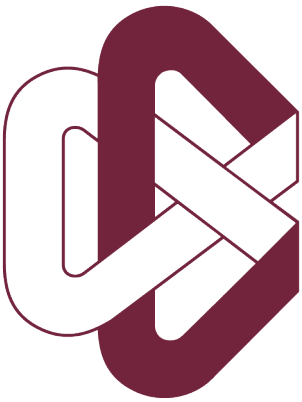

CIMAT

Búsqueda Automática de Arquitecturas Neuronales para clasificación mediante NNI

01 de junio de 2023

Autora: *Alondra Elizabeth Matos Mendoza*

Maestría en Cómputo Estadístico, CIMAT Sede Monterrey

Resumen

La Búsqueda Automática de Arquitecturas Neuronales (NAS) ha destacado al revelar arquitecturas superiores a las diseñadas manualmente, avanzando así el estado del arte en diversas áreas. Su papel es esencial para impulsar la eficiencia, el rendimiento y la accesibilidad en el desarrollo de soluciones de aprendizaje automático. NNI facilita la implementación y experimentación de nuevos algoritmos de AutoML y búsqueda de arquitecturas neuronales. Este documento explora las herramientas de NNI para implementar algoritmos de NAS, centrándose en la tarea de clasificación y destacando resultados utilizando la base de datos Fashion-MNIST.

Index Terms

AutoML, NAS, NNI, clasificación.

I. Introducción

El principal problema al buscar arquitecturas neuronales que logren un alto rendimiento predictivo en datos no vistos sin utilizar AutoML (aprendizaje automático automatizado) es la complejidad y la exigencia de tiempo asociadas con la exploración manual del vasto espacio de búsqueda de modelos. Sin AutoML, la tarea de seleccionar y ajustar manualmente los hiperparámetros, la arquitectura del modelo y otros componentes puede ser laboriosa y propensa a errores.

En la actualidad, la Búsqueda de Arquitecturas Neuronales (NAS) se realiza de forma automática en Python a través de varias bibliotecas de AutoML. AutoKeras, una de las más populares, se ha diseñado para ser accesible incluso para aquellos sin experiencia profunda en aprendizaje profundo. Aunque proporciona una interfaz fácil de usar para construir y entrenar modelos, tiene ciertas limitaciones en la flexibilidad para personalizar el espacio de búsqueda en comparación con enfoques NAS más avanzados.

Sin embargo, recientemente Microsoft creó un paquete llamado Neural Network Intelligence (NNI), el cual incluye un conjunto de herramientas de AutoML de código abierto diseñadas para optimizar hiperparámetros, explorar arquitecturas neuronales, comprimir modelos y realizar ingeniería de funciones.

A diferencia de AutoKeras, NNI ofrece una variada selección de APIs para construir el espacio de búsqueda. Incluye APIs de alto nivel que permiten la integración de conocimientos humanos sobre arquitecturas o espacios de búsqueda eficaces, así como APIs de bajo nivel, que consisten en una lista de primitivas para construir una red de una operación a otra.

Además, NNI ofrece diversas estrategias de exploración en el espacio de búsqueda, algunas potentes pero más lentas, otras menos óptimas pero eficientes. Los usuarios siempre pueden elegir la que mejor se adapte a sus necesidades.

El objetivo principal del presente reporte es explorar las características y capacidades de NNI para NAS. En particular, se utilizó NAS para encontrar algoritmo de clasificación de imágenes basado en redes convolucionales.

Como se reflejará en los resultados, NNI ofrece herramientas que cuentan con un gran potencial para descubrir arquitecturas altamente eficientes en tareas de clasificación, destacando la caracterización personalizada gracias a la flexibilidad de la paquetería. En la Sección II, se presentan las características de los experimentos NAS realizados con NNI, mientras que la Sección III detalla los hiperparámetros establecidos para la experimentación y muestra los resultados obtenidos.

II. Materiales y Métodos

NAS es una técnica de aprendizaje que automatiza la búsqueda de la arquitectura óptima de una red neuronal para una tarea específica. Emplea algoritmos para explorar un espacio de búsqueda predefinido de posibles arquitecturas, evaluar su rendimiento en una tarea dada y refinar

iterativamente la búsqueda en función de los resultados obtenidos.

Utilizando la paquetería NNI, se establecieron los siguientes parámetros para realizar el proceso NAS:

- *Espacio de Búsqueda*

  Se probaron dos espacios de búsqueda, uno personalizado y otro predefinido conocido como *NasBench201*, propuesto por Dong y Yang [3].

  - Espacio de búsqueda personalizado

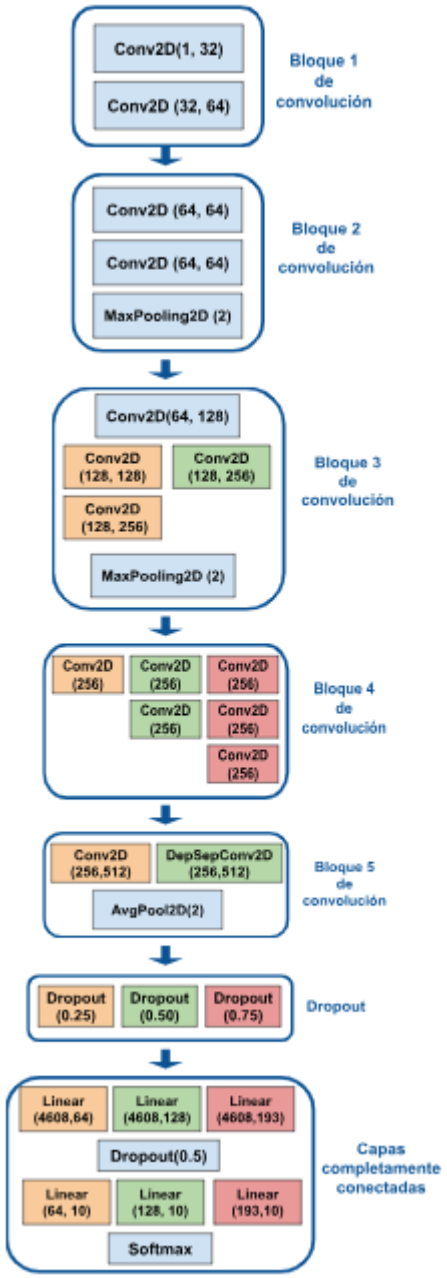

Figura 1: Esqueleto base de cada candidato a arquitectura. Los conjuntos de opciones se representan con los colores amarillo, verde y rojo.

La estructura del modelo base en el espacio personalizado se ilustra en la Figura 1. Este modelo consta de cinco bloques que incluyen capas convolucionales, capas de normalización por lotes (BatchNorm) y funciones de activación ReLU.

En el tercer bloque, hay una capa convolucional seguida de dos configuraciones posibles de capas convolucionales, lo que determina si se incluye una o dos capas adicionales de convolución.

El cuarto bloque es un bloque repetitivo que puede repetirse de una a tres veces.

En el quinto bloque, se elige entre dos capas convolucionales: una estándar y otra de convolución separable en profundidad (Depthwise Separable Convolution). Esta última descompone la convolución convencional en dos etapas: convoluciones separadas para cada canal de entrada (convolución en profundidad) y convoluciones 1x1 (convolución en puntos) para combinar la información de los canales de salida, reduciendo parámetros y operaciones.

Finalmente, se definen dos capas totalmente conectadas con dimensiones variables (64, 128 o 193). Además, se incorpora una tasa de dropout con valores aleatorios de 0.25, 0.50 o 0.75.

- NasBench201

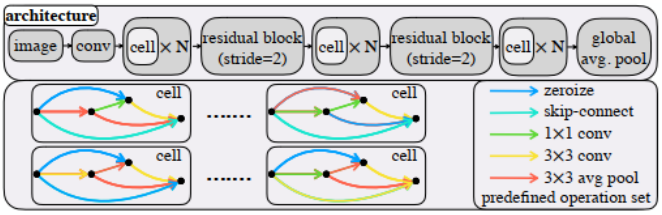

Figura 2: **Arriba:** esqueleto macro de cada candidato a arquitectura. **Abajo a la izquierda:** ejemplos de una celda neuronal con 4 nodos. Recuperado de [3].

El cuerpo principal del esqueleto del espacio de búsqueda NasBench201 incluye tres pilas de celdas conectadas por un bloque residual. Cada celda se apila N = 5 veces, con el número de canales de salida como 16, 32 y 64 para las primeras, segundas y terceras etapas, respectivamente. El bloque residual intermedio es el bloque residual básico con un paso de 2. La ruta de acceso directo en este bloque residual consta de una capa de average pooling 2x2 con un paso de 2 y una convolución de 1x1. El esqueleto termina con un average pooling global para aplanar el mapa de características en un vector de características. La clasificación utiliza una capa totalmente conectada con una capa softmax para transformar el vector de características en la predicción final.

Como se muestra en la Figura 2, cada celda en el espacio de búsqueda se representa como un Grafo Acíclico Dirigido (DAG) densamente conectado, en donde se conceptualiza a los nodos como tensores y a las aristas como operadores. Cada operador (o arista) se elige de un conjunto

de operadores predefinidos que incluye opciones como: $None$, $skip\ connect$, $convolution\ 1X1$, $convolution\ 3X3$ y $average\ pooling\ 3X3$.

La estructura básica de un bloque residual incluye una conexión de atajo (shortcut connection) que permite que la entrada original se sume a la salida de las capas convolucionales. Esto se hace mediante una conexión directa, lo que significa que la información original fluye sin cambios a través de la red.

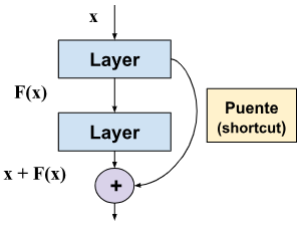

Figura 3: Ejemplo de bloque residual. Aquí la conexión residual se salta dos capas.

- *Estrategia de Búsqueda*

  Hay dos enfoques para explorar el espacio de modelos: la estrategia de múltiples intentos y la estrategia de un solo intento. La estrategia de múltiples intentos implica entrenar de manera independiente cada modelo muestreado en el espacio. En cambio, la estrategia de un solo intento implica muestrear el modelo desde un supermodelo.

  En este caso, se utilizó un enfoque de múltiples intentos mediante un algoritmo de evolución regularizada, también denominado evolución por envejecimiento. Este algoritmo se fundamenta en el .ªlgoritmo 1"delineado por Real y colaboradores [2].

  Las muestras en el algoritmo se llaman *individuos*. En un principio, los individuos de la población inicial se eligen al azar dentro del espacio de búsqueda, y los restantes son generados mediante un proceso de selección y mutación. A medida que se incorporan nuevos individuos, se elimina el más antiguo para mantener constante el tamaño poblacional.

  A partir de una muestra de la población, se seleccionan dos padres.En caso de que se realice *crossover*, se realiza un cruce aleatorio, donde cada característica del individuo resultante tiene un $50\,\%$ de probabilidad de ser heredada de uno de los padres.. Si los dos progenitores tienen conjuntos de claves distintos, se genera un error.

  Posteriormente, se realiza una mutación en el individuo resultante del cruce o en el individuo padre si no hay *crossover*. La mutación se realiza para cada dimensión del individuo con una probabilidad dada. Si la mutación ocurre, el valor en esa dimensión se reemplaza por un nuevo valor aleatorio generado dentro del espacio de búsqueda definido. Después de crear la arquitectura del hijo, se procede a entrenarla, evaluarla y agregarla a la población, ocupando el lugar del modelo que fue entrenado primero.

  Cabe mencionar que se verifica si la nueva muestra generada es una duplicada de alguna muestra existente. En caso afirmativo, se rechaza y se genera un error.

- *Evaluación del desempeño*

  Para evaluar el desempeño de una arquitectura, se implementó una función que calcula la exactitud y le pérdida (entropía cruzada) obtenida en la muestra de prueba. La exactitud proporciona una medida de la proporción de predicciones correctas, mientras que la pérdida, representada por la entropía cruzada, mide la discrepancia entre las predicciones del modelo y las etiquetas reales, ofreciendo así una evaluación integral de la calidad del modelo.

II-1. Descripción de los datos utilizados : Se empleó una base de datos denominada Fashion-MNIST, la cual contiene imágenes de artículos de Zalando, compuesta por 60,000 ejemplos para entrenamiento y 10,000 para pruebas. Cada ejemplo consiste en una imagen en escala de grises de 28x28 píxeles y una etiqueta asociada perteneciente a una de las siguientes 10 clases disponibles:

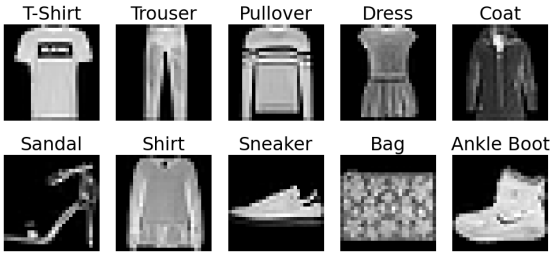

Figura 4: Ejemplo de las imágenes de la muestra

Es importante destacar que la base de datos está equilibrada, lo que significa que hay 600 imágenes por etiqueta. Por lo tanto, la exactitud proporciona una evaluación equitativa del desempeño del modelo.

III. Experimentos y Resultados

La experimentación computacional se realizó en un equipo equipado con un procesador Intel(R) Core(TM) i5-10210U (1.60GHz, 2.11 GHz) y 8 GB de memoria RAM. Los algoritmos se ejecutaron en la unidad central de procesamiento (CPU) debido a la incapacidad de instalar NNI en Google Colab, lo que impidió el uso de la GPU disponible.

Se llevaron a cabo tres experimentos de búsqueda automática de arquitecturas neuronales (NAS): dos utilizando NNI (uno para cada espacio de búsqueda descrito en la sección II) y, con fines de comparación, uno utilizando AutoKeras (ImageClassifier).

Los experimentos realizados con NNI se distinguen entre sí en términos del espacio de búsqueda utilizado. Dado que las arquitecturas que definen el espacio de búsqueda

Nas-Bench201 están diseñadas para imágenes RGB, en ese experimento se llevó a cabo un preprocesamiento de los datos. Esto incluyó la transformación de la dimensión de las imágenes originales para obtener tres canales (R = G = B).

Los parámetros del espacio de búsqueda NasBench201 fueron:

- Número de módulos (celdas) en cada pila: 5.
- Número de canales de salida de la capa inicial (*stem*): 16.

  El *stem* es la parte inicial del modelo que procesa la entrada antes de pasarla a las capas posteriores.

En cuanto a los parámetros del algoritmo de búsqueda evolutivo, en ambos experimentos se estableció un tamaño de población y de muestra de 30 y 10 individuos respectivamente, con una probabilidad de mutación del 0.05 y con crossover activado.

Los resultados se presentan de manera resumida en la Figura 5. Se destaca que el mejor rendimiento se alcanzó en el experimento 1, subrayando la utilidad de diseñar un espacio de búsqueda personalizado.

En términos de tiempo de ejecución, se evidencia que AutoKeras fue considerablemente más lento en comparación con NNI, a pesar de la utilización de un espacio de búsqueda definido explícitamente en ambos casos.

	Experimento 1	Experiemento 2	Experiemento 3
Espacio de búsqueda	Personalizado	NasBench201	ResNet (Bloques residuales)
Estrategia de búsqueda	Algoritmo evolutivo regularizado	Algoritmo evolutivo regularizado	Optimización bayesiana "Optuna"
Número máximo de pruebas	6	6	3
Número de épocas por prueba	3	3	3
Tiempo total de ejecución (s)	11940	15882	15497
Tiempo promedio por prueba (s)	1990	2647	5165
Exactitud del mejor modelo (%)	91.34	88.96	83.96

Figura 5: Tabla de comparación

Las Figuras6 y 7 muestran las mejores configuraciones obtenidas.

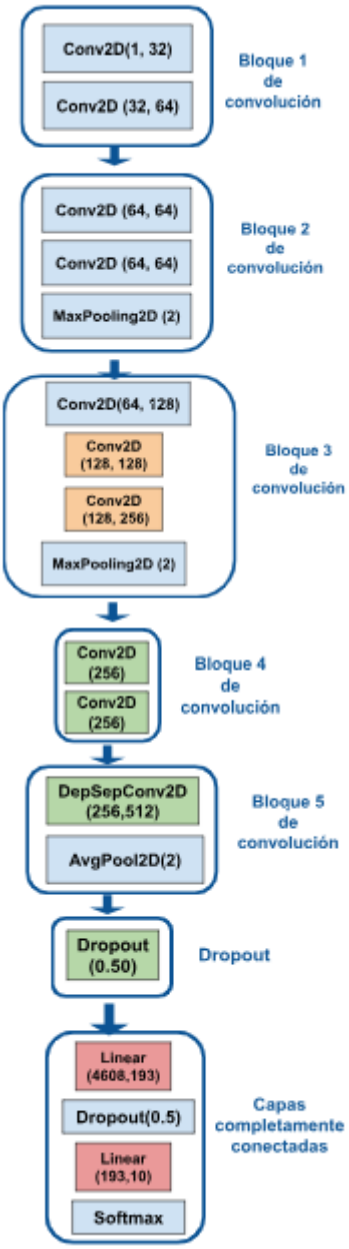

Figura 6: Arquitectura del mejor modelo del espacio de búsqueda personalizado

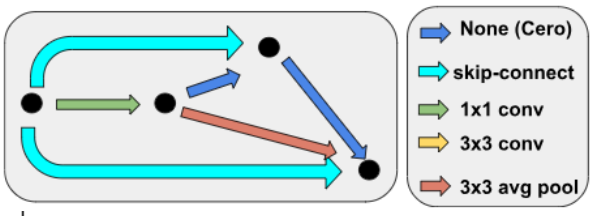

Figura 7: Mejor configuración de celda del espacio de búsqueda de NasBench201

El código de la implementación se puede encontrar en https://github.com/AlondraMM/AutoML-NNI.git.

IV. Conclusiones

La biblioteca NNI es una buena alternativa para llevar a cabo la Búsqueda Automática de Arquitecturas Neuronales (NAS), ya que proporciona una amplia variedad de implementaciones para crear experimentos NAS, incluyendo la capacidad de definir tu propio espacio de búsqueda. Además, la paquetería ya incorpora muchos de los espacios de búsqueda más populares en la actualidad.

A pesar de sus ventajas, es importante tener en cuenta que NNI es una biblioteca relativamente nueva, lo que podría dar lugar a algunos inconvenientes, ya sea en el proceso de instalación o durante la ejecución, debido a posibles errores asociados con su desarrollo y actualizaciones.

Referencias

[1] Microsoft(2020). *"NNI Documentation"*. https://nni.readthedocs.io/en/latest/

[2] Real, E., Aggarwal, A., Huang, Y., & Le, Q. V.(2019, July). *Regularized evolution for image classifier architecture search"*.In Proceedings of the aaai conference on artificial intelligence (Vol. 33, No. 01, pp. 4780-4789).

[3] Dong, X., & Yang, Y.(2020). *"Nas-bench-201: Extending the scope of reproducible neural architecture search"*.arXiv preprint arXiv:2001.00326.