EvoLSCompatDeep: Algoritmo Evolutivo para Diseñar Redes Neuronales Compactas

Helena Matilde-Milán[1 [0009−0007−7618−5784]] Daniel Molina[1,2 [0000−0002−4175−2204]] Francisco J. Rodríguez-Díaz[1,2 [0000−0002−8965−9148]]

[1] Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Granada

[2] Instituto Andaluz Interuniversitario en Ciencia de Datos e Inteligencia Computacional, Granada, 18016, España

{helenahmmj}@correo.ugr.es

{dmolina,fjrodriguez}@decsai.ugr.es

Resumen El diseño de arquitecturas neuronales para abordar un problema específico continúa siendo un desafío en el aprendizaje automático. Aunque existen sistemas automáticos para crear arquitecturas, como Google Cloud AutoML o AutoKeras, tienden a priorizar el rendimiento aunque sea a costa de producir modelos muy complejos computacionalmente costosos tanto en aprendizaje como en memoria. En este trabajo optimizamos una propuesta anterior, EvoDeep, que propone un algoritmo evolutivo capaz de explorar la topología neuronal y ajustar los hiperparámetros. Nuestra contribución aborda dos objetivos complementarios. Por un lado, mejorar la eficiencia de la exploración incorporando un operador de búsqueda local que optimice los parámetros, y, por el otro, priorizar en la búsqueda modelos neuronales más sencillos aunque eso suponga una ligera degradación de la precisión. Al combinar ambas estrategia, nuestra propuesta, EvoLSCompatDeep, comparada con el algoritmo original EvoDeep, consigue modelos con un rendimiento similar o mejor con menos de la mitad de parámetros. Estos resultados muestran que los algoritmos evolutivos pueden obtener redes compactas competitivas adaptadas a un problema concreto, si son adecuadamente guiadas para fomentar modelos pequeños, y que la reducción de los modelos no solo puede no perjudicar los resultados, sino mejorarlos al facilitar la exploración.

Keywords: Búsqueda de arquitecturas neuronales (NAS), Algoritmos evolutivos, Modelos neuronales compactos, Reducción de tamaño

1. Introducción

El Aprendizaje Profundo (AP)[3] es una de las áreas de investigación más atractivas del aprendizaje automático en tiempos recientes, debido a los excelentes resultados que ofrecen dichos modelos en una multitud de aplicaciones.

Estas arquitecturas neuronales se utilizan con éxito en diversos problemas, y en particular en reconocimientos de imágenes, tanto para reconocimiento de objetos [17] como clasificación en problemas complejos [6,9].

El diseño de una red adecuada para cada problema es un requisito fundamental para obtener un buen rendimiento [18]. Las Redes Neuronales Convolucionales [6] constituyen el estado del arte para la clasificación de imágenes [13]. Estas redes se componen de dos partes: la primera (*backend*) es esencialmente un extractor de características basado en operaciones de convolución y agrupación (*pooling*). La segunda parte suele contener una o más capas totalmente conectadas, cuya salida está estrechamente relacionada con las clases de clasificación.

La mayoría de los sistemas convolucionales utilizados en problemas reales emplean como *backend* arquitecturas ya establecidas como ResNet [4], ShuffleNet [16] o EfficientNet [14]. Este enfoque tiene la ventaja de disponer de modelos potentes sin tener que diseñarlos desde cero, pero son modelos con alta complejidad incluso en sus versiones más reducidas. Un mayor tamaño de red implica más recursos, especialmente memoria, aparte de requerir más ejemplos durante el entrenamiento. Reducir la complejidad manteniendo el rendimiento es un desafío importante en el AP que actualmente atrae una atención significativa de la comunidad.

Una alternativa para reducir el tamaño sería aplicar técnicas automáticas de diseño, como el uso de técnicas de búsqueda que permitan generar modelos específicos para cada problema, aplicando una Búsqueda de arquitecturas neuronales (Neural Architecture Search, NAS). En esta línea, el uso de algoritmos evolutivos es una opción prometedora. Estos son algoritmos aproximados aplicables a una amplia variedad de problemas de optimización que, aunque no garantizan encontrar la solución óptima, proporcionan soluciones satisfactorias en un tiempo razonable para problemas grandes y complejos [1,10]. Los algoritmos evolutivos han demostrado previamente ser adecuados para el desarrollo de modelos de redes neuronales complejas [12,7,2].

Existen otras herramientas recientes de automatización, como Google Cloud AutoML o AutoKeras, pero rápidamente generan modelos excesivamente complejos porque priorizan las métricas de optimización como la precisión (*accuracy*). Los modelos evolutivos, como EvoDeep [8], por contra, son más flexibles y pueden ofrecer modelos más reducidos. Sin embargo, también tienden a priorizar dichas métricas, lo que produce redes que, aunque más reducidas, siguen siendo bastante complejas, ya que tienden a incrementar rápidamente la complejidad del modelo al favorecer a aquellos con mejores resultados.

En este estudio, proponemos mejoras para EvoDeep, centrándonos en dos aspectos clave: primero, mejorar la eficiencia de la exploración mediante búsqueda local (BL) y, segundo, permitir una ligera degradación del rendimiento a cambio de reducir el tamaño del modelo. Hemos llevado a cabo experimentos con diversos conjuntos de datos, demostrando la conveniencia de nuestra propuesta. El proceso de búsqueda local nos permite mejorar los resultados acelerando la búsqueda. Además, permitir un ligero deterioro nos permite reducir claramente la complejidad de la red generada. Combinando ambos elementos, nuestra

propuesta logra resultados similares o incluso mejores más rápidamente, con la mitad o menos de parámetros.

El resto de este trabajo se estructura de la siguiente manera. En la Sección 2, describimos el algoritmo original. En la Sección 3, exponemos nuestra propuesta en detalle. Los resultados experimentales obtenidos se presentan en la Sección 4. Finalmente, en la Sección 5, resumimos las principales conclusiones obtenidas.

2. EvoDeep Original

2.1. Componentes de EvoDeep

EvoDeep [8] optimiza automáticamente las arquitecturas y los hiperparámetros de redes neuronales profundas, maximizando la precisión en tareas de clasificación de imágenes. Para ello, emplea una estrategia evolutiva para la creación y optimización de la red neuronal, en donde cada individuo representa una arquitectura y un conjunto de hiperparámetros específicos. Para optimizar los modelos, utiliza como función *fitness* el *accuracy* obtenido en la clasificación.

Los tipos de capas considerados por EvoDeep son: capas densas, convolucionales, *dropout*, *max pooling*, *reshape* y *flatten*. Cada capa almacena su categoría, el tipo de entrada y salida, y sus parámetros, generados inicialmente de forma aleatoria dentro de un rango. La última capa es siempre una capa densa. Cada individuo está compuesto por un grupo de capas (sus tipos y posiciones), su tamaño y diversos parámetros relacionados con el aprendizaje (optimizador, número de épocas y tamaño del lote o *batch size*). Internamente utiliza una máquina de estados finitos (MEF) para garantizar la construcción unicamente de secuencias de capas válidas.

Los operadores de cruce y mutación actúan a distintos niveles: uno interno, que modifica únicamente los hiperparámetros de cada capa, y otro externo, que afecta a los parámetros globales y a capas completas. A los parámetros se les aplica el cruce uniforme. Para la estructura se aplica una técnica de *cut and splice* (corte y empalme) a ambas soluciones, seleccionando aleatoriamente un punto de corte en cada una, y que crea las nuevas soluciones crean combinando la parte izquierda de un individuo con la parte derecha del otro. Se muestra un ejemplo en la Figura 1.

Para la mutación, cada parámetro posee una probabilidad mut_{pb} de mutar su valor aleatoriamente (dentro del rango). Adicionalmente, con probabilidad new_{pb}, se crean nuevas capas (entre 3 y 5) en posiciones aleatorias, utilizando la MEF para verificar su validez. Finalmente, se selecciona una de estas configuraciones aleatoriamente, si no se ha alcanzado la longitud máxima del individuo.

En EvoDeep, el *fitness* de cada individuo es el acierto (*accuracy*) de la red en la clasificación. Por tanto, evaluar cada solución implica generar y entrenar la red, lo cual representa un proceso computacionalmente muy costoso. El proceso de entrenamiento se realiza usando los parámetros específicos de cada individuo (optimizador, épocas, tamaño del lote) y, posteriormente, se evalúa el fitness sobre el conjunto de validación.

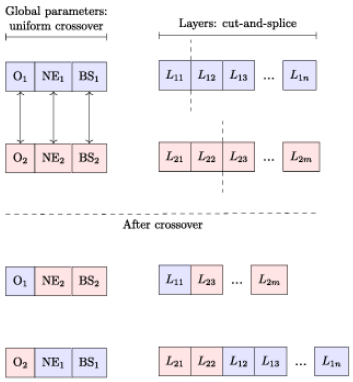


Figura 1: Ejemplo de cruce externo utilizando dos puntos de corte. O_i representa el optimizador, BS_i el tamaño del lote y L_i una capa de la arquitectura.

2.2. Esquema global

La estrategia evolutiva implementada es $(\mu + \lambda)$, donde $\lambda/2$ es el número de veces que se aplica el operador de cruce en cada generación (generando dos nuevos individuos por operación) y μ es el número de los mejores individuos seleccionados para conformar la siguiente población. Inicialmente, se genera y evalúa una población de individuos de forma aleatoria de tamaño $2 \cdot \mu$, del que se escogen los mejores μ individuos (se genera el doble de soluciones de las que perduran para que la población inicial presente un mejor comportamiento).

El algoritmo se ejecuta durante un número determinado de generaciones. En cada generación, se producen λ soluciones de forma iterativa. En cada iteración, se seleccionan aleatoriamente dos individuos y se aplica el operador de cruce con una probabilidad cx_{pb}, generando dos nuevos individuos. Con una probabilidad mut_{pb}, se aplica el operador de mutación de forma independiente a cada descendiente. Finalmente, los nuevos individuos resultantes son evaluados. Este procedimiento se repite hasta generar λ individuos. Después, solo se almacenan los μ mejores individuos (provenientes de la población anterior y de los λ nuevos). El criterio de parada es el número de generaciones, aunque el algoritmo también finaliza cuando se alcanza un número máximo de generaciones sin cambios en la población (cuando los λ nuevos individuos son inferiores a los existentes).

3. Nuestra propuesta: EvoLSCompatDeep

En esta sección, describimos nuestra propuesta, EvoLSCompatDeep, destacando las principales novedades con respecto a EvoDeep. Dichos elementos son:

- Método de búsqueda local sobre los hiperparámetros.
- Nueva función de selección para reducir la población $\mu+\lambda$ a una población μ en la siguiente generación. Originalmente, solo se consideraba el *fitness*. En nuestra propuesta, también tendremos en cuenta la complejidad de la red.

Algorithm 1 EvoDeep

```
Init parameters
Init population P with size initial_population
Evaluate each individual in P
gen_sin_cambios ← 0
for gen = 1, 2, ..., ngen do
    Offspring O ← ∅
    for i = 1, 2, ..., λ/2 do
        Choice randomly two individuals p1, p2 from P
        Apply crossover (with probability) p1 y p2 and get o1, o2
        Apply mutation (with probability) to o1 y o2
        Evaluate o1, o2
        Add o1, o2 to O
    end for
    P ← Select the best μ individual from P ∪ O
    if Not improved the average P fitness then
        gen_no_changes ← gen_no_changes + 1
    else
        gen_no_changes ← 0
    end if
    if gen_no_changes ≤ max_without_change then
        Early end the algorithm
    end if
end for
```

A continuación, describimos detalladamente cada uno de estos componentes.

3.1. Búsqueda Local

EvoDeep solo actualiza los hiperparámetros de los modelos mediante el cruce y la mutación. Sin embargo, dicho proceso es lento. Los hiperparámetros pueden impactar significativamente el *fitness*, ya que si son inadecuados, el algoritmo podría alterar sustancialmente la arquitectura para mejorar el modelo, cuando un simple ajuste de hiperparámetros podría lograr una mayor mejora más fácilmente. Incorporar un método de ajuste específico para los hiperparámetros puede acelerar la mejora, aprovechando mejor el potencial de cada arquitectura.

En nuestra propuesta, aplicamos una modificación del conocido algoritmo de Solis-Wets [11] a los parámetros numéricos de cada capa. Dado que en las primeras generaciones la arquitectura suele requerir más cambios, el método de BL se aplica únicamente a la mejor solución al alcanzar el 50 % y el 75 % de las generaciones, así como al finalizar el algoritmo. Para actualizar cada parámetro, se define un tamaño de salto, *step size*, y un valor mínimo y máximo para cada parámetro.

Debido a que la evaluación del *fitness* es muy lenta, el número de iteraciones está muy limitado (solo 5 por ejecución), por lo que, a diferencia del algoritmo Solis-Wets original, el tamaño de paso no varía, siendo su valor siempre aleatorio

entre $\{-step_size, +step_size\}$. En los demás aspectos, el método de BL es idéntico al de Solis-Wets, por lo cual omitimos una descripción más detallada.

3.2. Criterio de selección

La otra mejora consiste en modificar la función de selección utilizada por EvoDeep para reducir la población de $\mu + \lambda$ individuos a μ para la siguiente generación. Mientras que EvoDeep seleccionaba siempre a los individuos con mejor *fitness*, nuestra propuesta considera también la complejidad de cada modelo, para fomentar modelos competitivos más simples. La población combinada se ordena mediante un nuevo criterio de selección y se eligen los primeros μ individuos. Para cada par de soluciones, se considera la diferencia en su *fitness*. Si la diferencia es menor que un valor umbral, se selecciona la red menos compleja (medida por el número de parámetros). Si la diferencia es superior al umbral, se elige la solución con mejor *fitness*. La lógica de este criterio es razonable: cuando un individuo es claramente superior a otro, se selecciona el mejor. Esto evita que se seleccionen individuos con resultados significativamente peores. Los modelos más simples se prefieren únicamente cuando su rendimiento, aunque inferior, no empeora significativamente.

Tabla 1: Valores de parámetros para EvoDeep y EvoLSCompatDeep

Parámetro	Valor
Paciencia (*early stopping*)	100
Población inicial	10
μ	5
λ	20
Número de generaciones	10
Probabilidad de cruce	0.3
mut_{pb}	0.3
new_{pb}	0.3
Máx. generaciones sin cambio	5
Ejecuciones	5
BL_{itera} (EvoLSCompatDeep)	5

4. Sección Experimental

En esta sección vamos a demostrar experimentalmente la conveniencia de nuestra propuesta. Primero, vamos a definir el marco experimental. Luego, mostraremos la mejora de incorporar tanto el mecanismo de BL como el nuevo criterio de selección. Finalmente, compararemos los resultados finales obtenidos con la propuesta, EvoLSCompatDeep, frente al algoritmo original, tanto en precisión como en tamaño.

4.1. Marco experimental

Los conjuntos de datos empleados para evaluar la propuesta son Fashion-MNIST [15] y CIFAR-10 [5]. Fashion-MNIST es un problema conocido de clasificación de ropa, compuesto por 70,000 imágenes en escala de grises de 28×28 píxeles, distribuidas uniformemente en diez clases. CIFAR-10 es un conjunto de datos ampliamente usado con diferentes categorías, entre ellas animales y vehículos. Contiene 60,000 imágenes a color de 32×32 píxeles, igualmente distribuidas en diez clases. Aunque originalmente CIFAR-10 está en color, EvoDeep fue evaluado únicamente con imágenes en escala de grises, en nuestros experimentos usaremos además de la versión original en color (CIFAR-10C) una versión en escala de grises denominada CIFAR-10G.

Cada conjunto de datos define un conjunto de entrenamiento y otro de prueba. Para los experimentos, el conjunto de entrenamiento original se dividió en dos partes: un conjunto de entrenamiento (con el 80 % de los datos) y un conjunto de validación (el 20 % restante). Cada algoritmo evolutivo se ejecuta utilizando el conjunto de validación para calcular el *fitness*. Por ello, la comparación principal entre algoritmos evolutivos se centra en los resultados sobre la validación. Además, la mejor solución obtenida (según la validación) se evaluó sobre el conjunto de prueba para comprobar que la mejora en validación no se deba a un sobre entrenamiento.

La Tabla 1 muestra los parámetros comunes empleados por EvoDeep y por EvoLSCompatDeep. Los valores de EvoDeep siguen las recomendaciones de sus autores.

Tabla 2: Configuración de parámetros utilizada durante la de búsqueda local

Tipo de capa	Parámetro	Mínimo	Máximo	Paso
Dense	#neurons	50	350	50
Dropout	prob.	0.2	0.6	0.2
Convolutional	kernel size	3×3	7×7	2×2
Convolutional	filter size	5	50	10
Max Pooling	stride	1	6	1
Max Pooling	#Pool	2x2	6×6	1×1

La Tabla 2 muestra los parámetros del proceso de búsqueda local y, para cada uno, su valor mínimo, máximo, y tamaño paso.

Para ejecutar los experimentos se utilizó Python 3.11, TensorFlow v2.19 y scikit-learn v1.61. Todas las ejecuciones se realizaron en el mismo servidor GPU, equipado con 4×RTX2080Ti, 2×Intel Xeon E5-2630 y 128GB de memoria DDR3.

4.2. Resultados experimentales con BL

En primer lugar, se comparan los resultados obtenidos al introducir el componente BL. La Tabla 3 muestra la comparación entre EvoDeep y nuestra pro-

puesta incorporando BL. Para cada conjunto de datos se ejecutó cada algoritmo cinco veces y se compararon las métricas promedio. Las métricas presentadas en las tablas son: precisión de entrenamiento (acc_train), precisión de validación (acc_val), precisión de prueba (acc_test), el número de parámetros del modelo (en millones) y el tiempo de optimización (en minutos).

Tabla 3: EvoDeep vs (EvoDeep + BL), valores promedio

	FashionMNIST		CIFAR10G		CIFAR10C	
Métrica	EvoDeep	+BL	EvoDeep	+BL	EvoDeep	+BL
acc_val	0.901	0.893	0.470	0.469	0.498	0.488
acc_test	0.898	0.889	0.467	0.465	0.489	0.479
#Parám. (10^6)	**2.006**	2.635	4.870	**1.605**	2.02	**1.17**
Tiempo (min)	105.8	**63.3**	84.6	**65.6**	54.8	**52.2**

Los resultados en la Tabla 3 varían entre los conjuntos de datos. Aunque la precisión de validación con la BL es ligeramente inferior, la diferencia es mínima (1 %). En cuanto a la precisión de prueba, se puede observar que los resultados son muy similares (<1 %), por lo que el método de BL no parece hacer sobreajuste.

El número de parámetros aumenta un 31 % en FashionMNIST y disminuye un 12 % y un 22 % en CIFAR10G y CIFAR10C, respectivamente. Sin embargo, el tiempo de búsqueda se reduce significativamente (40 % en FashionMNIST y 22,5 % en CIFAR10G). Dado que se espera una reducción en el tamaño con el nuevo componente, y el tiempo de procesamiento no es crucial, se considera beneficiosa la incorporación de la BL.

4.3. Resultados experimentales con nueva selección

Dado que se ha observado que el valor umbral puede afectar considerablemente los resultados en FashionMNIST, se ha realizado una ajuste manual de este parámetro en la Tabla 4. Se puede apreciar que, a medida que aumenta el umbral, la complejidad disminuye, hasta que empieza a deteriorarse con valores de umbral superiores al 1 %. Curiosamente, el ratio de aceptación permanece relativamente estable. Basándose en estos resultados, se considera el valor 0,75 % como el mejor valor umbral, al mantener un acierto muy alto, y con menos de la mitad de los parámetros que cuando no se utiliza.

Para CIFAR10C y CIFAR10G, se aplicó con un umbral del 0 % y 3 %. La Tabla 5 muestra los resultados en el mismo formato que la Tabla 4. Se puede observar que no solo reduce la complejidad del modelo al 21 % del tamaño original, sino que también mejora la precisión.

En resumen, la nueva función de selección no solo permite al algoritmo reducir el tamaño del modelo, sino que también mejora la búsqueda.

Tabla 4: Ajuste del valor de umbral para FashionMNIST

Umbral (%)	acc_val	acc_test	#Parám. (10^6)	# Por_acc. (%)	#Por_complej. (%)
0.00	0.893	0.889	2.63	100	0
0.50	0.894	0.886	1.38	63	37
0.75	0.893	0.887	1.10	56	44
1.00	0.893	0.886	1.09	55	45
1.25	0.886	0.883	0.16	51	49
1.50	0.883	0.880	0.62	49	51

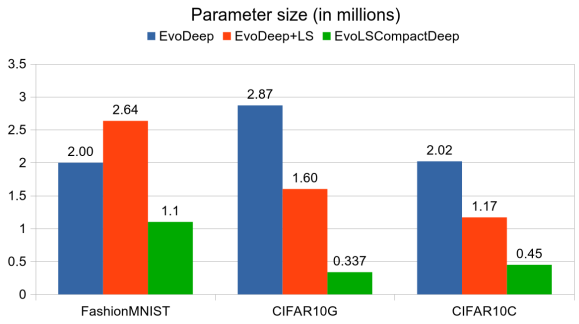
Tabla 5: Resultados basados en el valor de umbral para CIFAR10G

Umbral (%)	acc_val	acc_test	#Parám. (10^6)	# Por_acc. (%)	#Por_complej. (%)
0.0	0.469	0.465	1.61	100	0
3.0	0.530	0.524	0.34	51	49

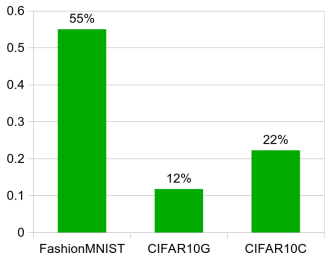
4.4. Resultados experimentales de EvoLSCompatDeep

En esta sección se resumen las diferencias entre EvoDeep y nuestra propuesta, EvoLSCompatDeep, para cada problema de clasificación.

En la Tabla 6 se puede observar que la precisión en el conjunto de prueba empeora solo un 1 % en FashionMNIST y CIFAR10C, mientras que en CIFAR10G mejora un 5 %. Pero lo más significativo es el tamaño, que observamos en la Figura 2.



(a) Tamaños del modelo para cada algoritmo



(b) Reducción conseguida por EvoLSCompatDeep

Figura 2: Diferencias del tamaño del modelo según el algoritmo

Tabla 6: EvoDeep vs EvoLSCompatDeep, para cada conjunto de datos, valores promedio

Conjunto de dato	Métrica	EvoDeep	+BL	EvoLSCompatDeep
FashionMNIST	acc_val	**0.901**	0.893	0.893
	acc_test	**0.898**	0.889	0.887
	#Parám. (10^6)	2.006	2.635	**1.100**
	Tiempo (min)	106	63	**63**
CIFAR10G	acc_val	0.470	0.469	**0.530**
	acc_test	0.467	0.465	**0.524**
	#Parám. (10^6)	2.870	1.605	**0.337**
	Tiempo (min)	84.6	**65.6**	88.73
CIFAR10C	acc_val	**0.498**	0.488	0.485
	acc_test	**0.489**	0.479	0.480
	#Parám. (10^6)	2.02	1.17	**0.45**
	Tiempo (min)	54.8	**52.2**	53

En la Figura 2, se puede ver que los tamaños obtenidos por EvoLSCompatDeep en FashionMNIST, CIFAR10G y CIFAR10C son reducidos por cada modificación del algoritmo original (Figura 2a), por lo que se puede calcular que EvoLSCompatDeep permite obtener modelos con un 55 %, un 12 % y un 22 %, respectivamente, del tamaño original obtenido por EvoDeep (Figura 2b). Por tanto, EvoLSCompatDeep obtiene resultados similares o mejores que EvoDeep con una gran reducción del tamaño de los modelos obtenidos.

5. Conclusiones

Para problemas reales, es frecuente emplear arquitecturas de redes neuronales profundas consolidadas, las cuales garantizan un comportamiento adecuado. Sin embargo, al haber sido diseñadas para problemas complejos, pueden introducir una dificultad excesiva para la tarea específica, lo que implica un gran número de parámetros y consumo de memoria. Aunque resultan deseables los modelos automáticos que puedan adaptar su estructura, al guiarse exclusivamente por el rendimiento, acaban generado también modelos bastante complejos.

En este trabajo proponemos un nuevo algoritmo, EvoLSCompatDeep, que mejora EvoDeep para el diseño de redes neuronales profundas. Nuestra propuesta se caracteriza por dos componentes clave: un método de búsqueda local que acelera el proceso de exploración y un criterio de selección de soluciones que acepta un empeoramiento limitado cuando ello permite reducir la complejidad.

Evaluamos nuestra propuesta en los conjuntos de datos Fashion-MNIST y CIFAR-10 (en versión de color y blanco y negro). Los resultados obtenidos demuestran que nuestro método logra una reducción significativa del modelo sin comprometer los resultados, dado que la precisión obtenida supone una variación inferior al 1 % en el peor de los casos. Por el contrario, reduce el tamaño

obtenido por EvoDeep un 55 %, 12 % y 22 % en FashionMNIST, CIFAR10G y CIFAR10C, respectivamente.

Como trabajo futuro estamos trabajando en mejorar el espacio de búsqueda a partir de un análisis forense de las soluciones exploradas, optimizando el proceso de búsqueda, de forma que se pueda seguir reduciendo el tamaño de los modelos obtenidos al mismo tiempo que mejoren los resultados.

En resumen, hemos presentado EvoLSCompatDeep, una propuesta evolutiva que logra una reducción sustancial de la complejidad del modelo sin comprometer el rendimiento, representando un avance significativo en el diseño automatizado de redes neuronales profundas para problemas específicos.

Agradecimientos. Esta investigación es parte del Proyecto TSI-100927-2023-1, financiado por Plan de Recuperación, Transformación y Resiliencia Europeo, de la Unión Europea-Next Generation EU a través del Ministerio para la Transformación Digital y de la Función Pública y está parcialmente financiada por el Ministerio de Ciencia y Tecnología de España en el marco del proyecto PID2023-149128NB-I00, financiado por MICIU/AEI/10.13039/501100011033 y por FEDER, UE.

Referencias

1. Back, T., Fogel, D.B., Michalewicz, Z.: Handbook of Evolutionary Computation. IOP Publishing Ltd., 1st edition edn. (1997)
2. Ghosh, A., Jana, N., Mallik, S., Zhao, Z.: Designing optimal convolutional neural network architecture using differential evolution algorithm. Patterns **3**, 100567 (2022)
3. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. Mit Press (2016)
4. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778 (2016)
5. Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep., University of Toronto (2009)
6. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
7. Martinez, A., Del Ser, J., Villar-Rodriguez, E., Osaba, E., Poyatos, J., Tabik, S., Molina, D., Herrera, F.: Lights and shadows in evolutionary deep learning: Taxonomy, critical methodological analysis, cases of study, learned lessons, recommendations and challenges. Information Fusion **67**, 161–194 (2021)
8. Martín, A., Lara-Cabrera, R., Fuentes-Hurtado, F., Naranjo, V., Camacho, D.: Evodeep: a new evolutionary approach for automatic deep neural networks parametrisation. Journal of Parallel and Distributed Computing **117** (2017)
9. Muhammad, K., Khan, S., Ser, J.D., Albuquerque, V.H.C.d.: Deep learning for multigrade brain tumor classification in smart healthcare systems: A prospective survey. IEEE Transactions on Neural Networks and Learning Systems **32**, 507–522 (2021)
10. Siarry, P. (ed.): Metaheuristics. Springer International Publishing (2016)

11. Solis, F.J., Wets, R.J.B.: Minimization by random search techniques. Mathematics of Operations Research **6**, 19–30 (1981)
12. Stanley, K., D'Ambrosio, D., Gauci, J.: A hypercube-based encoding for evolving large-scale neural networks. Artificial life **15**, 185–212 (2009)
13. Sultana, F., Sufian, A., Dutta, P.: Advancements in image classification using convolutional neural network. In: 2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN), Kolkata, India, 2018 (2018)
14. Tan, M., Le, Q.: EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In: Proceedings of the 36th International Conference on Machine Learning. pp. 6105–6114. PMLR (2019)
15. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms (2017)
16. Zhang, X., Zhou, X., Lin, M., Sun, J.: ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 6848–6856 (2018)
17. Zhou, X., Gong, W., Fu, W., Du, F.: Application of deep learning in object detection. In: IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS), Wuhan, China (2017)
18. Ünal, H.T., Başçiftçi, F.: Evolutionary design of neural network architectures: A review of three decades of research. Artificial Intelligence Review **55**, 1723–1802 (2022)