



ugr | Universidad
de **Granada**

TRABAJO FIN DE GRADO
INGENIERÍA EN INFORMÁTICA

**Técnicas de búsqueda de
arquitecturas neuronales para el
diseño automático de redes
convolucionales**

**Aplicación a la clasificación de lesiones
gastrointestinales**

Autor
Alejandro Pinel Martínez

Director
Pablo Mesejo Santiago



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, septiembre de 2021



Técnicas de búsqueda de arquitecturas neuronales para el diseño automático de redes convolucionales

Aplicación a la clasificación de lesiones gastrointestinales

Autor

Alejandro Pinel Martínez

Director

Pablo Mesejo Santiago

Técnicas de búsqueda de arquitecturas neuronales para el diseño automático de redes convolucionales: Aplicación a la clasificación de lesiones gastrointestinales

Alejandro Pinel Martínez

Palabras clave: Aprendizaje Automático, Aprendizaje Profundo, Búsqueda Automática de Arquitecturas Neuronales, Visión por Computador, Gastroenterología

Resumen

Este trabajo de fin de grado trata sobre Búsqueda Automática de Arquitecturas Neuronales, o Neural Architecture Search (NAS), y su aplicación a un problema real de relevancia clínica. NAS se ocupa de investigar técnicas que permitan generar de forma automática una arquitectura de red neuronal para un problema concreto, sin la necesidad de que un experto humano deba diseñar esta red de forma manual.

Se ha realizado una exhaustiva investigación sobre el estado actual del campo y sus publicaciones más importantes. Gracias a este análisis bibliográfico se ha podido concluir que las técnicas empleadas se pueden dividir en base a tres criterios: el espacio de búsqueda (macro-arquitectura y micro-arquitectura), la estrategia de búsqueda (aprendizaje por refuerzo, computación evolutiva y optimización bayesiana, entre otras) y la estimación del rendimiento (morfismo de redes o modelos one-shot, entre otros). En base a este estudio y buscando realizar la comparativa más representativa posible, se han escogido tres técnicas para su implementación y experimentación: Efficient Neural Architecture Search (ENAS), una técnica de aprendizaje por refuerzo y micro-arquitectura; Auto-Keras, que hace uso de optimización bayesiana y morfismo de redes; y Auto CNN, un enfoque evolutivo muy reciente que hace uso tanto de un operador de cruce como de mutación.

Estas tres técnicas han sido utilizadas en el problema de la clasificación automática de pólipos gastrointestinales. Se trata de un conjunto de 76 imágenes de pólipos que se deben clasificar de forma binaria entre aquellos pólipos cancerígenos que deben ser eliminados y aquellos que son benignos. Se parte de un trabajo anterior y se compara con el mejor modelo de deep learning empleado en el mismo. Los resultados muestran que las técnicas de NAS son capaces de encontrar arquitecturas con mejores resultados que los proporcionados por una red diseñada por un experto humano. Entre las tres técnicas, ENAS obtiene los mejores resultados, alcanzando el estado del arte en este problema concreto.

Neural architectures search techniques for automatic design of convolutional networks. Application to gastrointestinal lesions classification

Alejandro Pinel Martínez

Keywords: Machine Learning, Deep Learning, Neural Architecture Search, Computer Vision, Gastroenterology

Abstract

This dissertation is about Neural Architecture Search (NAS) and its application to a real problem of clinic relevance. NAS deals with the investigation of techniques that can automatically generate neural network architectures for a specific problem, without the need for a human expert to design this network manually.

Extensive research has been conducted on the current state of the field and its most important publications. Thanks to this bibliographic analysis, it has been possible to conclude that the techniques used can be divided based on three criteria: the search space (macro-architecture and micro-architecture), the search strategy (reinforcement learning, evolutionary computing and Bayesian optimization among others) and performance estimation (network map or one-shot models, among others). Based on this study and seeking to make the most representative comparison possible, three techniques have been chosen for its implementation and experimentation: Efficient Neural Architecture Search (ENAS), a reinforcement learning and micro-architecture technique; Auto-Keras, which makes use of Bayesian optimization and network morphism; and Auto CNN, a very recent evolutionary approach that makes use of both a crossover and a mutation operator.

These three techniques have been used in the problem of automatic classification of gastrointestinal polyps. It is a set of 76 images of polyps that must be classified in a binary way between those cancerous polyps that must be eliminated and those that are benign. It starts from a previous work and is compared with the best deep learning model used in it. The results show that NAS techniques are able to find architectures with better results than those provided by a network designed by a human expert. Among the three techniques, ENAS obtains the best results, reaching the state of the art in this specific problem.

Yo, **Alejandro Pinel Martínez**, alumno de la titulación Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 32732079C, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

A handwritten signature in blue ink that reads "Alejandro". The signature is fluid and cursive, with a slight flourish at the end. It is written over a thin, light blue curved underline.

Fdo: Alejandro Pinel Martínez

Granada a 6 de septiembre de 2021

D. **Pablo Mesejo Santiago**, Profesor del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado *Técnicas de búsqueda de arquitecturas neuronales para el diseño automático de redes convolucionales, aplicación a la clasificación de lesiones gastrointestinales*, ha sido realizado bajo su supervisión por **Alejandro Pinel Martínez**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a 6 de Septiembre de 2021.

El director:



Pablo Mesejo Santiago

Agradecimientos

Agradezco a Pablo su implicación en este proyecto y toda la ayuda que me ha proporcionado.

Este trabajo se lo dedico especialmente a mi madre, que ha tenido que ser muy fuerte este año y, a pesar de ello, no ha parado de apoyarme y animarme en ningún momento.

Índice general

1. Introducción	1
1.1. Descripción del problema	1
1.2. Motivación	4
1.3. Objetivos	5
1.4. Estructura de la memoria	5
2. Fundamentos teóricos	7
2.1. Machine learning y deep learning	7
2.2. Redes neuronales	8
2.2.1. Capa totalmente conectada	10
2.2.2. Capa Convolutional	11
2.2.3. Capa de Pooling	12
2.2.4. Dropout	13
2.3. Neural Architecture Search	13
2.3.1. Espacio de búsqueda	14
2.3.2. Estrategia de búsqueda	17
2.3.3. Estrategia de estimación del rendimiento	22
3. Estado del arte	25
3.1. Resumen estado del arte	25
3.2. Evolución de NAS	25
3.2.1. Aproximaciones evolutivas	25
3.2.2. Aprendizaje por refuerzo	28
3.2.3. Enfoque jerárquico	29
3.2.4. Optimización bayesiana	30
3.2.5. NAS en la actualidad	30
4. Materiales y métodos	31
4.1. Datos del problema	31
4.2. Métricas	35
4.3. Función de pérdida	38
4.4. Red diseñada por experto	38
4.5. Aprendizaje por refuerzo: ENAS	40

4.6. Optimización bayesiana: Auto-Keras	42
4.7. Algoritmo Evolutivo: Auto CNN	43
5. Planificación e Implementación	47
5.1. Planificación	47
5.2. Implementación	48
5.3. Lenguaje y entorno	50
6. Experimentación	51
6.1. Red diseñada por experto	53
6.2. ENAS	53
6.3. Auto-Keras	56
6.4. Auto CNN	58
6.5. Discusión	59
7. Conclusiones y Trabajos Futuros	63
Bibliografía	65

Índice de figuras

1.1.	Estructura básica de un MLP. Cada nodo representa una operación de cómputo sencilla y se agrupan en capas (representadas por colores). Imagen extraída de [61].	2
1.2.	NAS se engloba dentro del campo del AutoML y la optimización de hiperparámetros. Imagen extraída de [62].	3
1.3.	Tipos de pólipos y su evolución. Aquellos que son cancerígenos se caracterizan por ser de mayor tamaño y tener un carácter más invasivo. Imagen extraída de [64].	4
2.1.	Diferencia entre el ciclo de un modelo tradicional de machine learning y uno de deep learning. Imagen extraída de [11] . . .	8
2.2.	Formulación matemática de un MLP. Imagen extraída de [17].	9
2.3.	Arquitectura de una red neuronal convolucional. Imagen extraída de [26].	10
2.4.	Extracción de información jerárquica de una ConvNet. Se extraen características de bajo nivel en las primeras capas (líneas, esquinas) y de alto nivel (formas complejas, patrones) en las últimas. Imagen extraída de [19].	11
2.5.	Muestra visual de cómo funciona una capa convolucional. El filtro (3x3) procesa una sección de la imagen y la salida pasa a ser el valor de la posición correspondiente al píxel central de la sección. Imagen extraída de [20].	12
2.6.	Ejemplo de funcionamiento de una capa de pooling 2x2 con un <i>stride</i> de 2. Imagen extraída de [26].	12
2.7.	Capa de dropout aplicada sobre la segunda capa totalmente conectada de una pequeña red. Imagen extraída de [24]. . . .	13
2.8.	Esquema de cómo interactúan las tres dimensiones de NAS. Imagen extraída de [27] y traducida.	14
2.9.	A la izquierda una red de tipo cadena, cada capa sigue estrictamente a la anterior. A la derecha, una red añadiendo la posibilidad de escoger la capa anterior a la que se conecta cada capa. Imagen extraída de [8].	15

2.10. La micro-arquitectura optimiza celdas como las de la izquierda y luego se combinan para generar la red completa (derecha). Imagen de [8].	16
2.11. Ejemplo de arquitectura jerárquica. Capas simples son usadas para construir bloques, que a su vez pueden usarse para crear otros bloques más complejos. Imagen extraída de [30].	18
2.12. Esquema del ciclo de un algoritmo evolutivo aplicado a NAS. Imagen extraída de [9]	19
2.13. Esquema del funcionamiento de la estrategia que emplea aprendizaje por refuerzo. Una red controladora genera redes hijas. Imagen extraída de [27] y traducida al español.	20
2.14. En este ejemplo, el controlador produce múltiples salidas que representan cada uno de los parámetros que debe escoger para cada capa. Imagen extraída de [27].	20
2.15. En este ejemplo se puede observar cómo a partir de los resultados de las evaluaciones de tres redes y la función distancia se puede estimar la función original (línea sólida roja) y un intervalo de confianza (líneas de puntos rojos). Imagen extraída de [46]	21
2.16. Ilustración de una arquitectura one-shot. El modelo completo (izquierda) contiene todas los pesos guardados de todas las redes entrenadas. La arquitectura de estas redes es un subgrafo del modelo completo (derecha) y, por tanto, puede utilizar los pesos de este. Imagen extraída de [8].	23
3.1. Número de publicaciones por año dedicadas al estudio de NAS. Se puede notar cómo a partir de 2016 empezó a despertar un interés creciente. El año pasado, por ejemplo, se presentaron 853 artículos dedicados a este campo. Este gráfico ha sido creado con la herramienta Scopus el día 2 de septiembre de 2021 utilizando la query (<i>TITLE-ABS-KEY (neural AND architecture AND search)) AND (LIMIT-TO (SUBJAREA , “COMP”) OR LIMIT-TO (SUBJAREA , “ENGI”) OR LIMIT-TO (SUBJAREA , “MATH”)</i>) . . .	26
4.1. Comparativa entre los dos tipos de luz WL a la izquierda y NBI a la derecha. Estas imágenes son frames de los videos originales y todavía no tienen las regiones de las lesiones recortadas.	32

4.2. Ejemplos de los tres tipos de lesiones presentes en la base de datos. Las tipos (a) y (c) representan lesiones que deben ser eliminadas, mientras que el tipo (b) no presenta la misma gravedad y, por tanto, su eliminación (también llamada resección) no es obligatoria. La fila superior representa pólipos con luz WL, mientras que la de abajo emplea NBI.	33
4.3. Distribución de las clases en los datos.	34
4.4. Ejemplos de cómo dentro de una misma clase (adenomas) las imágenes pueden ser muy diferentes entre sí. Esto se conoce como variabilidad intraclase. La luz utilizada es WL (arriba) y NBI (abajo).	35
4.5. Distribución de las clases en los datos del problema binario. .	36
4.6. Resumen de la arquitectura de la red diseñada por un experto. .	39
4.7. Esquema de una red LSTM. Imagen extraída de Wikipedia .	40
4.8. El controlador (arriba) genera por cada nodo 4 elementos: dos nodos anteriores a los que se tiene que conectar y dos operaciones para aplicar a cada una de estas entradas. Por ejemplo, en el nodo 3 (verde) se puede observar cómo se generan los parámetros 2, 1, avg 5x5 y sep 5x5 que se traducen en la estructura verde del modelo hijo (imagen del centro). Debajo, se puede observar un grafo que representa la arquitectura final de la celda. Imagen extraída de [68].	41
4.9. Ciclo básico del algoritmo genético. Imagen extraída de [66]. .	43
4.10. Ejemplo de arquitectura generada por Auto CNN con código “32-64-max-64-256-mean-512-256-256-512”. Imagen extraída de [66].	44
5.1. Diagrama de flujo de la realización de los experimentos. . . .	49
6.1. Matriz de confusión de la red diseñada por un experto.	53
6.2. La arquitectura generada por ENAS, consistente en 3 capas normales (azul) y 2 capas de reducción (rojo)	54
6.3. Esquema resumen de la arquitectura generada. Es notable el reducido número de parámetros.	54
6.4. Esquema de una celda normal. Es notable que sólo se realice una operación de convolución en toda la celda y que el segundo nodo utilice dos capas de pooling.	55
6.5. Esquema de una celda de reducción. Al igual que en la celda normal, uno de los nodos se utiliza exclusivamente para capas de pooling. Además, todos los nodos utilizan la salida de la celda inmediatamente anterior, sin utilizar la del nodo input 0.	55
6.6. Matriz de confusión del modelo ENAS	56
6.7. Matriz de confusión del modelo Auto-Keras	57
6.8. Arquitectura final generada por Auto-Keras.	57

6.9. Matriz de confusión del modelo Auto CNN	58
6.10. Arquitectura final generada por Auto CNN.	59

Índice de cuadros

3.1. Resumen de la literatura más relevante sobre NAS ordenada de modo ascendente según año de publicación. Leyenda: AE: Algoritmo Evolutivo; AR: Aprendizaje por Refuerzo; OB: Optimización Bayesiana; BB: Búsqueda Bi-nivel; RS: <i>Random Search</i> ; BF: Estimaciones de Baja Fidelidad; CA: Curva de Aprendizaje; MR: Morfismo de redes; WS: <i>Weight Sharing</i> ; OS: Modelos <i>One-Shot</i> ; EP: Evolución de Pesos. CH: Caché.	27
4.1. Matriz de confusión	36
4.2. Resultados originales de [11]. Obtiene una accuracy del 80.3 % y un valor de ponderación de 71.1 %.	39
5.1. Planificación inicial del proyecto.	48
5.2. Planificación final del proyecto.	48
6.1. Resultados finales de los experimentos, en donde se compara el rendimiento de la red profunda diseñada por un experto [11], con la proporcionada por ENAS [33], Auto-Keras [43] y Auto CNN [66]. Se puede observar que los mejores resultados (indicados en negrita) son obtenidos por ENAS y, del mismo modo, todas las técnicas de NAS superan el rendimiento de la red diseñada manualmente.	52

Capítulo 1

Introducción

1.1. Descripción del problema

El campo de la inteligencia artificial (IA) ha sufrido un desarrollo espectacular en los últimos tiempos gracias a la aparición del aprendizaje profundo o deep learning (DL) [1, 2, 3], subcampo perteneciente al campo del conocimiento del aprendizaje automático o machine learning (ML). Gracias a esta familia de técnicas se ha podido avanzar notablemente en la resolución de problemas como la clasificación de imágenes [4], el procesado del lenguaje natural [5] o la conducción autónoma [6]. El DL consiste en la utilización de modelos computacionales que extraen información de los datos de entrada permitiéndoles obtener una representación jerárquica de los mismos [1]. Esta tecnología se basa en la utilización de redes neuronales artificiales [7]: estructuras formadas por múltiples elementos simples de procesado, interconectados formando capas, y parametrizados de forma que los pesos que tomen las conexiones representan el conocimiento aprendido de cara a realizar una tarea concreta.

En la Figura 1.1 se puede observar la estructura de una red neuronal conocida como Perceptrón Multicapa o Multilayer Perceptron (MLP). Cada uno de los nodos de la figura representa una operación sencilla¹ que se aplica sobre las salidas de los nodos anteriores a los que está conectado. Los nodos se agrupan en capas que realizan los cálculos sobre las mismas entradas de forma paralela (en la figura, las capas están representadas por los conjuntos de nodos de cada color). La entrada de la primera capa son los propios datos del problema y la salida de la última capa es la solución aportada por la red al problema. El número de nodos, su disposición, el tipo de operación que realiza cada uno y las conexiones entre ellos conforman la arquitectura de

¹Función no lineal de una suma ponderada de entradas por pesos de las conexiones aprendidos en el entrenamiento.

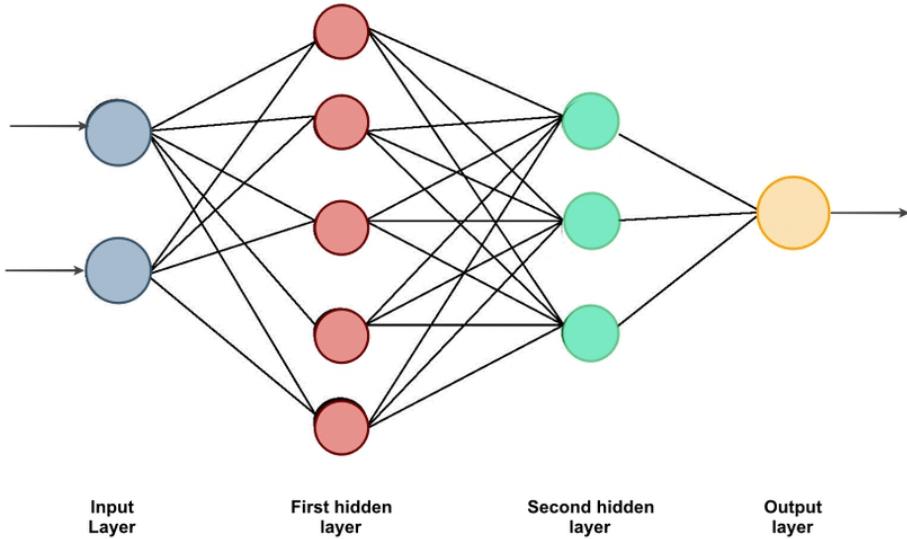


Figura 1.1: Estructura básica de un MLP. Cada nodo representa una operación de cómputo sencilla y se agrupan en capas (representadas por colores). Imagen extraída de [61].

la red.

Esta arquitectura requiere ser cuidadosamente seleccionada por un experto humano. No existe un “modelo universal” para la arquitectura sino que cada problema requiere un número y tipo de capas concreto. Estas pueden realizar distintas operaciones y tener un número arbitrario de neuronas, además de que pueden conectarse a una o varias de las capas anteriores. Todo esto resulta en que **seleccionar la arquitectura es un proceso lento y complicado, que involucra muchas pruebas y que requiere de la intervención de ingenieros con un alto conocimiento sobre el problema y sobre las propias arquitecturas**.

Para intentar subsanar este problema, se han dedicado grandes esfuerzos en la Búsqueda Automática de Arquitecturas Neuronales o Neural Architecture Search (NAS) [8], cuyo objetivo es la automatización del diseño de redes neuronales artificiales. Como se puede ver en la Figura 1.2, NAS se engloba dentro de la optimización de hiperparámetros (ya que la propia arquitectura puede considerarse uno) que a su vez pertenece al campo del AutoML [12]. El AutoML se define como el proceso de automatizar la aplicación de ML a problemas concretos del mundo real. A lo largo de los años, distintas técnicas de NAS han sido propuestas, aplicándose en una gran variedad de problemas donde han conseguido resultados similares o mejores a los diseños

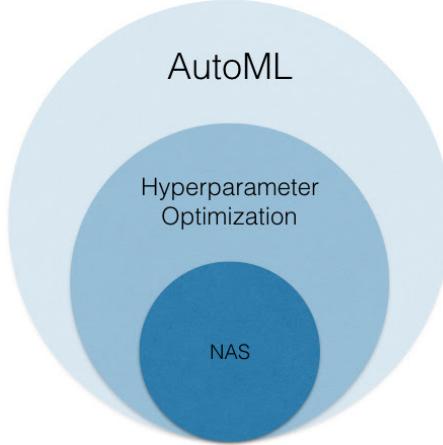


Figura 1.2: NAS se engloba dentro del campo del AutoML y la optimización de hiperparámetros. Imagen extraída de [62].

de los expertos [39, 40].

Este trabajo de fin de grado se ocupa de **explorar los enfoques más relevantes de NAS en la actualidad e implementar algunas de estas técnicas para realizar un estudio comparativo en un problema concreto**. Como caso de estudio se utilizará un problema relacionado con el cáncer colorrectal, un tipo de cáncer provocado comúnmente por pólipos adenomatosos [63]. Los pólipos (véase Figura 1.3) son protuberancias en la mucosa de la superficie interna del tubo digestivo, principalmente en el esófago, el estómago, el intestino delgado y el intestino grueso. Los pólipos adenomatosos son afecciones premalignas, que tienen una alta probabilidad de provocar cáncer si no son eliminados preventivamente. Para ello se realiza una operación conocida como resección endoscópica o polipectomía en la que se extirpan las lesiones detectadas. Durante esta operación, el gastroenterólogo tiene visión del interior del tubo digestivo y es su trabajo identificar las protuberancias existentes y clasificarlas entre benignas y malignas, extirmando estas últimas [63, 64]. Esta tarea, de identificación y clasificación, se presta a la automatización mediante técnicas de IA. De este modo, se podría ahorrar tiempo y recursos de profesionales altamente especializados al liberarlos de la necesidad de realizar manualmente procesos lentos y con gran margen para la subjetividad, como la clasificación visual de pólipos gastrointestinales.

En este trabajo, partiremos de una publicación científica previa [10] donde se trataba la clasificación automática de lesiones gastrointestinales en imágenes de colonoscopia convencional. En dicho artículo, se proponían métodos de ML para clasificar lesiones gastrointestinales. Dicho estudio fue posteriormente ampliado en un trabajo de fin de grado [11] en el que se

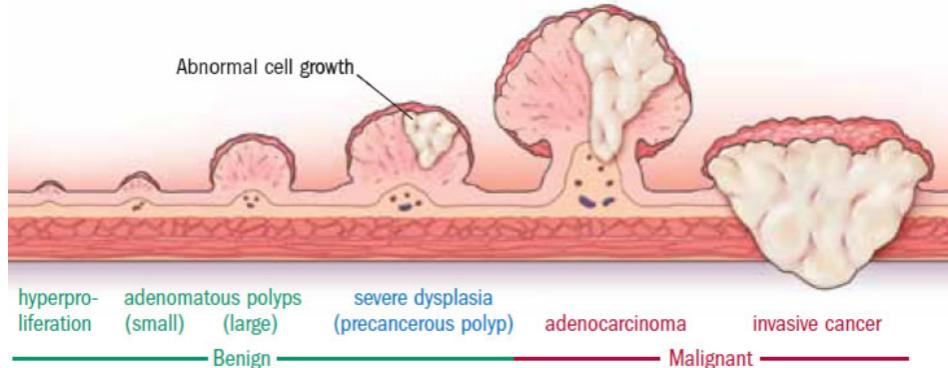


Figura 1.3: Tipos de pólipos y su evolución. Aquellos que son cancerígenos se caracterizan por ser de mayor tamaño y tener un carácter más invasivo. Imagen extraída de [64].

propuso una red neuronal profunda creada por un experto humano como modelo de DL. Ahora, se utilizarán los mismos datos y diversos métodos de NAS para comparar resultados con el modelo de DL propuesto previamente.

1.2. Motivación

Este trabajo tratará de identificar y comparar las técnicas más modernas de NAS aplicándolas en el problema de clasificación de pólipos gastrointestinales. El objetivo será encontrar una técnica que sea eficaz, entendiendo como eficaz que las redes generadas obtengan resultados comparables o mejores que los diseños de expertos humanos, y que sea capaz de entrenar en un tiempo lo suficientemente reducido.

El estudio y mejora de las técnicas actuales de NAS presenta los siguientes beneficios:

- Permite ahorrar muchos recursos y tiempo actualmente dedicados a buscar la mejor arquitectura para cada problema específico.
- Las redes generadas automáticamente no sufren de los mismos sesgos que un diseñador puede tener al elegir su arquitectura y, por tanto, su diseño puede ser más óptimo en algunos casos.
- Abre las puertas a que personas con menos experiencia pudieran aplicar modelos de DL a otros problemas.
- Proporciona un útil punto de partida al diseñador aún en el caso de querer ajustar la red manualmente.

Además, aplicado al problema de clasificación de pólipos, mejorar las técnicas existentes de clasificación automática de lesiones gastrointestinales conlleva los siguientes beneficios:

- Libera a expertos médicos de realizar una tarea rutinaria, pudiendo dedicar su tiempo a tareas más prioritarias.
- Ahorra costes de realizar operaciones de polipectomía al requerir menos tiempo por parte de los profesionales.
- Acelera el tiempo necesario para dar un diagnóstico, al poder ejecutarse los algoritmos de clasificación en tiempo real.
- Mejora la precisión del diagnóstico mediante una técnica bien entrenada que no sufre de la subjetividad propia de cualquier ser humano.

1.3. Objetivos

El objetivo principal de este trabajo es hacer un estudio amplio y sistemático del estado del arte en NAS y utilizar algunas de las técnicas más recientes en un problema real, la clasificación de pólipos gastrointestinales, comparando con el mejor modelo existente de DL para dicho problema. Este objetivo principal puede ser dividido en los siguientes objetivos parciales:

1. Estudio pormenorizado de la bibliografía al respecto.
2. Comparación de las técnicas y modelos del estado del arte en NAS y selección de aquellas a emplear en el caso de estudio.
3. Implementación, validación y aplicación de las técnicas seleccionadas al caso de estudio.
4. Análisis y discusión de los resultados obtenidos.

1.4. Estructura de la memoria

Esta memoria se estructura en 7 capítulos tal y como se describe a continuación:

- En la *Introducción*, se expone el problema de las redes neuronales, qué es el campo del NAS y el caso de estudio de la clasificación de lesiones gastrointestinales. A continuación, se explican las motivaciones del trabajo, sus objetivos y la estructura de la memoria.

- En el segundo capítulo, *Fundamentos Teóricos*, se explicará en más profundidad qué es el aprendizaje automático y el deep learning, cómo se estructura una red neuronal y sus principales capas. Después, se hablará del NAS: cómo se clasifican las distintas técnicas según tres criterios principales y los enfoques más populares de cada uno de estas dimensiones.
- En el tercer capítulo, *Estado del Arte*, se mostrará una tabla con los trabajos más relevantes en el campo del NAS y se hará una comparación de la evolución histórica de las distintas técnicas y los enfoques más populares en la actualidad.
- En el cuarto capítulo, *Materiales y Métodos*, se presentarán en profundidad los datos y particularidades del caso de estudio, la función de pérdida y métricas que se utilizarán en los experimentos y se explicarán las tres técnicas de NAS seleccionadas.
- En el quinto capítulo, *Planificación e Implementación* se hablará de la planificación temporal del proyecto, los detalles de su implementación y las herramientas utilizadas.
- En el sexto capítulo, *Experimentación* se mostrarán los resultados obtenidos de los modelos presentados en el problema de clasificación de pólipos gastrointestinales. Seguirá un análisis de los resultados y las redes obtenidas.
- Por último, en el capítulo siete, *Conclusiones y Trabajos Futuros* se resumirán las conclusiones extraídas de este trabajo, así como se tratará brevemente las posibilidades abiertas de trabajos futuros.

Capítulo 2

Fundamentos teóricos

2.1. Machine learning y deep learning

El machine learning (ML) o aprendizaje automático [13, 14] es un subcampo de la IA cuyo objetivo es desarrollar algoritmos que aprendan a realizar una o varias tareas a partir de los propios datos, es decir, sin necesidad de ser programados explícitamente para ello. Una definición más formal es propuesta en [15]: “*Un programa aprende con la experiencia E con respecto a una clase de tareas T y una métrica de rendimiento P si su rendimiento en T, medido por P, mejora con la experiencia E*”. Estas técnicas son especialmente útiles en problemas complejos donde es difícil emplear un algoritmo convencional como la visión por computador o el procesamiento del lenguaje natural [4, 5]. El ML ha sido ampliamente explorado y dispone de una gran variedad de técnicas relacionadas [16].

El aprendizaje automático se suele clasificar en tres grandes grupos dependiendo del tipo de datos disponible:

- **Aprendizaje supervisado:** Se dispone de una base de datos con ejemplos de *inputs* y sus respectivos *outputs*. El algoritmo debe analizar las características que relacionan cada input con su output y generalizar una serie de reglas para generar *outputs* a partir de nuevos *inputs*.
- **Aprendizaje no supervisado:** La base de datos solo dispone de *inputs*. El objetivo es que el algoritmo encuentre patrones ocultos en los datos, o que extraiga un vector de características de estos.
- **Aprendizaje por refuerzo:** Los datos que el algoritmo procesa provienen de un entorno dinámico en el que se mueve y debe tomar una serie de acciones. Al interactuar con el entorno obtendrá un valor de recompensa cuyo objetivo es maximizar.

Modelo tradicional de reconocimiento de características y patrones (pre-2012):

► Machine Learning



Modelo contemporáneo de reconocimiento de características y patrones (post-2012):

► Deep Learning

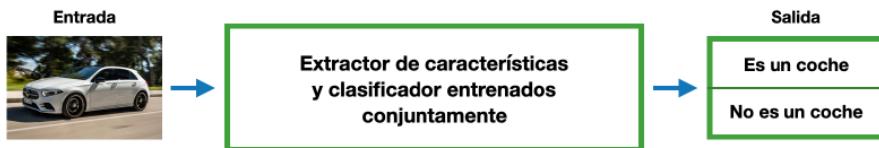


Figura 2.1: Diferencia entre el ciclo de un modelo tradicional de machine learning y uno de deep learning. Imagen extraída de [11]

Los modelos de ML clásicos tienen tradicionalmente dos partes bien diferenciadas (Figura 2.1, imagen superior): la extracción de características y el modelo de aprendizaje que procesa esas características (por ejemplo, un clasificador). Como evolución de este proceso, surgió el *deep learning*, que busca eliminar la diferenciación entre ambas fases y realizar una aproximación *end-to-end*, es decir, que un sólo modelo realice la extracción de características y el procesamiento (Figura 2.1, imagen inferior).

2.2. Redes neuronales

La unidad principal del deep learning es la red neuronal, que será la pieza central de este trabajo fin de grado.

Una red neuronal artificial es una serie de nodos llamados neuronas, usualmente agrupadas en estructuras llamadas capas que se conectan entre sí, imitando las redes neuronales biológicas. Uno de los tipos más sencillos y populares de red neuronal es el MLP, que ya presentamos en la introducción. Está formado por varias capas de neuronas, donde cada neurona recibe una señal de la suma de las salidas de la capa anterior y aplica una función no lineal al resultado. En la figura 2.2 se puede observar su formulación matemática visualmente.

Una pieza fundamental en los modelos de ML es la función de pérdida

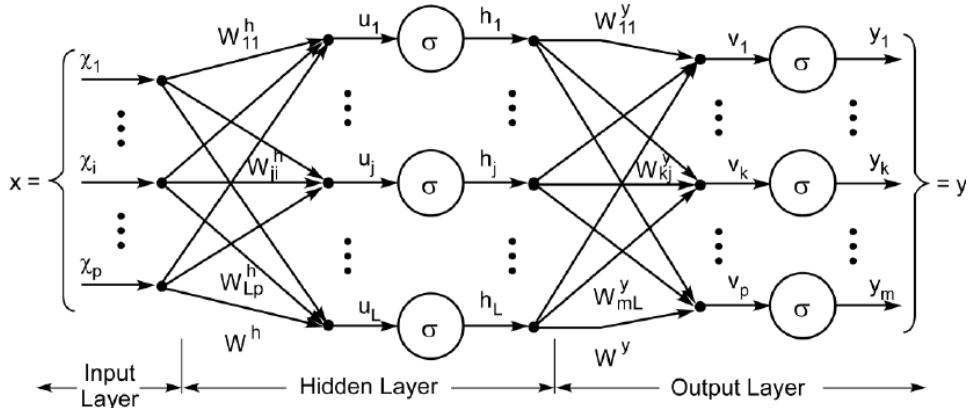


Figura 2.2: Formulación matemática de un MLP. Imagen extraída de [17].

o *loss*. Esta es una función que mide el rendimiento de un modelo y será la función que se optimizará durante el aprendizaje. Esta función dependerá del problema a resolver. En regresión, se suele usar el error cuadrático medio y en clasificación, depende de si queremos discriminar sólo entre dos clases, donde usaremos *binary crossentropy* o entre más clases, donde utilizaremos *categorical crossentropy*. En nuestro caso, nos enfrentamos a un problema de clasificación binaria, por lo que utilizaremos la función *binary crossentropy*, como se explicará en la sección 4.3.

Los nodos o neuronas disponen de ciertos parámetros (denominados pesos) que son aprendidos en la fase de entrenamiento. Debido a esto, la red es capaz de aprender a realizar una tarea a partir de la experiencia.

Como algoritmo de aprendizaje, el más común es el descenso del gradiente estocástico. Esta técnica consiste en dividir los datos de entrenamiento en conjuntos pequeños o *batches* y, de forma iterativa, calcular el gradiente de cada uno de ellos y utilizarlo para actualizar los pesos. Una vez que el modelo ha utilizado todos los *batches*, se considera que ha entrenado durante una época o *epoch*. Este proceso se extiende sobre un número predeterminado de épocas.

Para entrenar estos pesos, el método más utilizado es el gradiente descendente calculado mediante *backpropagation* [25], un algoritmo que computa los gradientes de la función de pérdida con respecto a cada peso de forma eficiente. El algoritmo utiliza la regla de la cadena, iterando desde la última capa hasta la primera para evitar cálculos redundantes.

Las Redes Convolucionales Profundas (ConvNet) (véase figura 2.3) son una evolución de los MLP especializados en *inputs* con forma de *grid* como, por ejemplo, las imágenes. Su principal innovación consiste en la introducción de las capas convolucionales, que se explican más adelante en la sección 2.2.2, como extractores de características. Estas capas permiten buscar pa-

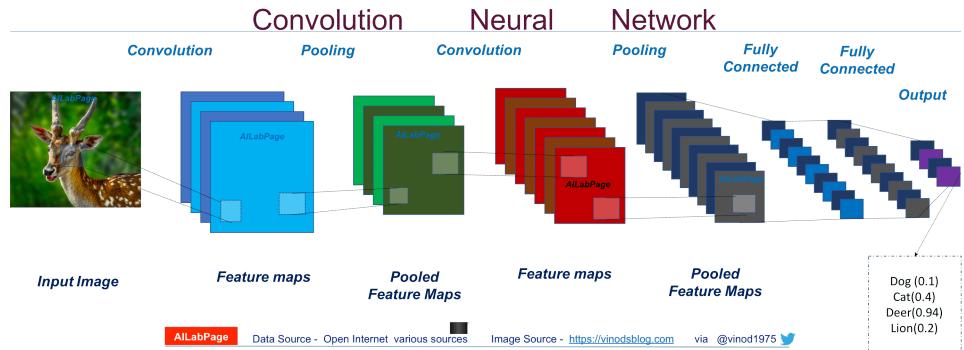


Figura 2.3: Arquitectura de una red neuronal convolucional. Imagen extraída de [26].

trones en las imágenes de modo jerárquico, buscando formas sencillas como líneas o esquinas en las primeras capas e información de más alto nivel (formas, rasgos, caras) en las siguientes (véase figura 2.4). Hoy en día, las redes convolucionales son el estado del arte en el campo de la visión por computador (VC)¹.

A continuación, explicaremos algunas de las capas más utilizadas.

2.2.1. Capa totalmente conectada

Las capas totalmente conectadas, *fully-connected* o *dense* fueron las primeras en ser desarrolladas y son utilizadas en todas las capas ocultas de MLP. En las redes convolucionales suelen utilizarse algunas capas fully-connected después de las capas de convolución. Estas capas toman las entradas de la capa anterior, multiplicarán esos valores por sus respectivos pesos y los sumará. A este resultado se le aplica una operación no lineal como la sigmoide, la tanh o ReLU. Como hiperparámetros, estas capas requieren el número de nodos y la función de activación.

En clasificación, la última capa es una totalmente conectada con una activación sigmoide o *softmax* (dependiendo de si es clasificación binaria o multiclase respectivamente), que dará una salida en forma de porcentaje. Este valor representará la seguridad con la que la red cree que ese input pertenece a una clase concreta.

¹Visión por computador: disciplina que busca analizar y procesar información a partir de información visual, imágenes o videos [18].

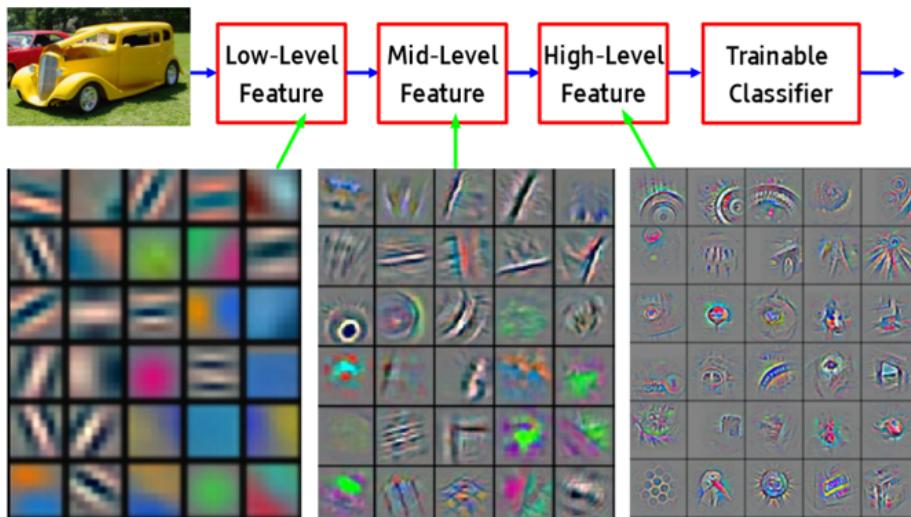


Figura 2.4: Extracción de información jerárquica de una ConvNet. Se extraen características de bajo nivel en las primeras capas (líneas, esquinas) y de alto nivel (formas complejas, patrones) en las últimas. Imagen extraída de [19].

2.2.2. Capa Convolucional

Estas capas están formadas por una serie de filtros que aplican la operación de convolución a cada píxel de la imagen. Cada filtro de convolución consiste en una matriz (usualmente de tamaño impar: 3x3, 5x5 o 7x7) que se desliza sobre la imagen colocando el elemento central de la matriz en cada píxel de la imagen. En cada posición se multiplica el valor del filtro por el del píxel sobre el que se encuentra y se suman los resultados. A este resultado se le suele aplicar una operación no lineal de forma análoga a las capas fully-connected. En la salida, se coloca en la posición del píxel central, el resultado de la operación. Un ejemplo se puede observar en la figura 2.5.

Los hiperparámetros más importantes de estas capas son:

- **Profundidad:** el número de filtros que se utilizarán.
- **Activación:** la función de activación no lineal utilizada.
- **Stride:** número de píxeles que se desplaza el filtro en cada deslizamiento. Si se establece mayor que 1, la imagen de salida tendrá dimensiones más pequeñas que la entrada.
- **Padding:** técnica de relleno de los bordes (para poder aplicar el filtro a los píxeles de los bordes).

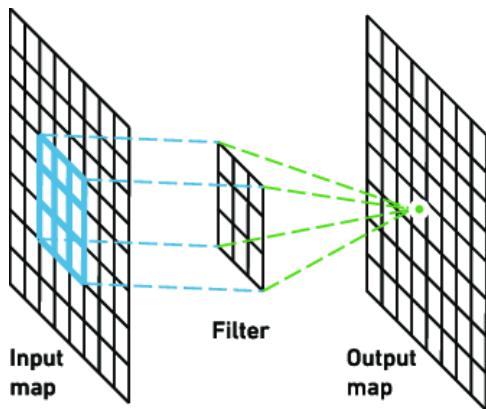


Figura 2.5: Muestra visual de cómo funciona una capa convolucional. El filtro (3x3) procesa una sección de la imagen y la salida pasa a ser el valor de la posición correspondiente al píxel central de la sección. Imagen extraída de [20].

2.2.3. Capa de Pooling

Capa que reduce el tamaño del input. Consiste en agrupar píxeles cercanos (usualmente agrupaciones de 2x2) y reducirlos a un único píxel mediante max pooling (mantener el valor máximo) o average pooling (la media de los valores). Aparte de la elección entre max o average, el hiperparámetro más relevante para esta capa es el tamaño de las agrupaciones. En la Figura 2.6 se puede observar un ejemplo visual de la funcionalidad ofrecida por las capas de *pooling* en el caso más común: una capa de *pooling* 2x2 con *strides* de 2.

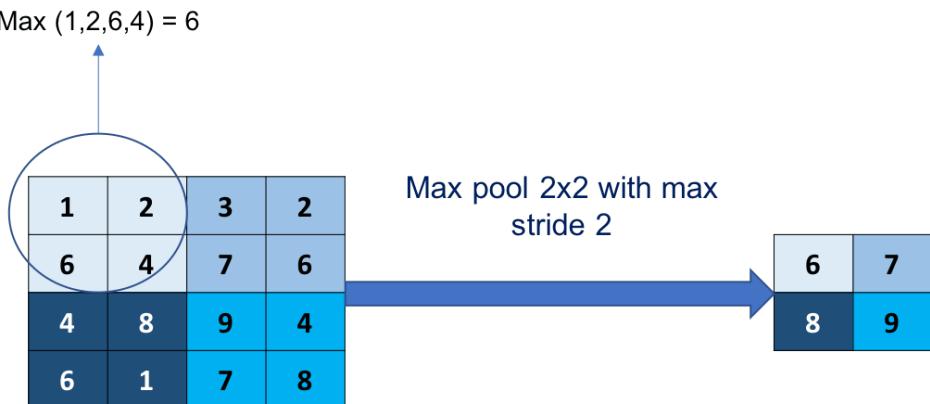


Figura 2.6: Ejemplo de funcionamiento de una capa de pooling 2x2 con un *stride* de 2. Imagen extraída de [26].

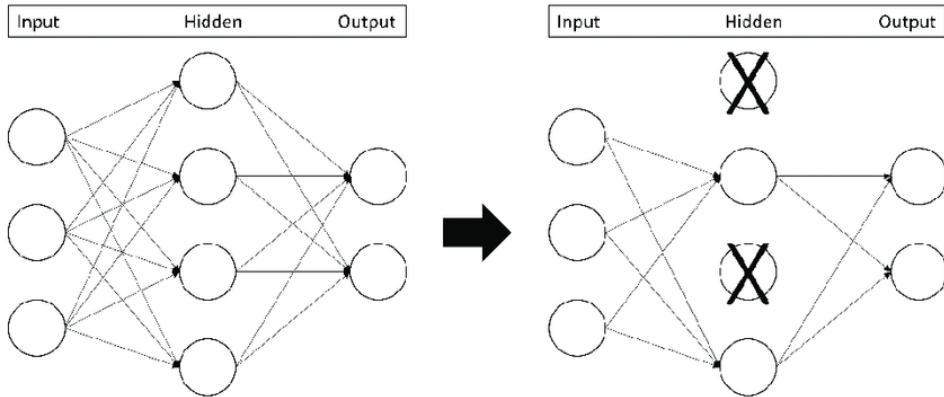


Figura 2.7: Capa de dropout aplicada sobre la segunda capa totalmente conectada de una pequeña red. Imagen extraída de [24].

2.2.4. Dropout

Capa que desactiva un porcentaje de las entradas que se le proporcionan. Se utiliza como una forma de regularización para evitar *overfitting*. Es una técnica muy eficiente, pues al evitar que las capas reciban toda la información, impide que todas las decisiones sean tomadas por unos pocos nodos y favorece que los pesos se distribuyan más equitativamente [23]. Un ejemplo de una capa (totalmente conectada) a la que se le aplica dropout se puede observar en la Figura 2.7. Esta capa sólo se activa en el entrenamiento. Al utilizar la red ya entrenada se utiliza toda la información disponible. Esta capa requiere como hiperparámetro el porcentaje de neuronas que son desactivadas.

Las redes están normalmente formadas por una primera capa de input, a la que no se le aplica ninguna función no lineal y una capa de output, que es la última capa y devuelve el resultado de la tarea de clasificación o regresión. Las capas que se encuentran entre estas dos se denominan capas ocultas. El número de estas capas, su tipo de operación y sus hiperparámetros es lo que denominamos la arquitectura de la red, y será objeto de estudio en este trabajo.

2.3. Neural Architecture Search

Como se ha visto, el diseño de la arquitectura conlleva muchas decisiones y mucho ajuste de hiperparámetros. Para automatizar este proceso nació a finales de los años 80 [54, 55, 48] el campo de la Neural Architecture Search (NAS) [8]. Este campo consiste en técnicas de búsqueda automática

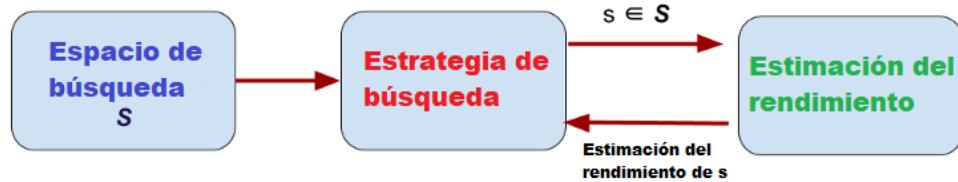


Figura 2.8: Esquema de cómo interactúan las tres dimensiones de NAS. Imagen extraída de [27] y traducida.

de arquitecturas de redes neuronales.

Tal y como se explica en los principales *surveys* [8, 9, 27], las técnicas de NAS pueden analizarse y catalogarse basándose en de qué forma se enfrentan y modelan los tres aspectos siguientes:

- **Espacio de búsqueda**
- **Estrategia de búsqueda**
- **Estrategia de estimación del rendimiento**

En la Figura 2.8 se puede ver cómo interactúan esas tres dimensiones: el espacio de búsqueda determina qué redes son posibles generar, la estrategia de búsqueda selecciona la siguiente arquitectura para generar y se obtiene una estimación del rendimiento de la red utilizando la estrategia elegida a tal fin. A continuación, caracterizaremos algunas de las alternativas propuestas para cada una de las tres dimensiones.

2.3.1. Espacio de búsqueda

El espacio de búsqueda se compone de todas las arquitecturas que puedan ser representadas. Vendrá determinado por el tipo de representación utilizada, los valores posibles para el número de capas y neuronas y por los tipos de capas permitidas.

Es el aspecto más importante de la búsqueda de arquitecturas neuronales, pues determina el tipo y la variedad posible de arquitecturas generables. Si se usa un espacio demasiado reducido, la limitada variedad de posibles redes no permitirá obtener buenos resultados. Sin embargo, si el espacio es demasiado amplio, la búsqueda puede resultar inasumible en términos computacionales. Por el otro lado, si el espacio de búsqueda es de un tamaño

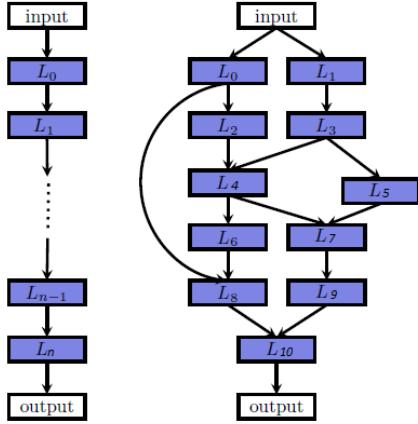


Figura 2.9: A la izquierda una red de tipo cadena, cada capa sigue estrechamente a la anterior. A la derecha, una red añadiendo la posibilidad de escoger la capa anterior a la que se conecta cada capa. Imagen extraída de [8].

apropiado, cuanto más reducido sea, menor impacto tendrá la estrategia de búsqueda escogida e incluso selectores aleatorios (Random Forest) podrán dar buenos resultados [9], como se puede observar en los experimentos de [40, 39, 35].

Para la generación de arquitecturas han surgido en la literatura dos alternativas principales, que suelen aparecer hibridadas, la macro-arquitectura, o espacio global, y la micro-arquitectura o búsqueda de celdas.

Macro-Arquitectura

La técnica de Macro-Arquitectura [40] fue la primera en ser utilizada al nacer el campo del NAS en los años 80 [54, 55, 48] y se basa en considerar la red como una secuencia de capas que se generan de forma secuencial. La estructura más simple a este respecto es considerar una red como una cadena de capas, donde el output de una capa se conecta directamente con el input de la siguiente (véase Figura 2.9). Mediante la macro-arquitectura, se genera la red capa a capa, determinando en cada una el tipo de operación y el resto de parámetros necesarios. Por tanto, el espacio de búsqueda se ve parametrizado por el número n de capas, las operaciones o tipos de capas permitidos (pooling, convoluciones, etc) y los hiperparámetros asociados a cada una de estas opciones, como podría ser el número de neuronas, el tamaño del kernel en una convolución o el factor de pooling.

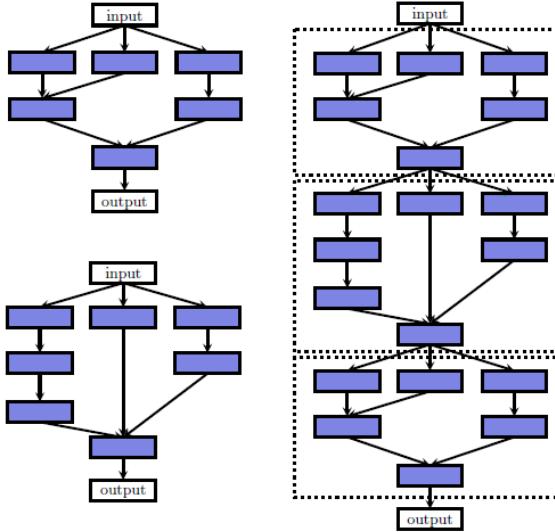


Figura 2.10: La micro-arquitectura optimiza celdas como las de la izquierda y luego se combinan para generar la red completa (derecha). Imagen de [8].

Las técnicas más modernas [47] de macro-arquitectura también permiten que las capas escojan a que capas anteriores se conectan para incorporar *skip connections*². En estos modelos, las capas no se conectan entre sí como una cadena, sino que pueden conectarse a otras capas anteriores, generando modelos más complejos como las famosas ResNet [21] o DenseNet [22]. En este caso, el espacio de búsqueda se expande aún más con la selección de la capa o capas anteriores a la que se conecta cada una de las capas. Este enfoque es el más sencillo e intuitivo y permite generar una gran variedad de redes, pero el espacio de búsqueda es muy amplio y los modelos pueden tardar mucho en ser entrenados.

Micro-Arquitectura

La Micro-Arquitectura [40] es una técnica motivada por el hecho de que muchas redes creadas a mano se construyen repitiendo un cierto patrón de capas un número determinado de veces. Imitando esa idea, se puede generar un bloque básico a partir de unas pocas capas y limitarse a repetir n veces esa estructura para generar la red final.

Esta técnica fue presentada por primera vez en 2017 [39]. Se propone la optimización de dos tipos de celda: una celda normal que tiene como

²Se considera una *skip connections* cuando una capa no se encuentra unida solamente a la capa inmediatamente anterior, sino que también toma el output de alguna otra capa.

particularidad que la dimensionalidad de su input debe ser igual que la de su output y una celda de reducción que, como su nombre indica, reduce la dimensión espacial del input. Estos bloques se generan siguiendo la técnica de búsqueda escogida (las técnicas de búsqueda se explican en el siguiente apartado, 2.3.2) y una vez que se han obtenido los bloques que se pretende usar, es posible generar una red completa organizándolos en una estructura predefinida. Este proceso se puede observar visualmente en la figura 2.10. La principal ventaja de este sistema es que el espacio de búsqueda se ve drásticamente reducido, lo cual redundará en una mayor eficiencia a la hora de buscar la arquitectura. Además, la red final es más generalizable, pues se podrá adaptar a otros problemas de forma sencilla cambiando el número de bloques de la red.

Este modelo ha tenido mucho éxito desde su publicación, pues el primer trabajo en utilizar micro-arquitectura [39] redujo enormemente el tiempo utilizado por su modelo anterior [40], además de mejorar sus resultados. No solo eso, sino que quedó demostrada la capacidad de generalización al utilizar bloques generados con CIFAR-10 en el dataset ImageNet y conseguir resultados del estado del arte. No obstante, esta técnica sufre de una gran desventaja y es que no soluciona el problema de escoger cuántas celdas usar, de qué tipo y de qué forma conectarlas. En principio, esta es una decisión que la debe tomar el diseñador, por lo cual, no estamos automatizando en realidad el diseño completo de la red. Para solucionar este problema, la mayoría de propuestas actuales se basan en aunar ambas técnicas, con una optimización de celdas de micro-arquitectura para su posterior organización con macro-arquitectura. Por ejemplo, [30] propone una búsqueda jerárquica a distintos niveles. En la figura 2.11 se puede ver un ejemplo de esta jerarquía: con capas simples (convoluciones y pooling) optimizan un bloque (operación superior). A continuación, pueden usarse esos bloques simples para construir bloques más complejos (operación inferior).

2.3.2. Estrategia de búsqueda

El espacio de búsqueda suele ser no continuo y altamente dimensional, por lo que encontrar la estrategia óptima de búsqueda no es un problema trivial. A continuación se explican algunos de las alternativas para realizar la búsqueda.

Algoritmos evolutivos

La estrategia más utilizada históricamente han sido los algoritmos evolutivos. Este enfoque se utiliza desde finales de los años 80 [48], y se puede encontrar una amplia compilación de su uso antes del 2000 en [32].

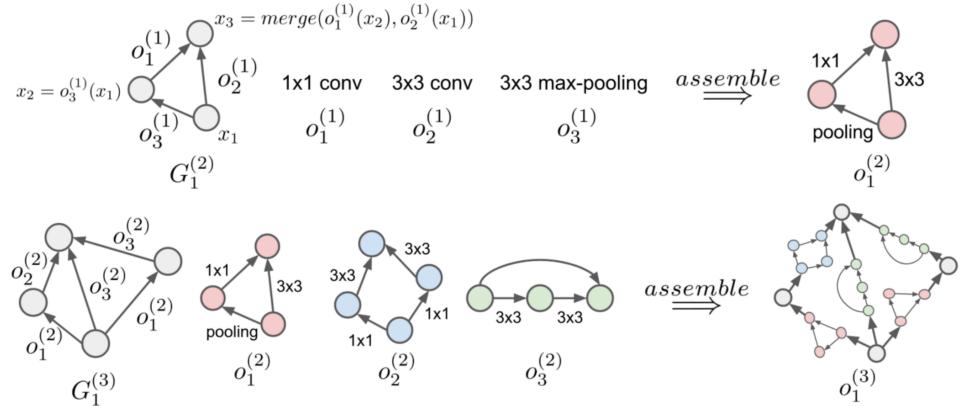


Figura 2.11: Ejemplo de arquitectura jerárquica. Capas simples son usadas para construir bloques, que a su vez pueden usarse para crear otros bloques más complejos. Imagen extraída de [30].

Los primeros usos de los algoritmos evolutivos no se centraban sólo en la arquitectura, sino que optimizaban al mismo tiempo la arquitectura y los pesos [34]. Sin embargo, posteriormente se ha ido abandonando esa idea al hacerse patente que la optimización con *backpropagation* y *stochastic gradient descent* es mucho más eficiente [8].

Las estrategias evolutivas se basan en mantener una población de individuos compuestos por genes. Aplicados a la búsqueda de arquitecturas, los genes suelen representar tipos de capas, parámetros y conexiones. Un algoritmo evolutivo sigue un ciclo medido en generaciones (véase Figura 2.12). Se parte de una población de redes neuronales con distintas arquitecturas y se comienza por evaluarlas (utilizando alguna estrategia de estimación del rendimiento de la sección 2.3.3). A partir de los rendimientos de cada individuo, se realiza un proceso de selección de un subgrupo de la población, que actuarán como progenitores para la siguiente generación. A continuación, al grupo seleccionado se le aplican los operadores de cruce³ y mutación⁴.

Este ciclo y los operadores utilizados son variables y cada técnica propuesta utiliza sus propias variantes. Sin embargo, el cruce suele ser una operación complicada, por lo que la mayoría de modelos [34, 37, 49] prefieren aplicar solo mutaciones. Los métodos de sólo mutación tienen otra gran ventaja y es que permiten conservar los pesos de las redes de una generación a otra, ahorrando gran parte del tiempo de entrenamiento.

Los algoritmos evolutivos son fácilmente paralelizables y su rendimiento

³El operador de cruce combina los genes de dos o más padres para producir un nuevo individuo.

⁴El operador de mutación altera los genes de un individuo para generar un descendiente no idéntico

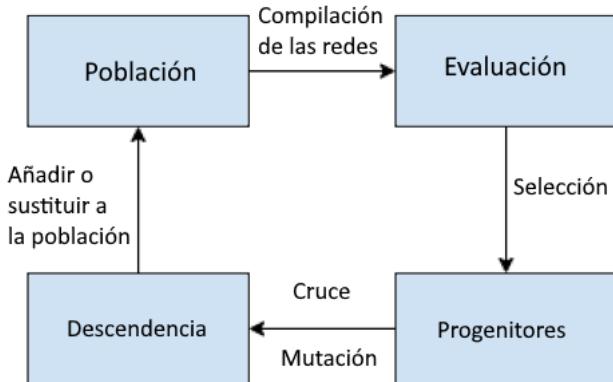


Figura 2.12: Esquema del ciclo de un algoritmo evolutivo aplicado a NAS.
Imagen extraída de [9]

es bastante bueno. Por ello, hoy en día, las aproximaciones evolutivas siguen siendo muy populares [34, 37].

Aprendizaje por Refuerzo

El aprendizaje por refuerzo (AR) aplicado a NAS se basa en el uso de otra red neuronal denominada controlador cuyo objetivo es tomar decisiones acerca de la generación de la arquitectura una red hija mediante aprendizaje por refuerzo (véase Figura 2.13). El aprendizaje por refuerzo (cómo se comentó en la sección 2.1) es un tipo de aprendizaje que utiliza datos no etiquetados y que se basa en una señal de recompensa que indica al modelo el resultado de su acción.

En el caso de AR aplicado a NAS, la recompensa de la red controladora está ligada al rendimiento estimado de la red hija. La red controladora es normalmente una Recurrent Neural Network (RNN) que predice las capas, el tipo de estas y los parámetros de cada una de ellas (véase Figura 2.14). Para ello, el modelo básico consiste en que la RNN proporciona como salidas los hiperparámetros de las capas de la red hija. A continuación, esa información se suministra como entrada a la propia red y se repite el proceso. Una vez se genera una red hija completa, se estima su rendimiento y se suministra como recompensa al controlador.

Esta estrategia ha sido muy utilizada en el campo del NAS, tanto en macro-architecture [40] como en micro-architecture [39] con buenos resultados.

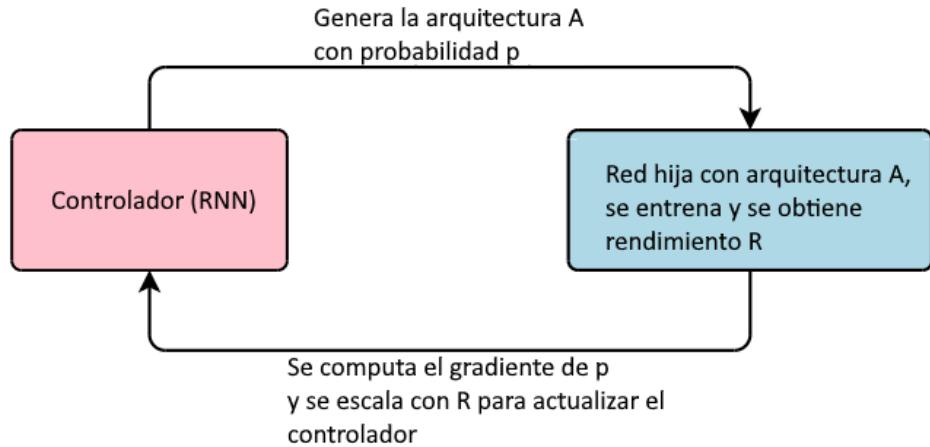


Figura 2.13: Esquema del funcionamiento de la estrategia que emplea aprendizaje por refuerzo. Una red controladora genera redes hijas. Imagen extraída de [27] y traducida al español.

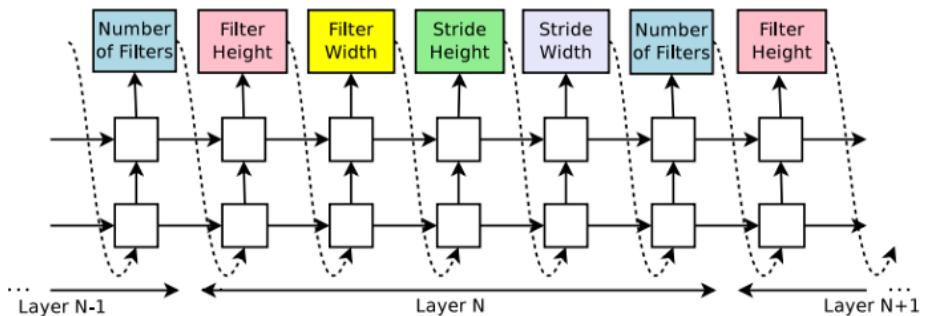


Figura 2.14: En este ejemplo, el controlador produce múltiples salidas que representan cada uno de los parámetros que debe escoger para cada capa. Imagen extraída de [27].

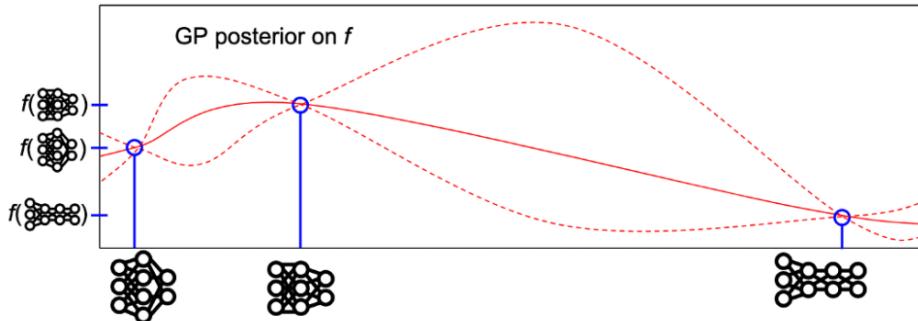


Figura 2.15: En este ejemplo se puede observar cómo a partir de los resultados de las evaluaciones de tres redes y la función distancia se puede estimar la función original (línea sólida roja) y un intervalo de confianza (líneas de puntos rojos). Imagen extraída de [46].

Optimización Bayesiana

La optimización Bayesiana [44, 45] ha sido tradicionalmente utilizada para la optimización de hiperparámetros pero ha destacado en los últimos tiempos como una alternativa muy útil para ser utilizada con NAS.

La optimización Bayesiana es un método de optimización de orden 0, es decir, que no necesita la derivada de la función que estamos optimizando (el gradiente descendente es una técnica de primer orden, por ejemplo). La idea detrás de esta optimización es utilizar una “función distancia” que representa la distancia entre dos puntos. Los siguientes puntos a explorar se escogen basándose en los resultados anteriores y en la distancia de los nuevos puntos a ellos. La idea detrás de esto es que los puntos cercanos a puntos que hayan dado mejores resultados tendrán más probabilidades de dar a su vez buenos resultados. Una buena optimización bayesiana mantendrá un equilibrio entre la exploración de nuevos entornos y la explotación de los mejores puntos. En la optimización Bayesiana aplicada a NAS, se utiliza como función objetivo el rendimiento de la red generada y como argumentos de esta función, la arquitectura de la red.

Para comparar la distancia entre dos arquitecturas distintas, se han propuesto funciones que comparan la similitud entre los hiperparámetros [28]. Con esta función, se pueden utilizar los métodos clásicos de Gaussian Process-based Bayesian Optimization, utilizando la función de distancia como función kernel. Estos métodos suponen que la distribución del fitness a partir de la distancia sigue una distribución gaussiana. Con ello, se puede estimar la mejora esperada de cada punto (véase Figura 2.15).

2.3.3. Estrategia de estimación del rendimiento

La estrategia de estimación del rendimiento es aquella heurística que nos permite estimar el rendimiento de una red sin necesidad de entrenarla por completo. Debido a que en todas las técnicas de búsqueda comentadas será necesario generar multitud de redes para elegir la mejor entre ellas, es inasumible entrenar (y testear) cada red generada de forma completa. Para ello, existen varias técnicas para ahorrar ese proceso.

Estimaciones de baja fidelidad

Es el método más sencillo y más utilizado en la literatura. Consiste en simplemente entrenar cada red con menos datos, menos tiempo, modelos reducidos o tamaños de datos reducidos. Así, el tiempo de entrenamiento se ve drásticamente reducido.

Con esas limitaciones, los resultados obtenidos serán inferiores a los reales, pero se espera poder comparar los resultados entre redes de forma adecuada. Sin embargo, es una técnica que introduce sesgos subestimando los resultados y que puede ser problemático cuando las diferencias de resultados entre las estimaciones de baja fidelidad y las de entrenamiento completo son demasiado grandes.

Extrapolación de la curva de aprendizaje

Con esta propuesta, se entrena cada red sólo unas pocas épocas y, a continuación, se estima el resultado del entrenamiento completo a partir de la curva de aprendizaje. Esta estimación se puede hacer a partir de otro modelo de ML que esté entrenado para estimar la curva de aprendizaje a partir de sólo unas pocas épocas, técnica propuesta por [30].

Morfismo de redes

Otra posibilidad es permitir que las redes hereden los pesos de redes anteriores que tengan una morfología similar. Esta técnica, similar a cuando se parte de redes pre-entrenadas, permite que el tiempo de entrenamiento se reduzca. Este método ha sido muy utilizado en algoritmos que se basan en crear redes a base de incrementar su tamaño. Sin embargo, estas técnicas provocan que se tienda a arquitecturas demasiado complejas. Enfoques más modernos también permiten que las redes disminuyan su tamaño.

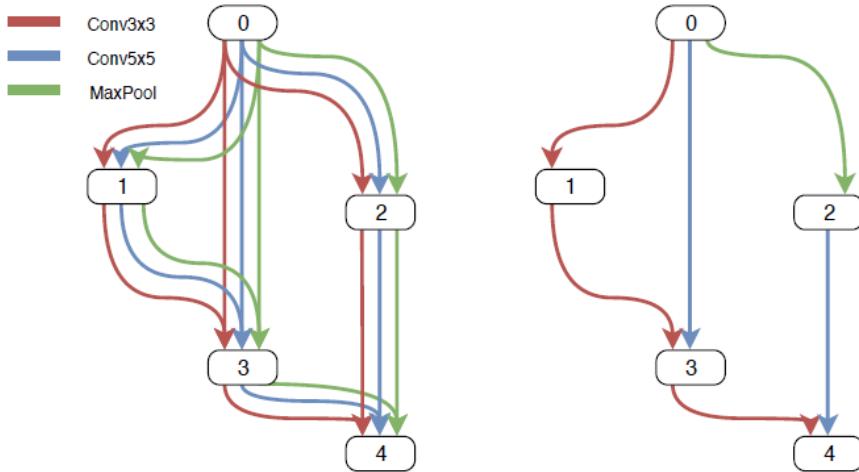


Figura 2.16: Ilustración de una arquitectura one-shot. El modelo completo (izquierda) contiene todas los pesos guardados de todas las redes entrenadas. La arquitectura de estas redes es un subgrafo del modelo completo (derecha) y, por tanto, puede utilizar los pesos de este. Imagen extraída de [8].

Modelos One-shot

Trata todas las redes del espacio de búsqueda como subgrafos de un supergrafo. Los pesos del supergrafo son únicos y, por tanto, todas las redes comparten pesos en aquellos nodos que comparten (Figura 2.16). Con esta técnica, sólo el supergrafo necesita ser entrenado y todas las redes generadas pueden ser evaluadas directamente como subgrafos del supergrafo. Esta técnica también introduce sesgos de subestimar los resultados de las arquitecturas, pero puede ser suficiente para comparar rendimientos entre sí.

Capítulo 3

Estado del arte

3.1. Resumen estado del arte

El NAS ha crecido mucho recientemente, como se puede observar en la Figura 3.1. A partir de, aproximadamente, el 2016, este campo ha sufrido una gran popularidad y sólo durante el año 2020 hubo 853 artículos académicos relacionados. Por ello, para la realización de este TFG, se ha realizado un estudio pormenorizado del estado actual de la literatura. Se comenzó con un estudio de las principales compilaciones [8, 9, 27] sobre el tema para, a continuación, estudiar los artículos más relevantes de las distintas técnicas de las que se ha hablado.

En la Tabla 3.1 se muestra un resumen de las referencias más importantes y las técnicas utilizadas en cada una de las tres dimensiones discutidas anteriormente: el espacio de búsqueda, la estrategia de búsqueda y la estrategia de estimación de resultados. Las abreviaturas empleadas se describen en la propia leyenda de la tabla. Para una explicación de estos conceptos, se pueden consultar sus respectivas secciones del capítulo 2, Fundamentos Teóricos.

3.2. Evolución de NAS

A continuación, se hará un repaso del estado del campo del NAS, desde una visión histórica hasta los enfoques más actuales.

3.2.1. Aproximaciones evolutivas

Las primeras publicaciones de las que se tiene constancia que aplicasen técnicas de búsqueda y optimización para las arquitecturas de redes fueron

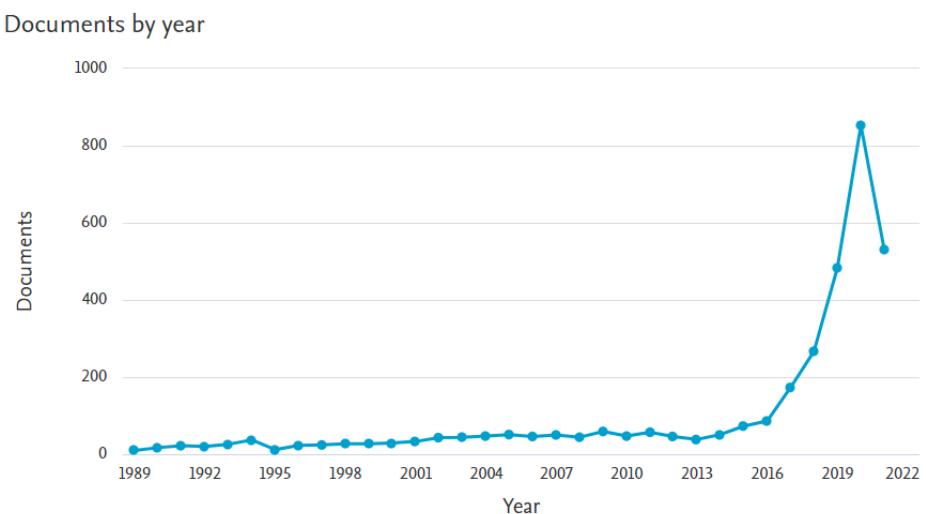


Figura 3.1: Número de publicaciones por año dedicadas al estudio de NAS. Se puede notar cómo a partir de 2016 empezó a despertar un interés creciente. El año pasado, por ejemplo, se presentaron 853 artículos dedicados a este campo. Este gráfico ha sido creado con la herramienta [Scopus](#) el día 2 de septiembre de 2021 utilizando la query (*TITLE-ABS-KEY (neural AND architecture AND search)) AND (LIMIT-TO (SUBJAREA , "COMP") OR LIMIT-TO (SUBJAREA , "ENGI") OR LIMIT-TO (SUBJAREA , "MATH")*)

Año	Paper	Espacio de búsqueda	Estrategia de búsqueda	Estrategia Estimación
1989	Miller et al. [48]	Global	AE	BF
1997	Moriarty et al. [49]	Global	AE	BF
2002	Stanley et al. [42]	Global	AE	EP
2014	swersky et al. [41]	Global	OB	CA
2017	Elsken et al. [51]	Global	AE	BF, WS
2017	Zoph et al. [39]	Global	AR	BF
2018	Cai et al. [52]	Global	AR	BF
2018	Pham et al. [33]	Global, Micro	AR	BF, WS
2018	Zoph et al. [40]	Micro	AR	BF
2018	Kandasamy et al. [28]	Global	OB	BF
2018	Zhong et al. [47]	Micro	AR	BF
2018	Liu et al. [29]	Micro	OB	BF
2018	Liu et al. [30]	Jerárquico	AE	BF
2019	Assunç et al. [53]	Global	AE	BF
2019	Bender [50]	Micro	RS	BF, OS
2019	Cui et al. [38]	Micro	BB	MR
2019	Jin et al. [43]	Global	OB	MR
2019	Stanley et al. [34]	Micro	AE	EP
2020	Suganuma et al. [37]	Global	AE	BF
2020	Sun et al. [66]	Global	AE	BF, CH

Cuadro 3.1: Resumen de la literatura más relevante sobre NAS ordenada de modo ascendente según año de publicación. Leyenda: AE: Algoritmo Evolutivo; AR: Aprendizaje por Refuerzo; OB: Optimización Bayesiana; BB: Búsqueda Bi-nivel; RS: *Random Search*; BF: Estimaciones de Baja Fidelidad; CA: Curva de Aprendizaje; MR: Morfismo de redes; WS: *Weight Sharing*; OS: Modelos *One-Shot*; EP: Evolución de Pesos. CH: Caché.

publicadas en 1988-1989 [54, 55, 48] y utilizaban algoritmos genéticos. Esta fue la aproximación más habitual en los inicios. Xin Yao realizó una extensa recopilación de los trabajos evolutivos de antes del año 2000 en [32]. Esta estrategia ha seguido siendo muy popular, destacando, entre otros los modelos NEAT [42, 72] (*NeuroEvolution of Augmenting Topologies*), algoritmo ampliamente utilizado en el campo de la IA dentro de los videojuegos.

Una peculiaridad de este método, es que el enfoque evolutivo puede usarse, no sólo para diseñar redes, sino que puede usarse también para generar los pesos de esas redes. Es decir, que las redes generadas no se entrenaban con el usual SGD con *backpropagation*, sino que evolucionaban los pesos con un algoritmo evolutivo [56, 57, 58, 59]. Esta aproximación ha sido dejada de lado en los trabajos más modernos, en gran parte porque la eficiencia de *backpropagation* es difícilmente superable por otros enfoques y resulta más conveniente entrenar las redes generadas con SGD y *backpropagation*, con independencia de la técnica de NAS usada para generar la red. Aún así, hay excepciones, y algunos autores modernos siguen experimentando con la idea de evolucionar pesos [31]. Sin embargo, la mayoría de los enfoques contemporáneos se basan en utilizar algoritmos evolutivos para optimizar la arquitectura y utilizan *backpropagation* para el entrenamiento.

Normalmente se suelen utilizar algoritmos con sólo mutaciones debido a la dificultad de diseñar un buen operador de cruce por las complicaciones derivadas de la posibilidad de generar redes de longitud variable. En la actualidad, los algoritmos evolutivos siguen siendo muy populares, aunque otras técnicas (el aprendizaje por refuerzo y la optimización Bayesiana), han mejorado los resultados y están siendo ampliamente explorados.

Entre los modelos más recientes, podemos destacar la Programación Genética Cartesiana [37], o Auto CNN [66], un algoritmo que sí introduce operador de cruce del que hablaremos en profundidad en la sección 4.7.

3.2.2. Aprendizaje por refuerzo

El término NAS fue creado en 2017 en una publicación de Zoph et al. [39]. En este trabajo, se utilizó por primera vez AR, en vez de los algoritmos evolutivos que habían dominado hasta entonces. Llamaron a su arquitectura, NAS, término que se ha acabado utilizando para referirse a todo el campo.

Las arquitecturas que formaban el espacio de búsqueda tenían sólo capas convolucionales pero se escogía el número de filtros, el tamaño de la ventana y el stride. Además, permitía la incorporación de *skip connections*. Se utilizaba una búsqueda global o macro-arquitectura, al optimizar la red capa a capa.

Los resultados fueron muy buenos en las bases de datos de CIFAR-10

y Penn Treebank¹, pero tenía una gran desventaja: el modelo propuesto es extremadamente lento: utilizando 800 GPUs, se tardó entre 21 y 28 días para su entrenamiento [33].

Para aliviar esta gran carga computacional, el mismo equipo de investigación desarrolló la idea de la micro-arquitectura (véase sección 2.3.1) en su modelo NASNet [40]. Se probó de nuevo con CIFAR-10 y se redujo siete veces el tiempo utilizado por el modelo anterior [39]. Aún así, el tiempo necesario para generar la red fue de 4 días entrenando con 450 GPUs [33]. Los resultados fueron el estado del arte al mejorar el rendimiento de redes clásicas como DenseNet.

A partir de este trabajo, la idea de la micro-aquitectura ha sido ampliamente explorada, pues permite obtener muy buenos resultados con un coste computacional muy inferior al espacio global. Los esfuerzos se centraron en reducir todavía más el tiempo necesario.

Entre las arquitecturas propuestas, destaca ENAS [33]. Su principal contribución es la aplicación de *Weight Sharing*, es decir, que los pesos de las redes generadas son compartidos. Esto significa que cuando la red genera una red que utiliza alguna capa o bloque de capas similares a alguna capa previamente entrenada, se reutilizan esos pesos, evitando reentrenar la red completa. Esto reduplica en que los tiempos necesarios sean mucho menores. Calculan que esta mejora aumenta la velocidad en 1000x en términos de horas de GPU. Frente a los 4 días del modelo NASNet en 450 GPUs, ENAS tardó menos de medio día en entrenarse con sólo una GPU.

3.2.3. Enfoque jerárquico

Actualmente, uno de los principales frentes abiertos es cómo lidiar con los puntos débiles de la micro-arquitectura, especialmente con el hecho de que, en el modelo original, es necesario establecer con anterioridad la organización de las celdas.

Una solución pasa por el enfoque jerárquico, es decir, aplicar micro-arquitectura para generar las celdas y después utilizar macro-arquitectura para elegir cómo se disponen las celdas [30].

Otra idea al respecto del espacio de búsqueda es no limitar la micro-arquitectura a crear uno o dos tipos de celdas, sino permitir que cada celda sea distinta. Cui et al [38] presenta una optimización en dos niveles que se propone ese objetivo, optimizar cada celda por separado, una por una, para permitir arquitecturas diversas. De hecho, también hicieron varias pruebas, demostrando que es más beneficioso tener celdas diversas en vez de sólo un

¹Base de datos de procesamiento del lenguaje natural. Incluye frases extraídas del Wall Street Journal para análisis sintáctico.

bloque que se repite.

3.2.4. Optimización bayesiana

En el estado del arte también ha cogido fuerza la estrategia de la optimización bayesiana (véase sección 2.3.2). El representante más importante de esta estrategia es el modelo Auto-Keras [43]. Su contribución es aunar la optimización bayesiana y el morfismo de redes. Utilizando morfismo de redes se reduce enormemente los tiempos, pues al tener sólo una red y modificarla durante el proceso, puede entrenarse solo con un par de épocas. Para realizar la búsqueda utilizan optimización bayesiana que selecciona una operación cada vez. Las elecciones están establecidas como un árbol en que cada arista es un cambio posible a la red. La mayoría de los modelos que utilizan optimización bayesiana lo hacen en combinación con una estrategia de macro-arquitectura [28, 43], pero también hay algunos trabajos que han optado por la micro-arquitectura [29]. La optimización bayesiana da muy buenos resultados en un tiempo mucho menor que el aprendizaje por refuerzo, por tanto, es una de las alternativas más prometedoras para el NAS.

3.2.5. NAS en la actualidad

El campo del NAS ahora se centra en intentar reducir el coste computacional necesario para sus distintos modelos. El tiempo necesario de cada modelo de NAS ha ido reduciéndose con el avance en la investigación, sin embargo, todavía son modelos lentos e inefficientes. Los esfuerzos en arquitecturas jerárquicas [38] y en morfismo de redes [43] han ido dirigidos en este sentido, reducir el tiempo necesario.

Actualmente, el estado del arte tiende a un espacio de búsqueda jerárquico y en utilizar diversas técnicas (morfismo de redes, weight sharing, modelos one-shot...) para aligerar el tiempo de la estimación del rendimiento de las redes generadas.

Capítulo 4

Materiales y métodos

Para este estudio se han seleccionado tres modelos concretos de NAS que representan tres enfoques distintos dentro de este ámbito. Estos modelos se compararán con una red profunda diseñada por un experto para nuestro caso de estudio. En este capítulo, se describirá el problema al que nos enfrentamos y las particularidades de sus datos, además de las métricas y función de pérdida utilizadas. Por último, se describirán la red diseñada por el experto y los tres modelos de NAS seleccionados para la experimentación.

4.1. Datos del problema

De cara a poner en práctica los conocimientos adquiridos sobre NAS y comparar distintos métodos en un contexto común, se parte de un caso de estudio concreto: la clasificación automática de lesiones gastrointestinales en imágenes de colonoscopia convencional. Este es un problema de clasificación de imágenes con grandes aplicaciones médicas, como ya se comentó en el apartado 1.1.

Los datos de los que se dispone son 76 vídeos de colonoscopia, pertenecientes a la base de datos *Colonoscopy Dataset*¹, también disponible en el repositorio UCI². Los vídeos muestran imágenes del interior del colon grabados con un colonoscopio, y dichos vídeos son utilizados por gastroenterólogos para realizar su diagnóstico.

Un detalle importante es el hecho de que los colonoscopios utilizan dos tipos de luz para examinar las lesiones:

- **Luz blanca o *white light (WL)*:** como su nombre indica, es una

¹http://www.depeca.uah.es/colonoscopy_dataset/

²<https://archive.ics.uci.edu/ml/datasets/Gastrointestinal+Lesions+in+Regular+Colonoscopy>

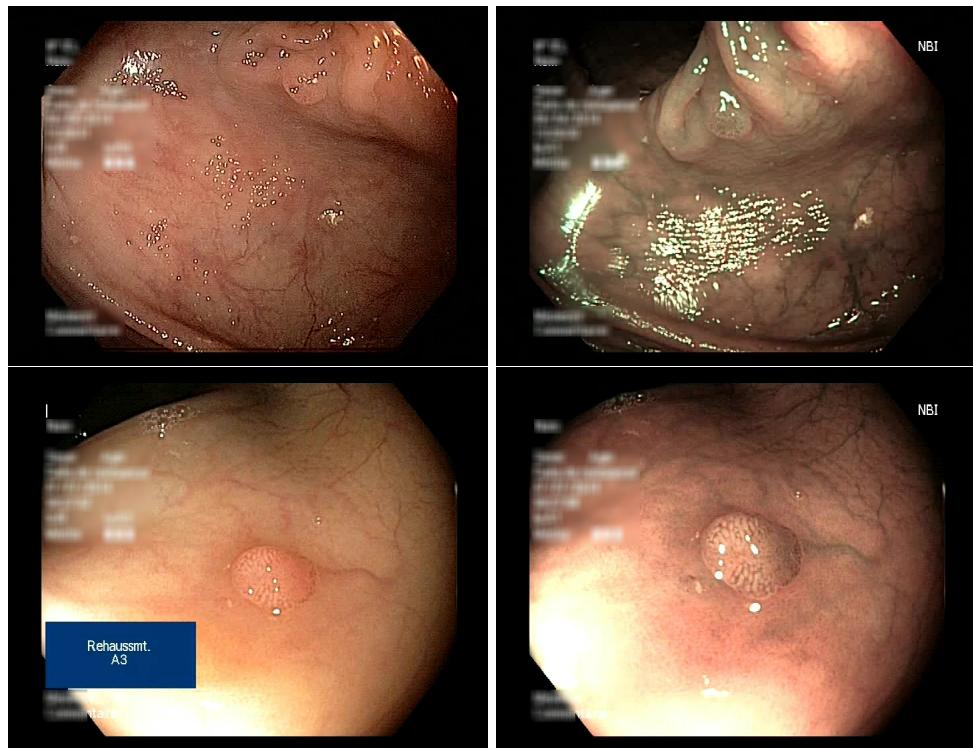


Figura 4.1: Comparativa entre los dos tipos de luz WL a la izquierda y NBI a la derecha. Estas imágenes son frames de los vídeos originales y todavía no tienen las regiones de las lesiones recortadas.

luz blanca que permite distinguir todo el espectro visible (ondas en el rango de 400 a 700 nm).

- **Imágenes de banda estrecha o *narrow band imaging* (NBI):** esta luz emplea solo longitudes de onda azul y verde, que es el rango en que la hemoglobina absorbe la luz, por lo que provoca que los vasos sanguíneos aparezcan más oscuros. Gracias a ello, con esta luz se es capaz de visualizar la estructura de la red vascular de la superficie mucosa.

La base de datos dispone de imágenes con los dos tipos de luz. Algunos ejemplos de las diferencias entre las imágenes generadas por cada tipo de luz puede observarse en la Figura 4.1. En nuestros experimentos utilizaremos los datos obtenidos con la luz NBI debido a que los experimentos previos han demostrado mejores resultados [11]. Estos resultados se pueden consultar en la sección 4.4.

Las imágenes presentes en la base de datos se corresponden con tres tipos de pólipos de lesiones pre-cáncerigenas (véase Figura 4.2):

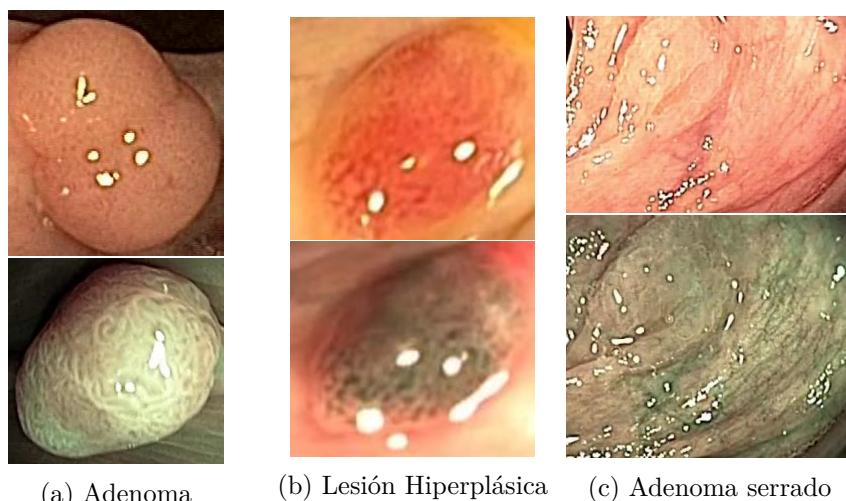


Figura 4.2: Ejemplos de los tres tipos de lesiones presentes en la base de datos. Las tipos (a) y (c) representan lesiones que deben ser eliminadas, mientras que el tipo (b) no presenta la misma gravedad y, por tanto, su eliminación (también llamada resección) no es obligatoria. La fila superior representa pólipos con luz WL, mientras que la de abajo emplea NBI.

- **Adenomas:** Se dispone de 40 casos. Son pólipos que clásicamente crecen con una progresión lenta de adenoma a adenoma displásico y carcinoma.
- **Lesiones hiperplásicas:** Se dispone de 21 casos. Son lesiones que no tienen capacidad de malignización, por lo que no entrañan peligro.
- **Adenomas serrados:** Se dispone de 15 casos. Estos pólipos tienen un aspecto muy característico y su proceso de malignización es diferente al resto, más sutil y difícil de detectar. Por ello son especialmente peligrosos.

Estos datos no están distribuidos equitativamente, podemos observar en la Figura 4.3 cómo el número de adenomas supera por mucho al resto de clases, concentrando más de la mitad de los datos, siendo seguido por las lesiones hiperplásicas con casi un 30 % y por último, los adenomas serrados, que ocupan poco menos del 20 %.

Los datos presentados cuentan dos dificultades adicionales. Primero, la gran variabilidad intra-clase. Dentro de cada clase, los pólipos toman una gran variedad de formas y aspectos, además de que debido a que las imágenes están extraídas de grabaciones con endoscopio, existen muchos cambios de iluminación y perspectiva, que incrementan la dificultad de la tarea (véase

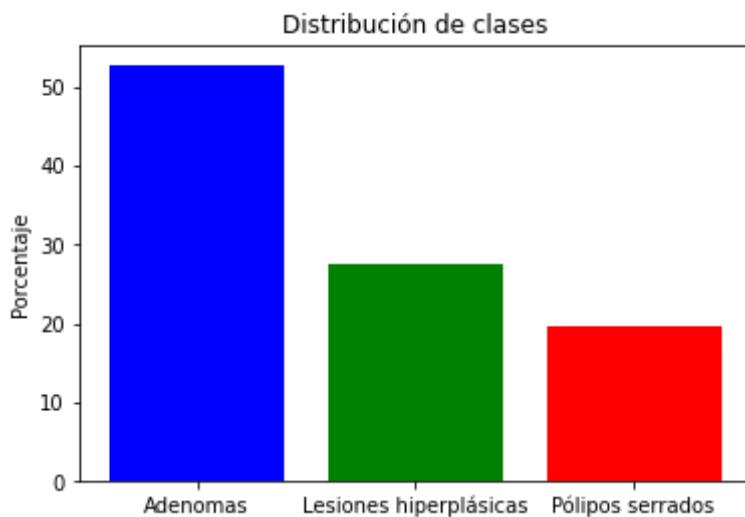


Figura 4.3: Distribución de las clases en los datos.

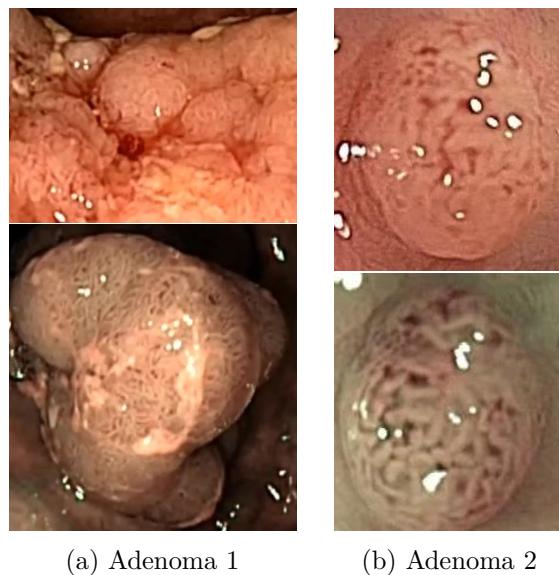
figura 4.4). La segunda característica es la poca variabilidad inter-clase, es decir, que imágenes de diferentes clases pueden ser muy parecidas entre sí.

Aunque el problema de clasificación de las tres clases por separado es un problema interesante desde el punto de vista científico, desde el punto de vista clínico es más útil enfocar el problema como una clasificación binaria entre únicamente dos clases:

- ***Resection:*** Aquellas lesiones que deben ser eliminadas directamente durante la colonoscopia (adenomas y serrados).
- ***No Resection:*** Las lesiones que no hace falta eliminar durante la misma colonoscopia, pues son pólipos no peligrosos (las lesiones hiperplásicas).

De esta forma, el problema sería clasificar una imagen de un pólipos en casos positivos (peligrosos, para eliminar) y casos negativos (benignos, no eliminar). Con nuestro modelo de ML, dispondríamos de una herramienta automática que asistiera al médico durante una colonoscopia en la decisión inmediata de retirar o no una lesión concreta en el momento. Con este planteamiento, debemos tener en cuenta que marcar una lesión benigna como peligrosa, implica eliminar un pólipos innecesariamente. Sin embargo, que una lesión peligrosa se marque como inofensiva puede significar que el paciente desarrolle un cáncer. Por ello, debemos priorizar minimizar el número de falsos negativos.

El problema binario cambia la distribución de clases de los datos, agrupando los adenomas y serrados como se muestra en la Figura 4.5. Esto con-



(a) Adenoma 1 (b) Adenoma 2

Figura 4.4: Ejemplos de cómo dentro de una misma clase (adenomas) las imágenes pueden ser muy diferentes entre sí. Esto se conoce como variabilidad intraclasses. La luz utilizada es WL (arriba) y NBI (abajo).

lleva el problema de que el desbalanceo entre las clases se ha visto aumentado con más de un 70 % de casos positivos. Esto hace pensar que cualquier modelo de aprendizaje entrenado con estos datos tendrá un sesgo inherente hacia clasificar como peligroso un pólipos. Esto no es necesariamente malo: como hemos explicado anteriormente, es preferible tener falsos positivos que falsos negativos que puedan dejar escapar pólipos cancerígenos. Aun así es una característica de los datos que debemos tener presente en todo el proceso.

Las imágenes de la base de datos han sido recortadas de los frames de los vídeos en donde se apreciaban las lesiones, pero tienen tamaños distintos, por lo que se han homogeneizado todos los datos a la dimensión 224 x 224. Como preprocesado de los datos se ha optado por únicamente realizar una normalización de los valores de las intensidades de los píxeles, para que pasen de la escala [1,255] al rango [0,1]. No se ha querido experimentar con más técnicas de preprocesado, para poder comparar con facilidad los resultados que se obtengan con los trabajos anteriores, en donde se había optado por sólo cambiar el tamaño y normalizar los datos [10, 11].

4.2. Métricas

Es necesario establecer una métrica para establecer el rendimiento de cada modelo en nuestro problema concreto. Esto permitirá tener una medida

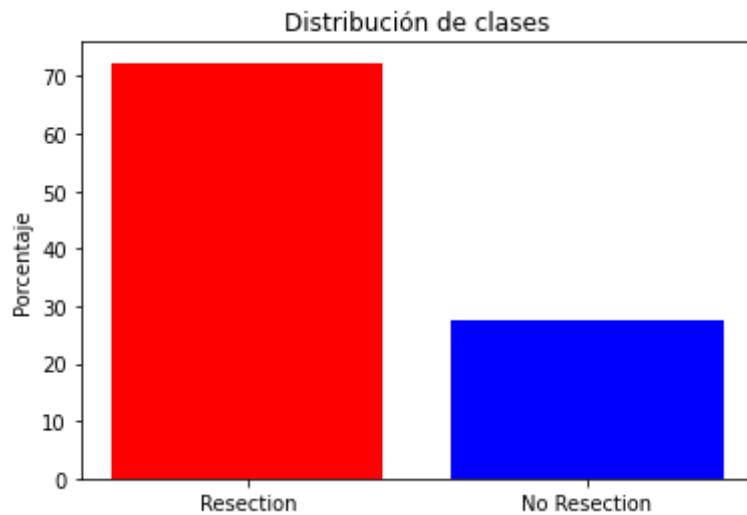


Figura 4.5: Distribución de las clases en los datos del problema binario.

		Salida del modelo	
		Casos Positivos	Casos Negativos
Clase real	Casos Positivos	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Casos Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

Cuadro 4.1: Matriz de confusión

estándar que utilizar para comparar los modelos empleados.

Recordamos que, ante un problema de clasificación supervisada, el modelo recibe una muestra como entrada y , y, como salida, ofrece una etiqueta que representa la clase a la que pertenece la muestra. El clasificador acierta si la etiqueta predicha es igual a la clase real. Por ello, para medir el rendimiento del clasificador, utilizaremos la matriz de confusión (véase Tabla 4.1). Esta matriz cuadrada representa el número de aciertos y errores de un modelo en un problema de clasificación. Cada fila de la matriz representa el número de casos de la clase real, mientras que cada columna representa el número de casos predichos por el modelo. Así, la matriz cuenta con 4 variables:

- **Verdaderos positivos (VP):** Número de casos en los que el modelo clasifica correctamente las muestras positivas.
- **Falsos negativos (FN):** Número de casos positivos que el modelo clasifica incorrectamente como negativos.

- **Falsos positivos (FP):** Número de casos negativos que el modelo clasifica incorrectamente como positivos.
- **Verdaderos negativos (VN):** Número de casos en los que el modelo clasifica correctamente las muestras negativas.

En un problema de clasificación binaria, solo tenemos una matriz de confusión correspondiendo los casos negativos con una de las clases y los casos positivos con otra. En nuestro problema, consideramos los casos de *resection*, los pólipos que hay que eliminar (adenomas y serrados), como casos positivos y los casos de *no resection* (lesiones hiperplásicas) como casos negativos. A partir de los valores de la matriz de confusión, podemos calcular métricas derivadas:

Accuracy (exactitud): la métrica más utilizada en los problemas de clasificación. Mide la proporción de casos correctamente clasificados sobre los casos totales.

$$\text{Accuracy} = \frac{VP + VN}{VP + FN + FP + VN}$$

Sensitivity (sensibilidad): mide la proporción de casos correctamente clasificados sobre todos los casos positivos.

$$\text{Sensitivity} = \frac{VP}{VP + FN}$$

Specificity (especificidad): análoga a la *sensitivity*, la *specificity* mide la proporción de casos negativos correctamente clasificados.

$$\text{Specificity} = \frac{VN}{VN + FP}$$

Precision (precisión): mide la proporción de casos positivos correctamente clasificados con respecto a todas las muestras clasificadas como positivas.

$$\text{Precision} = \frac{VP}{VP + FP}$$

Todas estas métricas tienen valores entre 0 y 1, siendo un 1 el mejor caso.

En nuestro problema, la métrica más importante es la *sensitivity*, es decir, priorizar el clasificar correctamente los casos positivos más que los casos negativos. Esto se explica desde un punto de vista clínico: como explicamos

anteriormente, es más peligroso dejar pasar un pólipos cancerígeno que un falso positivo. Por ello, se busca minimizar el número de falsos negativos. A este respecto, el estudio anterior que trató con estos datos [11], desarrolló una métrica *ad hoc* para recoger este hecho. Esta ponderación da mucho peso a la sensitivity, seguido de la accuracy. De este modo, se premia tener un equilibrio entre un buen rendimiento general y minimizar los falsos negativos.

$$\text{Ponderacion} = 0.35 \cdot \text{Sensitivity} + 0.25 \cdot \text{Accuracy} +$$

$$0.2 \cdot \text{Specificity} + 0.2 \cdot \text{Precision}$$

En los experimentos, se continuará usando la misma métrica, primero porque es un valor que refleja la importancia dada a la *sensitivity* y, segundo, porque de ese modo es posible comparar los resultados de los modelos con los experimentos anteriores.

4.3. Función de pérdida

Como función de pérdida o *loss* (véase Sección 2.2), se utilizará la *binary crossentropy*, debido a que es la función más utilizada en problemas de clasificación binarios. Su expresión matemática se muestra a continuación. En esta fórmula se compara la etiqueta real de la muestra (y_i) con la etiqueta predicha por el modelo ($p(y_i)$). La N hace referencia al número de muestras del conjunto que se ha predicho.

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

4.4. Red diseñada por experto

Como *baseline*, probaremos la red convolucional diseñada a mano por un experto, que se analizó en un TFG previo [11]. La red propuesta (véase Figura 4.6) es una red convolucional convencional, formada principalmente por capas convolucionales y max-pooling. Como método de regularización, utiliza dropout después de cada capa de pooling.

Luz	Acc	Spe	Sen	Pre	Ponderación
WL	0.566	0.72	0.269	0.333	0.446
NBI	0.803	0.812	0.75	0.429	0.711

Cuadro 4.2: Resultados originales de [11]. Obtiene una accuracy del 80.3 % y un valor de ponderación de 71.1 %.

Model: "sequential"

```

Layer (type)          Output Shape         Param #
=====
conv2d (Conv2D)      (None, 222, 222, 32) 896
conv2d_1 (Conv2D)    (None, 220, 220, 64) 18496
max_pooling2d (MaxPooling2D) (None, 110, 110, 64) 0
dropout (Dropout)   (None, 110, 110, 64) 0
conv2d_2 (Conv2D)    (None, 108, 108, 64) 36928
max_pooling2d_1 (MaxPooling2 (None, 54, 54, 64) 0
dropout_1 (Dropout) (None, 54, 54, 64) 0
flatten (Flatten)   (None, 186624) 0
dense (Dense)        (None, 1)           186625
=====

Total params: 242,945
Trainable params: 242,945
Non-trainable params: 0

```

Figura 4.6: Resumen de la arquitectura de la red diseñada por un experto.

Los resultados de la publicación original se pueden consultar en la Tabla 4.2. Todos los experimentos fueron con el problema de clasificación binaria y se probaron los datos con los dos tipos de luz. Los resultados son mucho mejores con la luz NBI, por lo tanto, esta será la luz utilizada en los experimentos.

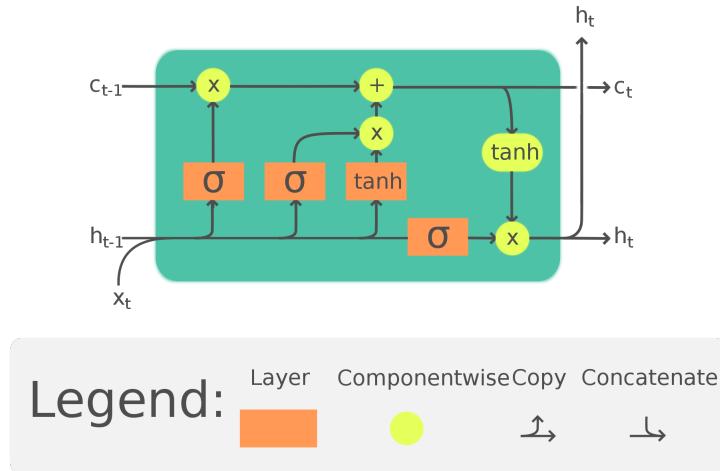


Figura 4.7: Esquema de una red LSTM. Imagen extraída de [Wikipedia](#)

4.5. Aprendizaje por refuerzo: ENAS

Como primera técnica de NAS, utilizaremos aprendizaje por refuerzo, específicamente, uno de los modelos más reconocidos y estudiados del campo, *Efficient Neural Architecture Search* (ENAS)[33]. Como se explicó en la Sección 2.3.2, en las técnicas de aprendizaje por refuerzo se utiliza una red controlador que genera redes hijas para resolver el problema, recibiendo como recompensa el rendimiento de la red generada.

La red controladora es una arquitectura *Long Short Term Memory* (LSTM) [73]. Esto significa que es una red recurrente que mantiene en memoria un flujo de información a través de las iteraciones. En la Figura 4.7, se puede observar que la salida del flujo de una iteración anterior (h_{t-1}) se utiliza como entrada de la siguiente iteración (h_t). Este flujo actúa como memoria a largo plazo [67, 69].

En el artículo original se proponía utilizar esta técnica tanto con macro-arquitectura como con micro-arquitectura. Sin embargo, dado que sus resultados fueron mejores y los tiempos más reducidos con micro-arquitectura, nos centraremos en esa versión (véase Sección 2.3.1). En este caso, el controlador es responsable de diseñar dos celdas (o bloques): la celda normal y la celda de reducción, organizados según una disposición preliminar que el diseñador debe fijar antes de comenzar. Cada celda de ambos tipos está formada por B nodos que se conectan en forma de grafo dirigido acíclico. La tarea del controlador será decidir cómo se conformará cada nodo de las celdas. Este proceso se puede observar en la Figura 4.8, que reproduce un ejemplo en el que se genera una celda de 4 nodos. En cada celda, los dos

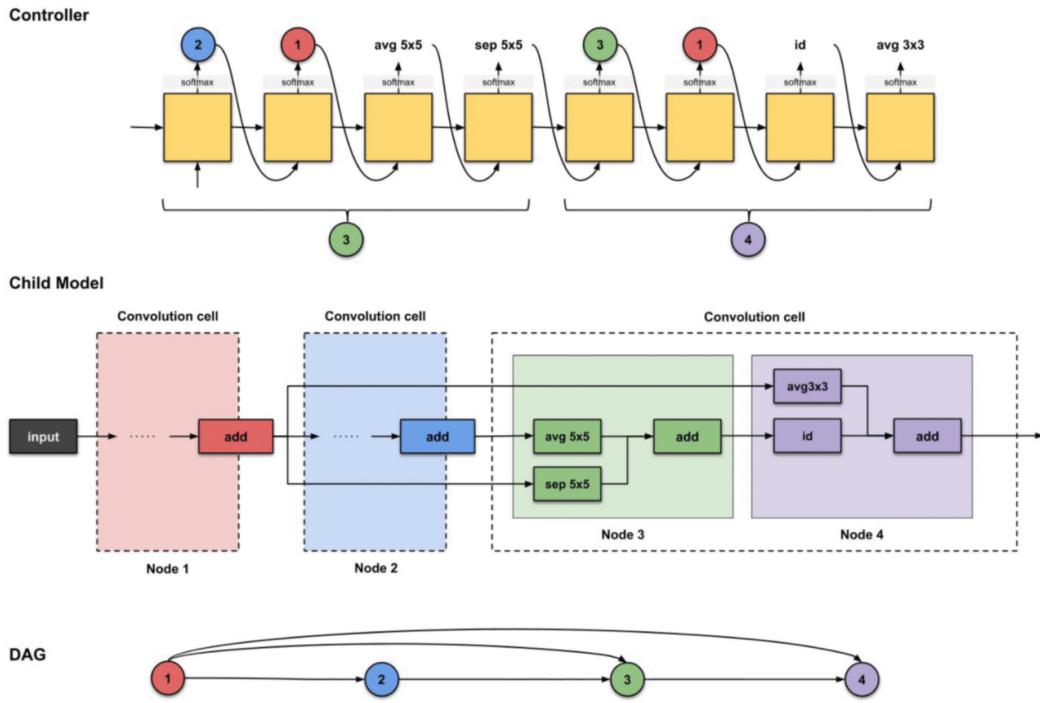


Figura 4.8: El controlador (arriba) genera por cada nodo 4 elementos: dos nodos anteriores a los que se tiene que conectar y dos operaciones para aplicar a cada una de estas entradas. Por ejemplo, en el nodo 3 (verde) se puede observar como se generan los parámetros 2, 1, avg 5x5 y sep 5x5 que se traducen en la estructura verde del modelo hijo (imagen del centro). Debajo, se puede observar un grafo que representa la arquitectura final de la celda. Imagen extraída de [68].

primeros nodos se consideran siempre *inputs* de las dos celdas anteriores, por lo que no es necesario realizar ninguna operación. En cuanto al resto de nodos, están formados por dos operaciones aplicadas a dos nodos anteriores y una operación *add* que agrega los resultados de ambas operaciones. Por tanto, cada nodo necesita 4 parámetros: las dos operaciones que se generan y los dos nodos a los que se aplican estas operaciones. Los nodos que no se utilizan como salida para ningún otro nodo son agregados y su media se convierte en el output del nodo. La única diferencia de la celda de reducción es que esta utiliza una convolución inicial en cada uno de los 2 primeros nodos de input con un valor de *strides* de 2 para reducir el tamaño de la entrada.

Una vez se tienen las dos celdas diseñadas, ya se puede generar la red completa, pues todas las celdas del mismo tipo compartirán arquitectura. Después de todas esas celdas, la red consistirá en una capa de *pooling* que

reducirá la entrada a un píxel y una capa densa con una neurona que devolverá el resultado a partir de una operación sigmoide. A continuación, la red se entrenará con los datos de entrenamiento y se obtendrá su rendimiento con los datos de validación. Este valor será utilizado para realimentar a la red controladora como recompensa para su próxima iteración.

Las posibles operaciones para los nodos son las siguientes:

- Convolución de 3x3 o 5x5
- Convolución separable en profundidad ³ de 3x3 o 5x5.
- Max Pooling 3x3
- Average Pooling 3x3

La gran innovación de ENAS con respecto a los modelos anteriores de RL [40] es el uso de *transfer learning*, es decir, si dos nodos se han entrenado previamente, sus pesos de las capas convolucionales se mantendrán, ahorrando tiempo en volver a entrenarlos.

En nuestros experimentos, hemos utilizado una implementación de ENAS en Pytorch incluida en la librería NNI [70].

4.6. Optimización bayesiana: Auto-Keras

Auto-Keras [43] es un modelo de NAS publicado en 2019 que dispone de una librería de software libre muy popular para NAS.

Este método construye la red secuencialmente tomando cada capa como una unidad, por lo que el espacio de búsqueda de esta aproximación es macro-arquitectura. Los bloques que conforman este espacio de búsqueda incluyen a los siguientes (la lista completa es más larga):

- Convolución
- Capa densa
- Bloque residual
- Normalización

La estrategia de búsqueda se estructura como un árbol. Cada nodo representa una arquitectura y están conectadas por aristas que representan

³Operación de convolución que opera con la dimensión de profundidad, no sólo las dimensiones espaciales.

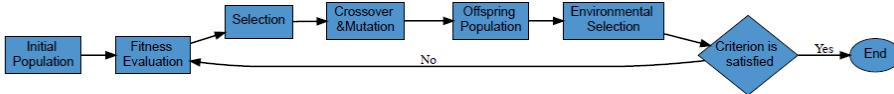


Figura 4.9: Ciclo básico del algoritmo genético. Imagen extraída de [66].

un cambio en una de las capas. Es decir, de un nodo (una arquitectura concreta) cuelgan tantos nodos como posibles cambios de capas haya. De estos nodos a su vez cuelgan otros tantos con los posibles cambios que se pueden hacer en sus capas. El rendimiento de todos estos nodos es aproximado mediante una función heurística que, básicamente, compara los resultados de las arquitecturas que ya se han entrenado con las arquitecturas potenciales utilizando una función de distancia que compara la cantidad de diferencias existentes entre dos arquitecturas. Para navegar por ese árbol y obtener el siguiente modelo que entrenar se utiliza una versión modificada de A* [71]. Como forma de reducir el tiempo de evaluación, se utiliza morfismo de redes, es decir, para las partes que no se modifican al pasar de una red a la siguiente, se reutilizan los pesos de la iteración anterior, ahorrando tiempo al no tener que reentrenar la red completa.

Para este modelo utilizaremos la implementación de la librería *open-source* Auto-Keras, publicada por los mismos autores.

4.7. Algoritmo Evolutivo: Auto CNN

Como representante de los algoritmos evolutivos se ha optado por una publicación del 2020 [66] en el que presentan un modelo denominado Auto CNN.

Este algoritmo sigue el esquema clásico de los algoritmos genéticos. Se comienza con una población inicial de N individuos generada aleatoriamente y se procede a realizar el ciclo que se puede observar en la Figura 4.9. El fitness de la población es evaluado y se aplican los operadores de cruce y mutación para generar nuevos individuos. Por último, se escogen N individuos de entre la unión de la generación anterior y los nuevos individuos para que el número de individuos se mantenga constante.

El espacio de búsqueda se aplica a la red neuronal completa, por lo que esta técnica utiliza macro-arquitectura (véase Sección 2.3.1). Los individuos son arquitecturas formadas por un número variable de bloques. Estos bloques pueden ser de dos tipos:

- **Skip layer:** capa formada por dos capas convolucionales y una *skip*

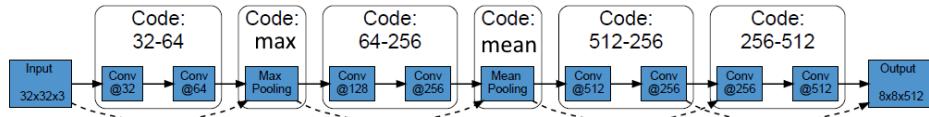


Figura 4.10: Ejemplo de arquitectura generada por Auto CNN con código “32-64-max-64-256-mean-512-256-256-512”. Imagen extraída de [66].

connection que suma la entrada del bloque con la salida. Un detalle importante es que la entrada del bloque se debe ajustar con una convolución 1x1 para mantener el número de filtros constante en la suma de la *skip connection*. Los tamaños de kernel de estas capas son de 3x3 (excepto la de 1x1) y los strides 1x1. El número de filtros de las dos capas será un parámetro que distinguirá unos bloques de otros. Estas capas se representarán por el número de filtros de sus capas convolucionales que pueden tomar los valores [32, 64, 128]. Por ejemplo, un bloque que tenga convoluciones de 32 y 64 filtros se denominará “32-64”.

- **Pooling:** las capas de pooling pueden ser *max pooling* o *average pooling* (véase Sección 2.2.3). Estos bloques se representan simplemente con el tipo de *pooling* que implementan. Así, estos bloques pueden representarse como “max” o “mean”.

Así pues, con el sistema de representación mostrado se puede codificar una red fácilmente y generarla a partir de este método de codificación directo. Un ejemplo se puede observar en la Figura 4.10. Además (como añadido al modelo propuesto al original), se han incorporado una capa de Dropout de 0.5 después de cada capa de *pooling* y justo antes de la última capa totalmente conectada para ayudar a la regularización.

Una vez explicado el espacio de búsqueda, podemos detallar cómo actúan los operadores de cruce y mutación. El operador de cruce se puede observar en el algoritmo 1. Dos individuos se escogen aleatoriamente de la población y, si las probabilidades de cruce indican que haya cruce, se selecciona un punto aleatorio de los individuos. Este punto debe ser el inicio de un bloque (de skip o pooling) y divide al individuo en dos partes. Se generan dos nuevos individuos: uno con la primera parte del primer individuo y la segunda del segundo y otro que tenga la primera parte del segundo y la segunda del

primero.

Algorithm 1: Operador de cruce

Input: Población P , probabilidad de cruce C

$Q \leftarrow \emptyset;$
 $p_1 \leftarrow$ seleccionar individuo aleatoriamente de P ;
 $p_2 \leftarrow$ seleccionar individuo aleatoriamente de P ;
 $r \leftarrow$ número aleatorio $(0, 1)$;
if $r < C$ **then**
 | Seleccionar un punto aleatorio de p_1 y dividirlo en dos;
 | Seleccionar un punto aleatorio de p_2 y dividirlo en dos;
 | $o_1 \leftarrow$ Unión de la primera parte de p_1 y la segunda de p_2 ;
 | $o_2 \leftarrow$ Unión de la primera parte de p_2 y la segunda de p_1 ;
 | $Q \leftarrow Q_t \cup o_1 \cup o_2$;
else
 | $Q \leftarrow Q_t \cup p_1 \cup p_2$;
end
return Q ;

En cuanto al operador de mutación, se puede observar en el Algoritmo 2. Cuando un individuo es seleccionado para la mutación, se selecciona aleatoriamente un punto en su arquitectura y en él recibe una de las siguientes posibles mutaciones:

- Añadir una *skip layer* con configuración aleatoria. Esta opción tiene más probabilidades de aparecer para favorecer el crecimiento de las arquitecturas.
- Añadir una capa de *pooling* con configuración aleatoria.
- Eliminar un bloque.
- Cambiar aleatoriamente los parámetros de un bloque.

Algorithm 2: Operador de mutación

Input: Nuevos individuos Q , probabilidad de mutación M

for $p \in P$ **do**
 | $r \leftarrow$ número aleatorio $(0, 1)$;
 | **if** $r < M$ **then**
 | | $i \leftarrow$ Punto aleatorio en p ;
 | | $m \leftarrow$ Seleccionar mutación aleatoria de la lista;
 | | Aplicar m a p en el punto i ;
 | **end**
end
return Q ;

Por último, una vez se dispone de todos los nuevos individuos y se calcula su fitness. Al conjunto de la unión de la población original y los nuevos individuos, se le realiza una selección mediante torneo con elitismo. Por cada miembro de la nueva generación, se seleccionan dos individuos y se comparan sus fitness. Aquel que tenga mejor fitness pasa a la nueva generación. Una vez todos los individuos han sido seleccionados, se comprueba si el individuo con mejor fitness ha sido seleccionado. Si no es así, el individuo con peor fitness de aquellos que han sido seleccionados es sustituido por él.

Por último, tenemos que comentar que para reducir los tiempos de entrenamiento se hace uso de una caché de memoria que almacena todas las arquitecturas ya generadas con su valor de fitness. De este modo, se ahorra el tiempo necesario para volver a calcular todas las arquitecturas ya entrenadas previamente. En otro caso el tamaño de la caché podría resultar muy grande, pero debido a que todas las arquitecturas se pueden almacenar con el código ya comentado, el tamaño de la caché no supone ningún problema.

Para este proyecto, utilizaremos la implementación proporcionada por los mismos autores del paper. Sin embargo, ha sido necesario ajustarla para nuestro problema concreto, pues el modelo no estaba preparado inicialmente para la clasificación binaria, además de otros detalles de implementación que fue necesario adaptar.

Capítulo 5

Planificación e Implementación

5.1. Planificación

Para la planificación de este trabajo se optó por un ciclo de vida en cascada realimentado. Este ciclo de vida es muy sencillo y el más indicado para proyectos de una pequeña complejidad y envergadura. El añadido de que es un ciclo realimentado permite volver atrás en caso de que algún cambio en el planteamiento lo hiciera necesario.

En cuanto a la planificación temporal, en un principio se planificó para terminar en Junio, como se indica en el diagrama de Gantt de la Tabla 5.1. Sin embargo, todo el proyecto se vio retrasado debido a la pandemia y la carga lectiva del segundo cuatrimestre (pues disponía de un número de asignaturas superior al habitual). Como resultado, durante los meses de enero a mayo se avanzó muy poco y hubo que continuar trabajando a partir de junio. Por ello, se resolvió dejar la entrega del proyecto para septiembre y utilizar el verano. La planificación final, como se puede observar en el diagrama de Gantt 5.2, se retrasó hasta agosto debido a que, durante el curso se pudo dedicar muchas menos horas de trabajo y que, sin embargo, durante el verano se pudo realizar un trabajo más intensivo.

El coste de esta investigación, se valoró con el supuesto de un Investigador Senior o Responsable de I+D de una empresa tecnológica¹. Se supuso un salario de 35 euros/h y 7h diarias de trabajo. La planificación inicial, por tanto, contaba con una duración de 230 días y un coste económico de 56.350€. La planificación final consistió en 290 días y 71.050€.

¹Datos obtenidos a partir de un proyecto europeo recientemente enviado por una empresa proveedora de servicios tecnológicos ([Panacea Cooperative Research](#))

Tarea	Duración	2020					2021				
		Sep	Oct	Nov	Dic	Ene	Feb	Mar	Abr	May	Jun
Análisis y estudio	69 días										
Diseño	69 días										
Implementación	46 días										
Experimentación	46 días										

Cuadro 5.1: Planificación inicial del proyecto.

Tarea	Duración	2020						2021				
		Sep	Oct	Nov	Dic	Ene	Feb	Mar	Abr	May	Jun	Jul
Análisis y estudio	92 días											
Diseño	138 días											
Implementación	30 días											
Experimentación	30 días											

Cuadro 5.2: Planificación final del proyecto.

El trabajo se planificó en base a las siguientes tareas:

- **Análisis y estudio:** revisión bibliográfica del estado del arte en NAS y las principales tendencias y corrientes metodológicas.
- **Diseño:** diseño de experimentos a realizar en nuestro caso de estudio, además de la elección de las técnicas de NAS a implementar.
- **Implementación:** implementación en Python de los experimentos y todas las técnicas escogidas.
- **Experimentación:** ejecución de los experimentos y extracción de conclusiones. Esta fase puede implicar la vuelta a la fase de diseño si los resultados obtenidos así lo aconsejasen.

5.2. Implementación

El propósito de esta implementación no estaba dirigido a crear un software plenamente funcional para un cliente final, sino dar soporte a la experimentación necesaria para llevar a cabo esta investigación. Pese a ello, se ha prestado una especial atención al diseño del software para asegurarse que su funcionamiento sea correcto. El código se ha organizado al estilo de Python, modularizando las funciones en distintos archivos para que el código cumpliera con los estándares esperables de calidad (por ejemplo, que el código sea fácilmente mantenible y extensible).

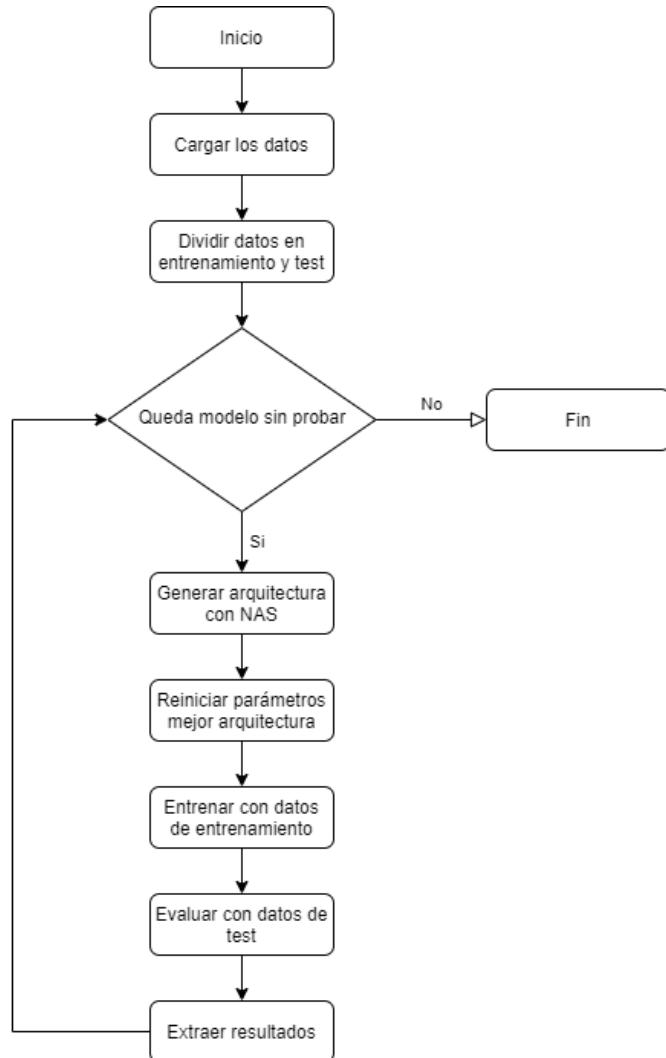


Figura 5.1: Diagrama de flujo de la realización de los experimentos.

El código se coordina desde un fichero *main*, que controla todo el flujo de entrenamiento. Haciendo uso de funciones de primer grado², el *main* llama a cada función correspondiente a un modelo de NAS y devuelve la mejor arquitectura encontrada. Estas funciones están ubicadas en otros ficheros, para una mayor modularidad y extensibilidad. Una vez se dispone de la arquitectura se procede con todo el protocolo de validación experimental (véase Sección 6). Todo el flujo principal se puede observar en la Figura 5.1.

Para comprobar el correcto funcionamiento de todo el código se ha hecho uso de las pruebas unitarias comunes al campo de la visión por computador:

²Funciones que actúan como variables, pudiéndose pasar como parámetros o ser devueltos por otras funciones.

análisis de las curvas de entrenamiento para verificar que este es correcto, depurar la generación de la matriz de confusión para asegurar que se contabilizan correctamente los distintos ejemplos y por tanto se mide el rendimiento real, etc.

Todo el código se ha publicado en GitHub (github.com/alekpinel/TFG-NAS) con un pequeño documento explicando cómo utilizarlo y las referencias de cada algoritmo.

5.3. Lenguaje y entorno

La implementación de todos los algoritmos ha sido llevada a cabo en el lenguaje de programación Python (versión 3.8.11), debido a ser el lenguaje más popular en la actualidad para proyectos de DL. Para la realización de este trabajo se han utilizado las librerías más comunes en este ámbito: OpenCV (4.0.1) y Scikit-learn (0.24.2), así como Numpy (1.18.5) para las operaciones matriciales o Matplotlib (3.4.2) para los gráficos. Dos de los modelos de NAS han sido implementados con Tensorflow (2.3.0) y Keras (2.4.3), pero también se ha utilizado Pytorch (1.4.0) para la implementación de ENAS. Todos los experimentos se han realizado en un entorno local con las siguientes especificaciones:

- **CPU:** AMD Ryzen 5 1500x
- **RAM:** 24GB
- **GPU:** NVIDIA GeForce GTX 1660 SUPER

Capítulo 6

Experimentación

Para los experimentos se ha optado por utilizar *hold-out* como protocolo de validación experimental. El procedimiento consiste en dividir *a priori* los datos en dos grupos: entrenamiento y test. Los datos de entrenamiento son utilizados para entrenar nuestros modelos y, una vez el modelo está completamente ajustado, se evalúa su resultado con los datos de test. Fueron considerados otros protocolos de experimentación como la validación cruzada¹ o *leave-one-out*². Estas técnicas son más robustas que *hold-out* porque los resultados dependen menos de la partición concreta escogida. Sin embargo, el entrenamiento de los modelos es muy costoso a nivel computacional y realizar un *leave-one-out* implicaría multiplicar los tiempos de entrenamiento por 76. Otro motivo para optar por *hold-out* reside en poder comparar fácilmente las arquitecturas resultantes. Dado que el objetivo de las técnicas de NAS es generar una arquitectura, si repitiéramos los experimentos 76 veces es probable que obtuviésemos diferentes arquitecturas. Esto dificultaría el agregar e interpretar los resultados y no permitiría mostrar las arquitecturas resultantes de modo sencillo. Por último, la gran mayoría de publicaciones en la literatura de NAS utilizan *hold-out* en sus estudios debido a las razones ya expuestas, por lo que se emplea esta técnica también en este trabajo.

El conjunto de test escogido es del 30 % del total, cuidando que las proporciones de las tres clases se mantuviesen en los dos conjuntos. La mayoría de técnicas de NAS requieren de un conjunto adicional de datos: el conjunto de validación. Este conjunto permite obtener una estimación de la bondad de las arquitecturas que se van generando. El conjunto de entrenamiento se

¹Consistente en dividir los datos en K partes iguales y realizar K entrenamientos diferentes, de modo que en cada entrenamiento se emplean todas las partes menos una que se utiliza como test.

²Validación cruzada llevada al extremo, tomando como K el número de datos de los que se dispone.

verá dividido entonces en entrenamiento y validación en todos los experimentos. El porcentaje de validación es del 30 % sobre los datos totales de entrenamiento.

El procedimiento concreto llevado a cabo en los experimentos ha sido el siguiente:

1. Utilizar la técnica de NAS para generar una arquitectura utilizando los datos de entrenamiento y validación.
2. Reiniciar los parámetros/pesos de la red generada.
3. Entrenar durante 100 épocas con los datos de entrenamiento y validación.
4. Probar el modelo con los datos de test para calcular su rendimiento.

A continuación, en la Tabla 6.1 se muestran los resultado obtenidos, indicando la *accuracy* (ACC), *sensitivity* (SEN), *specificity* (SPE), *precision* (PRE) y la ponderación definida en la Sección 4.2. También se indica el tiempo que se ha tardado en ajustar el modelo. Este tiempo es aproximado y depende de muchos factores, pero es útil para comparar globalmente modelos entre sí. Se hablará brevemente de los hiperparámetros utilizados en cada modelo y de los resultados obtenidos.

	Tiempo	# Params	ACC	SEN	SPE	PRE	Ponderación
Red Experto	-	242.945	0.783	0.824	0.667	0.875	0.792
ENAS	3.17h	35.321	0.913	0.882	1.000	1.000	0.937
Auto-Keras	2.28h	2.998.035	0.870	0.941	0.667	0.889	0.858
Auto CNN	0.45h	6.460.929	0.826	0.941	0.500	0.842	0.804

Cuadro 6.1: Resultados finales de los experimentos, en donde se compara el rendimiento de la red profunda diseñada por un experto [11], con la proporcionada por ENAS [33], Auto-Keras [43] y Auto CNN [66]. Se puede observar que los mejores resultados (indicados en negrita) son obtenidos por ENAS y, del mismo modo, todas las técnicas de NAS superan el rendimiento de la red diseñada manualmente.

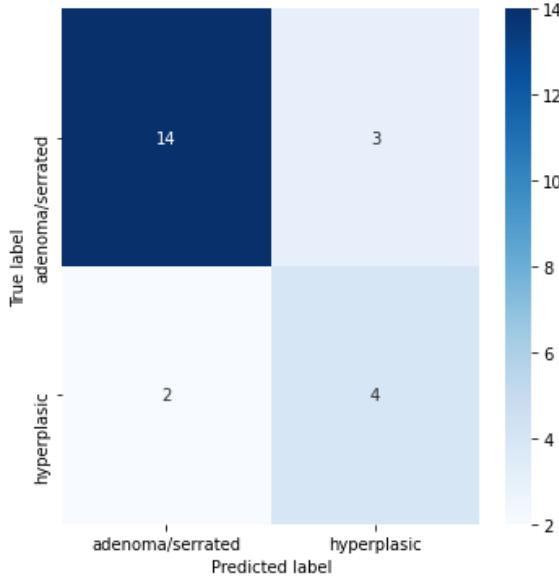


Figura 6.1: Matriz de confusión de la red diseñada por un experto.

6.1. Red diseñada por experto

La red diseñada por un experto ha sido ejecutada de la misma forma que en el trabajo previo [11]. Los resultados son similares a los obtenidos en dicho trabajo y la matriz de confusión se muestra en la Figura 6.1. Se obtiene un 78% de *accuracy* y un 79% de ponderación.

6.2. ENAS

El espacio de búsqueda de ENAS fue establecido en 3 celdas normales más 2 celdas de reducción. Cada celda normal, a su vez, estaba formada por 4 nodos, dos de ellos considerados inputs y otros dos con operaciones (de forma idéntica al esquema presentado en la Sección 4.5). El esquema se puede observar en la Figura 6.2. Se estableció un ratio de dropout de 0.4 y se dejó entrenar al controlador durante 200 épocas. Todos estos parámetros fueron escogidos en base a una batería experimental previa.

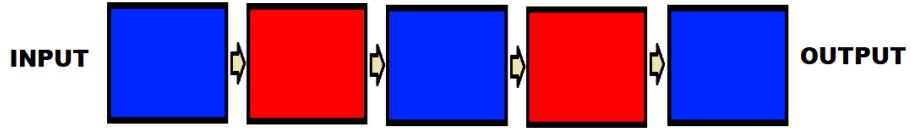


Figura 6.2: La arquitectura generada por ENAS, consistente en 3 capas normales (azul) y 2 capas de reducción (rojo)

El esquema de la arquitectura encontrada se muestra en la Figura 6.3, mientras que un esquema de las celdas normales y de reducción se incluye en las Figuras 6.4 y 6.5. Se puede observar que la arquitectura encontrada es bastante extraña (realiza la suma de dos capas de *max pooling*, por ejemplo) pero se puede interpretar como que la arquitectura tiende a no utilizar todas las operaciones a su disposición para reducir su propio tamaño y reducir el overfitting. A ese respecto, lo más notable de esta arquitectura es que se haya generado una red con un número de parámetros tan reducido. La red apenas tiene 35.000 parámetros, lo que contrasta con todos los demás modelos probados (incluyendo la red diseñada a mano). Este pequeño tamaño parece ser muy beneficioso para el rendimiento, pues los resultados son muy buenos (véase la matriz de confusión de la Figura 6.6), pues se obtiene un 91 % de *accuracy* y un 93.7 % de ponderación.

```
ENAS 3L 2N E200:=====
Layer (type:depth-idx)           Output Shape      Param #
=====
MicroNetwork
└─Sequential: 1-2               --                --
    └─Conv2d: 2-1              [32, 60, 224, 224]   --
    └─BatchNorm2d: 2-2          [32, 60, 224, 224]   1,620
    └─ModuleList: 1-1
        └─ENASLayer: 2-3       [32, 60, 224, 224]   3340
        └─ReductionLayer: 2-4   [32, 40, 112, 112]   5240
        └─ENASLayer: 2-6       [32, 40, 112, 112]   2680
        └─ReductionLayer: 2-7   [32, 80, 56, 56]    13680
        └─ENASLayer: 2-9       [32, 80, 56, 56]    6560
    └─AdaptiveAvgPool2d: 1-3  [32, 80, 1, 1]     --
    └─Dropout: 1-4             [32, 80]            --
    └─Linear: 1-5              [32, 1]             81
    └─Sigmoid: 1-6             [32, 1]            --
=====
Total params: 35,321
Trainable params: 35,321
Non-trainable params: 0
Total mult-adds (G): 13.21
=====
Input size (MB): 19.27
Forward/backward pass size (MB): 4367.32
Params size (MB): 0.14
Estimated Total Size (MB): 4386.73
=====
```

Figura 6.3: Esquema resumen de la arquitectura generada. Es notable el reducido número de parámetros.

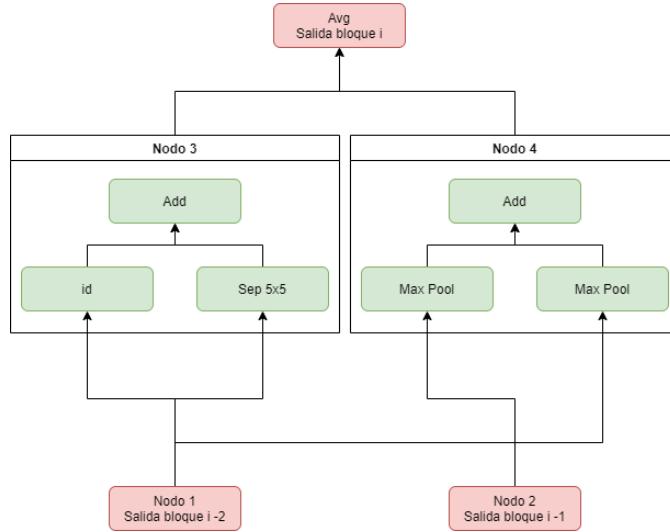


Figura 6.4: Esquema de una celda normal. Es notable que sólo se realice una operación de convolución en toda la celda y que el segundo nodo utilice dos capas de pooling.

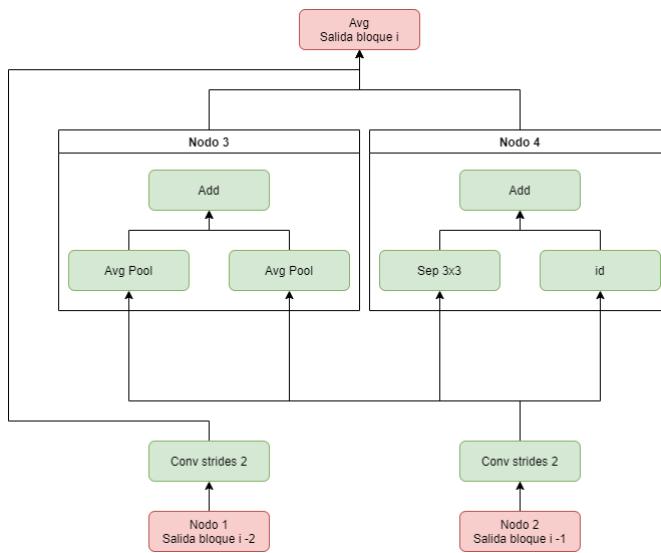


Figura 6.5: Esquema de una celda de reducción. Al igual que en la celda normal, uno de los nodos se utiliza exclusivamente para capas de pooling. Además, todos los nodos utilizan la salida de la celda inmediatamente anterior, sin utilizar la del nodo input 0.

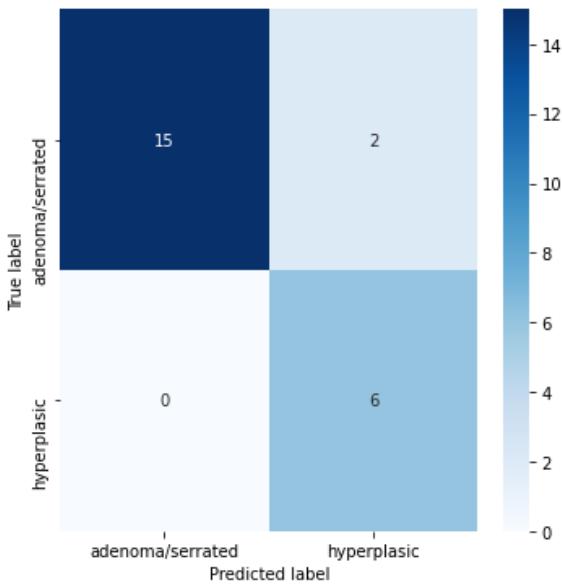


Figura 6.6: Matriz de confusión del modelo ENAS

6.3. Auto-Keras

Para este experimento, se utilizó el algoritmo de Auto-Keras con 200 iteraciones de 50 épocas cada una. Lo primero que se ha notado al utilizar este método comparado con el resto es que es el más “automático”, pues el único parámetro a ajustar es el número máximo de iteraciones. Auto-Keras encontró rápido una red con buen rendimiento y la depuró en las primeras 50 iteraciones. A partir de ahí, fue incapaz de encontrar una arquitectura mejor. Como se puede observar en la Figura 6.7 se obtiene un 87 % de *accuracy* y un 85 % de ponderación. Si nos centramos en la red generada, mostrada en la Figura 6.8, vemos que se ha optado por una red sin capas de pooling, lo cual redunda en un número de parámetros elevado (casi 3 millones). Podemos ver también que la red ha optado por utilizar convoluciones separables en profundidad en vez de capas convolucionales tradicionales.

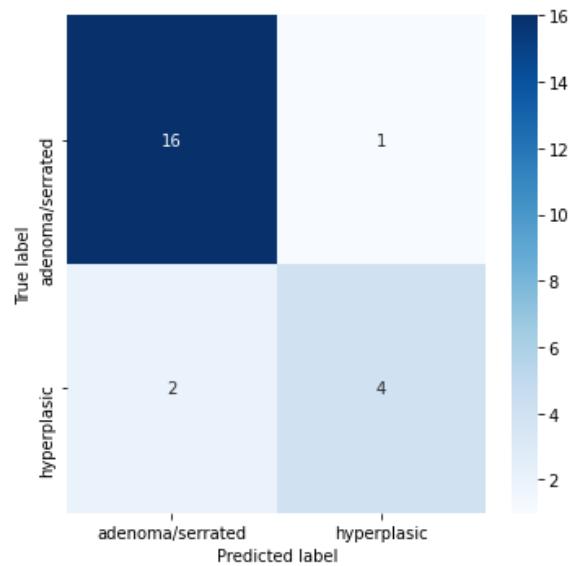


Figura 6.7: Matriz de confusión del modelo Auto-Keras

```
Model: "functional_1"
```

Layer (type)	Output Shape	Param #
<hr/>		
input_1 (InputLayer)	[None, 224, 224, 3]	0
cast_to_float32 (CastToFloat (None, 224, 224, 3))		0
normalization (Normalization (None, 224, 224, 3))		7
separable_conv2d (SeparableC (None, 220, 220, 128))		587
separable_conv2d_1 (Separabl (None, 216, 216, 64))		11456
dropout (Dropout)	(None, 216, 216, 64)	0
flatten (Flatten)	(None, 2985984)	0
dropout_1 (Dropout)	(None, 2985984)	0
dense (Dense)	(None, 1)	2985985
classification_head_1 (Activ (None, 1))		0
<hr/>		
Total params:	2,998,035	
Trainable params:	2,998,028	
Non-trainable params:	7	

Figura 6.8: Arquitectura final generada por Auto-Keras.

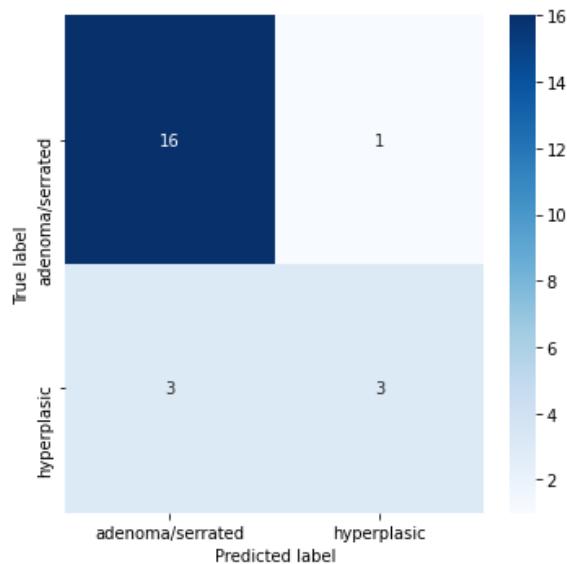


Figura 6.9: Matriz de confusión del modelo Auto CNN

6.4. Auto CNN

Los parámetros de Auto CNN han sido establecidos en 100 generaciones con una población de 10 individuos, una probabilidad de cruce del 90 % y una probabilidad de mutación del 20 %. Cada individuo se entrena durante 10 épocas. Todos estos parámetros fueron escogidos en base a una batería experimental previa. En la Figura 6.9 se puede observar la matriz de confusión obtenida, se consigue un 82 % de *accuracy* y un 80 % de ponderación.

Esta técnica consigue resultados mucho más rápido que el resto y permite explorar el espacio de búsqueda de forma más rápida al inicio. El inconveniente que tiene esta estrategia es el gran overfitting sobre el conjunto de validación. Al seleccionarse los descendientes basándose en la *accuracy* en validación, el modelo tiende a sobreajustar mucho, pues la mejor arquitectura tiende a acaparar toda la población, llegando a un óptimo local del que es imposible escapar sólo con el operador de mutación. Es más, las últimas generaciones del algoritmo, la población estaba formada exclusivamente por copias del mejor modelo, por lo que apenas se mejoró nada.

La arquitectura encontrada está formada por una *skip layers* con 32 filtros en la primera convolución y 128 en la segunda (código 32-128). Se puede ver el resumen en la Figura 6.10. Es destacable que la red planteada no utilice ninguna capa de *pooling*, pues esto ha hecho que la red utilice 6 millones y medio de parámetros.

Model: "functional_1"			
Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[None, 224, 224, 3] 0		
SkipLayer_1/Conv1 (Conv2D)	(None, 224, 224, 32) 896	896	input_1[0][0]
SkipLayer_1/ReLU1 (Activation)	(None, 224, 224, 32) 0	0	SkipLayer_1/Conv1[0][0]
SkipLayer_1/Reshape (Conv2D)	(None, 224, 224, 128) 512	512	input_1[0][0]
SkipLayer_1/Conv2 (Conv2D)	(None, 224, 224, 128) 36992	36992	SkipLayer_1/ReLU1[0][0]
SkipLayer_1/Add (Add)	(None, 224, 224, 128) 0	0	SkipLayer_1/Reshape[0][0] SkipLayer_1/Conv2[0][0]
SkipLayer_1/ReLU2 (Activation)	(None, 224, 224, 128) 0	0	SkipLayer_1/Add[0][0]
dropout (Dropout)	(None, 224, 224, 128) 0	0	SkipLayer_1/ReLU2[0][0]
flatten (Flatten)	(None, 6422528)	6422528	dropout[0][0]
dense (Dense)	(None, 1)	1	flatten[0][0]

Total params: 6,460,929
Trainable params: 6,460,929
Non-trainable params: 0

Figura 6.10: Arquitectura final generada por Auto CNN.

6.5. Discusión

El primer hecho que hay que tener en cuenta a la hora de valorar los resultados es tener en cuenta la poca cantidad de datos de que disponemos: sólo 76 muestras, de las cuales se han separado 23 imágenes como test. Por tanto, es muy probable que los modelos fallen todos más o menos en los mismos casos y compartan sesgos. Además, el desbalanceo de las clases hace que el mínimo de *accuracy* sea de alrededor de un 73 % (si se clasifican todos los casos como positivos).

Podemos observar que todos los métodos NAS empleados superan a la red diseñada por el experto, por lo que podemos considerar que las aproximaciones de NAS representan una alternativa claramente recomendable. Además, todos los modelos parecen haber dado con un óptimo local dado que, en las últimas iteraciones de entrenamiento, apenas mejoraba su rendimiento. Esto sugiere que el refinamiento *a posteriori* de los modelos encontrados puede resultar innecesario.

Como principal desventaja debemos considerar que el tiempo de entrenamiento de todos estos métodos es bastante elevado, aunque debemos de ponerlo en perspectiva con respecto a todo el tiempo que dedica el ingeniero que diseña una red neuronal a mano, pues suele involucrar muchas pruebas que pueden superar con creces el tiempo de estos métodos automáticos. Otra desventaja de estos modelos es su tendencia a diseñar modelos mucho

más grandes que los diseñados por humanos (excepto ENAS). El número excesivamente alto de parámetros causa mayores tiempos de entrenamiento y aumenta la probabilidad de overfitting.

ENAS ha sido la técnica con mejores resultados, tanto en *accuracy* (91.3 %) como en nuestra ponderación (93.7 %). Gran parte de su rendimiento podemos entender que se ha conseguido gracias a disponer de menos parámetros incluso que la red diseñada a mano (35.321 frente a 242.945). Esto ha redundado en un modelo muy pequeño y con menos tendencia al overfitting. Esto se debe, en buena medida, a su diseño basado en microarquitectura y demuestra la importancia vital de la elección de un espacio de búsqueda adecuado. Entre sus desventajas destaca, como ya hemos comentado, el tiempo: ENAS es la más lenta de las tres. Otra gran limitación de este modelo es la gran cantidad de hiperparámetros que se deben de fijar antes de empezar. Es necesario proporcionar el número de bloques y su disposición, el número de nodos de cada bloque y también otros hiperparámetros como el ratio de dropout.

Los resultados de Auto-Keras también han sido satisfactorios con un 85 % de ponderación. Su espacio de búsqueda es el más amplio (utiliza macroarquitectura y muchas opciones de bloques, como explicamos en la Sección 4.6). Por tanto, podemos entender que la optimización bayesiana es una estrategia de búsqueda eficaz a la hora de explorar este espacio de búsqueda. Por otro lado, el hecho de que Auto-Keras no tenga apenas hiperparámetros que ajustar la hacen una técnica muy buena para aquellos ingenieros con poca experiencia diseñando redes, o aquellos que se encuentren con unos datos de los que tengan poca información. A este respecto, podemos concluir que la optimización bayesiana, en general, y Auto-Keras, en particular, son excelentes opciones para generar redes automáticas.

Por último, el algoritmo evolutivo Auto CNN ha sido la técnica que menos tiempo ha tardado en converger: con apenas media hora de entrenamiento ha sido capaz de encontrar una arquitectura con ponderación de 80.4 %, superando la red del experto humano. El espacio de búsqueda de esta técnica es más reducido que en Auto-Keras (sólo cuenta con capas de pooling y *skip layers* de dos capas convolucionales) y por tanto, podríamos imaginar que es una técnica menos versátil. Aún así, su ventaja es el tiempo reducido de entrenamiento y es algo que puede primar a la hora de escoger una técnica u otra. Como posible ampliación de esta técnica, sería muy beneficioso añadir más bloques al espacio de búsqueda, como hace Auto-Keras.

Comparando estos resultados con la ponderación de 78.3 % de la red diseñada por un humano podemos comprobar que las redes generadas con NAS lo han superado ampliamente. Podemos incluso ampliar la comparación a técnicas de ML empleadas en este problema en un trabajo anterior [11]. El estado del arte de estas técnicas se establecía con un AdaBoost que lograba

un 90.8 % de *accuracy* y un 87.8 % de ponderación. Aunque de cara a extraer conclusiones rigurosas se debería emplear exactamente el mismo protocolo de validación experimental³, así como los mismos datos de entrada⁴, sí se puede inferir que los resultados proporcionados por ENAS son competitivos, y de hecho superan a los mejores resultados obtenidos hasta el momento en este problema concreto (clasificación automática de lesiones gastrointestinales).

Como conclusión, podemos recomendar utilizar Auto CNN u otros algoritmos genéticos para obtener una arquitectura de forma rápida y tener una idea de los resultados posibles. Auto-Keras es una buena técnica para obtener buenos resultados sin tener que ajustar demasiados parámetros a mano. Por último, ENAS y el aprendizaje por refuerzo obtienen las mejores redes, de las evaluadas en este trabajo, y se recomienda su uso cuando el objetivo es maximizar el rendimiento.

³En [11] se emplea *leave-one-out*, mientras que aquí se ha utilizado *hold-out* por los motivos presentados en 6

⁴En [11] se emplea información 3D para la clasificación de los pólipos y, por tanto, se usa más de un frame para esta tarea.

Capítulo 7

Conclusiones y Trabajos Futuros

Las técnicas de Búsqueda Automática de Arquitecturas Neuronales, o Neural Architecture Search (NAS), permiten automatizar el diseño de arquitecturas de redes neuronales, haciendo más sencillo y rápido obtener buenos resultados en problemas complejos y acercando el Aprendizaje Automático, o Machine Learning (ML), a personas con poca experiencia. En este trabajo de fin de grado (TFG) se ha realizado un minucioso estudio del campo del NAS, resumiendo todas las corrientes modernas y las publicaciones más relevantes. Tras comparar las opciones más prometedoras del estado del arte, se han implementado y aplicado tres técnicas concretas en un caso de estudio: Efficient Architecture Search (ENAS), una técnica de aprendizaje por refuerzo y micro-arquitectura; Auto-Keras, que hace uso de la optimización bayesiana y morfismo de redes; y Auto CNN, un enfoque evolutivo muy reciente que hace uso tanto de un operador de cruce como de mutación. Estos modelos representan tres de los enfoques más populares y recientes del estado del arte.

La aplicación concreta a la que nos hemos enfrentado es la clasificación automática de pólipos gastrointestinales: un problema clínico al que se enfrentan diariamente los médicos gastroenterólogos, consistente en determinar si una imagen colonoscópica presenta una lesión que debe ser eliminada/reseccionada o no. Nuestros experimentos han partido de un estudio anterior que utilizaba Deep Learning (DL), pero **este es el primer estudio en el que se ha abordado esta aplicación con técnicas de NAS**. Las arquitecturas generadas con nuestros modelos de NAS han dado **mejores resultados que la red diseñada a mano, logrando, en el caso de ENAS, una precisión de 91.3 % frente al 78.3 % de la red diseñada manualmente**. Esto es notable teniendo en cuenta la poca cantidad de datos disponibles, que dificulta obtener buenos resultados al aplicar DL a

este problema. Aún así, hay que destacar que los tiempos de entrenamiento son bastante altos, **llegando a tardar más de 3h en un problema con apenas 76 imágenes**. Además, las arquitecturas que puede generar cada técnica vienen determinadas y limitadas por su espacio de búsqueda. Esto provoca que todas las arquitecturas generadas por una técnica tengan a compartir sesgos, por ejemplo, generar arquitecturas con demasiados parámetros (como ha sido el caso de Auto-Keras y Auto CNN).

Si repasamos los objetivos propuestos inicialmente para este TFG, se puede ver que se han cumplido todos: se ha realizado un estudio pormenorizado de la bibliografía, se han seleccionado cuidadosamente los modelos más representativos para ser utilizados en nuestro caso de estudio, se han implementado y aplicado los modelos seleccionados y, por último, se han realizado los experimentos pertinentes. Todos estos experimentos han requerido una gran labor de adaptación e integración de estas técnicas para enfrentarse al problema concreto de clasificación binaria anteriormente mencionado. El resultado de este trabajo ha sido liberado en un repositorio público de GitHub¹, para contribuir al avance de las investigaciones en este campo.

Como trabajos futuros, proponemos el uso de *data augmentation* pues, generalmente, los modelos basados en DL se ven beneficiados por la presencia de más datos. También proponemos un estudio comparativo más amplio, que incluya más técnicas de DL diseñadas por expertos además de técnicas clásicas de ML. En cuanto a las técnicas de NAS, proponemos trabajar en la mejora de los espacios de búsqueda, para obtener técnicas que generen arquitecturas más diversas y adaptables (por ejemplo, que funcionen adecuadamente en problemas con pocos datos). Otro campo en el que todavía hay mucho margen de mejora es el nivel de automatización del NAS. Como se ha comentado con anterioridad, el objetivo último de NAS es generar redes sin intervención humana. Sin embargo, los modelos existentes todavía necesitan seleccionar valores para numerosos hiperparámetros (número de épocas, número de bloques, etc). Reducir el número de hiperparámetros y, por tanto, el tiempo dedicado a ajustarlos redundaría en hacer estas técnicas más accesibles y eficaces.

¹<https://github.com/alekpinel/TFG-NAS>

Bibliografía

- [1] Y. LeCun, Y. Bengio, and G. Hinton. *Deep learning*. Nature 521.7553 (2015): 436-444.
- [2] J. Schmidhuber, *Deep learning in neural networks: An overview*. Neural networks 61 (2015): 85-117.
- [3] I. Goodfellow, et al. *Deep learning*. Vol. 1. Cambridge: MIT press, 2016.
- [4] A. Krizhevsky, I. Sutskever and G. E. Hinton. *Imagenet classification with deep convolutional neural networks*. NIPS, 2012, 84-90.
- [5] Hinton, Geoffrey, et al. *Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups*. IEEE Signal processing magazine 29.6, 2012, 82-97.
- [6] S. Grigorescu, B. Trasnea, T. Cocias and G. Macesanu, *A survey of deep learning techniques for autonomous driving*, Journal of Field Robotics, vol. 37, no. 3, pp. 362-386, 2020.
- [7] C. M. Bishop, *Neural networks for pattern recognition*. Oxford university press, 1995.
- [8] T. Elsken J. Hendrik Metzen and F. Hutter, *Neural Architecture Search: A Survey*, Journal of Machine Learning Research 20, 1-21, 2019.
- [9] G. Kyriakides and K. Margaritis, *An Introduction to Neural Architecture Search for Convolutional Networks*, Department of Applied Informatics, University of Macedonia, preprint, 2020
- [10] P. Mesejo, D. Pizarro, A. Abergel, O. Rouquette, S. Beorchia, L. Poincloux and A. Bartoli, *Computer-aided classification of gastrointestinal lesions in regular colonoscopy*. IEEE transactions on medical imaging, 35(9), 2051-2063, 2016
- [11] J. Moyano Doña, *Estudio comparativo de métodos de clasificación de lesiones gastrointestinales en vídeos de endoscopia*, Trabajo Fin de Grado - Universidad de Granada, 2020.

- [12] X. He, K. Zhao and X. Chu, *AutoML: A survey of the state-of-the-art*, Knowledge-Based Systems, vol. 212, p. 106622, 2019.
- [13] E. Alpaydin, *Introduction to Machine Learning*, MIT Press, 2020.
- [14] C. M. Bishop, *Pattern Recognition and Machine Learning*, Cambridge: Springer-Verlag New York, 2006.
- [15] M. Jordan and T. Mitchell, *Machine learning: Trends, perspectives, and prospects*, Science, vol. 349, no. 6245, pp. 255-260, 2015. Available: 10.1126/science.aaa8415.
- [16] Y. Abu-Mostafa, *Learning from data*. New York: AMLBooks, 2012.
- [17] A. Faghfouri and M. Frish, *Robust discrimination of human footsteps using seismic signals*, Proceedings of SPIE - The International Society for Optical Engineering, 2011. [Accessed 9 August 2021].
- [18] D. H. Ballard and C. M. Brown, *Computer Vision*, Prentice Hall, 1982
- [19] M. D. Zeiler and R. Fergus, *Visualizing and Understanding Convolutional Networks*, Computer Vision – ECCV 2014 Lecture Notes in Computer Science, vol. 8689, 2013.
- [20] H. Yakura, S. Shinozaki, R. Nishimura, Y. Oyama and J. Sakuma, *Malware Analysis of Imaged Binary Samples by Convolutional Neural Network with Attention Mechanism*, 8th ACM Conference on Data and Application Security and Privacy, 2018.
- [21] K. He, X. Zhang, S. Ren and J. Sun, *Deep Residual Learning for Image Recognition*, Proc. Computer Vision and Pattern Recognition (CVPR), 2016
- [22] G. Huang, Z. Liu, L. van der Maaten and K. Weinberger, *Densely Connected Convolutional Networks*, CVPR, 2017.
- [23] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R. R. Salakhutdinov, *Improving neural networks by preventing co-adaptation of feature detectors*, 2012.
- [24] H. Sultan, N. Salem and W. Al-Atabany, *Multi-Classification of Brain Tumor Images Using Deep Neural Network*, IEEE Access, vol. 7, pp. 69215-69225, 2019.
- [25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, (1986). *Learning representations by back-propagating errors*. Nature, 323(6088), 533-536

- [26] V. Sharma, *Deep Learning – Introduction to Convolutional Neural Networks — Vinod Sharma’s Blog*, Vinod Sharma’s Blog, 2018. [Online]. Available: <https://vinodsblog.com/2018/10/15/everything-you-need-to-know-about-convolutional-neural-networks/>. [Accessed: 15- Jul- 2021].
- [27] A. Rahman, *Neural Architecture Search (NAS)- The Future of Deep Learning*, Theaiacademy.blogspot.com, 2020. [Online]. Available: <https://theaiacademy.blogspot.com/2020/05/neural-architecture-search-nas-future.html>. [Accessed: 15- Jul- 2021].
- [28] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing. *Neural architecture search with bayesian optimisation and optimal transport*, Advances in Neural Information Processing Systems, pages 2016-2025, 2018.
- [29] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L. Li, L. Fei- Fei, A. Yuille, J. Huang, and K. Murphy, *Progressive Neural Architecture Search*, European Conference on Computer Vision, 2018.
- [30] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, *Hierarchical Representations for Efficient Architecture Search*, International Conference on Learning Representations, 2018.
- [31] D. Omid E. and G. Iddo, *Genetic Algorithms for Evolving Deep Neural Networks*, 2014.
- [32] X. Yao, *Evolving artificial neural networks*, Proceedings of the IEEE, vol. 87, no. 9, pp. 1423-1447, 1999. Available: 10.1109/5.784219.
- [33] H. Pham, M. Guan, B. Zoph, Q. Le and J. Dean, *Efficient Neural Architecture Search via Parameter Sharing*, ICML, 2018. [Accessed 6 July 2021].
- [34] K. O. Stanley, J. Clune, J. Lehman and R. Miikkulainen, *Designing neural networks through neuroevolution*, Nature Machine Intelligence, vol 1, 2019
- [35] F.Petroski Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley and J. Clune, *Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning*, arXiv <https://arxiv.org/abs/1712.06567>, 2018
- [36] D. Floreano, P. Dürr and C. Mattiussi, *Neuroevolution: from architectures to learning*, Evolutionary Intelligence, vol. 1, no. 1, pp. 47-62, 2008. Available: 10.1007/s12065-007-0002-4.

- [37] M. Suganuma, M. Kobayashi, S. Shirakawa and T. Nagao, *Evolution of Deep Convolutional Neural Networks Using Cartesian Genetic Programming*, Evolutionary Computation, vol. 28, no. 1, pp. 141-163, 2020. Available: 10.1162/evco_a_00253.
- [38] J. Cui et al., *Fast and Practical Neural Architecture Search*, 2019 IEEE/CVF International Conference on Computer Vision, 2019.
- [39] B. Zoph and Q. Le, *Neural Architecture Search with Reinforcement Learning*, ICLR, 2017. [Accessed 19 November 2020].
- [40] B. Zoph, V. Vasudevan, J. Shlens and Q. Le, *Learning Transferable Architectures for Scalable Image Recognition*, IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018. [Accessed 16 November 2020].
- [41] K. Swersky, J. Snoek, and R. P. Adams. *Freeze-thaw bayesian optimization*, 2014.
- [42] K. Stanley and R. Miikkulainen, *Evolving Neural Networks Through Augmenting Topologies*, Evolutionary Computation, vol. 10, pp. 99-127, 2002. Available: <http://nn.cs.utexas.edu/?stanley:ec02>. [Accessed 26 November 2020].
- [43] H. Jin, Q. Song and X. Hu, *Auto-Keras: An Efficient Neural Architecture Search System*, the 25th ACM SIGKDD International Conference, 2019.
- [44] W. Wang, *Bayesian Optimization Concept Explained in Layman Terms*, towardsdatascience, 2018. [Online]. Available: <https://towardsdatascience.com/bayesian-optimization-concept-explained-in-layman-terms-1d2bcdeaf12f>. [Accessed: 27- Nov- 2020].
- [45] W. Koehrsen, *A Conceptual Explanation of Bayesian Hyperparameter Optimization for Machine Learning*, towardsdatascience, 2018. [Online]. Available: <https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f>. [Accessed: 27- Nov- 2020].
- [46] C. White, *An Introduction to Bayesian Optimization for Neural Architecture Search*, Abacus.AI, 2019. [Online]. Available: <https://medium.com/abacus-ai/an-introduction-to-bayesian-optimization-for-neural-architecture-search-d324830ec781>. [Accessed: 29- Jul- 2021].
- [47] Z. Zhong et al., *BlockQNN: Efficient Block-Wise Neural Network Architecture Generation*, IEEE Transactions on Pattern Analysis and Machi-

- ne Intelligence, vol. 43, no. 7, pp. 2314-2328, 2021. Available: 10.1109/t-pami.2020.2969193.
- [48] G.F. Miller, P.M. Todd, and S.U Hedge, *Designing neural networks using genetic algorithms*, 3rd International Conference on Genetic Algorithms (ICGA'89), 1989.
 - [49] D. E. Moriarty and R. Miikkulainen, *Forming neural networks through efficient and adaptive coevolution*, Evolutionary computation, vol. 5, pp. 373-399, 1997.
 - [50] G. Bender. *Understanding and simplifying one-shot architecture search* 2019.
 - [51] T. Elsken, J.-H. Metzen, and F. Hutter, *Simple and efficient architecture search for convolutional neural networks*, 2017.
 - [52] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. *Efficient architecture search by network transformation*. 32nd AAAI Conference on Artificial Intelligence, AAAI 2018, pages 2787-2794, 2018.
 - [53] F. Assunç, A. Assunção, N. Lourenço, P. Machado, and B. Ribeiro. *Denser: deep evolutionary network structured representation*, Genetic Programming and Evolvable Machines, 2019.
 - [54] A. Guha, S. Harp and T. Samad, *Genetic synthesis of neural networks*, Honeywell Corporate Systems Development Division, vol. -88-4852-1, 1988.
 - [55] R. Todd, *Evolutionary methods for connectionist architectures*, Psychology Department, Stanford University, unpublished manuscript, 1988.
 - [56] D. Whitley, T. Starkweather and C. Bogart, *Genetic algorithms and neural networks: optimizing connections and connectivity*, Parallel Computing, vol. 14, no. 3, pp. 347-361, 1990.
 - [57] D. Montana and L. Davis, *Training feedforward neural networks using genetic algorithms*, Proc. of Eleventh Int'l Joint Conf. on Artificial Intelligence, Morgan Kaufmann, San Mateo, CA, pp. 762-767, 1989
 - [58] D. White and P. Ligomenides, *GANNet a genetic algorithm for optimizing topology and weights in neural network design*, Proc. of Int'l Workshop on Artificial Neural Networks (IWANN'93), Springer-Verlag, Lecture Notes in Computer Science, vol. 686, pp. 322-327, 1993.
 - [59] E. Alba, J. Aldana and J. Troya, *Fully automatic ANN design: a genetic approach*, Proc. of Int'l Workshop on Artificial Neural Networks (IWANN'93), Springer-Verlag, Lecture Notes in Computer Science, vol. 686, pp. 399-404, 1993.

- [60] M. Sandler, A. Howard, Menglong Zhu, A. Zhmoginov, and L. Chen, *Mobilenetv2: Inverted residuals and linear bottlenecks*, CVPR, 2018.
- [61] M. Al Harrach, *Modeling of the sEMG/Force relationship by data analysis of high resolution sensor network*, Université de Rennes, 2016. [Accessed: 07- Jul- 2021].
- [62] A. Liam Li, *What is neural architecture search?*, O'Reilly Media, 2018. [Online]. Available: <https://www.oreilly.com/content/what-is-neural-architecture-search/>. [Accessed: 07- Jul- 2021].
- [63] A. S. Fauci, E. Braunwald, K. J. Isselbacher, J. B. Martin, D. L. Kasper, S. L. Hauser, D. L. Longo. *Harrison medicina interna*, Harrison medicina interna, pp. 1466-1466, 1998
- [64] *They found colon polyps: Now what?*, Harvard Health, 2019. [Online]. Available: <https://www.health.harvard.edu/diseases-and-conditions/they-found-colon-polyps-now-what>. [Accessed: 06- Jul- 2021].
- [65] A. Veit and S. Belongie, *Convolutional networks with adaptive inference graphs*, ECCV, 2018.
- [66] Y. Sun, B. Xue, M. Zhang, G. Yen and J. Lv, *Automatically Designing CNN Architectures Using the Genetic Algorithm for Image Classification*, IEEE Transactions on Cybernetics, vol. 50, no. 9, pp. 3840-3854, 2020.
- [67] S. Hochreiter and J. Schmidhuber, *Long Short-term Memory*, Neural Computation 9(8):1735-80, 1997.
- [68] R. Karim, *Illustrated: Efficient Neural Architecture Search*, Medium, 2019. [Online]. Available: <https://towardsdatascience.com/illustrated-efficient-neural-architecture-search-5f7387f9fb6>. [Accessed: 15- Jul- 2021].
- [69] C. Olah, *Understanding LSTM Networks*, Colah.github.io, 2015. [Online]. Available: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>. [Accessed: 29- Jul- 2021].
- [70] ENAS — An open source AutoML toolkit for neural architecture search, model compression and hyperparameter tuning, Nni.readthedocs.io. [Online]. Available: <https://nni.readthedocs.io/en/stable/NAS/ENAS.html>. [Accessed: 29- Jul- 2021].
- [71] P. Hart, N. Nilsson and B. Raphael, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, 1968.

- [72] K. Stanley, D. D'Ambrosio and J. Gauci, *A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks*, Artificial Life, vol. 15, no. 2, pp. 185-212, 2009.
- [73] A Greff, Klaus, et al. *LSTM: A search space odyssey*. IEEE transactions on neural networks and learning systems 28.10, 2016.