UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

OPTIMIZACIÓN DE ARQUITECTURAS DE REDES NEURONALES CONVOLUCIONALES PARA EL RECONOCIMIENTO DE PATRONES EN IMÁGENES MEDIANTE ALGORITMO GENÉTICO DE DOS NIVELES

TESIS PARA OPTAR AL GRADO DE MAGISTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA
MEMORIA PARA OPTAR AL TITULO DE INGENIERO CIVIL ELÉCTRICO

DANIEL ALBERTO MONTECINO MONTANARES

PROFESOR GUÍA:
CLAUDIO ANDRÉS PÉREZ FLORES

MIEMBROS DE LA COMISIÓN:
PABLO ESTEVEZ VALENCIA
DOMINGO MERY QUIROZ

Este trabajo ha sido parcialmente financiado por ANID (CONICYT) a través del proyecto FONDECYT 1191610, así como el Departamento de Ingeniería Eléctrica (DIE) y el Advanced Mining Technology Center (ANID Project AFB180004).

SANTIAGO DE CHILE
2021

RESUMEN DE LA MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO Y AL GRADO DE
MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA
POR: DANIEL ALBERTO MONTECINO MONTANARES
FECHA: 2021
PROF. GUÍA: CLAUDIO ANDRÉS PÉREZ FLORES

OPTIMIZACIÓN DE ARQUITECTURAS DE REDES NEURONALES CONVOLUCIONALES PARA EL RECONOCIMIENTO DE PATRONES EN IMÁGENES MEDIANTE ALGORITMO GENÉTICO DE DOS NIVELES

La Neuroevolución permite encontrar arquitecturas de redes neuronales convolucionales (CNN) automáticamente a través de algoritmos evolutivos. Sin embargo, el tiempo de búsqueda suele ser extenso, ya que se deben entrenar múltiples CNNs durante la evolución para calcular su *fitness*. Esto ha llevado a enfoques para acelerar este cálculo, generando un compromiso entre el tiempo de cómputo y la fidelidad del *fitness*. Para abordar el problema del compromiso, en este trabajo se propone un algoritmo genético de dos niveles (2LGA). Mientras que un nivel evalúa a todos los individuos rápidamente, el segundo evalúa más precisamente solo aquellos con los resultados mas promisorios. También se propone un espacio de búsqueda flexible y una codificación con genes no expresados para facilitar el crossover y la transmisión de información genética. El 2LGA se aplicó a siete datasets de clasificación de imágenes, cinco variantes de MNIST (MNIST-Variants), Fashion-MNIST y CIFAR-10, logrando resultados mejores que los publicados anteriormente. Los resultados muestran una mejora del 45 % en el dataset más complejo de MNIST, MRDBI, y una mejora promedio de 42.6 % en MNIST-Variants. Además, se mostró que el algoritmo igualó el desempeño de un algoritmo genético AG con entrenamientos precisos, pero tomando el tiempo de uno con entrenamientos rápidos.

Dedicado a Norma, mi querida abuela que en paz descance.

"It always seems impossible, until it is done."
–Nelson Mandela

Agradecimientos

Agradezco a todos quienes fueron parte de mi crecimiento académico, profesional, y personal. A mi familia y amigos por confiar en mí, y a quienes siempre se preocuparon por mi desarrollo. A Danitza, por siempre estar conmigo, apoyándome emocionalmente. A mi madre Elizabeth y mi padre Carlos por brindarme todas las herramientas que me han permitido crecer y alcanzar mis objetivos, pero por sobre todo, por brindarme los valores que definen la persona que soy.

Doy mis más sinceras gratitudes a mi profesor guía Claudio Pérez, quién me abrió las puertas para trabajar con él en el laboratorio de procesamiento de imágenes; quién me recibió como alumno tesista, me acompaño, y me apoyó a lo largo de todo el proceso, y quién me brindó su tiempo en incontables reuniones y revisiones.

Finalmente, gracias a mis compañeros y amigos, por hacer que el proceso universitario sea enriquecedor y gratificante más allá del ámbito académico.

Tabla de Contenido

1. Introducción **1**
1.1. Motivación . . . 1
1.2. Hipótesis . . . 4
1.3. Objetivo General . . . 4
1.4. Objetivos Específicos . . . 4
1.5. Contribuciones . . . 5

2. Marco Teórico **6**
2.1. Perceptrón . . . 6
2.2. Redes Neuronales . . . 7
2.3. Redes Neuronales Convolucionales . . . 7
2.4. Entrenamiento de Redes Neuronales . . . 9
2.5. Sobreajuste y Regularización . . . 10
2.6. Arquitecturas de CNN diseñadas a mano . . . 11
2.6.1. Red LeNet-5 . . . 11
2.6.2. Red AlexNet . . . 12
2.6.3. Red VGG . . . 12
2.6.4. Red GoogleNet . . . 13
2.6.5. Red ResNet . . . 13
2.6.6. Red DenseNet . . . 13
2.6.7. Red PyramidNet . . . 14
2.7. Algoritmos Genéticos . . . 15
2.7.1. Población Inicial . . . 15
2.7.2. Función de *Fitness* . . . 16
2.7.3. Selección . . . 17
2.7.4. *Crossover* . . . 17
2.7.5. Mutación . . . 18
2.7.6. Reemplazo . . . 18
2.7.7. Término del algoritmo . . . 19
2.7.8. Presión selectiva . . . 19
2.7.9. Hiper-parámetros importantes . . . 20
2.7.10. Selección de hiper-parámetros . . . 21
2.8. Espacio de Búsqueda de CNN . . . 21

3. Revisión Bibliográfica en Optimización de Arquitecturas de Redes Neuronales **24**

3.1. Métodos NAS 24
3.1.1. *Reinforcement Learning* 24
3.1.2. Gradiente Descendente 24
3.1.3. Optimización Bayesiana 25
3.2. Evolución de Redes Neuronales 25
3.3. Evolución de Redes Convolucionales 26
3.3.1. Aproximaciones del *Fitness* 26
3.3.2. Estado del Arte 27

4. Metodología **29**
4.1. Descripción general del algoritmo genético de dos niveles propuesto 29
4.1.1. Algoritmo Genético de Dos Niveles 30
4.1.2. Espacio de búsqueda 31
4.1.3. Codificación 34
4.1.4. Cálculo del *Fitness* 38
4.1.5. Implementación del método 39
4.2. Algoritmo genético aplicado al reconocimiento de patrones en imágenes . . . 41
4.2.1. Datasets 41
4.2.2. Competidores del estado del arte 43
4.3. Diseño de Experimentos 44
4.3.1. Configuración de parámetros 47

5. Resultados y Análisis de Experimentos **50**
5.1. Comparación con Algoritmo Genético Común 50
5.2. Comparación de Espacio de Búsqueda 52
5.3. Comparación con el estado del arte 53
5.3.1. Comparación en MNIST-V 53
5.3.2. Comparación en Fashion-MNIST 56
5.3.3. Comparación en CIFAR-10 57
5.4. Estudio de generalización 58
5.5. Análisis del ranking 59
5.6. Análisis 60

6. Conclusiones y Trabajo Futuro **62**
6.1. Conclusiones 62
6.2. Trabajo Futuro 63

Bibliografía **67**

A. Curvas de evolución **78**

Índice de Tablas

4.1. Resumen de la información codificada en el espacio de búsqueda propuesto. . 33
4.2. Descripción general de los conjuntos de datos utilizados para evaluar los métodos neuroevolutivos y al 2LGA. 42

5.1. Comparación de 2LGA con dos algoritmos genéticos ordinarios (O-GA). . . . 50
5.2. Resultados del test HSD de Tukey para el error de test de dos algoritmos genéticos comunes (AG-*short* y AG-*long*) y el 2LGA. 52
5.3. Resultados del test HSD de Tukey para el tiempo de evolución de dos algoritmos genéticos comunes (AG-*short* y AG-*long*) y el 2LGA. 52
5.4. Comparación de experimentos de RS con los competidores. Los competidores utilizan algoritmos evolutivos y espacios de búsqueda propios, mientras que RS utiliza el espacio de búsqueda propuesto. La desviación estándar de cada método se muestra entre paréntesis cuando está disponible. 52
5.5. Comparación de experimentos de RS con los algoritmos genéticos implementados. Todos los experimentos utilizan el espacio de búsqueda propuesto. . . 53
5.6. Comparación de los errores de test entre 2LGA y los mejores resultados publicados en MNIST-V (más bajo es mejor). La desviación estándar de cada método se muestra entre paréntesis cuando está disponible. 54
5.7. Comparación del test error del 2LGA con los resultados publicados anteriormente en Fashion-MNIST. La desviación estándar de cada método se muestra entre paréntesis cuando está disponible. 57
5.8. Comparación del test-error (%) entre el 2LGA y los mejores resultados publicados anteriormente en CIFAR-10 (más bajo es mejor). 58
5.9. Resultados del análisis de rankings. Cada valor es el promedio de todas las generaciones evaluadas y de 5 evoluciones independientes. 60

Índice de Ilustraciones

2.1. Representación gráfica de un perceptrón 6
2.2. Representación gráfica de una red neuronal. 7
2.3. Representación gráfica de operación convolución bidimensional 8
2.4. Efectos de un *learning rate* alto y bajo en el entrenamiento de una red neuronal 9
2.5. Ejemplo de sobreajuste en un problema de clasificación 10
2.6. Arquitectura de red LeNet-5 propuesta por LeCun 11
2.7. Arquitectura de AlexNet propuesta por Krizhevky 12
2.8. Módulo *inception* de GoogleNet propuesto por Szegedy 13
2.9. Bloque residual de ResNetNet propuesto por He *et al.* 14
2.10. Bloque Denso de DenseNet propuesto por Huang *et al.* 14
2.11. Ilustración visual del crecimiento gradual de PyramidNet 15
2.12. Diagrama de flujo de un algoritmo genético común 16
2.13. Codificación de 3 variables reales en forma binaria y real 16
2.14. *Crossover* de un solo punto 18
2.15. Ejemplo de mutación para codificación binaria 18
2.16. Ejemplos de espacios de búsqueda con conexiones en serie y con conexiones saltadas 21
2.17. Espacio de búsqueda serial de largo variable 22
2.18. Espacio de búsqueda basado en celdas 22

3.1. Codificación del espacio de búsqueda en HGAPSO 28
3.2. Ejemplo de la estrategia de codificación usada en AE-CNN 28
3.3. Ejemplo de la estrategia de codificación usada en CNN-GA 28

4.1. Proceso de reproducción cruzada del 2LGA. Primero, cada nivel genera su descendencia en (a). Luego, en (b), la descendencia del segundo nivel reemplaza parte de la descendencia del primer nivel, y luego los mejores individuos del primer nivel toman el lugar de la descendencia del segundo nivel. Finalmente, en cada nivel, la descendencia reemplaza a los peores individuos en (c). . . . 31
4.2. (a) Estructura general de las redes convolucionales en el espacio de búsqueda. Los bloques se representan por rectángulos azules, y están formados por un conjunto de celdas. (b) Estructura de la celda del espacio de búsqueda. . . . 32
4.3. Representación del cromosoma de un individuo. El cromosoma tiene información de parámetros estructurales, hiperparámetros de entrenamiento y parámetros relacionados con cada nodo. 35
4.4. Ejemplo de codificación de las conexiones de una celda. 36

4.5. Esquema de adición de nodos en el proceso de mutación. Primero (a), se elige un nodo al azar (nodo rojo A). Luego, en (b), se duplica, se muta (A') y se agrega justo antes del original. 36
4.6. Esquema de eliminación de un nodo en el proceso de mutación. En (a), cuando hay nodos disponibles para eliminar (por ejemplo, los nodos B y D), uno de ellos se selecciona al azar (nodo B marcado en rojo). Luego, en (b), se elimina y se reemplaza con una conexión (línea punteada). Cuando no hay nodos disponibles para ser eliminados como en (c), se selecciona aleatoriamente una conexión desmontable (línea verde). La conexión seleccionada (línea verde punteada) se elimina, lo que da como resultado (d). 37
4.7. Ejemplo de *crossover* entre dos individuos de largo distinto. 38
4.8. Esquema de evaluación distribuida de una población en 4 GPUs. 40
4.9. Ejemplo de los 7 datasets utilizados 42
4.10. Cambio en el ranking la población del 1L cuando recibe entrenamiento corto (columna izquierda) y cuando recibe entrenamiento largo (columna derecha). 47
4.11. Cambio en el ranking de los 8 mejores de la población 1L cuando recibe entrenamiento corto (columna izquierda) y cuando recibe entrenamiento largo (columna derecha). 48

5.1. Box-plots de los resultados de experimentos O-GA: (a) test error, y (b) tiempo de evolución. Cada experimento se repitió 5 veces en el conjunto de datos MRDBI. 51
5.2. Curvas de evolución de experimentos realizados en dataset MRDBI. Evolución del *fitness* promedio de la población (a), evolución del mejor *fitness* de la población (b), y comparación, en escala logarítmica, de la evolución del promedio (azul) y del mejor (rojo) *fitness* de la población (c). 55
5.3. Arquitectura ganadora de los experimentos realizados en el conjunto de datos MRDBI mediante el 2LGA. 56
5.4. Evaluación de modelos encontrados para dataset MRDBI, Fashion-MNIST, y CIFAR-10 en los mismos datasets. Las filas representan a los los modelos y las columnas a los datasets evaluados. En (a) el error mostrado es absoluto, y en (b) el error se normaliza por columna (en porcentaje) respecto al resultado obtenido por el modelo optimizado y evaluado en el mismo dataset (respecto a la diagonal de (a)). 59

A.1. Curvas de evolución de experimentos realizados en dataset MB. Evolución del *fitness* promedio de la población (a), evolución del mejor *fitness* de la población (b), y comparación, en escala logarítmica, de la evolución del promedio (azul) y del mejor (rojo) *fitness* de la población (c). 78
A.2. Curvas de evolución de experimentos realizados en dataset MRB. Evolución del *fitness* promedio de la población (a), evolución del mejor *fitness* de la población (b), y comparación, en escala logarítmica, de la evolución del promedio (azul) y del mejor (rojo) *fitness* de la población (c). 79
A.3. Curvas de evolución de experimentos realizados en dataset MBI. Evolución del *fitness* promedio de la población (a), evolución del mejor *fitness* de la población (b), y comparación, en escala logarítmica, de la evolución del promedio (azul) y del mejor (rojo) *fitness* de la población (c). 79

A.4. Curvas de evolución de experimentos realizados en dataset MRD. Evolución del *fitness* promedio de la población (a), evolución del mejor *fitness* de la población (b), y comparación, en escala logarítmica, de la evolución del promedio (azul) y del mejor (rojo) *fitness* de la población (c). 80
A.5. Curvas de evolución de experimentos realizados en dataset MRDBI. Evolución del *fitness* promedio de la población (a), evolución del mejor *fitness* de la población (b), y comparación, en escala logarítmica, de la evolución del promedio (azul) y del mejor (rojo) *fitness* de la población (c). 80

Capítulo 1

Introducción

1.1. Motivación

Desde el siglo XX, el hombre ha buscado dotar de inteligencia a máquinas para que realicen acciones complejas, desde percibir el entorno, hasta razonar y resolver problemas [1]. Así nacieron los primeros algoritmos de inteligencia artificial (IA), donde el programador indicaba exactamente qué acciones realizar al algoritmo, tratando de imitar las funciones cognitivas del humano [1]. Posteriormente, con el surgimiento del Aprendizaje de Maquinas, o ML por sus siglas en inglés *Machine Learning*, surgió un cambio de paradigma. En este paradigma, el programador le enseña al algoritmo que es lo que tiene hacer, a través de ejemplos, mas no explícitamente como hacerlo.

Desde entonces, muchos algoritmos de ML fueron desarrollados, logrando un gran avance en el área de la IA. Uno de los algoritmos de ML más exitosos del último tiempo, y que ha logrado los avances más extraordinarios, es el aprendizaje profundo o *Deep Learning* (DL) [2], y particularmente las redes neuronales convolucionales o *Convolutional Neural Networks* (CNN). En general, los modelos de *Deep Learning* están basados en una cascada de capas, donde cada capa usa la salida de la capa anterior como entrada y donde cada capa tiene un nivel de abstracción mayor que la capa anterior. Por lo tanto, los modelos de DL tienen múltiples niveles de abstracción. Gracias al DL se han logrado extraordinarios avances en tareas fundamentales de ML [2] tales como clasificación de imágenes, segmentación de imágenes, y reconocimiento de voz; así como en tareas de procesamiento de lenguaje, como respuesta automática de preguntas, traducción, y resumen de textos [2]. Debido a estos grandes avances, herramientas de DL se han usado con gran éxito en la industria, por ejemplo, en la industria automotriz, en la industria de la salud, y en asistentes digitales por voz (como Siri de Apple, y Alexa de Amazon) [2, 3, 4].

Uno de los aspectos fundamentales de las CNNs es el diseño de su arquitectura. Esta arquitectura influye directamente en el desempeño de la red [5, 6, 7, 8], y la arquitectura óptima varía según cada tarea. Las arquitecturas de las CNNs son tan relevantes que durante los últimos 20 años se ha hecho un esfuerzo considerable por mejorarlas. Los primeros modelos de CNNs estaban basados en estructuras cerebrales de los mamíferos, particularmente en las del sistema visual [9, 5, 10, 11, 12]. En 1979, Fukushima introdujo el Neocognitron [13], una

de las primeras CNN que modeló las estructuras del sistema visual. En 1998, LeCun introdujo las CNNs para el problema del reconocimiento de dígitos manuscritos [14]. Desde entonces, se han creado nuevas arquitecturas, modificando las existentes para lograr mejores resultados en diversas tareas. Por ejemplo, en 2012 Krizhevky *et al.* creó AlexNet [15], que fue la primera CNN en ganar la reconocida competencia *ImageNet Large Scale Visual Recognition* (ILSVR) [16]. A partir de entonces, las CNNs dominaron la competencia, logrando cada vez mejores resultados con arquitecturas cada vez más complejas.

Hoy en día, el diseño de buenas arquitecturas sigue siendo una limitante, particularmente para otras áreas que ven al DL como una potencial herramienta. El diseño de arquitecturas de CNNs es una desafío porque dicha arquitectura depende de cada problema; porque se requiere un conocimiento experto para diseñarlas; porque se requiere un esfuerzo considerable a través de prueba y error; y porque, incluso para los investigadores más versados en DL, es difícil encontrar arquitecturas óptimas [17, 18, 19, 20]. Es por esto que ha habido un creciente interés por automatizar el diseño o búsqueda de arquitecturas de CNNs.

Diversas formas de automatizar la optimización de arquitecturas de CNNs han surgido en el último tiempo, como Optimización Bayesiana (OB) o aprendizaje reforzado o *Reinforcement Learning* (RL). Sin embargo, uno de los primeros, más simples, y quizás el más elegante e intuitivo, es a través de la evolución. Así como la corteza visual de los mamíferos evolucionó gradualmente durante millones de años para realizar tareas de reconocimiento de patrones con una precisión excepcional [21, 22], la idea de evolución también es aplicable al desarrollo de mejores arquitecturas de CNNs. La evolución de las redes neuronales no es un concepto nuevo. Mas aún, ha sido un tema ampliamente abordado, y es llamado Neuroevolución [8, 9, 23, 24, 25, 26, 27].

La neuroevolución es un subtema del problema de *Neural Architecture Search* (NAS), que consiste en buscar arquitecturas CNN y NN de forma automática, pero utilizando cualquier método de búsqueda como *Reinforcement Learning* (RL) o la Optimización Bayesiana (OB). La Neuroevolución consiste en aplicar algoritmos genéticos (AG) al problema de búsqueda de arquitecturas de redes neuronales (NNs) y redes convolucionales [8]. Los algoritmos genéticos son algoritmos de búsqueda y optimización inspirados en los procesos de evolución y selección natural [28]. Los AGs están basados en poblaciones, que son un conjunto de posibles soluciones al problema. Estas soluciones, o individuos, son evaluadas mediante una función de *fitness*, función que debe representar al problema en cuestión, y que corresponde a la función objetivo a optimizar. Entonces, se seleccionan a los mejores individuos para que puedan generar descendencia, imitando a la selección natural. Luego, los mejores individuos se reproducen para generar descendencia (nuevas posibles soluciones), esperando que éstas sean superiores a sus padres en resolver la tarea objetivo. Esta reproducción se lleva a cabo mediante operadores genéticos como mutación y *crossover*. Finalmente, los nuevos individuos reemplazan a los peores de la generación anterior. Este proceso se repite un número determinado de generaciones, o hasta alcanzar un tiempo límite [8, 9, 23, 24, 25, 26, 27].

En el caso de la Neuroevolución, los individuos o soluciones son arquitecturas de CNNs, lo que conlleva dos grandes desafíos: la evaluación y la codificación de estos individuos. Por una parte, la forma más intuitiva de evaluar estos individuos es hacerlo como es común en las CNNs: entrenándolos en un conjunto de datos de entrenamiento, y evaluándolos en un

conjunto de validación [8, 23]. Sin embargo, debido a que el entrenamiento de las CNNs es considerablemente largo y a que se debe evaluar una gran cantidad de redes (poblaciones completas), el tiempo de evolución puede ser muy extenso. Dos de los primeros trabajos en utilizar este enfoque fueron Real et. al [27] y Liu et. al [18]. Ambos lograron los mejores resultados en ese entonces, pero tomando semanas en lograrlo y requiriendo centenares de procesadores gŕaficos (GPUs). Por lo tanto, algunos estudios propusieron técnicas de aproximación del *fitness* para reducir el tiempo de entrenamiento de las CNNs y, por ende, el tiempo de evolución. Algunas de estas técnicas son [23], entrenar los modelos con menos ejemplos [29], con imágenes de menor resolución [30], reducir el tamaño de los modelos [24], y entrenar los modelos con menos épocas [9, 18, 27, 31], siendo la última la más utilizada.

Sin embargo, la evaluación o cálculo del *fitness*, es menos confiable al usar estas técnicas, ya que son solo una aproximación. Según Zela et. al [32], el utilizar entrenamientos cortos como forma de aproximación afecta la elección de las mejores arquitecturas, resultando en una baja correlación entre los rendimientos obtenidos luego de un entrenamiento corto y los obtenidos luego de un entrenamiento completo. Por tanto, incluso si estas aproximaciones reducen el tiempo de entrenamiento, también reducen la fidelidad del *fitness* y pueden generar un error en el ranking de la población [23, 32]. En consecuencia, el AG buscará el mejor modelo que se ajusta a la restricción, pero que no es necesariamente el mejor para el problema global. Esto genera un compromiso entre fidelidad y tiempo para el cálculo del *fitness*, ya que cuanto más confiable es el *fitness* —y por lo tanto el ranking—, más tiempo se requiere para calcularlo y mayor es el tiempo de evolución [23].

Muchos de los trabajos que utilizan aproximaciones del *fitness* [9, 33, 34] no hacen comparaciones del algoritmo de búsqueda con y sin estas aproximaciones, debido al alto costo computacional requerido para realizar estos experimentos. Por lo tanto, en estos trabajos no se conoce ni se cuantifica la disminución del rendimiento producido exclusivamente por el uso de estas aproximaciones. En otras palabras, el problema de la compensación entre el tiempo de cálculo y la fidelidad del *fitness* se subestima o se desprecia.

El segundo gran desafío de la Neuroevolución es la representación de los individuos, la que puede dividirse en dos etapas. La primera etapa es la definición del espacio de búsqueda, que corresponde al conjunto de todas las posibles soluciones. Por lo tanto, la definición del espacio de búsqueda restringe las soluciones que el AG puede encontrar [8]. Por ejemplo, si el espacio de búsqueda corresponde a redes con un total de N capas (N fijo), el AG no podrá encontrar soluciones con más (o menos) capas, introduciendo un sesgo humano al algoritmo. Muchos autores han propuesto espacios de búsqueda restringidos, con el objetivo de reducir su dimensión y que el AG lo explore en mayor proporción. Sin embargo, no es posible asegurar que la mejor arquitectura se encuentre dentro del espacio definido, y mientras más reducido sea, es menos probable que esto ocurra.

La segunda etapa de la representación de las soluciones corresponde a la codificación, es decir, a la representación de cada individuo en un conjunto de genes. La codificación afecta la forma en que los individuos se reproducen, por lo que también es un factor importante dentro de la evolución en un AG [9, 20]. Por otra parte, si el espacio de búsqueda es complejo, tanto la codificación como la operación de *crossover* se complejizan. Por ejemplo, si el espacio de búsqueda contempla redes con un número indefinido de capas, el crossover entre individuos

no es directo ya que las codificaciones no tendrán en principio el mismo largo.

1.2. Hipótesis

1. El evaluar a todos los individuos de la población mediante entrenamientos precisos (largos) conlleva un costo computacional innecesario, ya que existen individuos de bajo desempeño que pueden detectarse como tal utilizando entrenamientos cortos.

2. Los algoritmos genéticos que utilizan entrenamientos aproximados (cortos) o entrenamientos precisos (largos), representan los extremos del problema de compromiso entre tiempo de cómputo y fidelidad del fitness. Por lo tanto, un algoritmo que combine ambos tipos de entrenamientos puede balancear ambos atributos, pudiendo alcanzar una mejor combinación de ambos.

3. Es posible mejorar el desempeño en reconocimiento de patrones en imágenes, en bases de datos pequeñas, optimizando la arquitectura de CNNs con un Algoritmo Genético de dos niveles.

1.3. Objetivo General

El objetivo de este trabajo es desarrollar e implementar un algoritmo *end-to-end* capaz de buscar, de forma automática, la arquitectura e hiperparámetros de redes neuronales convolucionales, obteniendo desempeños competitivos con el estado del arte en el problema de reconocimiento de patrones en imágenes.

1.4. Objetivos Específicos

1. Crear un nuevo tipo de algoritmo genético de dos niveles que aborde el problema del compromiso entre fidelidad y tiempo para el cálculo del *fitness* de las CNNs, mediante la combinación de entrenamientos cortos y largos.

2. Desarrollar e implementar un nuevo espacio de búsqueda que reduzca las restricciones de los parámetros, y una codificación que permita al AG realizar *crossover* entre arquitecturas de CNNs de distinto tamaño, y con distintas operaciones.

3. Aplicar, tanto el algoritmo genético de dos niveles como el espacio de búsqueda propuesto, a los conjuntos de datos MNIST-Variants, Fashion-MNIST y CIFAR-10, y comparar los resultados con el estado del arte.

4. Comparar, en el conjunto de datos MNIST-Variants, el tiempo de evolución del algoritmo genético de dos niveles propuesto, con un algoritmo genético ordinario que utilice entrenamientos cortos únicamente.

5. Comparar, en el conjunto de datos MNIST-Variants, el desempeño del algoritmo genético de dos niveles propuesto, con un algoritmo genético ordinario que utilice entrenamientos largos únicamente.

1.5. Contribuciones

Las principales contribuciones del presente trabajo de tesis pueden resumirse como sigue:

1. Un nuevo 2LGA que aborda el problema de la compensación entre fidelidad y tiempo de cálculo del *fitness* en las CNN, reduciendo los efectos negativos del uso de técnicas de aproximación del *fitness*. Con este método, cualquier AG que utilice algún tipo de aproximación para el cálculo del *fitness* puede beneficiarse, sacrificando una fracción de tiempo pero aumentando su rendimiento.

2. Un nuevo espacio de búsqueda con pocas restricciones de parámetros, y una codificación que permite realizar *crossover* entre arquitecturas de distinto largo y con diferentes operaciones.

3. Aplicación del 2LGA y del espacio de búsqueda propuesto al conjunto de datos MNIST-Variants, y al conjunto de datos Fashion-MNIST, alcanzando resultados significativamente mejores que los publicados anteriormente.

4. En el conjunto de datos MNIST-Variants, se muestra que el 2LGA reduce los efectos negativos de aproximar el *fitness*, alcanzando el rendimiento de un AG ordinario de entrenamientos largos pero demorando tanto como un AG ordinario de entrenamientos cortos.

Capítulo 2

Marco Teórico

2.1. Perceptrón

El perceptrón es uno de los modelos matemáticos más simples de una neurona, y está inspirado en su funcionamiento [35]. El perceptrón contiene los elementos más básicos de una neurona, es decir, un conjunto de entradas (similar a las dendritas), una función que opera sobre las entradas (cuerpo), y una salida (axón). En particular, el modelo matemático consiste en ponderar cada entrada x_{i} por un factor determinado para cada de estas entradas (w_{i}), y en operar dicha suma con una función de activación. De esta forma, un perceptrón se representa gráficamente como se muestra en la Figura 2.1, y se representa matemáticamente según 2.1

$$Y = f(\sum_{\mathrm{i}=1}^{N} x_{\mathrm{i}} \cdot w_{\mathrm{i}}), \tag{2.1}$$

donde Y es la salida del perceptrón, f la función de activación, y N el número de entradas del perceptrón.

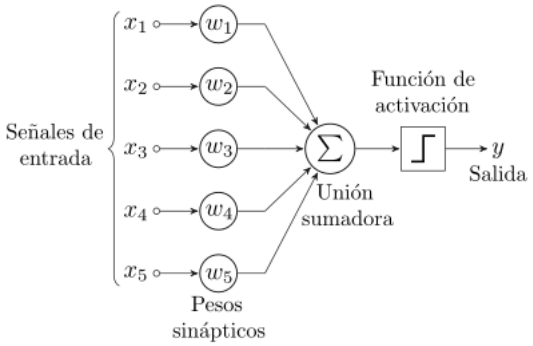


Figura 2.1: Representación gráfica de un perceptrón (imagen tomada de [36]).

2.2. Redes Neuronales

Al igual que las neuronas, los perceptrones adquieren mayor productividad al organizarse en redes que de forma individual. Esto da lugar a las redes neuronales o *Neural Networks* (NNs), que son un conjunto de neuronas conectadas entre sí, donde la salida de unas es la entrada de otras [37]. La forma más común de organización de las NNs es a través de capas, donde las salidas de una capa se convierten en las entradas de la capa siguiente, tal como se muestra en la Figura 2.2. Esta forma de organización se conoce como perceptrón multicapa, o MLP por sus siglas en inglés, *Multi Layer Perceptron*. Las MLP tienen tres componentes principales: la entrada (que recibe los datos), un número de capas ocultas (que procesan la entrada), y una salida.

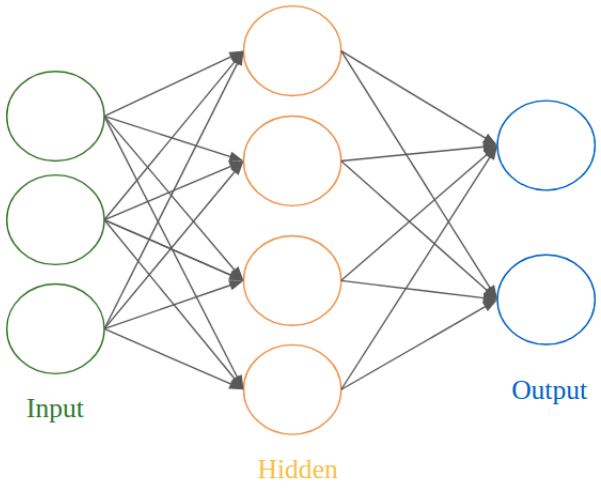


Figura 2.2: Representación gráfica de una red neuronal.

Las NNs son capaces de aproximar funciones, pese a que ésta no sea explícitamente conocida. Las NN pueden aproximar desde funciones matemáticas, hasta funciones cognitivas relacionadas con mamíferos, como la asociada a la clasificación de imágenes [37]. Mas aún, el Teorema de Aproximación Universal (TAU) afirma que toda función continua se puede aproximar por una red neuronal [38]. Para que una NN se ajuste a una función, se deben ajustar los pesos w_{ij} de cada perceptrón que la compone.

2.3. Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales, o *Convolutional Neural Networks* (CNN) son un tipo especial de Red Neuronal, diseñadas para procesar información en forma de grilla o donde el orden de los datos es un factor determinante, como en series de tiempo o imágenes [37]. Las CNNs están basadas en estructuras cerebrales del sistema visual de los mamíferos, y en particular, en la corteza visual primaria [37]. La corteza visual primaria de los mamíferos realiza una operación semejante a la convolución [12, 37], operación que además le otorga el nombre a las CNN. La convolución es un tipo de operación lineal que utiliza información del orden relativo entre los datos, como la información temporal en series de tiempo o la información espacial en imágenes.

Las CNN que procesan imágenes (arreglos bidimensionales) utilizan la convolución discreta en dos dimensiones (2D) definida en (2.2)

$$s(x,y) = g(x,y) * k(x,y) = \sum_{n=1}^{N} \sum_{m=1}^{M} g(n,m) \cdot k(x-n, y-m), \tag{2.2}$$

donde $g(x,y)$ representa los datos de entrada (imagen de tamaño NxM), $k(x,y)$ una matriz con los pesos w_{i} a optimizar (una máscara o *kernel* de tamaño definido), y $s(x,y)$ al resultado de la operación.

La operación convolución en dos dimensiones 2D equivale a superponer la máscara en la posición (x,y) de la imagen, y sumar los productos entre los elementos del *kernel* y los de la imagen que coinciden espacialmente, como se muestra en la Figura 2.3. Esta máscara se desplaza vertical y horizontalmente (proceso conocido como ventana deslizante) dando forma a la imagen de salida, que se conoce como *feature map*. Se define como capa convolucional a la concatenación de k *feature maps* (originados por k *kernels* distintos), seguido de una función de activación.

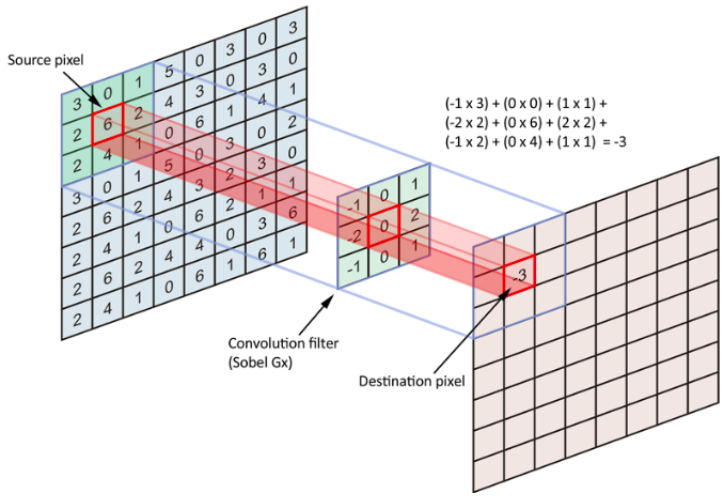


Figura 2.3: Representación gráfica de operación convolución bidimensional (imagen tomada de [39]).

Una de las ventajas de usar operaciones de convolución es que se requiere una cantidad de parámetros considerablemente menor que para una MLP. Para el ejemplo de la Figura 2.3, cada salida de una MLP tendría NxM pesos $w_{\text{i}j}$, un peso w por cada entrada. Por lo tanto, se necesitan (NxM)(NxM) parámetros para formar un *feature map* de dimensión NxM como el del ejemplo. Sin embargo, la convolución solo requiere la cantidad de parámetros que forman al *kernel* $k(x,y)$.

Además de las capas convolucionales, es común utilizar capas de *pooling* que reducen la dimensión de los *feature maps*. Éstas utilizan el mismo principio de ventana deslizante que las capas convolucionales, pero generalmente usando una función distinta a la suma de productos de la ventana [37]. Las operaciones más comunes son el máximo y el promedio de la ventana, originando las capas Max-Pooling y Average-Pooling respectivamente. Además, la operación

de *pooling* se realiza cada n pixeles ($n > 1$) para reducir la dimensión de los *feature maps*. Al valor de n se le conoce como *stride*.

2.4. Entrenamiento de Redes Neuronales

El entrenamiento de una red consiste en encontrar sus pesos a través de un proceso de optimización, utilizando ejemplos de entrenamiento. Para problemas supervisados, los ejemplos de entrenamiento corresponden a un conjunto de datos de entrada (imágenes, series de tiempo, etc.) con su salida esperada (como la clase si se trata de un problema de clasificación). El entrenamiento de una red consiste en entregarle datos de entrada a la red, calcular su salida, y compararla con la salida deseada. Esta comparación se realiza calculando una función de costo previamente definida y acorde al problema. La función de costo debe minimizarse, por lo que se utiliza algún método de optimización para ajustar los pesos de la red y así minimizar dicha función [37].

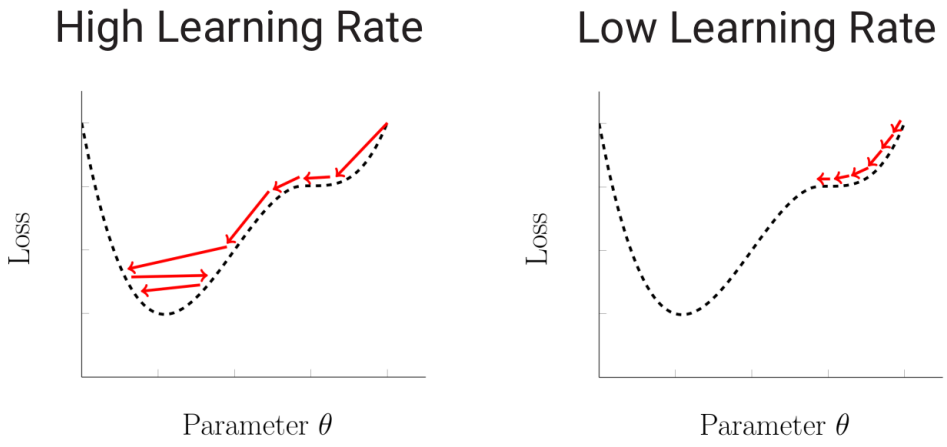


Figura 2.4: Efectos de un *learning rate* alto (izquierda) y bajo (derecha) en el entrenamiento de una red neuronal (imagen tomada de [40]).

Los métodos de optimización para ajustar los pesos w_{i} de las NN (incluidos los kernels de las capas convolucionales) son variados. Por ejemplo, Fukushima [41] utilizó un método de agrupamiento (clusterización) por capas; Pérez *et. al* utilizó algoritmos genéticos para encontrar campos receptivos (*kernels*) bio-inspirados; y Miller *et. al* usó algoritmos genéticos para encontrar directamente los pesos de la red [42]. Sin embargo, el método de optimización más exitoso hasta ahora es mediante *Stochastic Gradient Descent* (SGD) y *back-propagation*. Por un lado, SGD es un método iterativo de optimización que estima el gradiente al usar un subconjunto aleatorio de datos (este subconjunto se conoce como *batch*). Y, por otra parte, *back-propagation* es un método de cálculo de gradiente, que propaga hacia atrás la información de la función de costo [37]. Combinando ambos, se ajustan los pesos $w_{\mathrm{i}j}$ en función del gradiente propagado según (2.3)

$$w_{\mathrm{i}} = \hat{w}_{\mathrm{i}} - \alpha \frac{\partial}{\partial w_{\mathrm{i}}} L(f(X; W), Y), \tag{2.3}$$

donde w_i es el peso a ajustar, $\hat{w}_i$ es el valor previo de dicho peso, L es la función de costo que depende de las salidas deseadas Y, de la salida de la red f, de los pesos W y de los datos de entrada X. Finalmente α es el *learning rate*, un factor que determina la magnitud del ajuste.

Al ser un método iterativo, este proceso se debe repetir hasta converger. A la aplicación de este proceso usando como entrada un único *batch* se conoce como *step*, mientras que se llama época al conjunto de *steps* necesarios para que todos los ejemplos del conjunto de entrenamiento sean evaluados una vez [43].

El *learning rate* es uno de los parámetros más importantes en el entrenamiento de las NN y CNN, ya que define si la red logra converger y las épocas necesarias para ello. Como se ejemplifica en la Figura 2.4, un *learning rate* muy alto provocará que el cambio en el nuevo peso sea mayor que el necesario para converger, mientras que, si es muy bajo, el entrenamiento puede converger a mínimos locales o necesitar una cantidad excesiva *steps* [44]. Para acelerar el aprendizaje al principio y asegurar la convergencia al final del entrenamiento, se puede variar el *learning rate* a lo largo del entrenamiento. A esta técnica se conoce como *schedule learning rate* [44]. Por lo general, la variación del *learning rate* es decreciente, es decir, se comienza con un *learning rate* alto, y se disminuye hasta finalizar el entrenamiento. Pese a esto, es posible usar las primeras épocas del entrenamiento como calentamiento o *warmup*, donde el *learning rate* es muy bajo, o creciente hasta alcanzar el valor maximo [45].

2.5. Sobreajuste y Regularización

Un problema recurrente en el entrenamiento de CNN es que éstas pueden sobre ajustarse a los datos de entrenamiento, debido a la gran cantidad de parámetros que tienen las redes. En otras palabras, el modelo memoriza los ejemplos de entrenamiento para minimizar la función de costo, en lugar de extraer las características relevantes para ello [37]. Una representación gráfica de esto se puede ver en la Figura 2.5, donde la línea representa la línea de desición que separa a las dos clases (círculos azules y triángulos rojos). En el caso de la imagen (a), la línea de decisión rodea los triángulos rojos, pese a que la curva óptima o balanceada, presente en la imagen (b), es menos compleja. Al fenómeno del primer caso se conoce como *overfitting* o sobreajuste.

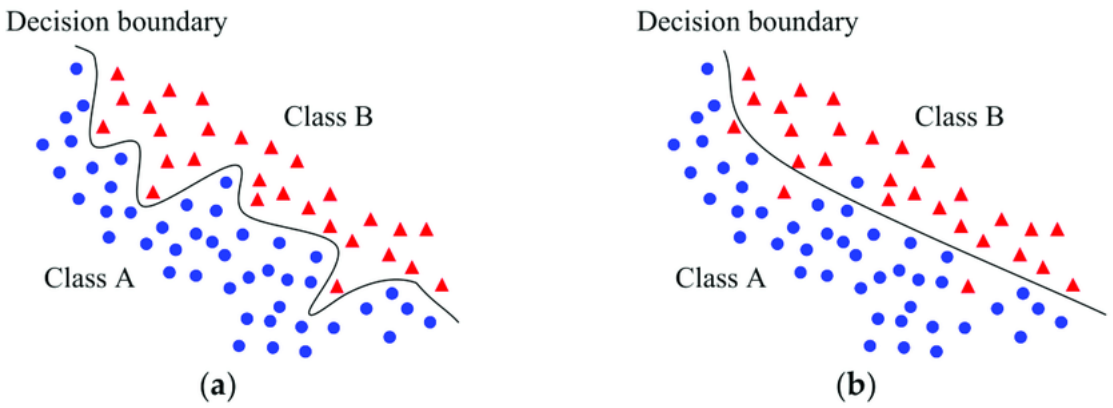


Figura 2.5: Ejemplo de sobreajuste en un problema de clasificación (imagen tomada de [46]).

Para afrontar el sobreajuste, existen técnicas de regularización que añaden restricciones o complejidad al entrenamiento y que dificultan la memorización de los datos. Entre las técnicas de regularización más usadas estan el *Dropout* (anulación de neuronas aleatorias durante el entrenamiento) [47], *Weight Regularization* como L1 o L2 (restricción o penalización a la magnitud de los pesos de la red) [37], *Data Augmentation* (alteración de los ejemplos de entrenamiento, agregando dificultad en ellos) [48], *Smooth Label* (alteración de las etiquetas de los ejemplos de entrenamiento) [49], entre otras.

2.6. Arquitecturas de CNN diseñadas a mano

La arquitectura de una CNN corresponde a los elementos que la componen, así como su distribución. La arquitectura considera principalmente el número de capas de la red, el tipo de cada una de estas capas (que puede ser convolucional, de *pooling*, *fully connected*, entre otras), los parámetros internos de cada capa (como el tamaño del *kernel* de las capas convolucionales), y la distribución de estas capas en la red, es decir, la forma en que se conectan entre ellas [23].

La arquitectura de una red influye fuertemente en su desempeño [6, 7, 8], y encontrar arquitecturas óptimas para una tarea determinada es un problema abierto, incluso para investigadores con experiencia en Deep Learning (DL) [17, 18, 19, 20]. La importancia de las arquitecturas de las CNN es tal que, en los últimos 20 años, se han realizado numerosos estudios al respecto, invirtiendo una gran cantidad de tiempo y esfuerzo.

A continuación, se presentan de forma cronológica las arquitecturas, y sus características, más relevantes para el desarrollo de las redes convolucionales.

2.6.1. Red LeNet-5

En 1998, LeCun introdujo las CNN para el problema del reconocimiento de dígitos escritos a mano, entrenadas con *back-propagation* [14] (anteriormente, Fukuchima [50] había utilizado CNNs, pero utilizando un método de entrenamiento distinto). La arquitectura propuesta por LeCun (LeNet-5) consta de 3 capas convolucionales y 2 de *pooling*. Las capas convolucionales tienen 6, 16 y 120 *feature maps* cada una, y *kernels* de tamaño 5x5. Además, la red tiene una capa *fully connected* de 84 unidades. Esta arquitectura se muestra en la Figura 2.6.

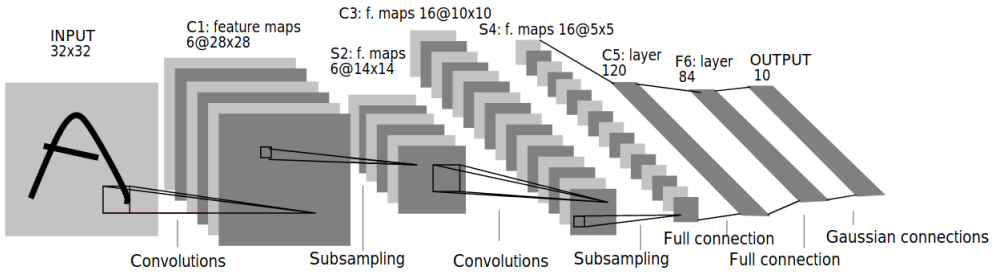


Figura 2.6: Arquitectura de red LeNet-5 propuesta por LeCun [14] (imagen tomada de [14]).

2.6.2. Red AlexNet

Posteriormente, Krizhevky *et al.* utilizó los conceptos desarrollados por LeCun para desarrollar la red AlexNet [15], pero aumentando las capas convolucionales, los *feature maps* y los tamaños de los kernels. En particular, AlexNet tiene 5 capas convolucionales, 2 de Max-Pooling y 3 *fully connected*. Las capas convolucionales tienen 96, 256, 384, 384 y 256 *feature maps*, con *kernels* cuadrados de tamaño 11, 5, 3, 3, y 3 respectivamente, mientras que las capas *fully connected* tienen 2048, 2048 y 1000 unidades cada una. La arquitectura original de esta red se muestra en la Figura 2.7. Por otra parte, los autores introdujeron la función de activación ReLU (*Rectified Linear Unit*) definida en (2.4)

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

función que ha sido fundamental en el desarrollo de las CNN debido a su eficiencia computacional, en comparación con funciones exponenciales utilizadas hasta entonces. Un logro importante de AlexNet es que fue la primera red convolucional en ganar el concurso ILSVR[16], un reconocido concurso de clasificación de imágenes.

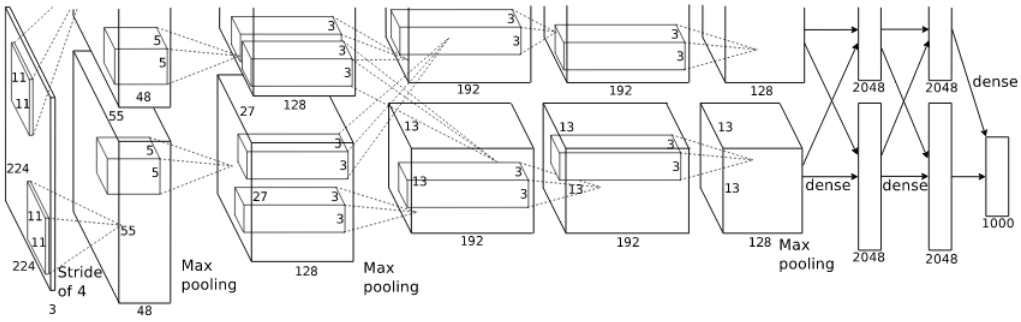


Figura 2.7: Arquitectura de AlexNet propuesta por Krizhevky *et al.* [15] (imagen tomada de [15]).

2.6.3. Red VGG

En 2013, Simonyan *et al.* [51] propusieron la arquitectura de red VGG, un nuevo tipo de CNN que ganó la competicion ILSVR del mismo año. Las redes VGG heredan algunos elementos de la red AlexNet, como el uso de la función de activación ReLU en sus capas convolucionales, pero tienen ciertas modificaciones en su arquitectura. En primer lugar, estas redes tienen mayor cantidad de capas (mayor profundidad), alcanzando para la red VGG-19 un total de 16 capas convolucionales, 5 capas de Max-pooling, y 3 capas *fully connected*. Para aumentar la profundidad sin aumentar el número de parámetros de la red, se redujeron los *kernels* de las convoluciones a tamaño 3x3, introduciendo además *kernels* de tamaño 1x1 para aumentar la no linealidad de la red. Además, la cantidad de *feature maps* también aumentó, alcanzando un máximo de 512. Finalmente, aunque el número de capas *fully connected* es el mismo que la red AlexNet, el número de unidades aumentó a 4096 para las 2 primeras capas [51].

2.6.4. Red GoogleNet

Posteriormente, Szegedy *et al.* diseñaron la arquitectura de red GoogleNet [52], una red más profunda que VGG. Las mejoras de GoogleNet respecto a sus predecesoras pueden resumirse en 4 elementos. Uso de convoluciones de 1x1, introducción del módulo *inception*, reemplazo de capas *fully connected* por una de *Global Average Pooling*, y la arquitectura en general. En primer lugar, a diferencia de VGG, GoogleNet utiliza las convoluciones de 1x1 para reducir la cantidad de *feature maps* de la red, disminuyendo la cantidad de parámetros y pudiendo aumentar el tamaño de la red. En segundo lugar, los autores introdujeron el módulo *inception* (Figura 2.8), una concatenación de convoluciones que usan distintos tamaños de *kernels*. Mediante este módulo se abrió la posibilidad de no conectar únicamente las capas de forma serial. En tercer lugar, el reemplazo de las 2 primeras capas *fully connected* por una de *Global Average Pooling* (que promedia todo un *feature map* para reducirlo a un único valor) logró reducir considerablemente la cantidad de parámetros de la red. Finalmente, la arquitectura misma aumento su tamaño, alcanzando 22 capas convolucionales, 4 de Max-Pool, y 1 *fully connected*, y aumentando la cantidad máxima de *feature maps* a 1024 [52].

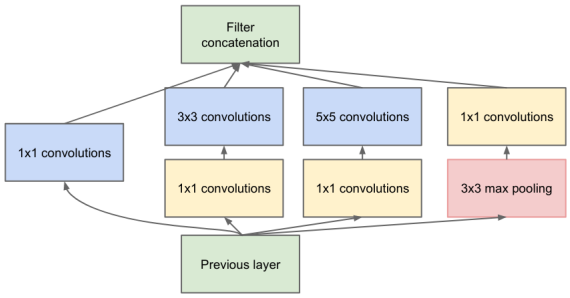


Figura 2.8: Módulo *inception* de GoogleNet propuesto por Szegedy *et al.* [52] (imagen tomada de [52]).

2.6.5. Red ResNet

Hasta ese momento, un aumento en la profundidad de las redes había demostrado tener mejores resultados, pero redes más grandes no eran factibles de entrenar debido al problema del desvanecimiento del gradiente [53, 54]. El desvanecimiento del gradiente es la tendencia del gradiente de la función de costo a volverse pequeño. Esto ocurre cuando se debe realizar una gran cantidad de derivadas parciales sobre la función de costo para obtener el gradiente de una capa (cuando las redes son muy profundas). Para afrontar esto, He *et al.* desarrolló un bloque residual que conecta la entrada y la salida de un bloque sumándolos [55], como se muestra en la Figura 2.9. Con este bloque, los autores lograron entrenar redes con hasta 151 capas, evitando el problema del desvanecimiento del gradiente ya que, debido a la conexión saltada, hay menos operaciones (y derivaciones) entre una capa y la función de costo.

2.6.6. Red DenseNet

Más adelante, Huang *et al.* ampliaron el concepto de conexión de saltada y desarrollaron un bloque Denso [56]. En este nuevo bloque, la entrada de una capa convolucional es la concatenación de todas las capas anteriores del mismo bloque (ver Figura 2.10). Gracias a

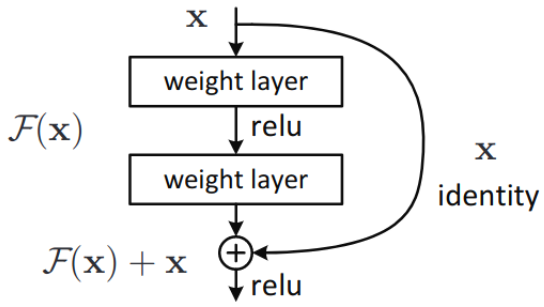


Figura 2.9: Bloque residual de ResNetNet propuesto por He *et al.* [55] (imagen tomada de [55]).

este bloque, las capas posteriores reutilizan los *feature maps* de las capas anteriores, por lo que se necesitan menos capas (y menos parámetros) para lograr un buen desempeño. Además, se reducen aún más los efectos del desvanecimiento del gradiente. Por otro lado, los autores introdujeron el concepto de factor de crecimiento, o *growth rate*, que corresponde a aumentar linealmente la cantidad de *feature maps* de un bloque, a medida que aumenta la profundidad. Un ejemplo de esto se muestra en la Figura 2.10, donde se agregan 32 *feature maps* en cada concatenación del bloque denso.

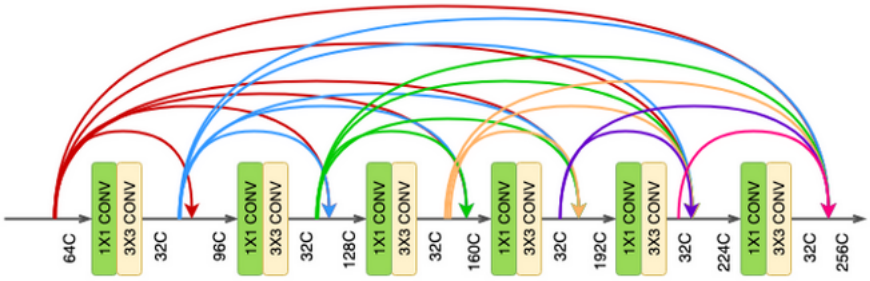


Figura 2.10: Bloque Denso de DenseNet propuesto por Huang *et al.* [56] (imagen tomada de [57]).

2.6.7. Red PyramidNet

Han *et al.* amplió el concepto de crecimiento gradual de los *feature maps* para generar la arquitectura de red PyramidNet [58], una red que aumenta los *feature maps* de forma lineal (2.5), o multiplicativa (2.6) a lo largo de toda la red según

$$D_k = \begin{cases} 16 & if \quad k = 1 \\ \lfloor D_{k-1} + \alpha/N \rfloor & if \quad 2 \leq k \leq N+1, \end{cases} \tag{2.5}$$

$$D_k = \begin{cases} 16 & if k = 1 \\ \lfloor D_{k-1} \cdot \alpha^{\frac{1}{N}} \rfloor & if 2 \leq k \leq N+1, \end{cases} \tag{2.6}$$

con α un factor de crecimiento, N el número de capas de la red, k el índice de la capa

número k, y D_k la cantidad de *feature maps* de la capa k. En la Figura 2.11 se ilustran los tipos de crecimiento gradual de PyramidNet, así como una comparación de ambos.

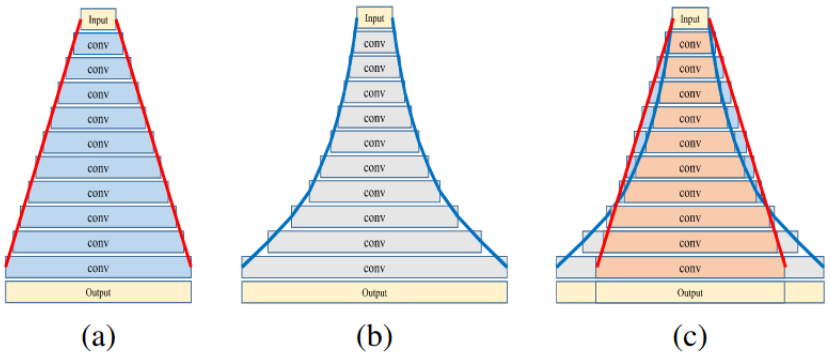


Figura 2.11: Ilustración visual del crecimiento gradual de PyramidNet de forma (a) aditiva, (b) multiplicativa, y (c) una comparación de ambos (imagen tomada de [58]).

2.7. Algoritmos Genéticos

Los Algoritmos Genéticos (AG) son algoritmos de optimización, búsqueda y aprendizaje inspirados en los procesos de Evolución Natural y Evolución Genética [28]. Estos algoritmos simulan el proceso de selección natural, en el que se eligen a los individuos mejor adaptados a ciertas condiciones y se reproducen con el fin de producir descendencia cada vez más apta a dichas condiciones. El proceso se simula por medio de operadores genéticos de recombinación, mutación y selección.

Los AG son apropiados para la búsqueda de arquitecturas de CNN ya que los AG no requieren calcular gradientes y son insensibles a los mínimos locales [9, 20, 59]. Además, los AG pueden abordar problemas en los que no hay una función objetivo explícita o exacta [59], lo que ocurre en la búsqueda de arquitecturas de CNN debido a la aleatoriedad del entrenamiento con SGD. Finalmente, los AG tienen buena capacidad de búsqueda global [60], a diferencia de los métodos de optimización por gradiente descendente que convergen a mínimos locales en superficies no convexas [61].

El proceso típico de un AG (ver Figura 2.12) consta principalmente de 7 fases: inicialización de la población, cálculo de la función de *fitness*, selección, *crossover*, mutación, reemplazo (nueva población), y término.

2.7.1. Población Inicial

El proceso comienza con la creación de un conjunto de individuos, conjunto denominado población. Cada individuo es una solución (en el espacio de búsqueda) del problema que se desea resolver. Los individuos están representados por un cromosoma, que es un conjunto concatenado de parámetros o variables, conocidos como genes [8, 63].

Existen numerosas formas de representar una solución mediante un conjunto de variables

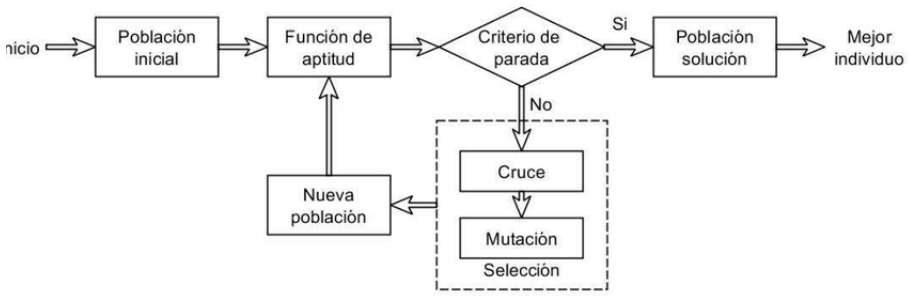


Figura 2.12: Diagrama de flujo de un algoritmo genético común (imagen tomada de [62]).

concatenadas (cromosoma), y dependen del problema en cuestión. Las formas más comunes de codificaciones son binaria y real [63, 64]. La codificación binaria se expresa en bits, la unidad mínima de información, mientras que la real utiliza variables reales. Por ejemplo, si se desean codificar 3 variables X_1, X_2, y X_3 (ver Figura 2.13), éstas se pueden representar en forma binaria (imagen izquierda), o de forma real (imagen derecha). Sin embargo, la codificación binaria solo admitiría, en este caso, valores enteros para X_1, X_2, y X_3.

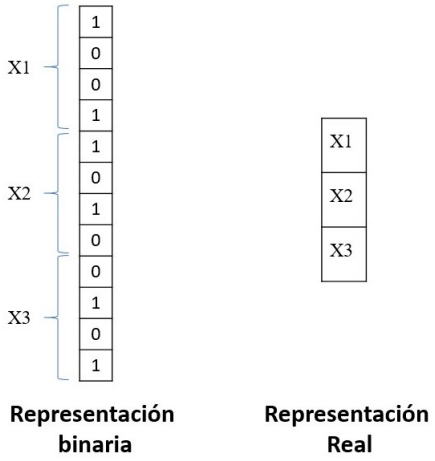


Figura 2.13: Codificación de 3 variables reales en forma binaria (izquierda) y real (derecha).

2.7.2. Función de *Fitness*

La función de *fitness* es la función a optimizar y determina que tan bueno es el individuo para resolver el problema en cuestión. Por lo tanto, la formulación del cálculo del *fitness* debe representar de la mejor forma posible al problema a resolver [28]. La función de *fitness* toma como entrada al cromosoma del individuo, y retorna un valor real. Este valor es usado como puntuación del individuo, y se utiliza posteriormente para definir que individuos serán seleccionados en la reproducción . El individuo óptimo debe tener el *fitness* más alto (o más bajo, dependiendo de la formulación del problema). La función de *fitness* es fundamental para los AG, ya que una elección inadecuada de esta función puede provocar que el AG no converja a soluciones deseadas, o que requiera más iteraciones para lograrlo [28].

2.7.3. Selección

La fase de selección consiste en seleccionar a los individuos que podrán reproducirse y traspasar sus genes a la siguiente generación. Para la reproducción, se selecciona un par de individuos, que serán los padres, en función de su *fitness*. De este modo, los individuos con mejor *fitness* tienen mayor probabilidad de ser elegidos para la reproducción [28].

Los individuos menos buenos también deben tener oportunidad de reproducirse, ya que pueden incluir material genético útil en el proceso de reproducción. Esto define la presión selectiva, que determina en qué grado la reproducción está dominada por los mejores individuos.

Los métodos de selección más utilizados son [63]:

- **Ruleta**: La probabilidad de selección es proporcional al valor del fitness del individuo.
- **Selección Aleatoria o *Random Selection* (RS)**: Los individuos se escogen de forma aleatoria.
- **Selección por Torneo o *Tournament Selection*(TS)**: Se selecciona un grupo de N individuos de forma aleatoria (N=1, 2, etc), y se escoge al de mejor *fitness* del grupo. Con N=1, se reduce a una búsqueda aleatoria (RS).
- **Orden Lineal o *Linear Order* (LO)**: La probabilidad de selección es proporcional a la posición del individuo dentro del ranking de la población, ranking formado en base al *fitness* de los individuos.

2.7.4. *Crossover*

El *crossover* es una de las fases más importantes de los AG, debido a que permite la explotación de espacio de búsqueda, es decir, una búsqueda en profundidad en zonas prometedores del espacio [8]. El *crossover* combina los genes de dos o más individuos padres para generar hijos, o nuevas soluciones [28, 63].

El *crossover* más aprobado y ampliamente utilizado en codificaciones binarias es el *crossover* de un solo punto (*Single point crossover*) [65]. Este consiste en seleccionar un punto del cromosoma y dividir en dos a los padres, ambos en el punto seleccionado [65]. Luego, un hijo se forma combinando la parte inicial de un padre, y la parte final del otro, tal como se muestra en la Figura 2.14.

Por otra parte, para codificaciones reales es común utilizar un crossover aritmético completo (*whole arithmetic crossover*) [67]. Este consiste en calcular, para cada gen, la suma ponderada por un parámetro aleatorio α entre los padres. El parámetro α es un valor real entre 0 y 1, y es el mismo para todos los genes [67]. El cromosoma del hijo (gen_{hijo}) se calcula según (2.7)

$$\text{gen}_{\text{hijo}} = \alpha \cdot \text{gen}_{p1} + (1 - \alpha) \cdot \text{gen}_{p2}, \tag{2.7}$$

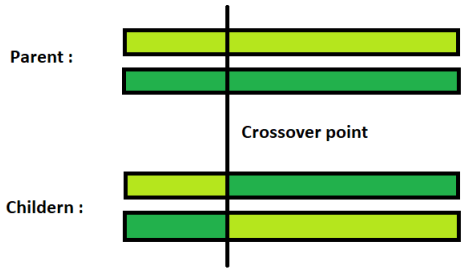


Figura 2.14: *Crossover* de un solo punto (imagen tomada de [66]).

donde gen_{p1} es el cromosoma del padre 1, y gen_{p2} el cromosoma del padre 2.

2.7.5. Mutación

La mutación corresponde a modificar uno o más genes de un individuo. Esta modificación es aleatoria y debe permitir alcanzar cualquier punto del espacio de búsqueda. Por esta razón, la mutación permite realizar una exploración del espacio de búsqueda, es decir realizar una búsqueda en amplitud y localizar nuevas zonas prometedoras [8, 68].

Al igual que para el *crossover*, cada tipo de codificación requiere un tipo de mutación distinta. Mas aún, para el mismo tipo de codificacion, existen muchos tipos de mutaciones [63]. Para la codificación binaria, la mutación más utilizada corresponde a cambiar el valor de un bit escogido al azar, con una cierta probabilidad. Un ejemplo de esto se muestra en la Figura 2.15. Por otro lado, si la codificación es real, uno de los métodos más utilizados es la mutación gaussiana [69], que utiliza la función de error de Gauss.

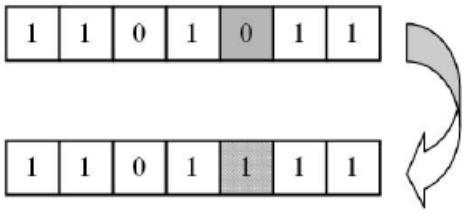


Figura 2.15: Ejemplo de mutación para codificación binaria (imagen tomada de [70]).

2.7.6. Reemplazo

El reemplazo consiste en eliminar individuos de la población para poder incorporar a la nueva generación. De esta forma, la población siempre tendrá el mismo número de individuos. Según el método de reemplazo, existen dos tipos de AG:

- **Generacional**: Durante cada iteración, una población completa de individuos nuevos es creada (descendencia), la que reemplaza completamente a la población antigua. Sin embargo, para no perder a los mejores individuos a lo largo de la evolución, se suelen mantener a los N mejores individuos de la población. Esto se denomina elitismo [71].

- **Estacionario**: Durante cada iteración, se escoge un par de padres para formar descendencia, la que se puede componer de 1 o 2 nuevos individuos. Posteriormente, la descendencia reemplaza a 1 (o 2) individuos de la población antigua [71].

Para algoritmos genéticos generacionales se suele reemplazar a los peores de la población para mantener el elitismo. Sin embargo, para algoritmos genéticos estacionarios existen más formas de reemplazo [71].

a) Reemplazar al peor de la población o *Replace Worst* (RW).

b) Torneo Restringido o *Restricted Tournament Selection* (RTS): Se seleccionan w ($w > 2$) individuos de forma aleatoria y se reemplaza al más parecido al descendiente.

c) Peor entre semejantes o *Worst Among Most Similar* (WAMS): Se seleccionan a los w ($w < 2$) padres más parecidos (de toda la población) al descendiente generado, y se reemplaza el peor de ellos.

d) Crowding Determinístico o *Deterministic Crowding* (DC): El hijo reemplaza a su padre más parecido.

2.7.7. Término del algoritmo

La evolución del AG termina cuando se logra un desempeño aceptable (o el óptimo), cuando se alcanza cierto número de generaciones, o cuando se supera una cantidad de tiempo definida. Otra alternativa es detener la evolución cuando ya no se presentan mejoras durante una cantidad de generaciones. Al terminar la evolución, se selecciona al mejor individuo de la población final, y éste corresponde a la solución entregada por el AG [8, 71].

Dada la aleatoriedad de los AG, se debe realizar una cantidad suficiente de evoluciones independientes para concluir sobre el desempeño del mismo. De esta forma, es posible calcular medidas estadísticas (medias, desviaciones, medianas, etc.) para evaluar y comparar entre distintos AGs, y con otros métodos.

2.7.8. Presión selectiva

Un concepto importante que se debe considerar en la implementación de los AGs es el de presión selectiva. La presión selectiva es un término informal que indica la fuerza de un mecanismo de selección [72]. Una presión selectiva alta se produce cuando el operador de selección requiere pocas iteraciones para que la mejor solución prevalezca en la población. Por otra parte, una presión selectiva baja se produce cuando el operador de selección toma una gran cantidad de iteraciones para que la mejor solución tome el control de la población [73]. En términos generales, la presión selectiva mide la relación entre la aptitud máxima y la aptitud media de la población [72].

La presión se relaciona directamente con la diversidad, que es un indicador de que tan distintas son los cromosomas de una población. Si un operador de selección tiene una gran presión selectiva, la población pierde diversidad rápidamente. Por lo tanto, para evitar una convergencia prematura a una solución incorrecta, se requiere una gran población u opera-

dores de mutación y *crossover* altamente disruptivos [73]. Por otra parte, un operador de selección con una presión selectiva demasiado baja no converge adecuadamente, haciendo la búsqueda inefectiva [74]. En cambio, una presión selectiva apropiada lleva a una convergencia lenta, permitiendo a los operadores de mutación y *crossover* explorar el espacio de forma adecuada al contar con una cantidad alta de iteraciones [73].

La presión selectiva guía la evolución, permitiendo su correcta convergencia y controlando el tiempo de evolución. Además, muchos de los parámetros de los algoritmos evolutivos (tamaño de población, número de generaciones, porcentaje de elitismo, etc.) ayudan a controlar la presión selectiva, por lo que la selección de estos parámetros es importante para una correcta convergencia y para un buen desempeño del algoritmo genético [75].

2.7.9. Hiper-parámetros importantes

Algunos de los parámetros más importantes de los algoritmos genéticos son aquellos que controlan la explotación y exploración del espacio de búsqueda. Por una parte, la mutación afecta directamente a la exploración, ya que permite alcanzar cualquier punto del espacio de búsqueda. Por lo tanto, la probabilidad de mutación controla el grado de exploración del AG: A mayor probabilidad de mutación, mayor es la exploración del espacio de búsqueda, lo que aumenta la diversidad. Al aumentar la diversidad también disminuye la presión selectiva, por lo que probabilidades de mutación muy altas pueden dificultar la convergencia del algoritmo.

Por su parte, el *crossover* es fundamental para la explotación del espacio de búsqueda, ya que, al realizar *crossover* entre dos individuos, el individuo resultante se ubicará en un espacio cercano a ambos, pero será distinto a los padres. Por lo tanto, una probabilidad baja de *crossover* impide que se explote el espacio de búsqueda, mientras que una probabilidad alta no permite que los padres pasen a la siguiente generación. La probabilidad de *crossover* también afecta la diversidad, ya que una baja probabilidad hará que los padres queden intactos en la siguiente generación.

El tamaño de la población y el número máximo de generaciones también afecta la evolución. Aumentar el tamaño de la población o el número de generación amplía la búsqueda, por lo que, al procesar más individuos, aumenta la probabilidad de llegar a mejores soluciones [76].

El tamaño del torneo y el porcentaje de elitismo son parámetros importantes que controlan la presión selectiva del AG. Si el tamaño del torneo es muy grande, los individuos que no estén en las mejores posiciones dentro del ranking tendrán una probabilidad muy baja de reproducirse [77]. Un caso extremo es cuando el tamaño del torneo es igual al tamaño de la población, en tal caso, siempre se elegirá al mejor individuo para reproducirse, generando una presión selectiva alta y provocando una convergencia prematura [77].

El porcentaje de elitismo es otro parámetro que afecta directamente la presión selectiva. El elitismo corresponde al porcentaje de los mejores individuos que se mantienen en la población luego de generar la descendencia. El elitismo puede conducir a mejoras rápidas en el fitness medio de la población, sin embargo, también puede conducir a una convergencia prematura ya que la población tiende a concentrarse rápidamente en el actual miembro más apto [77].

2.7.10. Selección de hiper-parámetros

Un aspecto importante que se debe considerar al implementar un algoritmo genético es cómo establecer los valores para los diversos parámetros, como el tamaño de la población, la probabilidad de *crossover* y la probabilidad de mutación. Estos parámetros generalmente se relacionan de forma no lineal, por lo que no pueden optimizarse de forma independiente [28]. La selección de parámetros de los AGs es un área de estudio y no hay consenso sobre el mejor método para dicha selección [28].

La forma más común de elegir estos parámetros es a prueba y error, sin embargo, este es un método que requiere mucho tiempo y esfuerzo [78]. Métodos más avanzados de selección de parámetros se basan en modelar el problema como uno de optimización, utilizando búsqueda aleatoria o incluso un segundo AG para optimizar dichos parámetros [79].

2.8. Espacio de Búsqueda de CNN

El espacio de búsqueda es el conjunto de todas las posibles soluciones de un problema de optimización o búsqueda. Para el problema de búsqueda de arquitecturas de CNNs, el espacio de búsqueda contiene todas las posibles arquitecturas, es decir, el tipo, número y parámetros internos de las capas de la red, así como las conexiones entre ellas. Además, es posible incluir hiperparámetros a este espacio, como el *learning rate*, ya que los hiperparámetros óptimos dependen de la arquitectura [8, 23].

Un buen espacio de búsqueda facilita la búsqueda de arquitecturas incluso para algoritmos de búsqueda simples como la búsqueda aleatoria, o *Random Search* (RS) [8, 23]. Por lo tanto, el espacio de búsqueda es fundamental para cualquier algoritmo de búsqueda, como los AGs [80]. En esta sección se describen los tipos de espacios de búsqueda más utilizados en la literatura.

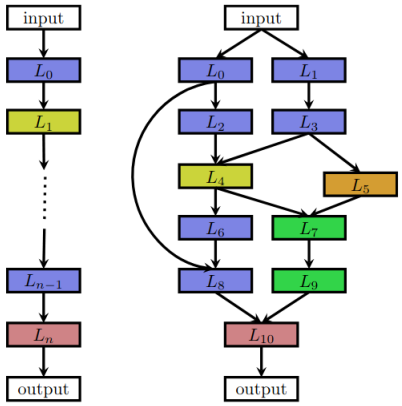


Figura 2.16: Ejemplos de espacios de búsqueda con conexiones en serie (izquierda), y con conexiones saltadas (derecha) (imagen tomada de [23]).

Uno de los espacios de búsqueda más simples es el que se muestra en la imagen izquierda de la Figura 2.16. En este espacio, las redes tienen un largo fijo de N capas, todas conectadas en serie. Además, cada capa tiene una operación asociada que se representa con un color: azul

indica capas convolucionales, amarillo indica capas de *pooling*, y rojo capas *fully connected* [23]. Un espacio más complejo se muestra en la imagen derecha de la Figura 2.16, donde las conexiones entre capas no son únicamente seriales, si no que una capa puede tener más de una entrada y/o salida, y la capa i-ésima puede tener como entrada a cualquier capa anterior [23].

Un nivel de complejidad mayor consiste en permitir que las redes tengan una cantidad indefinida de capas, es decir, un espacio de largo variable. Como el largo óptimo de la red es distinto dependiendo de cada problema, se pueden encontrar mejores soluciones que si el largo fuera fijo . Además, es el algoritmo de búsqueda quien debe encontrar el número óptimo de capas, y no el usuario o investigador [8]. En la Figura 2.17 se muestra un ejemplo de espacio de búsqueda serial de largo variable, donde los 3 cromosomas tienen una cantidad de capas distinta.

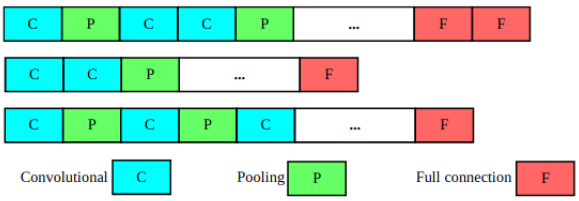


Figura 2.17: Espacio de búsqueda serial de largo variable (imagen tomada de [9]).

Espacios de búsqueda más recientes se basan en la estructura de bloques de las redes modernas, como ResNet y DenseNet [8]. En estos espacios, en lugar de buscar la arquitectura completa de la red se busca solo una celda de ésta. Luego, se generan celdas con la misma estructura y éstas se conectan entre sí para formar la red. Es común unir mas de una celda entre cada operación de pooling, formando un bloque [8]. Opcionalmente, se pueden buscar dos tipos de celdas, uno que mantenga la dimensión de entrada (*normal cell*), y otra que reduzca la dimensión de entrada (*reduction cell*). De esta forma, para construir la red se conectan celdas normales, separadas por celdas de reducción [24], como se muestra en la Figura 2.18.

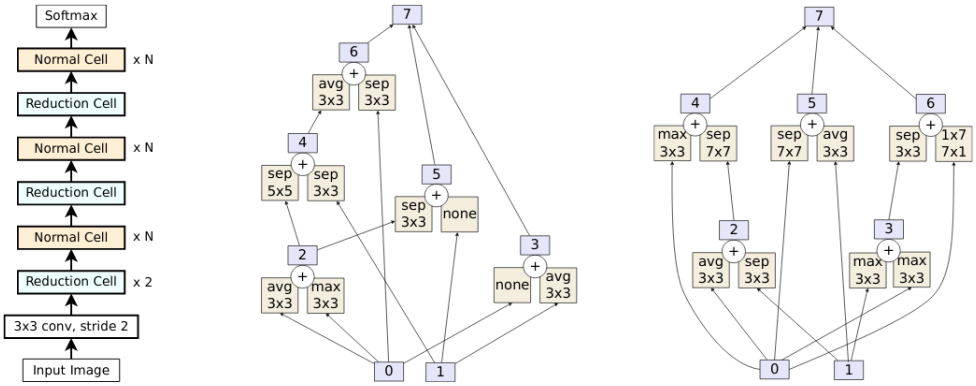


Figura 2.18: Espacio de búsqueda basado en celdas. La imagen izquierda corresponde a la red completa, la imagen derecha a una celda normal, y a la derecha una celda de reducción (imagen tomada de [24]).

El uso de celdas para formar el espacio de búsqueda requiere menos parámetros y reduce considerablemente el tamaño del espacio. Esto ya que, para espacios de arquitectura completa, la cantidad de posibles combinaciones de capas aumenta exponencialmente con la profundidad de la red. Sin embargo, para espacios de búsqueda basados en celdas, la profundidad de la red aumenta principalmente agregando mas celdas, lo que mantiene el número de posibles soluciones [8, 23].

A pesar de esto, el uso de celdas introduce nuevos parámetros a optimizar: el número de celdas y el número de bloques. Estos parámetros son los factores más importantes para determinar el tamaño de la red. Un enfoque utilizado para determinar la cantidad de celdas y bloques es primero encontrar una celda con buen desempeño, y luego buscar manualmente estos parámetros. Sin embargo, idealmente el número de celdas y el número de bloques debe ser buscado en conjunto con la celda, y con los hiperparámetros de entrenamiento [8, 23].

Capítulo 3

Revisión Bibliográfica en Optimización de Arquitecturas de Redes Neuronales

3.1. Métodos NAS

Diferentes estrategias de búsqueda se utilizan para explorar el espacio de las arquitecturas de CNNs. Los métodos mas populares son [23, 8] mediante *Reinforcement Learning* (RL), mediante Gradiente descendente (GD), mediante optimización bayesiana (OB), y mediante algoritmos evolutivos.

3.1.1. *Reinforcement Learning*

En el enfoque de RL, un controlador muestrea el espacio de búsqueda de CNNs para optimizar una recompensa. El controlador se retroalimenta con la recompensa para mejorar la toma de decisiones y elegir un modelo cada vez mejor. La recompensa se calcula entrenando el modelo muestreado desde cero y calculando su precisión en el conjunto validación [23, 81]. Un pionero en los métodos de RL fue Zoph *et al.* [80], quién utilizó una red neuronal recurrente como controlador y el algoritmo de *policy gradient* para entrenar este controlador. Otros enfoques como MetaQNN [82] entrenan al controlador usando Q-learning con una estrategia de exploración *e-greedy* y repetición de experiencias. Sin embargo, los métodos de RL tienden a requerir más recursos computacionales que los algoritmos evolutivos [9, 8]. Por ejemplo, Zoph *et al.* requirió 28 días y 800 GPU [80], mientras que MetaQNN requirió diez días y 10 GPU [82].

3.1.2. Gradiente Descendente

Los métodos basados en GD modelan arquitecturas de CNNs con variables continuas para optimizar la búsqueda usando gradiente descendente. En este enfoque, se construye un super-grafo o super-red que tiene un número predeterminado de nodos, pero cada nodo contiene todas las operaciones disponibles [23, 81]. El objetivo es encontrar un subgrafo con las conexiones óptimas entre las operaciones de los nodos. Uno de los trabajos basados en GD más icónicos es DARTS (*Differentiable Architecture Search*) [83], donde un conjunto de

variables α pondera cada conexión posible entre operaciones. Durante el entrenamiento, se optimizan los pesos de la red y los pesos α. Aunque este enfoque reduce el tiempo de búsqueda, la creación de un super-grafo requiere experiencia [8] y una GPU con suficiente memoria. Además, el tamaño del super-grafo crece linealmente con el número de operaciones [81], por lo que no es adecuado incluir múltiples operaciones en la búsqueda ni incluir parámetros para cada operación. Por lo tanto, los métodos evolutivos son preferibles cuando se tienen GPU más pequeñas, y cuando el espacio de búsqueda es complejo y con muchos parámetros para optimizar.

3.1.3. Optimización Bayesiana

La Optimización Bayesiana (OB) [84] es un método de optimización que no requiere un conocimiento explícito de la función objetivo (como los algoritmos genéticos), por lo que se usa ampliamente para la optimización de hiperparámetros de NNs y CNNs. El enfoque más común de los métodos bayesianos aplicados a NAS consiste en modelar la precisión en el conjunto de validación de las CNNs como procesos gaussianos [85, 86].Sin embargo, los métodos bayesianos no pueden manejar eficazmente codificaciones de largo variable, y el tiempo escala de forma cúbica con el número de observaciones [81].

Algunos de los trabajos más importantes que utilizan OB son AutoKeras [87] y AutoPytorch [88]. AutoKeras explora el espacio de búsqueda al transformar arquitecturas existentes, guiando dicha transformación mediante un algoritmo de optimización bayesiana. El procedimiento implementado en AutoKeras consiste en un ciclo de tres pasos: actualización, generación y observación. En el primer paso se entrena el modelo de proceso gaussiano con las arquitecturas existentes y sus rendimientos asociados. Luego, se genera la siguiente arquitectura en función de las previamente evaluadas. Finalmente, se obtiene el rendimiento de la arquitectura generada mediante un entrenamiento, y se repite el ciclo.

Por otra parte, AutoPytorch busca de forma simultánea los hiperparámetros y la arquitectura de redes convolucionales. Para esto, los autores utilizan *Bayesian Optimization & HyperBand* (BOHB), una combinación de optimización bayesiana e *HyperBand* (HB). El método HB consiste en asignar más recursos a las arquitecturas más prometedores al descartar las de peor desempeño. En AutoPytorch, los autores utilizan el algoritmo de HB para evaluar a un conjunto de arquitecturas, y usan optimización bayesiana para muestrear las siguientes arquitecturas, en vez de hacerlo de forma independiente o aleatoria. Los experimentos de AutoPytoch están enfocados principalmente en datos tabulares.

3.2. Evolución de Redes Neuronales

La Neuroevolución nació en la década de los 90 con trabajos que buscaban optimizar tanto la arquitectura de redes neuronales como sus pesos. A este enfoque se le denominó *Topology and Weight Evolving Artificial Neural Networks* (TWEANNs). Entre los autores que desarrollaron algoritmos de neuroevolución de este tipo están Miller *et al.* [42], y Yao *et al.* [59, 89]. Sin embargo, no fue hasta 2002 cuando Stanley y Miikkulainen desarrollaron NEAT [26], que la neuroevolución tuvo un avance importante en el desarrollo de redes neuronales. El algoritmo NEAT, por sus siglas en ingles *NeuroEvolution of Augmenting Topologies*, está enfocado en tareas de aprendizaje reforzado (*Reinforcement Learning*). Mediante una

codificación directa, donde cada peso de la red estaba representado en el cromosoma, NEAT implementa operaciones genéticas como *crossover* y mutaciones que permiten a la población evolucionar hacia mejores arquitecturas. NEAT fue incluso capaz de encontrar conexiones recurrentes (conexiones de un nodo con sigo mismo) que modela la derivada en una de las variables, elemento necesario para resolver el problema.

Pese al éxito que tuvo NEAT en optimizar redes neuronales en tareas de aprendizaje reforzado, tenía una limitante, y es que la codificación directa no permite codificar redes de gran tamaño. Por este motivo, en 2009 Stanley desarrollo HyperNEAT [90], una extensión de NEAT que realiza una codificación indirecta de la arquitectura. La codificación indirecta consiste en codificar reglas generativas de la arquitectura, y no la arquitectura en sí. De esta forma se logró evolucionar arquitecturas de redes neuronales de mayor tamaño. Pese a esto, aún con este enfoque no era posible evolucionar redes convolucionales profundas debido a su gran tamaño.

3.3. Evolución de Redes Convolucionales

Años más tarde, debido al éxito del gradiente descendente y del *back-propagation* en el entrenamiento de redes convolucionales, surgieron algoritmos evolutivos que difieren del enfoque TWEANN. En lugar de incluir los pesos de la red en la evolución, el nuevo enfoque consiste en entrenar la red en un conjunto de entrenamiento, evaluarla en un conjunto de validación, y utilizar dicha evaluación para calcular el *fitness* del individuo. Los primeros autores en utilizar este enfoque fueron Real *et al.* [27] y Liu *et al.* [18]. Real *et al.* codificaron las redes con una codificación global de largo variable e incluyeron el *learning rate* en ella. Sin embargo, solo utilizaron mutación como operador genético para generar variabilidad. Por su parte, Liu *et al.* codificaron las redes en tres niveles de forma jerárquica, donde el primer nivel codifica las operaciones, el segundo la conexión de dichas operaciones para formar un ensamble, y el tercer nivel que conecta distintos ensambles. Al igual que Real *et al.*, Liu *et al.* solo utilizó mutaciones como operador genético. Ambos obtuvieron resultados dentro del estado del arte en el problema de clasificación de imágenes aplicado al conjunto CIFAR-10 [91], pero requirieron una gran cantidad de tiempo y de recursos computacionales. Mientras que Real *et al.* demoraron 11 días en completar una evolución usando 250 GPUs en paralelo, Liu *et al.* utilizó 200 GPUs para evolucionar su algoritmo en 1.5 días.

El tiempo de evolución y los recursos computacionales requeridos por estos enfoques son enormes debido a que el cálculo del *fitness* de cada individuo es costoso. Este costo se multiplica por la cantidad de individuos de la población, y por la cantidad de generaciones del algoritmo genético. El costo del cálculo del *fitness* se debe principalmente al entrenamiento de las CNNs, que requiere el uso de GPUs y de un tiempo considerable para completarse. Por lo tanto, trabajos posteriores se enfocaron en disminuir el tiempo de entrenamiento de las CNNs , y así disminuir el cálculo del *fitness*, el costo computacional, y el tiempo de evolución.

3.3.1. Aproximaciones del *Fitness*

Los AGs calculan el *fitness* de cada individuo para ordenarlos y formar un *ranking*, el cual se utiliza para seleccionar a los padres para la reproducción (excepto en el método de selección por ruleta), y para eliminar individuos en la fase de reemplazo. Por lo tanto, si se usa

el método de selección por torneo u orden lineal, es más importante conocer el *ranking* de los individuos que un valor exacto del *fitness*. Basado en esto, surgió un interés por aproximar el *fitness* usando técnicas como: entrenar los modelos con menos épocas [9, 31], con menos datos [29], con imágenes de menor resolución [30], con menos celdas y mapas de características en los modelos evolucionados [24], o incluso combinando estas técnicas como en [92]. Dentro de estas técnicas, la más utilizada es entrenar las CNNs con menos épocas, es decir, obtener una red sub-entrenada con la cual evaluar el conjunto de validación.

3.3.2. Estado del Arte

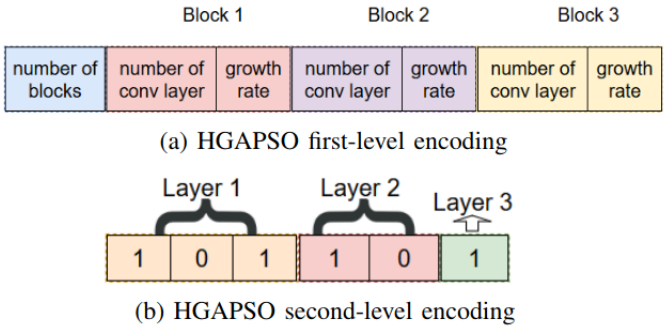
A continuación, se presentan distintos trabajos que abordan el problema de búsqueda de arquitecturas (e hiper-parámetros) de CNNs, donde las diferencias o avances principales están relacionadas con la creación de espacios de búsqueda, reducción del tiempo de cómputo del *fitness*, y modificaciones de AGs para el problema en cuestión.

Sun *et al.* desarrollaron evoCNN (*Evolving Convolutional Neural Networks*) [9], un algoritmo genético diseñado para encontrar automáticamente buenas arquitecturas. Las principales características de EvoCNN son: El diseño de un espacio de búsqueda, la implementación de una codificación de largo variable de arquitectura global, la implementación de operaciones genéticas *crossover* y mutación, y el cálculo eficiente del fitness al entrenar las redes por solo 5 o 10 épocas. El espacio de búsqueda en EvoCNN consiste en redes con solo 3 operaciones: Convolución, Pooling, y *fully-connected* (FC). Estas capas son conectadas en serie (no contempla *skip-connections*), y las capas FC son agrupadas a la salida de la red. Además, cada capa codifica alguno de sus parámetros: la capa de convolución codifica la cantidad de *feature maps*, la capa de pooling codifica el tipo (max-pooling o average-pooling), y la capa FC codifica su número de neuronas.

Wang *et al.* se basó en EvoCNN para crear HGAPSO (hybrid GA-PSO) [31]. HGAPSO implementa una codificación de bloques basado en DenseNet. Sin embargo, a diferencia del espacio de búsqueda basado en bloques descrito en el marco teórico, HGAPSO no repite el mismo bloque N veces, si no que busca N bloques distintos (ver Figura 3.1). En la codificación, la arquitectura es dividida en dos niveles: El primer nivel codifica la cantidad de capas convolucionales del bloque, la cantidad de bloques, y el *grouth rate*, mientras que el segundo nivel codifica las conexiones de cada capa. HGAPSO combina un algoritmo de *Particle Swarm Optimization* (PSO) y un algoritmo genético (AG) para encontrar la mejor arquitectura. Mientras que la arquitectura (primer nivel) es optimizada usando PSO, las conexiones (segundo nivel) son optimizadas utilizando el AG. Al igual que EvoCNN, HGAPSO entrena las redes durante 5 épocas para disminuir el tiempo de cómputo.

Sun *et al.* desarrollaron un algoritmo genético, llamado *Automatically Evolving CNN* (AE-CNN) [93], para encontrar automáticamente arquitecturas de CNN basadas en los bloques ResNet y DenseNet. En AE-CNN, las arquitecturas son de largo variable, es decir, pueden tener tantos bloques ResNet o DenseNet como sean necesarios. La codificación está compuesta por bloques o unidades, que pueden ser de 3 tipos: unidad ResNet, unidad DenseNet, o *pooling*. Si es una unidad ResNet o DenseNet, se codifican además al número de *feature maps* de entrada y salida, asi como la cantidad de bloques, como se muestra en la Figura 3.2.

Además, en AE-CNN proponen una crossover adecuado para codificaciones de largo va-



(a) HGAPSO first-level encoding

(b) HGAPSO second-level encoding

Figura 3.1: Codificación del espacio de búsqueda en HGAPSO (imagen tomada de [31]).

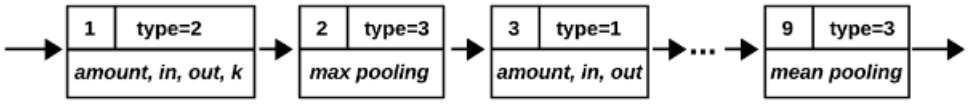


Figura 3.2: Ejemplo de la estrategia de codificación usada en AE-CNN (imagen tomada de [93]).

riable, lo que permite explorar localmente el espacio de búsqueda; así como una operación de mutación que permita modificar el largo de la red. La cantidad de individuos y el número de generaciones es igual a 20, y para el cálculo del *fitness*, cada individuo es entrenado durante 350 épocas. Dado que en AE-CNN no se utiliza algún método de aproximación del *fitness*, el tiempo de evolución alcanzó los 27 dias en el dataset CIFAR-10 [91].

Por otra parte parte, Sun *et al.* desarrollaron CNN-GA, un algoritmo genético para la búsqueda de CNNs. En CNN-GA los autores codificaron la red en bloques de 2 convoluciones, los que se conectan para formar la red global. La codificación es de largo variable, por lo que las redes pueden tener un número indefinido de bloques, pero incluye una cantidad finita de capas de *max-pooling*. Además, como se puede ver en la Figura 3.3, se incluyeron conexiones saltadas (*skip connections*) que van de un bloque al sub siguiente, siempre y cuando estos no estén separados por una capa de *pooling*.

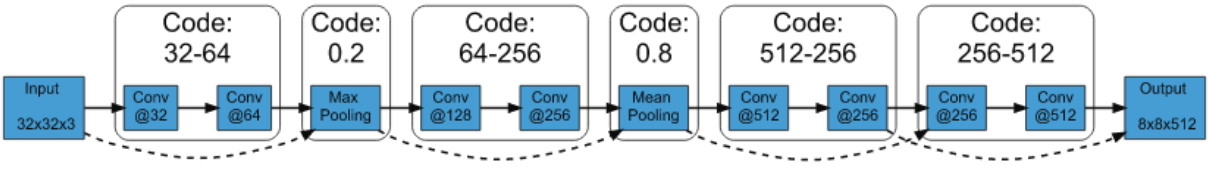


Figura 3.3: Ejemplo de la estrategia de codificación usada en CNN-GA (imagen tomada de [68]).

Al igual que para AE-CNN, en CNN-GA la cantidad de individuos y el número de generaciones se estableció en 20, y el cálculo del *fitness* de cada individuo se obtuvo con un entrenamiento de 350 épocas.

Capítulo 4

Metodología

La metodología propuesta contiene dos líneas de desarrollo orientadas a optimizar la evolución de arquitecturas de redes convolucionales. La primera consiste en un nuevo tipo de algoritmo genético, capaz de reducir los efectos negativos de utilizar aproximaciones del *fitness* mediante una combinación eficiente de entrenamientos cortos y entrenamientos largos. La segunda línea de desarrollo consiste en el diseño de un espacio de búsqueda flexible y con pocas restricciones. Este espacio de búsqueda está acompañado de una codificación que permite al AG realizar operaciones genéticas (mutación y *crossover*) sobre los individuos.

4.1. Descripción general del algoritmo genético de dos niveles propuesto

Como se mencionó anteriormente, la forma más intuitiva de evaluar el desempeño de una CNN para el cálculo del *fitness*, es entrenándola en un conjunto de entrenamiento y evaluando la red entrenada en un conjunto de validación. Este proceso es "ruidoso", ya que tanto la inicialización de los pesos como la aleatoriedad del entrenamiento mismo (muestreo, dropout, etc.) introducen variabilidad en cada entrenamiento. A este "ruido" se debe agregar el sesgo introducido por el sub-entrenamiento, ya que al entrenar la red durante menos épocas ésta no alcanza a converger, y el resultado será consistentemente peor que si fuera entrenada por completo.

Sin embargo, para el problema de búsqueda de arquitecturas mediante AG, lo importante no es el desempeño como tal, sino el ranking u orden relativo del desempeño de los individuos de una población. Por lo tanto, el sesgo introducido por el sub-entrenamiento no generaría problemas si el ranking de una población se mantiene al sub-entrenarla. Pese a esto, algunos autores demostraron que, para la misma población, el ranking obtenido con entrenamiento largo tiene una baja correlación con el ranking obtenido con entrenamiento corto. Más aún, dicha correlación disminuye a medida que se utilizan menos épocas en el entrenamiento corto [94].

De esta forma, el algoritmo evolutivo se ve afectado directamente por el compromiso entre fidelidad y tiempo de cómputo del *fitness*. Esto se debe a que una baja fidelidad produce un

ranking de mala calidad, mientras que un tiempo de cómputo elevado del *fitness* incrementa considerablemente el tiempo de evolución.

Para afrontar este problema, en este trabajo se propone un AG de dos niveles (*Two Level Genetic Algorithm* o 2LGA), donde se aplica un AG en cada nivel, cada uno con una población y operadores genéticos. En el primer nivel (1L), los individuos tienen un entrenamiento más corto para reducir el tiempo de búsqueda, mientras que en el segundo nivel (2L), los mejores individuos del primer nivel se evalúan de forma más precisa al entrenarlos durante más tiempo.

4.1.1. Algoritmo Genético de Dos Niveles

El algoritmo diseñado es un algoritmo genético generacional, por lo que en cada generación se evalúa a toda la población y se reemplaza a los peores. El algoritmo comienza creando al primer nivel (1L) con individuos generados al azar, permitiendo que cualquier solución del espacio de búsqueda pueda ser escogida. Entonces, este nivel evoluciona como un AG ordinario: calcula el *fitness* de toda la población con entrenamientos cortos, selecciona a los padres mediante selección por torneo, genera decendencia a través de los operadores genéticos *crossover* y mutación, y finalmente reemplaza parte de la población inicial por la descendencia. Este proceso se repite hasta que el segundo nivel es generado.

Para ahorrar tiempo de cómputo, el segundo nivel (2L) se evalúa solo cada K generaciones. Cuando se genera el 2L por primera vez en la generación K, se seleccionan a los N mejores individuos del 1L, donde N es el tamaño de la población del 2L. Una vez creada la población del 2L, se calcula su *fitness* de forma precisa, con entrenamiento largo, y comienza la reproducción cruzada.

En la reproducción cruzada, ambos niveles generan descendencia (Figura 4.1.a), luego, la descendencia del 2L reemplaza parte de la descendencia del 1L, mientras que los mejores de la población del 1L se convierten en la nueva descendencia del 2L (Figura 4.1.b). Finalmente ocurre el proceso de reemplazo genético, donde para cada nivel, la nueva descendencia reemplaza a los peores individuos de su respectiva población (Figura 4.1.c).

La reproducción cruzada tiene dos objetivos principales. El primero es darle una retroalimentación al 1L sobre la población del 2L, insertando los candidatos más prometedores (decendencia del 2L) en el 1L. En segundo lugar, se busca limitar el 2L solo a los mejores individuos del 1L. De esta forma, se evalúa con precisión a los individuos solo si demuestran ser buenos candidatos en el 1L, y no porque sean descendientes de buenos padres.

Una vez finalizado el proceso de reproducción cruzada, el 1L evoluciona como un AG ordinario durante $K - 1$ generaciones, mientras que el 2L se mantiene inalterado. Cada K generaciones, se evalúa el 2L y se repite el proceso de reproducción cruzada. La evolución termina cuando se alcanza un tiempo límite o un número máximo de generaciones. Al finalizar la evolución, la población del 2L se evalúa si hay individuos que no han sido entrenados y se escoge al individuo con mejor *fitness* de todas las generaciones.

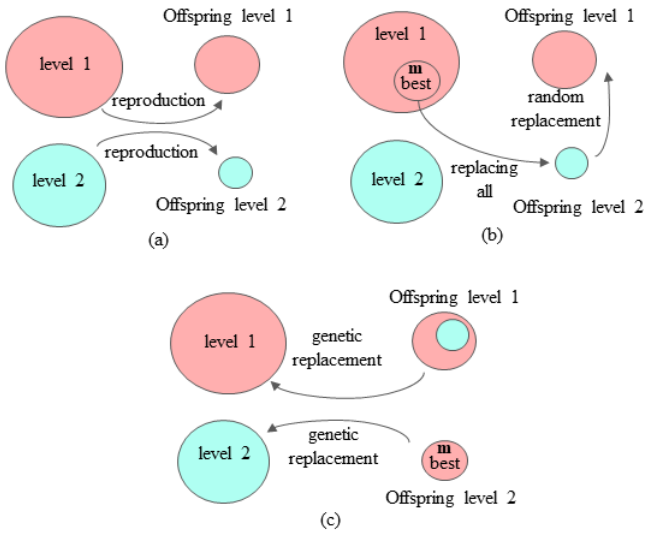


Figura 4.1: Proceso de reproducción cruzada del 2LGA. Primero, cada nivel genera su descendencia en (a). Luego, en (b), la descendencia del segundo nivel reemplaza parte de la descendencia del primer nivel, y luego los mejores individuos del primer nivel toman el lugar de la descendencia del segundo nivel. Finalmente, en cada nivel, la descendencia reemplaza a los peores individuos en (c).

4.1.2. Espacio de búsqueda

Según algunos autores, un espacio de búsqueda adecuado permite que los algoritmos de búsqueda encuentren buenas soluciones más fácilmente [18, 95, 96, 97]. Por otra parte, espacios de búsqueda reducidos restringen las posibles soluciones que el algoritmo genético puede encontrar. Por ejemplo, al usar un espacio de búsqueda serial, el algoritmo genético no podrá encontrar arquitecturas como ResNet o DenseNet que tienen conexiones saltadas. Otra restricción impuesta comúnmente es fijar los parámetros internos de las operaciones, por ejemplo, usando solo *kernels* de 3x3 o fijando el número de filtros en las capas de convolución. Otras restricciones del espacio de búsqueda incluyen el definir manualmente el número de celdas y de bloques, y definir una única forma de unión de dos nodos (suma o concatenación).

Por esta razón, en este trabajo se diseñó un espacio de búsqueda orientado a disminuir las restricciones descritas y reducir así el sesgo humano.

Caracterización del Espacio de Búsqueda

La estructura general del espacio de búsqueda propuesto consiste en una celda que se repite una cierta cantidad de veces para formar un bloque, los que a su vez se repiten para formar la red. Entre cada bloque se realiza una reducción de las dimensiones mediante una capa de *down sampling*. Además, se agrega una convolución inicial llamada *stem*, previa a los bloques. Finalmente, al último bloque se aplica un *Global Average Pooling*, o promediación en el eje de las filas y columnas, para obtener un vector de largo igual al número de canales. Este vector sirve como entrada de una capa *fully-connected* con función de activación *Softmax*, función usada típicamente en problemas de clasificación. En la Figura 4.2 (a) se muestra un ejemplo de la estructura general con 3 bloques y un número indefinido de celdas. En este

espacio, tanto el número de bloques como el número de celdas es variable, por lo que cambian para cada solución.

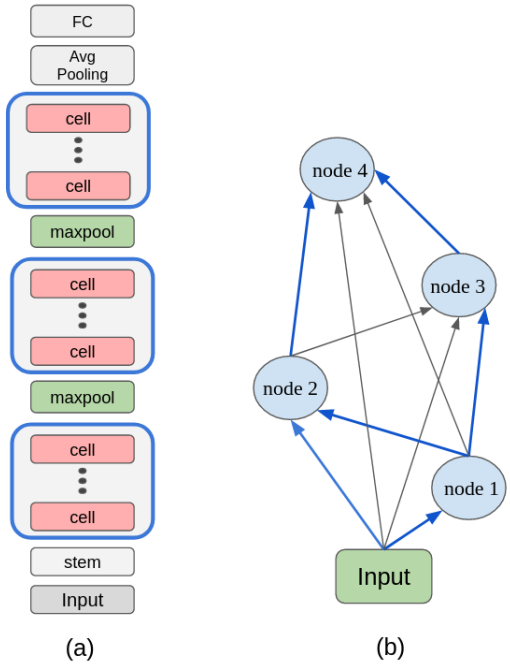


Figura 4.2: (a) Estructura general de las redes convolucionales en el espacio de búsqueda. Los bloques se representan por rectángulos azules, y están formados por un conjunto de celdas. (b) Estructura de la celda del espacio de búsqueda.

Por otra parte, cada celda contiene las operaciones a realizar y las conexiones entre ellas. La celda se define como un grafo acíclico directo o *Directed Acyclic Graph* (DAG), donde cada nodo representa una operación. Se dice que es acíclico directo ya que cada conexión se dirige de un nodo a otro, sin formar un bucle cerrado. En otras palabras, no se admite recurrencia en la red (como en una red neuronal recurrente). La cantidad posible de nodos es indefinida, y cada nodo puede conectarse con cualquiera de los nodos posteriores. Un ejemplo se muestra en la Figura 4.2 (b), donde cada flecha representa una posible conexión, y las flechas azules representan las conexiones efectivas de la celda.

Asimismo, en cada nodo se define la operación que se realizará, como convolución, *maxpooling* de *stride* 1, o identidad. Cada operación tiene sus propios parámetros, como el tamaño del *kernel*, que varían al moverse en el espacio de búsqueda. Además, como un nodo puede tener más de una entrada, en cada nodo con esta condición se define el método de unión, ya sea suma o concatenación. La operación convolución se compone de una capa de *Batch-Normalization*, convolución, y una función de activación. Los parámetros internos de cada operación, así como su dominio, se muestran en la Tabla 4.1.

Además de estos parámetros, se diseñó cada red CNN con un aumento gradual en el número de filtros, similar a PiramyNet [58]. Se define entonces la tasa de crecimiento o *growth-rate* (parámetro γ) que determina el aumento en los *feature-maps* en la salida de un bloque, respecto a los de entrada. De esta forma, el número de *feature-maps* de salida de un bloque viene dado por (4.1)

Tabla 4.1: Resumen de la información codificada en el espacio de búsqueda propuesto.

Description		Dominio
Parámetros Estructurales	factor de crecimiento	$\in [1.5, 4.5]$
	número de celdas	$\in \{1, 2\}$
	tamaño del *stem*	$\in \{32, 45\}$
Hiper parámetros	*warmup*	$\in [0, 0.5]$
	learning rate	$\in [0.002, 0.125]$
Parámetros de Nodo	operación	$\in$ {Conv, Identidad, Maxpool}
	método de unión	$\in$ {CAT, SUM}
Parámetros de Convolución	factor de feature-maps	$\in [0.1, 1.2]$
	activación	$\in$ {ReLU, ELU, PReLU}
	dropout	$\in [0, 0.6]$
	tamaño del kernel	$\in \{1, 3, 5\}$
Parámetros de Maxpool	tamaño del kernel	$\in \{2, 3, 5\}$

$$\mathrm{F_{out}}(\mathrm{block_i}) = \gamma \cdot \mathrm{F_{in}}(\mathrm{block_i}), \tag{4.1}$$

donde, $\mathrm{F_{out}}(\cdot)$ y $\mathrm{F_{in}}(\cdot)$ son funciones que representan el número de *feature-maps* de entrada y salida respectivamente. Dada la tasa de crecimiento, cada nodo que realiza una convolución tiene un número definido de *feature-maps* de salida dado por (4.2)

$$\mathrm{F_{out}}(\mathrm{node_k}) = F_0 \cdot (\gamma^{N_{\mathrm{blocks}}})^{k/N}, \tag{4.2}$$

donde $\mathrm{node_k}$ es el k-ésimo nodo, F_0 es el tamaño del *stem* (número inicial de *feature-maps*), N_{blocks} es el número de bloques de la red, y N es el número total de operaciones de la red. Así, $\gamma^{N_{\mathrm{blocks}}}$ es el crecimiento total de la red respecto al tamaño del *stem*. El parámetro γ también forma parte del espacio de búsqueda.

Con la configuración descrita, todas las convoluciones tienen un número definido de *feature-maps* de salida, que depende solo de la posición del nodo, y de la tasa de crecimiento γ, el número de celdas, y el número de bloques. Con el fin de agregar flexibilidad en la definición de la forma de la red, se agregó un segundo factor multiplicativo f_k en cada convolución. El factor f_k es multiplicado con (4.2) para obtener la cantidad de *feature-maps* de dicha convolución, según (4.3)

$$\mathrm{F_{out}}(\mathrm{node_k}) = f_k \cdot F_0 \cdot (\gamma^{N_{\mathrm{blocks}}})^{k/N}. \tag{4.3}$$

Con los parámetros f_k, el algoritmo de búsqueda puede ajustar la cantidad de *feature-maps* de cada convolución, teniendo a (4.2) como referencia. El rango de f_k, donde puede oscilar, es entre 0.1 y 1.2 (ver Tabla 4.1).

Por tanto, la forma de la red depende del número de celdas, el número de bloques, el tamaño del *stem*, la tasa de crecimiento y cada factor multiplicador f_k. Un aspecto a destacar es que para unir operaciones de diferente profundidad (mapas de características), se utiliza una convolución 1×1. Con esta convolución de 1×1 se proyecta el tensor de menor profundidad, incrementando sus *feature maps* e igualando al de mayor profundidad.

Por último, como hiperparámetros de entrenamiento, se agregaron a la búsqueda tanto el *learning rate* como el *warmup*. Mas detalles del entrenamiento y de estos parámetros son presentados en la sección 4.1.4 de cálculo del *fitness*. Todos los parámetros (hiperparámetros, parámetros estructurales, de cada operación, y de cada nodo) se resumen en la Tabla 4.1, donde además se muestra el rango o dominio asignado a cada uno.

4.1.3. Codificación

La codificación corresponde a la transformación de las soluciones del espacio de búsqueda en cromosomas, es decir, en un conjunto de variables. Para los algoritmos genéticos, dicha codificación debe permitir realizar mutaciones (alteraciones de uno o más genes del cromosoma), y *crossover* (combinación de la información genética de dos individuos distintos). Por un lado, la mutación permite explorar el espacio de búsqueda, ya que el gen alterado puede llegar a cualquier punto del espacio; y, por otro lado, el *crossover* ayuda a la explotación, o búsqueda local, del espacio de búsqueda, ya que el nuevo cromosoma se encontrará en un punto cercano a los padres.

Al igual que el espacio de búsqueda, la codificación se puede separar en dos partes, la codificación de la estructura general y la codificación de la celda. La codificación de la estructura general corresponde a una lista de largo fijo con una variable por cada parámetro estructural e hiperparámetro. Es decir, esta lista contiene: el número de celdas, el número de bloques, la tasa de crecimiento (γ), el *learning rate*, y *warmup*. En la Figura 4.3 se grafica la representación de un cromosoma, indicando en rojo la información de la estructura y en celeste la información de la celda.

Por otra parte, la codificación de la celda corresponde a una segunda lista, en la que cada elemento contiene la información de un nodo. Por lo tanto, esta lista es de largo variable, ya que mientras más nodos tenga la solución, mayor será el tamaño de dicha lista. A su vez, cada elemento de la lista codifica la información de su respectivo nodo, es decir, la operación a utilizar, las entradas, y el método de unión de las entradas (ver Tabla 4.1). Además, se codifican los parámetros internos de todas las posibles operaciones, y no solo de la operación utilizada en el nodo. En particular para las operaciones de la Tabla 4.1, se codifican los parámetros de la convolución y del *maxpooling* como se muestra en la Figura 4.3. De esta forma, la codificación admite genes que finalmente no se expresan en el fenotipo, o solución decodificada. Por ejemplo, si un nodo utiliza la operación de *maxpooling*, los parámetros de la convolución estarán presentes en el cromosoma, pese a no ser utilizados en la decodificación. Otro gen que puede no expresarse es el método de unión, ya que nodos con una sola entrada no lo requieren, pero dicha información se mantiene en el cromosoma.

Al permitir la existencia de genes que no se expresan en el cromosoma, se logran dos ventajas. En primer lugar, la información genética fluye con más facilidad a través de las generaciones. Esto ya que si un nodo muta de una operación a otra, los parámetros de

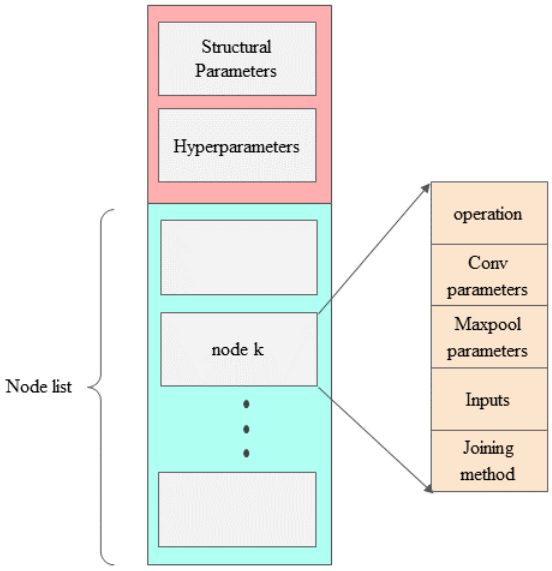


Figura 4.3: Representación del cromosoma de un individuo. El cromosoma tiene información de parámetros estructurales, hiperparámetros de entrenamiento y parámetros relacionados con cada nodo.

la operación inutilizada no se pierden y llegan a la nueva descendencia. En segundo lugar, facilita el *crossover* entre nodos de distintas operaciones, ya que al estar todos los parámetros presentes, se pueden combinar uno a uno. Por el contrario, si solo se codifica la operación expresada, al realizar crossover entre dos operaciones distintas solo se puede escoger una de ellas al azar.

Finalmente, cada nodo codifica las conexiones de entrada de la operación. Cada operación puede conectarse con cualquiera de las operaciones anteriores, y/o con la entrada de la celda. Por lo tanto, el nodo en la posición k-ésima puede tener 1 entrada como mínimo, y k entradas como máximo (todas las operaciones anteriores, más la entrada de la celda). Las conexiones se expresan como vectores binarios (0s y 1s), donde un valor 1 en la posición k-ésima significa una conexión con el k-ésimo nodo, siendo la posición $k = 0$ una conexión con la entrada de la celda. En la Figura 4.4 se muestra un ejemplo con una celda de 4 nodos. Un aspecto relevante de esta codificación y que influye en la operación de *crossover*, es que el vector de entradas del k-ésimo nodo siempre será de largo k.

Mutación

Bajo la codificación presentada, existen dos tipos de mutaciones: mutación de parámetros, y mutación de nodos. En la mutación de parámetros el cromosoma no altera su tamaño y solo se modifican los valores ya existentes en él. Para esto se selecciona un parámetro de forma aleatoria y se muta mediante mutación gaussiana [69] sí es real, o se elige un valor aleatorio entre las posibles opciones si el parámetro es cualitativo. En este tipo de mutación, todos los parámetros del cromosoma son modificables. Es decir, los hiperparámetros, parámetros estructurales, los parámetros de cada nodo, los parámetros de cada operación, y las conexiones

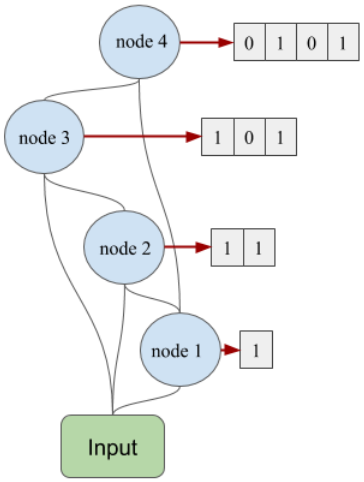


Figura 4.4: Ejemplo de codificación de las conexiones de una celda.

de cada nodo.

En el segundo tipo de mutación el tamaño del cromosoma si es modificable, ya que busca agregar nodos a la celda, o eliminar nodos existentes. En la adición de nodos, primero se selecciona un nodo existente de la lista de forma aleatoria. El nodo seleccionado se duplica, se muta, y se ubica justo antes del nodo original. Las entradas del nodo original se transfieren al nuevo gen y el nuevo gen se convierte en la única entrada del original. En la Figura 4.5 se muestra un ejemplo de este proceso.

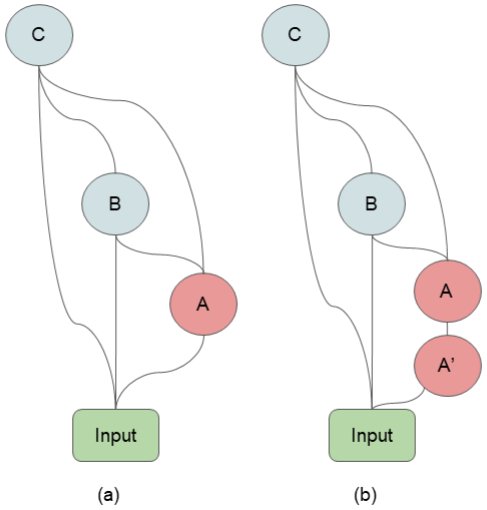


Figura 4.5: Esquema de adición de nodos en el proceso de mutación. Primero (a), se elige un nodo al azar (nodo rojo A). Luego, en (b), se duplica, se muta (A') y se agrega justo antes del original.

Al eliminar un nodo, como se muestra en la Fig. 4.6, se busca un nodo con una sola entrada y una sola salida para ser eliminado. Si existe un nodo con estas características, se elimina y se reemplaza por una conexión. Si no hay nodos con solo una entrada y una salida, se elimina

una conexión aleatoria de cualquier nodo, siempre y cuando el eliminar dicha conexión no deje a un nodo sin entradas o salidas. Con esta mutación se busca favorecer probabilísticamente el crecimiento de la red; además, el eliminar un nodo con más conexiones y/o entradas puede resultar en una solución completamente distinta a la original.

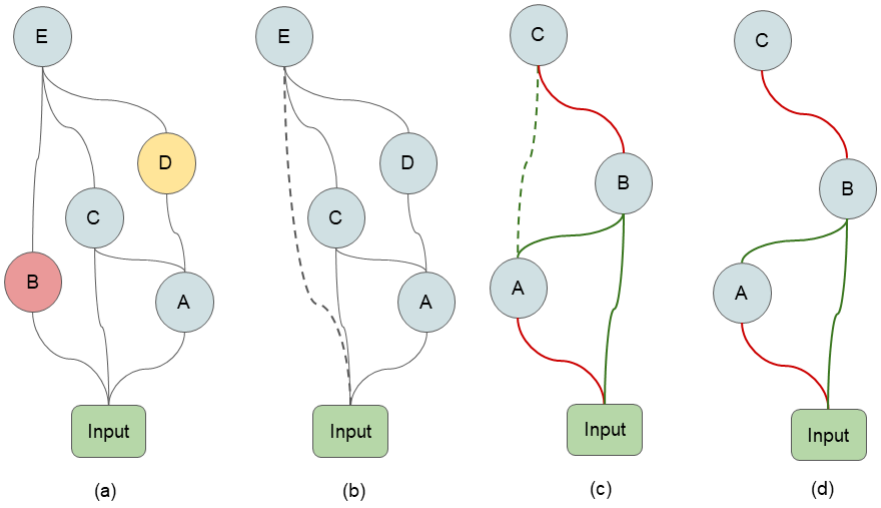


Figura 4.6: Esquema de eliminación de un nodo en el proceso de mutación. En (a), cuando hay nodos disponibles para eliminar (por ejemplo, los nodos B y D), uno de ellos se selecciona al azar (nodo B marcado en rojo). Luego, en (b), se elimina y se reemplaza con una conexión (línea punteada). Cuando no hay nodos disponibles para ser eliminados como en (c), se selecciona aleatoriamente una conexión desmontable (línea verde). La conexión seleccionada (línea verde punteada) se elimina, lo que da como resultado (d).

Crossover

Dado que la codificación tiene componentes de largo variable (nodos que conforman la celda) y componentes de largo fijo (hiperparámetros y parámetros estructurales), el *crossover* también tiene dos etapas. La primera etapa consiste en combinar las componentes de largo fijo mediante una combinación uno a uno. Es decir, para cada parámetro p, el parámetro resultante p_{hijo} es una función de los parámetros p de cada padre, p_{p1} y p_{p2}, de la forma:

$$p_{\text{hijo}} = f(p_{p1}, p_{p2}). \tag{4.4}$$

En particular, si el parámetro es cualitativo, se escoge una de ambas opciones al azar; en cambio, si el parámetro es real, un *crossover* aritmético completo (definido en 2.7) es utilizado.

Para realizar el *crossover* entre los nodos de la celda, primero se combinan los L primeros nodos de cada individuo, siendo L el menor tamaño entre los padres, como se muestra en la Figura 4.7. La combinación de los parámetros de los L primeros nodos se realiza uno a uno, al igual que el *crossover* de las componentes de largo fijo (con selección aleatoria o *crossover* aritmético completo). Es importante destacar que, debido a la codificación de genes no expresados, todos los nodos codifican la misma cantidad de información permitiendo

la combinación uno a uno entre todos los parámetros. Una excepción a esto el vector de conexiones de entrada, ya que el k-ésimo nodo codifica un vector de largo k (k posibles conexiones). Sin embargo, el k-ésimo nodo de un padre se combina solamente con el k-ésimo nodo del otro padre, por lo que ambos nodos tienen exactamente la misma cantidad de información (mismo largo). Finalmente, los nodos restantes del individuo de mayor tamaño son agregados al hijo con probabilidad 0,5.

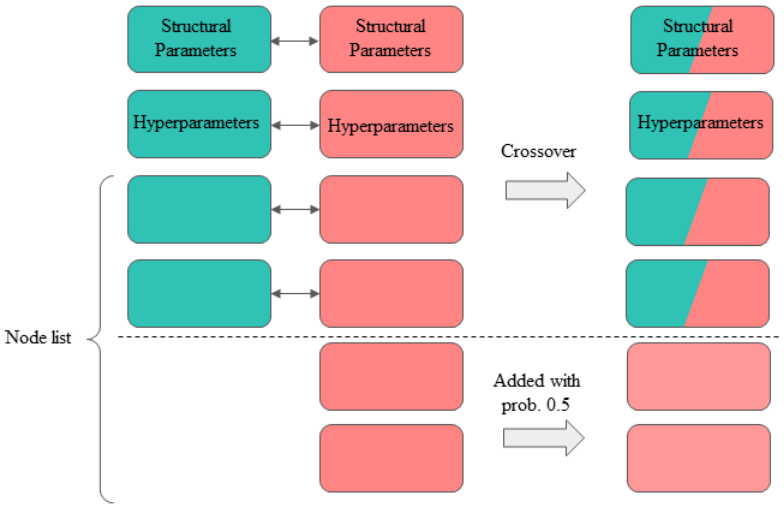


Figura 4.7: Ejemplo de *crossover* entre dos individuos de largo distinto.

Una alternativa a este *crossover* es seleccionar un punto al azar de cada individuo, alinear ambos individuos en este punto, y realizar la combinación nodo a nodo, pero siguiendo el alineamiento establecido por los puntos seleccionados. Sin embargo, esto presenta dos desventajas, la primera es que los nodos más cercanos a la entrada pueden tener características distintas a los más lejanos, por lo que combinar nodos en posiciones lejanas puede resultar en una mala comobinación de parámetros para dicha posición. El segundo aspecto es que los vectores de entradas (con 0s y 1s) de los nodos no tendrían el mismo largo, por lo que no sería posible combinarlos de forma directa.

4.1.4. Cálculo del *Fitness*

Para el cálculo del *fitness*, cada individuo es decodificado en una CNN, la cual es entrenada desde cero con el conjunto de entrenamiento. Al finalizar cada época del entrenamiento, se calcula el error en el conjunto de validación (*val-error*) y se almacena en un vector de memoria. Al finalizar el entrenamiento, el *val-error* mínimo del último 30 % de las épocas se obtiene para el cálculo del *fitness*. Además, se mide el tiempo de entrenamiento en segundos y se calcula el *fitness* mediante (4.5)

$$\text{Fitness} = \text{val}_{\text{error}} + \frac{\log(T_{\text{time}}/N_{\text{epochs}})}{1000}, \tag{4.5}$$

donde $\text{val}_{\text{error}}$ es el error de validación del último 30 % de las épocas, T_{time} es el tiempo total de entrenamiento, y N_{epochs} el número de épocas. Se debe notar que al dividir por N_{epochs},

lo relevante es el tiempo promedio por época en lugar del tiempo total. De esta forma, esta componente temporal no se ve afectada al entrenar con una cantidad distinta de épocas.

La componente temporal en el *fitness* se introdujo para evitar que el algoritmo genético busque redes con tiempos de entrenamiento prolongados que puedan aumentar demasiado el tiempo de evolución. La función logaritmo se utilizó para no penalizar demasiado a las redes más grandes, y el factor 1/1000 fue escogido para que la componente temporal tenga un orden de magnitud menor que el *val-error*.

En cada entrenamiento se usó un *schedule learning rate* basado en [98], donde el *learning rate* aumenta linealmente desde cero hasta un valor máximo en W épocas, y disminuye linealmente hasta cero en las épocas restantes. Al periodo en que el *learning rate* aumenta de cero al máximo se conoce como *warmup*.

En la codificación implementada, el *learning rate* codificado corresponde al máximo *learning rate*, mientras que el parámetro *warmup* codifica el número de épocas de *warmup* como una proporción del total de épocas. Si ω es la proporción codificada, la cantidad de épocas de *warmup* (W) se define según 4.6

$$W = \omega \cdot T, \tag{4.6}$$

con T el número total de épocas de entrenamiento.

Además, se utilizó SGD y Adam como optimizador, *label smoothing* [49] de 0.1 como regularización, y no se utilizó *data-augmentation* (DA) durante la evolución. Al igual que en trabajos anteriores, al finalizar la evolución se obtuvo al individuo de mejor *fitness* y se entrenó desde cero usando el conjunto de validación y de entrenamiento (combinados), y durante más épocas [20, 31, 99]. Al finalizar este entrenamiento, el error en el conjunto de test es reportado. Cade destacar que el conjunto de test solo se utiliza para evaluar al individuo ganador, y no es usado en ninguna etapa previa.

4.1.5. Implementación del método

En esta sección se presentan detalles de la implementación del método, en particular el entrenamiento distribuido en GPUs, así como el hardware y software utilizado para la realización de los experimentos.

Hardware y Software

El hardware utilizado para ejecutar los entrenamientos consta de un servidor con 4 GPUs NVIDIA RTX-2080Ti, 12 procesadores Intel(R) Xeon(R) Bronze de 1.70GHz, 64 Gb de memoria RAM, y 8Tb de memoria interna.

En cuanto al software, el sistema operativo corresponde a Ubuntu 18.04.4 LTS, el administrador de ambientes a Anaconda, y todos los códigos fueron escritos en Python 3. Por último, los modelos de redes convolucionales fueron creados y entrenados mediante la librería de aprendizaje automático Keras 2.4.3 con Tensorflow 2.2.0 como soporte.

Entrenamiento distribuido

Para optimizar el tiempo de evolución, se implementó un sistema de entrenamiento distribuido donde se utilizan todas las GPUs disponibles, de forma simultánea, para entrenar las CNNs.

Dado que el algoritmo genético es generacional y debe evaluar a toda la población en cada generación, el cálculo en paralelo del *fitness* de los individuos no afecta el comportamiento del AG. Esto se diferencia de un algoritmo genético estacionario, donde se debe generar, evaluar y reemplazar un individuo a la vez. Por otra parte, la disponibilidad de 4 GPUs posibilita el entrenamiento de 4 redes convolucionales en paralelo, permitiendo el cálculo del *fitness* de 4 individuos a la vez.

En primer lugar, al inicializar el algoritmo evolutivo se crea una tabla que almacena a cada individuo evaluado con su *fitness* respectivo. Entonces, si un individuo presente en la tabla debe ser evaluado nuevamente, se consulta dicha tabla y se retorna el valor del *fitness* almacenado sin necesidad de un nuevo entrenamiento. Este almacenamiento es principalmente usado al evaluar a los individuos que no fueron reemplazados, ya que se mantienen en la población debido al elitismo. Sin embargo, también es útil cuando un individuo se elimina de la población y vuelve a generarse de forma aleatoria (debido a mutaciones y/o *crossover*).

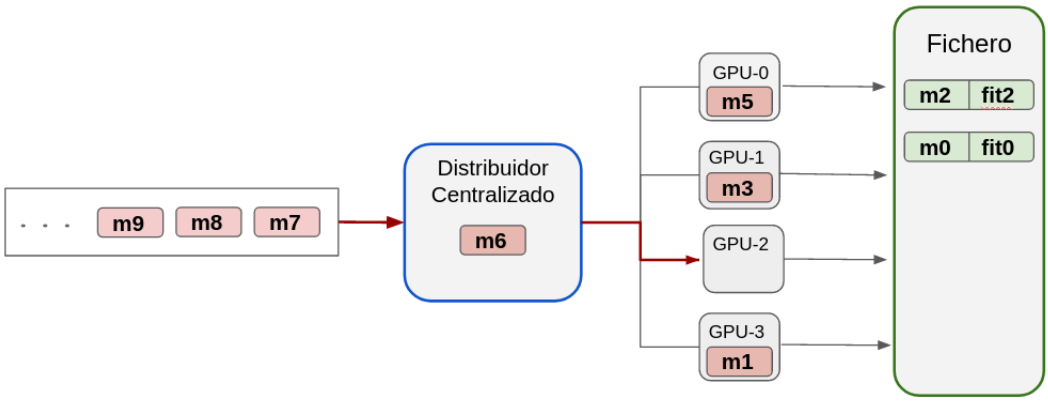


Figura 4.8: Esquema de evaluación distribuida de una población en 4 GPUs.

Al evaluar a una población, se consulta en la tabla de memoria y se obtiene el *fitness* de todos los individuos que ya habían sido evaluados. Luego, se crea una cola (*queue*) con los individuos restantes, que será utilizada como lista de espera. Posteriormente, el algoritmo genético actúa como un sistema centralizado, consultando la disponibilidad de GPU y distribuyendo los entrenamientos. A medida que se libera el uso de alguna GPU, el AG lanza hilos de procesamiento (o *threads*) en los que se evalúa el primer individuo de la cola, el cual es entonces eliminado. Al finalizar la evaluación o entrenamiento de cada individuo, el proceso respectivo se encarga de guardar los resultados en un fichero. Cuando todos los entrenamientos han finalizado, se lee el fichero para obtener el *fitness* de cada individuo. En la figura 4.8 se muestra un esquema, donde **mk** indica el k-ésimo modelo de la lista de espera, y donde la GPU-2 liberó su uso y será ocupada por el modelo número 6 (**m6**).

4.2. Algoritmo genético aplicado al reconocimiento de patrones en imágenes

Si bien el algoritmo genético propuesto se puede aplicar a una gran variedad de problemas como regresión o detección, se escogió realizar pruebas en problemas de reconocimiento de patrones en imágenes ya que éstos han sido ampliamente abordados por algoritmos neuroevolutivos [8, 9, 18, 20]. Para evaluar el sistema, se realizaron cuatro experimentos. En primer lugar, se ejecutó el sistema completo (2LGA + espacio de búsqueda) en siete problemas de clasificación de imágenes, y se compararon los resultados con competidores en el estado del arte. En segundo lugar, se comparó el 2LGA con dos algoritmos genéticos comunes, cada uno con un solo tipo de entrenamiento (corto o largo). En tercer lugar, se evaluó el espacio de búsqueda propuesto al implementar una búsqueda con dicho espacio y con el algoritmo de búsqueda más simple, *Random Search*; para luego comparar los resultados con algoritmos evolutivos de competidores que utilizan espacios de búsqueda distintos. Por último, se realizaron experimentos para estudiar el comportamiento del ranking de las poblaciones luego evaluarlas con un entrenamiento largo y con un entrenamiento corto.

En esta sección se presentan los datasests de clasificación de imágenes, los competidores con los mejores resultados publicados en dichos datasets, la configuración de parámetros, y una descripción de cada experimento.

4.2.1. Datasets

Se utilizaron siete datasets de reconocimiento de patrones en imágenes para evaluar el sistema. Los primeros 5 corresponden a MNIST-Variants [100] (MNIST-V), que provienen del ampliamente conocido MNIST [14], pero con ciertas modificaciones. Al igual que MNIST, MNIST-V contiene imágenes en escala de grises con números manuscritos del 0 al 9 (10 clases), con una resolución de 28×28 pixeles. El dataset de MNIST original contiene 70.000 imágenes en total (60.000 de entrenamiento y 10.000 de test). Sin embargo, los conjuntos de MNIST-V fueron redistribuidos para tener solo 12.000 imágenes de entrenamiento, y 50.000 de test, aumentando la dificultad respecto a MNIST.

Además, los conjuntos de MNIST-V tienen ciertas variaciones en las imágenes, exceptuando MNIST *Basic* (MB), que mantiene las imágenes originales pero con la redistribución descrita anteriormente. Dentro de los otros cojuntos, MNIST-*Random Background* (MRB) es MB con fondo aleatorio, MNIST *Background Images* (MBI) es MB con imágenes aleatorias de fondo, MNIST *Rotated Digits* (MRD) es MB con dígitos rotados, y MNIST *rotated digits and images as background* (MRDBI) corresponde a MB con imágenes aleatorias de fondo y con dígitos rotados. En la Tabla 4.2 se resumen las características de cada dataset, y en la Figura 4.9 se muestran ejemplos de cada uno de ellos.

El conjunto de datos MNIST-V es considerablemente más difícil que MNIST ya que las modificaciones dificultan la clasificación, y ya que cada conjunto tiene sólo el 20 % de las imágenes de entrenamiento de MNIST y cinco veces más imágenes de prueba. Con estas diferencias, los modelos requieren una mayor generalización para poder superar las modificaciones en base a menos ejemplos de cada uno de estos conjuntos.

Tabla 4.2: Descripción general de los conjuntos de datos utilizados para evaluar los métodos neuroevolutivos y al 2LGA.

Dataset	train-val	test	Descripción
MB	12,000	50,000	Redistribución de MNIST clásico
MBI	12,000	50,000	MB con imágenes de fondo
MRB	12,000	50,000	MB con ruido de fondo
MRD	12,000	50,000	MB con dígitos rotados
MRDBI	12,000	50,000	MB con dígitos rotados e imágenes de fondo
Fashion-MNIST	60,000	10,000	Productos de vestuario
CIFAR-10	50,000	10,000	Objetos comunes

El sexto dataset es Fashion-MNIST [101]. Este conjunto surgió como alternativa a MNIST, buscando ser un *benchmark* más complejo pero manteniendo el mismo tamaño. Fashion-MNIST contiene imágenes en escala de grises de productos de moda, separadas en 10 clases (camiseta, pantalón, suéter, vestido, abrigo, sandalia, camiseta, zapatilla, bolsa, y bota). Al igual que MNIST, Fashion-MNIST contiene 70.000 ejemplos (60.000 de entrenamiento y 10.000 de test), y las imágenes tienen un tamaño de 28×28 pixeles.

Figura 4.9: Ejemplo de los 7 datasets utilizados (tomados de [100, 101, 91]).

El séptimo conjunto de datos es el CIFAR-10 [91], un dataset con imágenes en color (3 canales) de objetos comunes (avión, automóvil, pájaro, gato, ciervo, perro, rana, caballo, embarcación, y camión). A diferencia de MNIST, CIFAR-10 tiene 50,000 imágenes de entrenamiento y 10,000 imágenes de test, y el tamaño de las imágenes es de 32×32 píxeles.

4.2.2. Competidores del estado del arte

Los competidores del estado del arte seleccionados son aquellos que presentaron resultados en alguno de los siete datasets descritos (ver sección 4.2.1). Principalmente se seleccionaron aquellos competidores que usaron algún tipo de algoritmo evolutivo para la búsqueda de arquitecturas de CNNs, sin embargo, también se incluyó trabajos que proponen arquitecturas desarrolladas manualmente.

Los competidores del estado del arte en los datasets de MNIST-V corresponden a PsoCNN [33], IPPSO [17], HGAPSO [31], DECNN [34], y evoCNN [9]. Los dos primeros, PsoCNN e IPPSO, utilizan PSO para optimizar la arquitectura de las CNNs. En DECNN y evoCNN, los autores utilizan Evolución Diferencial (DE) y Algoritmos Genéticos respectivamente. En HGAPSO, los autores implementan un método híbrido de dos niveles que combina PSO y AG para buscar las CNN óptimas. En HGAPSO un nivel codifica y optimiza los parámetros de las redes CNN, mientras que el otro nivel optimiza las conexiones entre capas. Pese a contar con dos niveles, HGAPSO se diferencia del 2LGA ya que este último tiene la misma codificación en ambos niveles, y la diferencia entre niveles radica en el cálculo del *fitness*. Por último, se realizaron experimentos utilizando la librería AutoKeras [87] en los datasets MB y MRDBI. Se realizó un total de cinco repeticiones para cada uno de los experimentos realizados con AutoKeras.

Para el conjunto de datos Fashion-MNIST, los mejores resultados de algoritmos evolutivos publicados anteriormente son: evoCNN [9], psoCNN [33], DeepSwarm [102] y REMNet [103]. Entre los trabajos no evolutivos con los mejores resultados están: WideResNet-28-10 [104, 105], GoogleNet [52], ResNet [106], DenseNet [56] y MobileNet [107]. Además, se incluyeron en la comparación los resultados de AutoKeras publicados [87]. En particular, AutoKeras AK corresponde al método base, mientras que AutoKeras AK-DP corresponde a una versión que utiliza varias técnicas avanzadas de entrenamiento para mejorar los resultados, incluyendo *schedule learning rate*, *cutout* [108], e inicialización de arquitecturas de forma manual.

La mayoría de los resultados de los métodos manuales utilizan un *data augmentation* (DA) básico para entrenar el modelo. Este DA básico consiste en agregar ceros a los bordes (*padding*), recortar una ventana de forma aleatoria (*Random Crop*, o RC), y realizar una reflexión horizontal de forma aleatoria (*Random Flip*, o RF). Por lo tanto, para los experimentos realizados en Fashion-MNIST, el individuo ganador se entrenó sin ningún tipo de DA y con el DA básico descrito anteriormente, y se registraron ambos resultados para una mejor comparación con los métodos manuales. Además, para una comparación justa, trabajos que utilizaron técnicas de DA más sofisticadas como *Random Erasing* [105] o *cutout* [108] no fueron incluidos en la comparación. No se realizaron comparaciones con técnicas más avanzadas de DA debido al tiempo requerido de entrenar con estas técnicas, y debido a que el objetivo es evaluar la capacidad de búsqueda de arquitecturas y no el como afectan las técnicas de DA.

Para CIFAR-10, se seleccionaron los métodos evolutivos (extraídos de [8]) que sólo usan un DA básico (*padding*, RC, y RF). Los métodos seleccionados son CNN-GA [68], AE-CNN [93], SOBA [109], Jhonson [110] y HGAPSO [31]. Para una comparación justa, otros métodos con técnicas más avanzadas de entrenamiento como *Scheduled-DropPath* [24, 111], *Auxiliary-Tower* [24, 111], *mixup* [5, 112], o *cutout* [5, 108, 111, 112, 113] no fueron incluidos. Al

igual que para Fashion-MNIST, se incluyeron en la comparación los resultados publicados en AutoKeras [87].

4.3. Diseño de Experimentos

Los experimentos se dividieron en 5 grupos de acuerdo al componente del sistema que se desea evaluar. A continuación, se detalla en que consiste cada uno de ellos.

1. **Comparación 2LGA con algoritmos genéticos comunes**: El primer experimento consistió en evaluar la efectividad del algoritmo genético de dos niveles propuesto. Dado que el 2LGA combina dos tipos de entrenamiento, es posible realizar dos comparaciones de forma directa: con un algoritmo genético común que solo haga entrenamientos cortos (AG-*short*), y con uno que solo haga entrenamientos largos (AG-*long*).

   El experimento consiste en ejecutar cada algoritmo de búsqueda (2LGA, AG-*short* y AG-*long*) un número determinado de veces, y registrar tanto el desempeño (test-error) como el tiempo de evolución de cada uno de ellos. Luego, con estos resultados se realiza una comparación de los promedios y desviaciones estándar de cada algoritmo, cuantificando dichas diferencias. Posteriormente se realizan dos pruebas estadísticas para validar (o rechazar) lo obtenido de la comparación anterior. En primer lugar, se realiza un test *Analysis of Variance* ANOVA [114] para determinar si las diferencias de test-error y tiempo de evolución son estadísticamente significativas. En segundo lugar, dado que el test ANOVA indica si existen diferencias entre los grupos, pero no entre cuales grupos, se utilizó el test HSD de Tukey (Tukey's *Honestly Significant Difference test*) [115] para encontrar cuales grupos son significativamente diferentes entre sí.

   Para permitir una comparación del tiempo, se ajustó en cada experimento la cantidad de individuos a evaluar. Por lo tanto, los algoritmos evolutivos de cada experimento evaluaron la misma cantidad de individuos. De esta forma, se espera que el AG-*short* complete la evolución en un menor tiempo, pero logrando un peor desempeño. Por otra parte, se espera que el AG-*long* logre un mejor desempeño, debido a que sus rankings son más confiables, pero en un tiempo de evolución mayor. Como el 2LGA combina ambos tipos de entrenamientos, el AG-*short* establece una cota inferior al tiempo de evolución, mientras que el AG-*long* establece una cota superior al desempeño que el 2LGA puede alcanzar. Tanto en este como en el resto de los experimentos, el desempeño equivale al error en el conjunto de test (test-error) del mejor individuo encontrado. Cabe señalar que el conjunto de test solo es usado para evaluar al individuo ganador, y no se utiliza en etapas previas.

   Estos experimentos se realizaron en el dataset más complejo de MNIST-V, MRDBI, ya que requiere menos capacidad de cómputo en los entrenamientos que Fashion-MNIST y CIFAR-10, pero es el de mayor dificultad en MNIST-V. Además, en todos los experimentos se usó el espacio de búsqueda propuesto, y todos los parámetros de entrenamiento y del AG se mantuvieron inalterados, exceptuando el número de épocas y el tamaño de la población. El tamaño de la población se ajustó en AG-*short* y AG-*long* para evaluar la misma cantidad de individuos que el 2LGA. Por último, cada experimento se repitió 5 veces para un posterior análisis estadístico.

2. **Comparación del espacio de búsqueda**: El segundo experimento consistió en comparar el espacio de búsqueda propuesto con el de los competidores del estado del arte. Para comparar el espacio de búsqueda, la forma más intuitiva es implementar los espacios de búsqueda con lo que se quiere comparar, y realizar experimentos con un solo algoritmo de búsqueda (un AG, por ejemplo). Sin embargo, esto requiere implementar no solo los espacios de búsqueda de los competidores, si no que la codificación y los operadores genéticos. Además de la dificultad que esto conlleva, en ocasiones es imposible reproducir de manera exacta lo presentado por los autores debido a la falta de detalles de implementación en los trabajos científicos. Una alternativa es implementar el algoritmo de búsqueda de cada competidor y ejecutarlo con el espacio de búsqueda propuesto, para luego comparar los resultados con los obtenidos de cada competidor. Así, las diferencias obtenidas son atribuibles al espacio de búsqueda únicamente, y es posible cuantificar su mejor o peor desempeño. Sin embargo, esto tiene las mismas limitaciones que el implementar los espacios de búsqueda de los competidores.

   En consiguiente, la forma de comparar el espacio de búsqueda fue la siguiente: implementar un algoritmo de búsqueda simple, utilizarlo con el espacio de búsqueda propuesto, y comparar los resultados con los competidores. Como los competidores utilizan algoritmos de búsqueda más sofisticados (AG o PSO), cualquier mejora respecto a ellos es atribuible al espacio de búsqueda.

   Por lo tanto, se implementó *Random Search* (RS), uno de los algoritmos de búsqueda más simples. Al igual que para el experimento de AG común, se realizaron dos experimentos con RS: uno con entrenamientos largos y otro con entrenamientos cortos. Los experimentos de RS fueron ejecutados hasta alcanzar un tiempo definido, tiempo elegido como el máximo tiempo de evolución del 2LGA para el mismo dataset. Además, los experimentos de RS optimizaron la misma función de *fitness* (4.5) que los AG, y el desempeño se midió calculando el test-error del individuo con mejor *fitness* encontrado.

   La creación de individuos aleatorios para RS se realizó mediante el siguiente procedimiento: Primero, se elige de forma aleatoria el número de nodos de la celda. El número posible de nodos es cualquiera entre 1 y 7. Para escoger el límite superior (7), se calculó el promedio de nodos de los individuos ganadores del 2LGA (igual a 6), y se sumó 1. En segundo lugar, para cada nodo se escogen aleatoriamente los parámetros del nodo, los parámetros de la operación convolución y los parámetros de la operación max-pool. Todos los parámetros son muestreados uniformemente dentro del dominio de cada parámetro (ver Tabla 4.1). En tercer lugar, para cada nodo en la posición k-ésima se generan conexiones aleatorias, con la restricción que todos los nodos tengan al menos una entrada y una salida. Finalmente, se escogen los parámetros estructurales e hiperparámetros de forma aleatoria, muestreando uniformemente dentro de los límites de cada parámetro (ver Tabla 4.1).

   Al igual que en los experimentos de AG comunes, se utilizó el dataset más complejo de MNIST-V, MRDBI. Además, se mantuvieron todos los parámetros de entrenamiento, excepto el número de épocas que varió dependiendo del experimento. Los experimentos de RS también se repitieron 5 veces para un análisis estadístico.

3. **Comparación con el estado del arte**: El tercer grupo de experimentos consistió en

ejecutar el sistema completo (2LGA y espacio de búsqueda propuesto) en los 7 datasets descritos en 4.2.1. Para los datasets MNIST-V y Fashion-MNIST, cada experimento se repitió 5 veces y se calculó el test-error promedio y máximo, al igual que los competidores. En cambio, debido al costo computacional requerido por los experimentos en CIFAR-10 (las imágenes son aproximadamente 4 veces más grandes, y tiene 5 veces más imágenes de entrenamiento que MNIST-V), solo se ejecutó un experimento en este dataset.

4. **Estudio de generalización**: El objetivo de este experimento es estudiar la capacidad de generalización de las soluciones encontradas, en conjuntos distintos al usado en la evolución. Es decir, se busca evaluar el desempeño de las arquitecturas encontradas en un dataset, luego de entrenarlas en datasets distintos. El experimento consiste en seleccionar la mejor arquitectura encontrada para cada dataset, y entrenarla desde cero en los demás datasets. En este experimento solo se consideraron los datasets MRDBI, Fashion-MNIST y CIFAR-10. El resto de los subconjuntos de MNIST-V (MB, MBI, MRB, y MRD) no se consideraron debido a la similitud que tienen con MRDBI, que corresponde a MB pero con dígitos rotados (similar a MRD) e imágenes de fondo (similar a MBI). Por otra parte, la cantidad de evaluaciones (entrenamientos) crece de forma cuadrática respecto a la cantidad de datasets.

   Dado que algunos parámetros de entrenamiento utilizados para evaluar los modelos ganadores en MRDBI y en Fashion-MNIST son distintos a los usados en CIFAR-10 (número de épocas, *data augmentation*, *label smoothing*), en cada dataset se entrenó con los parámetos de entrenamiento asociados al mismo.

5. **Análisis del ranking**: En este experimento se busca comparar el ranking obtenido mediante entrenamientos cortos con el obtenido mediante entrenamientos largos. Esta comparación se realiza para poblaciones completas y para los mejores de cada población.

   Para este experimento se extrajeron poblaciones de distintas generaciones de los experimentos de 2LGA aplicados en MRDBI. Entonces, cada población del 1L se entrenó con entrenamientos cortos y con entrenamientos largos, obteniendo así 2 rankings distintos por población (uno para cada tipo de entrenamiento). En la Figura 4.10 se muestra un ejemplo gráfico de este experimento, aplicado a una población. En este gráfico, el eje vertical corresponde a la posición dentro del ranking de cada individuo, y en el eje horizontal se encuentran los dos tipos de entrenamientos. Los individuos se representan con puntos, y las líneas conectan al mismo individuo, pero en rankings distintos. Mediante las líneas graficadas se puede observar los cambios de posición de los individuos. Por ejemplo, el mejor individuo de los entrenados con pocas épocas, es el octavo cuando se entrenan con más épocas.

   Luego, se calculó el coeficiente de correlación de Kendall [116], coeficiente que mide la correlación entre 2 rankings. Como cualquier coeficiente de correlación, si este es cercano a 1, los rankings se correlacionan de forma positiva (tienen un orden semejante), si el coeficiente es cercano a -1 la correlación es inversa (uno tiene un orden invertido al del otro), y si es cercano a 0 no hay una correlación entre ambos rankings.

   Posteriormente, se calculó el promedio de dichos coeficientes, calculado a partir de las

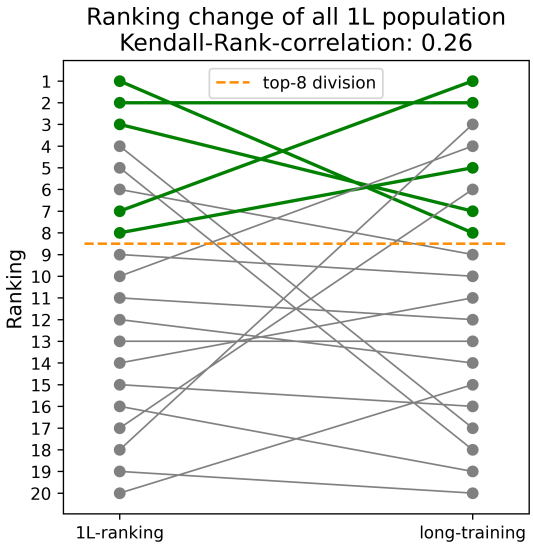


Figura 4.10: Cambio en el ranking la población del 1L cuando recibe entrenamiento corto (columna izquierda) y cuando recibe entrenamiento largo (columna derecha).

poblaciones pertenecientes a las generaciones donde se evalúa el 2L. Este experimento se repitió 5 veces, extrayendo las poblaciones de 5 ejecuciones independientes del 2LGA.

En segundo lugar, se estudió el ranking de los N mejores individuos. En este experimento, para cada población del 1L se extrajo a los N mejores individuos a partir del ranking obtenido con entrenamientos cortos (este proceso es el utilizado por el 2LGA para formar al 2L). Luego, se ordenaron estos N individuos usando entrenamientos cortos y usando entrenamientos largos, obteniendo nuevamente dos rankings (uno para cada tipo de entrenamiento). Un ejemplo de esto se muestra en la Figura 4.11, donde solo se consideran los N mejores individuos del 1L (con entrenamiento corto). Posteriormente se calculó el coeficiente de correlación de Kendall entre ambos rankings, y se promedió dicho coeficiente de la forma descrita anteriormente.

Finalmente, se midió el porcentaje de individuos que fueron correctamente asignados dentro del top-N. Es decir, de los N mejores individuos del ranking de entrenamiento corto, se calculó cuantos de ellos siguen perteneciendo al top-N en el ranking de entrenamiento largo. Estos individuos se representan con líneas verdes en la Figura 4.10, donde además se muestra en línea punteada al límite de los N mejores. El porcentaje de individuos correctamente asignados al top-N corresponde entonces, al porcentaje de individuos presentes en el top-N de ambos rankings respecto a N (con N el 100 %).

4.3.1. Configuración de parámetros

La configuración de parámetros puede dividirse en dos: parámetros del algoritmo genético y parámetros de entrenamiento de las redes convolucionales. A continuación, se detallan ambas configuraciones.

- **Parámetros del Algoritmo Genético**: Dado que el primer nivel del 2LGA actúa como un AG común, la selección de parámetros de este nivel se basó en los utilizados

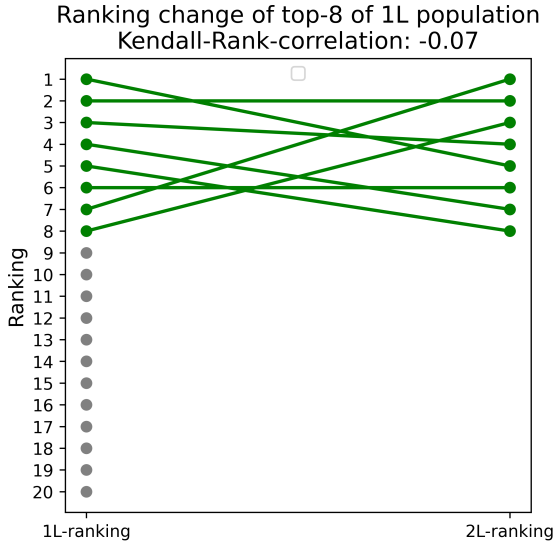


Figura 4.11: Cambio en el ranking de los 8 mejores de la población 1L cuando recibe entrenamiento corto (columna izquierda) y cuando recibe entrenamiento largo (columna derecha).

comúnmente por los competidores y en publicaciones anteriores [20, 68]. Tanto el tamaño de la población como la cantidad de generaciones fueron establecidas en 20; las probabilidades de *crossover* y de mutación fueron 0.9 y 0.1, respectivamente; el porcentaje de elitismo fue de un 25 % (similar a [9]), y el tamaño del torneo fue de un 25 % de la población [24]. Para el segundo nivel se usó una población de 8, un porcentaje de elitismo de 50 %, y el parámetro K fue establecido en 3, es decir, el 2L se evalúa cada 3 generaciones.

La elección de éstos parámetros se realizó en base a estudios anteriores [20, 68, 24], manteniendo un porcentaje de elitismo alto (25 %) para acelerar la convergencia, pero estableciendo probabilidades altas de mutación y *crossover* para mantener una diversidad alta, y evitar una convergencia prematura. Además, se realizó una sensibilización local de los parmámetros K (frecuencia de evaluación del 2L) y tamaño de población, en torno a los obtenidos de la literatura. Una alternativa para determinar la configuración de estos parámetros es optimizándolos con un segundo AG, donde cada individuo correspondería a un 2LGA con sus propios parámetros. Sin embargo, esto no se realizó ya que se requieren demasiados recursos computacionales, y ya que, los resultados obtenidos con la configuración descrita son competitivos con el estado del arte.

- **Parámetros de Entrenamiento de CNNs**: Para el entrenamiento de las CNNs se dividió el dataset (excluyendo el set de test), asignando un 20 % para validación y un 80 % para entrenamiento. Los individuos fueron entrenados con el conjunto de entrenamiento y se utilizó el error en el conjunto de validación para calcular *fitness*. El proceso de optimización se realizó mediante SGD y Adam, y tanto el *learning rate* como el *learning rate schedule* son los descritos en la seccion 4.1.4. Para los experimentos en MNIST-V y Fashion-MNIST, el *batch size* fue 128, el *label smoothing* fue 0.1, y el número de épocas del primer y segundo nivel fueron 18 y 54, respectivamente. Para los experimentos en CIFAR-10, el *batch size* fue de 96, y el número de épocas fue de 36 y

108 para el primer y segundo nivel. Al finalizar la evolución, se escogió al individuo de menor *fitness* y se entrenó con el conjunto de entrenamiento y validación combinados, y el error en el conjunto de test se registra en cada evolución. En los experimentos de MNIST-V y Fashion-MNIST, el individuo ganador se entrenó durante 90 épocas, mientras que en CIFAR-10 fue entrenado con 600 épocas, al igual que en [111]. Por último, para una mejor comparación con los competidores, en los experimentos de CIFAR-10 la componente temporal del *fitness* (4.5) fue eliminada, no se utilizó *label smoothing*, y el individuo gandor se entrenó con *data augmentation* (RF y RC).

Capítulo 5

Resultados y Análisis de Experimentos

En esta sección se presentan los resultados y el análisis de los experimentos descritos en la sección 4.3. Dichos experimentos son: comparación del 2LGA con dos algoritmos genéticos comunes, comparación del espacio de búsqueda mediante el uso de RS, comparación con los competidores del estado del arte, y resultados del estudio de los rankings de cada tipo de entrenamiento. Los resultados presentados a continuación corresponden al error en el conjunto de test obtenido por el mejor individuo, de cada experimento de búsqueda, conjunto que se utiliza únicamente para esta evaluación. Por otra parte, todos los experimentos, excepto el realizado en CIFAR-10, fueron repetidos 5 veces, y se reporta el promedio y el mejor resultado al igual que los competidores [9, 31, 33, 34].

5.1. Comparación con Algoritmo Genético Común

En la Tabla 5.1 se muestran los resultados de los experimentos realizados con el 2LGA, AG-*short* y AG-*long*. En esta tabla, se muestra para cada experimento el mejor test-error obtenido, el test-error promedio, su desviación estándar, el *fitness* promedio del individuo ganador, y el tiempo promedio de evolución. Estos resultados se complementan con los presentados en la Figura 5.1 donde se muestra un gráfico de caja, o box-plot, permitiendo una comparación visual de los resultados.

Tabla 5.1: Comparación de 2LGA con dos algoritmos genéticos ordinarios (O-GA).

Experimento	% Error (mejor)	% Error (media)	Fitness (media)	Tiempo de evolución promedio (hrs.)
AG-*short* (18 eps)	7.12	7.34	7.93	5.8
AG-*long* (54 eps)	6.45	6.78	7.05	19.9
2LGA (18, 54 eps)	6.33	6.73	7.13	6.6

Como se puede observar, el AG-*short* obtiene el peor error promedio, pero los resultados se alcanzan en el menor tiempo (5.8 horas). En contraste, el AG-*long*, logra un mejor test-error, pero demorando casi 20 horas en promedio. Al comparar el AG-*short* con el AG-*long*, los resultados son los esperados. Es decir, dado que el AG-*long* utiliza un entrenamiento con

3 veces más épocas (54 vs 18 del AG-*short*), el ranking que obtiene en cada generación es más confiable y por lo tanto obtiene un test-error menor. Sin embargo, el AG-*long* demora aproximadamente 3 veces más, ya que los entrenamientos son 3 veces más largos. Por su parte, el 2LGA alcanza los mismos resultados que el AG-*long*, pero demorando un tiempo similar al AG-*short*. De la misma forma, esto se aprecia en la Figura 5.1, donde el 2LGA tiene valore cercanos al AG-*long* en (a), y más cercanos al AG-*short* en (b).

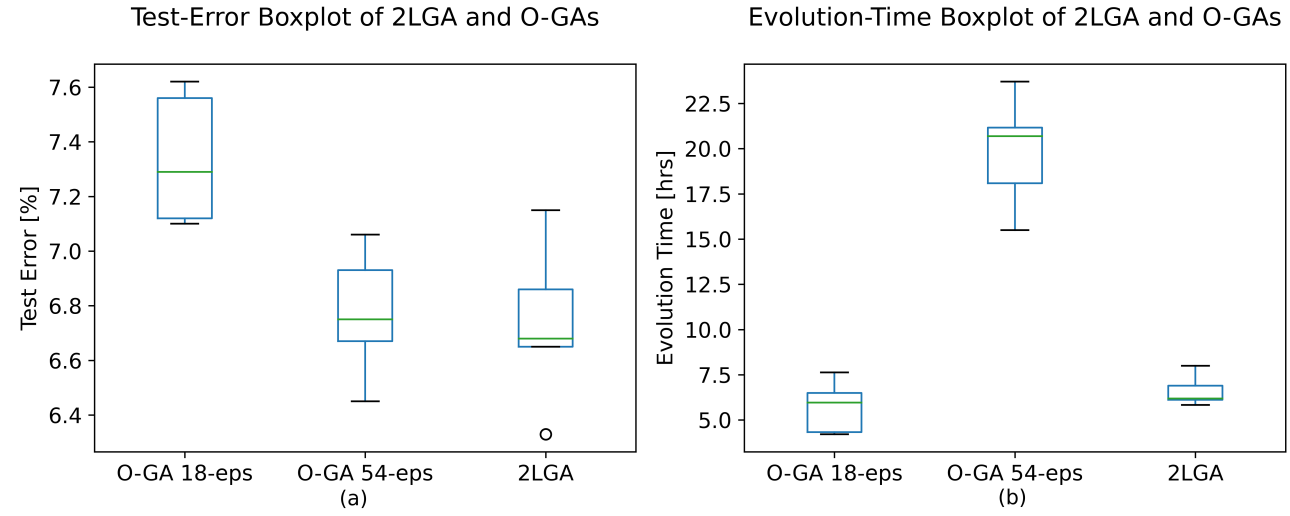


Figura 5.1: Box-plots de los resultados de experimentos O-GA: (a) test error, y (b) tiempo de evolución. Cada experimento se repitió 5 veces en el conjunto de datos MRDBI.

Para corroborar lo señalado, se realizó un test ANOVA el cual resultó en un F-value = 8.37 y un p-value = 0.005 para el test-error, y un F-value = 73.6 y un p-value = 1,8e-7 para el tiempo de evolución. Por lo tanto, al ser ambos p-values menor a 0.05, se determinó que existen diferencias significativas entre los experimentos, tanto para el error de prueba como para el tiempo de evolución.

Para la prueba de Tukey HSD, se comparó el test-error y el tiempo de evolución de los experimentos, al igual que para el test ANOVA. Los resultados del Tukey HSD aplicado al test-error se muestran en la Tabla 5.2, mientras que la Tabla 5.3 muestra los resultados para el tiempo de evolución. De estos resultados se puede observar lo siguiente:

- Los resultados del test HSD de Tukey en el test-error (Tabla 5.2) muestran que el error del 2LGA es significativamente diferente al error del AG-*short*, pero no es distinto al error del AG-*long*.

- Los resultados del test HSD de Tukey HSD en el tiempo de evolución (Tabla 5.3) indican que el tiempo de evolución del 2LGA es significativamente diferente al del AG-*long*, pero no es diferente al del AG-*short*.

Por lo tanto, en estas pruebas realizadas, *el 2LGA logra el mismo rendimiento que un algoritmo genético con entrenamientos largos (54 épocas), pero tomando un tiempo similar a un algoritmo genético con entrenamientos cortos (18 épocas).*

Tabla 5.2: Resultados del test HSD de Tukey para el error de test de dos algoritmos genéticos comunes (AG-*short* y AG-*long*) y el 2LGA.

Grupo1	Grupo2	Diferencia de medias (% error)	Rechazar
AG-*short*	AG-*long*	-0.566	True
AG-*short*	2LGA	-0.604	True
AG-*long*	2LGA	-0.038	False

Tabla 5.3: Resultados del test HSD de Tukey para el tiempo de evolución de dos algoritmos genéticos comunes (AG-*short* y AG-*long*) y el 2LGA.

Grupo1	Grupo2	Diferencia de medias (hrs.)	Rechazar
AG-*short*	AG-*long*	14.1	True
AG-*short*	2LGA	-0.9	False
AG-*long*	2LGA	-13.2	True

5.2. Comparación de Espacio de Búsqueda

Los resultados de los experimentos de RS se muestran en la Tabla 5.4, donde además se muestran los resultados de los competidores que utilizan algoritmos evolutivos para encontrar arquitecturas de forma automática, aplicados al dataset MRDBI.

Tabla 5.4: Comparación de experimentos de RS con los competidores. Los competidores utilizan algoritmos evolutivos y espacios de búsqueda propios, mientras que RS utiliza el espacio de búsqueda propuesto. La desviación estándar de cada método se muestra entre paréntesis cuando está disponible.

Experimento	Error de test % (mejor)	Error de test % (media)
PsoCNN [33]	14.28	20.98
IPPSO [17]	34.50	33.00 (2.96)
HGAPSO [31]	10.53	12.23 (0.86)
DECNN [34]	32.85	37.55 (2.45)
EvoCNN [9]	35.03	37.38 (1.75)
RS (18 eps)	8.13	8.42 (0.73)
RS (54 eps)	7.04	7.25 (0.34)

En la Tabla 5.4 se puede notar que ambos experimentos de RS obtienen considerables mejoras respecto a los competidores. En particular, respecto al competidor con un menor error promedio (HGAPSO), RS disminuye el error en un 4.98 % cuanto se entrenan los individuos con 54 épocas, y en un 3.81 % al entrenar con 18 épocas (se debe notar que los resultados se obtienen luego de entrenar al individuo ganador durante 90 épocas). Como se mencionó anteriormente, como RS es uno de los algoritmos de búsqueda más simples, es posible afirmar que el espacio de búsqueda propuesto es quién otorga las mejoras.

Por otra parte, en promedio, RS con 54 épocas logró 1.17 % menos error que RS con 18

épocas, por lo que si se generan diferencias al entrenar con más (o menos) épocas. Pese a que RS no utiliza explícitamente un ranking en la búsqueda, si lo hace de forma indirecta al almacenar al mejor individuo evaluado hasta entonces. El peor desempeño de RS+18 se debe a que, al calcular el *fitness* con menos fidelidad, es menos probable que el mejor individuo entrenado en estas condiciones sea el mejor al entrenarlo por completo.

En la Tabla 5.5, se muestra una comparación de los algoritmos RS con los algoritmos genéticos implementados, todos usando el espacio de búsqeuda propuesto. Un aspecto relevante es que, al comparar los métodos que utilizan el espacio de búsqueda propuesto (ver Tabla 5.5), los métodos evolutivos no superan a RS con un amplio margen. Al comparar los métodos que usan la misma cantidad de épocas de entrenamiento, el AG-*short* supera en 1.08 % a RS + 18, mientras que el AG-*long* supera en 0.47 % a RS + 54. Esto respalda la hipótesis que RS es una base difícil de superar, a pesar de ser uno de los algoritmos de búsqueda más simples [18, 95, 96, 97].

Tabla 5.5: Comparación de experimentos de RS con los algoritmos genéticos implementados. Todos los experimentos utilizan el espacio de búsqueda propuesto.

Experimento	Error de test % (mejor)	Error de test % (media)	Tiempo de evolución (hrs.)
AG-*short* (18 eps)	7.12	7.34	5.8
AG-*long* (54 eps)	6.45	6.78	19.9
2LGA (18, 54 eps)	6.33	6.73	6.6
RS (18 eps)	8.13	8.42	9.1
RS (54 eps)	7.04	7.25	9.2

Además, la diferencia entre los AG y RS pasó de 1.08 % al comparar ambos métodos con 18 épocas, a 0.47 % con 54. Por lo que la diferencia entre AG y RS disminuye a medida que aumentan las épocas de entrenamiento. Más aún, RS+54 logra un desempeño levemente superior al AG-*short* (0.09 % menor test-error). Esto sugiere que la cantidad de épocas de entrenamiento, o la fiabilidad del fitness, es uno de los factores más importantes para obtener buenos resultados, y que los algoritmos de búsqueda se comportan de forma similar al evaluar con suficiente fiabilidad [1].

5.3. Comparación con el estado del arte

5.3.1. Comparación en MNIST-V

En la Tabla 5.6 se muestra una comparación de distintos métodos evolutivos de búsqueda de arquitecturas de CNN, y AuoKeras, aplicados a los datasets MNIST-V. Como se puede ver en esta tabla, el 2LGA logró resultados significativamente mejores que los publicados anteriormente. Al comparar el test-error promedio de 2LGA con el del mejor competidor por cada dataset , el 2LGA logró una mejora de 0.19 en MB (vs HGAPSO), de 1.16 en MRB (vs PsoCNN), de 1.06 en MBI (vs PsoCNN), de 3.03 en MRD (vs EvoCNN), y de 3.3 en MRDBI (vs AutoKeras). Como las mejores tienen un orden similar a los resultados previos,

[1] Un estudio más acabado se requiere para avalar o refutar esta hipótesis, lo que abre nuevas líneas de investigación en torno a estos resultados.

es apropiado representar dicha mejora en términos relativos. Porcentualmente, la mejora del 2LGA respecto al mejor competidor es de 22.6 % en MB (vs HGAPSO), de 45.8 % en MRB (vs PsoCNN), de 44.2 % en MBI (vs PsoCNN), 55.5 % en MRD (vs EvoCNN), y 33.2 % en MRDBI (vs AutoKeras). En resumen, el método propuesto mejoró los resultados previos en un 42.6 % en promedio. Además, para el dataset más difícil, MRDBI, la mejora en el test-error fue la mayor en térmminos absolutos, alcanzando un error de test 21.5 % más bajo que el promedio de los competidores, y superándolos ampliamente.

Tabla 5.6: Comparación de los errores de test entre 2LGA y los mejores resultados publicados en MNIST-V (más bajo es mejor). La desviación estándar de cada método se muestra entre paréntesis cuando está disponible.

Método	MB	MRB	MBI	MRD	MRDBI
AutoKeras[a] (mean) [87]	0.85 (0.10)	–	–	–	10.07 (1.20)
AutoKeras[a] (best) [87]	0.67	–	–	–	8.10
IPPSO (mean) [17]	1.21 (0.10)	–	–	–	33.00 (2.96)
IPPSO (best) [17]	1.13	–	–	–	34.50
HGAPSO (mean) [31]	0.84 (0.07)	–	–	–	12.23 (0.86)
HGAPSO (best) [31]	0.74	–	–	–	10.53
DECNN (mean) [34]	1.46 (0.11)	3.56 (1.71)	8.69 (1.41)	5.53 (0.45)	37.55 (2.45)
DECNN (best) [34]	1.03	3.46	5.67	4.07	32.85
EvoCNN (mean) [9]	1.28 (0.15)	3.59 (0.89)	4.62 (0.12)	5.46 (0.25)	37.38 (1.75)
EvoCNN (best) [9]	1.18	2.80	4.53	5.22	35.03
PsoCNN (mean) [33]	–	2.53	2.40	6.42	20.98
PsoCNN (best) [33]	–	1.79	1.90	3.58	14.28
Ours 2LGA (mean)	**0.65** (0.05)	**1.37** (0.13)	**1.34** (0.13)	**2.43** (0.25)	**6.73** (0.27)
Ours 2LGA (best)	0.61	1.35	1.26	2.26	6.33

[a]Resultados obtenidos experimentalmente.

Para estudiar cómo evoluciona el algoritmo genético a lo largo de las generaciones, se graficó la curva de evolución del 2LGA para cada uno de los datasets de MNIST-V. En la Figura 5.2 se muestran las curvas de evolución en el dataset MRDBI, mientras que las curvas de los dataset MB, MRB, MBI y MRD se muestran en al Apéndice A.

Como se realizaron 5 experimentos independientes para cada dataset, se obtuvieron 5 curvas de evolución distintas. La Figura A.5 se compone de tres gráficos. En (a) se grafica la evolución del promedio del *fitness* de la población del 1L, donde además se muestra la desviación estándar. En (b) se muestra la evolución del mejor *fitness* de la población en cada generación. Por último, en (c) se muestran tanto la evolución del *fitness* promedio de la población (azul), como la evolución del menor *fitness* (rojo).

Como se observa en (a), el *fitness* promedio inicia con valores cercanos a 0.5, lo que equivale a un 50 % de error en el set de validación aproximadamente. Se desprecia la componente temporal del *fitness*, ya que fue diseñada para tener un orden de magnitud menor que el error de validación. El elevado fitness se debe a que la población del 2LGA se inicializa con arquitecturas aleatorias, existiendo algunas que no convergen durante su entrenamiento. En las siguientes 5 generaciones, el *fitness* desciende hasta alcanzar valores cercanos a 0.15,

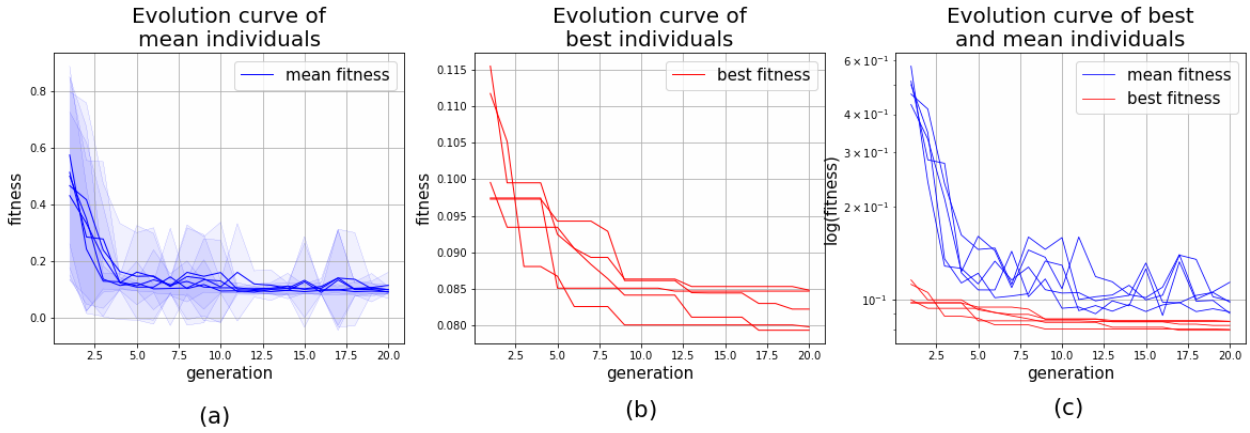


Figura 5.2: Curvas de evolución de experimentos realizados en dataset MRDBI. Evolución del *fitness* promedio de la población (a), evolución del mejor *fitness* de la población (b), y comparación, en escala logarítmica, de la evolución del promedio (azul) y del mejor (rojo) *fitness* de la población (c).

pero manteniendo una variabilidad (desviación estándar) alta. Entonces, el *fitness* promedio desciende aún más hasta alcanzar valores menores a 0.1 como se observa en la Figura A.5 (c).

Por otra parte, en la Figura A.5 (b) se aprecia un descenso del mejor *fitness* al transcurrir las generaciones, siendo estos descensos cada vez menores debido a la convergencia del algoritmo genético. Por último, en la Figura A.5 (c) se puede notar que el *fitness* promedio mantiene una distancia relativamente constante respecto al mejor *fitness*, lo que indica que el algoritmo no converge a soluciones demasiado similares en el transcurso de la evolución, y que no hay una convergencia prematura del mismo.

Pese a que solo se describen las curvas de evolución correspondientes al dataset MRDBI, esta descripción también es aplicable al resto de los dataset, ya que como se observa en las Figuras A.1, A.2, A.3 y A.4, el comportamiento de dichas curvas es similar al descrito y presentado en la Figura 5.2.

Finalmente, en la Figura 5.3 se muestra la arquitectura con los mejores resultados para el conjunto de datos MRDBI. Esta red logró un error en el conjunto de test de 6.33 % (ver Tabla 5.6), y el experimento que encontró esta arquitectura tardó 6.9 horas en finalizar, incluyendo el tiempo de entrenamiento del individuo ganador.

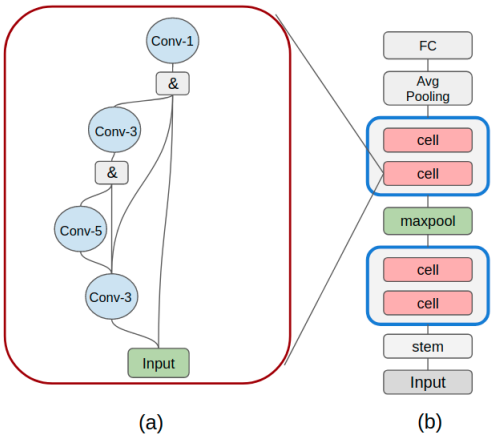


Figura 5.3: Arquitectura ganadora de los experimentos realizados en el conjunto de datos MRDBI mediante el 2LGA.

5.3.2. Comparación en Fashion-MNIST

La Tabla 5.7 muestra la comparación de resultados del 2LGA y de los trabajos publicados anteriormente para el dataset Fashion-MNIST. Entre los métodos presentados, los primeros cinco corresponden a trabajos donde se propone una arquitectura de CNN, pero diseñada manualmente. Los siguientes cuatro métodos utilizan algoritmos evolutivos (AG o PSO) para encontrar automáticamente arquitecturas de CNN, además de AutoKeras. Los métodos evolutivos no utilizan ningún tipo de DA, pero los enfoques no evolutivos sí. Por lo tanto, se presentan los resultados del 2LGA sin ningún DA en la segunda columna, y con un DA compuesto por *Random Crop* (RC) y *Random Flip* (RF) en la tercera columna. Cabe destacar que este DA solo fue utilizado para entrenar al individuo ganador y no en la evolución, como se explicó en la sección 4.2.2. De esta forma, la Tabla 5.7 muestra en cada columna, métodos con las mismas condiciones de entrenamiento para que los resultados sean comparables.

Como se puede ver en la Tabla 5.7, el 2LGA logró mejores resultados que los publicados anteriormente sin utilizar DA. En concreto, la mejora corresponde a un 0.72 en el test-error promedio respecto al mejor competidor (REMNet), lo que equivale a una disminución de un 13 %. Al comparar los métodos que usan DA, DenseNet supera en 0.02 al 2LGA si se observa el test-error promedio. Sin embargo, DenseNet solo reporta un resultado, por lo que se puede comparar con el mejor obtenido por 2LGA. En tal caso, 2LGA obtiene 0.04 menos test-error (una disminución de 0.9 %). Sin embargo, no es posible afirmar que estos resultados sean estadísticamente diferentes. Por otra parte, AutoKeras obtiene un peor desempeño que la mayoría de los algoritmos evolutivos, considerando que los últimos no utilizan DA.

En resumen, el 2LGA obtiene el mejor resultado cuando no se utiliza DA, y obtiene resultados dentro del estado del arte cuando si se utiliza. Debido a que el DA solo se usa para entrenar al individuo ganador, mejores resultados pueden alcanzarse si en la evolución se calcula el *fitness* incluyendo DA. De esta forma, el AG podría adaptarse mejor a las condiciones finales de evaluación. Esto también es válido para los algoritmos evolutivos de los competidores, que obtienen mejores resultados que los métodos manuales que no usan DA.

Tabla 5.7: Comparación del test error del 2LGA con los resultados publicados anteriormente en Fashion-MNIST. La desviación estándar de cada método se muestra entre paréntesis cuando está disponible.

Método	% Error sin DA	% Error con DA (RF + RC)
GoogleNet [52]	6.30	–
ResNet18 [55]	–	5.10
DenseNet [56]	–	**4.60**
MobileNet[a] [107]	–	5.00
WideResNet-28-10[b] [104]	–	4.63
AutoKeras AK [87]	–	7.42
AutoKeras AK-DP [87]	–	6.72
REMNet [103]	5.54	–
EvoCNN (mean) [9]	7.28 (1.69)	–
EvoCNN (best) [9]	5.47	–
DeepSwarm (mean) [102]	6.75	–
DeepSwarm (best) [102]	6.44	–
PsoCNN (mean) [33]	5.90	–
PsoCNN (best) [33]	5.53	–
Ours 2LGA (mean)	**4.82** (0.05)	4.62 (0.04)
Ours 2LGA (best)	4.76	4.56

[a]En esta implementación, solo se utilizó RF.

[b]En la primera versión del dataset, habían algunas imágenes duplicadas en los conjuntos de test y entrenamiento. Por lo tanto, los resultados informados en el artículo original [105] son mejores. Los resultados informados aquí provienen de [117], donde los experimentos se repitieron con la última versión del conjunto de datos.

5.3.3. Comparación en CIFAR-10

En la tabla 5.8 se muestran los resultados de los experimentos en CIFAR-10. Los competidores de la Tabla 5.8 utilizaron técnicas de DA comunes, y se excluyeron a aquellos que utilizan técnicas más complejas de entrenamiento excepto AutoKeras AK-DP, que si incluye dichas técnicas). Como se puede ver, el método propuesto logra resultados del estado del arte en este conjunto de datos. Más aún, el 2LGA obtiene un error 0.35 menor que AE-CNN, el mejor competidor, lo que equivale a una mejora de un 8.1 % respecto al mismo. Al comparar los resultados con AutoKeras AK-DP, se observa que el 2LGA obtiene un test-error un 0.35 mayor. Sin embargo, para una comparación más justa se re-entrenó a la arquitectura ganadora del 2LGA con técnicas avanzadas de entrenamiento, alcanzando un error de 2.68, un 25.6 % mejor que AutoKeras AK-DP.

Tabla 5.8: Comparación del test-error (%) entre el 2LGA y los mejores resultados publicados anteriormente en CIFAR-10 (más bajo es mejor).

Método	Error de test (%)
Johnson [110]	15.15
SOBA [109]	4.78
CNN-GA [68]	4.78
HGAPSO [31]	4.37
AE-CNN [93]	4.30
AutoKeras AK [87]	11.44
AutoKeras AK-DP [87]*	3.60
Ours 2LGA	**3.95**
Ours 2LGA*	**2.68**

* Entrenamientos realizados con *cutout* y otras técnicas avanzadas de entrenamiento como *schedule learning rate*.

5.4. Estudio de generalización

Los resultados de los experimentos de generalización se muestran en la Figura 5.4, donde las filas representan los modelos encontrados en los respectivos datasets, mientras que las columnas representan los datasets en los cuales se evaluaron dichos modelos. Por ejemplo en (a), el mejor modelo encontrado para el dataset MRDBI ocupa la primera fila, obteniendo un error de test de 6.33 % al evaluarlo en el dataset MRDBI, un 5.11 % al evaluarlo en Fashion-MNIST, y un 5.17 % al evaluarlo en CIFAR-10. Por otra parte, en (b) se muestran los resultados normalizados por columna, en porcentaje, respecto al modelo encontrado para el mismo dataset. Es decir, al observar una columna de (b), los valores representan la diferencia porcentual del error de test de cada modelo respecto al modelo encontrado en el mismo dataset. Por ejemplo, el modelo encontrado para el dataset Fashion-MNIST obtuvo en el dataset MRDBI un error 12.95 % mayor que el modelo encontrado para MRDBI.

Respecto a la capacidad de generalizar de los modelos, al observar los errores obtenidos en el dataset MRDBI (primera columna de la Figura 5.4 (a)), los 3 modelos obtienen resultados dentro del estado del arte, obteniendo un error menor que HGAPSO (10.53 %), el mejor competidor en este dataset (ver Tabla 5.6). Lo mismo ocurre para los resultados obtenidos en Fashion-MNIST, donde los 3 modelos obtienen un menor error que evoCNN (5.47 %), el mejor competidor de este dataset (ver Tabla 5.7). Por otra parte, para el dataset CIFAR-10, los resultados de los 3 modelos evaluados están cerca del estado del arte, alcanzando resultados competitivos (ver Tabla 5.8). Por lo tanto, los modelos tienen capacidad suficiente de generalización como para obtener resultados dentro del estado del arte, pese a no haber sido optimizados para estos datasets. Esto excluye los resultados en la diagonal, donde los modelos si fueron optimizados para el dataset respectivo.

Por otra parte, en la Figura 5.4 (b) se observa que la diagonal contiene los valores mas bajos para cada columna. Esto quiere decir que, pese a que los modelos obtienen resultados competitivos con los competidores, obtienen consistentemente un peor desempeño que el modelo optimizado para el mismo dataset. Es decir, para cada dataset, el modelo optimizado

en dicho dataset es el que obtiene el mejor resultado.

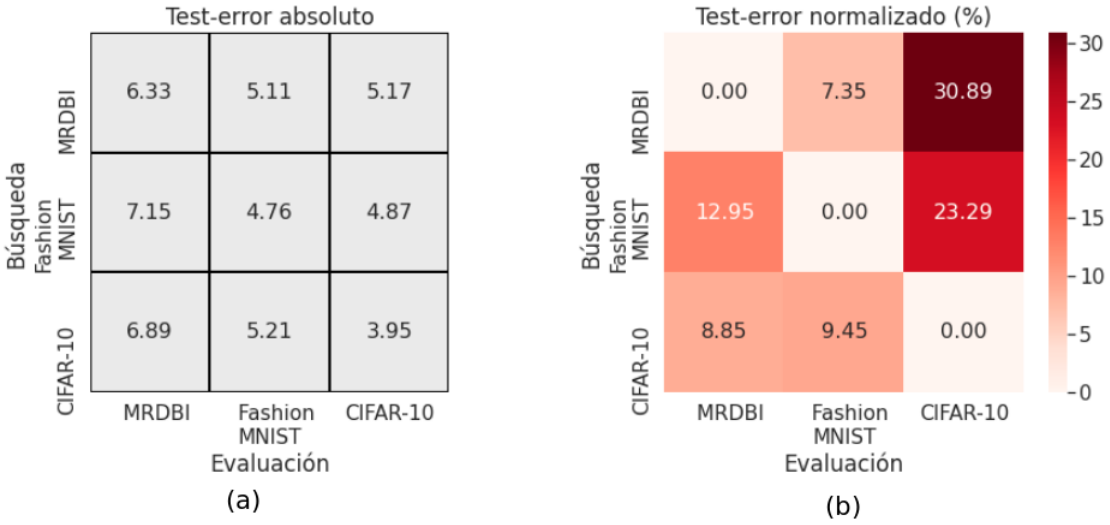


Figura 5.4: Evaluación de modelos encontrados para dataset MRDBI, Fashion-MNIST, y CIFAR-10 en los mismos datasets. Las filas representan a los los modelos y las columnas a los datasets evaluados. En (a) el error mostrado es absoluto, y en (b) el error se normaliza por columna (en porcentaje) respecto al resultado obtenido por el modelo optimizado y evaluado en el mismo dataset (respecto a la diagonal de (a)).

Otro aspecto a considerar es que los modelos MRDBI y Fashion-MNIST desempeñan un 31 % y un 23.3 % peor que el modelo CIFAR-10, al evaluarlo en CIFAR-10. Por su parte, el modelo CIFAR-10 obtiene resultados un 8.9 % y un 9.5 % peor en los datasets MRDBI y Fahion-MNIST, al compararlo con los modelos de cada dataset. Esto quiere decir que el modelo CIFAR-10 tiene una mejor capacidad de generalización que los modelos MRDBI y Fashion-MNIST. Esta desigualdad en la generalización puede explicarse por la diferencia entre los dataset CIFAR-10 respecto a MRDBI y Fashion-MNIST: Mientras que CIFAR-10 tiene imágenes a color de tamaño 32×32, los datasets MRDBI y Fashion-MNIST tienen imágenes en blanco y negro de tamaño 28×28. Por lo tanto, el evaluar el modelo CIFAR-10 en MRDBI o Fashion-MNIST conlleva una reducción de complejidad y dimensionalidad, mientras que al evaluar el modelo MRDBI (o Fashion -MNIST) en CIFAR-10 implica un aumento en la complejidad y dimensinalidad, dificultando su generalización.

5.5. Análisis del ranking

La Tabla 5.9 muestra los resultado de la correlación entre los rankings de entrenamiento corto y entrenamiento largo, calculada tanto para la población completa, como para el los N mejores (N=8). Además, se muestra el porcentaje de individuos que pertenecen al top-N en ambos rankings, respecto a N.

De estos resultados se puede observar que ambos coeficientes de correlación de Kendall son positivos, pero no cercanos a 1. Es decir, al entrenar con menos épocas el ranking obtenido cambia respecto a si se entrenan durante más épocas, pero manteniendo cierto orden. En

Tabla 5.9: Resultados del análisis de rankings. Cada valor es el promedio de todas las generaciones evaluadas y de 5 evoluciones independientes.

Correlación del 1L	Correlación del top-8 del 1L	Porcentaje de individuos seleccionados correctamente
0.36	0.17	55.6 %

particular, si se considera la población completa la correlación es de 0.36, mientras que para el top-8, dicha correlación cae a 0.17.

Esto significa que, al usar entrenamientos cortos, el ranking obtenido en el top-8 es menos confiable que el obtenido para la población general. En otras palabras, el ranking de la población general mantiene cierto orden respecto a si se entrenan con más épocas, pero en el top-8 el orden empeora significativamente, dificultando la selección del mejor individuo.

La correlación de la población completa es suficiente para que algoritmos de búsqueda que evalúan con entrenamientos cortos encuentren buenas soluciones (como lo hace el AG-*short*). Sin embargo, la baja confiabilidad del ranking en el top-8 impide que algoritmos como AG-*short* obtengan los mismos resultados que un AG-*long*. Por otra parte, según [94], un entrenamiento corto no es apropiado para clasificar redes de desempeño similar, efecto que ocurre en el top-8 y razón por la cual su coeficiente de correlación es cercano a 0.

Estos resultados explican la efectividad del 2LGA, que aprovecha la correlación positiva (0.36) entre entrenamiento largo y corto para seleccionar una parte de las mejores redes (un 55.6 % de los mejores individuos son seleccionados, en promedio). Luego, la baja correlación en la parte superior del 1L se corrige volviendo a entrenar y reorganizando los N primeros. Gracias a esto, el 2LGA puede superar significativamente a un AG de entrenamiento corto como el AG-*short*.

5.6. Análisis

El espacio de búsqueda propuesto contiene una gran cantidad de posibles soluciones, las que varían tanto de forma como de tamaño. Esto permite obtener soluciones diversas y, por lo tanto, las arquitecturas muestreadas tienden a ser diferentes unas de otras. Según el estudio realizado en [94], cuanto más diferentes son las arquitecturas de una población, menos épocas se necesitan para generar un ranking confiable. Por lo tanto, la amplia diversidad de soluciones es beneficiosa para la búsqueda ya que el ranking obtenido con pocas épocas es confiable, permitiendo que el AG encuentre mejores soluciones. Esto se ve reflejado en los resultados de AG-*short* y RS (18 eps) de la Tabla 5.5, que demuestran que combinando el espacio de búsqueda con una baja cantidad de épocas (18 épocas), es posible alcanzar al estado del arte presente en la Tabla 5.4.

Sin embargo, los métodos que utilizan 18 épocas obtienen resultados consistentemente peores que aquellos que usan 54. Esto puede deberse a que, a medida que avanza la evolución, las soluciones tienden a parecerse debido a la explotación del espacio de búsqueda. En consecuencia, más épocas son necesarias para discriminar correctamente el ranking de la

población.

Por otra parte, el 2LGA propuesto aprovecha tanto los entrenamientos precisos como los aproximados. En primer lugar, como la mayoría de los individuos son evaluados con un entrenamiento aproximado, y solo una fracción de la población es entrenada de forma precisa, el tiempo de evolución no es afectado mayormente por los entrenamientos largos. Esto permite que el 2LGA requiera un tiempo similar a un AG-*short*. En segundo lugar, el 2LGA aprovecha los entrenamientos largos para generar un ranking más confiable de esta sub-población. Como resultado, el 2LGA logra el mismo rendimiento que si todos los modelos fueran entrenados con precisión, pero en un tiempo significativamente más corto.

En otras palabras, el uso de 2LGA reduce los efectos negativos del uso de técnicas de aproximación del *fitness*, como entrenamientos cortos, por lo que le permite al AG encontrar arquitecturas que logran un error menor con un aumento mínimo del tiempo de búsqueda. Un aspecto fundamental es que cualquier AG que incorpore este método podría mejorar su rendimiento, sacrificando apenas una fracción de tiempo.

Un último aspecto a considerar es que los parámetros del 2LGA no fueron optimizados debido al costo computacional que esto requiere. Pese a esto, como se observa en las tablas 5.6, 5.7, y 5.8, el 2LGA logra resultados competitivos con el estado del arte. Una búsqueda más exhaustiva de los parámetros del 2LGA podría mejorar aún más los resultados en cuanto a error de test del individuo final como a tiempo de evolución, por lo que se establece como trabajo futuro y como una extensión del trabajo presente.

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Conclusiones

En este trabajo se propuso un nuevo algoritmo genético de dos niveles (2LGA) para abordar el problema del compromiso entre la fidelidad y el tiempo del cálculo del *fitness* en la búsqueda de arquitecturas de CNNs aplicadas al reconocimiento de patrones en imágenes. En el primer nivel del 2LGA, el *fitness* de cada individuo se calcula con menor fidelidad para ahorrar tiempo. Luego, el *fitness* de los mejores individuos, que componen el segundo nivel, se calcula con mayor fidelidad. Dado que solo una fracción de la población compone el segundo nivel, solo hay un pequeño aumento en el tiempo de evolución, mientras que al evaluar de forma precisa a los individuos más prometedores, se obtiene un mejor ranking de dichos individuos permitiendo al algoritmo genético realizar una búsqueda más eficaz.

Además, la propuesta incluye un nuevo espacio de búsqueda que reduce las restricciones de los parámetros, y se incluye una codificación que permite realizar operaciones genéticas como mutación y *crossover*. El espacio de búsqueda, que está basado en celdas, permite que las arquitecturas de CNN pueden tener cualquier tamaño. Este tamaño depende directamente del número de operaciones por celda, del número de celdas por bloque, y del número de bloques. Todos éstos son parámetros incluidos en el espacio de búsqueda. Además, este espacio de búsqueda permite que las arquitecturas tengan diversas formas, debido a las interconexiones entre nodos (*skip-connections*), y a la cantidad de *feature-maps* por nodo. Cada nodo de la celda se asocia a solo un tipo de operación, la que a su vez tiene parámetros internos incluidos en el espacio.

El esquema de codificación diseñado es de largo variable, y posibilita las operaciones de mutación y *crossover* en este complejo espacio de búsqueda. Si bien cada nodo de la celda se asocia a solo un tipo de operación, dicho nodo codifica los parámetros de todas las posibles operaciones, y no solo la asociada a ésta nodo. Los parámetros de operaciones no utilizadas no se expresan, pero si se transmiten a la descendencia. De esta forma, realizar cruces entre individuos es directo, ya que todos los individuos tienen los mismos parámetros codificados, y es posible transmitir la información genética con mayor fluidez al no tener que eliminar la información no utilizada.

Luego de aplicar el 2LGA a los problemas de reconocimiento de patrones en los siete datasets de MNIST-V, Fashion-MNIST, y CIFAR-10, se logró resultados significativamente mejores que los publicados anteriormente. La mejora en el test-error que logró el 2LGA en dichos datasets, en relación a los resultados publicados, corresponde a: 22.6 % en MB respecto a HGAPSO, 45.8 % en MRB respecto a PsoCNN, 44.2 % en MBI respecto a PsoCNN, 55.5 % en MRD respecto a EvoCNN, y 45 % en MRDBI respecto a HGAPSO. En Fashion-MNIST, el 2LGA logró una mejora de 12.98 % sin DA respecto a REMNet, y de 0.87 % con RF + RC respecto a DenseNet. Por último, en CIFAR-10 la mejora respecto AE-CNN, el mejor competidor publicado en revista que solo utiliza un DA común, fue de 8.1 %.

También se comprobó que el 2LGA iguala a un AG-*short* (AG con entrenamientos cortos) en el conjunto de datos MNIST-V en términos de tiempo de evolución, mientras que logra los mismos resultados que un AG-*long* (AG con entrenamientos largos) en términos de test-error. De esta forma, el 2LGA no solo logra reducir los efectos negativos de utilizar aproximaciones del *fitness*, si no que logra un mejor balance entre tiempo de evolución y desempeño, que los AG ordinarios.

Por otra parte, se evaluó el espacio de búsqueda propuesto combinándolo con uno de los algoritmos de búsqueda más simples, *Random Search*, y comparando los resultados con algoritmos genéticos del estado del arte. Los resultados mostraron que el espacio de búsqueda ayudó a RS a superar incluso a los mejores resultados publicados anteriormente, que utilizaron algoritmos de búsqueda más sofisticados que RS. Por lo tanto, al ser RS uno de los algoritmos de búsqueda más simples (aunque difícil de superar [18, 95, 96, 97]), la mejora se atribuye al espacio de búsqueda y no a RS, demostrando la efectividad del espacio de búsqueda propuesto.

Finalmente, cabe destacar que el sistema diseñado recibe un conjunto de datos como entrada y devuelve una CNN completamente entrenada. Para utilizar este sistema, el usuario no necesita tener conocimientos de *Deep Learning*, o gastar esfuerzo y tiempo en ajustar parámetros. Por lo tanto, el sistema puede ser útil para que los investigadores de *Deep Learning* encuentren CNNs que mejoren los resultados del estado del arte en problemas de reconocimiento de patrones, con ahorros significativos de tiempo, y para ayudar a investigadores de otras áreas, sin conocimientos de *Deep Learning*, a incorporar modelos de CNNs en sus investigaciones. Además, el algoritmo genético de dos niveles puede ser utilizado por otros algoritmos evolutivos que usen aproximaciones del *fitness*, permitiendo que mejoren su desempeño sacrificando una fracción del tiempo total de evolución.

Los resultados de este trabajo de Tesis dieron origen a la aceptación y publicación de un artículo en una revista científica [118].

6.2. Trabajo Futuro

Existen varias líneas a seguir para mejorar los resultados obtenidos, así como líneas que se desprenden de la investigación original, dando lugar a otras ramas de investigación.

En primer lugar, pese a que se realizaron sensibilizaciones de algunos parámetros del 2LGA, esto no se hizo de forma exhaustiva ni para todos los parámetros. Esto ya que se usaron configuraciones de parámetros obtenidas de la literatura, y a que el costo computacional de

realizar una optimización de ellos es considerablemente grande. Entre los parámetros que se pueden optimizar están el número de generaciones, los tamaños de poblaciones, la frecuencia del segundo nivel, las épocas de entrenamiento de cada nivel, el porcentaje de elitismo, y el tamaño de torneo del método de selección.

Sumado a esto, es posible realizar una sensibilización de las variables del espacio de búsqueda para estudiar la importancia de cada uno. Para esto, se puede muestrear una cierta cantidad de soluciones del espacio de búsqueda, y para cada una, sensibilizar la variable en estudio. De esta forma, se podría cuantificar estadísticamente el efecto que tiene cada variable sobre el desempeño de la red. Con estos resultados se podrían acotar o expandir los límites de cada variable (presentes en la Tabla 4.1), o simplemente eliminar aquellas variables que no influyen de forma considerable en el desempeño de las redes.

Relacionado con lo anterior, se puede expandir el espacio de búsqueda agregando más operaciones y/o parámetros de cada operación. Por ejemplo, se pueden incluir convoluciones separables en profundidad (*depthwise Convolutions*) o separables en *kernels* de filas y columnas. Dentro de las posibles variables a incluir en cada operación están la inicialización de pesos y el decaimiento de los pesos.

Otra modificación al espacio de búsqueda posible es permitir que las celdas de bloques distintos sean distintas, es decir, que se busquen tantas celdas como bloques halla en la red. De esta forma, se permite que las soluciones tengan distintos tipos de celdas, dependiendo de la profundidad en la que se encuentren.

Por otra parte, dentro del trabajo futuro relacionado con el algoritmo genético está el utilizar una cantidad creciente de épocas de entrenamiento a medida que avanza la evolución. Entonces, a medida que las soluciones se especializan y se parecen más, la cantidad de épocas de entrenamiento aumenta, permitiendo la obtención de rankings más adecuados. Además, es posible incorporar 3 o más niveles al algoritmo genético, permitiendo que las redes se evalúen de forma cada vez más precisa.

Sumado a esto, en este trabajo solo se usó la cantidad de épocas de entrenamiento para aproximar el *fitness*. Sin embargo, se pueden estudiar otras formas de aproximación de fitness, como usar menos ejemplos, imágenes de menor resolución, o menos clases que el dataset original.

Finalmente, durante el desarrollo del presente trabajo surgieron nuevas ramas de investigación, algunas de las cuales fueron abordadas, pero no incluidas. Una de estas ramas es la clasificación de arquitecturas para definir si tendrán un buen desempeño sin ser entrenadas previamente. Este concepto se puede extender, y formular el problema como uno de regresión, donde se busca obtener el fitness de cada individuo sin necesidad de entrenarlo.

Acrónimos

1L *First Level*. 30

2D Dos Dimensinoes. 8

2L *Second Level*. 30

2LGA *Two Level Genetic Algorithm*. i

AE-CNN *Automatically Evolving Convolutional Neural Networks*. 27

AG Algoritmo Genético. i

ANOVA *Analysis Of Variance*. 44

CIFAR-10 *Canadian Institute For Advanced Research*. i

CNN *Convolutional Neural Network*. i

CNN-GA *Convolutional Neural Network Genetic Algorithm*. 28

DA *Data Augmentation*. 39

DAG *Directed Acyclic Graph*. 32

DARTS *Differentiable Architecture Search*. 24

DC *Deterministic Crowding*. 19

DECNN *Differential Evolution of Convolutional Neural Networks*. 43

DL *Deep Learning*. 1

evoCNN *Evolving Convolutional Neural Networks*. 27

FC *Fully Connected*. 27

GD *Gradient Descent*. 24

GPU *Graphics Processing Unit*. 3

HGAPSO *Hybrid Genetic Algorithm Particle Swarm Optimization*. 27

HSD *Honestly Significant Difference*. 44

IA Inteligencia Artificial. 1

ILSVR *ImageNet Large Scale Visual Recognition Challenge*. 2

IPPSO *Internet Protocol Particle Swarm Optimization*. 43

LO *Linear Order*. 17

MB MNIST *Basic*. 41

MBI MNIST *with Background Images*. 41

ML *Machine Learning*. 1

MLP *Multi Layer Perceptron*. 7

MNIST *Modified National Institute of Standards and Technology*. i

MRB MNIST *with Random Background*. 41

MRD MNIST *with Rotated Digits*. 41

MRDBI MNIST *with Rotated Digits and Background Images*. i, 41

NAS *Nerual Architecture Search*. 2

NEAT *Neuroevolution of Augmenting Topologies*. 25

NN *Neural Network*. 7

OB Optimización Bayesiana. 2

PSO *Particle Swarm Optimization*. 45

PsoCNN *Particle Swarm Optimization for Convolutional Neural Networks*. 43

RC *Random Crop*. 43

ReLU *Rectified Linear Unit*. 12

RF *Random Flip*. 43

RL *Reinforcement Learning*. 2

RS *Random Selection*. 17

RS *Random Search*. 21

RTS *Restricted Tournament Selection*. 19

RW *Replace Worst*. 19

SGD *Stochastic Gradient Descent*. 9

SOBA *Swarm Optimised Block Architecture*. 43

TAU Teorema de Aproximación Universal. 7

TS *Tournament Selection*. 17

TWEANN *Topology and Weight Evolving Artificial Neural Network*. 25

VGG *Visual Geometry Group*. 12

WAMS *Worst Among Most Similar*. 19

Bibliografía

[1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. USA: Prentice Hall Press, 2009.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015. doi:https://doi.org/10.1038/nature14539

[3] M. Galeso, *Apple Siri for Mac: An Easy Guide to the Best Features*. North Charleston, SC, USA: CreateSpace Independent Publishing Platform, 2017.

[4] M. Alexa, *Amazon Alexa: 2017 User Guide + 200 Ester Eggs*. Independently published, 2017.

[5] H. Zhu, Z. An, C. Yang, K. Xu, E. Zhao, and Y. Xu, "Eena: Efficient evolution of neural architecture," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, 2019, pp. 1891–1899.

[6] A. Baldominos, Y. Saez, and P. Isasi, "On the automated, evolutionary design of neural networks: past, present, and future," *Neural Computing and Applications*, vol. 32, p. 519–545, 2020. doi:10.1007/s00521-019-04160-6

[7] S. Litzinger, A. Klos, and W. Schiffmann, "Compute-efficient neural network architecture optimization by a genetic algorithm," in *International Conference on Artificial Neural Networks*. Springer, 2019, pp. 387–392.

[8] Y. Liu, Y. Sun, B. Xue, M. Zhang, and G. Yen, "A survey on evolutionary neural architecture search," *arXiv preprint, arXiv:2008.10937*, 2020.

[9] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 394–407, 2020. doi:10.1109/TEVC.2019.2916183

[10] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013. doi:10.1109/TPAMI.2013.50

[11] G. Zhong, W. Jiao, W. Gao, and K. Huang, "Automatic design of deep networks with neural blocks," *Cognitive Computation*, vol. 12, p. 1–12, 2020. doi:10.1007/s12559-019-09677-5

[12] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of Physiology*, vol. 160, no. 1, pp. 106–154, 1962. doi:https://doi.org/10.1113/jphysiol.1962.sp006837

[13] K. Fukushima, "Neural network model for a mechanism of pattern recognition unaffected by shift in position-neocognitron," *IEICE Technical Report, A*, vol. 62, no. 10, pp. 658–665, 1979.

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. doi:10.1109/5.726791

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, 2017. doi:10.1145/3065386

[16] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, pp. 211–252, 2015. doi:10.1007/s11263-015-0816-y

[17] B. Wang, Y. Sun, B. Xue, and M. Zhang, "Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification," in *2018 IEEE Congress on Evolutionary Computation (CEC)*, 2018, pp. 1–8.

[18] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

[19] Q. Song, D. Cheng, H. Zhou, J. Yang, Y. Tian, and X. Hu, "Towards automated neural interaction discovery for click-through rate prediction," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. New York, NY, USA: Association for Computing Machinery, 2020, p. 945–955.

[20] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Automatically evolving cnn architectures based on blocks," *arXiv preprint arXiv:1810.11875*, 2018.

[21] C. Perez, C. Salinas, P. Estevez, and P. Valenzuela, "Genetic design of biologically inspired receptive fields for neural pattern recognition," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 33, no. 2, pp. 258–270, 2003. doi:10.1109/TSMCB.2003.810441

[22] B. A. Olshausen and D. J. Field, "Sparse coding with an overcomplete basis set: A strategy employed by v1?" *Vision Research*, vol. 37, no. 23, pp. 3311–3325, 1997. doi:https://doi.org/10.1016/S0042-6989(97)00169-7

[23] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, vol. 20, pp. 1–21, 2019. [Online]. Available: https://www.jmlr.org/papers/volume20/18-598/18-598.pdf

[24] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 4780–4789, Jul. 2019. doi:10.1609/aaai.v33i01.33014780

[25] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, p. 24–35, 2019. doi:10.1038/s42256-018-0006-z

[26] K. O. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 06 2002. doi:10.1162/106365602320169811

[27] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, 2017, p. 2902–2911.

[28] M. Mitchell, *An Introduction to Genetic Algorithms.* Cambridge, MA, USA: MIT Press, 2020.

[29] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast bayesian optimization of machine learning hyperparameters on large datasets," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, A. Singh and J. Zhu, Eds., vol. 54, 20–22 Apr 2017, pp. 528–536.

[30] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of imagenet as an alternative to the cifar datasets," *arXiv preprint arXiv:1707.08819*, 2017.

[31] B. Wang, Y. Sun, B. Xue, and M. Zhang, "A hybrid ga-pso method for evolving architecture and short connections of deep convolutional neural networks," in *Pacific Rim International Conference on Artificial Intelligence*. Springer, 2019, pp. 650–663.

[32] A. Zela, A. Klein, S. Falkner, and F. Hutter, "Towards automated deep learning: Efficient joint neural architecture and hyperparameter search," *arXiv preprint arXiv:1807.06906*, 2018.

[33] F. E. Fernandes Junior and G. G. Yen, "Particle swarm optimization of deep neural networks architectures for image classification," *Swarm and Evolutionary Computation*, vol. 49, pp. 62–74, 2019. doi:https://doi.org/10.1016/j.swevo.2019.05.010

[34] B. Wang, Y. Sun, B. Xue, and M. Zhang, "A hybrid differential evolution approach to designing deep convolutional neural networks for image classification," in *Australasian Joint Conference on Artificial Intelligence*, vol. 11320, 2018, pp. 237–250.

[35] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943. doi:https://doi.org/10.1007/BF02478259

[36] Wikipedia, the free encyclopedia, "Diagrama de un perceptrón con cinco señales de entrada." 2021, [accessed May 17, 2021]. [Online]. Available: https://es.wikipedia.org/

wiki/Perceptr%C3%B3n#/media/Archivo:Perceptr%C3%B3n_5_unidades.svg

[37] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[38] F. S. Caparrini, "Redes neuronales: una visión superficial," 2017. [Online]. Available: http://www.cs.us.es/~fsancho/?e=72

[39] Eddy Decena, "Convolutional layer." 2019, [accessed May 17, 2021]. [Online]. Available: https://miro.medium.com/max/678/0*cdDIFTFg3burRkXS

[40] Nick Winovich, "Deep learning." 2021, [accessed May 17, 2021]. [Online]. Available: https://www.math.purdue.edu/~nwinovic/deep_learning_optimization.html

[41] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.

[42] G. F. Miller, P. M. Todd, and S. U. Hegde, "Designing neural networks using genetic algorithms." in *Proceedings of the 3rd International Conference on Genetic Algorithms*, vol. 89, 1989, pp. 379–384.

[43] F. Chollet, *Deep learning with python*. Manning Publications Co., 2017.

[44] Y. Wu, L. Liu, J. Bae, K.-H. Chow, A. Iyengar, C. Pu, W. Wei, L. Yu, and Q. Zhang, "Demystifying learning rate policies for high accuracy training of deep neural networks," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 1971–1980.

[45] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[46] H. Park and J.-H. Son, "Machine learning techniques for thz imaging and time-domain spectroscopy," *Sensors*, vol. 21, p. 1186, 02 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/4/1186. doi:10.3390/s21041186

[47] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[48] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, 2019. doi:https://doi.org/10.1186/s40537-019-0197-0

[49] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2818–2826.

[50] K. Fukushima, "Neocognitron: A hierarchical neural network capable of visual pattern recognition," *Neural Networks*, vol. 1, no. 2, pp. 119–130, 1988. doi:https://doi.org/10.1016/0893-6080(88)90014-7

[51] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1409.1556

[52] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[53] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen," *Unpublished doctoral dissertation, Institut für Informatik, Technische Universität, Munchen*, 1991.

[54] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994. doi:10.1109/72.279181

[55] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[56] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261–2269.

[57] Mukul Khanna, "Dense block (db) with six dense layers (dl)," 2019, [accessed May 17, 2021]. [Online]. Available: https://towardsdatascience.com/paper-review-densenet-densely-connected-convolutional-networks-acf9065dfefb

[58] D. Han, J. Kim, and J. Kim, "Deep pyramidal residual networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6307–6315.

[59] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999. doi:10.1109/5.784219

[60] O. Boyabatli and I. Sabuncuoglu, "Parameter selection in genetic algorithms," *Journal of Systemics, Cybernetics and Informatics*, vol. 4, no. 2, p. 78, 2004.

[61] S. Ruder, "An overview of gradient descent optimization algorithms," *ArXiv*, vol. abs/1609.04747, 2016.

[62] M. Bonelli and H. F. Begliardo, "Optimización de armaduras planas mediante diseño paramétrico y algoritmos genéticos: Efectos de la no correspondencia objeto real-modelo idealizado," in *Mecánica Computacional*, 2016, pp. 501–515.

[63] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm:

past, present, and future," *Multimedia Tools and Applications*, p. 8091–8126, 2021. doi:https://doi.org/10.1007/s11042-020-10139-6

[64] M. Bessaou and P. Siarry, "A genetic algorithm with real-value coding to optimize multimodal continuous functions," *Structural and Multidisciplinary Optimization*, vol. 23, pp. 63–74, 12 2001. doi:10.1007/s00158-001-0166-y

[65] U. A.J. and S. P.D., "Crossover operators in genetic algorithms: A review," *ICTACT Journal on Soft Computing*, vol. 06, pp. 1083–1092, 10 2015. doi:10.21917/ijsc.2015.0150

[66] Avik Dutta, "One point crossover," 2019, [accessed May 17, 2021]. [Online]. Available: https://www.geeksforgeeks.org/crossover-in-genetic-algorithm/

[67] S. Picek, D. Jakobovic, and M. Golub, "On the recombination operator in the real-coded genetic algorithms," in *2013 IEEE Congress on Evolutionary Computation*. IEEE, 2013, pp. 3103–3110.

[68] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing cnn architectures using the genetic algorithm for image classification," *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3840–3854, 2020. doi:10.1109/TCYB.2020.2983860

[69] K. Deb and ayan Deb, "Analysing mutation schemes for real-parameter genetic algorithms," *International Journal of Artificial Intelligence and Soft Computing*, vol. 4, pp. 1–28, 2014. doi:10.1504/ijaisc.2014.059280

[70] M. Azimipour, M. r. Bonyadi, and M. Eshghi, "Using immune genetic algorithm in atpg," *Australian Journal of Basic and Applied Sciences*, vol. 2, pp. 920–928, 01 2008.

[71] A. E. Eiben, J. E. Smith *et al.*, *Introduction to evolutionary computing*. Springer, 2003, vol. 53.

[72] G. Ochoa, I. Harvey, and H. Buxton, "Optimal mutation rates and selection pressure in genetic algorithms," in *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO'00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, p. 315–322.

[73] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Evolutionary computation 1: Basic algorithms and operators*. CRC press, 2018.

[74] Z. Michalewicz, *Genetic algorithms+ data structures= evolution programs*. Springer Science & Business Media, 2013.

[75] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[76] M. A. A. Albadr, S. Tiun, M. Ayob, and F. T. AL-Dhief, "Genetic algorithm based on natural selection theory for optimization problems," *Symmetry*, vol. 12, p. 1758, 2020.

[77] A. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, ser. Natural Com-

puting Series. Springer, 2015.

[78] P. Reed, B. Minsker, and D. E. Goldberg, "Designing a competent simple genetic algorithm for search and optimization," *Water Resources Research*, vol. 36, no. 12, pp. 3757–3761, 2000.

[79] J. J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Transactions on systems, man, and cybernetics*, vol. 16, no. 1, pp. 122–128, 1986.

[80] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *5th International Conference on Learning Representations, ICLR*, 2017. [Online]. Available: https://openreview.net/forum?id=r1Ue8Hcxg

[81] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, p. 106622, 2021. doi:https://doi.org/10.1016/j.knosys.2020.106622

[82] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[83] H. Liu, K. Simonyan, and Y. Yang, "DARTS: differentiable architecture search," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[84] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016. doi:10.1109/JPROC.2015.2494218

[85] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter, "Towards automatically-tuned neural networks," in *Workshop on Automatic Machine Learning*. PMLR, 2016, pp. 58–65.

[86] M. Wistuba, "Bayesian optimization combined with successive halving for neural network architecture optimization." in *AutoML@ PKDD/ECML*, 2017, pp. 2–11.

[87] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1946–1956.

[88] L. Zimmer, M. Lindauer, and F. Hutter, "Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 9, pp. 3079–3090, 2021. doi:10.1109/TPAMI.2021.3067763

[89] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE transactions on neural networks*, vol. 8, no. 3, pp. 694–713, 1997.

[90] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artificial life*, vol. 15, no. 2, pp. 185–212, 2009.

[91] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," MIT and NYU, Tech. Rep., 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[92] D. Zhou, X. Zhou, W. Zhang, C. C. Loy, S. Yi, X. Zhang, and W. Ouyang, "Econas: Finding proxies for economical neural architecture search," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 393–11 401.

[93] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Completely automated cnn architecture design based on blocks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 4, pp. 1242–1254, 2020. doi:10.1109/TNNLS.2019.2919608

[94] G. Kyriakides and K. Margaritis, "The effect of reduced training in neural architecture search," *Neural Computing and Applications*, p. 17321–17332, 2020. doi:10.1007/s00521-020-04915-6

[95] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8697–8710.

[96] K. Yu, C. Sciuto, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," *arXiv preprint arXiv:1902.08142*, 2019.

[97] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, vol. 115. PMLR, 22–25 Jul 2020, pp. 367–377.

[98] L. N. Smith and N. Topin, "Super-convergence: very fast training of neural networks using large learning rates," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, vol. 11006, 2019, pp. 369 – 386.

[99] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, 7 2018, pp. 5369–5373.

[100] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proceedings of the 24th International Conference on Machine Learning*, ser. ICML '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 473–480.

[101] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[102] E. Byla and W. Pang, "Deepswarm: Optimising convolutional neural networks using swarm intelligence," in *Advances in Intelligent Systems and Computing*, vol. 1043, 2020,

pp. 119–130.

[103] G. Kyriakides and K. Margaritis, "Regularized evolution for macro neural architecture search," in *Artificial Intelligence Applications and Innovations*, vol. 584, 2020, pp. 111–122.

[104] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proceedings of the British Machine Vision Conference (BMVC)*, September 2016, pp. 87.1–87.12.

[105] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 13 001–13 008, 2020. doi:10.1609/aaai.v34i07.7000

[106] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[107] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[108] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *arXiv preprint arXiv:1708.04552*, 2017.

[109] B. Fielding and L. Zhang, "Evolving image classification architectures with enhanced particle swarm optimisation," *IEEE Access*, vol. 6, pp. 68 560–68 575, 2018. doi:10.1109/ACCESS.2018.2880416

[110] F. Johnson, A. Valderrama, C. Valle, B. Crawford, R. Soto, and R. Ñanculef, "Automating configuration of convolutional neural network hyperparameters using genetic algorithm," *IEEE Access*, vol. 8, pp. 156 139–156 152, 2020. doi:10.1109/ACCESS.2020.3019245

[111] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "Nsga-net: neural architecture search using multi-objective genetic algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 419–427.

[112] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," *arXiv preprint arXiv:1804.09081*, 2018.

[113] A. A. Ahmed and S. M. Darwish, "A meta-heuristic automatic cnn architecture design approach based on ensemble learning," *IEEE Access*, vol. 9, pp. 16 975–16 987, 2021. doi:10.1109/ACCESS.2021.3054117

[114] E. R. Girden, *ANOVA: Repeated measures.* Sage, 1992, no. 84.

[115] H. Abdi and L. J. Williams, "Tukey's honestly significant difference (hsd) test," *Encyclopedia of research design*, vol. 3, pp. 583–585, 2010.

[116] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, pp. 81–93, 6 1938. doi:10.1093/biomet/30.1-2.81

[117] A. Nøkland and L. H. Eidnes, "Training neural networks with local error signals," *arXiv preprint arXiv:1901.06656*, 2019.

[118] D. A. Montecino, C. A. Perez, and K. W. Bowyer, "Two-level genetic algorithm for evolving convolutional neural networks for pattern recognition," *IEEE Access*, vol. 9, pp. 126 856–126 872, 2021. doi:10.1109/ACCESS.2021.3111175

Apéndice A

Curvas de evolución

Curvas de evolución del 2LGA en los dataset de MNIST Variants. Para cada dataset se realizaron 5 experimentos independientes, resultando en 5 curvas de evolución distintas. Las Figuras A.1, A.2, A.3, A.4 y A.5 se componen de tres gráficos. El primero corresponde a la evolución del promedio del *fitness* de la población del 1L, donde además se muestra la desviación estándar. En segundo lugar, se muestra la evolución del mejor *fitness* de la popblación en cada generación. Por último, en el tercer gráfico se muestran tanto la evolución del *fitness* promedio de la población (azul), como la evolución del menor *fitness* (rojo).

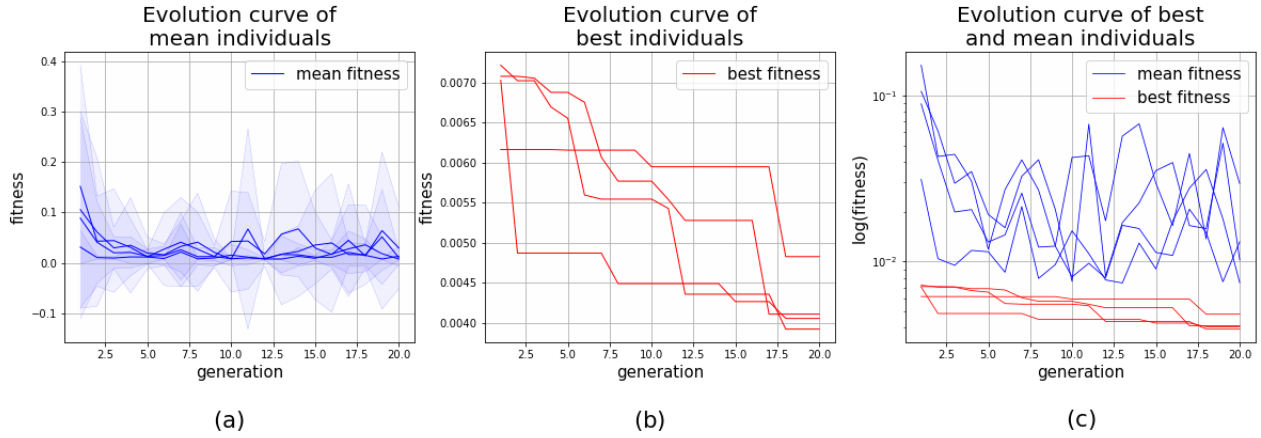


Figura A.1: Curvas de evolución de experimentos realizados en dataset MB. Evolución del *fitness* promedio de la población (a), evolución del mejor *fitness* de la población (b), y comparación, en escala logarítmica, de la evolución del promedio (azul) y del mejor (rojo) *fitness* de la población (c).

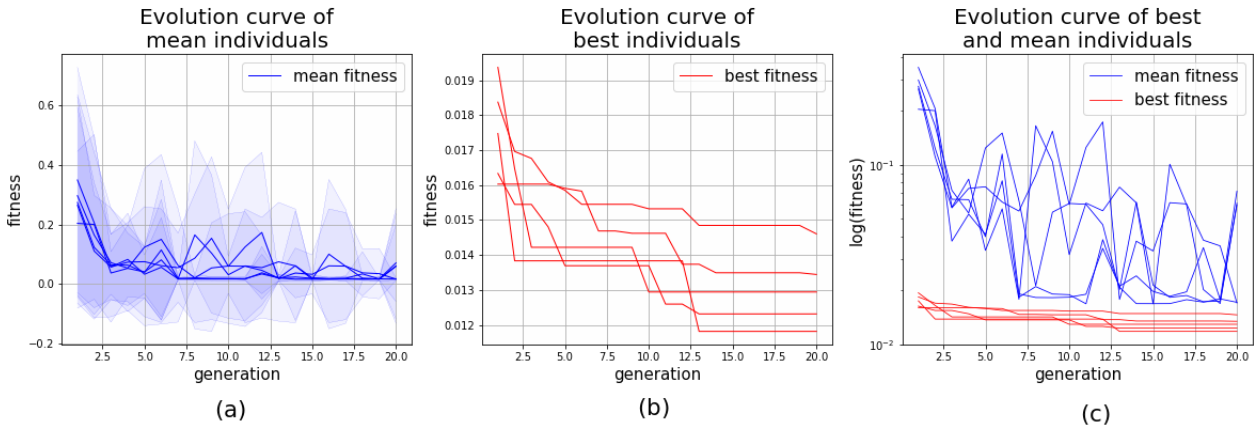


Figura A.2: Curvas de evolución de experimentos realizados en dataset MRB. Evolución del *fitness* promedio de la población (a), evolución del mejor *fitness* de la población (b), y comparación, en escala logarítmica, de la evolución del promedio (azul) y del mejor (rojo) *fitness* de la población (c).

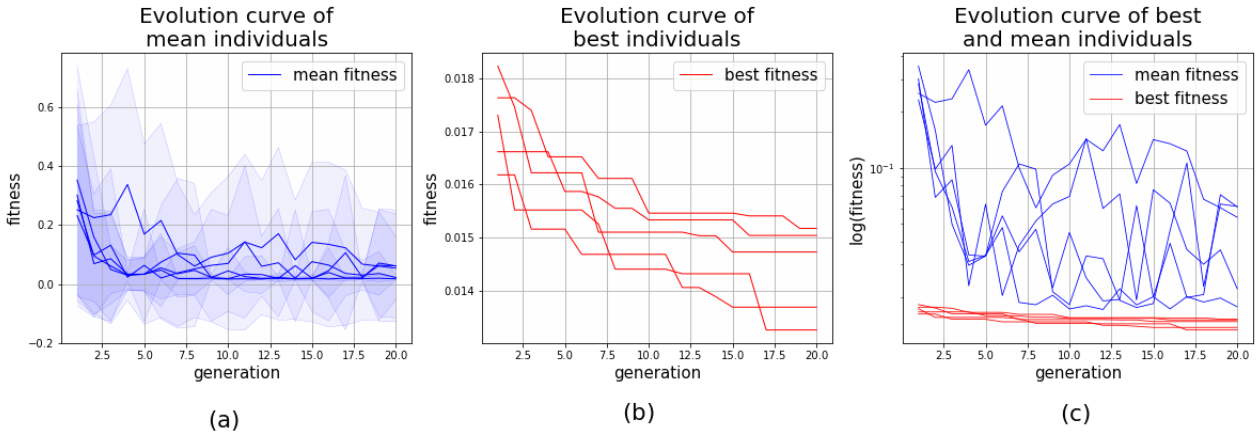


Figura A.3: Curvas de evolución de experimentos realizados en dataset MBI. Evolución del *fitness* promedio de la población (a), evolución del mejor *fitness* de la población (b), y comparación, en escala logarítmica, de la evolución del promedio (azul) y del mejor (rojo) *fitness* de la población (c).

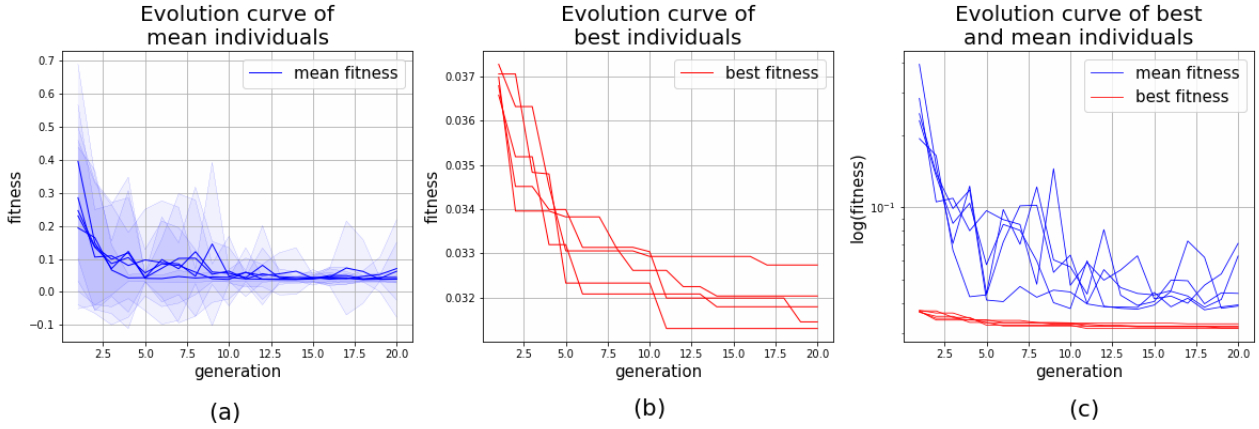


Figura A.4: Curvas de evolución de experimentos realizados en dataset MRD. Evolución del *fitness* promedio de la población (a), evolución del mejor *fitness* de la población (b), y comparación, en escala logarítmica, de la evolución del promedio (azul) y del mejor (rojo) *fitness* de la población (c).

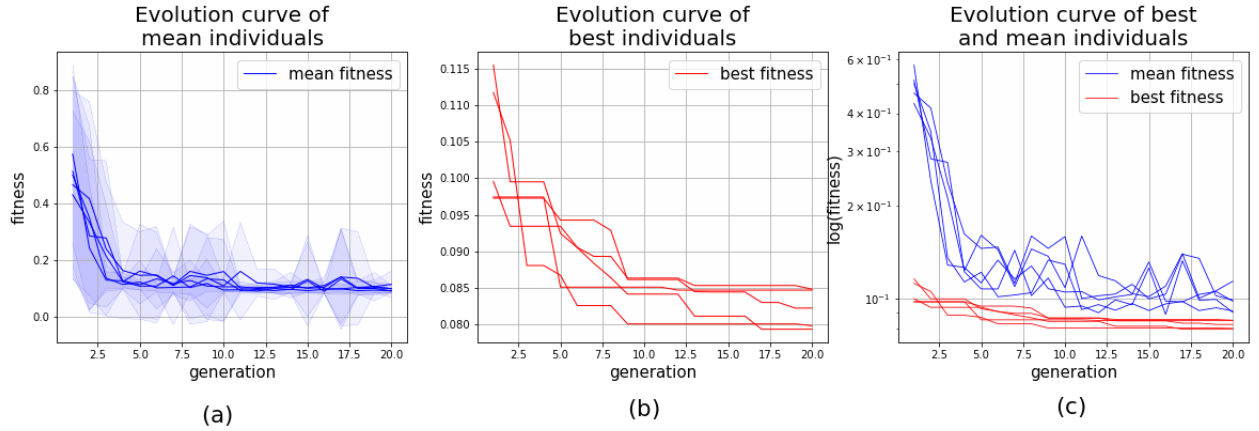


Figura A.5: Curvas de evolución de experimentos realizados en dataset MRDBI. Evolución del *fitness* promedio de la población (a), evolución del mejor *fitness* de la población (b), y comparación, en escala logarítmica, de la evolución del promedio (azul) y del mejor (rojo) *fitness* de la población (c).