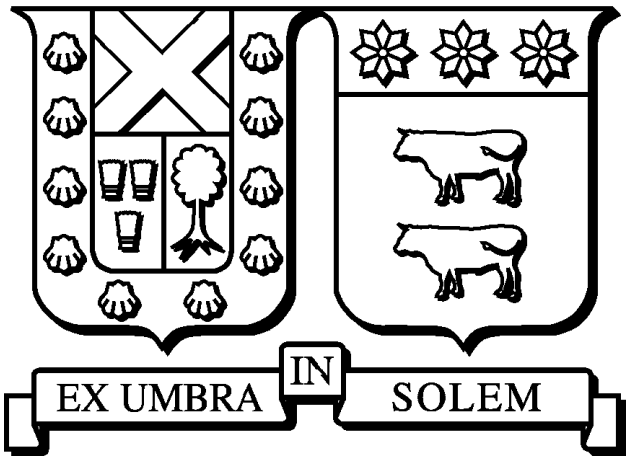
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

Departamento de Matemática
Valparaíso-Chile



Búsqueda de Hiperparámetros de Redes Neuronales Convolucionales utilizando Metaheurísticas

Memoria presentada por:

Alvaro Gabriel Valderrama Bustos

Como requisito parcial para optar al título profesional de

Ingeniero Civil Matemático

Profesores Guías:

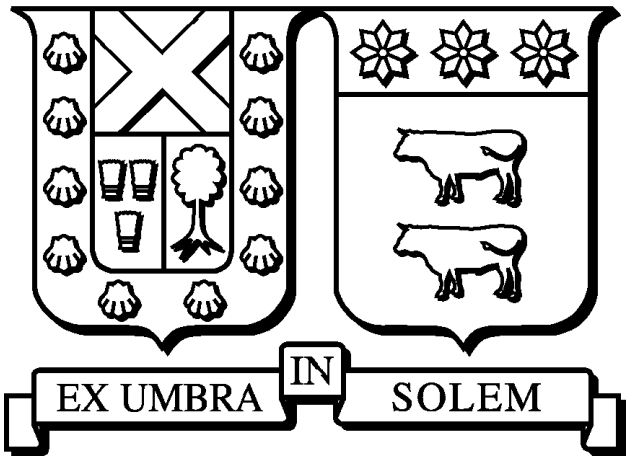
Carlos Antonio Valle Vidal

Cristopher Adrian Hermosilla Jimenez

Abril 2020

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

Departamento de Matemática
Valparaíso-Chile



Búsqueda de Hiperparámetros de Redes Neuronales Convolucionales utilizando Metaheurísticas

Memoria presentada por:

Alvaro Gabriel Valderrama Bustos

Como requisito parcial para optar al título profesional de

Ingeniero Civil Matemático

Profesores Guías:

Carlos Valle

Cristopher Hermosilla

Examinador:

Julio Deride

Abril, 2020

TÍTULO DE LA MEMORIA:

Búsqueda de Hiperparámetros de Redes Neuronales Convolucionales Utilizando Metaheurísticas.

AUTOR: Alvaro Gabriel Valderrama Bustos.

TRABAJO DE MEMORIA, presentado como requisito parcial para optar al título profesional Ingeniero Civil Matemático de la Universidad Técnica Federico Santa María.

COMISIÓN EVALUADORA:

Integrantes	Firma
Carlos Antonio Valle Vidal Universidad Técnica Federico Santa María, Chile.	_____
Cristopher Adrian Hermosilla Jimenez Universidad Técnica Federico Santa María, Chile.	_____
Julio Deride Silva Universidad Técnica Federico Santa María, Chile.	_____

Valparaíso, Abril 2020.

Agradecimientos

Agradezco a mi familia y pareja que siempre me han apoyado en todo ámbito. A todos mis amigos, que, a su propia manera, me han acompañado en este importante proceso. A mis compañeros que me ayudaron cuando lo necesité e hicieron que estos años pasaran entre risas. Y a mi profesor que siempre supo creer en mí y confiar en mis habilidades.

A mi familia y amigos.

Índice general

En los últimos años, en el campo de visión computarizada, las redes neuronales convolucionales (en adelante abreviadas CNNs por su nombre en inglés *Convolutional Neural Networks*) han sido utilizadas ampliamente para aplicaciones a problemas reales. Su desempeño de vanguardia, sin embargo, depende fuertemente de la arquitectura de la red dado un problema. En la mayoría de los casos, estas son optimizadas manualmente por los investigadores, proceso que tarda demasiado tiempo y es difícil de realizar sin tener conocimientos previos de CNNs. En esta memoria, proponemos un algoritmo genético particular para la optimización de las arquitecturas de CNNs para el problema de clasificación de imágenes. Este algoritmo extiende y refina la investigación actual en el campo, permitiendo exploración de la profundidad, cruzamiento secuencial, definiendo de manera más detallada el problema de optimización subyacente y sus variables, y controlando de maneras especificas la convergencia del algoritmo. La técnica se valida en tres conjuntos de datos de clasificación de imágenes ampliamente usado en el campo, y se compara con las propuestas del estado del arte de los pares. En todos menos un caso obtuvimos mejores resultados, o resultados equivalentes en un número significativamente menor de generaciones. El único caso donde nuestra propuesta fracasó frente a los pares, lo atribuimos a la inclusión de técnicas convolucionales recientes y no al algoritmo de optimización propiamente tal.

Capítulo 1

Introducción

El área de aprendizaje de máquinas o *machine learning* (ML) y, en particular, las redes neuronales profundas o *deep learning*, han proporcionado varios resultados de vanguardia en diversos campos de estudio. La clasificación de imágenes es uno de los problemas actuales de visión artificial en el cual técnicas de *deep learning* son ampliamente utilizadas. Esta tarea, donde un algoritmo debe clasificar una imagen en una o varias categorías sin intervención humana, es importante para varios otros problemas igualmente, como detección de objetos o segmentación de imágenes. El problema de clasificación tiene igualmente relevancia en diversos problemas actuales reales, como, por ejemplo, la conducción automatizada donde un algoritmo debe clasificar objetos como personas, autos u obstáculos [30]; en productos como Google Photos, donde software permite de manera automatizada reconocer los rostros de las personas presentes en imágenes [17] y organizar álbumes por persona de manera automatizada; e incluso en el campo de la medicina, donde algoritmos de aprendizaje aprenden a clasificar imágenes de órganos como saludables o con presencia de cáncer [2]. Para abordar este problema, los métodos comúnmente más utilizados son las redes neuronales artificiales y el *deep learning*, particularmente las redes neuronales convolucionales profundas [24,31]. Estas arquitecturas han demostrado obtener mejores desempeños que los modelos tradicionales, incluso alcanzando rendimientos a nivel humano en algunos problemas [15].

Sin embargo, la precisión de estas redes está fuertemente relacionada con su estructura y

diseño. De hecho, algunos investigadores han demostrado mejoras significativas de desempeños simplemente encontrando buenas elecciones de estos hiperparámetros [23], los cuales en los casos más simples incluyen la profundidad, el número de filtros y el tamaño de estos filtros en cada una de las capas. Incluso con tan solo estos atributos a ajustar, el problema es bastante complejo, y aún no existe un método para encontrar la mejor arquitectura dado un problema. En la mayoría de los casos, el ajuste lo realiza cada investigador, de manera manual, a partir de su conocimiento del área o basándose en estructuras validadas en trabajos previos.

La literatura ha demostrado la utilidad de las aproximaciones con metaheurísticas para resolver problemas combinatorios complejos [10, 29], los cuales tomarían tiempo excesivo de resolver con métodos tradicionales de optimización o simplemente son imposibles de resolver directamente. Algoritmos basados en poblaciones de soluciones, como por ejemplo algoritmos genéticos [3], tienen resultados particularmente buenos en problemas de optimización con grandes espacios de búsqueda y funciones objetivo patológicas [5]. Actualmente hay investigación en el uso de metaheurísticas para optimizar hiperparámetros de redes neuronales convolucionales donde la metaheurística es utilizada para ajustar distintas propiedades de las redes, tales como tamaño del kernel o secuencia de bloques convolucionales en un problema específico. Sun *et al.* [35] utilizan algoritmos genéticos para optimizar una secuencia de bloques residuales o *skip blocks* en el original, y capas de MaxPool, incluyendo la optimización de la profundidad, pero fijando el tamaño de los kernels convolucionales. Otros investigadores [4,28] han utilizado arquitecturas de redes neuronales convolucionales más tradicionales y algoritmos genéticos para optimizar no solo la estructura de las redes si no también los parámetros de aprendizaje, lo que lleva a comparaciones entre redes entrenadas y redes no entrenadas durante el proceso evolutivo. La mayoría de los desarrollos actuales [4, 27, 28] utilizan el operador de cruzamiento del algoritmo genético tradicional, es decir mediante lista binaria, destruyendo la estructura de las redes de los padres al momento de crear la población de hijos, ignorando así la naturaleza subyacente del problema de aprendizaje de máquinas. En la presente investigación, proponemos un método para utilizar la metaheurística que busca responder a las preocupaciones descritas previamente, en particular la optimización de los parámetros de aprendizaje y la destrucción de estructuras de red exitosas durante el cruzamiento. Extendemos la investigación realizada en el

tema utilizando algoritmos genéticos para encontrar el óptimo número de filtros y tamaño de kernels de cada capa convolucional de nuestras redes, además de optimizar igualmente el número de capas convolucionales o profundidad de las redes. El desempeño de cada red está drásticamente influenciado por cambios en estos hiperparámetros, los cuales definen el número de parámetros libres de la sección convolucional de la red y su arquitectura. Igualmente diseñamos un nuevo operador de cruzamiento que toma en cuenta la estructura de cada individuo padre, conservando los números de filtros y tamaños de kernel de manera secuencial permitiendo que los progenitores hereden mejores rasgos a las siguientes generaciones. Con el presente trabajo investigativo buscamos validar nuestro diseño de un algoritmo genético para la optimización de los hiperparámetros de redes neuronales convolucionales, por un lado, facilitando el futuro diseño de este tipo de redes para investigadores sin conocimiento del campo y por otro lado ayudando a delimitar de mejor manera las particularidades de este problema tanto en términos de optimización, de aprendizaje automatizado y del proceso de búsqueda evolutiva observado.

Los siguientes capítulos se organizan de la siguiente manera. Primeramente, introducimos la naturaleza del problema de optimización en el capítulo 2 como consecuencia de las restricciones y objetivos del aprendizaje de máquinas. Luego en el capítulo 3, una corta introducción a las particularidades de las redes neuronales convolucionales es presentada, para luego discutir como estas son traducidas al problema de optimización. En el capítulo 4 discutimos conceptos clave de las metaheurísticas y las particularidades de los algoritmos genéticos y su adecuación al problema actual. Luego, en el capítulo 5, presentamos nuestra propuesta frente al problema descrito, primero dando una vista general de trabajos similares y sus falencias para en una segunda parte describir nuestra aproximación para resolver las falencias descritas. A continuación, en el capítulo 6 describimos nuestros experimentos y resumimos nuestros resultados, para finalmente presentar nuestras conclusiones y líneas de trabajo futuras en el capítulo 7.

Capítulo 2

Aprendizaje Automatizado

La actual investigación corresponde al área de aprendizaje automatizado o machine learning (ML), la cual ha producido algunas de las herramientas más exitosas para tareas de predicción en varios campos [8, 25]. A diferencia de otras aproximaciones (entiéndase, por ejemplo, en el caso de predicción de valores reales, herramientas como regresiones lineales, modelos ARIMA, etc.), en ML se evita seguir modelos teóricos preformulados para cada problema, sino más bien se buscan y utilizan algoritmos eficientes al momento de elegir el modelo que mejor representa la relación entre la respuesta y sus variables predictores, basándose en los datos disponibles. Es decir, no se pretende encontrar una modelación teórica de los fenómenos, sino más bien una herramienta que permita predecir el comportamiento futuro del fenómeno observado de la manera más precisa posible.

Actualmente se investigan distintos paradigmas de aprendizaje, entre ellos el aprendizaje supervisado, donde el algoritmo aprende a partir de datos etiquetados [22]; aprendizaje no supervisado, donde tenemos acceso a datos pero sin las etiquetas correspondientes y el algoritmo tiene que deducir las estructuras inherentes a los datos [9]; o el aprendizaje reforzado, en el cual un algoritmo es entrenado para realizar una tarea sin conocimiento *a priori* de la utilidad o valor de sus diferentes posibles acciones [36], y debe evaluar el efecto que tienen sus distintas elecciones en el medio donde se desenvuelve.

En nuestro caso particular, estamos ante una tarea de aprendizaje supervisado. En términos generales, una tarea de aprendizaje supervisado puede describirse formalmente como sigue.

Sea $\mathcal{X}$ el espacio de características, es decir, el espacio desde el cual las variables de entrada o predicción de la tarea se obtienen (en general un subconjunto de $\mathbb{R}^n$ para algún $n \in \mathbb{N}$), e $\mathbb{Y}$ el espacio de salida, *i.e.* el espacio desde donde se obtienen los valores de la variable objetivo. Si denotamos por $\mathcal{S}$ el conjunto de todos los posibles pares $(x, y) \in \mathcal{X} \times \mathcal{Y}$ obtenidos de una distribución conjunta $P(x, y)$ desconocida (distribución teórica que representa el proceso que genera los datos observados); y por $\mathcal{H}$ el espacio de hipótesis, el espacio de todas las soluciones funcionales $f : \mathcal{X} \longrightarrow \mathcal{Y}$ permitidas por el diseñador del algoritmo; entonces, un algoritmo de aprendizaje $\mathcal{A}$ es un mapeo

$$\begin{aligned} \mathcal{A} : \mathcal{S} &\longrightarrow \mathcal{H} \\ S_t &\longmapsto f, \end{aligned}$$

donde S_t denota el conjunto de datos de ejemplos (que no es más que una realización de la distribución P) y f es llamada una hipótesis. El objetivo del algoritmo de aprendizaje para la tarea de aprendizaje supervisado es, dada la realización S_t, encontrar la hipótesis $\mathcal{A}(S_t) = f$ tal que el criterio de error elegido sea minimizado, es decir:

$$\mathcal{A}(S_t) = \underset{f \in \mathcal{H}}{\operatorname{argmin}} \sum_{(\mathbf{x}, y) \in S_t} \ell(f(\mathbf{x}), y). \tag{2.0.1}$$

Aquí, $\ell : \mathcal{Y} \times \mathcal{Y} \longmapsto \mathbb{R}^+$ es denominada función de pérdida, una función positiva que busca medir la diferencia entre los valores predichos por la hipótesis f y los valores de la muestra, y por lo tanto cuantifica la calidad de la hipótesis propuesta frente a los datos. Usualmente corresponde a una distancia.

Sin embargo, dado el hecho que el conjunto de datos de ejemplo S_t de nuestro proceso es finito, un modelo suficientemente grande o complejo (es decir con un número elevado de parámetros libres) podrá eventualmente "memorizar" cada par de datos disponible, con error despreciable sobre el conjunto de datos. Este fenómeno en el mundo de ML es conocido como sobreajuste u *overfitting*, y es uno de los problemas comunes encontrados por muchos algoritmos de ML que exploran espacios de hipótesis de altas dimensiones. En este

caso, la hipótesis f seleccionada tendrá error cercano a cero sobre el conjunto de datos de entrenamiento, pero su desempeño sobre datos futuros será considerablemente menor [13]. En este caso decimos que el modelo no tiene capacidad de generalización, es decir, no logra generalizar el aprendizaje sobre un conjunto de observaciones a observaciones futuras. Por lo tanto, es esencial estimar la capacidad de generalización de la hipótesis seleccionada por el algoritmo, para lo cual necesitaríamos datos nuevos o desconocidos. En ML esto se logra separando el conjunto original de observaciones en dos subconjuntos disjuntos, uno llamado el conjunto de entrenamiento, el cual el algoritmo utiliza para seleccionar la hipótesis, y otro llamado el conjunto de validación, sobre el cual la hipótesis seleccionada es evaluada. La Figura 2.1 presenta el gráfico de los errores de entrenamiento y validación de un proceso de aprendizaje iterativo, representativo de lo observado en redes neuronales, por ejemplo.

Por otra parte, usualmente los algoritmos de aprendizaje dependen de hiperparámetros, valores que definen el comportamiento del algoritmo y que afectan su desempeño, e incluso, dependiendo del tipo de algoritmo, pueden alterar la dimensión del espacio de hipótesis. Sea Λ el conjunto de los posibles hiperparámetros de una familia de algoritmos para un problema dado y, para $\lambda \in \Lambda$, denotemos por $\mathcal{A}_\lambda$ y $\mathcal{H}_\lambda$ el algoritmo de aprendizaje y el espacio de hipótesis asociados a los hiperparámetros λ respectivamente. Si denotamos por S_t el conjunto de entrenamiento, entonces, dado $\lambda \in \Lambda$, el algoritmo de aprendizaje A_λ selecciona la hipótesis $f_{S_t}^\lambda$ que resuelve el siguiente problema de minimización:

$$f_{S_t}^\lambda := \mathcal{A}_\lambda(S_t) = \underset{f \in \mathcal{H}_\lambda}{\operatorname{argmin}} \sum_{(\mathbf{x},y)\in S_t} \ell(f(\mathbf{x}), y). \tag{2.0.2}$$

Utilizando esta notación, podemos formular el problema de optimización de hiperparámetros para una familia de algoritmos dada y un conjunto de datos como sigue:

$$\underset{\lambda \in \Lambda}{\text{mín}} \sum_{(\mathbf{x},y)\in S_v} \ell\left(f_{S_t}^\lambda(\mathbf{x}), y\right), \tag{2.0.3}$$

donde S_v denota el conjunto de validación y $f_{S_t}^\lambda$ es la hipótesis escogida por el algoritmo dados sus hiperparámetros y el conjunto de entrenamiento, como en (2.0.2), es decir, se busca encontrar los hiperparámetros que logren que el algoritmo de aprendizaje seleccione

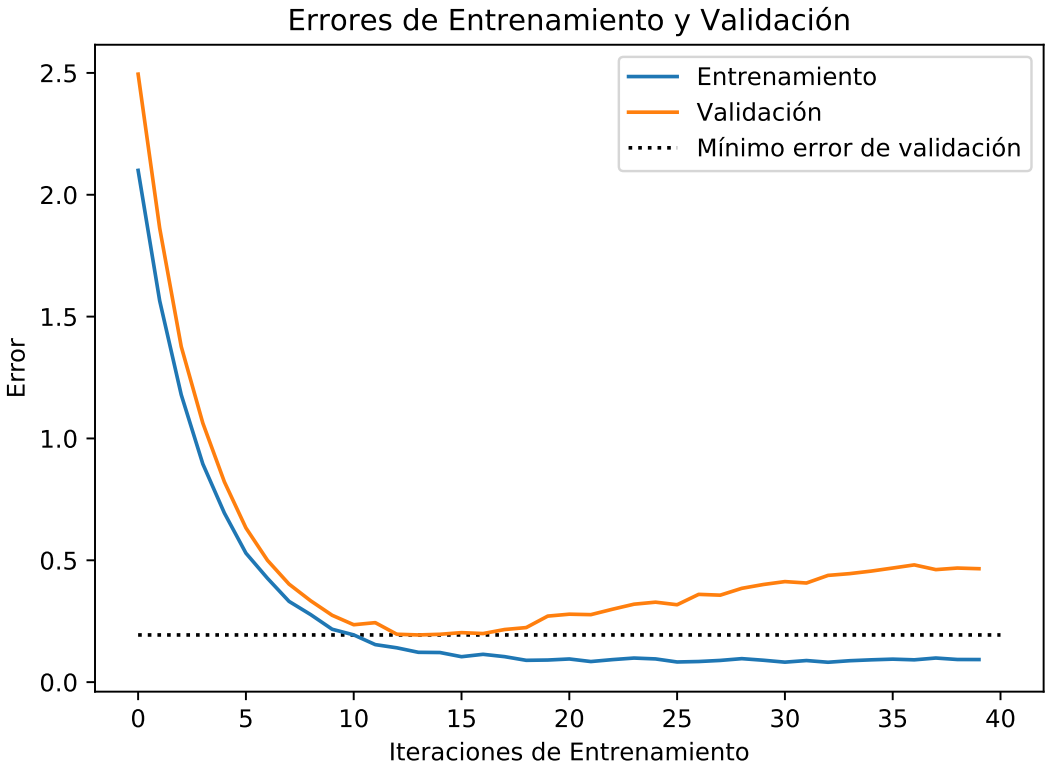


Figura 2.1: El gráfico muestra la evolución del error sobre conjuntos de entrenamiento y validación a lo largo de un proceso de entrenamiento iterativo. La línea punteada muestra el menor error de validación, es decir en esa iteración se alcanza el mejor modelo en términos de generalización. Antes de alcanzar ese punto el modelo no ha aprendido suficientemente bien la relación subyacente. Luego de ese punto, el modelo comienza a "memorizar" demasiado los datos que conoce, perdiendo capacidad para predecir datos futuros.

la hipótesis que mejor logra predecir datos futuros, independiente *a priori* de si selecciona la mejor hipótesis para los datos de entrenamiento.

Al considerar este problema, debemos superar dos principales dificultades. La primera es la ausencia de una relación conocida entre los hiperparámetros λ y la función objetivo, es decir, no hay manera *a priori* de saber en qué dirección un cambio de λ afectará la función objetivo. La segunda es el hecho que algunos algoritmos de ML, y en particular la mayoría (si no es todos) [26] los algoritmos actualmente utilizados para el entrenamiento de redes neuronales tienen una componente estocástica importante. Esto significa que no solamente no conocemos la relación entre λ y nuestra función objetivo, si no también que relaciones locales observadas mediante experimentación pueden ser resultados de las elecciones aleatorias involucradas en el proceso de aprendizaje.

En el siguiente capítulo, presentaremos una discusión más detallada de los hiperparámetros involucrados en las redes neuronales convolucionales en particular, y como estos modifican el comportamiento del algoritmo de aprendizaje subyacente y los espacios de hipótesis involucrados, para delimitar de mejor manera las particularidades del problema de optimización descrito.

Capítulo 3

Redes Neuronales Convolucionales

En aprendizaje de máquinas, redes neuronales artificiales se refiere a una familia de algoritmos de aprendizaje y sus modelos o hipótesis resultantes. Inicialmente inspirados en la estructura de las redes neuronales naturales encontradas en la corteza cerebral de animales, hoy en día hay una gran variedad de diferentes algoritmos y aproximaciones para desarrollar y entrenar distintas arquitecturas de redes neuronales artificiales [1].

Particularmente, en el área de reconocimiento de imágenes, las redes neuronales convolucionales (CNNs por su sigla en inglés) muestran resultados prometedores [24], en algunos casos incluso alcanzando o sobrepasando desempeños obtenidos por seres humanos [15]. Las CNNs obtienen estos resultados en parte gracias al patrón de conectividad particular usado por sus neuronas, inspirado originalmente en la organización de la corteza visual de los animales [18], en los cuales se observó que distintos patrones (*e.g.* líneas verticales o líneas horizontales) activan distintas rutas neuronales. Las CNNs replican este comportamiento usando los llamados filtros convolucionales, o kernels. Un kernel puede entenderse como un marco móvil, que barre la imagen completa, ponderando cada pixel de esta de acuerdo con los pesos del kernel. Al usar pesos fijos, el kernel extrae el mismo resultado cuando un patrón se repite en la imagen, independiente de su ubicación en esta.

Más formalmente, en el caso discreto, una convolución entre dos funciones reales f y g definidas sobre $\mathbb{Z}$, se denota por $f * g$ y se define como sigue:

$$\begin{aligned}(f * g) : \mathbb{Z} &\longrightarrow \mathbb{R} \\ n &\longmapsto \sum_{m=-\infty}^{\infty} f(m)g(n-m).\end{aligned} \tag{3.0.1}$$

La definición (3.0.1) se extiende de manera natural al caso 2-dimensional, donde tanto f como g mapean $\mathbb{Z}^2$ a $\mathbb{R}$ por:

$$\begin{aligned}(f * g) : \mathbb{Z}^2 &\longrightarrow \mathbb{R} \\ (i,j) &\longmapsto \sum_{(p,q)\in\mathbb{Z}^2} f(p,q)g(i-p,j-q).\end{aligned} \tag{3.0.2}$$

Si bien esta definición involucra una suma infinita, en la práctica no es el caso en este contexto. Supongamos que definimos g es nuestro filtro convolucional, en este caso g tendrá soporte finito (*i.e.* el subconjunto de $\mathbb{Z}^2$ donde g es no nula es finito). Por lo tanto, la convolución en un punto puede calcularse sumando solamente sobre los términos donde g es no nula. Lo mismo aplica para la imagen f, donde su soporte vendrá dado por las dimensiones (en pixeles), y por lo tanto es finito.

Si representamos por f la imagen, entonces podemos computar el resultado y de un solo filtro convolucional asociado al kernel g como:

$$y(i,j) = \sigma\left[(f * g)(i,j) + b_0\right], \tag{3.0.3}$$

donde σ es llamada la función de activación (una función real, usualmente no lineal) y $b_0 \in \mathbb{R}$ es llamado el sesgo, una constante entrenable. El resultado y de la operación de kernel, llamado filtro en este contexto, es igualmente 2-dimensional, y generalmente la imagen de entrada f se *rellena* (en inglés *padding*) con valores cero en los bordes, de tal manera que se preserven las dimensiones originales luego de la operación de kernel sobre el dominio en el cual está definida la imagen.

Una vez procesada toda la imagen con la operación de kernel, obtenemos el filtro o mapa de características. En este, cada pixel es resultado de la combinación ponderada de cada

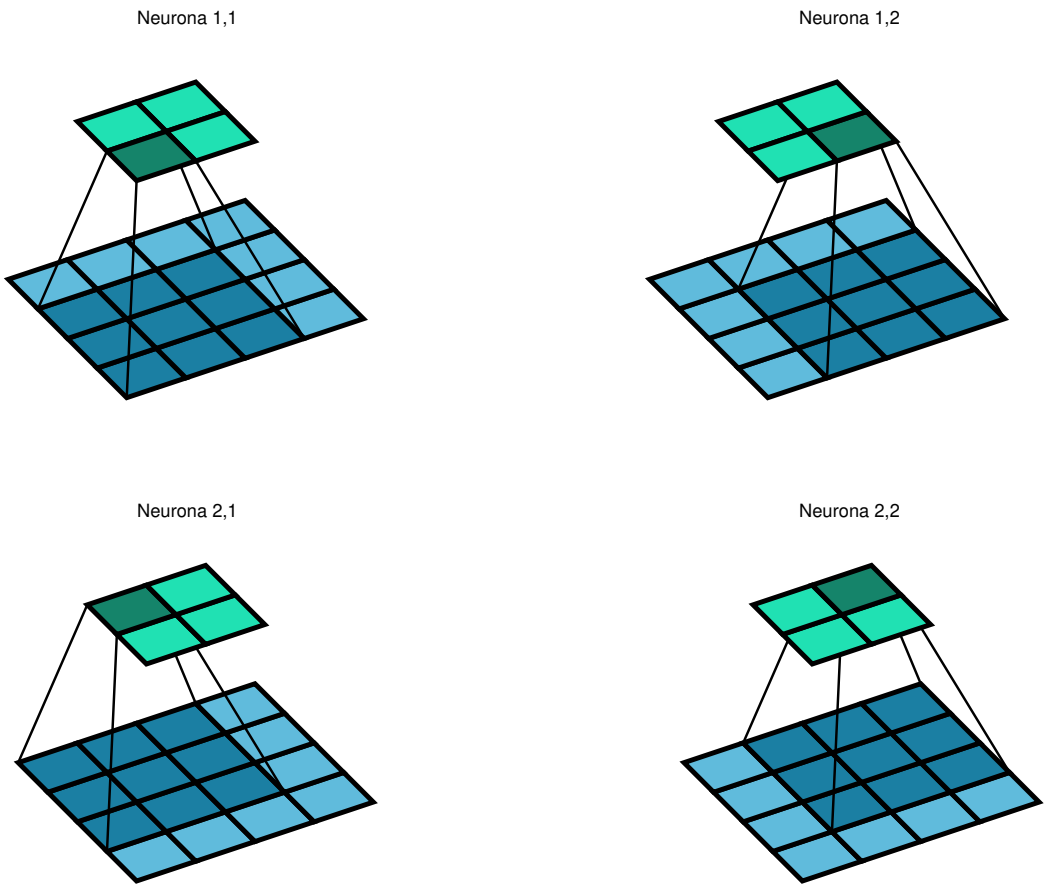


Figura 3.1: Imagen de una operación de convolución y las neuronas o pixeles resultantes. Como se puede observar, la imagen de arriba pierde dimensionalidad.

pixel de la imagen original a través de la operación de kernel. Los pesos del kernel son aprendidos por la red durante el proceso de entrenamiento. Esta aproximación les permite a los filtros de las CNNs identificar patrones independientemente de su posición al interior de las imágenes, al mismo tiempo que se reducen considerablemente el número de parámetros libres de las hipótesis al comparar con las arquitecturas densas o *fully connected*, donde cada valor del input se conecta mediante un peso entrenable a la salida, ayudando así a evitar el fenómeno de sobreajuste.

Para luego construir una capa de una CNN, varios kernels operan de manera paralela sobre la imagen, y los filtros resultantes son apilados. Inicialmente los kernels se inicializan con pesos aleatorios, los cuales se ajustan a lo largo del entrenamiento, aprendiendo así a

extraer patrones relevantes para el problema a resolver. La diversidad de kernels permite a la red, en teoría, aprender a extraer distintos patrones simultáneamente, permitiéndole a cada kernel especializarse en la extracción de alguna característica relevante particular.

Para construir una CNN completa, varias capas son concatenadas, aplicando el kernel de una capa sobre los filtros obtenidos en la capa anterior, y entrenando todas las capas simultáneamente. Aplicando capas sucesivas de kernels, la red puede aprender a extraer mapas de características de mayor nivel de abstracción, refinando secuencialmente los patrones observados en cada capa. Por ejemplo, uno podría imaginar fácilmente una red que identifique rectángulos, cuya primera capa identifique la presencia de bordes en las imágenes, la segunda capa identifique los lugares donde coinciden bordes distintos y la última capa que identifique la presencia de 4 bordes unidos.

Sin embargo, para las tareas de reconocimiento de objetos o clasificación de imágenes, la extracción de características y el reconocimiento de patrones no es suficiente. Usualmente estas habilidades de las redes se complementan con otros dos tipos de capas, a parte de las capas convolucionales recién descritas. Habitualmente, una o más capas densas o *fully connected* son agregadas al final de la red, luego de las capas convolucionales [23]. Esta capa se comporta de manera distinta que las capas convolucionales. Una capa densa con un número n_d de neuronas opera de la siguiente manera: cada neurona corresponde a la aplicación de una función (usualmente no lineal) llamada función de activación, a una combinación lineal de los valores de la capa anterior. Así, si denotamos por σ la función de activación, x_i para $i \in \{1, \ldots, n_c\}$ los n_c valores de la capa anterior, podemos calcular el valor y_k retornado por la neurona k-ésima, para cada $k \in \{1, \ldots, n_d\}$, por la siguiente formula:

$$y_k = \sigma\left(b_0^k + \sum_{i=1}^{n_c} b_i^k x_i\right).$$

Aquí, los valores $b_i^k \in \mathbb{R}$ para $(i, k) \in \{0, \ldots, n_c\} \times \{1, \ldots, n_d\}$ corresponden a los pesos de la capa, y son ajustados al momento del entrenamiento.

Es fácil notar de esta definición que el número de parámetros de una capa densa crece tanto con el número de neuronas como con el número de valores de entrada, por lo cual el número de parámetros será muy grande en los casos en que sus datos de entrada tengan

alta dimensión. Considerando que las imágenes más pequeñas usadas usualmente en el campo de reconocimiento de imágenes tienen 32×32 pixeles RGB [7] (dando un total de 3072 valores), es importante lograr reducir la dimensionalidad de la representación de los datos antes de conectar las capas densas. Es por esto que las capas *Max Pooling* o *Maxpool layers* se utilizan usualmente en conjunto con las capas convolucionales [33]. Su operación permite extraer la información relevante de las capas anteriores al mismo tiempo reduciendo la dimensionalidad de la representación. Estas capas operan de manera similar que las capas convolucionales, en el sentido que "barren" la imagen con una ventana fija. A diferencia de las capas convolucionales, las capas de *maxpooling* no ponderan los valores encontrados, si no que solamente retornan el mayor valor entre los pixeles hallados. Para lograr reducir efectivamente la dimensionalidad, se configura la capa, por ejemplo, para que la ventana tenga una dimensión de 2×2, pero cada vez que se mueve, en vez de moverse una distancia de 1 pixel, lo hace una distancia de 2 pixeles; finalmente dividiendo en 2 la dimensión del resultado respecto a la imagen de entrada. Igualmente incluimos otras capas, usualmente utilizadas en el diseño de redes neuronales en general, como la capa de *batch normalization* y la capa *dropout*, las cuales describiremos a continuación.

Introducidas en una publicación científica del año 2015 [19], la capa de *batch normalization* corresponde a una técnica usualmente utilizada en redes donde la profundidad comienza a tener impacto en el proceso de entrenamiento (aparte de problemas de sobreajuste, existe el problema de desvanecimiento del gradiente [16] que implica que en la práctica las redes profundas no modifican los pesos de la mayoría de sus capas al momento de entrenar si no se aplican técnicas que remedien el problema). La técnica de *batch normalization* corresponde a utilizar una capa que normaliza los valores recibidos de la capa anterior a una desviación estándar y promedio constantes, definidos en la capa de *batch normalization*. Este promedio y desviación estándar son entrenables por la red, y toman un valor para cada dimensión del *input* de la capa de *batch normalization*, y en cada iteración se calculan sobre el "batch", o subconjunto del conjunto de entrenamiento utilizado para completar parcialmente una iteración. Si denotamos por $\mathcal{B}$ al "batch" o subconjunto de entrenamiento actual, y para $i \in \mathcal{B}$ notamos x_i las entradas de la capa de *batch normalization*, cada uno de ellos vectores en $\mathbb{R}^d$ para algún $d \in \mathbb{N}$, entonces podemos calcular los vectores y_i de salida de la capa de

batch normalization siguiendo la siguiente fórmula:

$$y_i = \gamma \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \beta,$$

donde:

$$\mu_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} x_i \,, \quad \sigma_{\mathcal{B}}^2 = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} (x_i - \mu_{\mathcal{B}})^2,$$

corresponden a la media y varianza de los valores de entrada dado el *batch* $\mathcal{B}$ y los parámetros de la red en la iteración actual, y $(\gamma, \beta) \in \mathbb{R}^{2 \times d}$ corresponden a los parámetros entrenables de la capa y $\epsilon > 0$ es un valor fijo arbitrario utilizado para estabilizar los cálculos en caso la varianza se acerque a cero.

Según los autores, esta aproximación facilita el proceso de aprendizaje pues soluciona un problema: al momento de actualizar los pesos de una capa para reducir el error, hay que tomar en cuenta que los pesos del resto de las capas varían igualmente, por lo cual los nuevos pesos luego recibirán valores distintos a los recibidos anteriormente. El *batch normalization* permite por lo tanto que los rangos de valores recibidos por una capa sean similares, independiente de los cambios de la capa anterior. La adición de los parámetros γ y β da la libertad a la capa de no centrar sus valores en cero necesariamente, si no aprender la mejor distribución para el problema dado. La literatura ha demostrado empíricamente la eficacia de este procedimiento, lo cual ha hecho que las capas de *batch normalization* sean una manera habitual de acelerar la convergencia durante el entrenamiento y mejorar el desempeño de las redes neuronales finales.

Por otro lado, las capas de *dropout* [34], implementan una técnica solo enfocada a reducir el sobreajuste, la cual ha demostrado mejorar la capacidad de generalización de la gran mayoría de las redes neuronales. Esto lo logran aplicando una "máscara" estocástica de ceros sobre los pesos de la red anterior. Es decir, en cada iteración del proceso del algoritmo, una fracción aleatoria predeterminada de los valores de salida de la capa anterior son transformados en cero, y de igual forma se mantienen fijos a lo largo del proceso de entrenamiento correspondiente a esa iteración. Luego, en las siguientes iteraciones, la máscara estocástica se computa nuevamente, permitiendo que cada neurona tenga una probabilidad de mantenerse fuera del proceso de entrenamiento en cada iteración. Los autores proponen

que este método permite que diferentes partes de la red aprendan a reconocer los patrones relevantes de los datos, creando redundancias que ayudan a mejorar el desempeño de la red en datos desconocidos. En la práctica este método enlentece el proceso de entrenamiento, sin embargo, al utilizar estas capas en la estructura de la red el desempeño en el conjunto de validación mejora significativamente.

Teniendo una estructura de CNN fija, nuestro espacio de hipótesis $\mathcal{H}$ está definido por todas las posibles combinaciones de valores reales para cada parámetro del modelo convolucional. Es decir, cada peso para cada kernel de cada capa convolucional, incluyendo los sesgos, y los pesos de las capas densas. Nuestro algoritmo de entrenamiento debe, por lo tanto, encontrar idealmente la mejor combinación de todos esos valores para nuestro problema, considerando el número finito de ejemplos que se tienen. El número de parámetros a fijar en algunas redes grandes pueden alcanzar cientos de millones [33], por lo cual encontrar incluso una combinación decente de parámetros ya es una tarea compleja de por sí. El proceso de entrenamiento que busca encontrar tal combinación de parámetros comienza inicializando los pesos de toda la red aleatoriamente, en intervalos definidos previamente [11]. Luego, métodos iterativos basados en gradiente descendiente estocástico son utilizados para ajustar los parámetros, minimizando el error total sobre el conjunto de entrenamiento. Dada una función de pérdida, el gradiente para el conjunto de entrenamiento y los valores actuales de los parámetros puede computarse y utilizarse para actualizar los pesos, siguiendo alguno de los métodos validados por la literatura [6, 20, 37]. En cada iteración, los pesos son actualizados, y, con esos valores, el error sobre el conjunto de validación puede ser computado nuevamente y evaluado.

Las capas convolucionales, de *maxpooling* y capas densas, son los bloques básicos de construcción de las CNN. Nos enfocamos en optimizar la estructura de estas secuencias de bloques, en particular los siguientes hiperparámetros:

- Profundidad: La profundidad de la red se define como el número de capas convolucionales de la red. Este valor tiene serias repercusiones en la habilidad del algoritmo para correctamente aprender la tarea, pues una red muy poco profunda no logra aprender los patrones complejos que representan las categorías de las imágenes, y una red demasiado profunda tiene una cantidad excesiva de parámetros y sobre ajusta.

- Numero de filtros: También es necesario decidir para cada capa convolucional el número de filtros o neuronas que tendrá. Este hiperparámetros es crucial para permitir la trasmisión de información a las capas más profundas, un valor muy pequeño de filtros puede causar la perdida de información significativa al atravesar esa capa. Nuevamente tenemos que tomar en cuenta el problema de sobreajuste al momento de aumentar este valor.
- Tamaño de kernel: También, para cada capa convolucional, debemos elegir el tamaño de la convolución o kernel. Por motivos de consistencia y simplicidad todos los kernels en una capa son del mismo tamaño, sin embargo, el tamaño óptimo dependerá del problema y de la estructura del resto de la red. Este valor será esencial al momento de permitir a la red aprender patrones que se presenten dentro de la imagen a distintas escalas.

Intencionalmente no consideramos otros posibles hiperparámetros, tales como el número de capas densas, el número de neuronas en cada capa densa o el número de capas *maxpooling*. Consideramos que esos parámetros pueden dejarse en valores por defecto validados y la sección convolucional de la red le permitirá aprender las características inherentes a los datos. Además, estos hiperparámetros están fuertemente relacionados a la dimensionalidad del problema y el espacio de hipótesis resultante, y por lo tanto deben ser tratados delicadamente para evitar sobreajuste o problemas de capacidad de cómputo.

Incluso ignorando los parámetros mencionados en el párrafo anterior, nuestro problema de optimización es complejo, considerando no solo un espacio de búsqueda grande, sino también el hecho que para distintos valores del hiperparámetro de profundidad, el número de hiperparámetros a optimizar varía. Debemos igualmente considerar que las CNN son algoritmos de aprendizaje estocástico complejos que toman considerable tiempo en entrenar, haciendo una búsqueda exhaustiva en la práctica imposible. Son estas características del problema que nos han llevado a considerar el uso de metaheurísticas por sobre otros métodos de optimización. En el próximo capítulo describiremos más en detalle las metaheurísticas y sus beneficios y falencias para nuestro problema.

Capítulo 4

Metaheurísticas

En los párrafos a seguir presentaremos los métodos de optimización combinatoria conocidos como metaheurísticas. Primeramente, presentaremos algunos de los desafíos encontrados en problemas de optimización combinatoria complejos y el compromiso que imponen sobre los algoritmos de solución. Luego describiremos los conceptos de exploración y explotación en el marco de heurísticas para luego entender el balance de estos conceptos en el diseño de algoritmos a partir de metaheurísticas. Finalmente describiremos más en detalle los algoritmos genéticos, como fueron originalmente propuestos, los mecanismos que implementan para manejar la exploración del espacio de búsqueda y por qué pensamos se adaptan bien a nuestro problema actual.

Un problema de optimización combinatoria es un problema donde el espacio de búsqueda, es decir los conjuntos de soluciones factibles al problema son discretos en cada caso. En términos generales, un problema de optimización combinatoria puede definirse por una tupla (I, h, m, g), donde:

- I denota el conjunto de posibles instancias del problema, usualmente definido por las características de este (por ejemplo, considerando el problema del vendedor viajero, las posibles instancias son los distintos grafos posibles asociados al problema teórico).
- h denota un mapeo $h : I \longrightarrow \mathcal{P}(\mathcal{C})$ del conjunto de posibles instancias del problema al conjunto potencia (es decir el conjunto de todos sus subconjuntos) del conjunto $\mathcal{C}$

de soluciones factibles del problema. Es decir, dado $x \in I$, $h(x)$ denota el conjunto de posibles soluciones a la instancia x del problema, o, en otras palabras, las soluciones que cumplen las restricciones impuestas por la instancia x y la definición del problema general.

- m por otro lado, denota un mapeo $m : I \times \mathcal{C} \longrightarrow R^+$. El valor $m(x, y)$, donde $x \in I$ e $y \in h(x)$, denota la calidad de la solución factible y para la instancia x del problema.
- g denota un objetivo, es decir $g \in \{\text{mín}, \text{máx}\}$.

Con estas definiciones, una instancia x de un problema de optimización combinatoria (I, h, m, g) donde $x \in I$, se resuelve si encontramos una solución óptima, es decir una solución factible $y \in h(x)$ tal que:

$$m(x, y) = g(\{m(x, y')|y' \in h(x)\}).$$

Esta definición formal general oculta muchas complicaciones que condicionan profundamente el diseño de métodos y algoritmos de resolución de problemas de optimización combinatoria. Podemos por ejemplo considerar que en algunos problemas la función de factibilidad h no está definida explícitamente, si no implícitamente por un conjunto de restricciones que pueden ser en sí mismas complejas de verificar, por lo cual en algunos problemas incluso encontrar una solución factible de una instancia puede ser un problema costoso computacionalmente. Más aún, la función de calidad del problema m, comúnmente llamada función objetivo, puede ser costosa de computar para soluciones factibles, lo cual limita nuevamente la capacidad de explorar el espacio de búsqueda de los algoritmos, como es nuestro caso donde para evaluar m es necesario entrenar una red convolucional en decenas de miles de ejemplos.

Sin embargo, el mayor problema en gran parte de la optimización combinatoria es que dada una instancia del problema, la cantidad de posibles soluciones factibles suele ser excesivamente grande, haciendo una búsqueda exhaustiva imposible en la práctica [21]. A esto podemos agregar la complicación que en algunos casos de estudios la función objetivo no se relaciona de manera "predecible" frente a cambios pequeños en las soluciones, es decir usualmente son funciones complejas o incluso desconocidas (la función objetivo puede ser

por ejemplo el resultado de un proceso estocástico, o el resultado de otro problema de optimización) y con comportamiento atípico o patológico (con muchos óptimos locales y sin tendencias identificables). Este es de hecho el caso de nuestro problema de optimización, donde la función a optimizar se calcula de manera costosa y su valor depende de procesos estocásticos resultantes de un problema de optimización cuya instancia viene dada por la solución factible del problema original.

Son estas dificultades que han incentivado el desarrollo de heurísticas y metaheurísticas, reconociendo la imposibilidad de búsquedas exhaustivas y la falta de métodos numéricos directos para encontrar soluciones razonablemente buenas. En general, las heurísticas operan explorando el espacio de soluciones posibles del problema, balanceando dos estrategias contrapuestas: la exploración o diversificación de la búsqueda frente a la explotación o intensificación de la búsqueda.

La exploración o diversificación de un proceso de búsqueda, se refiere a la habilidad de un algoritmo de evaluar soluciones factibles representativas de diversas zonas del espacio de búsqueda. En cambio, la explotación o intensificación de la búsqueda corresponden a los mecanismos que implementa el algoritmo para, dentro de una zona acotada del espacio de búsqueda, buscar mejores soluciones que las conocidas hasta el momento. Ninguna de ambas estrategias entrega buenos resultados por sí misma, basta con considerar los casos extremos: un algoritmo que solamente utilice exploración podría ser la búsqueda completamente aleatoria de soluciones buenas, confiando solamente en la probabilidad de obtener un buen resultado eventualmente, mientras una búsqueda completamente intensificada, a partir de una solución factible exploraría exhaustivamente todo el espacio cercano a esa solución, explorando una fracción ínfima del espacio de búsqueda. Es la interacción entre estas dos estrategias que permite a las heurísticas obtener buenos resultados en tiempos razonables.

La creación e implementación de heurísticas implica un ajuste del balance entre ambas estrategias, el cual a veces es especifico no solo a la heurística o el problema si no a la instancia del problema. Incluso las heurísticas más simples como *Hill Climbing* (estrategia de búsqueda local donde partiendo de un punto inicial se varia en una pequeña cantidad una de las coordenadas del punto y se evalúa el cambio de la función objetivo, luego repi-

tiendo el proceso iterativamente sobre puntos mejores y reiniciando el proceso sobre puntos aleatorios una vez se converge a extremos locales) requieren balancear ambas estrategias, para asegurar que el algoritmo encuentre soluciones suficientemente buenas sin estancarse en óptimos locales lejanos al óptimo global. Una de las ventajas de las metaheurísticas es que sus estrategias de alto nivel usualmente implican un balance entre explotación y exploración desde el mismo diseño de estas. En los siguientes párrafos presentaremos una breve introducción al mundo de metaheurísticas, en particular a algoritmos bioinspirados.

Las metaheurísticas surgen como estrategias de búsqueda e implementación de heurísticas que permitan definir tácticas de búsqueda exitosas no solo para distintas instancias de un problema si no también para problemas de distintas naturalezas. Gran parte de la investigación en estos temas es de carácter empírico precisamente por la amplia gama de posibles problemas que abordan y por surgir como soluciones a problemas que no son adecuados para resolver con métodos directos o iterativos, los cuales sí tienen mayor desarrollo teórico en sus cuerpos respectivos del conocimiento. Sin embargo, la literatura ha demostrado reiteradas veces la utilidad y versatilidad de varias metaheurísticas [5] en diversos ámbitos de optimización. Para enfrentar el desafío definido en las secciones anteriores optamos por utilizar la metaheurística de los algoritmos genéticos, la cual se enmarca en la familia general de metaheurísticas bioinspiradas. Estas metaheurísticas se caracterizan por emular mediante el proceso de búsqueda fenómenos observados en el mundo natural, en particular fenómenos que se relacionan con el manejo y búsqueda de información por parte de sistemas naturales. Por ejemplo, el algoritmo de colonia de hormigas, o *ant colony optimization* (ACO), se inspira en el proceso de búsqueda de alimento observado en colonias de hormigas. Probablemente dada la naturaleza espacial del fenómeno que inspiró originalmente el algoritmo, la literatura demuestra que este se comporta particularmente bien para problemas de optimización de carácter espacial (como por ejemplo el problema del vendedor viajero), e incluso sobre algunas familias de problemas sobre grafos su convergencia es asegurada [12].

Por la naturaleza de las redes neuronales convolucionales, su comportamiento frente a cambios en los hiperparámetros relevantes para esta investigación y la interacción entre secuencias de hiperparámetros consecutivos (en nuestra representación) es que optamos por utilizar algoritmos genéticos para nuestro problema. El algoritmo genético canónico se

estableció como un método de optimización que intenta replicar los mecanismos mediante los cuales poblaciones de individuos de una especie evolucionan a lo largo del tiempo encontrando formas novedosas y eficientes de sobrellevar las dificultades del medio ambiente. Para emular el proceso evolutivo, los algoritmos genéticos operan de forma iterativa sobre una población de soluciones factibles del problema, o "individuos" en este contexto. El objetivo del algoritmo es hacer evolucionar esta población hasta que las soluciones converjan a algún óptimo razonablemente bueno.

Para este proceso, primeramente, cada individuo debe estar codificado, es decir toda la información de esa solución en particular debe estar codificada siguiendo un código común a todos los individuos (por ejemplo, si una solución corresponde a un punto en $\mathbb{R}^n$, una codificación podría ser la tupla de sus coordenadas cartesianas, o alguna otra representación que describa inequívocamente tal punto). En este contexto hablamos de esa representación de la solución codificada como cromosoma, pues será este el equivalente a la información genética portada por los individuos de una especie. A cada individuo se le calcula luego su rendimiento en términos de la función objetivo, asociándole luego un valor de aptitud que represente la calidad de la solución tentativa asociada a este individuo frente al resto de la población (por ejemplo, el ranking dentro de la población; pero puede ser distintas con tal que representen la calidad del individuo frente a sus pares). Este valor de aptitud es luego usado para asignar probabilidades de cruzamiento a cada individuo, proporcionales a su aptitud. De esta forma los individuos exitosos tendrán más probabilidades de reproducirse frente a individuos menos exitosos, de igual forma que en la evolución natural.

El proceso de cruzamiento del algoritmo genético permite representar la función de la reproducción en la evolución natural: la recombinación de rasgos exitosos para crear nuevos individuos. De igual forma viene asociado a la representación cromosómica y debe representar una combinación de la información contenida en los cromosomas de ambos individuos padres. Usualmente se seleccionan al azar dos individuos para realizar el cruzamiento, siguiendo una probabilidad proporcional a la aptitud de cada individuo padre. Luego se generan dos individuos nuevos, hijos que obtendrán información genética de ambos de sus padres, por ejemplo, creando sus cromosomas con partes aleatorias de cada uno de los cromosomas de los padres.

Igualmente se implementa una operación de mutación, que busca representar el proceso natural de mutación genética espontánea que a lo largo de largos periodos de tiempo nos ha dado la diversidad actual de especies biológicas. Para esto usualmente se asigna una probabilidad fija de que cualquier individuo mute, en cuyo caso se modifica alguno de los valores de su cromosoma para introducir nueva diversidad en la población.

Por otra parte, para intensificar la búsqueda se implementa una operación de selección ambiental, la cual busca representar la supervivencia natural de individuos mejor adaptados al medio ambiente. Este operador permitirá, mediante una probabilidad proporcional a la aptitud, que individuos exitosos tengan una posibilidad de sobrevivir inalterados a la generación siguiente. De esta forma, cuando se utiliza el operador de selección ambiental, la población converge en esperanza a una población donde abundan los rasgos que han demostrado ser estrategias exitosas en términos del problema de optimización.

Estos operadores trabajan en conjunto, balanceando la exploración que permite el operador de mutación y la intensificación que se origina del operador de cruzamiento y selección ambiental. Para avanzar a lo largo de las iteraciones del algoritmo, o generaciones como usualmente se les denomina, se evalúan las aptitudes de cada individuo de la población actual, y luego se crea la población siguiente con los hijos de los cruzamientos y los individuos mutados. Este algoritmo básico se describe en el Algoritmo 1. Igualmente, a veces se pueden agregar nuevos individuos aleatorios a la población para introducir aún más diversidad en la población. Otra estrategia usualmente utilizada para mejorar los desempeños del algoritmo es el llamado elitismo, el cual asegura que en cada generación el mejor individuo de esta sobreviva a la siguiente generación, preservando así siempre la mejor solución independientemente de los procesos estocásticos.

Es la combinación de todas estas estrategias que permite al algoritmo genético en general tener un buen balance entre exploración y explotación, entregando mejores resultados que lo que podría alcanzarse con algún método más exhaustivo considerando los costos computacionales involucrados. Es por esto por lo que optamos por utilizar este tipo de algoritmos para optimizar los hiperparámetros de las redes neuronales convolucionales para problemas de clasificación. En el siguiente capítulo describimos nuestra propuesta al problema actual y la motivación detrás de las elecciones realizadas. Para esto, primeramente presentamos

otros trabajos del área que han implementado estrategias similares, pero han tomado decisiones que nos parecen cuestionables, por lo que no permiten evaluar el potencial real de esta técnica para el problema propuesto; para luego describir nuestra propuesta particular y cómo esta responde a los reparos presentados.

Algorithm 1 Algoritmo genético general

```
1: Generar población inicial aleatoria.
2: Computar la función de evaluación para cada individuo.
3: while NOT criterio de término do
4:    for tamaño de población/2 do
5:       Seleccionar 2 individuos de la población para cruzamiento.
6:       Reproducir ambos individuos con cierta probabilidad asociada al rendimiento,
         obteniendo 2 descendientes.
7:       Mutar los individuos con cierta probabilidad.
8:       Computar la función de evaluación para los nuevos individuos.
9:       Insertar los nuevos individuos en la nueva generación
10:   end for
11:   if La población converge o máximo de iteraciones then
12:      criterio de término := TRUE
13:   end if
14: end while
```

Capítulo 5

Propuesta

5.1. Propuestas de pares

Algunos trabajos recientes han explorado el uso de metaheurísticas para aprender las estructuras de las CNNs para distintos problemas de reconocimiento de imágenes con buenos resultados. En [35] los autores utilizan algoritmos genéticos para optimizar la profundidad, secuencia y el número de kernels de cada capa para el conjunto de imágenes de dígitos manuscritos MNIST. Utilizan un desarrollo relativamente reciente en el campo de las CNNs, el uso de las llamadas *skip connections*, o redes residuales [14]. Esto les permitió alcanzar precisiones muy altas con las estructuras optimizadas. Sin embargo, ellos utilizan un tamaño de kernel fijo, un hiperparámetro que podría ser un factor a optimizar. En otra dirección, tanto [32] para CIFAR10 y [4] para MNIST, utilizan algoritmos genéticos para optimizar casi todos los hiperparámetros posibles para una CNN. Los autores no solo optimizan la estructura de la red, si no también intentan optimizar simultáneamente los parámetros del algoritmo de entrenamiento de la red. Si bien una buena elección del algoritmo de entrenamiento y sus parámetros puede mejorar drásticamente el desempeño de una red en particular, comparar distintas redes entrenadas con distintos métodos no permitiría diferenciar la red con mejor entrenamiento por sobre la red con mejor estructura. En contraste, para nuestros experimentos preferimos optimizar solo los hiperparámetros asociados a la estructura de las CNNs mientras entrenamos todas las redes con el mismo

método. Utilizamos el mismo algoritmo y parámetros de entrenamiento. Igualmente utilizamos un criterio de convergencia para detener el entrenamiento, en vez de un número fijo de *epochs* o iteraciones del algoritmo de entrenamiento. También decidimos no incluir el uso de redes residuales en nuestros diseños, considerando que las diferencias de las redes y el problema subyacente de optimización permitan considerarlo a este como un problema a parte.

En [27], los autores trabajan con el conjunto de datos Caltech-256, más demandante por su naturaleza y variedad. Para esta tarea, optimizan el tamaño del kernel, el número de kernels por capa convolucional mientras mantienen la profundidad de la red fija para todos los casos. Los autores utilizan además la operación de cruzamiento binaria del algoritmo genético original. Esto implica que la naturaleza secuencial de la estructura de la red no se preserva *a priori* al momento del cruzamiento y solo los valores exitosos se preservan a la siguiente generación, sin necesariamente de preservar secciones exitosas. Este es el caso igualmente de [4], donde los autores aplican el mismo cruzamiento binario para sus experimentos. En nuestra investigación exploraremos una aproximación secuencial al cruzamiento junto con la aproximación binaria.

Los resultados prometedores de la investigación reciente en este problema muestran que utilizar metaheurísticas es una aproximación interesante para la optimización de las estructuras de CNNs para un problema dado. En la siguiente sección describimos nuestra aproximación particular a este problema y sus particularidades. Explicamos igualmente el razonamiento detrás de nuestras elecciones.

5.2. Enfoque propuesto

En esta sección, presentamos nuestra aproximación particular a la optimización de la estructura de CNNs. En las siguientes subsecciones describimos como construimos los individuos de nuestra población, incluyendo sus secciones fijas y optimizadas. Luego discutimos nuestra elección de codificación para los cromosomas y los intervalos de definición relevantes involucrados, para después introducir los operadores de nuestro algoritmo genético. También presentaremos nuestras funciones de aptitud y métodos de selección, para finalmente dar una visión general de los hiperparámetros del algoritmo genético y el proceso de

entrenamiento de las CNNs subyacentes. En el Algoritmo 2 se describe de manera resumida todos los procedimientos descritos a continuación.

5.2.1. Plantilla de las redes

A continuación, describimos la plantilla para construir las redes convolucionales asociadas a cada individuo. Cada red tiene una sección convolucional, compuesta de una secuencia de capas convolucionales, *max pooling* y de *batch normalization* [19]. Las capas convolucionales están ordenadas en pares, cada par seguido de una capa de *max pool*, *dropout* [34] (de 10 %) y una de *batch normalization*. Luego de esta sección convolucional, sigue una sección *fully connected* o densa, compuesta de dos capas densas, cada una de ellas seguida de una capa *dropout* igualmente (esta vez de 50 %). Esta capa densa completa la red, como se ve en la Figura 5.1. La sección convolucional se encarga de extraer características de alto nivel de las imágenes de entrada, las cuales pueden luego ser interpretadas por las capas densas. Sus capas convolucionales le permiten a la red aprender a reconocer patrones a partir de los datos, mientras las capas de *batch normalization* ayudan el proceso de entrenamiento, como lo demuestra la literatura. Las capas de *max pooling* de esta sección logran que la red reduzca la dimensionalidad de la representación de los datos a medida la información avanza por las capas. Luego de la sección convolucional, las capas *fully connected* son capaces de aprender la relación entre las características extraídas y la categoría de cada imagen. Las capas de *dropout* ayudan este proceso al prevenir el overfitting durante el proceso de entrenamiento, como ha demostrado la investigación relevante.

Vale la pena mencionar que utilizamos relleno (o *padding*) "igual" para las capas convolucionales, es decir que extendemos los bordes de la imagen por valores 0 de tal manera que las dimensiones de entrada se preserven al momento de pasar por las operaciones convolucionales. Si no se realiza *padding*, en cada capa se pierde dimensionalidad, como se puede ver en la Figura 3.1. Optamos por este método para ayudar a definir más fácilmente los valores máximos y mínimos de número de capas convolucionales (considerando la restricción impuesta por la dimensionalidad del problema), al mismo tiempo permitiendo utilizar redes más profundas en conjuntos de datos de baja resolución. Por lo tanto, a lo largo de la sección convolucional, el único cambio de resolución en los datos viene dado por las capas de *max pooling*. En nuestro caso particular, si la imagen de entrada a la capa de *max pool*

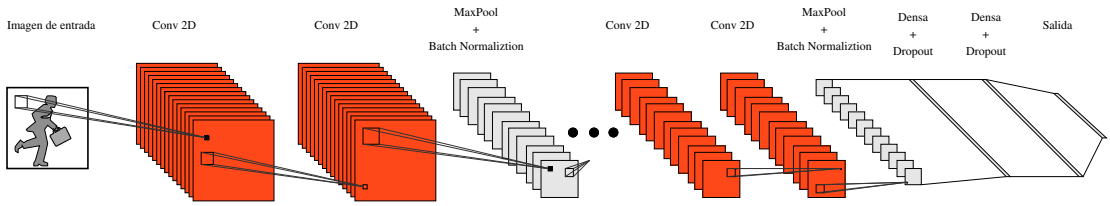


Figura 5.1: Ejemplo de una CNN. En nuestro caso, los hiperparámetros de las capas en rojo son optimizados, siendo fijos aquellos de las capas grises.

tiene dimensiones $d \times d$, la imagen de salida tendrá resolución $d' \times d'$ donde $d' = \left\lfloor \frac{d}{2} \right\rfloor$.

Optimizamos los hiperparámetros de la sección convolucional, en particular el número de capas convolucionales y los hiperparámetros de estas. La sección densa y las capas de *batch normalization* y *max pooling* son fijas (al menos en el sentido que no optimizamos sus hiperparámetros), como se puede ver en la figura 5.1.

5.2.2. Codificación de individuos

Ahora presentamos la codificación utilizada para representar estos individuos descritos, los intervalos relevantes de definición de sus valores, y su relevancia para el problema de aprendizaje de máquinas subyacente.

Para la codificación de un cromosoma, incluimos la información de profundidad, el número de filtros de cada capa convolucional y de tamaño del kernel utilizado en esta capa. Por simpleza, para representar un individuo con n capas convolucionales, utilizamos una lista de tamaño $2n + 1$, siendo el primer valor de esta n. Los siguientes $2n$ valores son pares de número de filtros en una capa y el tamaño de kernel utilizado en esa misma capa. Luego, si denotamos por L_i el número de filtros utilizados en la i-ésima capa convolucional y k_i su tamaño de kernel, el cromosoma correspondiente a tal individuo es el siguiente:

$$[n, L_1, k_1, L_2, k_2, \ldots, L_n, k_n].$$

Para definir los valores posibles que cada alelo (valor del cromosoma) puede tomar, primero debemos definir las profundidades posibles. Este valor dependerá fuertemente del conjunto de datos. Para los conjuntos de datos con menor resolución (tales como CIFAR10 con

32×32 pixeles en cada imagen), consideramos profundidades de 2 a 7 para la mayoría de nuestros experimentos. Con esta configuración no consideramos arquitecturas más profundas por las restricciones dimensionales. Para conjuntos de datos de mayor resolución (como Caltech256, el cual recortamos a un tamaño uniforme de 128×128 pixeles), fue posible considerar arquitecturas más profundas, de hasta 12 capas convolucionales, con un mínimo de 6. Impusimos un mínimo mayor para las imágenes más grandes, permitiendo suficiente reducción de dimensionalidad de las imágenes antes de la sección densa de la red, para evitar una explosión en el número de parámetros. Para los números de filtros de cada capa, utilizamos 4 como mínimo con un máximo de 128, mientras para los tamaños de kernel permitimos tamaños entre 2 y 9. Estos valores fueron elegidos considerando elecciones usuales hechas en arquitecturas del para problemas similares.

Es importante remarcar la importancia de estos hiperparámetros sobre el espacio de hipótesis. Mientras la elección de estos valores no afecta de manera directa las elecciones que realizará el algoritmo de aprendizaje (como sí lo haría cambiar los parámetros de entrenamiento), estos hiperparámetros delinean el espacio de hipótesis sobre el cual la máquina de aprendizaje (en este caso el optimizador elegido) puede buscar una solución para el problema de clasificación. Distintos valores de profundidad, número de unidades convolucionales o tamaños de kernel, cambian la dimensión del espacio de hipótesis y por lo tanto la complejidad de las funciones hipótesis posibles.

5.2.3. Operadores del algoritmo genético

A continuación, introducimos los principales componentes de nuestro algoritmo genético. Durante el diseño y las pruebas de nuestro algoritmo, implementamos unas cuantas estrategias enfocada a fomentar la exploración durante las primeras generaciones y la explotación hacia el final, buscando mejorar la convergencia del algoritmo.

Nuestro operador de mutación le asigna a cada individuo una probabilidad fija de mutar. Si un individuo muta, puede hacerlo de dos maneras diferentes, con una probabilidad de 50 % cada una. La primera opción es la modificación de uno de los alelos de su cromosoma, distinto a aquel que codifica la profundidad. Para esto, un alelo es elegido al azar del cromosoma exceptuando el primer valor. Luego su valor es remplazado por un valor al azar

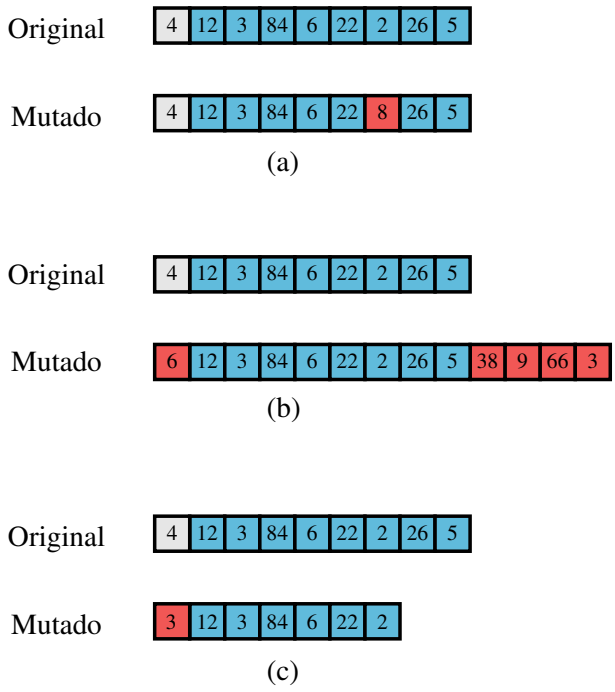


Figura 5.2: Ejemplos de los tres tipos de mutación: (a) alelo aleatorio dentro del cromosoma, (b) cambio de profundidad hacia un valor mayor y (c) cambio de profundidad hacia un valor menor. El color rojo representa nuevos valores aleatorios, el azul los valores de los alelos originales mientras el color gris representa el valor original de capas convolucionales (profundidad).

dentro del posible intervalo de valores que puede tomar ese alelo en particular (considerando si corresponde al número de filtros o al tamaño de kernel de una capa). La segunda opción de mutación corresponde a un cambio en el número n de capas convolucionales, en cuyo caso un nuevo valor de n es escogido aleatoriamente de los posibles valores. En el caso que este valor resultara ser menor al valor original, el cromosoma simplemente es recortado hasta alcanzar el tamaño correspondiente. En el caso contrario, si el nuevo valor es mayor que el valor original, el cromosoma se extiende por valores aleatorios escogidos de los intervalos relevantes. Estos tres posibles resultados de la mutación se muestran en la Figura 5.2.

Para la operación de cruzamiento, definimos igualmente dos aproximaciones, una basada en el cruzamiento binario del algoritmo genético original y otro método secuencial diseñado específicamente para preservar la estructura de las redes al momento de la reproducción. Si bien encontramos que el método binario no corresponde a la naturaleza secuencial de

la estructura que estamos optimizado, lo utilizamos de todas formas para aumentar la diversidad durante las primeras generaciones, pues su operación retorna redes radicalmente modificadas. A lo largo de la evolución, cada método tiene una probabilidad de ser usado, proporcional al progreso de las generaciones sobre el total de generaciones. La probabilidad exacta de utilizar cruzamiento secuencial en la generación g, si el total de generaciones que se explorará son G, será $\frac{g}{G}$, con la probabilidad de cruzamiento binario siendo $1-\frac{g}{G}$. Nuestra estrategia para el cruzamiento en general corresponde a iterar sobre cada individuo, dándole una probabilidad de aparearse proporcional a su función de aptitud. Si un individuo es seleccionado para apareamiento, una pareja es elegida aleatoriamente utilizando el mismo criterio (*i.e.* los individuos más exitosos tienen mayor probabilidad de ser seleccionados). Una vez ambos padres son determinados, el método de cruzamiento es elegido, siguiendo las probabilidades correspondientes. A continuación, describimos ambos métodos en detalle.

El cruzamiento binario primero genera una lista binaria aleatoria con un largo del doble de la mayor profundidad de los padres. Los cromosomas de los descendientes son inicializados con sus profundidades correspondiendo a alguno de ambos padres (cada padre está representado en un descendiente). Luego, el resto de cada cromosoma se llena siguiendo la lista binaria, es decir eligiendo valores de un padre u otro según la lista tenga valores 0 o 1 en la celda correspondiente, como se muestra en la Figura 5.3. Si el largo de un hijo excede aquel de uno de los padres, el resto del cromosoma se llena con los valores del padre con mayor profundidad. Este operador, si bien mantiene el mismo conjunto de valores de los cromosomas originales, nos entrega estructuras nuevas, radicalmente diferentes a las originales, al combinar estos valores de manera aleatoria. Esto fomenta la diversidad y exploración en las generaciones tempranas.

En las generaciones más tardías, sin embargo, buscamos intensificar la búsqueda. Logramos esto al permitir que el cruzamiento secuencial tome precedencia por sobre el binario. Este método de cruzamiento permite que la generación de hijos contenga secuencias de estructuras exitosas, reconociendo la naturaleza secuencial de la estructura optimizada. Sea C_a y C_b los cromosomas de los padres, con los siguientes valores:

$$C_a : [n_a, L_1^a, k_1^a, L_2^a, k_2^a, \ldots, L_{n_a}^a, k_{n_a}^a]$$
$$C_b : [n_b, L_1^b, k_1^b, L_2^b, k_2^b, \ldots, L_{n_b}^b, k_{n_b}^b].$$

El método primero elige aleatoriamente un punto de corte, un valor p entre 1 y el menor de los largos de los cromosomas de los padres, es decir $p \in \{1, 2, \ldots, min(n_a, n_b)\}$. Luego, los cromosomas S_1 y S_2 de los descendientes entregados por el método estarían dados por:

$$S_1 : [n_a, L_1^b, k_1^b, \ldots, L_p^b, k_p^b, L_{p+1}^a, k_{p+1}^a, \ldots, L_{n_a}^a, k_{n_a}^a]$$
$$S_2 : [n_b, L_1^a, k_1^a, \ldots, L_p^a, k_p^a, L_{p+1}^b, k_{p+1}^b, \ldots, L_{n_b}^b, k_{n_b}^b].$$

Notemos que este procedimiento mantiene la diversidad de profundidades de la población, sin perder las subestructuras exitosas, sino más bien recombinando estas para generar nuevas estructuras. Este método puede observase gráficamente en la Figura 5.4.

5.2.4. Evaluación de aptitud y selección

Para calcular la aptitud de un individuo primero debemos entrenar su red asociada completamente para poder medir su precisión sobre el conjunto de validación, es decir la fracción de ejemplos del conjunto de validación que la red logra clasificar correctamente. Describimos el proceso de entrenamiento en la última sección de este capítulo. Durante la primera mitad del proceso de evolución, calculamos la aptitud de un individuo proporcionalmente a su precisión sobre validación. Por lo tanto, si denotamos por A_i la precisión de validación del individuo i de la población P, su aptitud f_i puede computarse con la siguiente fórmula:

$$f_i = \frac{A_i}{\sum\limits_{j \in P} A_j}.$$

Podemos ver que la aptitud de un individuo representa la fracción que el contribuye al total

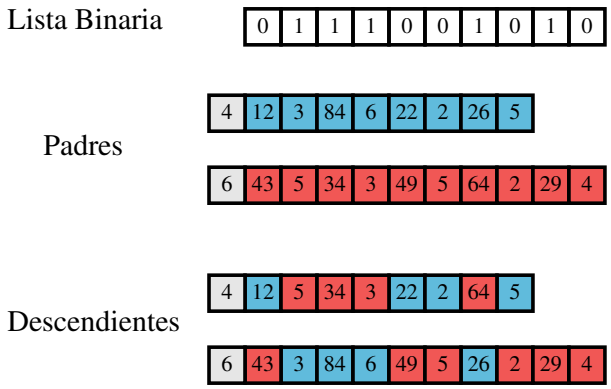


Figura 5.3: Ejemplo del método de cruzamiento binario. En blanco la lista binaria. Rojo y azul representan los genes de cada padre. El alelo de profundidad está representado en gris.

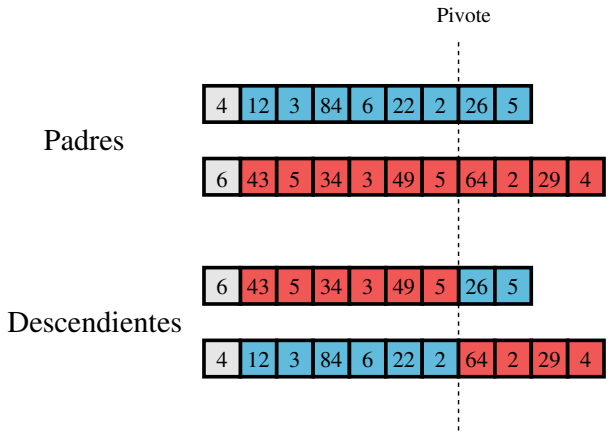


Figura 5.4: Ejemplo de método de cruzamiento secuencial. El punto "Pivote" es el punto p de corte elegido aleatoriamente.

de precisiones acumuladas por la población completa. Si bien esta aproximación diferencia fácilmente los individuos con desempeños pobres frente a buenos modelos predictivos en las primeras etapas, luego de unas cuantas generaciones observamos que las diferencias entre el mejor y el peor individuo se vuelven cada vez más pequeñas: el algoritmo comienza a converger a una solución. Eventualmente, observamos que todos los individuos tienen valores de aptitud similares y el algoritmo ya no puede diferenciar entre sus desempeños, por lo cual las operaciones genéticas se realizan en la práctica sin información de la calidad de cada individuo. Este es el motivo por el cual, luego de alcanzar la mitad de la evolución cambiamos la estrategia a una basada en un sistema de ranking. Para computar este valor de aptitud, primero debemos ordenar a todos los individuos de acuerdo con sus precisiones. Luego le asignamos un ranking a cada individuo, aquel con mayor precisión se le asigna el ranking 1, al segundo mejor el ranking 2, y así hasta llegar al peor con ranking $n + 1$ donde n es el tamaño de la población. Ahora, si denotamos por r_i el ranking del individuo i, podemos computar su aptitud con la siguiente formula:

$$f_i = \frac{n + 1 - r_i}{\sum_{j \in P} n + 1 - r_j}.$$

Esta fórmula es equivalente a la fórmula anterior si reemplazamos la precisión de validación por $n-1-r_i$, un valor que representa que tan bueno es el individuo cuando se compara con el resto de la población. A lo largo de toda la evolución, también utilizamos una aproximación elitista, es decir que, independiente de lo que ocurra con el resto de la población, el mejor individuo de cada generación siempre sobrevive a la siguiente, sin alterarse de forma alguna.

También preferimos mantener un número constante de individuos en la población. Esto lo logramos mediante dos métodos distintos. Durante la primera mitad de la evolución, la generación siguiente a la actual se inicializa con el mejor individuo, los individuos mutados y los hijos, ignorando los individuos de la generación actual que no han sufrido ningún tipo de modificación. Luego, la población deseada se obtiene generando nuevos individuos aleatorios que se incorporan a la siguiente generación. Esto hace que el único individuo que sobrevive inalterado lo hace a través del elitismo. De esta forma, durante las primeras generaciones obtenemos mayor diversidad. Los nuevos individuos incorporados, al ser to-

talmente aleatorios, traen rasgos variados que probablemente no se encontraban antes en la población.

Durante la segunda mitad de la evolución, implementamos una estrategia de selección evolutiva. En esta porción de la evolución, las nuevas generaciones están compuestas inicialmente por el mejor individuo y los individuos modificados (ya sea por mutación o cruzamiento) de igual forma que se realizaba anteriormente. Luego, en cambio, para obtener la población deseada, individuos de la generación anterior tienen una chance de sobrevivir a la siguiente generación, con probabilidad proporcional a su valor de aptitud. Esto permite que algunos individuos exitosos de las generaciones anteriores sobrevivan a las siguientes sin ser modificados en ninguna forma, intensificando la búsqueda del algoritmo y ayudándole a converger.

5.2.5. Hiperparámetros del algoritmo genético y entrenamiento de las redes convolucionales

A continuación describiremos los hiperparámetros relevantes del algoritmo genético comunes a todos nuestros experimentos, al igual que la configuración del algoritmo de entrenamiento con el cual se entrenaron todas las redes neuronales convolucionales subyacentes al problema.

Nuestros experimentos utilizaron una población constante de 12 individuos a lo largo de 12 generaciones. Utilizamos una tasa de mutación de 0,3, lo que significa que el número esperado de individuos con mutación corresponda al 30 % de la población. Utilizamos 0,25 como la probabilidad base de que un individuo tenga descendencia, de tal forma que el número esperado de hijos sea igual a 0,5 veces la población, pues cada individuo que se aparea engendra dos hijos.

Respecto al algoritmo de entrenamiento de las redes, cada individuo fue entrenado con el optimizador Adam [20] con sus parámetros por defecto, por sus buenos desempeños al momento de entrenar redes convolucionales. Durante el entrenamiento de todos los individuos monitorizamos su precisión sobre el conjunto de validación, y detuvimos el entrenamiento luego de 3 iteraciones consecutivas donde la precisión de validación no mejorara. Almacenamos los mejores valores obtenidos de la precisión sobre el conjunto de validación y

Algorithm 2 Algoritmo genético propuesto

```
Generar Población inicial aleatoria.
while NOT Generaciones > Máximo generaciones do
    Entrenar cada individuo utilizando ADAM, hasta obtener criterio de convergencia
    if Generaciones < Máximo Generaciones then
        Se calcula aptitud mediante valor de precisión
    else
        Se calcula aptitud mediante valor de ranking
    end if
    for Individuo en Población do
        if Individuo seleccionado estocásticamente para cruzamiento según su aptitud then
            while NOT Pareja Seleccionada do
                Elegir Pareja tentativa aleatoria de Población
                Pareja tiene chance de ser seleccionada estocásticamente según su aptitud
            end while
            if Con probabilidad igual a Generaciones/Máximo generaciones then
                Cruzamiento Individuo y Pareja de manera Secuencial
            else
                Cruzamiento Individuos y Pareja por lista binaria
            end if
            Se agregan ambos descendientes a Lista de Hijos
        end if
        if Individuo seleccionado estocásticamente para mutar then
            Se elije aleatoriamente tipo de mutación (Profundidad o valor)
            Nuevo individuo mutado se agrega a Lista de Mutados
        end if
    end for
    Nueva Población ← Mejor Individuo + Lista de Hijos + Lista de Mutados
    if Generaciones < Máximo generaciones/2 then
        Se completa Nueva población con nuevos individuos aleatorios hasta alcanzar tamaño deseado
    else
        while Nueva Población no tiene suficientes individuos do
            Se elije Individuo aleatoriamente de Población que no esté en Nueva Población con probabilidad asociada a su aptitud y se agrega a Nueva Población
        end while
    end if
    Población ← Nueva Población, Generaciones ← Generaciones + 1
end while
```

utilizamos esos para computar la aptitud de cada individuo. Igualmente, cuando un individuo, ya entrenado, sobrevive a la generación siguiente sin alterarse, no lo entrenamos nuevamente, pues tenemos en memoria la medida de su calidad a través de su precisión de validación calculada previamente. Vale la pena notar que la separación de los conjuntos de datos en conjunto de entrenamiento, validación y testing se realiza antes del proceso de evolución, y todos los procesos de entrenamiento y validación son, por lo tanto, realizados sobre los mismos conjuntos de ejemplos.

En el siguiente capítulo describimos los diferentes experimentos realizados con esta propuesta, describiendo las consideraciones particulares para cada conjunto de datos.

Capítulo 6

Resultados

Realizamos todos nuestros experimentos utilizando aceleración por tarjeta gráfica para el entrenamiento de las redes (Nvida RTX 2060 SUPER), apreciando la reducción considerable de tiempo de entrenamiento que este hardware permite. Utilizamos nuestro algoritmo propuesto para encontrar estructuras eficientes para la tarea de clasificación sobre los conjuntos de datos MNIST, CIFAR10 y Caltech256, repitiendo cada experimento 10 veces con distintas semillas para los procesos estocásticos del algoritmo genético, excepto para Caltech256 donde el costo computacional de la tarea solo permitió realizar una ejecución. A continuación describiremos más detalladamente la composición de cada conjunto de datos y sus particularidades, además de los ajustes necesarios al algoritmo para cada uno de ellos.

El conjunto MNIST (Modified National Institute of Standards and Technology database en su nombre completo) está compuesto de 70.000 imágenes en blanco y negro de 28×28 pixeles, cada una con un dígito (de 0 a 9) escrito manualmente. De estas, utilizamos 45.000 imágenes para el entrenamiento, 15.000 imágenes para la validación y reservamos 10.000 imágenes como conjunto de test. Este conjunto de datos es aquel que tiene la menor resolución de los tres, lo cual impone una restricción sobre la profundidad máxima que pueden alcanzar nuestras redes. Permitimos, en este caso, a nuestro algoritmo explorar profundidades entre 2 y 6, con número de filtros convolucionales de cada capa entre 4 y 64 y explorando tamaños de kernel de cada capa entre 2 y 9. Considerando la complejidad y tamaño de la base de datos, utilizamos tan solo 64 neuronas para las capas densas. Nuestro algoritmo

	MNIST	CIFAR10	Caltech256
Loussaief [27]	$99,29\,\% \pm 0,0006\,\%$	$77,8\,\% \pm 0,007\,\%$	$25,59\,\% \pm 0,014\,\%$
Reiling [32]	-	$85,27\,\% \pm 0,471\,\%$	-
Bhandare [4]	$97,2775\,\% \pm 0,916\,\%$	-	-
Propuesta actual	$99,37\,\% \pm 0,0046\,\%$	$83,44\,\% \pm 0,00464\,\%$	$28,83044\,\%$

Tabla 6.1: Tabla comparativa de los desempeños de los distintos algoritmos. Para resultados de Loussaief *et al.* [27] se replicó la metodología descrita por los autores. Para el resto de los casos se presentan los valores reportados por los autores.

obtuvo una precisión final sobre el conjunto de validación de $99,3666\,\% \pm 0,02558\,\%$, con profundidades promedio de $5,4 \pm 0,351$ capas convolucionales. Como medida de comparación, los valores finales reportados por [4] para 10 ejecuciones se traducen a un promedio de precisión de $97,2775\,\% \pm 0,9159\,\%$. Se puede observar la evolución de la precisión alcanzada sobre el conjunto de validación por las elites de los 10 experimentos a lo largo de las generaciones en la Figura 6.2, en un gráfico de diagrama de caja.

La segunda base de datos con la que trabajamos es CIFAR10 (Canadian Institute For Advanced Research 10 dataset originalmente). Está compuesta de 60.000 imágenes de 32×32 pixeles RGB, es decir arreglos de tamaño $32 \times 32 \times 3$. Notemos que cada imagen contiene aproximadamente 4 veces más información que la contenida en las imágenes del conjunto MNIST, pues a parte de tener mayor resolución, se incorpora información de coloración a cada pixel. Cada imagen de esta base de datos está etiquetada en una de las 10 siguientes categorías: avión, automóvil, pájaro, gato, ciervo, perro, rana, caballo, barco o camión. Para este conjunto permitimos a nuestro algoritmo explorar profundidades entre 2 y 7 capas convolucionales, con numero de filtros variando entre 4 y 128 y tamaños de kernel entre 2 y 9, con las capas densas finales compuestas por 128 neuronas cada una. Luego del proceso evolutivo, nuestro algoritmo encontró redes que alcanzaron una precisión promedio de $85,3522\,\% \pm 0,4649\,\%$ sobre el conjunto de validación con profundidad promedio de $6,2 \pm 0,4613$. Los resultados resumidos de la evolución de las 10 diferentes elites pueden observarse en el diagrama de caja en la Figura 6.3. Igualmente probamos una segunda configuración de la cual corrimos un único experimento, explorando redes más grandes. Para este consideramos un máximo de 256 filtros en cada capa y agregamos capas de maxpool cada tres capas convolucionales en vez de las dos utilizadas en los demás experimentos.

Para este experimento además utilizamos una población de 16 individuos por un total de 16 generaciones. Esta prueba entregó una mejor red que las pruebas anteriores, alcanzando una precisión sobre el conjunto de validación de $87,18\,\%$, pero con una arquitectura considerablemente más profunda de 10 capas convolucionales. La evolución de la elite de este experimento puede verse en la Figura 6.1, donde se aprecia empíricamente la convergencia de nuestro algoritmo, considerando que el mejor individuo encontrado mejora rápidamente a lo largo de las generaciones para luego converger a un valor relativamente estable. En [32], el autor reporta valores de precisión de $85,27\,\% \pm 0,471\,\%$ sobre un total de 5 experimentos. Sin embargo, vale la pena considerar que ellos exploraron por 150 generaciones, optimizando tanto la estructura de la red como los parámetros de entrenamiento, mientras nosotros alcanzamos resultados equivalentes con un proceso evolutivo mucho más corto y optimizando una cantidad más acotada de parámetros, realizando una búsqueda, por lo tanto, más eficiente.

Finalmente, la base de datos más compleja que utilizamos es la de Caltech256. Compuesta de 30.607 imágenes a color (RGB) de distintas dimensiones y relaciones de aspecto, Caltehc256 tiene 256 categorías distintas (entre las cuales podemos encontrar objetos tan diversos como ametralladoras AK-49, murciélagos, osos, dirigibles, Buda, harpas, tinas calientes, llamas, botellas de vino y unicornios) más una categoría adicional de "clutter" o desorden, la cual representa imágenes sin ningún objeto particular identificable en ellas. Para poder utilizar las imágenes, comenzamos por completar la dimensión más pequeña de la imagen con ceros de manera simétrica, hasta obtener una relación de aspecto cuadrada. Luego, transformamos todas las imágenes a 128×128 pixeles, una resolución que aún permitía identificar gran parte de las imágenes al visualizarlas. Igualmente utilizamos una separación de $75\,\%$, $15\,\%$ y $10\,\%$ para los conjuntos de entrenamiento, validación y test respectivamente, con la precaución de preservar las proporciones de representación de las distintas categorías, las cuales tienen ligeras diferencias en la cantidad de ejemplos. Para esta tarea, nuestro algoritmo explora profundidades entre 6 y 12 capas convolucionales, con 4 a 128 filtros en cada capa convolucional, y tamaños de kernel entre 2 y 9. Las capas densas del final de la red tienen 128 unidades. Los resultados de esta prueba fueron de $28,8330\,\%$, con una profundidad de la elite final de 8. En la Figura 6.4 se puede ver la evolución de la precisión de la elite a lo largo de las generaciones. En [27] se reporta una

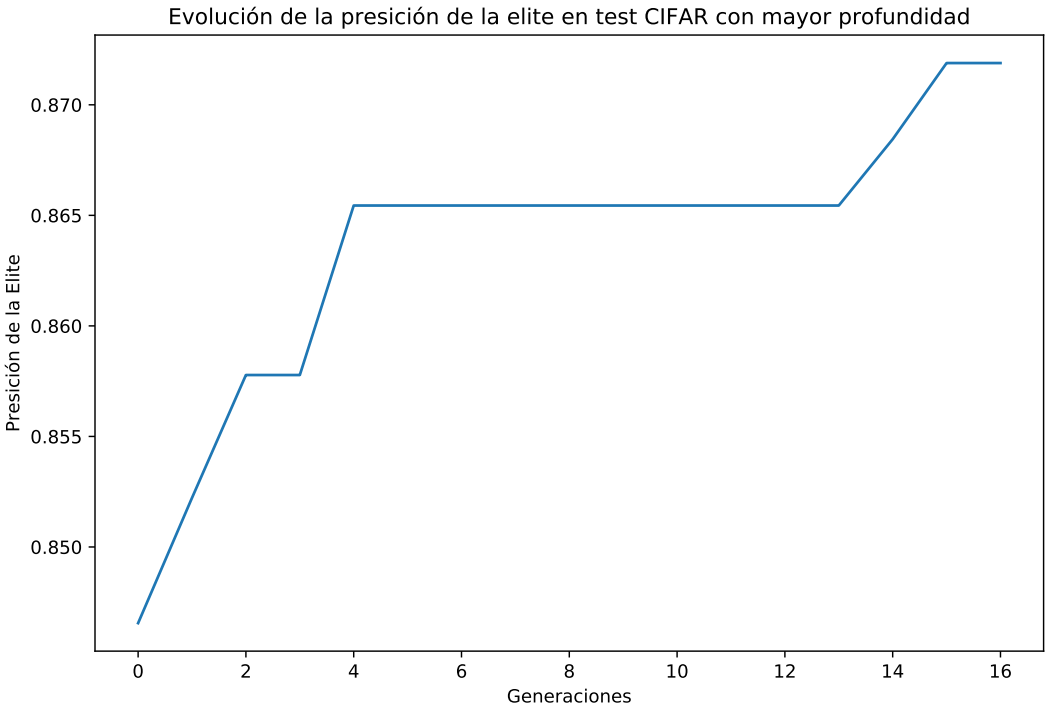


Figura 6.1: Evolución de la precisión de la elite de una sola ejecución del algoritmo sobre el conjunto de datos CIFAR10 con mayor profundidad. Se observa que el algoritmo converge luego de mejoras rápidas en las primeras iteraciones.

precisión final de $93,37\,\%$, sin embargo, fuimos incapaces de replicar sus hallazgos al momento de entrenar la arquitectura de red que supuestamente alcanzaba tal precisión. En cambio, encontramos que el mejor desempeño logrado con su red fue de no más de $25\,\%$ en varios intentos, mientras los valores de precisión sobre el conjunto de entrenamiento se acercaban más a los valores reportados. Creemos por lo tanto que en [27] se reportaron los valores de precisión sobre el conjunto de entrenamiento en lugar de los valores sobre conjunto de validación, sin embargo, los autores no respondieron a nuestras consultas. Por lo tanto, replicamos sus experimentos y corrimos 10 iteraciones para poder comparar los resultados sobre las métricas adecuadas. Logramos encontrar una precisión sobre validación de $25,9049\,\% \pm 0,2499\,\%$. Aprovechamos la instancia de utilizar el método propuesto por los autores en los otros dos conjuntos de datos, cuyos resultados pueden verse en la Tabla comparativa 6.1.

Los resultados de los experimentos descritos pueden verse resumidos en la tabla comparativa, cuando. Cuando es posible se entrega intervalo de confianza mediante t-Student con $5\,\%$ de significancia. El único experimento no resumido en la tabla es la versión profunda para CIFAR10, donde se alcanzó $87,17\,\%$ en una sola corrida, superando a ambos pares.

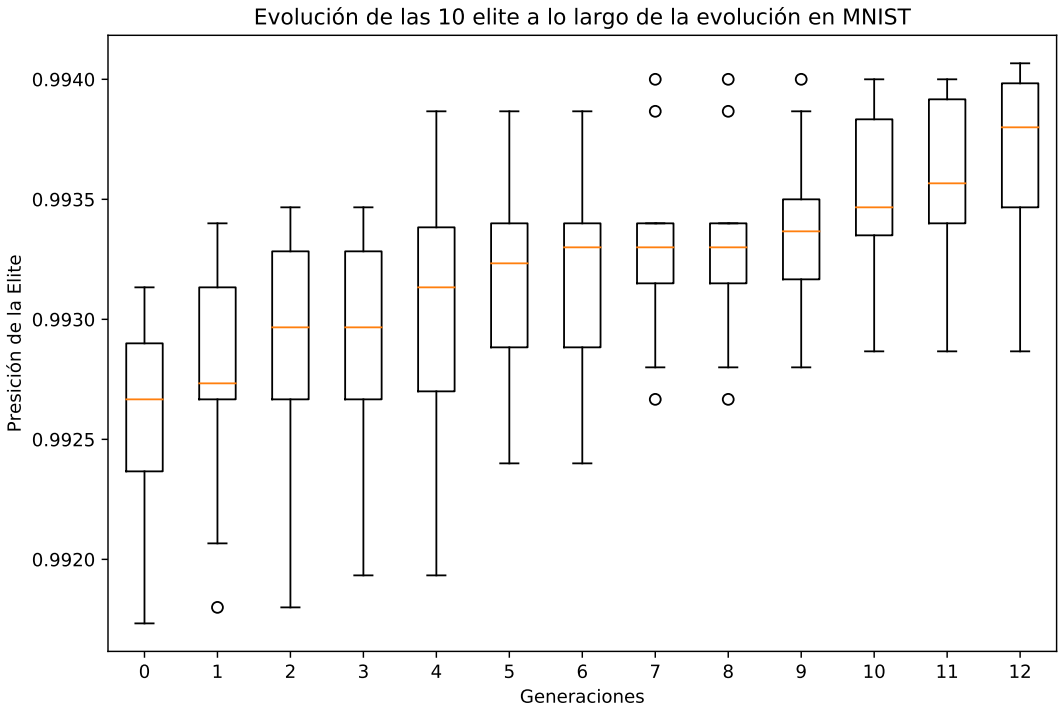


Figura 6.2: Evolución de la precisión sobre conjunto de validación de las elites a lo largo de las generaciones en MNIST. Los resultados corresponden a 10 semillas diferentes.

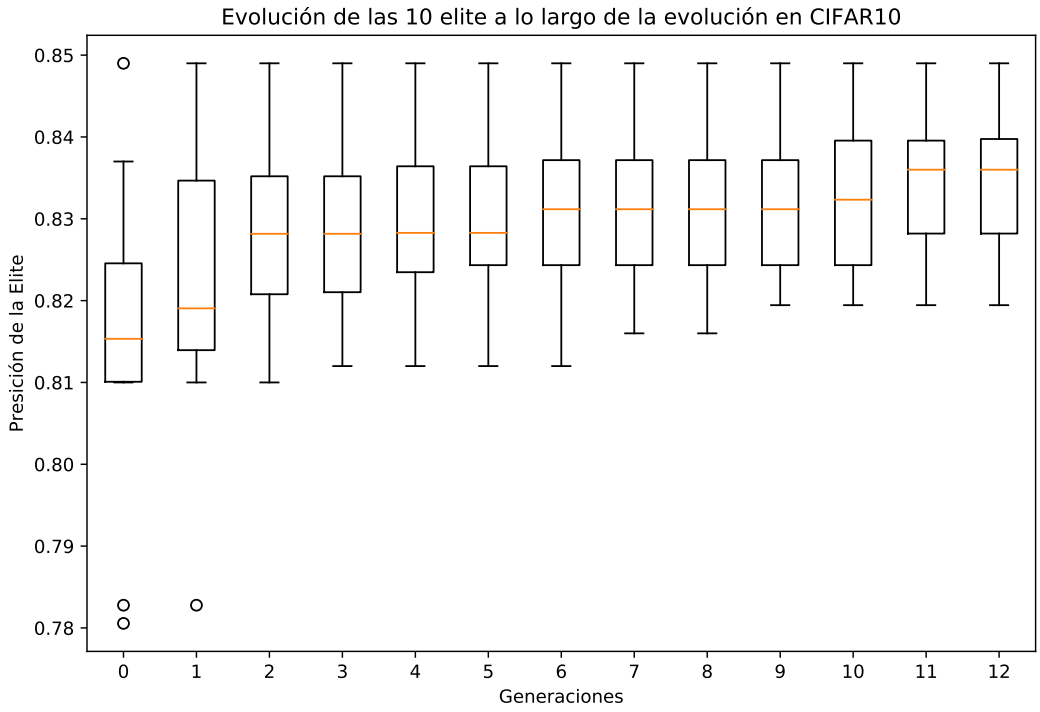


Figura 6.3: Evolución de la precisión sobre conjunto de validación de las elites a lo largo de las generaciones en CIFAR10. Los resultados corresponden a 10 semillas diferentes.

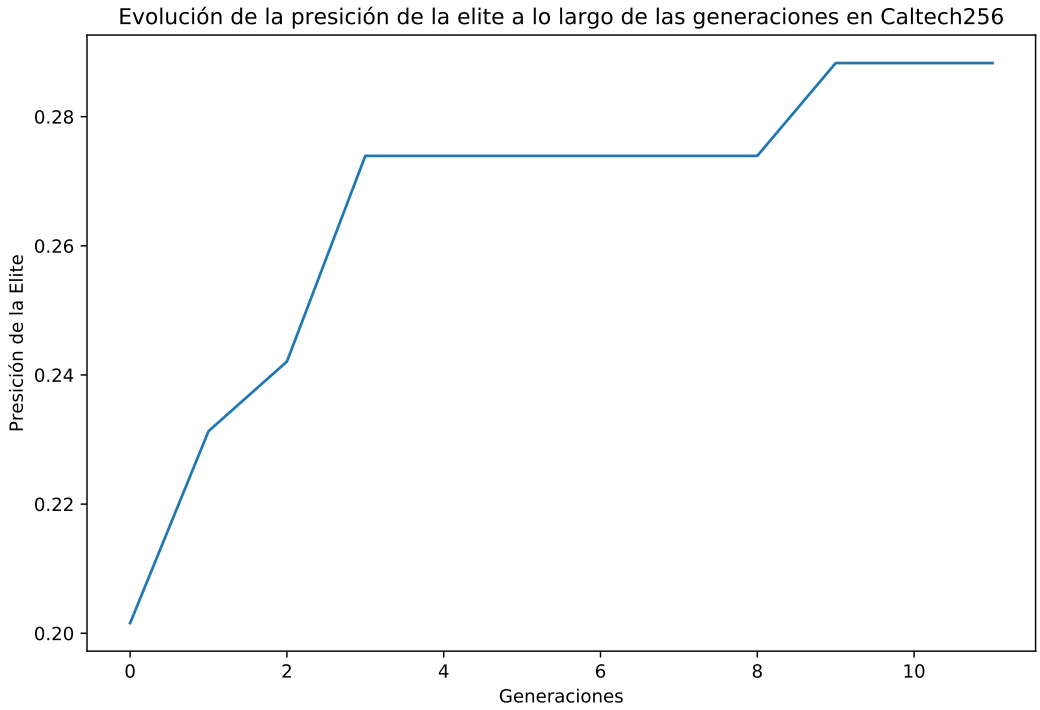


Figura 6.4: Evolución de la precisión de la elite de una sola ejecución del algoritmo sobre el conjunto de datos Caltech256. Se observa que el algoritmo converge luego de mejoras rápidas en las primeras iteraciones.

Capítulo 7

Conclusiones

La presente investigación tiene como fin validar un diseño e implementación exitoso de algoritmos genéticos para la optimización y diseño automático de la arquitectura de redes neuronales convolucionales para problemas de clasificación de imágenes, buscando facilitar el diseño de estas redes en futuras investigaciones donde los investigadores no tienen conocimiento del campo. Igualmente, buscamos delimitar de forma más precisa la naturaleza del problema de optimización y como este se relaciona con el aprendizaje automático y el proceso de búsqueda evolutiva propuesto. Para lograr estos objetivos propusimos una aproximación al operador de cruzamiento del algoritmo, implementándolo de tal forma que tome en consideración la naturaleza de las estructuras subyacentes permitiendo combinarlas de manera significativa y modificando las profundidades de las redes asociadas a los individuos. Igualmente circunscribimos los hiperparámetros relevantes al problema, permitiendo una exploración más eficiente en un espacio de búsqueda más pequeño y al mismo tiempo permitiendo que distintos individuos se comparen sobre su potencial real y no en función de la calidad del entrenamiento recibido. Además, implementamos diferentes aproximaciones para el cruzamiento y la selección ambiental a lo largo de las generaciones, fomentando mayor diversidad en las primeras generaciones e intensificando la búsqueda en las porciones finales de la evolución.

Logramos demostrar la eficacia de nuestra propuesta realizando experimentos con buenos resultados, en la mayoría de los casos sobrepasando los desempeños del o alcanzando re-

sultados similares en una cantidad significativamente menor de generaciones. Definimos igualmente de manera precisa los distintos problemas de optimización involucrados en el proceso evolutivo, presentando el problema general de aprendizaje automático de la optimización de hiperparámetros, diferenciando entre los hiperparámetros estructurales de las redes y los hiperparámetros asociados al proceso de entrenamiento. El caso de las redes residuales resulta interesante, donde obtuvimos resultados ligeramente peores que los reportados por pares, probablemente por utilizar una configuración más conservadora en términos estructurales que los autores. Esto sugiere la posibilidad de desarrollar diseños estructurales más complejos incluyendo *skip connections* y optimizaciones de sus hiperparámetros a futuro. Igualmente nos resulta interesante la posibilidad de incorporar elementos de metaheurísticas no necesariamente en el diseño de las redes, si no en la optimización de los pesos de estas en el proceso de entrenamiento.

Bibliografía

[1] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938, 2018.

[2] Teresa Araújo, Guilherme Aresta, Eduardo Castro, José Rouco, Paulo Aguiar, Catarina Eloy, António Polónia, and Aurélio Campilho. Classification of breast cancer histology images using convolutional neural networks. *PloS one*, 12(6), 2017.

[3] Zahra Beheshti and Siti Mariyam Hj Shamsuddin. A review of population-based meta-heuristic algorithms. *Int. J. Adv. Soft Comput. Appl*, 5(1):1–35, 2013.

[4] Ashray Bhandare and Devinder Kaur. Designing convolutional neural network architecture using genetic algorithms. In *Proceedings on the International Conference on Artificial Intelligence (ICAI)*, pages 150–156. The Steering Committee of The World Congress in Computer Science, Computer …, 2018.

[5] Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella, and Walter J Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8(2):239–287, 2009.

[6] Léon Bottou. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, 91(8):12, 1991.

[7] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

[8] Joseph A Cruz and David S Wishart. Applications of machine learning in cancer prediction and prognosis. *Cancer informatics*, 2:117693510600200030, 2006.

[9] Ayon Dey. Machine learning algorithms: a review. *International Journal of Computer Science and Information Technologies*, 7(3):1174–1179, 2016.

[10] Michel Gendreau and Jean-Yves Potvin. Metaheuristics in combinatorial optimization. *Annals of Operations Research*, 140(1):189–213, 2005.

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[12] Walter J Gutjahr. A graph-based ant system and its convergence. *Future generation computer systems*, 16(8):873–888, 2000.

[13] Douglas M Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[16] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[17] Guosheng Hu, Yongxin Yang, Dong Yi, Josef Kittler, William Christmas, Stan Z Li, and Timothy Hospedales. When face recognition meets with deep learning: an evaluation of convolutional neural networks for face recognition. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 142–150, 2015.

[18] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, Mar 1968.

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[20] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[21] Bernhard Korte, Jens Vygen, B Korte, and J Vygen. *Combinatorial optimization*, volume 2. Springer, 2012.

[22] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24, 2007.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[24] YD Li, ZB Hao, and Hang Lei. Survey of convolutional neural network. *Journal of Computer Applications*, 36(9):2508–2515, 2016.

[25] Wei-Yang Lin, Ya-Han Hu, and Chih-Fong Tsai. Machine learning in financial crisis prediction: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):421–436, 2011.

[26] IE Livieris and P Pintelas. A survey on algorithms for training artificial neural networks. *Department of Mathematics, University of Patras. Tech. Rep*, 2008.

[27] Sehla Loussaief and Afef Abdelkrim. Convolutional neural network hyper-parameters optimization based on genetic algorithms. *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, 9(10):252–266, 2018.

[28] Nurshazlyn Mohd Aszemi and Dhanapal Durai Dominic Panneer Selvam. Hyperparameter optimization in convolutional neural network using genetic algorithms. *International Journal of Advanced Computer Science and Applications*, 10:269 – 278, 06 2019.

[29] Sergio Nesmachnow. An overview of metaheuristics: accurate and efficient methods for optimisation. *International Journal of Metaheuristics*, 3(4):320–347, 2014.

[30] Cuong Cao Pham and Jae Wook Jeon. Robust object proposals re-ranking for object detection in autonomous driving using convolutional neural networks. *Signal Processing: Image Communication*, 53:110–122, 2017.

[31] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

[32] Anthony J Reiling. *Convolutional Neural Network Optimization Using Genetic Algorithms*. PhD thesis, University of Dayton, 2017.

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[34] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[35] Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G Yen. Automatically designing cnn architectures using genetic algorithm for image classification. *arXiv preprint arXiv:1808.03818*, 2018.

[36] Richard S Sutton. Introduction: The challenge of reinforcement learning. In *Reinforcement Learning*, pages 1–3. Springer, 1992.

[37] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.