Comparación de métodos Zero-shot Proxy en la Búsqueda de Arquitecturas de Redes Neuronales basada en Algoritmos Genéticos y aplicados en la Clasificación de Imágenes

Eduardo Centurión Funes, José D. Colbes Sanabria y Diego P. Pinto-Roa

Facultad Politécnica, Universidad Nacional de Asunción
San Lorenzo, Paraguay
edu.centurion.funes@fpuna.edu.py, {jcolbes,dpinto}@pol.una.py

Resumen. La Búsqueda de Arquitecturas de Redes neuronales (NAS, *Neural Architecture Search*) es una línea de investigación fundamental para aplicaciones de Redes Neuronales Convolucionales (CNN, *Convolutional Neural Network*) de alto rendimiento. Normalmente, los métodos NAS propuestos en la literatura requieren un alto costo computacional para calcular la solución CNN óptima dentro de los problemas de clasificación de imágenes. Por lo tanto, una línea de investigación emergente propone estimadores de rendimiento llamados métodos *Zero-Shot Proxy*. Las métodos *Zero-Shot Proxy* reducen el costo de evaluación dentro de la búsqueda de la arquitectura óptima, reemplazando parcial o completamente el entrenamiento y evaluación. Claramente, estos métodos ayudan a reducir el costo computacional, sin embargo, es necesario examinar su impacto en NAS. Este trabajo estudia el efecto de métodos Zero-Shot Proxy del estado del arte –*Gradient norm*, *SNIP*, *Synflow*, *GraSP*, *Fisher information*, y *Jacobian covariant*– como guías estratégicas en un Algoritmo Genético. La simulación experimental en la base de datos *CIFAR-10* indica el buen rendimiento de los métodos considerados, siendo el *Jacobian covariant* el más preciso.

Palabras claves: Búsqueda de Arquitecturas de Redes Neuronales · Zero-shot Proxy · Algoritmos Genéticos

1 Introducción

Las Redes Neuronales Convolutivas (CNN, *Convolutional Neural Networks*) [16] son extensamente aplicadas a tareas de visión artificial como clasificación de imágenes, segmentación de imágenes entre otros desafíos. Las CNN tienen una gran capacidad para la extracción de características que, sumado a las redes neuronales profundas, son fácilmente aplicadas y transferidas a diferentes problemas. Las CNN han ido evolucionando a lo largo del tiempo donde varios componentes han sido diseñados de forma cuidadosa para lograr un gran desempeño. Estas arquitecturas son buenas en promedio, pero para lograr un desempeño más preciso en problemas específicos es necesario adoptar estrategias más avanzadas que

el diseño manual. En este contexto surge el diseño automático de arquitecturas dando surgimiento a la Búsqueda de Arquitecturas de Redes Neuronales (NAS, *Neural Architecture Search* [5]).

Los métodos NAS se utilizan ampliamente para la obtención automática de una Red Neuronal con el mejor rendimiento posible [26]. Típicamente, NAS tiene asociado tres componentes: el Espacio de Búsqueda, el Estrategia de Búsqueda, y la Estrategia de Estimación del Rendimiento de las arquitecturas. Considerando algunas condiciones estructurales, se puede definir un Espacio de Búsqueda limitado donde cada solución es una arquitectura candidata. La cantidad y calidad de las arquitecturas candidatas pueden verse limitada en base a los altos costes computaciones en que se incurre al evaluar el rendimiento de cada candidato, principalmente en la etapa de entrenamiento.

Para abordar el problema del alto costo computacional en NAS han surgido métodos que buscan reducir este coste, afrontado al problema a través del entrenamiento supervisado [26]. En esta línea se destacan los enfoques basados en *regresión* [3], *one-shot* [17], y *zero-shot* [8]. Los métodos *zero-shot* se peuden utilizar como Estrategias de Predicción de Rendimiento de arquitecturas neuronales que se caracterizan por tener un menor costo computacional. El principal foco de estudio del presente trabajo será realizar un estudio comparativo de sus bondades y limitaciones estos métodos.

Basado en lo anterior, este trabajo propone estudiar experimentalmente el desempeño de las distintas técnicas de estimación *Zero-shot* como métricas que guían el proceso evolutivo de un Algoritmo Genético (GA, *Genetic Algorithm*) [12] para la búsqueda de arquitecturas CNN de clasificación de imágenes sobre la base de datos CIFAR-10 [6]. En este estudio se aborda simulaciones en los espacios de soluciones NAS previamente calculados en NAS-Bench-101 [24] y NAS-Bench-201 [4].

En este contexto, este trabajo queda organizado de la siguiente forma: la Sección 2 presenta los principales trabajos del estado del arte en el problema NAS, la Sección 3 presenta el enfoque de Evolución Guiada como Estrategia de Búsqueda, la Sección 4.2 los enfoques de predicción de arquitecturas basadas en Zero-shot proxy, la Sección 5 presenta y discute los resultados de las simulaciones numéricas, mientras que la Sección 6 expone las conclusiones principales y trabajos futuros.

2 Trabajos Relacionados

En base al interés en la automatización de NAS, se han desarrollado varios trabajos que proponen y/o evalúan varios métodos. Estos aportes se pueden categorizar según el Espacio de Búsqueda, la Estrategia de Búsqueda y la Estrategia de Estimación del rendimiento [5].

El Espacio de Búsqueda define el conjunto de arquitecturas posibles que la Búsqueda tendrá en cuenta, la incorporación de conocimientos previos sobre las propiedades típicas de las arquitecturas adecuadas para una tarea puede simplificar y reducir el tamaño del espacio de búsqueda. Trabajos previos, como

NAS-Bench-101 [24] y NAS-Bench-201 [4], publican datos sobre el rendimiento de las arquitecturas dentro de los Espacios de Búsqueda que definen.

Diversas Estrategias de Búsqueda pueden ser utilizadas a la hora de explorar el Espacio de Búsqueda. Los métodos utilizados en la literatura para NAS se basan en Búsqueda Aleatoria [2], Aprendizaje Reforzado [26], Optimización Bayesiana [19], Algoritmos Evolutivos [18], métodos basados en gradiente [9], entre otros.

Diversas Estrategias de Estimación del Rendimiento se han propuesto para abordar el problema del alto costo computacional en NAS, estas propuestas buscan que buscan reducir o reemplazar el entrenamiento y evaluación. Por ejemplo, se pueden encontrar las estimaciones de *baja fidelidad*, que se basan en reducción del esfuerzo de entrenamiento, optando por tiempos de entrenamiento reducidos al entrenar durante menos épocas [27], selección de un subconjunto de datos, reducción modelos y/o reducción de la calidad de los datos. En esta línea se destacan los enfoques basados en *regresión* [3], *one-shot* [17], y *zero-shot* [8].

Los métodos basados en regresión toman del espacio de búsqueda un subconjunto de arquitecturas candidatas (muestra) y evalúan sus rendimientos. Posteriormente, a través de técnicas de regresión obtienen una función de predicción de desempeño que es utilizada en el proceso de la búsqueda sobre el entero espacio de soluciones [4, 24]. Estas técnicas mejoran cuando el número de arquitecturas de muestra aumentan lo que pueden incurrir en altos costes computacionales.

Los métodos *one-shot* se basan en técnicas de *parameter-sharing*, tales como se propone en ENAS [17] u otros métodos que continúen esa línea. La idea central de estos métodos consiste en visualizar a cada arquitectura candidata como un sub-grafo dentro un grafo principal. La red neuronal que represente este grafo principal es entrenada y los parámetros obtenidos se comparten con las arquitecturas candidatas, manteniendo la correlación entre el sub-grafo de la arquitectura candidata y el grafo principal. Si bien, este método constituye un ahorro de tiempo de computación al requerir un solo entrenamiento, este único entrenamiento puede resultar altamente costoso dado debido a complejidad de la red neuronal resultante.

Los métodos *zero-shot* se basan en el uso de agentes (*proxy*) que pueden clasificar la precisión de las arquitecturas candidatas en la etapa de inicialización, es decir, sin entrenamiento, a modo de reemplazar el costoso proceso de entrenamiento en NAS con algunas alternativas computacionalmente eficientes [13, 17, 26]. Un *zero-shot* ideal debería proporcionar información sobre la capacidad de la arquitectura candidata para aprender representaciones complejas, generalizar las muestras invisibles, y converger a valores de pérdida mínima.

Para guiar el proceso de búsqueda, sin recurrir al costoso entrenamiento de cada arquitectura, se han desarrollado varias Estrategias de Estimación del Rendimiento *zero-shot* [5] en la que se destacan: Gradient norm [1], SNIP [7], Synflow [21], GraSP [23], Jacobian covariant [11]. Estos enfoque serán descriptos en secciones posteriores.

3 Evolución Guiada

La meta de un algoritmo NAS es encontrar una arquitectura óptima a^* dentro de un Espacio de Búsqueda $A, a^* \in A$, tal que maximice una función objetivo O.

El método de Evolución Guiada, propuesto en [12], enmarca NAS como un problema de optimización, donde una estrategia evolutiva explora arquitecturas $a \in A$ basada en operaciones de evolutivas, donde la evolución puede ser guiadas por evaluaciones exactas o aproximadas. Por ende, podemos definir al problema NAS como un problema optimización anidada, donde el objetivo es encontrar una red neuronal, L, creada desde el entrenamiento de la arquitectura óptima a^*, en un conjunto de entrenamiento, $d^{(train)}$, tal que a^* maximiza una función objetivo O para una tarea determinada [12]:

$$a^* = \arg\max_{a \in A} O(L(a, d^{(train)}))$$

El Algoritmo de Evolución Guiada propuesto en [12], resumido en el Algoritmo 1, propone una Estrategia de Búsqueda basada en un algoritmo genético y una estrategia de estimación del rendimiento basado en estimadores. Para uso del presente estudio, abordamos el problema con la misma Estrategia de Búsqueda, variando la Estrategia de Estimación del Rendimiento por diversas alternativas, siendo estas alternativas los agentes *zero-shot*.

Algoritmo 1: Evolución Guiada

Entrada: Parámetros evolutivos, Base de Datos, Zero_shot proxy
Salida: Mejor Modelo

historial ← ∅
poblacion ← ∅
mientras *tamanho_poblacion* < C **hacer**
    modelo.arq ← *arquitectura_aleatoria*()
    modelo.acc ← *estimar*(*modelo.arq*, *zero_shot*)
    agregar *modelo* a la derecha de *poblacion*
descartar los $C - P$ peores individuos de la *poblacion*
mientras *modelo* ∈ *poblacion* **hacer**
    modelo.acc ← *entrenar_y_evaluar*(*modelo.arq*)
    agregar *modelo* al *historial*
mientras *tamanho_historial* < C **hacer**
    muestra ← seleccionar S candidatos aleatorios de la población
    padre ← seleccionar el modelo de mayor precisión en *muestra*
    generacion ← ∅
    mientras *tamanho_generacion* < P **hacer**
        hijo.arq ← *mutacion*(*padre.arq*)
        hijo.acc ← *estimar*(*hijo.arq*, *zero_shot*)
        agregar *hijo* a *generacion*
    mejor_hijo ← modelo de mayor rendimiento en *generacion*
    mejor_hijo.acc ← *entrenar_y_evaluar*(*mejor_hijo.arq*)
    agregar *mejor_hijo* a la derecha de *poblacion*
    agregar *mejor_hijo* al *historial*
    remover *descarte* de la izquierda de *poblacion*
devolver *modelo de mayor precisión del conjunto –historial–*

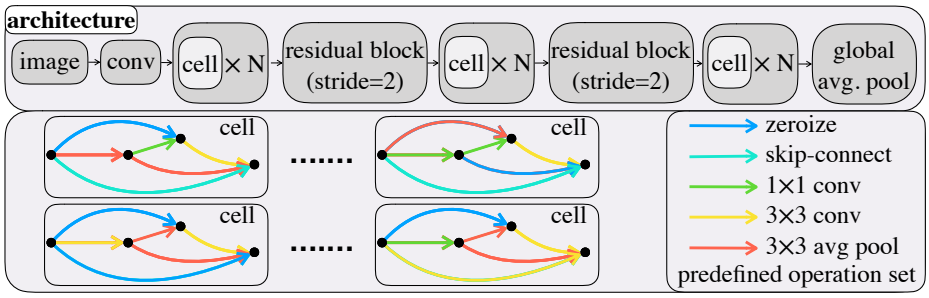


Fig. 1: **Arriba**: el macro esqueleto de cada arquitectura candidata. **Abajo-izquierda**: ejemplos de una celda con 4 nodos. Cada celda es un grafo acíclico dirigido, donde cada arista es asociada con una operación seleccionada a un conjunto de operaciones predefinidas, como se detalla **Abajo-derecha**. Fuente: [4, Figure 1].

Dado que, una CNN se compone principalmente de una o más capas de convolución para extracción de características de las imágenes y una capa de clasificación [16]. La línea de investigación estudiada en este trabajo propone la búsqueda de la arquitectura variando la estructura de las capas de convolución, quedando fijo la capa de clasificación [24]. En la Figura 1 se puede observar una arquitectura CNN. En la misma se puede apreciar los patrones de capas de convolución repetibles, denominados *celdas*, donde las operaciones básicas se organizan como un grafo acíclico de operaciones.

El Algoritmo de búsqueda 1 se basa en un Algoritmo Genético de estado estable [18], en el cual, el cromosoma de un individuo codifica el patrón de las de capas repetibles de una arquitectura (conocidos como *celdas*) mediante un grafo acíclico, como se observa en la Figura 2. Las operaciones convolutivas son definidas de ante mano y dispuestas en un listado de capas convolutivas, junto otras limitaciones que puedan presentar las arquitecturas. Dependiendo de las características del Espacio de Búsqueda, los grafos acíclicos pueden tomar diversas formas para proveer una adecuada representación. Por ejemplo, un grafo donde los nodos representen las operaciones convolutivas y la cantidad de nodos sea variable, u otro grafo donde las aristas representen las operaciones convolutivas y la cantidad de aristas sea constante. Estos ejemplos se presentan en los espacios de búsqueda NAS-Bench-101 y NAS-Bench-201, respectivamente.

El Algoritmo 1 inicia generando C individuos desde el espacio de soluciones posibles, $C \subset A$. Este paso se puede observar en las líneas 3 a 6 del Algoritmo. Los individuos que pertenecen al Espacio de Búsqueda tienen iguales posibilidades de ser aleatoriamente seleccionados como parte de una muestra inicial. Los individuos de la muestra son analizados usando un estimador *zero-shot* que puntúa el rendimiento/aptitud de los individuos en la etapa de inicialización, pudiendo prescindir de la necesitad de algún entrenamiento. Los estimadores posibles serán detallados en la sección 4.2.

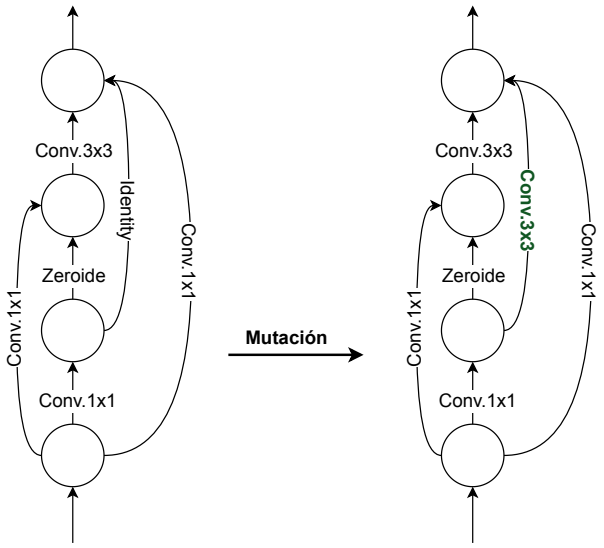


Fig. 2: Las operación *mutación* genera una modificación relativamente pequeña de un individuo.

Luego, de que las C soluciones son puntuadas, solo los mejores P individuos puntuados son entrenados para extraer su rendimiento de forma más precisa *acc* y ser añadidos a la población. El rendimiento, *acc*, se obtiene después de un entrenamiento parcial de pocas épocas. Al puntuar los C individuos en etapa de inicialización, el algoritmo adquiere conocimiento sobre el espacio de búsqueda y puede seleccionar el individuo con mejor rendimiento. Este proceso ayuda a guiar la próxima búsqueda eliminando las malas soluciones candidatas [12]. Este paso se observa en la líneas 7 a 10 del Algoritmo 1.

Una vez definida la población inicial, se produce la evolución durante C ciclos. En cada iteración, el primer paso es realizar una selección de individuos por medio de un torneo. Para ello, S soluciones se seleccionan aleatoria y uniformemente de la población, a modo de muestra. Luego, la solución con el rendimiento más alto, *acc*, del conjunto de soluciones S se selecciona para ser la matriz de la siguiente generación (ciclo). Este paso se observa en las líneas 12 y 13 del Algoritmo 1. Para generar nuevas soluciones, el algoritmo realiza una mutación sobre la solución principal. La mutación funciona modificando ligeramente la composición de las *celdas*.

En cada ciclo el algoritmo generan P individuos nuevos mediante operación de mutaciones sobre el padre seleccionado, ver línea 16. Luego, se obtiene la puntuación para los nuevos individuos utilizando el estimador *zero-shot*. La solución con puntuación más alta se entrena y evalúa su rendimiento, y se agrega a la población. Un ejemplo de mutación de un individuo es dado en la Figura 2

donde la operación Identidad es remplazada por la operación Convolución de filtro 3 x 3. Generar y evaluar P soluciones fortalece el método de búsqueda para encontrar la mejor dirección para la evolución de los padres en el Espacio de Búsqueda [12]. Esto permite al método ser guiado a través de un espacio complejo sin comprometer el tiempo necesario para realizar la evolución ni la complejidad del método de búsqueda.

Cuando la nueva solución se agrega a la población, tiene lugar un mecanismo de regularización (selección de supervivientes), donde la solución más antigua se elimina y descarta, lo que obliga a la exploración del espacio de búsqueda favoreciendo arquitecturas más jóvenes que representan nuevos entornos evolucionados por conocimientos previos adquiridos [12].

Inherentemente, valores de P más altos representan un mayor grado de exploración del Espacio de Búsqueda, mientras que valores de S más altos representan una mayor explotación al aumentar la probabilidad de que las mejores arquitecturas de la población sean seleccionadas como padres para la próxima generación.

En resumen el Algoritmo 1 explora $C+C\cdot P$ individuos que son puntuados por un estimador de los cuales $C + P$ individuos son entrenados. Note que $P \leq C$.

4 Modelos de *zero-shot*

La meta principal del *zero-shot* en NAS es diseñar un agente que pueda puntuar, con buena precisión, las capacidades de las arquitecturas de red candidatas de converger a redes con buen rendimiento en la etapa de inicialización (o sea, sin entrenar), de forma relativa entre las arquitecturas candidatas. A modo que se pueda reemplazar el costoso proceso de entrenamiento con algunas alternativas computacionalmente eficientes.

4.1 Expectativas Teóricas

Antes de profundizar en los detalles de los *zero-shot* existentes, establezcamos primero los principios fundamentales para el diseño ideal. De hecho, un agente de precisión ideal debería abordar tres aspectos principales [8]:

- **Capacidad Expresiva**: El agente debe reflejar qué tan bien la red profunda puede capturar y modelar patrones y relaciones complejos dentro de los datos, lo que puede ser crucial para tareas complejas como conjuntos de datos a gran escala.
- **Capacidad de Generalización**: El agente también debe reflejar la capacidad de la red para generalizar desde los datos de entrenamiento hasta los datos invisibles o fuera de distribución.
- **Entrenabilidad y convergencia**: El agente también debe indicar qué tan rápido la red converge a un nivel de rendimiento deseable.

4.2 Agentes zero-shot abordados

La literatura a propuesto varios enfoques, sumarizados en [8], que son explicados en este trabajo y posteriormente analizados experimentalmente.

Gradient norm es la suma de las normas de cada vector gradiente de cada capa. Para calcular la norma de la gradiente, se ingresa un pequeño conjunto de datos en la red y luego se propagan los valores de pérdida hacia atrás. Posteriormente, se calcula la norma ℓ_2 [1] del gradiente de cada capa y luego se las suma para todas las capas lineales y convolucionales de la red dada. Formalmente, la definición de norma de gradiente G es la siguiente:

$$\text{Grad\_norm} \triangleq \sum_{i=1}^{D} \|\nabla_{\boldsymbol{\theta}_i} L\|_2 \tag{1}$$

donde D, $\boldsymbol{\theta}_i$ y L son el número de capas, el vector de parámetros de la i-ésima capa de una red determinada y los valores de pérdida, respectivamente.

SNIP *Gradient norm* solo mide la propiedad de propagación del gradiente para una red determinada. Para medir conjuntamente la importancia de los parámetros tanto en la inferencia directa como en la propagación de gradientes, SNIP consiste en multiplicar el valor de cada parámetro y su correspondiente gradiente [7]. Formalmente, SNIP se define de la siguiente manera:

$$\text{SNIP} \triangleq \sum_{i}^{D} |\langle \boldsymbol{\theta}_i, \nabla_{\boldsymbol{\theta}_i} L \rangle| \tag{2}$$

donde $\langle \cdot, \cdot \rangle$ representa el producto interno; D, $\boldsymbol{\theta}_i$ y L son el número de capas, el vector de parámetros de la i-ésima capa de una red determinada y los valores de pérdida, respectivamente.

Synflow similar a SNIP, Synflow consiste en mantener el signo del proxy SNIP [21]:

$$\text{Synflow} \triangleq \sum_{i}^{D} \langle \boldsymbol{\theta}_i, \nabla_{\boldsymbol{\theta}_i} L \rangle \tag{3}$$

donde, al igual que en SNIP, $\langle \cdot, \cdot \rangle$ representa el producto interno; D, $\boldsymbol{\theta}_i$ y L son el número de capas, el vector de parámetros de la i-ésima capa de una red determinada y los valores de pérdida, respectivamente.

GraSP Los tres agentes mencionados anteriormente sólo tienen en cuenta las derivadas de primer orden de las redes neuronales. El agente GraSP considera también las derivadas de segundo orden [23]. Específicamente, GraSP se define

por el producto interno de los parámetros y el producto de la matriz de Hessian y los gradientes:

$$\text{GraSP} \triangleq \sum_{i}^{D} -\langle \boldsymbol{H}_i \nabla_{\boldsymbol{\theta}_i} L, \boldsymbol{\theta}_i \rangle \tag{4}$$

donde $\boldsymbol{H}_i$ es la matriz de Hessian de la i-ésima capa. Existen múltiples análisis teóricos para los tres agentes anteriores [8, 20, 25]. Específicamente, se ha demostrado que Synflow y SNIP son constantes por capas en redes lineales durante el proceso de retro-propagación [7, 21]. Además, varios trabajos muestran que Synflow y GraSP son aproximaciones diferentes de las expansiones de Taylor de primer orden de redes neuronales profundas [15, 23].

Fisher information La información de Fisher de una red neuronal se puede aproximar mediante el cuadrado del valor de activación y sus gradientes [10, 22]:

$$\text{Fisher} \triangleq \sum_{i}^{D} \langle \nabla_{\boldsymbol{z}_i} L, \boldsymbol{z}_i \rangle^2 \tag{5}$$

donde $\boldsymbol{z}_i$ es el vector del mapa de características del i-ésima capa de una red dada.

Trabajos anteriores muestran que una aproximación de segundo orden de la expansión de Taylor en una red neuronal es equivalente a una estimación empírica de la información de Fisher [10]. Por lo tanto, medir la información de Fisher de cada neurona/canal de una red determinada puede reflejar la importancia de estas neuronas/canales.

Jacobian covariant Además del gradiente sobre parámetros y activaciones: la covariante jacobiana (Jacob_cov) aprovecha el gradiente sobre los datos de entrada $\boldsymbol{x}$ [11, 14]. Para calcular el agente del Jacob_cov, dado un conjunto de entrada con B muestras de entrada $\{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_B\}$, se calcula la matriz de gradientes $\boldsymbol{J}$ de los resultados de salida $\{y_1, y_2, ..., y_B\}$ con respecto a estas entradas:

$$\boldsymbol{J} = (\nabla_{\boldsymbol{x}_1} y_1, \nabla_{\boldsymbol{x}_2} y_2, ..., \nabla_{\boldsymbol{x}_B} y_B)^T \tag{6}$$

Posteriormente, la matriz de covarianza bruta se genera como:

$$\boldsymbol{G} = (\boldsymbol{J} - \boldsymbol{M})(\boldsymbol{J} - \boldsymbol{M})^T \tag{7}$$

donde $M_{i,j} = \frac{1}{B} \sum_{n=1}^{B} J_{i,n}$. Luego, la matriz de covarianza sin procesar se normaliza para obtener la matriz de covarianza real $\boldsymbol{\Gamma}$:

$$\boldsymbol{\Gamma}_{i,j} = \frac{G_{i,j}}{\sqrt{G_{i,i} G_{j,j}}} \tag{8}$$

donde $\Gamma_{i,j}$ denota las entradas de $\boldsymbol{\Gamma}$. Sea $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_B$ los valores propios B de $\boldsymbol{\Gamma}$; entonces la covariante jacobiana se genera de la siguiente manera:

$$\text{Jacob\_cov} \triangleq -\sum_{i=1}^{B} \left[(\lambda_i + \epsilon) + (\lambda_i + \epsilon)^{-1} \right] \tag{9}$$

donde ϵ es un valor pequeño utilizado para la estabilidad numérica. Como se analiza en [11,14], Jacob_cov puede reflejar la expresividad de redes profundas, por lo que los valores más altos de Jacob_cov indican una mayor precisión.

5 Simulaciones numéricas

Las simulaciones numéricas buscan determinar la eficiencia de los métodos *zero-shot* presentados en la sección anterior para el problema de NAS en el contexto de clasificación de imágenes. La simulación se basa en la ejecución del Algoritmo 1, fijando la función *estimador* en concordancia con los métodos señalados en la sección 4.2, acotando el espacio de búsqueda según es detallado en la sección 5.1. Además, se fijaron como parámetros del Algoritmo 1 los valores $P/S/C = 10/5/200$ como se sugiere en [12], y se establece el conjunto de datos de entrada como CIFAR-10 [6] siguiendo las configuraciones utilizadas y evaluadas en trabajos previos [12]. Cabe mencionar que CIFAR-10 es una clásica base de datos de imágenes construida para análisis de desempeño en clasificación imágenes que consta de 60.000 imágenes a colores de 32x32 de dimensión con 10 tipos de clases. CIFAR-10 consta de 6.000 imágenes para cada una de las clases: aeroplano, automóvil, pájaro, gato, ciervo, perro, rana, caballo, barco, y camiones.

5.1 Espacio de Búsqueda

Para evaluar la efectividad de los modelos se utilizan como Espacio de Búsqueda al conjunto de arquitecturas incluidas en NAS-Bench-101 y NAS-Bench-201. Estos *benchmarks* proveen datos estadísticos sobre el rendimiento de un determinado conjuntos de arquitecturas. La estructura de las arquitecturas que las conforman se define a través un conjunto de capas agrupadas y apiladas (*celdas*) dentro de una macro-estructura definida.

NAS-Bench-101 provee un Espacio de Búsqueda basado en celdas que consta de 423.624 redes neuronales que han sido entrenadas, con tres inicializaciones diferentes, en CIFAR-10 durante 108 épocas cada una. En el espacio de búsqueda NAS-Bench-101 hay tres operaciones posibles: 1×1 y 3×3 convolution y 3×3 max-pooling. Las operaciones de convolución se combinan con la normalización por lotes y las operaciones ReLU para crear un patrón Conv-BNReLU. Para formar arquitecturas completas, cada celda se apila 3 veces, seguida de una capa de max-pooling que sirve para reducir a la mitad la altura y el ancho de la imagen y duplicar el número de canales. Este patrón se repite 3 veces y le sigue

Métodos	Precisión	
	NasBench-101	NasBench-201
grad_norm	89,76 ± 1,08	93,48 ± 0,04
snip	90,88 ± 1,43	93,50 ± 0,13
synflow	90,97 ± 1,27	92,56 ± 0,64
grasp	90,24 ± 0,91	93,41 ± 0,13
fisher	89,93 ± 1,22	93,47 ± 0,16
jacob_cov	**92,42 ± 1,92**	**94,02 ± 0,16**

Table 1: Precisión de clasificación de los modelos *zero_shot*.

una capa global average pooling y una capa de clasificación final con una función de activación softmax [24].

NAS-Bench-201 provee un Espacio de Búsqueda basado en celdas con 5 operaciones posibles: zeroize, skip connection, 1×1 y 3×3 convolution, y 3×3 max-pooling. El diseño de la celda comprende seis aristas y cuatro nodos, donde una arista representa una posible operación a través de dos nodos. Al fijar el tamaño de la celda y el grupo de operaciones, el espacio de búsqueda comprende $5^6 = 15.625$ celdas posibles. Para formar redes completas, las celdas se replican en un esqueleto definido exteriormente. El esqueleto es inicializado con una capa 3×3 convolution, seguido de 3 pilas de celdas conectadas a través de un *residual block*. Las pilas de celdas contienen N = 5 celdas apiladas y el *residual block* consta de una capa 2×2 *average pooling* con *stride* = 2 y una capa 1×1 convolution. El esqueleto finaliza con una capa global average pooling y una capa de clasificación final con una función activación softmax [4, Figure 1].

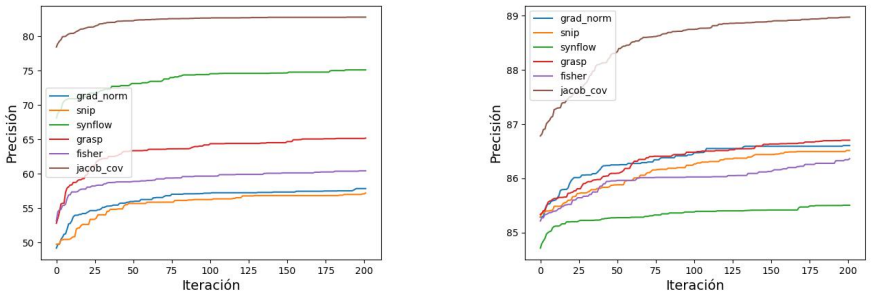
Se utilizan los datos proveídos por los *benchmarks* sobre el rendimiento de estas arquitecturas tanto fines estadísticos como para prescindir de entrenar las arquitecturas, reduciendo los costes de este trabajo.

5.2 Ejecución y Resultados

El desarrollo del experimento se basa en códigos fuente proveídos en trabajos previos [8, 12]. Los códigos fuentes fueron adaptados y ejecutados dentro de un cuaderno Google Colab con Tipo de entorno de ejecución Python 3 (versión 3.10), contando con un acelerador por hardware T4 GPU y capacidad de RAM configurada como alta (aproximadamente 54 GB).

El experimento se realiza ejecutando el Algoritmo 1 utilizando un *zero-shot* como función *estimar* y acotando las funciones *arquitectura_aleatoria* y *mutacion* dentro de uno de los Espacio de Búsqueda. Para cada combinación entre *zero-shot* y Espacio de Búsqueda, el algoritmo se ejecuta treinta veces y los resultados promediados se expresan en la Tabla 1, agrupados según la combinación pertinente. En la Figura 3 se presentan las curvas precisión promedio en función al número de iteraciones transcurridas en el ciclo ubicado entre las lineas 11 y 23 del Algoritmo 1, agrupadas según el *zero-shot* utilizado.

Tanto los resultados consolidados y curvas de convergencias indican que el modelo basado en *jacob_cov* presenta el mejor resultado promedio para la clasifi-



(a) Nas-Bench-101 (b) Nas-Bench-201

Fig. 3: Curva de convergencia de Precisión de clasificación.

cación de imágenes de CIFAR-10, considerando las arquitecturas de los Espacios de Búsquedas. Dentro del análisis estadístico de los resultados, el *p–value* correspondiente al *F-static* del one-way ANOVA es inferior a 0.05, específicamente $1.8636e-12$, lo que sugiere que uno o más conjuntos de resultados son significativamente diferentes. Las pruebas post-hoc sugieren tambien suguieren que *jacob_cov* es significativamente distinta al resto de pruebas.

Por otra parte, observamos que Synflow presenta un buen rendimiento para las pruebas dentro del espacio de búsqueda NAS-Bench-101 y un desempeño deficiente dentro del espacio de búsqueda NAS-Bench-201. Sugiriendo que un enfoque particular podría tener diferentes desempeños dependiendo del Espacio de Búsqueda.

En cuanto a los tiempos de computo de las búsquedas se puede indicar que las mismas, son ampliamente inferiores a una alternativa con entrenamiento y evaluación. El tiempo medio de la estimación sin entrenamiento de un individuo es de 216 y 277 segundos para NAS-Bench-101 y NAS-Bench-201, respectivamente. En contraste con el tiempo de una ejecución con entrenamiento y evaluación, que ronda entre 43.200 y 55.400 segundos para NAS-Bench-101 y NAS-Bench-201, respectivamente.

En el Espacio de Búsqueda del NAS-Bench-101, la Estrategia de Búsqueda puntúa en promedio $327,90 \pm 44,79$ individuos (0.08% del Espacio de Búsqueda) y entrena parcialmente en promedio 60.37 ± 24.94 individuos (el 0.01% del Espacio de Búsqueda). En el Espacio de Búsqueda del NAS-Bench-201, la Estrategia de Búsqueda puntúa en promedio $718,38 \pm 129,60$ individuos (4.6% del Espacio de Búsqueda) y entrena parcialmente en promedio $110,95 \pm 39,40$ individuos (el 0.71% del Espacio de Búsqueda).

6 Conclusiones y Trabajos Futuros

Dentro de NAS, la exploración del espacio de búsqueda tiende a ser costosa debido al costo inherente al entrenamiento de las arquitecturas. Por ello, el uso de

modelos *Zero-Shot* puede disminuir la cantidad de entrenamientos requeridos, en particular desechando aquellos que tienen una probabilidad de tener un pobre desempeño, a través de proveer un método de comparación del rendimiento potencial de las arquitecturas incluso antes de su respectivo entrenamiento.

Dada la cantidad de modelos *Zero-Shot* existentes propuestos en la literatura, en este trabajo hemos realizados simulaciones numéricas para determinar el desempeño sobre la base de datos CIFAR-10 en dos espacios de soluciones NAS-bench-101 y NAS-bench-201. Para ello, expandimos un algoritmo evolutivo propuesto en un trabajo anterior, y formamos una manera de evaluar el rendimiento de un modelo en comparación a otros. Los resultados indican que el modelo basado en covarianza jacobiana presenta resultados prometedores en los escenarios estudiados.

Como trabajo futuro los autores proponen estudiar el desempeño de los modelos *Zero-Shot* sobre otros espacios de soluciones y base de datos disponibles. Por otra parte también se propone extender este estudio en el contexto de optimización multi-objetivo.

References

1. Abdelfattah, M.S., Mehrotra, A., Dudziak, Ł., Lane, N.D.: Zero-cost proxies for lightweight nas. In: International Conference on Learning Representations (2021)
2. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of machine learning research **13**(2) (2012)
3. Bouzidi, H., Niar, S., Ouarnoughi, H., Talbi, E.G.: Sonata: Self-adaptive evolutionary framework for hardware-aware neural architecture search. arXiv preprint arXiv:2402.13204 (2024)
4. Dong, X., Yang, Y.: Nas-bench-201: Extending the scope of reproducible neural architecture search. arXiv preprint arXiv:2001.00326 (2020)
5. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. Journal of Machine Learning Research **20**(55), 1–21 (2019)
6. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
7. Lee, N., Ajanthan, T., Torr, P.: SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY. In: International Conference on Learning Representations (2019)
8. Li, G., Hoang, D., Bhardwaj, K., Lin, M., Wang, Z., Marculescu, R.: Zero-shot neural architecture search: Challenges, solutions, and opportunities. arXiv preprint arXiv:2307.01998 (2023)
9. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055 (2018)
10. Liu, L., Zhang, S., Kuang, Z., Zhou, A., Xue, J., Wang, X., Chen, Y., Yang, W., Liao, Q., Zhang, W.: Group fisher pruning for practical network compression. In: Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event. Proceedings of Machine Learning Research, vol. 139, pp. 7021–7032. PMLR (2021)
11. Lopes, V., Alirezazadeh, S., Alexandre, L.A.: Epe-nas: Efficient performance estimation without training for neural architecture search. In: International Conference on Artificial Neural Networks. pp. 552–563. Springer (2021)

12. Lopes, V., Santos, M., Degardin, B., Alexandre, L.A.: Guided evolutionary neural architecture search with efficient performance estimation. arXiv preprint arXiv:2208.06475 (2022)
13. Mellor, J., Turner, J., Storkey, A., Crowley, E.J.: Neural architecture search without training. In: International Conference on Machine Learning. pp. 7588–7598. PMLR (2021)
14. Mellor, J., Turner, J., Storkey, A., Crowley, E.J.: Neural architecture search without training. In: International Conference on Machine Learning. pp. 7588–7598. PMLR (2021)
15. Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J.: Importance estimation for neural network pruning. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019. pp. 11264–11272. Computer Vision Foundation / IEEE (2019)
16. O'shea, K., Nash, R.: An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458 (2015)
17. Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient neural architecture search via parameters sharing. In: International conference on machine learning. pp. 4095–4104. PMLR (2018)
18. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: Proceedings of the aaai conference on artificial intelligence. vol. 33, pp. 4780–4789 (2019)
19. Shen, Y., Li, Y., Zheng, J., Zhang, W., Yao, P., Li, J., Yang, S., Liu, J., Cui, B.: Proxybo: Accelerating neural architecture search via bayesian optimization with zero-cost proxies. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 37, pp. 9792–9801 (2023)
20. Shu, Y., Dai, Z., Wu, Z., Low, B.K.H.: Unifying and boosting gradient-based training-free neural architecture search. Advances in Neural Information Processing Systems **35**, 33001–33015 (2022)
21. Tanaka, H., Kunin, D., Yamins, D.L., Ganguli, S.: Pruning neural networks without any data by iteratively conserving synaptic flow. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems. vol. 33, pp. 6377–6389. Curran Associates, Inc. (2020)
22. Theis, L., Korshunova, I., Tejani, A., Huszár, F.: Faster gaze prediction with dense networks and fisher pruning. CoRR **abs/1801.05787** (2018)
23. Wang, C., Zhang, G., Grosse, R.B.: Picking winning tickets before training by preserving gradient flow. In: International Conference on Learning Representations. OpenReview.net (2020)
24. Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., Hutter, F.: Nas-bench-101: Towards reproducible neural architecture search. In: International conference on machine learning. pp. 7105–7114. PMLR (2019)
25. Zhou, Q., Sheng, K., Zheng, X., Li, K., Sun, X., Tian, Y., Chen, J., Ji, R.: Training-free transformer architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 10894–10903 (2022)
26. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)
27. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 8697–8710 (2018)