10), 959-962.

Dorafshan, S., Thomas, R., & Maguire, M. (2018). SDNET2018: An annotated image dataset for non-contact concrete crack detection using deep convolutional neural networks. Data Brief 21, 1664–1668 (2018).

Dorafshan, S., & Maguire, M. (2017). Autonomous detection of concrete cracks on bridge decks and fatigue cracks on steel members. *Proceedings of Digital Imaging*.

Dorafshan, S., Thomas, R. J., Coopmans, C., & Maguire, M. (2018). Deep learning neural networks for sUAS-assisted structural inspections: Feasibility and application. *2018 international conference on unmanned aircraft systems (ICUAS)*, 874-882.

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.

Escalona, U., Arce, F., Zamora, E., & Sossa, H. (2019). Fully convolutional networks for automatic pavement crack segmentation. *Computación y Sistemas*, *23*(2), 451-460.

Fujino, S., Mori, N., & Matsumoto, K. (2017). Deep convolutional networks for human sketches by means of the evolutionary deep learning. *2017 joint 17th world congress of international fuzzy systems association and 9th international conference on soft computing and intelligent systems (IFSA-SCIS)*, 1-5.

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.

Gibb, S., La, H. M., & Louis, S. (2018). A genetic algorithm for convolutional network structure optimization for concrete crack detection. *2018 IEEE congress on evolutionary computation (CEC)*, 1-8.

Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. En *Foundations of genetic algorithms* (pp. 69-93). Elsevier.

Hoffer, E., Hubara, I., & Soudry, D. (2017). Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *Advances in neural information processing systems*, *30*.

Hussain, A., Muhammad, Y. S., Nauman Sajid, M., Hussain, I., Mohamd Shoukry, A., Gani, S., y col. (2017). Genetic algorithm for traveling salesman problem with modified cycle crossover operator. *Computational intelligence and neuroscience, 2017*.

Hussain, M., Bird, J. J., & Faria, D. R. (2019). A study on CNN transfer learning for image classification. *Advances in Computational Intelligence Systems: Contributions Presented at the 18th UK Workshop on Computational Intelligence, September 5-7, 2018, Nottingham, UK*, 191-202.

Itano, F., de Sousa, M. A. d. A., & Del-Moral-Hernandez, E. (2018). Extending MLP ANN hyper-parameters Optimization by using Genetic Algorithm. *2018 International joint conference on neural networks (IJCNN)*, 1-8.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., y col. (2017). Population based training of neural networks. *arXiv preprint arXiv:1711.09846*.

Jebari, K., Madiafi, M., y col. (2013). Selection methods for genetic algorithms. *International Journal of Emerging Sciences*, *3*(4), 333-344.

Katoch, S., Chauhan, S. S., & Kumar, V. (2021). A review on genetic algorithm: past, present, and future. *Multimedia tools and applications*, *80*, 8091-8126.

Kaveh, M., & Mesgari, M. S. (2023). Application of Meta-Heuristic Algorithms for Training Neural Networks and Deep Learning Architectures: A Comprehensive Review. *Neural Processing Letters*, *55*, 4519-4622.

Khan, S. (2018). *A Guide to Convolutional Neural Networks for Computer Vision*. Morgan; Claypool.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, *1*(4), 541-551.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324.

Li, Z., Liu, F., Yang, W., Peng, S., & Zhou, J. (2021). A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, *33*(12), 6999-7019.

Lin, M., Chen, Q., & Yan, S. (2013). Network in network. *arXiv preprint arXiv:1312.4400*.

Liu, Y., Gao, W., Zhao, T., Wang, Z., & Wang, Z. (2023). A Rapid Bridge Crack Detection Method Based on Deep Learning. *Applied Sciences*, *13*(17), 9878.

Michalewicz, Z., & Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary computation*, *4*(1), 1-32.

Miguel A., L. (2017). *Schemes for Solving Large Scale Multi-objective Optimization Problems Using Evolutionary Algorithms* (Tesis doctoral). Centro de Investigacion y de Estudios Avanzados del Instituto Politecnico Nacional.

Mohan, A., & Poobal, S. (2018). Crack detection using image processing: A critical review and analysis. *alexandria engineering journal*, *57*(2), 787-798.

Munawar, H. S., Hammad, A. W., Haddad, A., Soares, C. A. P., & Waller, S. T. (2021). Image-based crack detection methods: A review. *Infrastructures*, *6*(8), 115.

Prasanna, P., Dana, K. J., Gucunski, N., Basily, B. B., La, H. M., Lim, R. S., & Parvardeh, H. (2014). Automated crack detection on concrete bridges. *IEEE Transactions on automation science and engineering*, *13*(2), 591-599.

Rattanavorragant, R., & Jewajinda, Y. (2019). A hyper-parameter optimization for deep neural network using an island-based genetic algorithm. *2019 16th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, 73-76.

Salman, M., Mathavan, S., Kamal, K., & Rahman, M. (2013). Pavement crack detection using the Gabor filter. *16th international IEEE conference on intelligent transportation systems (ITSC 2013)*, 2039-2044.

Sekhari, A., Sridharan, K., & Kale, S. (2021). Sgd: The role of implicit regularization, batch-size and multiple-epochs. *Advances In Neural Information Processing Systems*, *34*, 27422-27433.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, *25*.

Soon, G. K., Guan, T. T., On, C. K., Alfred, R., & Anthony, P. (2013). A comparison on the performance of crossover techniques in video game. *2013 IEEE international conference on control system, computing and engineering*, 493-498.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929-1958.

Talab, A. M. A., Huang, Z., Xi, F., & HaiMing, L. (2016). Detection crack in image using Otsu method and multiple filtering in image processing techniques. *Optik*, *127*(3), 1030-1033.

Wu, Y., Liu, L., Bae, J., Chow, K.-H., Iyengar, A., Pu, C., Wei, W., Yu, L., & Zhang, Q. (2019). Demystifying learning rate policies for high accuracy training of deep neural networks. *2019 IEEE International conference on big data (Big Data)*, 1971-1980.

Yan, T.-s. (2010). An improved genetic algorithm and its blending application with neural network. *2010 2nd International Workshop on Intelligent Systems and Applications*, 1-4.

Yang, L., & Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, *415*, 295-316.

Zhang, Q., Barri, K., Babanajad, S. K., & Alavi, A. H. (2021). Real-time detection of cracks on concrete bridge decks using deep learning in the frequency domain. *Engineering*, *7*(12), 1786-1796.

Zhao, T.-t., Zhao, J.-y., Zheng, R.-r., & Zhang, L.-l. (2010). Study on RMB number recognition based on genetic algorithm artificial neural network. *2010 3rd International Congress on Image and Signal Processing*, *4*, 1951-1955.

Zöller, M.-A., & Huber, M. F. (2021). Benchmark and survey of automated machine learning frameworks. *Journal of artificial intelligence research*, *70*, 409-472.