SEP TNM

INSTITUTO TECNOLÓGICO DE CULIACÁN

CONSTRUCCIÓN DE UN MODELO AUTOMÁTICO DE REGRESIÓN Y CLASIFICACIÓN EN UN AMBIENTE DISTRIBUIDO

TESIS

PRESENTADA ANTE EL DEPARTAMENTO ACADÉMICO DE ESTUDIOS DE POSGRADO DEL INSTITUTO TECNOLÓGICO DE CULIACÁN EN CUMPLIMIENTO PARCIAL DE LOS REQUISITOS PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

POR:

ING. MANUEL ALEJANDRO MEDRANO DÍAZ
INGENIERO EN SISTEMAS COMPUTACIONALES

DIRECTOR DE TESIS:

DR. HÉCTOR RODRÍGUEZ RANGEL

CO-DIRECTOR DE TESIS:

DR. Juan José Flores

CULIACÁN, SINALOA 5 de Agosto del 2021

"2021, Año de la Independencia"

Culiacán, Sin., 10 de Agosto del 2021

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
OFICIO: DEPI/260/8/2021

ASUNTO: **Autorización Impresión**

C. MANUEL ALEJANDRO MEDRANO DÍAZ
ESTUDIANTE DE LA MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN
PRESENTE.

Por medio de la presente y en virtud de que ha completado los requisitos para el examen de grado de **Maestro en Ciencias de la Computación**, se concede autorización para la impresión de la tesis titulada: **"CONSTRUCCIÓN DE UN MODELO AUTOMÁTICO DE REGRESIÓN Y CLASIFICACIÓN EN UN AMBIENTE DISTRIBUIDO"** bajo la dirección del(a) **Dr. Héctor Rodríguez Rangel.**

Sin otro particular reciba un cordial saludo.

ATENTAMENTE
Excelencia en Educación Tecnológica®

M.C. MARÍA ARACELY MARTÍNEZ AMAYA
JEFE(A) DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE CULIACÁN
DEPARTAMENTO DE DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

C.c.p. archivo

MAMA/lucy *

"CONSTRUCCIÓN DE UN MODELO AUTOMÁTICO DE REGRESIÓN Y CLASIFICACIÓN EN UN AMBIENTE DISTRIBUIDO"

Tesis presentada por el(a):

C. MANUEL ALEJANDRO MEDRANO DÍAZ

Aprobada en contenido y estilo por:

Dr. Héctor Rodríguez Rangel
Director de Tesis

Dr. Ramón Zatarain Cabada
Secretario

M.C. Gloria Ekaterine Peralta Peñúñuri
Vocal -1

M.C. Ricardo Rafael Quintero Meza
Vocal -2

M.C. María Aracely Martínez Amaya
Jefe(a) de la División de Estudios de Posgrado e Investigación

Dedicatoria

Dedico el presente trabajo a mis padres Jaime Medrano Camacho y Luz María Díaz León, y mis hermanos Jaime Demetrio Medrano Díaz y María Fernanda Medrano Díaz. Junto con ellos he podido lograr muchos de mis objetivos de vida, siendo uno de estos, mis estudios de maestría. Gracias a ellos sé que tengo la habilidad y mentalidad para lograr cualquier cosa que me proponga. Les agradezco mucho pues han sido de un gran apoyo emocional durante toda mi vida y carrera profesional, tanto de licenciatura, como estudios de posgrado. Han estado junto a mí en todo momento y situación difícil. No me queda más que agradecerles por el apoyo infinito que me han otorgado todos estos años, gracias por todo.

Agradecimientos

Agradezco a mis **padres y hermanos** por su apoyo durante toda la vida, sobre todo en mis estudios profesionales.

A mis **compañeros y amigos** que me brindaron un gran apoyo moral en los buenos y malos momentos que he vivido.

Al **Dr. Héctor Rodríguez Rangel** por su apoyo y su guía durante la realización de este trabajo y residencias profesionales de licenciatura.

Al **Dr. Juan José Flores** por brindarme su conocimiento e ideas innovadoras para realizar este proyecto.

Al **Dr. Víctor González** por ser un docente apasionado por el aprendizaje y aplicación de tecnologías innovadoras e ir siempre a la vanguardia.

A la **Dra. Lucía Barrón** por ser un ejemplo de como debe ser una investigadora por excelencia llena de conocimiento y pasión.

Al **Dr. Ramón Zatarain** por compartir su experiencia en la investigación de proyectos.

Al **Dr. Ricardo Quintero** por demostrar una gran pasión por la docencia al momento de compartir su conocimiento.

A las maestras **Ekaterine Peralta y Lucy López** por su gran gestión dentro del departamento de posgrado, siempre llevando un excelente control de trámites.

Al **Tecnológico Nacional De México Campus Culiacán** por ofrecer un espacio de aprendizaje que permitió mi formación profesional de licenciatura y maestría.

Al **Consejo Nacional de Ciencia y Tecnología (CONACyT)** por su apoyo financiero que me permitió permanecer enfocado en mis estudios de maestría.

Declaración de autenticidad

Por la presente, declaro que, salvo cuando se haga referencia específica al trabajo realizado por otras personas, el contenido del presente trabajo de tesis es original y no se ha presentado total o parcialmente para su consideración para cualquier otro título o grado en esta o en cualquier otra universidad. Esta tesis es el resultado de mi propio trabajo y esfuerzo, y no incluye nada que sea resultado de algún trabajo realizado en colaboración, salvo que se indique específicamente en el texto.

Manuel Alejandro Medrano Díaz. Culiacán, Sinaloa, México, 2021.

Resumen

El aprendizaje profundo ha recibido mucha atención en años recientes debido principalmente a su capacidad de aprender características y patrones de gran complejidad a partir de grandes conjuntos de datos. Logrando de esta manera, automatizar tareas que antes solo eran posible ser realizadas con intervención humana. Sin embargo, desarrollar este tipo de tecnología requiere un alto nivel de conocimiento informático avanzado. Además se espera una importante inversión de tiempo para lograr adaptar y optimizar cada problema de manera manual.

En la actualidad, existen cada vez más esfuerzos para lograr automatizar las tareas de diseño de arquitecturas y optimización de hiperparámetros de modelos, creando así el área llamada Aprendizaje Máquina Automatizado (AutoML). Esta área engloba de manera particular o en conjunto las técnicas de Optimización de Hiperparámetros (HPO) y la Búsqueda de Arquitectura Neuronal (NAS).

El presente proyecto se encuentra ubicado en el área de NAS. Donde, su estructura consiste en la implementación de 3 partes, las cuales son: Espacio de búsqueda, Estrategia de búsqueda y Estrategia de estimación de rendimiento. La anterior estructura presentada ha sido implementada en múltiples trabajos que se presentan en el estado del arte. En la primera parte de NAS se implementa un espacio de búsqueda con estructura de árbol, posteriormente la estrategia de búsqueda es compuesta por el algoritmo bayesiano TPE y un salón de la fama para clasificar modelos de alto rendimiento. Finalmente, se implementa la estrategia de estimación de rendimiento en la fase de entrenamiento parcial y la fase de entrenamiento profundo. Lo anterior planteado está diseñado para que se pueda implementar en un ambiente de sistemas distribuidos.

El sistema fue desarrollado para soportar la búsqueda de tareas de clasificación de imágenes, regresión vectorial, series de tiempo y pronóstico de series de tiempo de imágenes. A su vez, fueron realizadas múltiples pruebas con diferentes conjuntos de datos tomados de diferentes fuentes. El sistema ofrece métricas de exactitud para clasificación y pérdida para las demás tareas, además presenta una alta eficiencia en tiempo invertido y uso de hardware, así como alta flexibilidad para ser usado por diferentes tipos de usuario en diferentes equipos con configuraciones variadas y sistemas operativos.

Palabras clave

- Inteligencia artificial
- Aprendizaje máquina
- Aprendizaje profundo
- Aprendizaje máquina automatizado (AutoML)
- Optimización de hiperparámetros
- Búsqueda de arquitectura neuronal (NAS)
- Optimización bayesiana
- Clasificación de imágenes
- Regresión vectorial
- Series de tiempo vectorial
- Pronóstico de series de tiempo de imágenes
- Sistemas distribuidos

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En los tiempos actuales, la aplicación de técnicas de inteligencia artificial como las redes neuronales y el aprendizaje profundo han tenido cada vez más impacto en la tecnología y en la sociedad. Gran parte del éxito de estas técnicas se deben al amplio uso de aplicaciones que pueden tener. Tales como: reconocimiento de rostros, tareas automatizadas, recomendaciones personalizadas, mejoras en eficiencia de sistemas, entre otras. Mientras que otros métodos, como los estadísticos tradicionales, no pueden utilizar de manera satisfactoria tipos múltiples de información de manera simultánea. Las técnicas de aprendizaje profundo han demostrado la capacidad de aprender patrones a partir de diferentes tipos de datos (Benediktsson et al., 1990).

A pesar de los beneficios que ofrecen las técnicas de aprendizaje máquina, estas presentan una variedad de dificultades para ser aplicadas. Una de las principales consiste en la necesidad de realizar una gran cantidad de configuraciones de arquitecturas y parámetros para obtener un modelo que se ajuste apropiadamente al problema. Cada problema presenta un conjunto de características que afectan de manera directa las características de la solución que pueda resolverlo de manera apropiada (Zoph & Le, 2016).

El objetivo de este proyecto de investigación consiste en la automatización del diseño y optimización de arquitecturas e hiperparámetros de modelos de aprendizaje profundo mediante la implementación de un algoritmo bayesiano. Este tipo de técnicas de optimización recaen en el área llamada Aprendizaje Máquina Automatizado (AutoML) y su proceso principal consiste en los siguientes pasos: Preparación de datos, ingeniería de características, generación del modelo y la evaluación del modelo (He et al., 2021). Mediante esta metodo-

logía es posible resolver problemas de clasificación, regresión y series de tiempo tanto de manera vectorial como matricial.

En este proyecto se cubren cada uno de los pasos anteriormente descritos, mediante diferentes procesos. Para la preparación de los datos se implementa un proceso de normalización para mantener los datos en un rango entre 0 y 1. El siguiente paso es la ingeniería de características, donde se implementa una búsqueda con una estructura de tipo árbol, que mediante la implementación de un algoritmo bayesiano, promete generar modelos de acuerdo con los resultados obtenidos en los modelos previos. Por último, la estrategia de evaluación del modelo se implementa con un formato distribuido empleando diferentes equipos de cómputos y tarjetas gráficas (GPUs).

La metodología propuesta en este trabajo fue probada en diferentes condiciones con conjuntos de datos, tanto pequeños como grandes, acondicionados y no acondicionados. El equipo de cómputo utilizado para realizar el aprendizaje máquina automatizado, cuenta con dos GPU para simular la implementación distribuida de la optimización. De igual manera, las pruebas de pronóstico de series de tiempo de imágenes fueron probadas con diferentes tamaños de codificación en el mismo equipo.

El presente proyecto es una mejora y extensión de un proyecto más grande que involucra la generación de modelos optimizados. El trabajo previo fue realizado por el Maestro en Ciencias de la Computación Mario Valenzuela Partida titulado “Sistema de diseño y optimización de redes convolucionales por razonamiento bayesiano utilizando entrenamientos parciales y distribuidos”. Donde involucra principalmente la generación de soluciones a clasificación de imágenes por medio de bloques VGG e *Inception*. Para este trabajo, se agregó una interfaz gráfica en donde se buscó tener una aplicación amigable y se agregaron los módulos correspondientes para la resolución de problemas de regresión vectorial y matricial.

1.1. Definición del problema

En la actualidad cada vez más sistemas implementan la utilización de técnicas de inteligencia artificial. Lo cual provoca una alta demanda para dar solución a problemas académicos, sociales e industriales. Sin embargo, la aplicación de estas técnicas presenta un grado

de dificultad alto, pues es requerido tener conocimiento en el área de computación y del aprendizaje máquina.

Existen diversos tipos de modelos de aprendizaje máquina, por lo que se vuelve complicado hacer una selección adecuada si no se cuenta con experiencia previa en el diseño de modelos. Lo anterior se debe a que cada técnica o método de aprendizaje máquina suelen tener distintas ventajas y desventajas en los diferentes tipos de problemas. Entre los tipos de modelos más utilizados se encuentran las Redes Neuronales de Convolución (CNN por sus siglas en Inglés) que se especializan a problemas de manipulación de imágenes y las Redes de Memoria de Corto Plazo (LSTM por sus siglas en Inglés) que trabajan especialmente mejor para datos de secuencias, por ejemplo, series de tiempo, procesamiento de lenguaje natural o reconocimiento de voz.

De manera tradicional, el diseño de arquitecturas es realizados por expertos con conocimientos en la inteligencia artificial y el dominio del problema. El proceso tradicional consiste en que el experto primero haga el diseño de la arquitectura a partir de su experiencia, y mediante un proceso de prueba y error, va iterando hasta encontrar la arquitectura e hiperparámetros adecuados de un modelo. Este procedimiento suele ser muy tardado de aplicar y no hay una garantía de que se consigan buenos resultados. Por lo tanto, surge la necesidad de desarrollar un proceso automático de diseño y optimización de arquitecturas de aprendizaje máquina.

En la actualidad ya existen soluciones a este problema de optimización automática (i.e. AutoML, Auto-keras, DataRobot, entre otros), aunque estas soluciones suelen presentar limitaciones que dificultan su implementación. La principal dificultad consiste en la gran cantidad de poder de cómputo que se requiere para su implementación, la cual puede no estar disponible para el ciudadano común, además de que su costo puede ser muy elevado. Otras soluciones se presentan en forma de servicio, sin embargo, estos servicios pueden presentar altos costos y ofrecen poca transparencia en los métodos utilizados para la construcción del modelo.

1.2. Hipótesis

Mediante la implementación de espacios de búsqueda múltiples, llevados por una estructura de árbol, y la aplicación de un algoritmo bayesiano, es posible generar modelos múltiples que, con un entrenamiento parcial, proporcionen estimaciones de rendimiento en un corto periodo de tiempo y posteriormente puedan ser clasificados con un salón de la fama para la selección de los mejores que pasaran a un entrenamiento completo, generando así un modelo optimizado especifico para un problema de regresión.

1.3. Objetivo

Diseñar e implementar un sistema capaz de construir de manera automática modelos de aprendizaje máquina específicos para tareas de clasificación y regresión vectorial y matricial utilizando técnicas de auto aprendizaje máquina con un enfoque distribuido.

1.3.1. Objetivos específicos

- Generar modelos de alta precisión específicos para cada problema.
- Optimizar modelos apropiados para clasificación, regresión, series de tiempo (vectorial) y generación de imágenes (matricial).
- Reducir los tiempos de optimización mediante la implementación de una arquitectura distribuida y entrenamientos parciales.
- Diseñar e implementar una interfaz gráfica que facilite el proceso de optimización para usuarios no expertos.
- Desarrollar una aplicación que permita la generación de pronósticos de series de tiempo de imágenes.

1.4. Justificación

La optimización de modelos de aprendizaje máquina son una de las principales áreas dentro de la inteligencia artificial, pues se encarga de la mejora de dichos modelos. De tal manera que, se encuentran soluciones óptimas a los diferentes problemas que se enfrentan. Aunque en la actualidad se cuentan con soluciones a estos problemas, algunas de estas soluciones son privadas y requieren una subscripción de paga para ser utilizados, pero cuentan con excelentes resultados y un cierto grado de facilidad de uso. Un ejemplo es AutoML de Google que cobra $3.15 dólares por hora de procesamiento, Por lo que si se realiza un entrenamiento de 10 horas, se tendrá que hacer un pago de 35 dólares por un modelo al cual no se tendrá un acceso directo, además se hace un cobro por cada 1000 predicciones de $3 dólares .

Las soluciones actualmente desarrolladas siguen siendo un impedimento a utilizar para algunos equipos de trabajo por su mismo costo. Además, otros equipos que tengan un capital suficiente, pueden presentar limitaciones por algún tipo de política y seguridad que no permitan que proveedores externos tengan algún control sobre la información. Aquí es donde entran las soluciones de código abierto, debido a que ayudan a estos equipos con su desarrollo, manteniendo su sistema cerrado y acercándose al área de la inteligencia artificial.

Muchos equipos o personas, ya sea de ámbito académico y empresarial, suelen tener a su disposición equipos de cómputo que no estén a tiempo completo en uso, de tal manera que, para aprovechar el hardware existente, se implementa un procesamiento distribuido entre múltiples computadores para reducir la carga. De tal manera que, se puedan encontrar modelos con alta aptitud utilizando un amplio espacio de búsqueda de arquitecturas para diferentes tipos de problemas.

Gran parte del éxito de las técnicas de aprendizaje máquina se debe a la colaboración que son capaces de aplicar con otras áreas de conocimiento. Estas áreas son capaces de generar un gran conjunto de datos propios, los cuales pueden obtener información valiosa para ser implementada. Esta información en conjunto con el aprendizaje máquina es capaz de generar conocimiento valioso para la ciencia. Mediante la aplicación de la solución planteada se ampliará el espectro de personas expertas en otras áreas, teniendo la posibilidad de acceder a estas nuevas características, y ampliar así, su área de conocimiento.

1.5. Estructura de la tesis

El presente trabajo está dividido en 6 capítulos, incluyendo este capítulo de introducción, y se organiza de la siguiente forma:

- Capítulo 1.- Introducción: Se presentó y abordo la definición del problema al que se enfrenta este trabajo, seguido de la hipótesis y objetivos que esperan resolver el problema presentado, y finalmente, se presenta la justificación del ”por que”se realizó este trabajo.

- Capítulo 2.- Marco Teórico: Se describe la teoría fundamental de los temas principales que conforman el proyecto, como la inteligencia artificial, aprendizaje máquina, aprendizaje profundo, tipos de aprendizaje, algoritmos de optimización y sistemas distribuidos.

- Capítulo 3.- Estado del arte: Se presentan las investigaciones y propuestas actuales que se encuentran en desarrollo de temas relacionados, como el aprendizaje máquina automatizado y optimización de hiperparámetros, presentando un resumen de la metodología que implementan y sus resultados que servirán a modo de comparación.

- Capítulo 4.- Metodología: Se describe el proceso de desarrollo que se tuvo para la creación del sistema y los argumentos lógicos que se tomaron para las decisiones de diseño, como el preprocesamiento de la información, el procedimiento de optimización de modelos y el diseño de los espacios de búsqueda, así como la implementación de las herramientas tecnológicas utilizadas.

- Capítulo 5.- Análisis de resultados: Se muestran los resultados obtenidos del sistema en los diferentes módulos implementados junto con los diferentes conjuntos de datos utilizados, se analiza los resultados de la perdida de información y el tiempo de optimización tomado para obtenerlo.

- Capítulo 6.- Conclusiones: Se describe un resumen de la metodología implementada en el proyecto y de la aportación que ofrece al área del aprendizaje máquina auto-

matizado. Este capítulo es complementado con algunas ideas de mejoras que podrían implementarse, en este proyecto o similares, a futuro.

Capítulo 2

Marco teórico

En este capítulo se describen una serie de conceptos, los cuales presentan la información necesaria en la cual está basado el presente trabajo. El capítulo comienza con conceptos básicos y generales para después introducir conceptos específicos a este trabajo. Inicialmente se comienza con el concepto de la inteligencia artificial y su historia. Después, se definen una serie de conceptos sobre el aprendizaje máquina. Se avanza a una serie de conceptos que definen parte del aprendizaje profundo y posteriormente se definirán algunos conceptos sobre el aprendizaje máquina automatizado. Finalmente, se incluye el tema de sistemas distribuidos y las aplicaciones utilizadas.

2.1. Inteligencia artificial

Para llegar a la definición de inteligencia artificial, es necesario primero definir que es la inteligencia. La inteligencia es la habilidad de alcanzar objetivos que afecten el mundo que lo rodea (McCarthy, 1998). Con lo anterior se puede definir la Inteligencia Artificial (IA) como una ciencia que es capaz de crear máquinas que realicen una o más tareas que requieran inteligencia como si fueran realizadas por humanos (Negnevitsky & Intelligence, 2005).

La perspectiva moderna toma en consideración que cada vez que se habla de IA, se refieren a máquinas que son capaces de realizar una o más tareas. Tales como: Entender el lenguaje humano, realizar tareas mecánicas que involucren maniobras complejas, resolver problemas complejos basados en computadora que posiblemente involucren grandes volúmenes de datos en un tiempo corto y que retorne respuestas de manera humana (Joshi, 2020).

2.1.1. Historia de la Inteligencia Artificial

En 1943 se plantea el primer trabajo reconocido en el área de Inteligencia Artificial (IA) por Warren McCulloch y Walter Pitts. Mediante una investigación en el sistema nervioso central, realizaron el primer modelo de neuronas cerebrales. El modelo propuesto consiste en que cada neurona inicia con un estado binario, en cualquier condición encendido o apagado. De esta manera sé probo que cualquier función computable puede ser computada por alguna red de neuronas conectadas (Negnevitsky & Intelligence, 2005).

Otro de los fundadores de la IA es el matemático Húngaro John Von Neumann. Mediante el apoyo y financiamiento a dos estudiantes graduados del departamento de matemáticas de Princeton, Marvin Minsky y Dean Edmons, construyeron la primera computadora de redes neuronales en el año de 1951.

Otro de los investigadores de la primera generación fue Claude Shannon. En 1950 pública un artículo sobre máquinas que juegan ajedrez. Demostrando que, aunque la computadora de Von Neumann pudiera examinar un movimiento por microsegundo, tomaría $3x10^{106}$ años para realizar el primer movimiento. De esta manera se demuestra la necesidad de heurísticas para la búsqueda de soluciones.

En 1956 John McCarthy convence a Martin Minsky y Claude Shannon a organizar el primer verano de investigación de la IA en el Dartmouth College. Trayendo investigadores interesados en el estudio de inteligencia máquina, redes neuronales artificiales y teoría de autómatas. Se reunieron 10 investigadores que dieron el nacimiento a una nueva ciencia llamada inteligencia artificial.

En los años de 1960s, los investigadores intentaron simular el proceso de pensamiento complejo, mediante la invención de métodos generales para resolver amplias clases de problemas. Utilizando el mecanismo de búsqueda de propósito general para buscar una solución al problema. Tales acercamientos, se les conoce ahora como métodos débiles, aplicando información débil acerca del dominio del problema, el resultado obtenido fue un rendimiento débil de los programas desarrollados.

Sin embargo, para los años 1970s la euforia inicial de la IA se fue, los gobiernos cancelaron las fundaciones a proyectos de IA debido a que no tenían una aplicación fuera de jugar

juegos. Existen múltiples razones para que esto pasara, siendo las principales: los investigadores de la IA desarrollaban métodos generales para una gran gama de diferentes problemas; también muchos problemas que se intentaban resolver fueron demasiado amplios y difíciles; Por último, en 1971 el gobierno británico suspendió el apoyo a las investigaciones de la IA.

Durante los años 1970s se desarrolló una tecnología llamada sistemas expertos. Estos sistemas son capaces de hacer inferencias simples dada una base de hechos y un conjunto de reglas definidas por un experto humano. Estos sistemas tuvieron un número creciente de aplicaciones a finales de los 1970s, demostrando que la tecnología de IA podría pasar con éxito de los laboratorios de investigación a un ambiente comercial. Aunque al inicio la necesidad de hardware costoso y lenguajes de programación complicados se dejó en manos de desarrolladores expertos. En los años 1980s, con la llegada de las computadoras personales (PC) y el desarrollo de herramientas de sistemas expertos fáciles de usar, permitieron que ingenieros e investigadores de todas las disciplinas tuvieran la oportunidad de desarrollar sistemas expertos.

En el año de 1986 se desarrolla el algoritmo de aprendizaje de propagación hacia atrás. Este fue el algoritmo que se convirtió en la técnica más popular para el entrenamiento de perceptrones multicapa (redes neuronales). Lo anterior combinado con las limitaciones de los sistemas expertos, sobre que no todo conocimiento puede ser expresado con base en reglas, permitió que en los años 1990s se empezaron a utilizar las redes neuronales como alternativas para la extracción de conocimiento escondido en largos conjuntos de datos.

2.2. Aprendizaje máquina

El término “Aprendizaje Máquina” fue introducido por Arthur Samuel en 1959. Se refiere a un programa de computadora que puede aprender a producir un comportamiento que no está explícitamente programado por el autor del programa. Incluso puede ser capaz de mostrar un comportamiento que el autor puede desconocer por completo (Joshi, 2020).

El aprendizaje máquina involucra una serie de mecanismos adaptativos que permite a las computadoras a aprender de la experiencia, ejemplo y analogía. Las capacidades de aprendizaje pueden ir mejorando en el rendimiento de un sistema inteligente sobre el tiempo (Neg-

nevitsky & Intelligence, 2005). Dicho de una manera más formal se encuentra la siguiente definición: “Un programa de computadora se dice que aprende de la experiencia *E* con respecto a una clase de tareas *T* y medida de rendimiento *P*, si su rendimiento con respecto a las tareas *T*, medidas por *P*, mejoran con la experiencia *E*” (Mitchell et al., 1997).

En el aprendizaje máquina, el aprendizaje ocurre extrayendo la mayor cantidad de información del conjunto de datos, también llamado comúnmente *Dataset*. A través de algoritmos que analizan la estructura base de los datos y distinguen las señales del ruido, encuentran la señal, o el patrón que definirá su salida (Conway & White, 2012).

Estos patrones o comportamiento son aprendidos basándose en tres factores (Joshi, 2020):

1. La información o datos que es consumida por el programa.

2. Una métrica que cuantifica el error o alguna forma de distinguir la distancia entre el comportamiento actual y el comportamiento ideal.

3. Un mecanismo de retroalimentación que utilice la métrica del error cuantificado para guiar al programa a producir un mejor comportamiento en los eventos subsecuentes.

Los algoritmos de aprendizaje máquina pueden ser clasificados en varios tipos, cada uno de estos tipos trabaja de manera diferente y utiliza tipos de datos diferentes, estos son (Burkov, 2019):

- Aprendizaje Supervisado: El objetivo del aprendizaje supervisado es utilizar un *Dataset* para producir un modelo que toma un vector de características *x* como entrada, obteniendo como salida información que permita deducir la etiqueta de este vector de características.

- Aprendizaje No supervisado: En este aprendizaje el *Dataset* es una colección de muestras no etiquetadas. Donde *x* es un vector de características, y la meta del algoritmo consiste en crear un modelo que tome el vector de características *x* como entrada y lo transforma en otro vector, o si no, a un valor que puede ser utilizado para resolver problemas prácticos.

- Aprendizaje Semi-Supervisado: En este tipo de aprendizaje el *Dataset* contiene tanto muestras etiquetadas como no etiquetadas. Usualmente, la cantidad de muestras no

etiquetadas es mucho mayor que el número de muestras etiquetadas. Su objetivo es el mismo que el aprendizaje supervisado con la esperanza que al utilizar muestras no etiquetadas se produzca un mejor modelo.

- Aprendizaje por refuerzo: Es un sub-campo del aprendizaje máquina donde la máquina "vive" en un ambiente y es capaz de percibir el estado de ese mismo ambiente como un vector de características. La máquina realiza acciones en cada uno de los estados del ambiente. Su objetivo consiste en aprender una política, definida como una función f que toma el vector de características de un estado como entrada, obteniendo de salida una acción óptima a ejecutar.

Cada uno de estos tipos de aprendizaje ofrecen diferentes soluciones dependiendo de las características del problema. En este trabajo se enfocará en el aprendizaje supervisado, el cual será profundizado a continuación.

2.2.1. Aprendizaje supervisado

Los algoritmos de aprendizaje supervisado utilizan un *Dataset* que contiene características, pero cada una de las muestras esta también asociada a una etiqueta u objetivo. Este tipo de aprendizaje involucra la observación de múltiples ejemplos de un vector aleatorio x y un vector y o valor asociado. De esta manera el algoritmo aprende a predecir el valor y del valor x, p(y|x). En otras palabras, el término de aprendizaje supervisado se origina desde la vista de que el objetivo y es provisto por un instructor que muestra a la máquina de aprendizaje que hacer (Bengio & Courville, 2017).

2.2.1.1. Clasificación

La clasificación es un problema de asignar automáticamente una etiqueta a una muestra no etiquetada. En un problema de clasificación, una etiqueta es un miembro de un conjunto finito de clases. En el aprendizaje máquina el problema de clasificación se resuelve mediante un algoritmo que toma una colección de muestras etiquetadas como entradas y produce un modelo que es capaz de tomar muestras no etiquetadas como entrada y directamente obtener como salida una etiqueta o un valor numérico (Burkov, 2019).

La forma más básica de clasificación es una clasificación binaria, donde tiene una sola salida con dos etiquetas (o dos clases, p. ej. 0 y 1, respectivamente). La salida también puede ser de tipo flotante, donde se obtiene un número entre 0.0 y 1.0 que indica una clasificación por debajo de la certeza absoluta. Para este caso, es necesario definir un umbral (usualmente 0.5) que limita las dos clases (Patterson & Gibson, 2017).

Más allá de tener solo dos etiquetas, se pueden tener modelos que tengan *N* etiquetas en las cuales se puntuaría cada una de las etiquetas de salida, y luego la etiqueta con la puntuación más alta es la etiqueta de salida. Definido de manera formal "El objetivo de la clasificación consiste en aprender a relacionar las entradas *x* con las salidas *y*, donde $y \in \{1, \ldots, C\}$, donde *C* es el número de clases. Si $C = 2$, entonces se le llama clasificación binaria, en cambio si $C > 2$, se le llama clasificación multiclase" (Murphy, 2012).

2.2.1.2. Regresión

La regresión es un problema que busca predecir el valor real de la etiqueta dada una muestra no etiquetada. Este problema es resuelto mediante un algoritmo de regresión que toma una colección de muestras etiquetadas como entradas y produce un modelo que puede tomar una muestra no etiquetada como muestra y obtener el valor real objetivo como salida (Burkov, 2019).

La palabra regresión se refiere a funciones que intentan predecir el valor real de la salida. Este tipo de función estima la variable dependiente conocimiento la variable independiente. El caso más común de regresión es la regresión lineal. La regresión lineal intenta generar una función que describa la relación entre la variable independiente *x* y la variable dependiente *y*, es decir, para los valores conocidos de *x*, predice el valor de *y* (Patterson & Gibson, 2017).

Aunque el tipo de regresión más común es la regresión lineal, también existe la regresión polinomial, esta regresión es comúnmente utilizada para problemas donde los datos no muestran una clara división entre ellos. En la figura 2.1 se muestran ejemplos de los dos tipos de regresión.

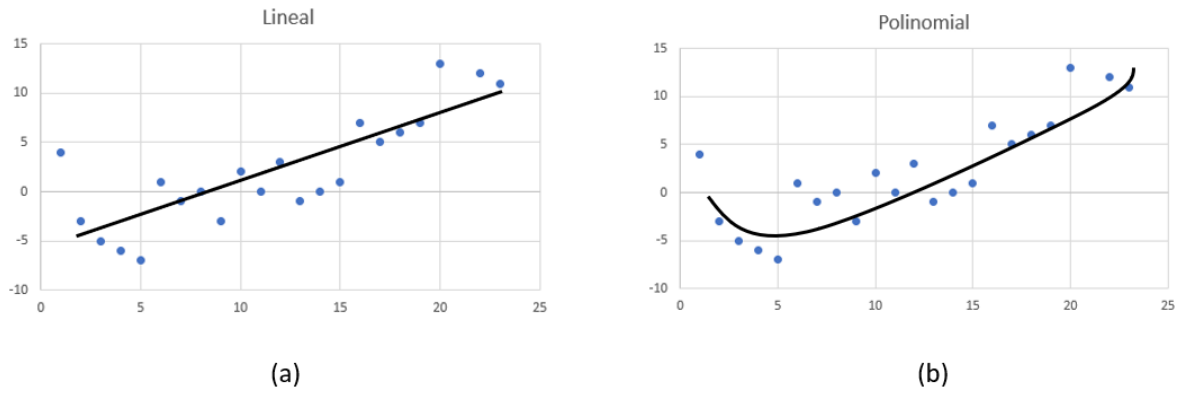

Figura 2.1: (a) Muestra de regresión lineal. (b) Muestra de regresión polinomial

2.2.2. Redes neuronales

Una red neuronal puede ser definida como un modelo de razonamiento basado en el cerebro humano. El cerebro consiste en un conjunto densamente conectado de células nerviosas o unidades de procesamiento de información básica, llamados neuronas. Mediante el uso de múltiples neuronas de forma simultánea, el cerebro puede realizar sus funciones de una manera más rápida que la computadora más rápida. A pesar de que las neuronas tienen una estructura muy simple, un conjunto de esos elementos constituye un poder de procesamiento tremendo (Negnevitsky & Intelligence, 2005).

Las redes neuronales son un modelo computacional que comparte algunas propiedades con el cerebro, en donde muchas unidades simples trabajan en paralelo con una unidad de control no centralizada. Donde los pesos entre las unidades son los medios principales de almacenamiento de información a largo plazo en las redes neuronales. La actualización de esos pesos son la forma principal de que la red neuronal aprende nueva información (Patterson & Gibson, 2017).

2.2.3. Perceptrón

El perceptrón es un modelo lineal utilizado para clasificación binaria. En el campo de las redes neuronales el perceptrón es considerado una neurona artificial que utiliza la función paso *Heaviside* como función de activación. El precursor del perceptrón era el *Threshold Logic Unit* (TLU) desarrollado por McCulloch y Pitts en 1943, esta unidad lógica puede

aprender las funciones lógicas AND y OR. La Figura 2.2 muestra la representación gráfica de TLU. El algoritmo de entrenamiento del perceptrón es considerado como aprendizaje supervisado (Patterson & Gibson, 2017)

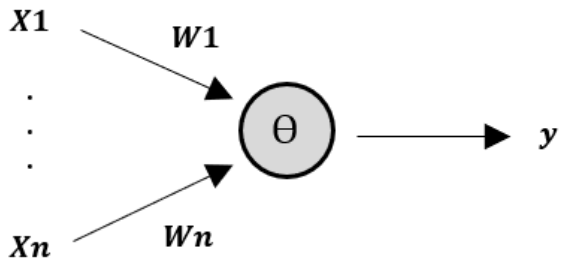

Figura 2.2: *Threshold Logic Unit*. Basado en Kruse et al., s.f.

La operación del perceptrón de Rosenblatt's esta basada en el modelo neuronal de McCulloch y Pitts. El modelo consiste de un combinador lineal seguido de un limitador duro. La suma de los pesos de las entradas es aplicada al limitador duro, que produce una salida +1, si su entrada es positiva, y -1, si su entrada es negativa. El objetivo del perceptrón es clasificar las entradas, o en otras palabras, estímulos aplicados externamente $x_1, x_2, \ldots, x_n$, en una de dos clases. Así, en caso de que un perceptrón elemental, el espacio n-dimensional es dividido en por un hiperplano en dos regiones de decisión. Este hiperplano es definido por la Eq. 2.1 (Negnevitsky & Intelligence, 2005).

$$\sum_{i=1}^{n} x_i w_i - \theta = 0 \tag{2.1}$$

Siendo el caso de dos entradas, x_1 y x_2, el límite de decisión toma la forma de una línea recta, como se muestra en la Figura 2.3 (a). De esta manera se puede ver que el punto 1 pertenece a la clase A_1; en cambio el punto 2 pertenece a la clase A_2.

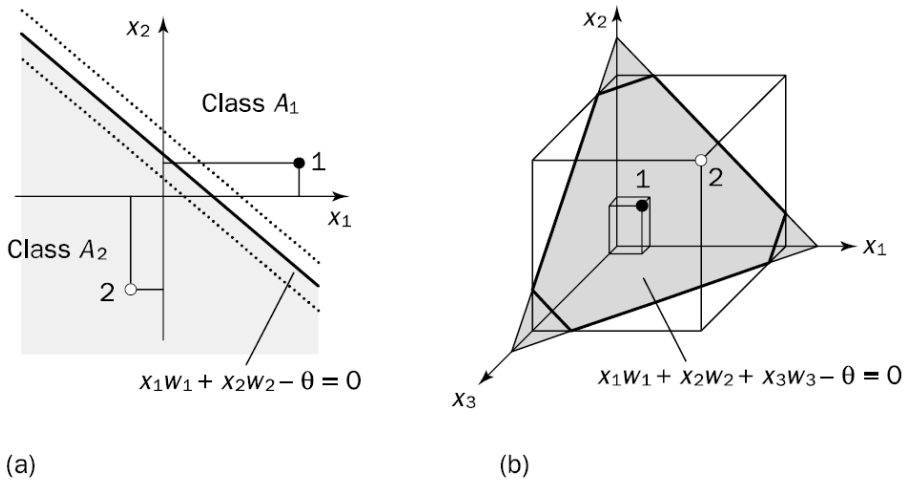

Figura 2.3: Separabilidad lineal en los perceptrones: (a) perceptrón de dos entradas; (b) perceptrón de tres entradas. Tomado de Negnevitsky & Intelligence, 2005

Con tres entradas el hiperplano aún puede ser visualizado. La Figura 2.3 (b) muestra un plano de tres dimensiones para el perceptrón de tres entradas. De manera que, el plano que separa las clases está definido por la Eq. 2.2.

$$x_1w_1 + x_2w_2 + x_3w_3 - \theta = 0 \tag{2.2}$$

Donde el punto 1 se representa en el plano de la Figura 2.3 (b) como punto negro y pertenece a la clase 1, y el punto 2 representa el punto blanco y pertenece a la clase 2.

2.2.4. Perceptrón multicapa

El perceptrón multicapa, también conocido como red neuronal multicapa, es una red neuronal de propagación hacia adelante con una o más capas ocultas. Comúnmente, la red consiste de una capa de entrada fuente de neuronas, al menos una capa oculta de neuronas computacionales, y una capa de salida de neuronas computacionales. Las señales de entrada son propagadas con dirección hacia adelante capa por capa. La Figura 2.4 muestra un perceptrón multicapa (Negnevitsky & Intelligence, 2005).

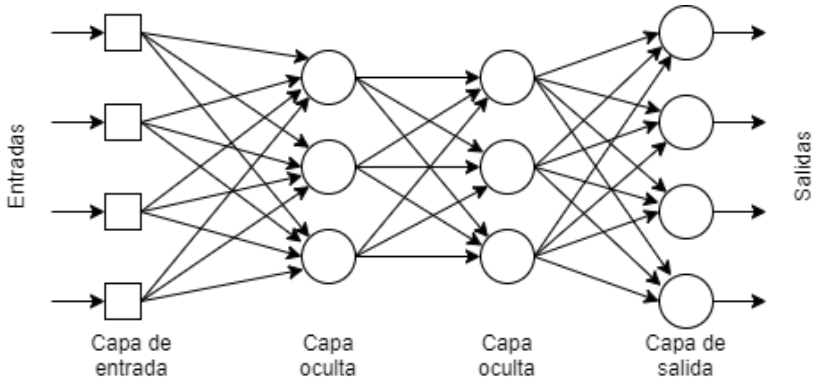

Figura 2.4: Perceptrón multicapa con dos capas ocultas. Basado en Negnevitsky & Intelligence, 2005

Cada capa en una red neuronal multicapa tiene su propia función específica. La capa de entrada acepta señales de entrada del mundo exterior y redistribuye esas señales hacia todas las neuronas en la capa oculta. De hecho, la capa de entrada rara vez incluye neuronas de cómputo. La capa de salida acepta señales de salida, o en otras palabras un patrón de estímulo, desde la capa oculta y establece el patrón de salida de toda la red entera.

Una red neuronal multicapa puede representar cualquier función, si se le dan las suficientes unidades de neuronas artificiales. Generalmente se entrena mediante un algoritmo de aprendizaje llamado propagación hacia atrás (*Backpropagation* en Inglés). Este algoritmo utiliza un descenso de gradiente en los pesos de las conexiones en una red neuronal para minimizar el error de la salida de la red (Patterson & Gibson, 2017).

El perceptrón multicapa es solo una función matemática que mapea un conjunto de valores de entrada a valores de salida. La función está formada por la composición de muchas funciones simples. La meta de la red neuronal multicapa es aproximar una función f^*. Por ejemplo, para un clasificador, $y = f^*(x)$ mapea una entrada *x* a una categoría *y*. Una red de propagación hacia adelante define un mapa $y = f(x; \theta)$ y aprende el valor de los parámetros θ que resultan ser la mejor aproximación a la función (Goodfellow et al., 2016).

A esos modelos se les llama propagación hacia adelante debido a que la información fluye a través de la función que se evalúa desde *x*, a través de los cálculos intermedios utilizados para definir *f*, y finalmente a la salida *y*.

2.2.5. Generalización, subajuste y sobreajuste

En el aprendizaje máquina el objetivo de los modelos consiste en tener un buen rendimiento en muestras nuevas y no vistas anteriormente, no solamente aquellas muestras con las cuales el modelo se entrenó. Se le llama Generalización a la habilidad del modelo para interpretar de manera correcta entradas que aún no han sido observadas (Bengio & Courville, 2017).

Usualmente, cuando se entrena un modelo de aprendizaje máquina, tenemos acceso al conjunto de entrenamiento. Con este se puede computar alguna medida de error en el conjunto de entrenamiento, llamada comúnmente error de entrenamiento, y lo que se busca es reducir ese error de entrenamiento. Hasta este punto parece que el entrenamiento se puede interpretar como un problema de optimización. Sin embargo, lo que separa el aprendizaje máquina de la optimización es que además, se busca disminuir el error de generalización. El error de generalización es definido como el valor esperado del error de una nueva entrada. Así el modelo aprende la expectativa a través de las diferentes entradas posibles que pueda llegar a encontrar en la práctica. Típicamente, se estima el error de generalización de un modelo de aprendizaje máquina mediante la medición del rendimiento con un conjunto de muestras de validación, diferentes del conjunto de entrenamiento.

Existen dos factores que determinan que tan bueno es el rendimiento de un algoritmo de aprendizaje máquina, estos son: Reducir el error del entrenamiento y reducir el espacio entre el error de entrenamiento y el error de validación. Estos dos factores corresponden los dos principales desafíos en el aprendizaje máquina, que son: el subajuste y el sobreajuste. El subajuste ocurre cuando un modelo no es capaz de obtener un valor de error lo suficientemente bajo en el conjunto de entrenamiento. En cambio, el sobreajuste ocurre cuando la distancia entre el error de entrenamiento y el error de validación es muy grande.

El primer intento de los algoritmos de optimización consiste en resolver el problema del subajuste, esto se representa dibujando una línea recta que atraviesa una gráfica de dispersión, como se muestra en la Figura 2.5. Pero si la línea es trazada demasiado bien, se obtiene el problema opuesto, esto es el sobreajuste. Aunque resolver el subajuste es la prioridad, mucho del esfuerzo del aprendizaje máquina es dedicado a intentar no sobreajustar la línea de los

datos (Patterson & Gibson, 2017).

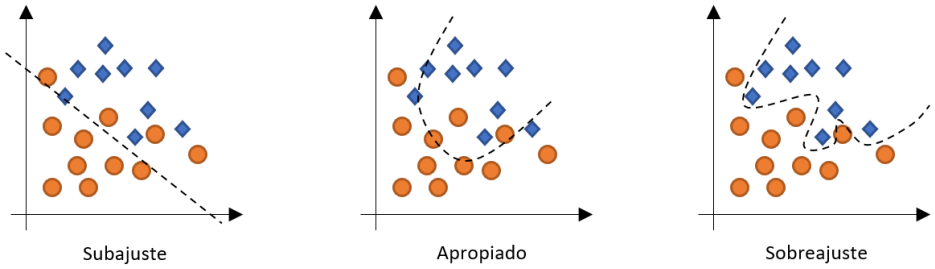

Figura 2.5: Subajuste y sobreajuste en aprendizaje máquina. Basado en Patterson & Gibson, 2017

Hasta el modelo más simple de aprendizaje máquina puede llegar a sobre ajustar los datos. Esto usualmente pasa cuando los datos son de dimensiones altas, pero el número de muestras de entrenamiento es muy bajo. Un modelo complejo puede predecir mal los valores de las muestras. Esto se debe a que, al intentar predecir perfectamente los valores de todas las muestras de entrenamiento, el modelo también aprende las idiosincrasias del conjunto de entrenamiento, como: el ruido de los valores de características de las muestras de entrenamiento, la imperfección de las muestras debido al pequeño tamaño del *Dataset*, entre otros artefactos extrínsecos al problema de decisión en cuestión (Burkov, 2019).

2.3. Aprendizaje profundo

La inteligencia artificial es un campo muy amplio el cual existe desde hace tiempo. Este campo engloba el área del aprendizaje máquina, que a su vez engloba el campo del aprendizaje profundo. En otras palabras el aprendizaje profundo es un sub-campo del aprendizaje máquina. Para asignar en un contexto visual el aprendizaje profundo, la Figura 2.6 ilustra las relaciones entre las áreas.

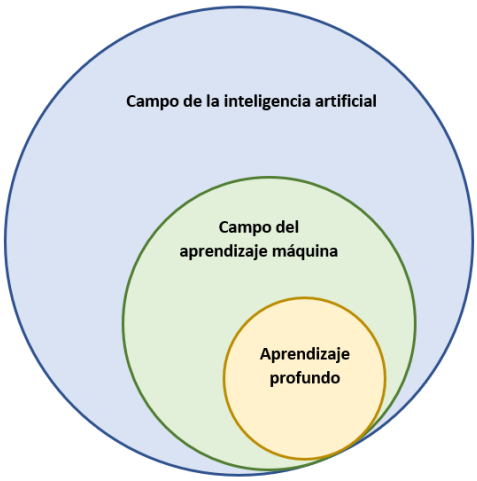

Figura 2.6: Relación entre las diferentes áreas de la inteligencia artificial. Basado en Patterson & Gibson, 2017

El concepto del aprendizaje profundo ha ido cambiando en la década pasada, y es un término que sigue a discusión. Sin embargo, una definición que puede servir es que el aprendizaje profundo trata con "Redes neuronales con más de dos capas" (Patterson & Gibson, 2017). Aunque no es un concepto propiamente dicho, esta definición nos dice que el aprendizaje profundo engloba técnicas de aprendizaje máquina con un nivel de complejidad alto.

Los aspectos que diferencian una red de aprendizaje profundo con redes multi-capa son las siguientes:

- Más neuronas que las anteriores redes.
- Formas más complejas de conectar las capas.
- Una explosión en la cantidad de poder de cómputo disponible para entrenar.
- Extracción de características automática.

El concepto de más neuronas, quiere decir que la cantidad de neuronas utilizadas aumento con respecto a años pasados. En cambio el concepto de más conexiones significa que las redes cuentan con más parámetros a optimizar, y esto requiere una explosión en el poder computacional necesario. De esta forma, las redes profundas pueden modelar espacios del problema más complejos. En otras palabras el aprendizaje profundo son redes neuronales

con un gran número de parámetros y capas en alguna de las cuatro arquitecturas de redes fundamentales:

- Redes pre-entrenadas no supervisadas.
- Redes neuronales convolucionales.
- Redes neuronales recurrentes.
- Redes neuronales recursivas.

Muchas tareas que consisten en mapear un vector de entrada a un vector de salida, y sean fáciles de realizar por una persona, pueden también realizarse por medio del aprendizaje profundo, solo si el modelo y el *Dataset* son lo suficientemente grandes (Goodfellow et al., 2016).

2.3.1. Entrenamiento de redes neuronales

Una red neuronal muy bien entrenada tiene pesos que amplifican la señal y amortiguan el ruido. Un mayor peso significa una menor correlación entre una señal y la salida de la red. Las entradas emparejadas con grandes pesos afectarán a la interpretación de los datos por parte de la red aún más que las entradas emparejadas con los pesos más pequeños (Patterson & Gibson, 2017).

El proceso de aprendizaje de cualquier algoritmo que utiliza pesos, es el proceso de re-ajustar los pesos y los sesgos, haciendo algunos más pequeños y otros grandes, de esta manera se le asigna importancia a ciertos bits de información y minimizando otros bits. Esto ayuda al modelo a aprender cuáles predictores (o características) están vinculados a que salidas, y en consecuencia ajustar así los pesos y los sesgos.

En la mayoría de los *Datasets*, ciertas características están fuertemente correlacionadas con ciertas etiquetas. Las redes neuronales son capaces de aprender esas relaciones ciegamente mediante una suposición basada en las entradas y los pesos, y así medir que tan precisos son los resultados. Las funciones de perdida en los algoritmos de optimización recompensan a la red por las suposiciones correctas y las penalizan por las suposiciones incorrectas.

2.3.1.1. Propagación hacia atrás (Backpropagation)

Cuando utilizamos una red neuronal de propagación hacia adelante para aceptar una entrada *x* y producir así una salida *y*, la información fluye hacia adelante a través de la red. Las entradas *x* proveen la información inicial que es propagada hacia las unidades ocultas en cada capa y finalmente produce la salida *y*. A esto se le llama propagación hacia adelante. Durante el entrenamiento la propagación hacia adelante puede continuar hacia adelante hasta que se produce un costo escalar $J(\theta)$. El algoritmo de propagación hace atrás, permite que la información del costo dada por el algoritmo de propagación hacia adelante fluya hacia atrás a través de la red, con el fin de calcular el gradiente (Goodfellow et al., 2016).

Calcular una expresión analítica para el gradiente es sencillo, pero evaluar numéricamente dicha expresión puede resultar computacionalmente costoso. El algoritmo de propagación hacia atrás lo hace mediante un procedimiento simple y económico. Lo que se busca es calcular las muestras de salida mediante un pase directo a través de la red. Si la salida coincide con la etiqueta, no se hace nada. En cambio, si la salida no coincide con la etiqueta, se necesita ajustar los pesos en las conexiones en la red neuronal (Patterson & Gibson, 2017).

El algoritmo de propagación hacia atrás es un enfoque pragmático para dividir la contribución del error por cada peso. Con este algoritmo, se intenta minimizar el error entre la etiqueta de salida asociada con la entrada de entrenamiento y el valor generado por la salida de la red.

2.3.1.2. Optimización por descenso de gradiente

La forma de abordar los entrenamientos del aprendizaje profundo es como un problema de optimización. Donde su objetivo consisten en minimizar o maximizar una función $f(x)$ por medio de la alteración de x. La función que se requiere minimizar o maximizar es también llamada la función objetivo o criterio. También cuando se busca minimizar la función, se le llama función de costo, función de perdida o función de error (Goodfellow et al., 2016).

Entonces la optimización por descenso de gradiente se define de la siguiente manera: Una función determinada $y = f(x)$, donde y y x son números reales, la derivada de esta función puede denotarse como $f'(x)$ o $\frac{dy}{dx}$. De esta manera, la derivada $f'(x)$ proporciona la pendiente

de $f(x)$ en un punto x. Descrito de otra manera, la derivada $f'(x)$ especifica como escalar un pequeño cambio en la entrada x para obtener un cambio correspondiente en la salida y. La derivada $f'(x)$ es útil para minimizar una función, debido a que dice como cambiar x para obtener pequeñas mejoras en y. La Figura 2.7 muestra la optimización por descenso de gradiente.

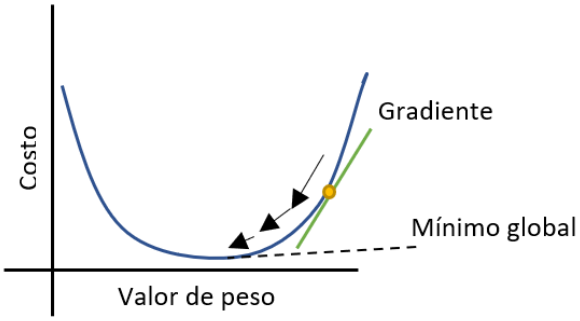

Figura 2.7: Optimización por descenso de gradiente. Basado en Patterson & Gibson, 2017

2.3.2. Hiperparámetros

Los hiperparámetros son típicamente un conjunto de parámetros que pueden ser ilimitados. El usuario teóricamente puede escoger un número entre 1 e infinito, pero no puede simplemente utilizar los datos de entrenamiento para encontrar el número correcto de nodos. Por lo que, la definición de los límites de los hiperparámetros es especificada por el experto, donde los límites son creados con base en múltiples restricciones, tales como: requisitos de cálculo, dimensionalidad, tamaño de los datos, etc. Típicamente sería necesario crear los límites para más de un hiperparámetro, por lo que, se obtendría una cuadrícula n-dimensional de los hiperparámetros para elegir (Joshi, 2020).

En el aprendizaje máquina existen los parámetros de modelo y parámetros que se ajustan para hacer que las redes se entrenen mejor y más rápido. A estos parámetros de ajuste se les denominan hiperparámetros, y su función principal consiste en controlar las funciones de optimización y selección de modelos durante el entrenamiento del algoritmo de aprendizaje. La selección de hiperparámetros se enfoca en asegurar que el modelo no tenga un subajuste o sobreajuste el conjunto de datos, mientras que aprende la estructura de los datos. Aunque los hiperparámetros son diferentes para cada tipo de modelo de aprendizaje, los que comúnmente

se encuentran en las redes neuronales son los siguientes (Patterson & Gibson, 2017):

- Número de capas
- Número de neuronas por capa
- Funciones de activación
- Funciones de perdida
- Función de optimización
- Épocas de entrenamiento
- Inicializador de pesos.
- Tamaño de lotes.
- Tasa de aprendizaje
- Regularización
- Impulso
- Escasez

2.3.2.1. Tasa de aprendizaje

El rol de la tasa de aprendizaje consiste en moderar el grado en el cual los pesos del modelo son cambiados. Usualmente se establece un valor bastante pequeño (por ejemplo, 0.1) y comúnmente se configura para decaer a medida que el número de iteraciones de ajustes de pesos aumenta (Mitchell et al., 1997).

La tasa de aprendizaje afecta la cantidad en la que se ajustan los parámetros durante la optimización, esto para minimizar el error en las conjeturas de la red neuronal. Un coeficiente de tasa de aprendizaje grande (por ejemplo, 1) hará que los parámetros den saltos, y los coeficientes pequeños (por ejemplo, 0.0001) hará que los parámetros avancen lentamente. Si los saltos son grandes, aunque ahorraran tiempo inicialmente, pueden llegar a ser desastrosos si sobrepasa el mínimo establecido. En cambio, una tasa de aprendizaje pequeña eventualmente llevará al modelo a un error mínimo, pero puede llegar a tomar una mayor cantidad de tiempo (Patterson & Gibson, 2017).

2.3.2.2. Funciones de activación

Una función de activación es una función matemática que mapea las entradas de una neurona a su correspondiente salida (Negnevitsky & Intelligence, 2005). Las funciones de activación controlan el comportamiento de las neuronas artificiales. Estas funciones se encargan de transformar una combinación de entradas, pesos y sesgos para producir una salida. Las funciones de activación son utilizadas para propagar las salidas de los nodos de una capa hacia la siguiente capa. Se utilizan las funciones de activación en las neuronas de las capas ocultas de una red neuronal para introducir la no-linealidad en las capacidades de la red (Patterson & Gibson, 2017).

Cada una de las capas que componen una red neuronal puede utilizar una sola función de activación diferente, cada función tiene su objetivo y depende del tipo de solución en la capa y el tipo de datos que se estén manejando. Para la capa de entrada comúnmente se utilizan los valores originales sin modificar del conjunto de datos. En cuanto a las capas ocultas, estas se concentran en extraer las características de orden superior de manera progresiva de los datos originales. Algunas de las funciones de activación más comunes son:

- Sigmoid
- Tanh
- ReLU

La función *Sigmoid* (Sigmoidea) puede reducir valores extremos o valores atípicos en los datos sin eliminarlos manteniendo los valores en un rango entre 0 y 1. La función *Tanh* (Tangente hiperbólica) es una función trigonométrica que representa una razón entre los lados opuesto y adyacente de un triángulo rectángulo, además maneja más fácilmente los valores que son negativos, y su rango de normalización es entre -1 y 1. Por último, la función *ReLU* (Rectificador Lineal) que solo permite la activación de nodos si la entrada anterior está arriba de un umbral, es decir, mientras que la entrada sea menor a 0 la salida será 0. El rango de valores de la función *ReLU* se define como: $f(x) = max(0, x)$.

Por último, en la capa de salida se especifica la función de activación dependiendo del dominio del problema que se está solucionando. Es común que para una capa de salida para

regresión se busque un valor real, en este caso se utiliza una función de activación lineal. Para una capa de salida de clasificación binaria se espera una salida de una única neurona con un valor real entre 0.0 y 1.0, para esto se utiliza una función de activación de tipo sigmoidea. En cuanto a una capa de salida para clasificación multi-clase la función de activación *softmax* ofrece una distribución de probabilidad sobre todas las clases.

2.3.2.3. Optimizador

Los algoritmos del aprendizaje profundo involucran la optimización en muchos contextos. La optimización es usualmente utilizada para escribir pruebas o diseñar algoritmos. El problema más difícil de optimización en el aprendizaje profundo es el entrenamiento de las redes neuronales. Un caso en particular de optimización de una red neuronal es el siguiente: Encontrar los parámetros θ de una red neuronal que significativamente reduzca el costo de la función $j(\theta)$, que comúnmente incluye una medida de rendimiento evaluada en todo el conjunto de datos (Goodfellow et al., 2016).

Otra manera de ver el aprendizaje máquina es como un problema de optimización, en donde se busca minimizar la función de perdida con respecto a los parámetros de la función de predicción. El algoritmo de Descenso de Gradiente Estocástico (SGD) y sus variantes son de los más utilizados en la optimización de modelos de aprendizaje máquina. Sus puntos fuertes consisten en que es de fácil implementación y tiene un procesamiento rápido en conjuntos de datos grandes (Patterson & Gibson, 2017).

Otro de sus puntos es que el SGD puede ser ajustado mediante la adaptación de la tasa de aprendizaje o utilizando información de segundo orden. La versión original del SGD utiliza el gradiente de manera directa, esto deriva en que el gradiente puede ser muy cercano a cero para cualquier parámetro, así el SGD toma pasos muy pequeños o muy grandes en algunos casos. Para solucionar esto existen los algoritmos adaptativos, estos algoritmos son variantes del SGD y pueden adaptar una tasa de aprendizaje diferente a cada parámetro. Alguno de los algoritmos adaptativos más populares son:

- AdaGrad
- RMSProp

- Adam
- AdaDelta

Cada uno de estos algoritmos presentan sus ventajas y desventajas. Ninguno de estos es mejor que los demás en todos los casos. El uso de cada uno de ellos depende en gran medida de las necesidades del modelo y las características de los datos que se manejan para el entrenamiento.

2.3.2.4. Funciones de pérdida

Las funciones de perdida cuantifican que tan cercanos son los pronósticos de una red neuronal con los datos reales cuando se está en fase de entrenamiento. Se hace un cálculo de una métrica basada en el error que se observa en las predicciones de la red. Posteriormente, se agregan los errores en todo el conjunto de datos y se calcula el promedio del error. De esta manera, se obtiene un número que representa que tan cerca se encuentra la red neuronal de su ideal. De esta manera el problema del entrenamiento de redes neuronales se trata como un problema que minimiza la "perdida" resultante de los errores. Estas funciones de perdida permiten el manejo de métodos de optimización iterativos para aproximarse a una solución satisfactoria (Patterson & Gibson, 2017).

Dependiendo del problema que se requiera resolver, será la función de pérdida a elegir, pues cada una de ellas es adecuada para resolver un problema diferente. Los tipos de problemas pueden ser de tres categorías: Regresión, Clasificación y Reconstrucción. Las funciones de perdida comúnmente utilizadas en regresión son:

- Pérdida de error cuadrático medio (MSE)
- Pérdida de error absoluto medio (MAE)
- Pérdida de error cuadrático medio logarítmico (MSLE)
- Pérdida de error absoluto medio porcentual (MAPE)

Por otra parte, las funciones de perdida para clasificación son:

- Perdida de bisagra (*Hinge Loss*)
- Pérdida logística
- Pérdida de verosimilitud logarítmica negativa

Por último las funciones de perdida utilizadas para reconstrucción son funciones de perdida que tienen sus raíces en la teoría de la información. Las funciones de perdida pueden ser: Divergencia KL y entropía cruzada.

2.3.3. Red Neuronal Recurrente

Las Redes Neuronales Recurrentes (RNN por sus siglas en Inglés) se caracterizan por su habilidad de mandar información sobre los pasos de tiempo. El entrenamiento de este tipo de red neuronal se realiza tomando cada uno de los vectores de una secuencia de vectores de entrada, y posteriormente se modelan uno por cada instante de tiempo. De esta manera, se le permite a la red retener un estado mientras modela cada vector de entrada a través de una ventana de vectores de entrada. La especialidad de las RNN consiste en tener la capacidad de modelar la dimensión del tiempo (Patterson & Gibson, 2017).

Una RNN es un tipo de red que se especializa en procesar una secuencia de valores $x(1), \ldots, x(t)$. Estas son capaces procesar secuencias de larga escala de manera práctica mediante una serie de operaciones de secuencia que contienen vectores $x(t)$, donde t es el índice del paso del tiempo de 1 a t. En la práctica, las RNN se comportan de manera diferente, estas usualmente operan en pequeños lotes, con un tamaño de secuencia t diferente para cada miembro del pequeño lote (Goodfellow et al., 2016).

2.3.4. Redes de Gran Memoria de Corto Plazo (LSTM)

Las Redes de Gran Memoria de Corto Plazo (LSTM por sus siglas en Inglés) son una mejora de la arquitectura de RNN tradicional, debido principalmente a que contienen una gran memoria de corto plazo. Esta mejora ofrece una solución a las limitaciones que tiene una RNN. El cambio fundamental establecido consiste en el uso de una puerta del olvido, su objetivo consiste en combinar la entrada con la salida anterior y devolver una fracción entre 0

y 1, donde dicho valor corresponde a que tanto el estado anterior debe ser preservado (Joshi, 2020). Los componentes de una LSTM consisten en:

- Tres puertas:
  - Puerta de entrada
  - Puerta de olvido
  - Puerta de salida
- Bloque de entrada
- Celda de memoria
- Función de activación de salida
- Conexiones de mirilla

La puerta de entrada protege a las unidades de eventos de entrada irrelevantes. En cambio, la puerta de olvido ayuda a la unidad a olvidar el contenido previo de la memoria. Finalmente, la puerta de salida expone el contenido de la memoria en la salida de la unidad LSTM (Patterson & Gibson, 2017).

2.3.5. Red neuronal convolucional

La Red Neuronal de Convolución (CNN por sus siglas en Inglés) aprende características de orden superior mediante la aplicación de la convolución. Su principal especialización es el reconocimiento de objetos en imágenes y más aún en la clasificación de imágenes (Patterson & Gibson, 2017). La Figura 2.8 muestra como una CNN realiza una clasificación.

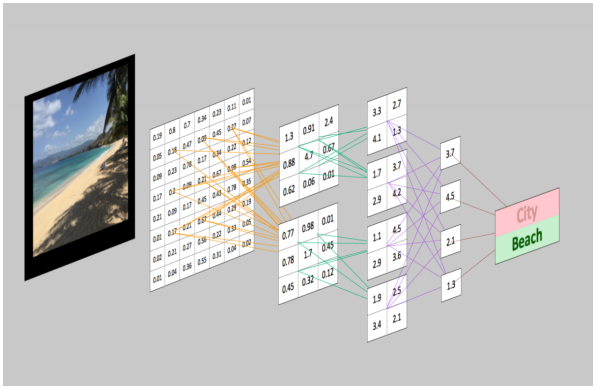

Figura 2.8: Proceso de clasificación de CNN. Tomado de Patterson & Gibson, 2017

La CNN realiza un proceso de transformación de la información de entrada, desde la capa de entrada a través de todas las capas conectadas, en un conjunto de puntajes de clase dados por la capa de salida. La arquitectura de la CNN se puede dividir en tres grupos principales:

- Capa de entrada
- Capas de extracción de características
- Capas de clasificación

Generalmente la capa de entrada acepta información tridimensional con el tamaño espacial de la imagen (Anchura x altura) y una profundidad que representa el canal de los colores. Conforme a la capa de extracción de características, tienen un patrón repetitivo que se conforma por la siguiente secuencia:

1. Capas de convolución. Expresando la función de activación "Unidad de Rectificador Lineal" (ReLU por sus siglas en Inglés)
2. Capa de agrupación (*Pooling* en Inglés)

Las capas anteriormente descritas son capaces de construir progresivamente características de orden superior. Finalmente, las capas de clasificación se constituyen por una o más capas totalmente conectadas, estas tomas las características de orden superior y producen probabilidades de clases o puntajes. Estas capas de clasificación comúnmente producen salidas bidimensionales ($b * N$), donde b es el número de ejemplos en el pequeño lote y N es el número de clases en el cual se tiene interés.

2.3.6. Autocodificador

Los autocodificadores aprender representaciones comprimidas de los conjuntos de datos. Comúnmente son utilizados para reducir la dimensionalidad del conjunto de datos. La salida de la red del autocodificador es una reconstrucción de los datos de entrada de la forma más eficiente. Los autocodificadores comparten una fuerte similitud con el perceptrón multicapa, su principal diferencia es que la capa de salida en un autocodificador contiene el mismo número de unidades que la capa de entrada (Patterson & Gibson, 2017). La Figura 2.9 muestra la arquitectura de una red de autocodificador.

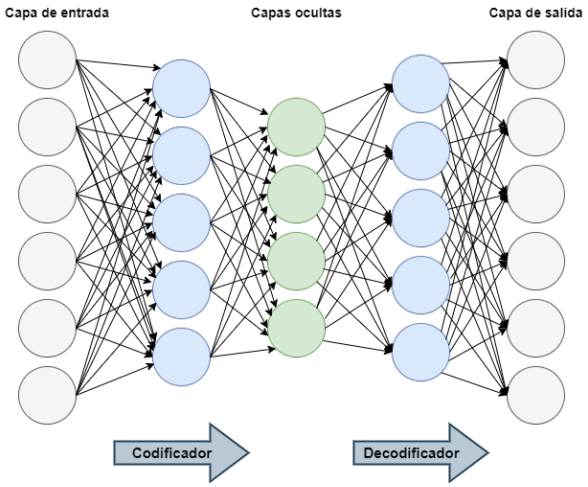

Figura 2.9: Arquitectura de un autocodificador

Los autocodificadores son redes capaces de aprender de manera eficiente las representaciones de los datos de entrada, llamados codificaciones, de manera no supervisada (Conjunto de datos sin etiquetar). Estas codificaciones comúnmente contienen una menor dimensionalidad que los datos de entrada, haciendo que los autocodificadores sean útiles en la reducción de dimensionalidad. Además, son capaces de generar aleatoriamente nuevos datos muy parecidos a los datos de entrenamiento originales, a esto se le llama modelo generativo (Géron, 2019).

2.3.7. Teorema del *"No Free Lunch"*

En el aprendizaje máquina se trata principalmente de aplicar una solución adecuada a una situación específica. Por lo que pueda parecer, una sola técnica o solución dada no puede observar todo el panorama que existe, por lo que es necesario evaluar el espacio del problema y los datos disponibles cada vez que se requiera obtener un buen modelo. Lo anterior se basa en teorema del *"No Free Lunch"*.

El teorema de *"No Free Lunch"* establece que "No existe algún modelo que funcione para todos los problemas. Las suposiciones de un gran modelo para un problema dado pueden no ser válidas para otro problema. Es común que en el aprendizaje máquina probar múltiples modelos diferentes y encontrar así uno que mejor funcione para un problema en particular" (Patterson & Gibson, 2017).

Este teorema es necesario tomarlo más como guía que como una ley, debido a que es muy posible que un algoritmo bien diseñado supere a otro algoritmo que no esté bien diseñado en todas las clases posibles de problemas. Sin embargo, en la práctica, se puede inferir del teorema de *"No Free Lunch"* que no se puede tener una solución para todos los problemas y esperar a que esta solución funcione bien en todas las situaciones.

2.3.8. Dimensionalidad

La dimensionalidad puede ser un concepto confuso cuando se manejan varios conjuntos de datos. Tomando el punto de vista físico, en un escenario real no se encontrarán más de tres dimensiones, que son: largo, ancho y alto. Sin embargo, es muy común manejar cientos o más dimensiones cuando se trata con datos para aprendizaje máquina. Para entender estas grandes dimensiones se define la propiedad fundamental de las dimensiones "Las dimensiones espaciales se definen de manera que cada una de las dimensiones sea perpendicular u ortogonal a las otras dos. Esta propiedad de ortogonalidad es esencial para tener una representación única de todos los puntos en el espacio dimensional". Matemáticamente hablando, es fácil agregar más dimensiones, pero es muy difícil visualizar estas dimensiones espacialmente. De manera que, al agregar una nueva dimensión, es necesario que esta sea ortogonal a todas las demás dimensiones (Joshi, 2020).

Muchas de las dificultades básicas en el aprendizaje máquina se derivan del uso de grandes cantidades de variables de entrada. De tal manera que, algunas tareas se vuelvan exponencialmente más difíciles a medida que el número de entradas aumente. A este problema se le conoce como la “maldición de la dimensionalidad”. Esta consiste en que, desde un punto de vista matemático, está bien añadir un número arbitrario de dimensiones, pero hay un problema, con el incremento de las dimensiones la densidad de los datos se reduce exponencialmente. La densidad de los datos es importante, debido a que, entre más alta sea la densidad de los datos, es más probable que se encuentre un buen modelo y hay más confianza en la precisión de ese modelo. La Figura 2.10 muestra la maldición de la dimensionalidad.

Figura 2.10: Aumento de complejidad del conjunto de datos en múltiples dimensiones. Tomado de Goodfellow et al., 2016

La figura 2.10 muestra el aumento del espacio entre los datos cuando el número de dimensiones aumenta. Esto es que a medida que las dimensiones relevantes de los datos aumenta, el número de configuraciones de interés también aumenta. La primera figura muestra un espacio de interés de 10 regiones que a su vez están casi completamente ocupados. Al agregar otra dimensión, la representación central, donde existen 10x10 = 100 espacios, se vuelve más complicado llenar todos los espacios que se encuentran. Aumentando a una dimensión más, la última representación, serian 10x10x10 = 1000 espacios y serían necesarios muchos más datos para poder llenar todos los espacios disponibles.

2.3.9. Entrenamiento de aprendizaje profundo con Unidades de Procesamiento Gráfico

La Unidad de Procesamiento Gráfico (GPU por sus siglas en Inglés) son utilizadas en la implementación de redes neuronales más modernas. Estas unidades originalmente fueron desarrolladas para aplicaciones gráficas, tales como los sistemas de videojuegos. Aunque las características de rendimiento de este hardware de procesamiento gráfico para videojuegos, resulta muy beneficioso para la aplicación de las redes neuronales (Goodfellow et al., 2016).

En esencia el renderizado de videojuegos realizan muchas operaciones rápidamente en paralelo. Debido a lo anterior las gráficas deben realizar multiplicaciones de matrices y divisiones de muchos vértices en paralelo para convertir las coordenadas 3D en coordenadas 2D para la pantalla. Además, las tarjetas gráficas deben realizar muchos cálculos para cada uno de los píxeles en paralelo y determinar el color de cada píxel. En otras palabras, el resultado de este conjunto de necesidades las tarjetas gráficas son diseñadas para tener un alto nivel de paralelismo y una alta banda ancha de memoria, a costa de tener menor velocidad de reloj y menos capacidad de ramificación en relación con las CPU.

Es por esto que las redes neuronales toman beneficio de estas características de rendimiento. En las redes neuronales usualmente se ven involucradas un alto y numeroso conjunto de parámetros, valores de activación, y valores de gradiente, donde cada uno de estos debe ser actualizado durante cada paso del entrenamiento. Debido a que las redes neuronales pueden ser divididas en múltiples individuos llamados neuronas, estas pueden ser procesadas de manera independiente de las otras neuronas en la misma capa.

2.4. Auto Aprendizaje Máquina (AutoML)

En años recientes las técnicas de aprendizaje profundo han invadido todos los aspectos de la vida de la sociedad y han traído una gran conveniencia para la misma. Sin embargo, la creación de sistemas de aprendizaje profundo de alta calidad para una tarea específica depende mucho de la experiencia del humano, lo anterior hace que la aplicación del aprendizaje profundo se vea limitado a pocas áreas. Aquí es donde entra el Aprendizaje Máquina

Automático (AutoML por sus siglas en Inglés), esta área se vuelve una solución prometedora para la construcción de sistemas de aprendizaje profundo sin la necesidad de asistencia humana (He et al., 2021).

El interés reciente por modelos de aprendizaje máquina complejos y computacionalmente caros con una gran cantidad de hiperparámetros, como marcos de trabajo de AutoML y redes neuronales profundas, ha resultado en el resurgir de la investigación de Optimización de Hiperparámetros (HPO por sus siglas en Inglés) (Hutter et al., 2019).

Un sistema de aprendizaje máquina contiene hiperparámetros, estos son modificados de manera automática para optimizar un modelo de aprendizaje máquina. Los modelos de redes neuronales profundas dependen crucialmente de un rango de opciones de hiperparámetros, entre estos se encuentran: Arquitectura de la red neuronal, regularización, y optimización.

2.4.1. Optimización de hiperparámetros

La optimización de hiperparámetros (HPO por sus siglas en Inglés) es uno de los pasos fundamentales del AutoML. Este problema nace en los años 90, donde Kohavi & John encontraron y establecieron que diferentes configuraciones de hiperparámetros tienden a trabajar de mejor manera para diferentes conjuntos de datos (Kohavi & John, 1995). Debido a lo anterior es necesario, para obtener soluciones aceptables para cada problema, buscar y encontrar la mejor configuración para el problema que se está resolviendo. Sin embargo, la aplicación de esta técnica de optimización se encuentra con diferentes retos, estos son:

- Las funciones de evaluación pueden ser extremadamente caras de aplicar para modelos muy grandes (Más específicamente aprendizaje profundo), *pipelines* complejos de aprendizaje máquina, o conjuntos de datos muy grandes.
- El espacio de configuración suele ser complejo y de altas dimensiones. Además, no se sabe cuál de los hiperparámetros de los algoritmos debe ser optimizado, y en que rangos.
- Usualmente no se tiene un acceso al gradiente de la función de perdida con respecto a los hiperparámetros.

- No se puede realizar una optimización directa para rendimientos de generalización para conjuntos de datos de entrenamiento de tamaño limitado.

Debido principalmente a que el funcionamiento interno y el razonamiento de inferencias de los modelos de redes neuronales es complicado de observar, este será tratado como un modelo de *caja negra*. Un modelo de caja negra es, en términos sencillos, un modelo en el cual solo se pueden observar sus entras y salidas, sin la necesidad de tener un conocimiento de su funcionamiento interno.

En general, es posible utilizar cualquier método de optimización de caja negra con HPO. Entre los métodos de HPO se encuentran: búsqueda de cuadrícula, búsqueda aleatoria, optimización bayesiana, optimización basada en gradientes, optimización evolucionaría, y optimización basada en poblaciones (He et al., 2021).

La búsqueda de cuadrícula divide el espacio de búsqueda en intervalos regulares, y selecciona el punto de mejor rendimiento. En caso contrario, la búsqueda aleatoria selecciona el mejor punto de un conjunto de puntos tomados aleatoriamente. La Figura 2.11 muestra las diferencias entre estas dos búsquedas.

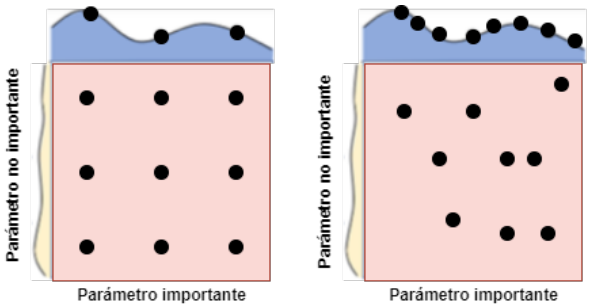

Figura 2.11: Diferencia entre búsqueda de cuadrícula (izquierda) y búsqueda aleatoria (derecha). Basado en He et al., 2021

Donde la zona superior azul de la Figura 2.11 es representada como g(x) y los parámetros son de importancia, en cambio la zona amarilla de la izquierda del cuadro rojo representa h(x) y los parámetros son de poca importancia. La diferencia radica en que la búsqueda de cuadrícula solo contiene tres parámetros importantes, mientras que la búsqueda aleatoria contiene más valores distintos a explorar. De esta manera, es más probable que la búsqueda aleatoria encuentre la combinación óptima.

A pesar de que pueda parecer que la búsqueda aleatoria es más práctica y eficiente que la búsqueda de cuadrícula, la búsqueda aleatoria no promete un punto óptimo. Esto quiere decir que a mayor sea la búsqueda, es más probable que encuentre los hiperparámetros óptimos, pero consumirá más recursos. Para contrarrestar este tipo de problemas se han desarrollado diferentes métodos. Li & Jamieson et al proponen el algoritmo *hyperband* que realiza una compensación entre el rendimiento de los hiperparámetros y los recursos disponibles. *Hyperband* asigna los recursos limitados disponibles a solo los hiperparámetros que sean más prometedores, de manera que, descarta sucesivamente la peor mitad de las configuraciones mucho antes de que el proceso haya finalizado (Li et al., 2017).

Otro método de optimización es la Optimización Bayesiana (BO por sus siglas en Inglés), este método es eficiente en cuestiones de optimización global para funciones de caja negra costosas. Este método construye un mapeo de modelo probabilístico de hiperparámetros a las métricas objetivo que son evaluadas en el conjunto de validación. BO realiza un buen balance entre la exploración y la explotación (He et al., 2021).

2.4.2. Búsqueda de un modelo de arquitectura neuronal

En los últimos años el aprendizaje profundo ha tenido progresos remarcables en una gran diversidad de tareas. Lo anterior se debe principalmente a las novedosas arquitecturas neuronales que se han desarrollado recientemente. Sin embargo, las arquitecturas fueron en su mayoría desarrolladas por expertos, lo cual provoca que el desarrollo consuma grandes cantidades de tiempo con un proceso de prueba y error. Debido a lo anterior, recientemente ha habido un interés creciente por las técnicas de búsqueda de arquitecturas automatizado, llamado Búsqueda de Arquitectura Neuronal (NAS por sus siglas en Inglés).

NAS es un proceso de ingeniería de arquitectura automática, que se considera un subcampo del AutoML y contiene una superposición significativa con la HPO. Los métodos de NAS son categorizados de acuerdo con tres dimensiones, las cuales son: Espacio de búsqueda, Estrategia de búsqueda, y estrategia de estimación de rendimiento (Hutter et al., 2019). La Figura 2.12 muestra el proceso de búsqueda NAS.

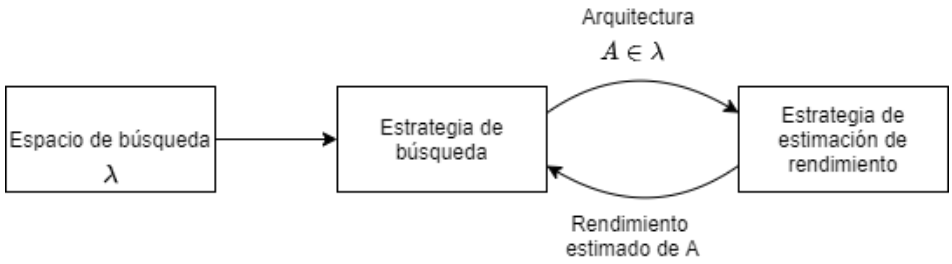

Figura 2.12: Proceso de búsqueda del método NAS. Basado en Hutter et al., 2019

La mayoría de los métodos NAS utilizan el mismo conjunto de hiperparámetros para todas las arquitecturas candidatas durante toda la etapa de búsqueda. De esta manera, después de la búsqueda de las arquitecturas neuronales más prometedoras, es necesario llevar a cabo un rediseño del conjunto de hiperparámetros, y utilizar esté conjunto de hiperparámetros para reentrenar o perfeccionar la arquitectura (He et al., 2021).

Para entender el problema al cual se enfrenta NAS, es necesario examinar de cerca las características de una ANN (Benardos & Vosniakos, 2007). Los elementos de una ANN son los siguientes:

1. El número de capas
2. El número de neuronas en cada capa
3. La función de activación en cada capa
4. El algoritmo de entrenamiento (determina el valor final de pesos y sesgos)

Sin importar el número de capas que tenga la arquitectura, lo único certero es que deba contener una capa de entrada y de salida, de esta manera es capaz de obtener información y ofrecer una respuesta. En cuanto al número de neuronas, para las dos capas antes especificadas, dependerá del número de entradas y los parámetros de salida del problema a solucionar. Finalmente, el objetivo es poder encontrar el número de capas ocultas y el número de neuronas de cada una de esas capas. Sin embargo, no es posible determinar teóricamente cuantas capas ocultas o neuronas son necesarias para resolver el problema. Conformen a las funciones de activación, estas son seleccionadas dependiendo del tipo de dato que se encuentre disponible (binario, decimal, etc.) y el tipo de capa. Por último, el algoritmo de entrenamiento

influye más a las cuestiones de velocidad de entrenamiento, rendimiento de entrenamiento o a la potencia de cálculo necesario de la propia arquitectura.

2.4.2.1. Espacio de búsqueda

El espacio de búsqueda define como será la arquitectura que pueda ser representada en un principio. La incorporación de conocimiento previo sobre las propiedades que son apropiadas para una tarea puede reducir el tamaño del espacio de búsqueda. Sin embargo, esto también introduce un sesgo humano, que puede impedir nuevos bloques de construcción arquitectónicos que vayan más allá del conocimiento humano actual. El espacio de búsqueda define como será la arquitectura que el enfoque NAS pueda descubrir (Hutter et al., 2019).

Los espacios de búsqueda más comunes y frecuentemente utilizados son los siguientes: completamente estructurados, basados en células, jerárquicos, basado en morfismos. Una arquitectura neuronal poder ser representada por un Grafo Acíclico Directo (DAG por sus siglas en Inglés), este grafo está conformado por nodos ordenados Z y por aristas que conectan pares de nodos, donde cada nodo indica un tensor z y cada arista representa una operación o se selecciona de un conjunto de operaciones candidatas O. A su vez, debido a las limitaciones de los recursos computacionales que puedan estar disponibles, se establece un umbral máximo N para el grado (He et al., 2021). La Eq. 2.3 muestra el cálculo en un nodo k.

$$z^{(k)} = \sum_{i=1}^{N} o^{(i)}(z^{(i)}), o^{(i)} \in O \tag{2.3}$$

Donde O, representa al conjunto de operaciones candidatas que incluyen la operación primitiva, tales como: la convolución, *pooling*, funciones de activación, omitir conexiones, concatenación, y adición. De esta manera, el espacio de búsqueda define el paradigma estructural que los algoritmos de optimización de arquitectura pueden explorar.

2.4.2.2. Estrategia de optimización

La estrategia de búsqueda define como será explorado el espacio de búsqueda. Esta estrategia intenta abarcar un equilibrio entre la exploración y la explotación. Lo anterior se debe a que es deseable encontrar rápidamente las arquitecturas que presentan un buen rendimien-

to, pero debe evitarse la convergencia prematura a una región de arquitecturas subóptimas (Hutter et al., 2019).

2.4.2.3. Estimación de rendimiento

El objetivo principal del enfoque NAS es comúnmente encontrar las arquitecturas que tengan un alto rendimiento de predicción de datos no vistos. Sin embargo, la opción más simple de estimación de rendimiento es realizar un entrenamiento y validación estándar, pero realizar esta actividad es computacionalmente caro, y en consecuencia habría un alto límite en cuanto al número de arquitecturas que pueden ser exploradas. Para contrarrestar lo anterior la estimación de rendimiento puede ser realizada utilizando una menor fidelidad del rendimiento real, entre estas fidelidades se encuentran: Menores tiempos de entrenamiento, entrenar con un subconjunto de datos, imágenes a menor resolución, o con menores filtros por capa (Hutter et al., 2019).

2.4.3. Optimización bayesiana

El método de optimización bayesiana es una propuesta de marco de trabajo de optimización para la optimización global de funciones de caja negra costosas. La optimización bayesiana se caracteriza por ser un algoritmo iterativo que consta de dos componentes principales: Un modelo sustituto probabilístico y una función de adquisición para decidir cuál será el siguiente punto para evaluar. Durante cada iteración, el modelo sustituto es ajustado a todas las observaciones de la función objetivo que se han realizado hasta el momento. Posteriormente, la función de adquisición determina la utilidad de los diferentes puntos candidatos, utilizando la distribución predictiva del modelo probabilístico, mediante un intercambio entre la exploración y la explotación (Hutter et al., 2019).

Muchos de los avances de la optimización bayesiana ya no hacen uso del HPO como caja negra. Además, muchos de los desarrollos recientes en la optimización bayesiana no se dirigen a HPO, pero se pueden aplicar fácilmente a HPO. Tales como: nuevas funciones de adquisición, nuevos modelos y *kernels*, y nuevos esquemas de paralelización.

La optimización bayesiana es un algoritmo iterativo con dos partes principales: un modelo sustituto probabilístico y una función de adquisición para decidir cual es el siguiente punto a

evaluar. Durante cada iteración, el modelo sustituto se ajusta a todas las observaciones de la función objetivo realizadas hasta el momento. Después, la función de adquisición determina la utilidad de los diferentes puntos candidatos, intercambiando exploración y explotación.

En el algoritmo de optimización bayesiana, la primera población de soluciones, representadas en formato de tipo cadena, es generada de manera aleatoria. Partiendo de la población actual, las mejores soluciones son seleccionadas, utilizando cualquier criterio de selección. Posteriormente, se construye una red bayesiana que concuerde con el conjunto de soluciones construido, utilizando cualquier métrica como medida para calidad de las redes y cualquier algoritmo de búsqueda para maximizar el valor de esta. Por último, las nuevas soluciones son generadas utilizando la distribución conjunta codificada por la red construida, de esta manera las nuevas soluciones son añadidas a la población reemplazando a las desechadas (Pelikan et al., 1999, July). El algoritmo 2.1 muestra el proceso de optimización bayesiana.

Algoritmo 2.1 Optimización bayesiana. Tomado de Pelikan et al., 1999, July

```
1: t = 0
2: while t != numeroCandidatos do
3:     P = generarPoblacionInicial()          ▷ Se genera la población inicial P(0) de manera aleatoria.
4:     S = obtenerCadenas(P[t])               ▷ Se selecciona un conjunto de cadenas candidatas S(t) de P(t)
5:     B = construirRed(S)                    ▷ Se construye la red B utilizando las métricas y restricciones elegidas
6:     O = generarCadenas(B)                  ▷ Se genera un conjunto de nuevas cadenas O(t) de acuerdo con las
   características de B
7:     P(t+1) = generarNuevaPoblacion(P(t), O(t))   ▷ Se crea una nueva población P(t+1) reemplazando
   algunas cadenas de P(t) con O(t)
8:     t = t+1
9: end while
```

Otra forma de construir un modelo sustituto es el *Tree-Structured Parzen Estimator* (TPE). TPE fue desarrollado especialmente para la optimización de hiperparámetros porque aumenta en proporción al número de puntos evaluados y al número de variables optimizadas. El TPE ofrece al modelo sustituto un menor costo, pues modela cada variable individualmente, eliminando la posibilidad de correlación entre variables, pero al mismo tiempo ganando en tiempo de ejecución (Bergstra et al., 2011).

TPE modela $p(x|y)$ y $p(y)$, donde $p(x|y)$ es definido utilizando dos distribuciones $l(x)$ y $g(x)$. Las distribuciones son formadas al dividir el historial de puntos explorados H, mediante el uso del umbral y^* que toma el valor de una parte del total de puntos. La Figura 2.13 muestra una representación gráfica del comportamiento del algoritmo de optimización bayesiana TPE.

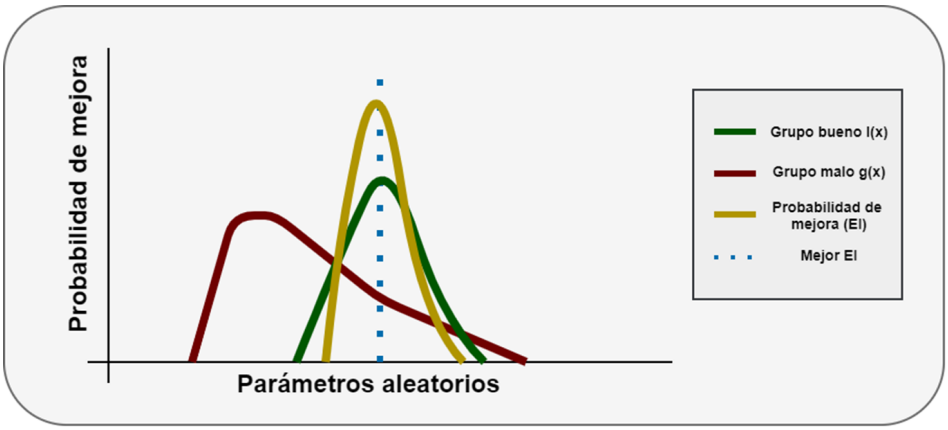

Figura 2.13: Gráfica de representación del proceso de optimización bayesiana.

Tal como se puede observar en la Figura 2.13, las distribuciones $l(x)$ y $g(x)$ son representadas por el grupo bueno y el grupo malo, respectivamente. El objetivo del algoritmo es generar candidatos con más probabilidades de caer en la distribución $l(x)$ que en la distribución $g(x)$. Durante cada iteración el algoritmo realiza un proceso de prueba en puntos sin explorar, para posteriormente regresar al punto x^* con una probabilidad de mejora mayor.

2.5. Series de tiempo

Una serie de tiempo es una secuencia de tiempo o cronológica de observaciones de una variable de interés. Por ejemplo, la Figura 2.14 muestra una secuencia de datos de una constante de 10 años, desde abril de 1953 hasta diciembre del 2006 (Montgomery et al., 2015).

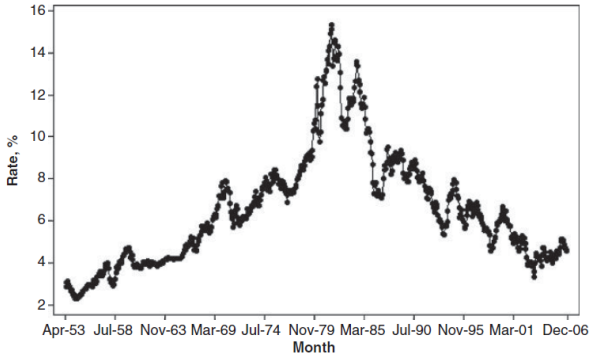

Figura 2.14: Gráfico de series de tiempo del rendimiento del "Mercado de valores de EE. UU." con una madurez constante de 10 años. Tomado de Montgomery et al., 2015

La tasa de la variable es recopilada en periodos de tiempo equitativamente espaciosos,

como se muestra en la Figura 2.14. Muchas de las aplicaciones comerciales de pronóstico utilizan datos diarios, semanales, mensuales, trimestrales o anuales, pero se podría utilizar cualquier intervalo deseado. Además, los datos deben ser instantáneos, acumulativos o estadísticos, de tal manera que, refleje la actividad de la variable durante el periodo de tiempo.

Los datos temporales son un tipo de dato que varía con el cambio en el tiempo. Estos datos se ven representados utilizando marcas de tiempo. Las series de tiempo son en esencia una colección de un gran número de datos en un intervalo de tiempo uniforme. Las series de tiempo son analizadas para predecir los cambios que ocurren dentro de los datos dados y para predecir los cambios que pasaran en el futuro. Esta es representada como una tupla de 5 valores, estos son: Tiempo de inicio, el patrón, el valor del período, la confianza y el tiempo final (Mahalakshmi et al., 2016).

2.6. Medidas de rendimiento

El rendimiento de un modelo puede ser medido de manera cuantitativa mirando subjetivamente un conjunto de resultados. Sin embargo, si el tamaño de los datos es grande, las pruebas subjetivas y cualitativas dejan de ser una fuente confiable de información con respecto al rendimiento general del sistema. Para evaluar de manera correcta las habilidades de un algoritmo de aprendizaje máquina, se debe diseñar una medida cuantitativa de su rendimiento (Joshi, 2020).

2.6.1. Medida para clasificación

Para las tareas de clasificación usualmente se mide la precisión del modelo. La precisión es solo la proporción de las muestras con respecto a las salidas correctas producidas por el modelo. La Eq. 2.4 muestra la fórmula para medir la precisión.

$$Precisión = \frac{Número\ de\ aciertos}{Número\ total\ de\ predicciones} \tag{2.4}$$

Otra forma de representar la precisión de clasificación es mediante la matriz de confusión. En esta matriz se muestra que tan buen rendimiento tiene un modelo o clasificador basado en

la respuesta correcta dada. La Figura 2.15 muestra la representación de la matriz de confusión (Patterson & Gibson, 2017).

	P' (Predicho)	N' (Predicho)
P (Actual)	Verdadero positivo (VP)	Falso negativo (FN)
N (Actual)	Falso positivo (FP)	Verdadero negativo (VN)

Figura 2.15: Matriz de confusión: Positivo (P), Negativo (N). Basado en Patterson & Gibson, 2017

Con base en las métricas presentadas en la matriz de confusión, se puede obtener un análisis más detallado sobre el desempeño del modelo más allá del porcentaje básico de conjeturas que fueron correctas. Se puede hacer un cálculo de evaluaciones diferentes del modelo basado en las combinaciones de la matriz de confusión. Las evaluaciones son:

2.6.1.1. Precisión

La precisión es el grado de cercanía de las mediciones de una cantidad al valor real. Se representa con la Eq. 2.5.

$$Precisión = \frac{TP + TN}{TP + FP + FN + TN} \tag{2.5}$$

2.6.1.2. Exactitud

Es el grado en que las mediciones repetidas en las mismas condiciones ofrecen los mismos resultados. Se representa con la Eq. 2.6.

$$Exactitud = \frac{TP}{TP + FP} \tag{2.6}$$

2.6.1.3. Recuperación

Recuperación (*Recall* en Inglés) es una medida que muestra la tasa de verdaderos positivos o tasa de aciertos. Se representa con la Eq. 2.7.

$$Recuperación = \frac{TP}{TP + FN} \tag{2.7}$$

2.6.1.4. F1

En la clasificación binaria, a la puntuación F1 se le considera una medida de la precisión del modelo. La puntuación F1 es la media armónica de las medidas de exactitud y recuperación en una única puntuación. Se representa con la Eq. 2.8.

$$f1Matrix = \frac{2TP}{2TP + FP + FN} \tag{2.8}$$

2.6.2. Medida para regresión

Para tareas de regresión existen varias maneras para evaluar el rendimiento del modelo. Estas métricas consisten en: si se tiene una lista de valores esperados $(y_i, i = 1, \ldots, n)$ y una lista de valores predichos $(\hat{y}_i, i = 1, \ldots, n)$, el error de predicción puede ser obtenido mediante diferentes formas. La primera de estas métricas es el Error Absoluto Medio (MAE por sus siglas en Inglés), aunque es una métrica que usualmente se evita, permite obtener valores bajos debido a la cancelación de los errores negativos y positivos. Se denota por la Eq. 2.9 (Joshi, 2020).

$$MAE = \frac{\sum_{i=1}^{i=n} |\hat{y}_i - y_i|}{n} \tag{2.9}$$

Otra de las métricas es el Error Medio al Cuadrado (MSE por sus siglas en Inglés). Por lo general, MSE penaliza un número mayor de errores grandes (valores atípicos) en comparación con un número mayor de errores pequeños. Esta métrica es denotada por la Eq. 2.10.

$$MSE = \frac{\sum_{i=1}^{i=n} (\hat{y}_i - y_i)^2}{n} \tag{2.10}$$

La siguiente métrica es la Raíz del Error Cuadrático Medio (RMSE por sus siglas en Inglés). Se caracteriza por reducir la sensibilidad del error a unos pocos valores atípicos,

pero sigue siendo más sensible en comparación con la métrica MAE. La Eq. 2.11 denota esta métrica.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{i=n}(\hat{y}_i - y_i)^2}{n}} \tag{2.11}$$

2.6.3. Medida para series de tiempo

Para evaluar el desempeño de las series de tiempo, se pueden implementar diferentes medidas cuantitativas. Una de las más populares para series de tiempo es la antes mencionada RMSE. Otra de las medidas más utilizadas es el Promedio del Error Porcentual Absoluto (MAPE por sus siglas en Inglés). La Eq. 2.12 denota la métrica MAPE.

$$MAPE = \frac{1}{T}(\sum_{t=1}^{T} \frac{|Y_t^{\beta} - Y_t^{\alpha}|}{Y_t^{\alpha}}) * 100 \tag{2.12}$$

Donde T es el número de periodos, Y_t^{α} es el valor pronosticado de Y_t, y Y_t^{β} es el valor real de Y_t.

2.6.4. Evaluación de modelo

Para seleccionar el modelo más adecuado para un problema o tarea especifica se requiere el entrenamiento de diversos tipos de modelos diferentes y configuraciones, y después evaluarlos. Usualmente el interés recae en que tan bien un algoritmo de aprendizaje máquina maneja los datos que no ha visto antes, debido a que esto es lo que determina que tan bien trabajará el algoritmo cuando se enfrente a situaciones reales (Bengio & Courville, 2017). En otras palabras importa más que el algoritmo tenga alto rendimiento y aprenda a generalizar con información nunca vista que con los datos de entrenamiento.

Existen varias formas de hacer que un modelo de aprendizaje máquina aprenda a generalizar. La forma más sencilla consiste en dividir los datos en dos conjuntos, estos son: el conjunto de entrenamiento y el conjunto de prueba. Aunque la cantidad de muestras para cada conjunto varía dependiendo de la cantidad de ejemplos disponibles, una medida comúnmente utilizada es 80 % de los valores para el conjunto de entrenamiento y el 20 % restante para el

conjunto de prueba. Sin embargo, a pesar de que es una solución sencilla y puede resultar efectiva, existe el problema de que si se mide el error de generalización múltiples veces con el conjunto de prueba, es muy probable que el modelo se haya adaptado también al conjunto de prueba. Esto quiere decir que es poco probable que el modelo funcione bien con datos nuevos (Géron, 2019).

La solución a este problema es sencilla, se necesita crear un tercer conjunto de datos a partir de la partición del conjunto de entrenamiento, a este conjunto se le llama conjunto de validación. Su objetivo consiste en utilizar el conjunto de entrenamiento y el conjunto de validación para crear un modelo que se adapte bien a esos datos, y después se realiza una última prueba con el conjunto de prueba para obtener un estimado del error de generalización del modelo. De esta manera el conjunto de prueba no influye de ninguna manera en el entrenamiento del modelo y se puede verificar su grado de generalización.

2.7. Sistemas distribuidos

En este proyecto es necesario la implementación de múltiples módulos que funcionen en conjunto, además de que sean flexibles y que tengan un alto rendimiento durante su implementación. Para lograr lo anterior planteado, es necesario desarrollar un sistema que implemente las tecnologías de sistemas distribuidos y el paralelismo.

Un sistema distribuido es todo aquel en el cual sus componentes son localizados a través de una red de computadoras comunicando y coordinando sus acciones mediante el pase de mensajes. Lo anterior lleva a definir las características significantes de un sistema distribuido: Concurrencia de componentes, falta de un reloj global y fallas independientes de componentes (Coulouris et al., 2012).

Los sistemas distribuidos se caracterizan por ser aplicaciones que son creadas para múltiples aplicaciones diferentes, ejecutándose en diferentes equipos, o muchas réplicas ejecutándose entre diferentes equipos, todas comunicándose juntas para implementar un sistema. Debido a su naturaleza distribuida, los sistemas distribuidos son extremadamente confiables, cuando se estructuran de manera adecuada. Estos pueden conducir a modelos organizativos mucho más escalables (Burns, 2018).

2.7.1. Procesamiento por lotes

El procesamiento por lotes está diseñado para que los procesos por lotes solo se ejecuten durante un período breve de tiempo. Entre los ejemplos de este proceso se encuentran: generación de agregación de datos de telemetría de usuario, análisis de datos de ventas para informes diarios o semanales, o la transcodificación de archivos de vídeo. Los procesos por lotes generalmente son utilizados porque permiten el procesamiento de grandes volúmenes de datos rápidamente, mediante la utilización del paralelismo para acelerar el procesamiento (Burns, 2018).

El modelo más simple de procesamiento por lotes es por medio de una cola de trabajo. En un sistema de cola de trabajos, existe un lote de trabajo por realizar. Cada trabajo es independiente de otro trabajo y puede ser procesado sin interacciones entre ellos. Generalmente, el objetivo de las colas de trabajo son garantizar que cada trabajo se procese dentro de un cierto período de tiempo. La Figura 2.16 muestra un modelo de cola de trabajo genérica.

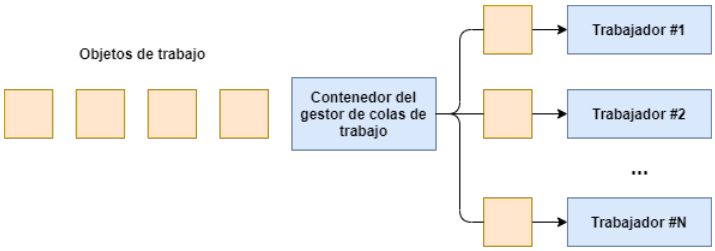

Figura 2.16: Modelo de cola de trabajo genérica. Basado en Burns, 2018

2.7.2. Intercambio de mensajes

La comunicación entre una red de computadoras puede realizarse de diferentes maneras en un sistema distribuido. Esta comunicación tiene una serie de características de rendimiento que se relacionan con la latencia, el ancho de banda y la fluctuación (Coulouris et al., 2012):

- **Latencia:** Es la demora entre el inicio de la transmisión del lenguaje de un proceso y el comienzo de su receptor por otro.
- **Ancho de banda:** Es el monto total de información que puede ser transmitida sobre una red de computadoras en un tiempo dado. Cuando se utiliza un gran número de

canales de comunicación en la misma red, estos tienen que compartir el ancho de banda disponible.

- **Fluctuación:** Es la variación en el tiempo tomado para entregar una serie de mensajes. La fluctuación es relevante en cuestiones de datos multimedia.

Una manera de realizar la implementación de intercambio de mensajes es mediante un intermediario de mensajes. El Protocolo Avanzado de Cola de Mensajes (AMQP por sus siglas en Inglés) es un protocolo Máquina a Máquina (M2M) ligero, el cual está diseñado para ofrecer confiabilidad, seguridad, aprovisionamiento e interoperabilidad. Además, el protocolo AMQP soporta arquitecturas de petición/respuesta y publicación/subscriptor (Naik, 2017, October).

Para iniciar una comunicación AMQP, este requiere el publicador o el consumidor creen un intercambio (*exchange* en Inglés) con un nombre para después mandar una retransmisión (*broadcast* en Inglés) con ese nombre. De esta manera, el publicador y el consumidor puede descubrirse unos a otros. Posteriormente, el consumidor se encarga de crear una cola (*queue* en Inglés) y esta es adjuntada al intercambio al mismo tiempo. Finalmente, el publicador manda mensajes al *exchange*, y este se encarga de rutear a la cola correspondiente, a este proceso se le denomina *binding*. El protocolo AMQP realiza el intercambio de mensajes de diferentes maneras: directa, en forma de abanico, por tema o basado en encabezados.

2.8. Tecnologías

Para el desarrollo de este proyecto de investigación fueron necesarias la implementación de múltiples sistemas elaborados en diferentes herramientas de desarrollo, lenguajes de programación, frameworks y librerías. El proyecto principal está desarrollado en lenguaje Python, mediante la aplicación de librerías como TensorFlow 2.2, para la implementación de modelos de redes neuronales, Keras, para la construcción de arquitecturas de redes neuronales, y Optuna, para la implementación de la optimización bayesiana. La gestión de librerías por parte de Python se realizó con un plugin del mismo Python, Venv nos permite crear ambientes digitales para afectar el entorno propio de la aplicación. A su vez el intercambio de mensajes entre nodos se realiza mediante la aplicación RabbitMQ.

Otro lenguaje de programación utilizado es C# para la implementación de la interfaz de usuario desarrollada para manejar la aplicación de Python. La comunicación entre el lenguaje C# y Python se realiza mediante la librería de Python FlaskSocketIO para una comunicación a tiempo real, mediante la implementación de *sockets*, entre ambas plataformas. A su vez el paquete de librerías de C# *Windows Forms* soporta la implementación de la interfaz gráfica.

Para el desarrollo de la aplicación se utilizaron los editores Visual Studio Code, para la implementación de Python, y Visual Studio 2019, para la implementación de C#. A su vez, el controlador de versiones Git y el protocolo de comunicación SSH fueron de gran utilidad para gestionar la implementación del sistema en múltiples equipos.

2.8.1. Python

Lenguaje interpretado con sintaxis caracterizada por ser sencilla y elegante. Es un lenguaje de propósito general, y ofrece un ambiente apropiado para aplicaciones científicas, lo cual lo hace una buena opción para los científicos (Oliphant, 2007). Entre las características que ofrece Python se encuentra:

- Licencia de código abierto que permite la venta, utilización o distribución de aplicaciones escritas en Python.
- Permite la disponibilidad de ejecutar aplicaciones en múltiples plataformas.
- Permite la construcción de aplicaciones sofisticadas de manera procedural u orientada a objetos.
- Tiene la habilidad de interactuar con una gran variedad de otros software.
- Contiene una gran cantidad de módulos de librerías, esto significa que se puede construir programas más sofisticados.
- Tiene una comunidad famosa para recibir respuestas rápidas y útiles a las consultas de usuarios.

2.8.2. TensorFlow

TensorFlow es una interfaz creada para expresar y ejecutar algoritmos de aprendizaje máquina. Un programa que utilice TensorFlow puede ejecutar, con poco o ningún cambio, en una amplia variedad de sistemas heterogéneos, desde dispositivos pequeños, como móviles, hasta sistemas distribuidos a gran escala de cientos de máquinas y dispositivos computacionales como tarjetas GPU. Entre las aplicaciones que puede tener se incluyen el entrenamiento y la inferencia de algoritmos para modelos de redes neuronales profundas. La API de TensorFlow fue desarrollada en noviembre de 2015 por Google y fue lanzado como paquete de código abierto bajo la licencia de Apache 2.0 (Abadi et al., 2016).

Es una librería de software que está bien adecuada y ajustada para el aprendizaje automático a gran escala. Su principio básico consiste en definir un grafo de cálculos en Python a implementarlo, luego TensorFlow toma el grafo definido y lo ejecuta de manera más eficiente, utilizando código optimizado en C++. Lo más importante de este procedimiento, es que el grafo puede ser dividido en varias partes y ejecutar dichas partes en paralelo entre múltiples CPU o GPU. De esta manera, TensorFlow permite el entrenamiento de redes con millones de parámetros en un conjunto de entrenamiento compuesto por billones de instancias con millones de características cada uno.

2.8.3. Rabbit MQ

RabbitMQ es un corredor (*broker* en Inglés) de mensajes de código abierto que implementa un estándar AMQP. Fue lanzado en el año 2007, hasta que la última versión más estable apareció en 2015. Esta aplicación fue desarrollada en el lenguaje de programación Erlang y está basado en el framework de “Plataforma de telecomunicaciones abierta”. De entre las características más importantes de RabbitMQ están: muchos servidores RabbitMQ de una red local pueden ser agrupadas juntas, formando un solo *broker* lógico, permitiendo la implementación de características como el balance de carga y la tolerancia a fallos; Otra característica es el protocolo AMQP que utiliza para aceptar las conexiones entre las diferentes plataformas (Ionescu, 2015). La Figura 2.17 muestra cada paso de la implementación de RabbitMQ.

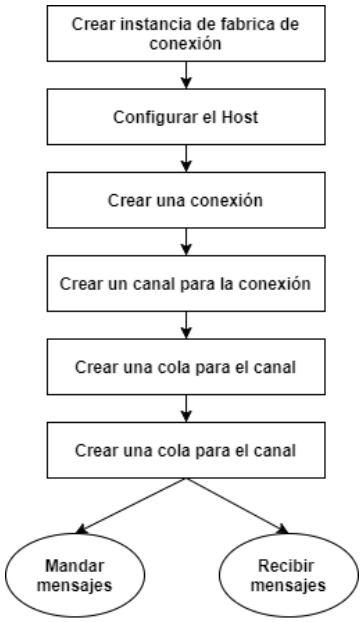

Figura 2.17: Implementación de RabbitMQ. Basado en Ionescu, 2015

2.8.4. C#

C# es un lenguaje de programación moderno, orientado a objetos y de tipado seguro. Permite a los desarrolladores la construcción de muchos tipos de aplicaciones seguras y robustas que son ejecutadas en un ecosistema .NET. Además, es un lenguaje de programación orientado a componentes y provee un soporte para construir aplicaciones con estos conceptos (Microsoft, 2021)

Uno de los frameworks más utilizados en el lenguaje de programación de C# es Windows Forms, que permite construir aplicaciones de escritorio de Windows. Este framework provee uno de las formas más productivas de crear aplicaciones de escritorio basado en un diseño visual. Contiene funciones como el arrastrar y soltar, esto facilita la creación de aplicaciones de escritorio. Con esta aplicación, se pueden desarrollar aplicaciones ricas en gráficos que son fáciles de desplegar, actualizar y trabajar en línea o no en internet.

2.8.5. Flask SocketIO

Es una librería especializada en ofrecer a aplicaciones Flask, para conexiones HTTP en el lenguaje de Python, con acceso a comunicaciones bidireccionales de baja latencia entre los

clientes y el servidor. Por el lado del cliente, este puede utilizar cualquiera de las librerías oficiales de SocketIO, ya sea en JavaScript, C++, Java, Swift, C# O cualquier cliente compatible que permita establecer conexiones permanentes al servidor (Grinberg, 2021).

Capítulo 3

Estado del arte

En la actualidad existen equipos de cómputo cada vez más potentes y están cada vez más disponibles al público general. Lo anterior ha generado un creciente interés en la utilización de técnicas de AutoML para buscar soluciones óptimas a problemas de aprendizaje profundo en el menor tiempo posible. En este trabajo se utiliza el área de *Neural Architecture Search* (NAS), subárea del AutoML, que realiza el procedimiento necesario para la búsqueda de arquitecturas y la optimización de los hiperparámetros de esas arquitecturas. La arquitectura NAS comprende tres partes principales: Espacio de búsqueda, Estrategia de búsqueda y Estrategia de estimación de rendimiento. Cada una de las partes anteriores puede ser implementada de diferentes maneras y con diferentes combinaciones.

3.1. Optimización de hiperparámetros

Una de las posibles soluciones para el aprendizaje máquina automatizado se basa en la optimización de hiperparámetros (HPO), donde solo se mejoran los hiperparámetros sin optimizar la arquitectura. Algunos trabajos han utilizado este enfoque y han presentado que la arquitectura implementada cambia poco durante el proceso de optimización, además, los espacios de búsqueda son más limitados en general.

Uno de los trabajos que utilizan está implementación es el *Hyperband*, una alternativa a los métodos de optimización bayesiana, debido a que se enfoca más a la exploración. Durante su implementación se utiliza una búsqueda aleatoria junto con una asignación adaptativa de recursos y *Early Stopping*. Su idea consiste en la inicialización del proceso con una cantidad

n de modelos candidatos con recursos limitados, tales como: épocas, tiempo, batch, etc. Durante cada ronda, los modelos menos prometedores son desechados, y se amplía la cantidad de recursos utilizados (Li et al., 2017).

Otro trabajo de HPO es el *Bayesian Optimization Hyperband* (BOHB), donde se utiliza la combinación de Hyperband con optimización bayesiana. En el artículo de BOHB presentan las características que son deseables que contengan los algoritmos de HPO, entre estos se encuentran: el rendimiento en cualquier momento (*Anytime performance* en Inglés) y el rendimiento final (*Final performance* en Inglés). La primera característica es tomada mediante la fase de experimentación, que suele presentar el rendimiento de múltiples modelos en periodos cortos, la implementación *Hyperband* funciona adecuadamente para esta característica. La segunda característica habla sobre el rendimiento obtenido mediante la implementación de un gran presupuesto de tiempo y hardware, la implementación del algoritmo bayesiano logra obtener buenos rendimientos finales pues aprende de su pasado (Falkner et al., 2018).

3.2. Búsqueda de arquitectura neural

Otra solución para el problema del aprendizaje máquina automatizado es la NAS, especificado en el capítulo anterior. Las estrategias de búsquedas implementadas para este problema son: Aprendizaje por refuerzo, algoritmos genéticos, optimización bayesiana y búsqueda aleatoria. El objetivo concreto de este método es optimizar la arquitectura del modelo de aprendizaje máquina.

Uno de los trabajos para la solución de NAS aplica una Red Neuronal Recurrente (RNN), para generar descripciones de modelos, entrenada por medio del aprendizaje por refuerzo, para maximizar la exactitud de los modelos con el Dataset de validación. La RNN permite generar cadenas con longitud variable, estas cadenas representan la arquitectura de los modelos, y estos son entrenados y evaluados con el Dataset de validación. Las medidas de rendimiento obtenidas retroalimentan al algoritmo de aprendizaje por refuerzo (Zoph & Le, 2016).

Otro trabajo se enfoca en la utilización de algoritmos genéticos mediante la utilización del método *Cartesian Genetic Programming* (CGP). Este método permite la construcción de

un grafo acíclico dirigido, y representa al espacio de búsqueda. Donde el espacio de búsqueda se compone de bloques funcionales en lugar de capas de red neuronal. Lo anterior permite aumentar la eficiencia de la búsqueda de la optimización (Suganuma et al., 2017).

En el siguiente trabajo, denominado Auto-Keras, se propone utilizar una optimización bayesiana junto con el morfismo de redes (*Network Morphism* en Inglés). Su implementación consta de la creación de modelos a partir de modelos previamente entrenados mediante la agregación de nuevas capas y evitar así volver a entrenarlo (Jin et al., 2019).

3.3. Tabla comparativa

La medida de rendimiento utilizada en este trabajo, para evaluar el modelo generado, consiste en una métrica de eficiencia llamada "Días GPU", que evalúa la cantidad de GPU con respecto a la cantidad de días que toma general el modelo. Aunque la medida dependa del hardware implementado, algunos proyectos han logrado resultados similares con cantidades dispares de recursos de Hardware (He et al., 2021). La Eq. 3.1 muestra el cálculo de la medida de rendimiento.

$$gpuDays = NxD \tag{3.1}$$

Donde N es la cantidad de GPU en uso y D es la cantidad de días que toma la implementación del proceso de estimación. La Tabla 3.1 muestra una comparación de los resultados obtenidos mediante la implementación de cada método utilizando el Dataset CIFAR-10.

Tabla 3.1: Comparativa de métodos de HPO y NAS con el *Dataset* CIFAR-10

Método	**Referencia**	**# Params (millones)**	**Precisión (%) de CIFAR-10**	**Días GPU**	**Tipo**
Hyperband	Li et al., 2017	—	83.00 %	0.5	HPO
BOHB	Falkner et al., 2018	8.5	97.22 %	33	HPO
NAS v3 max pooling	Zoph & Le, 2016	7.1	95.53 %	22,400	NAS
CGP-ResSet	Suganuma et al., 2017	1.68	94.02 %	27.4	NAS
Auto-Keras	Jin et al., 2019	—	88.56 %	0.5	NAS

En la Tabla anterior se puede observar el tipo de método implementado en los diferentes trabajos recolectados. También se especifican la cantidad de parámetros que el modelo resultante contiene y la precisión del mismo implementando el conjunto de datos de CIFAR-10. A su vez, se especifica la cantidad de días GPU necesarias para la implementación del método y el tipo de arquitectura utilizada.

3.4. Pronóstico de series de tiempo de imágenes

El pronóstico de series de tiempo se puede encontrar en diferentes áreas. Entre las áreas se encuentran: la Economía, Energía, Medicina o la Ingeniería (Gamboa, 2017). Mediante la utilización de diversas técnicas o métodos, se pueden obtener estimaciones de un problema dado, minimizando la incertidumbre lo más posible (Brownlee, 2018). Entre las técnicas que existen para diseñar, implementar y evaluar pronósticos de series de tiempo existen: Redes neuronales artificiales (ANN por sus siglas en Inglés), Media Móvil Integrada Auto-Regresiva (ARIMA por sus siglas en Inglés), Máquinas de Soporte Vectorial (SVM por sus siglas en Inglés), Razonamiento basado en casos (CBR por sus siglas en Inglés), Series de tiempo difusas, Modelo de predicción gris, Media Móvil y Suavizado Exponencial (MA & ES por sus siglas en Inglés), K-Vecinos cercanos (KNN por sus siglas en Inglés) y Modelos híbridos. Donde, la utilización de las técnicas antes mencionadas ofrece una serie de ventajas y desventajas. Mediante la utilización de los modelos híbridos, permiten tomar ventajas de las fortalezas y debilidades que cada uno de los métodos presentan (Deb et al., 2017).

Existen diferentes enfoques para el pronóstico de series de tiempo. Muchos de estos enfoques hacen uso de las características distintivas de las imágenes, definidas por la aplicación especificada, para realizar un pronóstico basado en una secuencia de imágenes. Uno de los enfoques básicos de pronóstico es utilizar dos imágenes consecutivas para obtener un pronóstico de un vector de características. En el trabajo de Magnone et al., 2017 se utiliza el enfoque antes descrito, y se asume que todos los objetos de las imágenes se trasladan en la misma dirección. Con base en lo anterior planteado se encuentra el mejor vector que representa la traslación de los objetos.

El enfoque antes descrito, puede ser interpretado de manera diferente. Se puede inter-

pretar que las imágenes no son consecutivas, pero los píxeles se intensifican y permanecen constantes en el tiempo. En el trabajo de Chow et al., 2011 Utilizan un enfoque basado en la estimación de vectores de movimiento por el método de correlación cruzada, mediante la partición de imágenes en subconjuntos de píxeles que tengan el mismo tamaño, además, se asigna el vector que une los subconjuntos de imágenes consecutivas con el mayor coeficiente de correlación cruzada.

Existen múltiples métodos para realizar pronósticos de series de tiempo de imágenes. Estos métodos pueden caer en las categorías de Aprendizaje Máquina (ANN, SVM, KNN, árboles de decisión y bosques al azar), modelos estadísticos (Modelos de regresión, ARMA, filtro Kalman y modelo trigonométrico armónico), o modelos varios (*Cellular automata*, teoría de sistemas, modelo de transmitancia y velocimetría de imagen de partículas).

Las aplicaciones del pronóstico de series de tiempo de imágenes pueden ser clasificadas dependiendo del área al que están enfocadas. Entre las aplicaciones se encuentran: Urbanización y Cambio de Uso del Suelo (UCUS), agricultura, peligros naturales, cambio global, energía solar y Monitoreo e Ingeniería Costera (MIC). La Tabla 3.2 muestra una comparativa de diferentes trabajos, sobre el pronóstico de series de tiempo de imágenes, enfocado a las técnicas que implementan y el área de aplicación a la cual se enfocan.

Tabla 3.2: Técnicas y aplicaciones de pronóstico de series de tiempo de imágenes

Aplicación / Técnica	UCUS	Agricultura	Peligros naturales	Cambio global	Energía renovable	MIC
ANN	Pal et al., 2016; Das & Ghosh, 2016; Ma et al., 2018	Zambrano et al., 2018; Hao et al., 2018; Bose et al., 2016	Yuan et al., 2020	G. Zhang et al., 2020	Marquez et al., 2013; Wen et al., 2020; Dong et al., 2014; Xu et al., 2019	
SVM						Cornejo-Bueno et al., 2016
KNN					Ayet & Tandeo, 2018	
Árboles de decisión		Nduati et al., 2019	Berhan et al., 2013		Carriere et al., 2019	
Modelos de regresión			Yan et al., 2017		Dong et al., 2014	Mafi-Gholami et al., 2020
ARMA					Carriere et al., 2019; Dambreville et al., 2014	
Filtro Kalman					Cheng & Yu, 2016	
Modelo trigonométrico armónico						
Cellular automata	Rafiee et al., 2009; Norman et al., 2009; Aliani et al., 2019					Mafi-Gholami et al., 2020
Teoría de sistemas	Meng et al., 2011					
Modelo de transmitancia					Alonso-Montesinos et al., 2019	
Velocimetría de imagen de partículas					Yang et al., 2019	

Cada uno de los trabajos descritos en la Tabla 3.2 presentan diferentes características en la forma en que realizan el proceso de pronóstico. Como puede ser en el trabajo de W. Zhang et al., 2018 donde se realiza un análisis de sensibilidad de la respuesta sobre el crecimiento de la altura del tallo, el índice de área foliar y la biomasa al número de días posterior a la siembra. También es posible utilizar SVM, como en el trabajo de Cornejo-Bueno et al., 2016 donde hacen un trabajo de análisis de la evolución espacio-temporal de la superficie del mar mediante la extracción de series de tiempo de imágenes de radar, enfocándose en determinar si se pueden separar los puntos de datos con un hiperplano (k-1) dimensional.

Otro de los enfoques que ha sido utilizado de manera exitosa es la aplicación del aprendizaje profundo, mediante la utilización de Redes Neuronales de Convolución (CNN por sus siglas en Inglés), para el manejo de imágenes, y la utilización de Redes de Gran Memoria de

Corto Plazo (LSTM por sus siglas en Inglés), para el manejo de características temporales. En el trabajo de Ma et al., 2018 se hace una transformación de una red de usuarios de metro en imágenes mediante una CNN para la obtención de información espacial de los datos. De manera separada, los datos temporales son manipulados por una LSTM para extraer de manera apropiada las características temporales, y posteriormente combinar estas características con una ANN que realiza el proceso de pronóstico del número de pasajeros del metro.

Capítulo 4

Metodología

En este capítulo se presenta la metodología propuesta para el diseño e implementación de este proyecto. Esta sección puede ser dividida en partes por los diferentes temas que la conforman. En un principio se describirá la arquitectura general del sistema, la arquitectura con formato distribuido y la comunicación entre los diferentes nodos existentes. Posteriormente, se detallará el proceso llevado a cabo para la construcción y evaluación de los modelos de aprendizaje profundo, el preprocesamiento de datos y la comunicación de la interfaz gráfica con el sistema de optimización. Finalmente, se describirá el módulo de pronóstico de series de tiempo de imágenes que abarca la construcción del autocodificador, codificación y decodificación de muestras, entrenamiento de múltiples modelos, el pronóstico del siguiente paso, y la obtención de imágenes.

La presente metodología se encuentra basada en el trabajo previamente elaborado por el MCC. Mario Valenzuela Partida. Donde se crea un ambiente distribuido para la generación y optimización de modelos de aprendizaje profundo capaces de obtener soluciones a problemas de clasificación de imágenes. Con base en el proyecto previo, se mejoraron y agregaron nuevos apartados para ofrecer la capacidad de generar soluciones a diferentes tipos de problemas. Más detalladamente, todo el apartado de generación y optimización se mantiene igual al proyecto previo, así como el espacio de búsqueda para clasificación de imágenes. A su vez, se mantiene el formato de comunicación entre los nodos maestro y trabajador.

4.1. Arquitectura general del sistema

La arquitectura planteada en este trabajo se basa en un modelo de NAS que fue presentado en el Capítulo 2, concretamente en la Figura 2.12. Las partes principales que se pueden obtener de este modelo son: Espacio de búsqueda, estrategia de búsqueda y estrategia de estimación de rendimiento. Además, el sistema contempla la implementación de búsqueda de soluciones a diferentes tipos de problemas. La Figura 4.1 muestra la arquitectura las propuestas para el funcionamiento de estas partes.

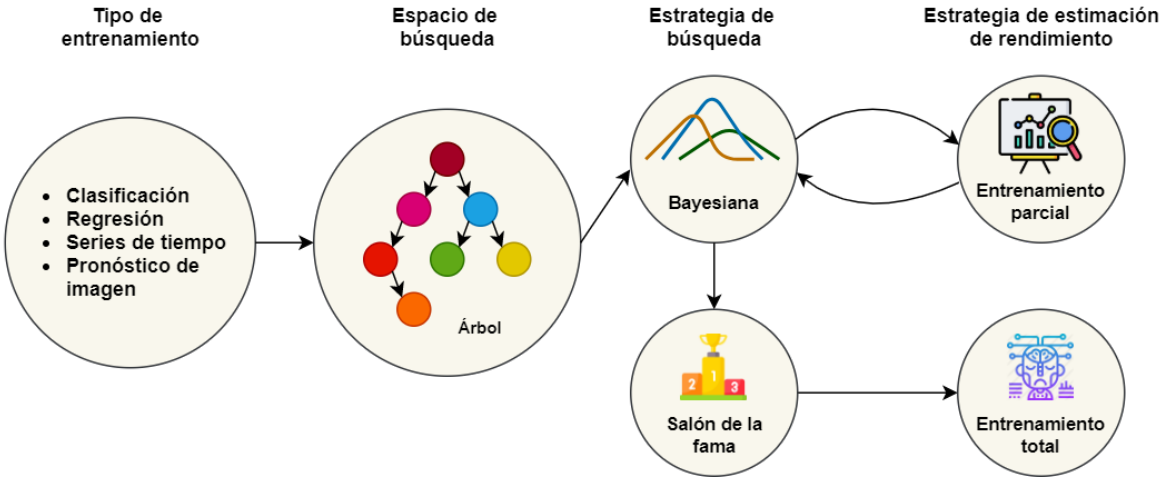

Figura 4.1: Funcionamiento general del sistema

En primera instancia se define el tipo de entrenamiento o solución que se aplicará para el problema especificado. Posteriormente, dependiendo del tipo de problema se define el espacio de búsqueda que se utilizará. En el espacio de búsqueda se definen condicionales que permitan limitar los componentes subyacentes, teniendo en cuenta los componentes que se han agregado previamente al modelo. De esta manera, se permite la incorporación de múltiples rutas o arquitecturas base dentro del mismo espacio de búsqueda y evitar combinaciones innecesarias. La implementación del espacio de búsqueda permite definir: Tipos de capas a implementar, cuantas capas tendrá la arquitectura y la cantidad de unidades por capa.

El espacio de búsqueda es implementado por medio de un árbol, lo anterior es debido a la implementación del algoritmo *Tree-structured Parzen Estimator* (TPE). El TPE guía a la estrategia de búsqueda, debido a que ha demostrado tener un buen rendimiento en HPO

y NAS, pues permite a los algoritmos de optimización navegar por espacios de búsqueda condicionales. Gracias a su disponibilidad de implementación con código abierto es posible aplicar esta estrategia dentro del proyecto. La estrategia de estimación de rendimiento aplica un entrenamiento parcial, complementado por un salón de la fama (HoF por sus siglas en Inglés) para mantener una clasificación de los modelos que tengan mayor rendimiento durante la fase de TPE.

Mediante esta estrategia de estimación de rendimiento, se evita realizar un entrenamiento completo de cada uno de los modelos de la fase de exploración, de tal manera que, permite al sistema reducir el tiempo total de optimización debido a que no tiene que realizar entrenamientos completos. Este entrenamiento parcial consiste en realizar un entrenamiento con un número reducido de épocas y junto con un *Early Stopping*, si el entrenamiento no va bien, será parado antes de finalizar. El componente de HoF utiliza las métricas obtenidas durante el entrenamiento parcial, pasando las mejores a un entrenamiento profundo.

La arquitectura general del sistema consiste en una comunicación de un nodo maestro que maneja el proceso de optimización. Este nodo maestro se comunica con un sistema de colas por parte del *Broker* RabbitMQ. A su vez, un conjunto de trabajadores se suscriben al *Broker* RabbitMQ para escuchar las peticiones del nodo maestro y retornar la respuesta de entrenamiento del modelo. La Figura 4.2 muestra la comunicación entre los componentes principales.

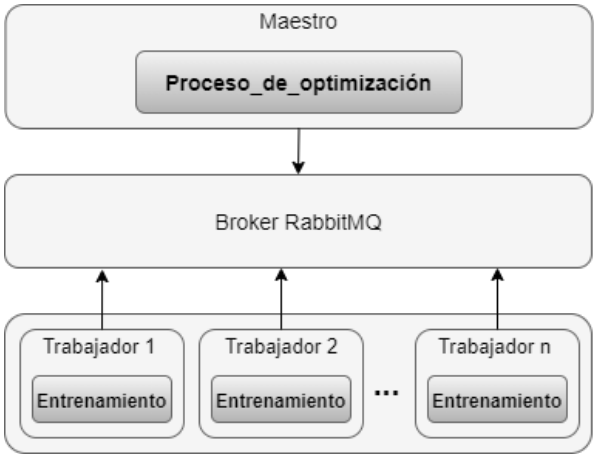

Figura 4.2: Arquitectura general del proceso de optimización

4.1.1. Arquitectura distribuida

En este trabajo se implementa una arquitectura distribuida para realizar entrenamientos de modelos de redes neuronales. Un nodo maestro se encarga de definir el espacio de búsqueda, de tal manera que, construye y verifica el modelo. Este modelo posteriormente es mandado a un conjunto de nodos trabajadores por medio de un intermediario. Para evitar mandar modelos completos, los modelos generados se transportan mediante diccionarios codificados en JSON entre los nodos y el *Broker*. La Figura 4.3 muestra la manera en la cual se realiza la comunicación entre nodos mediante RabbitMQ.

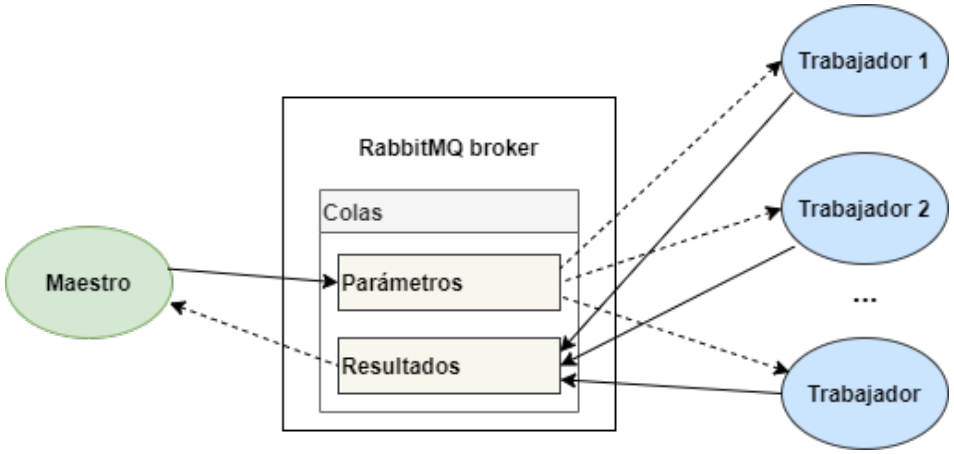

Figura 4.3: Comunicación desglosada del nodo maestro con los nodos trabajador

El nodo Maestro se encarga de definir y enviar un conjunto de parámetros a la cola "parámetros", a la cual los nodos trabajadores que estén subscritos pueden escuchar. Posteriormente los nodos trabajadores envían los resultados a la cola "resultados", a la cual está subscrito el nodo maestro para tomar los resultados y mandarlos al algoritmo Bayesiano y generar los siguientes modelos.

4.1.1.1. Petición de entrenamiento de modelos

Una petición de entrenamiento sucede cuando un nodo maestro manda al *Broker* un conjunto de parámetros para ser tomado por un trabajador. Esta petición contiene la estructura de la arquitectura del modelo, el conjunto de parámetros de entrenamiento y los *Hashes* de verificación. Los mensajes son codifican en un formato JSON y contiene el siguiente conjunto

de datos:

- ID: Identificar del modelo del proceso de optimización.
- Arquitectura: Conjunto de capas e hiperparámetros generados a partir del espacio de búsqueda para la construcción del modelo.
- Épocas: Cantidad de épocas que tomará el entrenamiento del modelo, varía dependiendo si es entrenamiento parcial o completo.
- Paciencia *Early Stopping*: Parámetro para controlar el número de épocas seguidas donde no haya mejora en el entrenamiento.
- Tipo de espacio de búsqueda: Etiqueta que define el tipo de espacio de búsqueda que será implementado.
- *Hash* del espacio de búsqueda: Es un valor que se genera por una función *Hash* sobre el contenido del espacio de búsqueda, de esta manera se asegura la compatibilidad entre los nodos.
- Etiqueta del *Dataset*: Etiqueta para identificar el *Dataset* que será utilizado para cargar y utilizar para entrenamiento.
- Es Entrenamiento parcial: Valor booleano que especifica si es un entrenamiento parcial o completo.

4.1.1.2. Respuesta de entrenamiento de modelos

Una respuesta de entrenamiento sucede cuando un nodo trabajador manda al *Broker* información sobre el resultado del entrenamiento realizado. La respuesta representa los valores de estimación de rendimiento de los modelos entrenados. Los mensajes se codifican en formato JSON y contienen la siguiente información:

- ID: Identificador del modelo entrenado, corresponde con el ID de la petición.
- Rendimiento: Valor de rendimiento del modelo entrenado, se genera mediante una prueba con el conjunto de datos de prueba.

- Épocas terminadas: Valor booleano que indica si el modelo termino de entrenarse completamente con las épocas especificadas en la petición o si paro por el *Early Stopping*.

4.1.2. Proceso del nodo maestro

El nodo maestro maneja el proceso principal de la aplicación. Este nodo se encarga de coordinar el proceso de optimización, definir la arquitectura de los modelos y los parámetros de entrenamiento. Además, se encarga de la comunicación entre los nodos. En el nodo maestro se alojan los componentes del espacio de búsqueda y la estrategia de búsqueda.

El proceso de optimización funciona de la siguiente manera. En primera instancia se establece una comunicación con el *Broker* y hace una petición para obtener el número de n trabajadores que están conectados. El número de modelos a generar por el modelo es definido por $n + 1$, lo anterior con el fin de que la cola siempre exista un modelo listo para entrenar y evitar que queden trabajadores sin utilizar.

Una vez que un nodo trabajador retorna una respuesta, el nodo maestro maneja esa respuesta, interpreta los resultados e informa al módulo de estrategia de búsqueda. La estrategia de búsqueda toma la decisión de que acción tomar a continuación y retorna una respuesta al proceso principal. Al recibir una respuesta, el nodo maestro tiene un conjunto de acciones que pueden realizar, estas son:

- Generar modelo: Indica al proceso principal que realice una llamada a la estrategia de búsqueda para generar un nuevo modelo.

- Esperar: Indica al proceso principal no generar más modelos de búsqueda todavía, debido a que está todavía en espera de los resultados de los entrenamientos que se están generando. A su vez, el proceso espera si queda un último modelo por entrenar y ya terminaron los otros trabajadores.

- Comenzar nueva fase: Avisa al proceso principal que ha terminado la fase de exploración del proceso de optimización y comienza la fase de entrenamiento profundo

- Finalizar: Notifica al proceso principal que se ha completado el proceso de optimización, posteriormente este proceso despliega la información sobre el proceso de optimi-

zación, detalles de arquitectura y detalles de rendimiento del mejor modelo.

La Figura 4.4 muestra el diagrama de secuencia utilizado para la implementación de inicialización de proceso de optimización, y posteriormente se envían las primeras peticiones de entrenamiento al *Broker*.

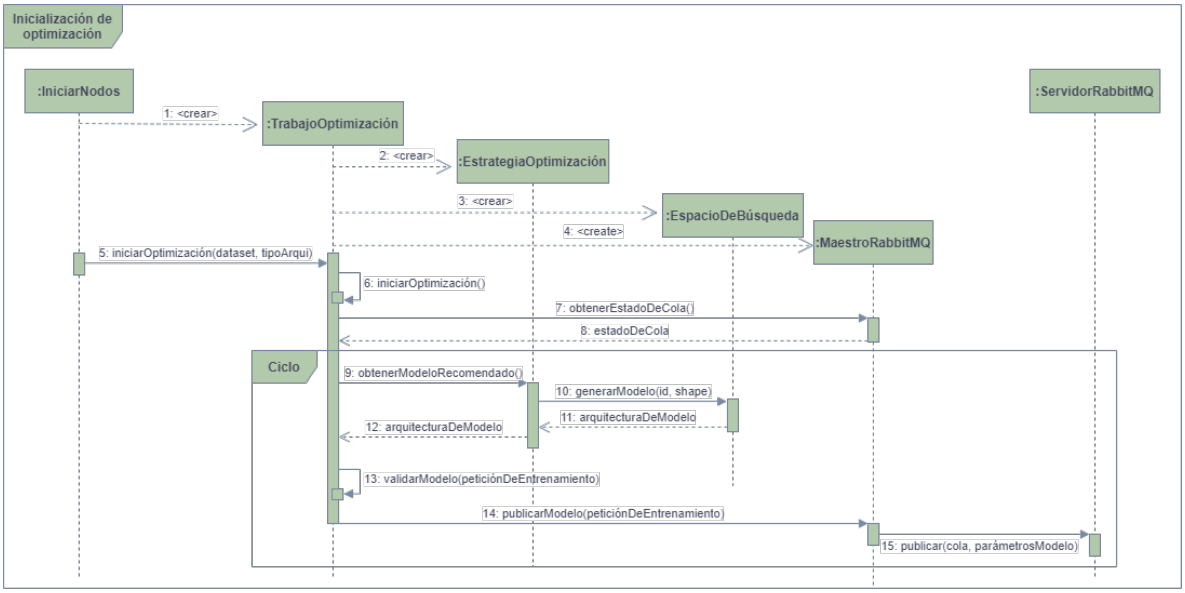

Figura 4.4: Diagrama de secuencia de inicialización de optimización

4.1.3. Proceso del nodo trabajador

Todos los nodos trabajadores manejan el mismo proceso, este proceso consiste en interpretar y ejecutar las peticiones de entrenamiento que llegan al *Broker*. En el nodo maestro recae la mayor parte de la estrategia de estimación de rendimiento. Su función consiste en interpretar las peticiones del *Broker*, construir los modelos, entrenarlos y retornar por medio del *Broker* la información del rendimiento.

El proceso del trabajador consiste en conectarse a la cola de peticiones de modelos por entrenar del *Broker*, cuando se recibe un modelo se ejecutan los siguientes pasos:

- Interpretar mensaje: Recibe el mensaje por parte del *Broker* en un formato JSON, el trabajador interpreta el mensaje y construye y entrena el modelo a partir de los hiperparámetros especificados.

- Validar espacio de búsqueda: Válida la compatibilidad de los diccionarios para la construcción del modelo mediante un valor *Hash*. Si el trabajador fue diseñado para soportar otro espacio de búsqueda que el especificado en el mensaje, rechaza el mensaje.

- Cargar el *Dataset*: El proceso se encarga de buscar y cargar el *Dataset* para entrenar el modelo en memoria. Si no se encuentra el *Dataset*, intenta descargarlo de la fuente. Una vez cargado se aplica el proceso de normalización.

- Construir modelo: Construye un modelo de aprendizaje profundo a partir de las especificaciones de la petición de entrenamiento.

- Entrenar modelo: Se empieza el entrenamiento del modelo creado con el *Dataset* y los hiperparámetros que se especifican en la petición.

- Enviar resultados: Crea un mensaje en formato JSON con las medidas de rendimiento del entrenamiento y lo manda al *Broker* a la cola de resultados.

La Figura 4.5 muestra el diagrama de secuencia utilizado para la implementación del nodo trabajador.

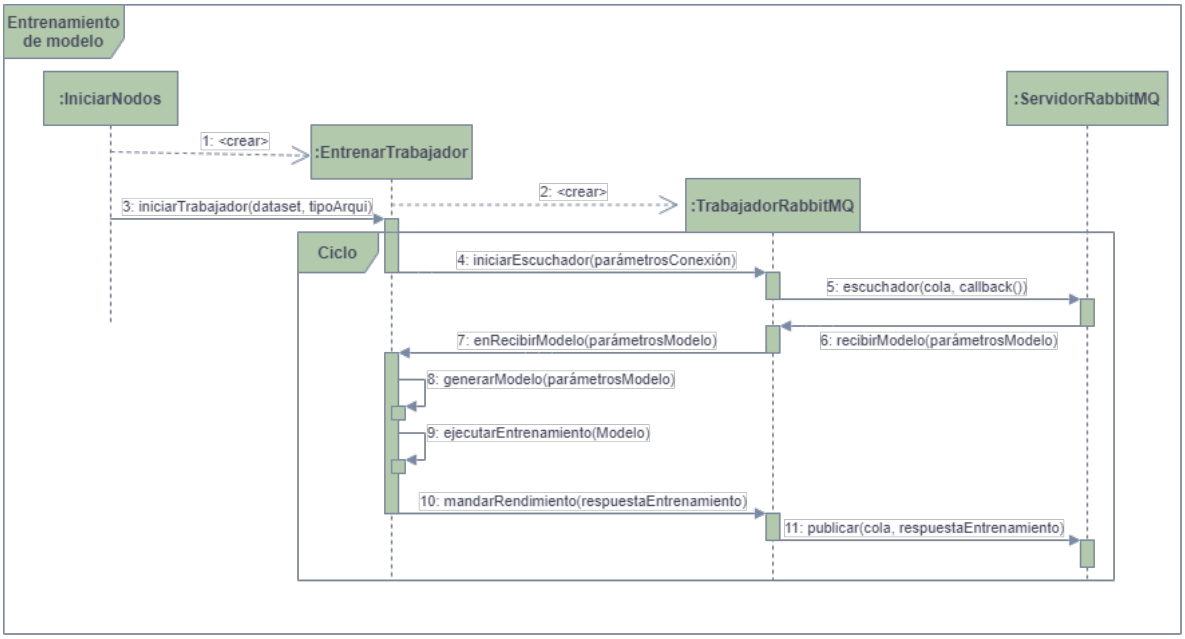

Figura 4.5: Diagrama de secuencia de inicialización de trabajadores para entrenamiento

4.2. Carga y preprocesamiento de datos

Los conjuntos de datos que se implementan en el entrenamiento de un modelo de aprendizaje máquina tienen un gran impacto sobre los resultados que se obtienen al momento del entrenamiento. Debido a lo anterior, se busca que los datos estén lo mejor adaptados posibles, para esto se utiliza el preprocesamiento. El preprocesamiento de la información consiste en la normalización de los datos, y en *data augmentation* para datos de tipo imagen. Mediante la implementación de las técnicas antes mencionadas, se busca mejorar algunos factores de los entrenamientos, tales como: Velocidad de convergencia, capacidad de generalización y reducción de sobreajuste.

4.2.1. Normalización

Las redes neuronales suelen tener convergencia más rápido cuando el promedio de los valores del conjunto de datos se acerca a 0 (LeCun et al., 2012). Debido a lo anterior a toda la información del conjunto de datos se le aplica un reescalado para transformar los datos a un rango entre 0 y 1.

Para cuestiones de imágenes, debido a que las imágenes están formadas por 8 bits por cada canal de color, estos tienen un valor desde 0 hasta 255, por lo que solo es necesario realizar una división entre 255 aplicadas a cada uno de los valores de la imagen. La Eq. 4.1 muestra la normalización implementada para imágenes.

$$N_{(x,y,z)} = \frac{I_{(x,y,z)}}{255} \tag{4.1}$$

Donde I es la imagen original, N es el resultado de la normalización, x es la fila de la imagen, y es la columna de la imagen, y z es el canal de color de la imagen.

Para los conjuntos de datos que se manejen por vectores, a estos se les realiza un proceso de mapeo de datos, donde se especifica en un documento los rangos de valores mínimo y máximo que contiene cada columna. Además, a los valores de tipo cadena se les hace una transformación a números, cada número representa una cadena distinta y se emparejan en forma de tuplas. Posteriormente, los valores son reescalados a un rango entre 0 y 1. Esta

información es retornada al usuario como retroalimentación del proceso de transformación. La Eq. 4.2 muestra la normalización implementada para vectores.

$$N_x = \frac{(V_x - min(V))}{(max(V) - min(V))} \tag{4.2}$$

Donde V es el vector original de características, N es el valor resultante de la normalización, y x es la posición del elemento en el vector.

4.2.2. Aumento de datos

El proceso de aumento de datos (*Data Augmentation* en Inglés) es un método utilizado para reducir el sobreajuste de los modelos que consiste en la generación de nuevas muestras a partir del *Dataset* de entrenamiento original.

Este procedimiento solo es implementado, en este proyecto, para el módulo de clasificación de imágenes. Donde se les aplica un conjunto de pequeñas transformaciones a copias de la muestra original, de esta manera se busca duplicar el tamaño del conjunto de datos. Las transformaciones realizadas consisten en: espejo horizontal, brillo y contraste, alteraciones de tono y saturación a imágenes a color. Este procedimiento de aumento de datos solo se aplica durante el entrenamiento profundo, debido principalmente a que los modelos rara vez se sobre ajustan durante pocas fases de entrenamiento.

4.3. Diseño de espacio de búsqueda

Un espacio de búsqueda se define mediante el tipo de problema a resolver y se selecciona por medio de un conjunto específico de características que pueda resolver ese problema específico. En este trabajo se implementa el *Cell-based Structure* para la construcción del espacio de búsqueda que divide la definición del espacio de búsqueda en componentes de extracción de características y componentes para clasificación (He et al., 2021). La Figura 4.6 muestra las diferentes combinaciones de capas que pueden implementarse dependiendo del problema a solucionar.

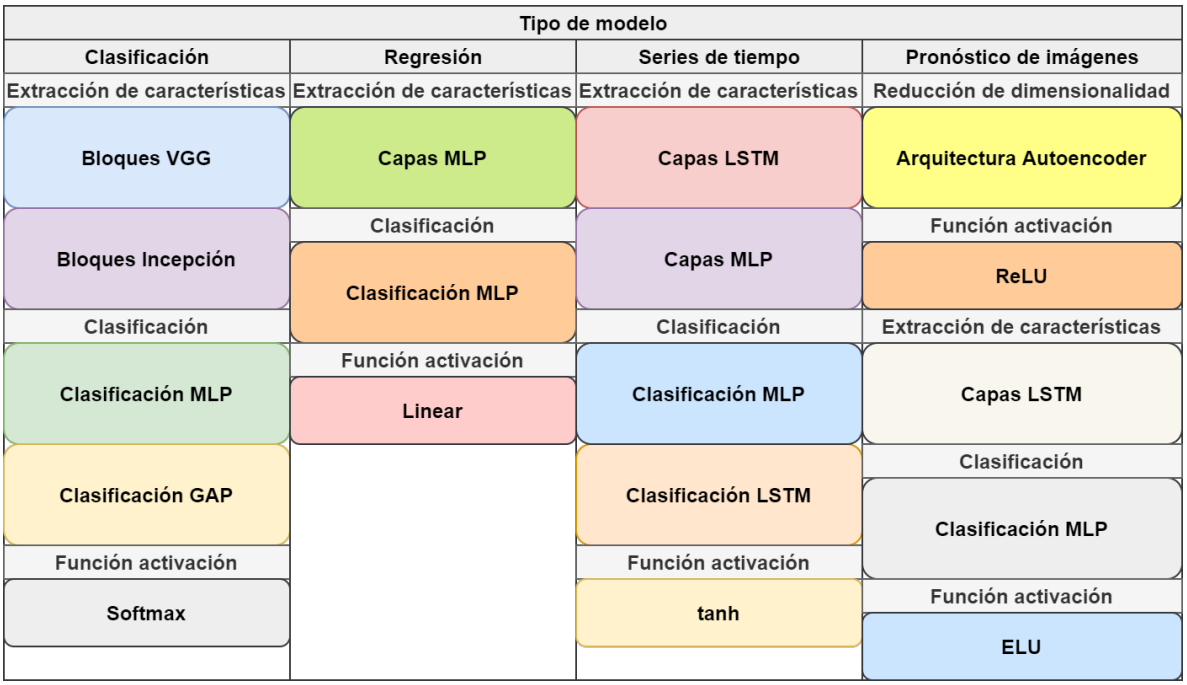

Figura 4.6: Espacio de búsqueda por tipo de implementación

Para definir un buen espacio de búsqueda es necesario crear una buena relación entre la granularidad y la eficiencia de búsqueda. Cada una presenta ventajas y desventajas. Si se busca centrarse en la alta granularidad, este permite el ajuste fino de cada hiperparámetros, pero hace que el espacio de búsqueda sea más amplio y en consecuencia sea más difícil de explorar. Por otro lado, otra solución consiste en incluir más hiperparámetros con menos opciones en cada uno, lo cual favorece la variedad de configuraciones sobre la granularidad. En este proyecto se optó por incluir más hiperparámetros con menos opciones.

La construcción de arquitecturas de aprendizaje profundo mediante el apilamiento de bloques y/o capas implica el riesgo de generar modelos con cuellos de botella. Para combatir lo anterior, se definen un conjunto de restricciones para la construcción, las restricciones consisten en incluir valores base y multiplicadores en la construcción. Con lo anterior se puede tomar un índice de capas y el número de neuronas o filtros de la capa anterior para delimitar el valor de las siguientes capas.

En este trabajo el espacio de búsqueda se representa mediante la utilización de objetos que definen una serie de constantes para la construcción de la arquitectura general del problema. Por otro lado, una vez que se define la arquitectura a implementar, las características de la

arquitectura son almacenadas en clases de datos, lo que permite tener objetos que puedan ser fácilmente transformados en datos de tipo JSON para ser mandados fácilmente por medio del *broker* RabbitMQ. Lo que se busca es definir una serie de rangos constantes que pueda tomar la estrategia de búsqueda para definir las características de la arquitectura del modelo y los parámetros que lo definen.

4.3.1. Extracción de características

En este apartado se especificarán cada una de las combinaciones del bloque de extracción de características para la construcción de modelos. La selección del estilo de extracción de características es por parte del algoritmo de optimización.

4.3.1.1. Bloques VGG

El primer módulo de capas es la arquitectura convolucional VGG, durante el proceso de generación de modelos, pueden generarse uno o más bloques, donde cada bloque está constituido por las siguientes capas en ese orden:

- Una o más capas convolucionales
- Capa *Batch Normalization*
- Capa *Max Pooling*
- Capa *Dropout*

Cada una de las capas internas tienen un conjunto de hiperparámetros que deben ser optimizados por algoritmo TPE. Para las capas convolucionales se definen la cantidad de filtros y su tamaño, y para las capas *dropout* se definen el porcentaje de neuronas a desactivar. La Tabla 4.1 muestra el conjunto de hiperparámetros utilizados en esta arquitectura.

Tabla 4.1: Conjunto de hiperparámetros para bloques VGG

Hiperparámetro	**Valor**
Rango de bloques VGG	$[1, 4]$
Rango de capas de Convolución por bloque	$[1, 4]$
Valor base de filtros de Convolución	32
Rango multiplicador de filtros	$[1, 2]$
Tamaño de filtros de Convolución	(3x3, 5x5)
Rango de valores de Dropout	$[0, 0.5]$

4.3.1.2. Módulos *Inception*

Este módulo está basado en la arquitectura *GoogleLeNet/Inception V1* para la extracción de características. Esta arquitectura *Inception* contiene dos partes para la extracción de características. La primera parte está conformada por un conjunto de capas convolucionales y *pooling*, es llamada **Stem**. Esta primera parte busca simplificar y reducir el tamaño de la imagen antes de pasar a la segunda parte.

El tamaño de salida de la primera parte de este módulo depende del número de capas, tamaño de filtros de convolución y tamaños de parches de *pooling*. En la Tabla 4.2 se define el apartado del espacio de búsqueda.

Tabla 4.2: Conjunto de hiperparámetros para primera parte de módulo Stem

Hiperparámetro	**Valor**
Tamaño filtro de Convolución	(3x3, 5x5, 7x7)
Tamaño de parche de Pooling	(2x2, 3x3)

La segunda parte de esta arquitectura son los módulos ***Inception***, que sigue un patrón establecido. Este patrón consiste en definir el espacio de búsqueda con bloques y módulos, donde un bloque contiene uno o más módulos *Inception* y una capa de *Max Pooling*. Además, en la implementación de este proyecto se agrega una capa de *Dropout* después de una capa de *pooling* para reducir el sobreajuste. La Tabla 4.3 muestra el conjunto de hiperparámetros definidos para esta arquitectura.

Tabla 4.3: Conjunto de hiperparámetros para módulos *Inception*

Hiperparámetro	**Valor**
Rango de bloques *Inception*	$[1, 3]$
Rango de módulos *Inception* por bloque	$[1, 3]$
Valor base de filtros de Convolución	4
Multiplicador de filtros Conv 1x1	$[1, 16]$
Multiplicador de filtros Conv 1x1 (reducción pre-Conv3x3)	$[1, 12]$
Multiplicador de filtros Conv 3x3	$[1, 24]$
Multiplicador de filtros Conv 1x1 (reducción pre-Conv5x5)	$[1, 3]$
Multiplicador de filtros Conv 5x5	$[1, 8]$
Multiplicador de filtros Conv 1x1 (reducción post-Pooling)	$[1, 8]$

4.3.1.3. Capas MLP

El perceptrón multicapa (MLP) son las capas consideradas básicas pues solo implementan las funciones básicas de un modelo de red neuronal. Estas capas son capaces de extraer y aprender características. Usualmente son utilizadas para problemas no muy complejos o para clasificación. En este proyecto se utilizan un conjunto de capas MLP para solucionar problemas de regresión. La Tabla 4.4 muestra el conjunto de hiperparámetros utilizados para la aplicación de esta arquitectura.

Tabla 4.4: Conjunto de hiperparámetros para capas MLP

Hiperparámetro	**Valor**
Total de capas MLP	$[1, 10]$
Unidades de capas MLP	$[1, 32]$
Multiplicador base de unidades	16
Valores de Dropout	(0, 0.1, 0.2, 0.3, 0.5)

4.3.1.4. Capas LSTM

Las capas LSTM son especialmente utilizadas para la extracción de características temporales, por lo que en este proyecto son utilizadas para el apartado de series de tiempo. Su implementación es realizada especificando una capa LSTM como entrada, y un conjunto de capas ocultas LSTM. Cabe destacar que para implementar una capa LSTM como entrada es necesario definir el parámetro "*return_sequences*" como verdadero. La Tabla 4.5 muestra el conjunto de hiperparámetros utilizados para la aplicación de esta arquitectura.

Tabla 4.5: Conjunto de hiperparámetros para capas LSTM

Hiperparámetro	Valor
Total de capas LSTM	$[1, 6]$
Unidades de capas LSTM	$[1, 32]$
Multiplicador base de unidades	16

4.3.1.5. Capas de Autocodificador

Los autocodificadores son una arquitectura utilizada para eliminación de ruido en imágenes, identificación de anomalías o reducción de dimensionalidad, siendo esta última de gran importancia para este proyecto en el pronóstico de imágenes. Su implementación se basa en la construcción de un codificador y un decodificador. Donde ambas partes contienen características similares, pero a la inversa, es decir uno reduce las características y el otro las amplía.

Un bloque de capas de un autocodificador se basa en la utilización de dos capas, una de convolución para la manipulación de características y una capa de *MaxPooling* para la reducción de dimensiones. Siguiendo estás características los parámetros, la Tabla 4.6 muestra las opciones para generar un autocodificador.

Tabla 4.6: Conjunto de hiperparámetros para construcción de autocodificadores

Hiperparámetro	Valor
Total bloques de codificación	$[2, 6]$
Unidades de capas de convolución	$[2, 16]$
Kernel de convolución	(3,5,7)
Reducción x de *Max Pooling*	$[1, 16]$
Reducción y de *Max Pooling*	$[1, 16]$

4.3.2. Capas de clasificación

Una vez que en la arquitectura se pasa por la extracción de características de las muestras, lo siguiente es pasar esta información a un clasificador para determinar su categoría o valor de error, dependiendo del tipo de entrenamiento que se realice. Para este apartado se puede optar por utilizar cualquiera de los dos tipos de clasificador, el primero consiste en una red neuronal sencilla y el segundo es una omisión de las capas de neuronas mediante la utilización

del *Global Average Pooling*.

4.3.2.1. Clasificación MLP

Este es un método de clasificación tradicional. Se compone de múltiples capas ocultas totalmente conectadas seguidas de capas de *Dropout* y una capa de salida totalmente conectada asociada a una función de activación, esta función varía dependiendo del espacio de búsqueda que se está ejecutando. La Tabla 4.7 muestra los hiperparámetros definidos en el espacio de búsqueda para la implementación de esta arquitectura.

Tabla 4.7: Conjunto de hiperparámetros para clasificación MLP

Hiperparámetro	**Valor**
Total de capas MLP	$[1, 10]$
Unidades de capas MLP	$[1, 32]$
Multiplicador base de unidades	16
Valores de Dropout	(0, 0.1, 0.2, 0.3, 0.5)

Cabe destacar que la generación de cantidad de neuronas y tasa de desactivación se hace de manera independiente en cada capa. En otras palabras, se generan de manera aleatoria por cada capa cada uno de los parámetros en el rango definido. La última capa del clasificador consta del número de neuronas correspondiente al número de clases o valores de salida del conjunto de datos para el cual se está realizando la optimización.

4.3.2.2. Clasificación GAP (Checar comentario)

El clasificador *Global Average Pooling* (GAP) es un método utilizado para reemplazar las capas de clasificación MLP por una única capa de salida. En cuestiones de clasificación de imágenes, se obtiene el promedio de cada mapa de características, convirtiendo un volumen de $alto \times ancho \times canal$ en $1 \times 1 \times canales$, en otras palabras, un escalar por canal o mapa de características (Lin et al., 2013).

Esta capa solo es utilizada por el espacio de búsqueda de clasificación de imágenes y el de series de tiempo. Donde las capas de clasificación son substituidas por una única capa de salida con una función de activación. La función de activación GAP para clasificación de imágenes y tanh para series de tiempo.

Esta idea proviene del trabajo "*The All Convolutional Net*" donde se obtienen resultados competitivos, reduciendo la complejidad de la arquitectura del modelo al tener muchos menos parámetros y costo computacional debido a la eliminación de las capas MLP (Springenberg et al., 2014).

4.4. Estrategia de búsqueda

La estrategia de búsqueda se encarga de generar los modelos a partir de la definición de un espacio de búsqueda y se guía mediante la retroalimentación obtenida por la estrategia de estimación de rendimiento. Sin la implementación de un algoritmo apropiado que guíe la generación de los modelos de aprendizaje, se obtendrían un conjunto de modelos que serían guiados por la aleatoriedad, con lo cual las soluciones obtenidas con el proceso de optimización podrían no ser las mejores. Debido a lo anterior en este proyecto se utiliza una estrategia de búsqueda que consta de dos partes, la fase de exploración que implementa el algoritmo bayesiano TPE y la fase de entrenamiento completo, donde solo los mejores modelos definidos por el salón de la fama pueden pasar.

Mediante la implementación del algoritmo bayesiano TPE lo que se busca es la definición y generación de modelos que tengan mejores características conforme se vayan generando. La estrategia de búsqueda se encarga de definir la arquitectura de los modelos y de los hiperparámetros de entrenamiento que llevaran a cabo los trabajadores. A su vez, el algoritmo bayesiano TPE genera las predicciones con base en la experiencia previa, sin embargo, este mismo algoritmo no puede ser implementado sin la experiencia previa, por lo que se implementa un algoritmo de búsqueda aleatoria para la inicialización de los primeros entrenamientos.

Con el algoritmo bayesiano las soluciones generadas pueden dividirse en un grupo malo y un grupo bueno, donde el propio algoritmo analiza cuáles son las mejores y peores características de cada solución. Lo anterior junto con una probabilidad de mejora, permite al algoritmo generar mejores soluciones conforme se obtiene más información. De esta manera, la implementación del algoritmo bayesiano puede proporcionar mejores soluciones en un menor tiempo.

Durante la fase de exploración se generan una gran cantidad de modelos diferentes, estos modelos en una primera instancia son generados por la búsqueda aleatoria y posteriormente por TPE. Para comenzar la optimización, se deben de especificar el número de modelos de búsqueda aleatoria y la cantidad total de modelos por explorar. En este trabajo se implementa un máximo de 30 % del total de modelos, para ser generados por el algoritmo de búsqueda aleatoria, y posteriormente por el TPE para el proceso de optimización.

4.5. Estrategia de estimación de rendimiento

El módulo de estrategia de estimación de rendimiento se encarga de entrenar y evaluar los modelos de aprendizaje profundo que se están implementando. En este proyecto esta estrategia es manejada por los nodos trabajadores, además, está diseñada para maximizar la exploración y explotación de la búsqueda.

Como se ha mencionado anteriormente el entrenamiento está divido en dos fases. Durante la fase de exploración se busca evaluar una gran cantidad de modelos en poco tiempo, esto se logra mediante la utilización de entrenamiento parcial, que consiste en entrenar por pocas épocas un modelo de aprendizaje profundo. Además, se implementa un *Early Stopping Patience* que para el entrenamiento cuando el modelo, durante el entrenamiento, no muestra una mejora.

Cuando se pasa a la fase de entrenamiento completo o profundo se utiliza una mucho mayor cantidad de épocas, al menos 100 épocas. Durante esta fase lo que se busca es extraer el mayor rendimiento de los modelos seleccionador por el salón de la fama durante la fase de exploración. A su vez, la implementación del *Early Stopping Patience* es diferente, pues implementa un mayor número de épocas antes de que pare el entrenamiento.

4.6. Pronóstico de imágenes con autocodificadores y redes neuronales

El autocodificador es una arquitectura constituida por capas de convolución y *pooling* que reducen y reconstruyen las dimensiones de las muestras. Esta arquitectura está constituida

por dos partes principales. La primera parte es el codificador que se encarga de reducir las dimensiones de las muestras hasta un punto especificador. Por otro lado, la segunda parte es el decodificador que se encarga de reconstruir las muestras codificadas a su forma original.

Aunque sus aplicaciones principales consisten en la codificación de la información y la eliminación de ruido en imágenes. En este proyecto se implementa esta arquitectura para reducir la dimensionalidad de las muestras y, reducir así, el tiempo computacional requerido para la implementación de pronóstico de imágenes.

La aplicación de múltiples arquitecturas de autocodificadores se obtiene mediante un procedimiento un conjunto de combinaciones de capas de convolución y *pooling*, tal que, se obtienen múltiples modelos de autocodificadores que reducen las muestras a diferentes dimensiones. Posteriormente, se realiza un proceso de entrenamiento de series de tiempo, y se obtiene el siguiente paso en el tiempo. La Figura 4.7 muestra el proceso principal de esta aplicación.

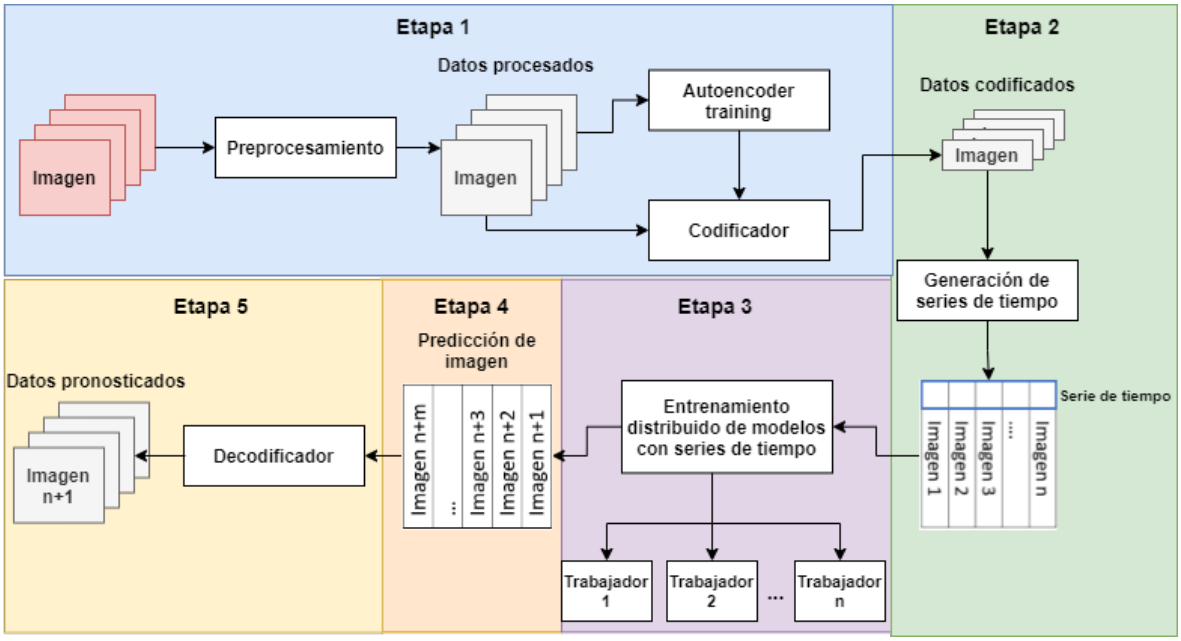

Figura 4.7: Proceso de pronóstico de imágenes de series de tiempo

El proceso principal de esta implementación puede dividirse en cinco etapas. La primera etapa consiste en la carga de la información, el procesamiento y el entrenamiento del autocodificador. Una vez realizado el entrenamiento del autocodificador, se implementa la parte del codificador para obtener un conjunto de muestras codificadas.

Durante la etapa 2 se utiliza el conjunto de muestras codificadas de la etapa 1 y se generan un conjunto de series de tiempo. Para la generación de las series de tiempo, cada característica de las muestras codificadas representa una serie de tiempo independiente. De tal manera que, al juntar cada una de las características de las muestras se obtiene un conjunto de series de tiempo. Cada una de estas series de tiempo será manejada por un modelo independiente. La Figura 4.8 muestra la transformación de las muestras a un conjunto de series de tiempo.

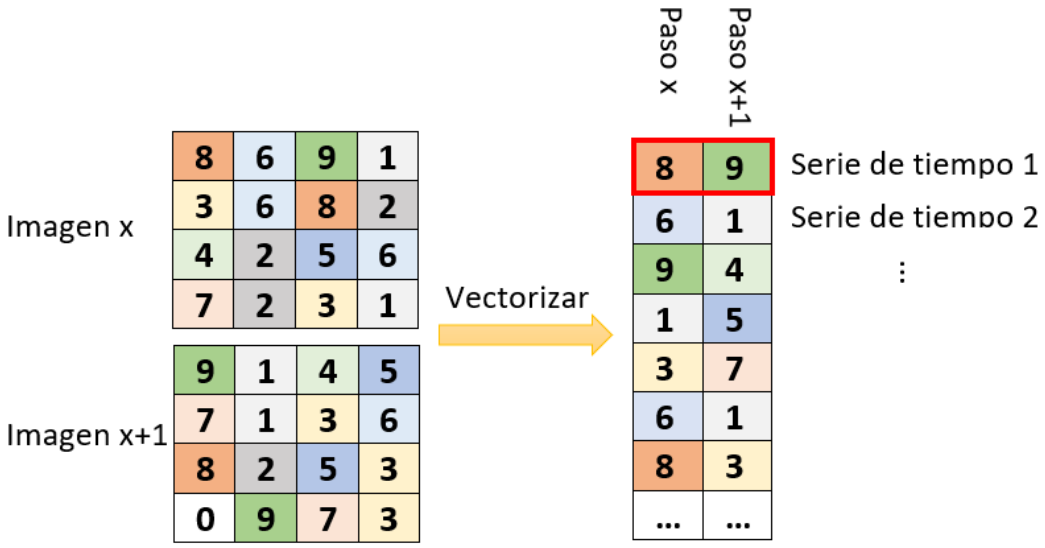

Figura 4.8: Generación de múltiples series de tiempo

La etapa 3 utiliza el conjunto de series de tiempo generado y lo distribuye para ser entrenado por diferentes trabajadores. Una vez realizados los entrenamientos se retornan un conjunto de miles de modelos entrenados para realizar un pronóstico. En la etapa 4 se implementan cada uno de los modelos generados para realizar un pronóstico de los siguientes n pasos de tiempo, de tal manera que cada pronóstico sea una parte de una muestra codificada.

Finalmente, la etapa 5 maneja este conjunto de muestras codificadas pronosticadas para pasarlas por el decodificador del autocodificador y obtener así un conjunto de imágenes pronosticadas que correspondan a los siguientes n pasos del tiempo.

4.7. Interfaz gráfica

Para explotar más la implementación de esta herramienta, y que a su vez sea capaz de llegar a más usuarios, se realizó una implementación de una interfaz de usuario que ejecuta en paralelo la herramienta desarrollada. La aplicación principal del optimizador se mantiene en su lenguaje de origen, es decir, Python, mientras que la aplicación gráfica se encuentra desarrollada en el lenguaje C#.

Mediante la implementación de *sockets* se hace posible la comunicación entre estos lenguajes. Más concretamente, se utiliza la librería de Python "Flask-socketIO" que permite la implementación de peticiones HTTP y *sockets* mediante Python. Por su parte, en C# se utiliza una conexión a *sockets* mediante la librería "SocketIoClientDotNet" para el envío de mensajes entre el *front-end* y el *back-end* en tiempo real. La Figura 4.9 muestra la comunicación que se genera entre ambas partes.

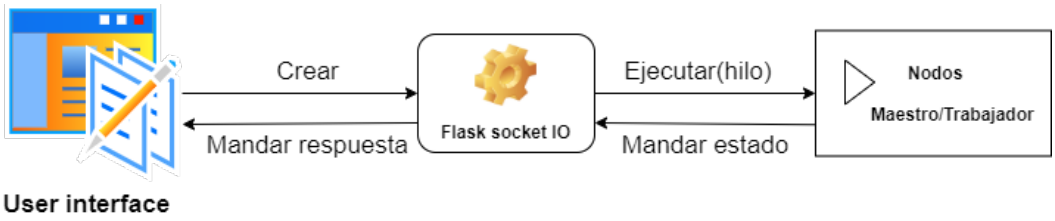

Figura 4.9: Comunicación entre la interfaz de usuario y el *back-end* de optimización

Capítulo 5

Análisis de resultados

En este capítulo se presenta el conjunto de resultados obtenidos en la fase de experimentación de cada uno de los espacios de búsqueda implementados. A su vez, se describe brevemente el hardware utilizado para la experimentación e implementación del proyecto. Para demostrar la efectividad de la metodología propuesta, se han realizado múltiples experimentos con diferentes conjuntos de datos (*Dataset* en Inglés). Estos *Datasets* presentan diferentes características y tipos de valores, tales como: Número de entradas, número de salidas, dimensiones, tipos de datos y tamaño del conjunto de datos.

5.1. Hardware

Durante la experimentación se ha utilizado una computadora de escritorio con las siguientes características:

- CPU: Intel Core i7-8700 (6 núcleos, 12 Hilos a 4.6 GHz)
- RAM: 16 GB DDR4 2666 MHz
- GPU: 2x Nvidia RTX 2080 Ti 11GB
- SO: Ubuntu 20.04 LST
- Almacenamiento: SSD 240 GB

Este equipo de cómputo fue diseñado para soportar múltiples procesos de manera simultánea. Entre los procesos se encuentra el entrenamiento de múltiples modelos de apren-

dizaje máquina de al mismo tiempo. Esto se logra mediante la implementación de dos GPU instaladas en el equipo, además la capacidad del procesador permite manejar y organizar múltiples tareas para manipular las dos GPU.

5.2. Interfaz gráfica

Una de las primeras integraciones de este proyecto consiste en la implementación de la interfaz gráfica para la manipulación de parámetros que guiarán al proceso de optimización. La interfaz gráfica es capaz de manejar toda la retroalimentación otorgada por el optimizador de modelos y desplegarla en la interfaz. La interfaz gráfica se divide en 4 secciones, donde cada sección permite al usuario manipular las características del optimizador en menor complejidad. En la primera sección se despliega la información de los autores del proyecto, la escuela y el programa afiliado al proyecto. La figura 5.1 muestra la primera sección de la interfaz.

Figura 5.1: Primer sección de la interfaz gráfica

La segunda sección presenta los archivos de configuración del ambiente del sistema de optimización. Se le solicita al usuario ubicar y seleccionar archivos especificos para el funcionamiento completo del programa. Se le solicita al usuario la dirección del ejecutable de Python, el Script de Flask para el soporte de *sockets*, el Script de "clear_queues" para limpiar

las colas de RabbitMQ, la dirección de almacenamiento de la imagen del mejor modelo actual para el maestro y la dirección de almacenamiento de la imagen del modelo actual para el trabajador. La Figura 5.2 muestra la pantalla de la segunda sección de la interfaz.

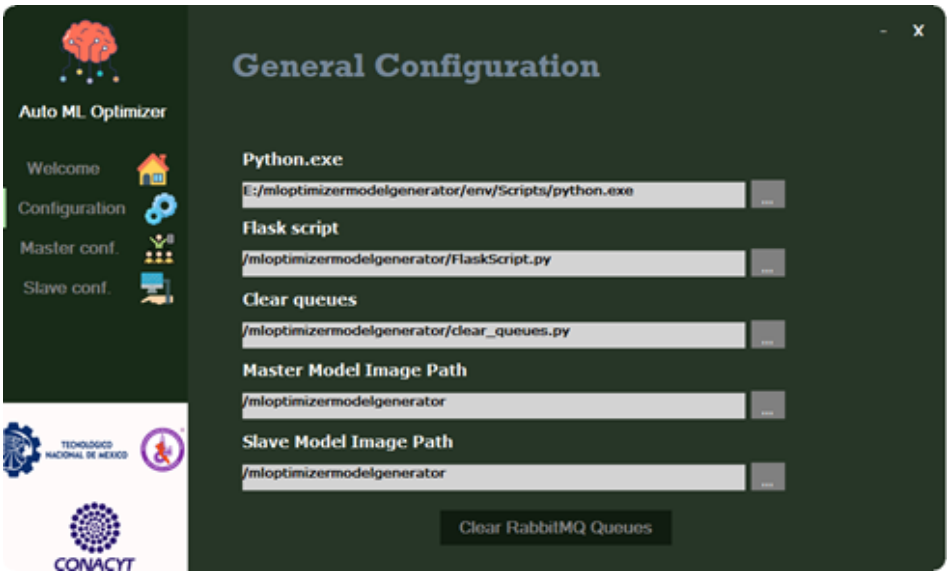

Figura 5.2: Segunda pantalla de la interfaz gráfica

La tercera sección de la interfaz es la configuración del maestro. Dentro de esta sección se pueden controlar las configuraciones del maestro y ejecutar un proceso de optimización. Cada sección de parámetros afecta a un proceso diferente dentro del proceso de optimización. La Figura 5.3 muestra la pantalla de la tercera sección de la interfaz. La Figura 5.4 muestra los parámetros de conexión a RabbitMQ. La Figura 5.5 muestra los parámetros del conjunto de datos a utilizar. La Figura 5.6 muestra el conjunto de parámetros para el AutoML. Por último, la Figura 5.7 muestra los parámetros para el entrenamiento de los modelos.

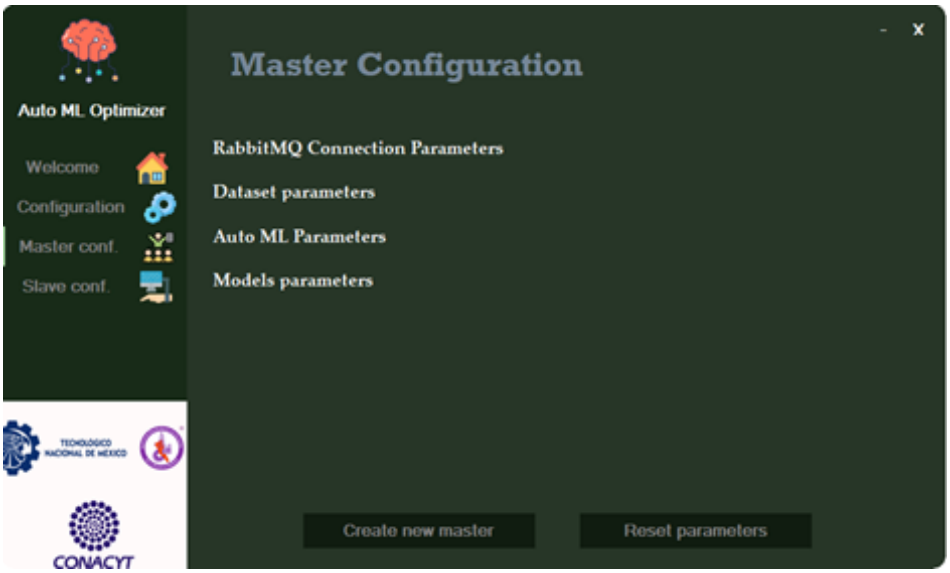

Figura 5.3: Tercera pantalla de la interfaz gráfica, configuración del nodo maestro

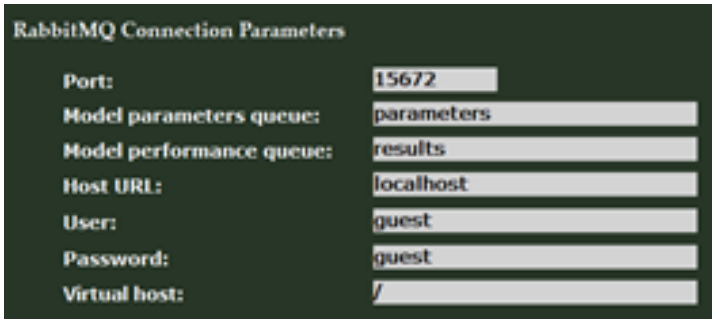

Figura 5.4: Configuración del nodo maestro, conexión de RabbitMQ

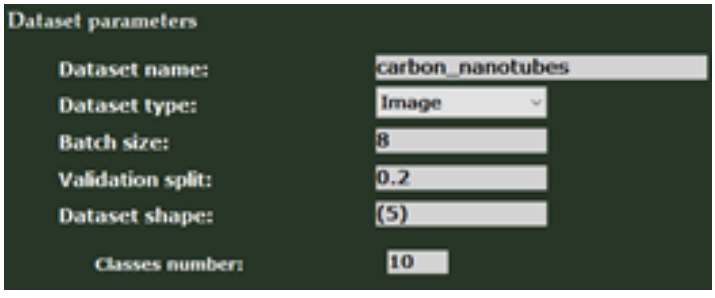

Figura 5.5: Configuración del nodo maestro, conjunto de datos

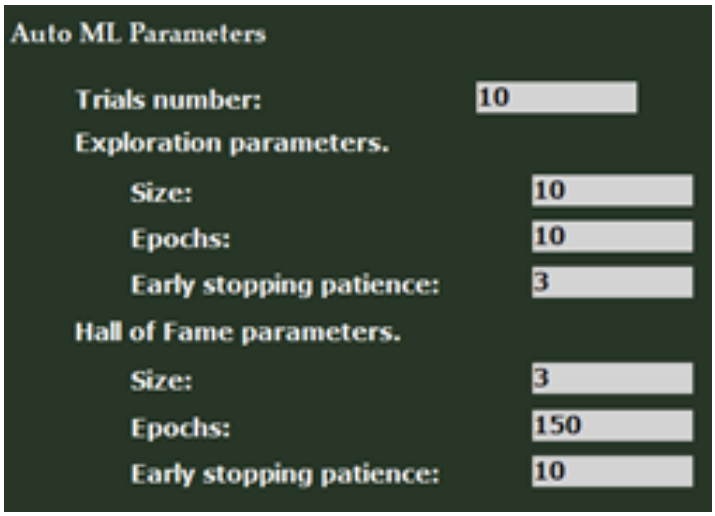

Figura 5.6: Configuración del nodo maestro, AutoML

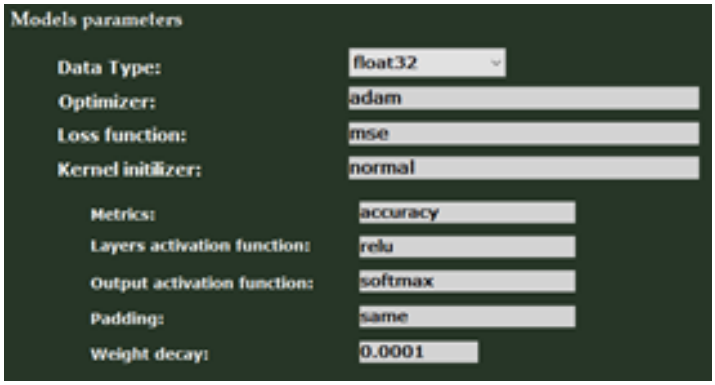

Figura 5.7: Configuración del nodo maestro, modelos

Una vez terminada la especificación de parámetros se ejecuta y levanta un nodo maestro. De manera interna se levanta un servicio de *sockets* y una instancia de un nodo maestro para iniciar el proceso de optimización. Mientras que por parte de la interfaz se especifican el estatus del nodo, la progresión de los modelos y la mejor solución actual mediante una imagen. La Figura 5.8 muestra la implementación del nodo maestro.

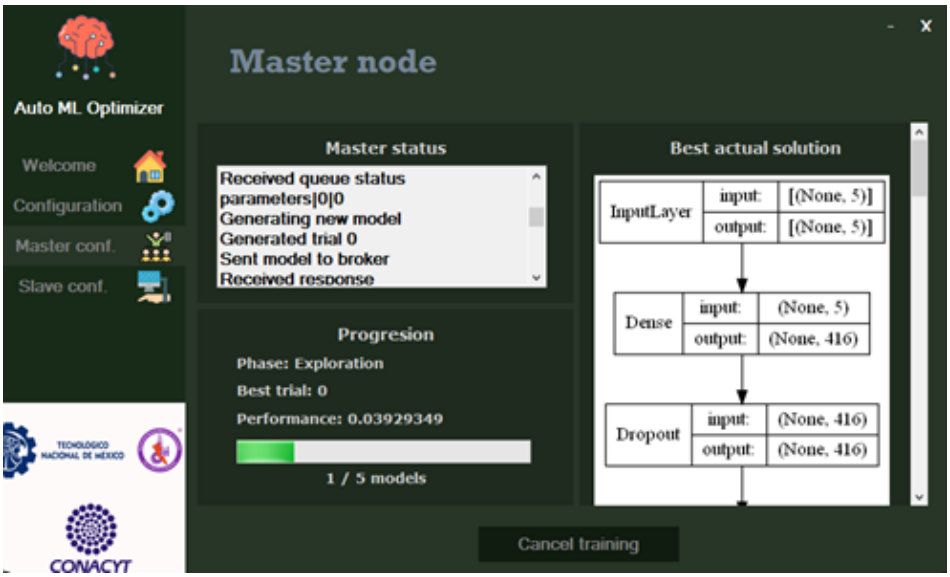

Figura 5.8: Implementación del nodo maestro en interfaz gráfica

La última sección de la interfaz es la configuración del trabajador. Al igual que la sección pasada, se especifican una serie de parámetros para definir un nodo trabajador. A su vez, es posible la creación de un nodo trabajador para la ejecución del proceso de optimización. Los parámetros que se le solicitan al usuario son la conexión a RabbitMQ, la especificación del *Dataset* y se utiliza la GPU del equipo o no. La Figura 5.9 muestra la pantalla de la cuarta sección de la interfaz. La especificación de parámetros de RabbitMQ y *Dataset* es la misma que la del nodo maestro, ver Figura 5.4 y Figura 5.5.

Figura 5.9: Cuarta pantalla de la interfaz gráfica, configuración del nodo trabajador

Una vez terminada la especificación de parámetros se ejecuta y levanta un nodo trabajador. De manera interna se levanta un servicio de *sockets* y una instancia de un nodo trabajador, para iniciar el proceso de entrenamiento de modelos. Mientras que por parte de la interfaz se especifican el estatus del nodo, la progresión de los entrenamientos de los modelos, junto con el valor de rendimiento, y el modelo actual mediante una imagen. La Figura 5.10 muestra la implementación del nodo trabajador.

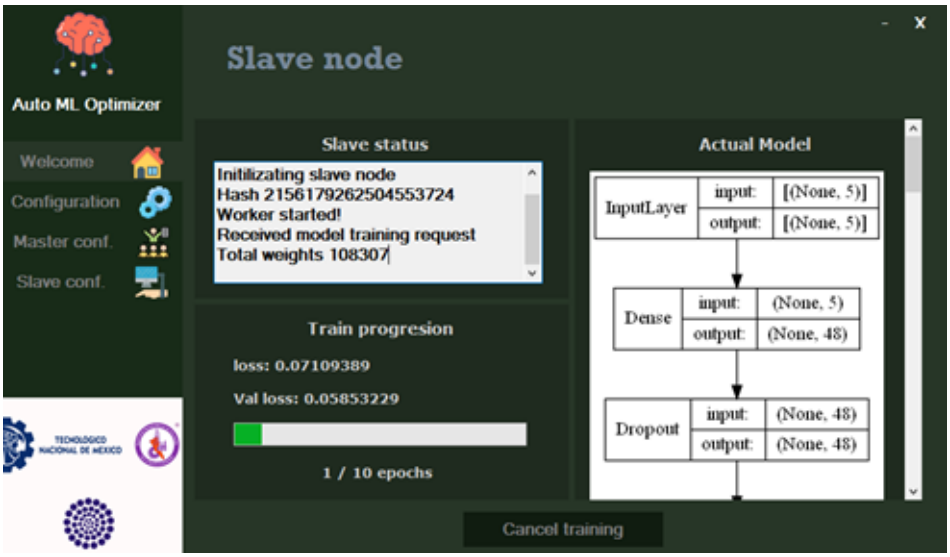

Figura 5.10: Implementación del nodo trabajador en interfaz gráfica

5.3. Conjuntos de datos

Para probar la efectividad de la metodología propuesta se utilizaron diferentes conjuntos de datos de múltiples fuentes. Las fuentes de datos principales son Tensorflow *Datasets*, para descargar conjuntos de datos de imágenes, la página web "Kaggle", repositorio de conjuntos de datos diversos, y la página "*UCI Machine Learning Repository*". Además, se hicieron pruebas con conjuntos de datos propios sobre la velocidad del viento. La Tabla 5.1 muestra la información de los conjuntos de datos utilizados.

Tabla 5.1: Descripción de conjuntos de datos utilizados

Origen	Tipo	Nombre	Descripción
TF-Datasets	Clasificación de imágenes	MNIST	60,000 imágenes de 10 dígitos escritos a mano.
TF-Datasets	Clasificación de imágenes	Fashion MNIST	70,000 imágenes de artículos de moda en 10 categorías
TF-Datasets	Clasificación de imágenes	CIFAR-10	60,000 imágenes a color de 10 categorías
TF-Datasets	Clasificación de imágenes	CIFAR-100	60,000 imágenes a color de 100 categorías
TF-Datasets	Clasificación de imágenes	Horses or Humans	500 imágenes de caballo y 527 de humanos
Kaggle	Regresión	Abalone Dataset	4177 muestras de 8 atributos sobre la edad del abulón
Kaggle	Regresión	Graduate admission 2	500 muestras de 7 atributos y una salida sobre el grado de admisión de estudiantes
Kaggle	Regresión	CCPD	9568 muestras de 4 atributos y una salida sobre la generación de Planta de poder de ciclo combinado
Kaggle	Regresión	Concrete Data	1030 muestras de 8 atributos y una salida sobre la creación de concreto
Kaggle	Regresión	Carbon nanotubes	10721 muestras de 5 atributos y 3 salidas sobre coordenadas atómicas de nanotubos de carbono
UCI repository	Regresión	Electrical Grid	10000 muestras de 12 atributos y 2 salidas sobre la estabilidad de un centro de producción eléctrica
UCI repository	Regresión	ENB2012	768 muestras de 8 atributos y 2 salidas sobre la eficiencia energética
Kaggle	Regresión	SGEMM	241600 muestras de 18 atributos y 4 salidas sobre el rendimiento del kernel de GPU
Kaggle	Series de tiempo	Daily min temperatures	3650 muestras de un atributo y una salida sobre la temperatura mínima diaria en Melbourne, Australia
Kaggle	Series de tiempo	Electric production	397 muestras de un atributo y una salida sobre la producción eléctrica diaria
Kaggle	Series de tiempo	Montly Beer production	476 muestras de un atributo y una salida sobre la producción mensual de cerveza en Australia
Propia	Series de tiempo	La palma	8812 muestras de la velocidad del viento tomada en La palma, Michoacán
Propia	Series de tiempo	Manzanillo	3760 muestras de la velocidad del viento tomada en Manzanillo, Michoacán
Propia	Series de tiempo	Corrales	7504 muestras de la velocidad del viento tomada en Corrales, Michoacán
Drought monitor	Pronóstico de imágenes	Drought time series	1079 muestras de imágenes de sequías tomadas semana a semana

5.4. Clasificación de imágenes

La experimentación para clasificación de imágenes fue realizada mediante la implementación de la interfaz gráfica. Se utilizaron 5 *Datasets* obtenidos de TensorFlow *Datasets* 2.0, los cuales son: CIFAR-10, CIFAR-100, MNIST, *Fashion*-MNIST y *Horses or Humans*. Cada uno de estos *Dataset* presentan sus propias características, dimensiones y clases. La Tabla 5.2 muestra los resultados de los diferentes experimentos de clasificación de imágenes.

Tabla 5.2: Experimentos de clasificación de imágenes.

#	Batch Size	Dataset	ID mejor	Conv	Clasif.	Params $x10^6$	Tiempo	% Exp.	% Final
1	128	CIFAR-10	96	VGG	MLP	1.97	01:52:25	81.49 %	88.49 %
2	512	CIFAR-10	59	VGG	MLP	3.95	01:12:14	80.02 %	87.01 %
3	128	Fashion-MNIST	92	VGG	MLP	0.38	01:05:17	92.90 %	93.74 %
4	512	Fashion-MNIST	93	VGG	MLP	1.03	00:34:27	92.44 %	93.33 %
5	8	Horses or Humans	89	Inception	GAP	0.29	00:58:51	96.88 %	91.41 %
6	16	Horses or Humans	41	Inception	GAP	0.23	00:35:08	95.31 %	92.58 %
7	128	MNIST	96	VGG	MLP	2.48	01:00:29	99.35 %	99.36 %
8	512	MNIST	2	VGG	MLP	0.38	00:49:32	99.23 %	99.27 %
9	128	CIFAR-100	36	VGG	GAP	0.91	01:25:31	50.52 %	59.71 %
10	512	CIFAR-100	58	VGG	MLP	0.89	00:44:22	42.52 %	52.74 %

Todos los experimentos fueron realizados con dos trabajadores, donde cada trabajador utilizaba exclusivamente una GPU NVIDIA RTX 2080 TI. Los experimentos fueron ejecutados con 100 modelos para la fase de exploración, entrenados por un total de 10 épocas o menos (dependiendo de la métrica *Early Stopping*). Durante la fase del entrenamiento completo, el HoF toma los mejores tres modelos que serán entrenados durante 300 épocas o menos.

Se realizaron dos experimentos por *Dataset*, el cambio entre cada experimento consiste en la ampliación del valor de *Batch size*. Este hiperparámetro indica la cantidad de muestras que se ingresan de manera simultánea en una iteración dentro de una época de entrenamiento, con lo cual su uso puede acelerar el entrenamiento. Como se puede observar en cada par de experimentos se reduce el tiempo de manera significativa. Para los experimentos #5 y #6 el *Batch Size* se ve reducido en gran medida, pues debido a las características del *Dataset*, la memoria de la GPU solo puede soportar pocas muestras a la vez por iteración. Sin embargo,

de igual manera el tiempo de implementación se reduce a medida que el *Batch Size* aumenta.

Otra de las cosas que se pueden notar en los experimentos, es que la implementación del *Dataset* seleccionado cambia la arquitectura que se selecciona mediante el proceso de optimización. En la mayoría de los casos se utiliza una arquitectura VGG + MLP, a excepción del experimento # 9 que implementa una arquitectura VGG + GAP. A su vez, los experimentos #5 y #6 utilizan una arquitectura *Inception* + GAP, dando a entender que el *Dataset Horses or Humans* funciona mejor con esta arquitectura.

5.5. Regresión

Para probar la efectividad de este espacio de búsqueda, se hizo la optimización de 8 *Datasets* diferentes. Cada uno de ellos tiene una cantidad de entradas y salidas diferentes. Sin embargo, el proceso de optimización es capaz de normalizar la información y usar un rango grande de características. Todos los experimentos de regresión implementan solo capas MLP para extracción de características y clasificación. En otras palabras el proceso de optimización manejará la cantidad de capas, unidades por capa y valor de *Dropout*.

Se utilizaron un total de 500 modelos para la fase de exploración y 10 para la fase de HoF con entrenamiento profundo. La Tabla 5.3 muestra los resultados obtenidos de los diferentes experimentos con los *Dataset* de regresión, cabe destacar que lo que diferencia a las arquitecturas es el número de capas MLP implementadas y los hiperparámetros.

Tabla 5.3: Experimentos de regresión vectorial

#	Batch Size	Dataset	ID mejor	Capas MLP	Params $x10^3$	Tiempo	MSE Exp.	MSE Final
1	128	Abalone	10	6	34.90	00:16:14	0.0061	0.0044
2	512	Abalone	13	2	8.43	00:13:42	0.0062	0.0045
3	128	Graduate admission	388	4	27.33	00:15:38	0.0139	0.0058
4	512	Graduate admission	397	3	11.17	00:12:49	0.0124	0.0073
5	128	CCPD	236	2	218.24	00:17:25	0.0030	0.0027
6	512	CCPD	11	3	56.50	00:11:19	0.0053	0.0029
7	128	Concrete	47	7	199.7	00:10:41	0.0159	0.0132
8	512	Concrete	5	7	430.48	00:12:12	0.0191	0.0133
9	128	Carbon nanotubes	357	1	3.60	00:15:18	0.0003	0.0001
10	512	Carbon nanotubes	313	1	3.17	00:14:57	0.0002	0.00008
11	128	Electrical Grid	414	4	437.51	00:16:19	0.0244	0.0139
12	512	Electrical Grid	10	5	69.52	00:15:33	0.0506	0.0144
13	128	SGEMM	28	1	8.21	01:20:55	0.0358	0.0053
14	512	SGEMM	261	5	232.31	00:38:02	0.0367	0.0122
15	128	ENB	26	3	43.78	00:15:51	0.0524	0.0051
16	512	ENB	48	3	137.7	00:13:25	0.0264	0.0052

Debido a la gran capacidad de cómputo del equipo utilizado para la experimentación, solo el experimento #13 paso de una hora de implementación. A pesar de la gran cantidad de modelos utilizados en la fase de exploración se puede notar que el modelo más apropiado para el *Dataset* puede ser obtenido a los pocos modelos de la implementación, en la parte de la implementación de búsqueda aleatoria.

Se ejecutaron dos experimentos por conjunto de datos. Cada par de experimentos implementa un *Batch size* distinto para corroborar la afirmación de la sección anterior. De manera similar, el tiempo de ejecución de los experimentos se ve reducido en la mayoría de los casos.

Durante la experimentación de regresión la información es vectorial, esto ocasiona que no sé presente una larga duración de optimización a pesar del amplio valor de la fase de exploración, pues presenta menos características. Por otra parte, el valor de error presentado es muy bajo a pesar de las arquitecturas simples implementadas. En los experimentos #9, #10 y #13 se aprecia que solo implementan una capa oculta y aun así puede manejar la información.

5.6. Series de tiempo

Para probar la efectividad del espacio de búsqueda de series de tiempo, se realizo el proceso de optimización con 3 *Datasets* diferentes. Todos los experimentos realizados utilizan 500 modelos para la fase de exploración y 10 modelos para el entrenamiento completo. Además, el proceso decide entre utilizar capas LSTM o MLP para extracción de características temporales, y capas MLP o solo una capa de salida (llamado GAP) para la etapa de clasificación. La Tabla 5.4 muestra los resultados obtenidos de los diferentes experimentos con los *Dataset* de series de tiempo.

Tabla 5.4: Experimentos de series de tiempo vectorial

#	Batch Size	Dataset	ID mejor	Extra.	Clasif.	Params $x10^6$	Tiempo	MSE Exp.	MSE Final
1	128	Min temperatures	383	MLP	GAP	0.16	00:10:34	0.0080	0.0073
2	512	Min temperatures	4	LSTM	MLP	1.18	00:38:27	0.0101	0.0072
3	128	Electric production	371	LSTM	GAP	0.02	00:18:43	0.0204	0.0048
4	512	Electric production	102	MLP	GAP	0.298	00:28:41	0.0175	0.0059
5	128	Beer production	28	MLP	MLP	0.34	00:09:41	0.0081	0.0036
6	512	Beer production	386	MLP	GAP	0.27	00:30:08	0.0101	0.0129
7	128	La palma	129	MLP	MLP	0.99	00:29:55	0.0231	0.0177
8	512	La palma	150	MLP	MLP	0.90	01:02:05	0.0234	0.0182
9	128	Manzanillo	27	MLP	MLP	0.70	00:10:10	0.0489	0.0494
10	512	Manzanillo	1	LSTM	GAP	0.42	00:11:34	0.0493	0.0492
11	128	Corrales	24	MLP	GAP	0.001	01:00:40	0.0024	0.0021
12	512	Corrales	14	MLP	MLP	0.24	01:05:33	0.022	0.0021

Se implementaron dos experimentos por conjunto de datos, donde a cada experimento se le cambió el *Batch size*. En el caso de series de tiempo, la implementación de un mayor valor fue contraproducente pues el tiempo aumento de manera significativa. Sin embargo, lo anterior puede deberse al tamaño de los propios *Dataset* pues no eran muy grandes.

Lo más importante a destacar es que se realizó apropiadamente un proceso de optimización para series de tiempo con el espacio de búsqueda seleccionado. Los valores de perdida obtenidos durante esta fase de experimentación fueron bastante bajos, por lo que resulta apro-

piado implementar el espacio de búsqueda diseñado para la resolución de problemas de series de tiempo.

Hasta este punto se han mostrado los resultados de 3 espacios de búsqueda distintos. Con base en los resultados obtenidos, es propio decir que la aplicación de múltiples espacios de búsqueda no afecta al proceso de optimización en la estrategia de búsqueda y la estrategia de estimación de rendimiento. Lo anterior es debido a que la aplicación del espacio de búsqueda es independiente de los otros dos procesos, por lo que solo se encarga de definir la arquitectura.

5.7. Pronóstico de series de tiempo de imágenes

Como se especifica en la metodología, en la Figura 4.7 el pronóstico de series de tiempo se realiza en varias etapas. De estas etapas, solo la creación del autocodificador puede realizarse con un proceso de optimización. Por lo tanto, para probar la efectividad de la metodología propuesta para el pronóstico de series de tiempo de imágenes, diferentes experimentos fueron realizados.

Durante la experimentación se implementaron dos esclavos para entrenar los autocodificadores generados por el espacio de búsqueda, y a su vez, entrenar cada una de las series de tiempo generadas por el proceso de codificación del pronóstico de imágenes. La figura 5.11 muestra la arquitectura general implementada por cada uno de estos modelos de series de tiempo. Cada modelo utiliza dos capas LSTM como extracción de características temporales, junto con dos capas de clasificación MLP para el proceso de interpretar esas características y obtener un resultado. Por último, el modelo cuanta con una capa de salida de un valor de salida, a su vez, se implementó la función de activación “ELU”, El modelo es compilado con la función de perdida Error Cuadrático Medio (MSE por sus siglas en Inglés) y la función de optimización “RmsProp”. Cabe destacar que se probaron múltiples arquitecturas y la anteriormente especificada, otorgo en promedio el mejor rendimiento.

Figura 5.11: Arquitectura de modelos de pronóstico de series de tiempo de imágenes

Se optó por implementar un modelo estático para cada uno de los modelos de series de tiempo. Lo anterior se debe a que cada proceso de optimización toma una cantidad considerable de tiempo, esto agregado a que son miles de modelos a entrenar, tardaría una grandísima cantidad de tiempo optimizar de manera apropiada cada uno de ellos.

Durante el entrenamiento de los modelos se implementó un *Batch size* de 64 muestras, por 150 épocas por cada modelo. Se implementó a su vez un *Early Stopping* con un valor de paciencia de 15 para parar el entrenamiento en caso de que no haya una mejora. La Tabla 5.5 muestra una tabla comparativa con los resultados de los experimentos realizados a dos pasos hacia el futuro.

Tabla 5.5: Experimentos de pronóstico de series de tiempo de imágenes

Dimensiones	**Total Modelos**	**Tiempo**	**Error autocod.**	**Precc. paso 1**	**Error paso 1**	**Precc. paso 2**	**Error paso 2**
(120,16,7)	13,440	0d-07h-14m-23s	0.0146	0.8263	0.1737	0.8265	0.1735
(60,40,10)	24,000	0d-11h-56m-39s	0.0126	0.8342	0.1658	0.8349	0.1651
(96,32,11)	33,792	0d-16h-29m-32s	0.0120	0.8929	0.1071	0.8931	0.1069
(80,128,5)	51,200	1d-00h-32m-20s	0.0056	0.9067	0.0933	0.9057	0.0943

Donde las dimensiones representan la forma de codificación de las muestras y el total de modelos implementados por el experimento. La columna del tiempo representa el tiempo total tomado desde que inicia el proceso de entrenamiento de múltiples series de tiempo. El error de autocodificador representa el valor de perdida obtenido después del entrenamiento

del autocodificador. Las siguientes columnas muestran el valor de precisión y error de las imágenes pronosticadas del paso 1 y el paso 2. Esta medida se obtiene mediante la Eq. 5.1, para la precisión, y la Eq. 5.2, para el error.

$$\frac{\sum_{i=1}^{n}\sum_{j=1}^{m} 1 - |original_{ij} - pronóstico_{ij}|}{nm} \tag{5.1}$$

$$\frac{\sum_{i=1}^{n}\sum_{j=1}^{m} |original_{ij} - pronóstico_{ij}|}{nm} \tag{5.2}$$

Donde n es la anchura y m es la altura de la imagen. Mediante la utilización de estas mediciones se obtienen el número de coincidencias entre la imagen original y la pronosticada, en otras palabras, se hace una comparación de que tantos píxeles son iguales entre ambas imágenes.

Para complementar los resultados de la Tabla 5.5, la Figura 5.12 muestra el tiempo, obtenido en minutos, que duraron los experimentos del pronóstico de imágenes. En donde se compara la cantidad de tiempo invertido con la cantidad de modelos a implementar.

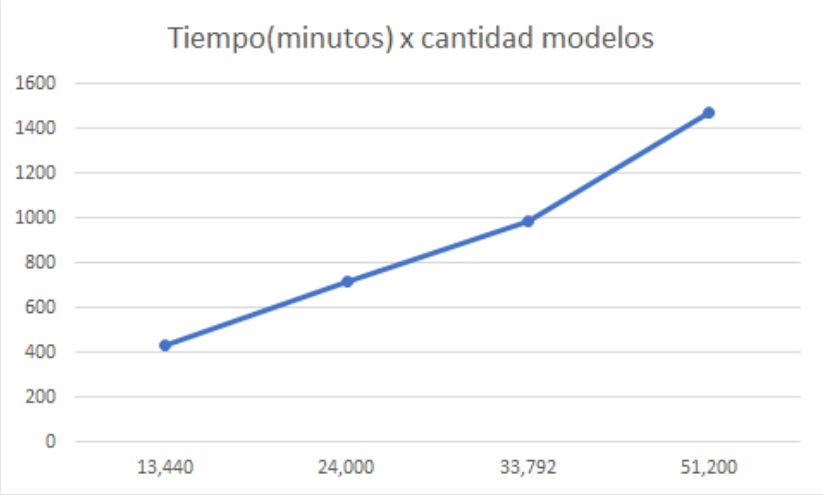

Figura 5.12: Relación cantidad de modelo por tiempo de ejecución

Tal como se muestra en la Figura 5.12 al tener un menor nivel de codificación, es decir, una mayor cantidad de modelos a implementar, se requiere una inversión mayor en el tiempo. A su vez, la Figura 5.13 y 5.14 muestran una gráfica con los valores de precisión y error, respectivamente, de los pasos 1 y 2 de los pronósticos.

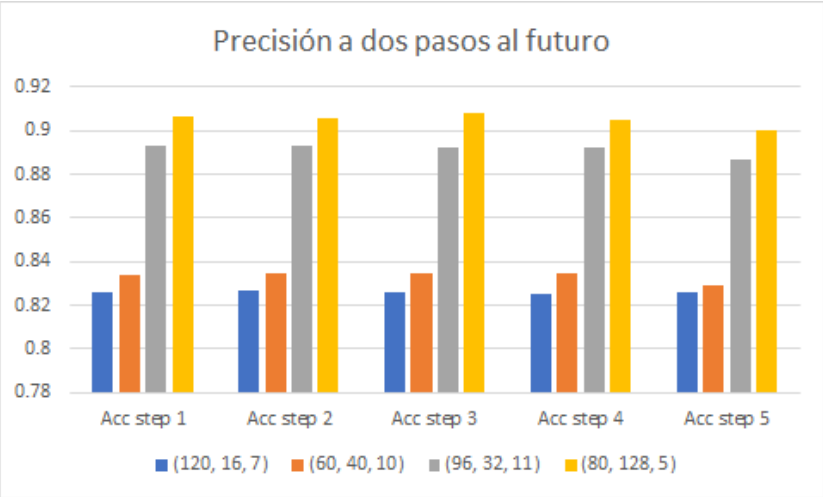

Figura 5.13: Precisión de pronóstico a diferentes niveles de codificación, paso 1 y 2

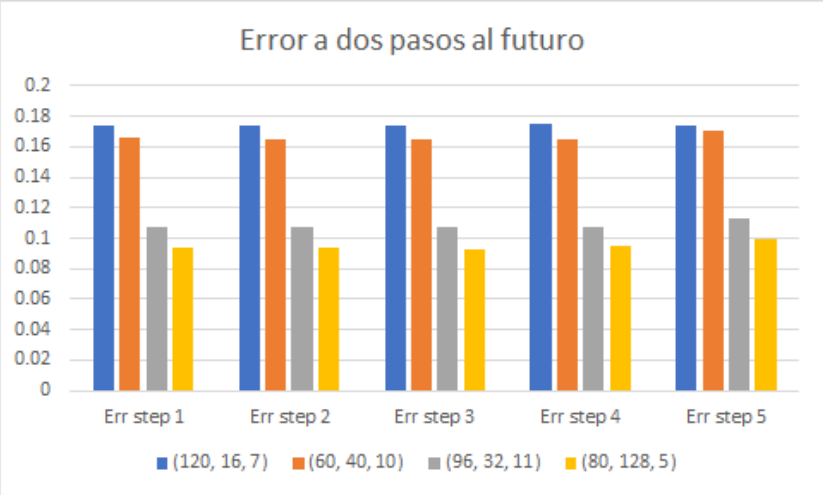

Figura 5.14: Error de pronóstico a diferentes niveles de codificación, paso 1 y 2

Poniendo en perspectiva la Figura 5.12 muestra que a mayor cantidad de características, mayor es el tiempo de implementación. Sin embargo, la Figura 5.1 y 5.2 muestran que no hay una ganancia significativa que justifique el tiempo de implementación a niveles tan bajos de codificación.

El objetivo consiste en que cada uno de los modelos utilizados pueda pronosticar, después de pasar por el proceso de entrenamiento, n pasos hacia el futuro, es decir una parte de una muestra codificada. Y posteriormente, utilizar este pronóstico codificado por el decodificador del autocodificador entrenado y obtener una imagen con las dimensiones originales

que representen el futuro. A manera de ejemplo la Figura 5.15, 5.16, 5.17 y 5.18 muestran el resultado de los pronósticos obtenidos en formato de imágenes a dos pasos hacia el futuro.

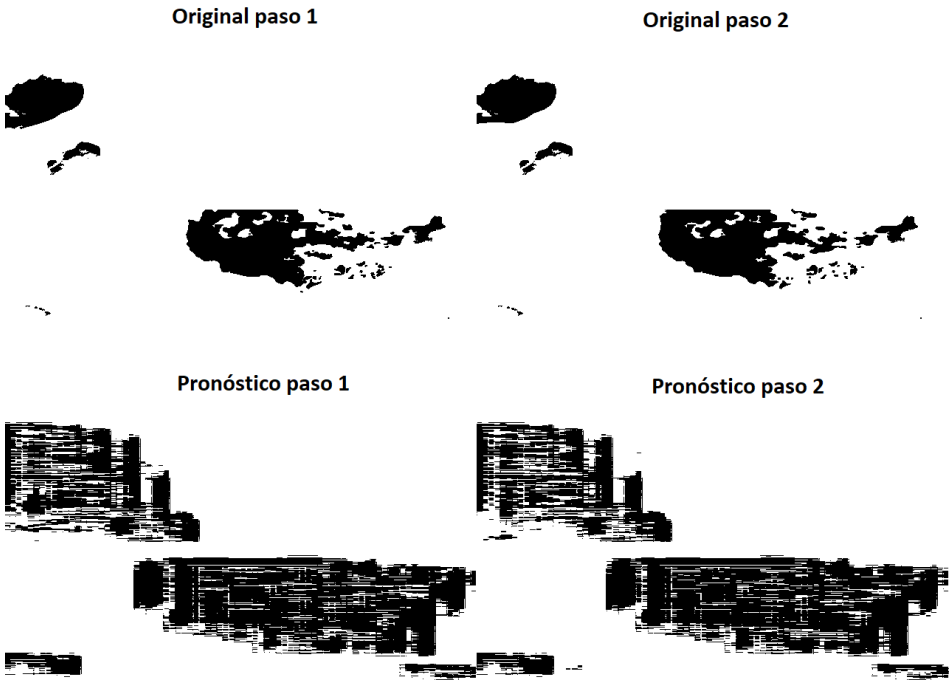

Figura 5.15: Imagen pronosticada con dimensiones de codificación (120,16,7)

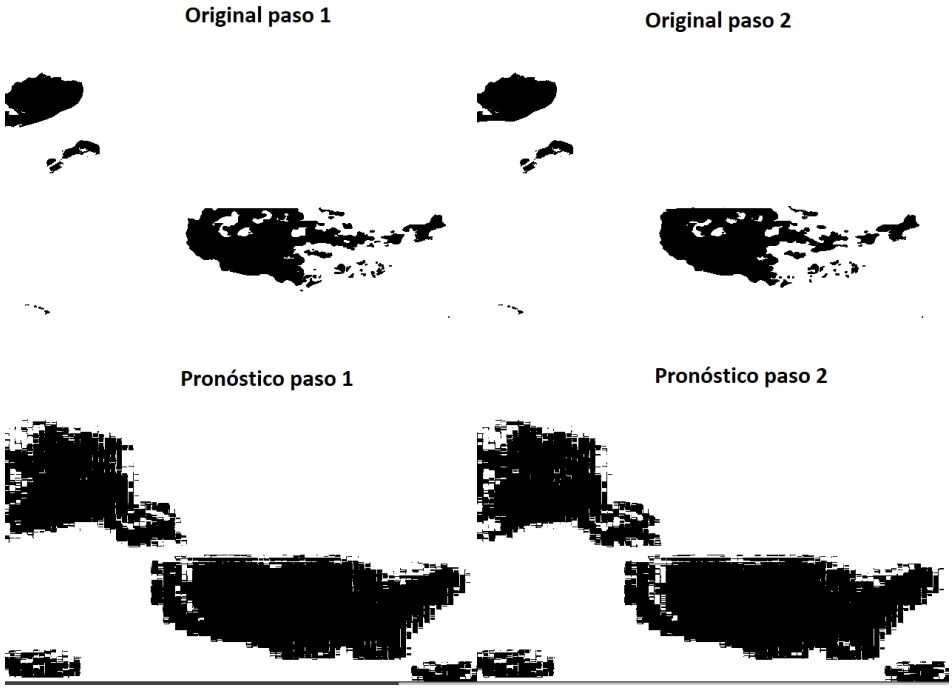

Figura 5.16: Imagen pronosticada con dimensiones de codificación (60,40,10)

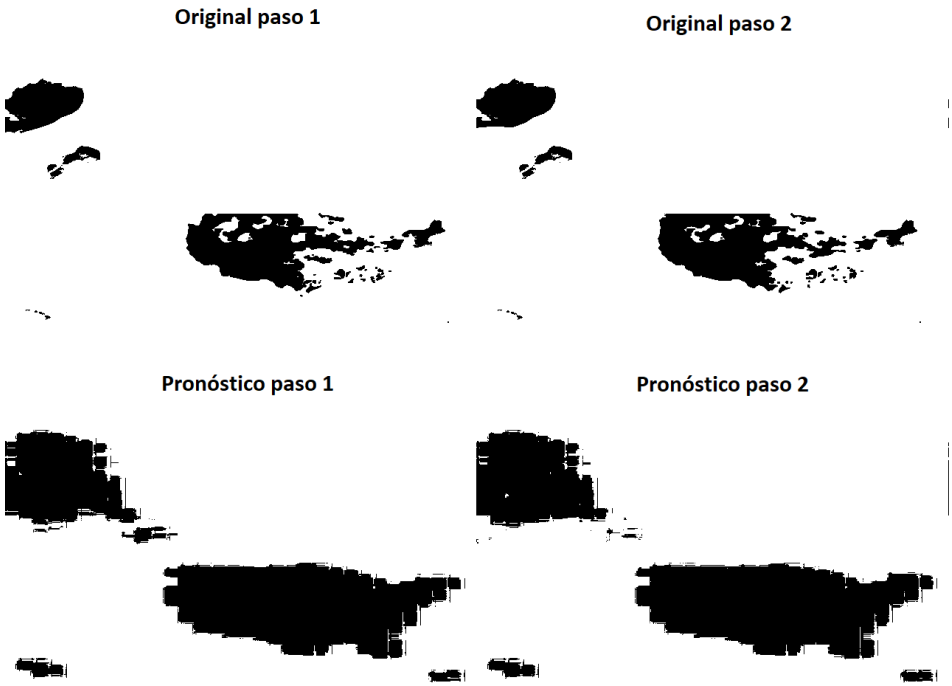

Figura 5.17: Imagen pronosticada con dimensiones de codificación (96,32,11)

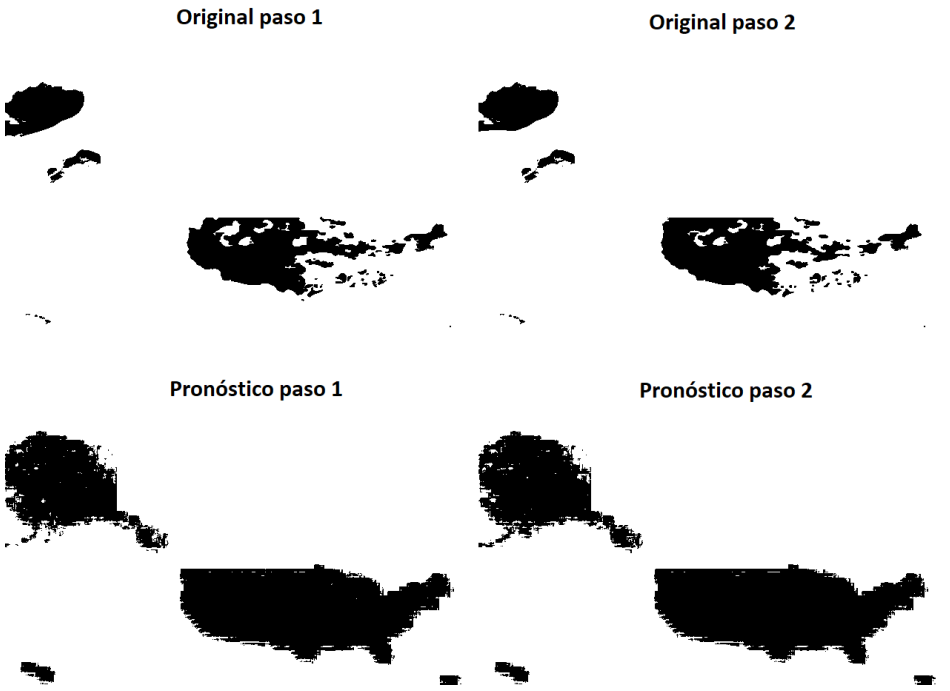

Figura 5.18: Imagen pronosticada con dimensiones de codificación (80,128,5)

Capítulo 6

Conclusiones

En este capítulo se describen las conclusiones obtenidas con base en la experimentación realizada y el flujo de trabajo del proyecto. A su vez se describen las aportaciones que ofrece el desarrollo de este proyecto y el trabajo futuro que puede ser realizado para mejorar y añadir más componentes a este proyecto de investigación.

6.1. Conclusiones del trabajo

El área de aprendizaje profundo es un objeto de gran interés para diferentes entidades, como pueden ser el académico, como objeto de investigación, desarrollo de proyectos innovadores, o de implementación industrial. Lo anterior se debe principalmente a los grandes logros que se han obtenido con la implementación de estas técnicas, entre las áreas que han obtenido grandes avances son el procesamiento de imágenes, la aplicación de series de tiempo, toma de decisiones empresariales, procesamiento de lenguaje natural y sistemas de enseñanza.

A pesar de los logros obtenidos mediante el aprendizaje profundo, estos han sido realizados gracias a grandes esfuerzos e inversiones, debido principalmente a que fueron diseñados de manera manual con recursos de hardware que suelen ser inaccesibles tanto para el usuario común, como para muchas de las organizaciones, esto debido al gran costo del hardware utilizado.

Sin embargo, el área de aprendizaje máquina automatizado tiene como objetivo contrarrestar estas limitaciones de esfuerzo y tiempo. Mediante una serie de técnicas hace más fácil

la utilización de técnicas de aprendizaje máquina, incentivando a más personas en muchas más áreas del conocimiento. Aunque los primeros proyectos en esta área utilizaban hardware muy costoso y potente, generando así resultados excelentes en los modelos, en la actualidad están en desarrollo tecnologías cada vez más eficientes y accesibles para el usuario común.

En este proyecto se implementan un conjunto de distintas técnicas que conforman las 3 partes básicas de un proyecto de *Neural Architecture Search* (NAS), que son: Espacio de búsqueda, estrategia de búsqueda y estrategia de estimación de rendimiento. En este proyecto se utilizó un espacio de búsqueda diverso que soporta soluciones para clasificación de imágenes, regresión y series temporales. Para la estrategia de búsqueda se implementaron dos algoritmos, el principal siendo el algoritmo bayesiano de tipo *Tree-structured Parsen Estimator* (TPE), que guía a la fase de exploración, junto con el algoritmo de Búsqueda aleatoria para generar los primeros modelos, además se implementó un salón de la fama (HoF) para la fase de entrenamiento completo, donde se implementaron solo las arquitecturas más prometedoras. Por último, para la estrategia de estimación de rendimiento se implementa una técnica de entrenamiento parcial, esta estrategia aporta una serie de beneficios, como pueden ser: mayor rapidez de exploración en el espacio de búsqueda, menores tiempos de ejecución y obtener los modelos más prometedores con una clasificación de los mejores en un HoF.

El proyecto de optimización fue probado con diferentes tipos de *Dataset* para diferentes tareas, estas son: Clasificación de imágenes, regresión y series de tiempo. En los cuales se han entregado resultados aceptables comparándolos con el estado del arte. A su vez, los resultados obtenidos en las diferentes áreas son muy prometedores, por ejemplo, con esta implementación es posible resolver problemas de regresión de manera satisfactoria en una cantidad de tiempo reducido. Lo anterior puede deberse a diferentes puntos, puede ser por el tipo de problema al que se le está buscando solución, así como también por el número de trabajadores que se implementan o el hardware utilizado. Sin embargo, gracias a la flexibilidad que provee el proyecto, debido al diseño del modelo maestro-trabajador, permite el despliegue del sistema de manera distribuida y así implementar múltiples GPU en diferentes equipos o en el mismo equipo.

Gracias a la aplicación de un espacio de búsqueda dinámico, que permite la aplicación de múltiples espacios de búsqueda, es posible para el proceso de optimización de arquitecturas

de aprendizaje máquina obtener modelos apropiados para múltiples tipos de problemas con diferentes características entre ellos.

Este proyecto ofrece un nuevo apartado que es el pronóstico de series de tiempo de imágenes, donde para obtener un conjunto de imágenes que represente los siguientes n pasos en el tiempo, se necesita la implementación de múltiples herramientas en conjunto para su implementación. Si bien el procedimiento puede ser complicado, principalmente debido a la gran cantidad de recursos y tiempo de implementación necesarios, los resultados mostrados en la Tabla 5.5, en conjunto con las imágenes resultantes del pronóstico, son muy positivos y se puede concluir que si se sigue desarrollando esta línea de investigación, los resultados puede mejorar de manera significativa.

Los experimentos realizados para el pronóstico de imágenes fueron un gran reto. Los experimentos más extensos realizados llegaban a necesitar alrededor de los 80 GB de espacio para almacenar los miles de modelos necesarios para realizar un pronóstico. De igual manera, el tiempo necesario para la implementación con grandes cantidades de características puede llegar a tomar varios días de implementación, aunque dependa de la cantidad de trabajadores y recursos disponibles.

6.2. Aportaciones

Entre las aportaciones principales que engloban a este proyecto, se encuentran su combinación única de estrategias y algoritmos que le permiten la generación de modelos de aprendizaje profundo optimización para las tareas de clasificación de imágenes, regresión y series de tiempo con una alta eficiencia en cuanto a tiempo y los recursos limitados. Otra de las aportaciones principales es el pronóstico de series de tiempo de imágenes que aporta una forma innovadora para el pronóstico de imágenes que puede mejorarse si se llegará a implementar un proceso automatizado para generar modelos de series de tiempo para cada serie de tiempo implementada. Aunque la sugerencia anterior llevaría una gran cantidad de tiempo a implementar podría llegar a mejorar muchísimo el rendimiento de la aplicación. De manera específica, las aportaciones de este proyecto son las siguientes:

- Implementación de fase de exploración y fase de entrenamiento completo en el espacio

de búsqueda.

- Diseño del espacio de búsqueda que soporte diferentes tipos de optimización, para clasificación de imágenes, regresión y series de tiempo.
- Proceso de exploración guiado por el algoritmo Bayesiano TPE.
- Mejora en eficiencia de optimización mediante sistemas distribuidos.
- Generación de modelos de aprendizaje profundo específicos para tareas de clasificación de imágenes, regresión y series de tiempo.
- Pronósticos de series de tiempo de imágenes a n pasos hacia el futuro.
- Utilización de autocodificadores para la reducción de la dimensionalidad.
- Implementación de muestras codificadas para realizar pronósticos.

6.3. Trabajo futuro

El presente trabajo está diseñado para poder ser ampliado y/o mejorado de manera sencilla. Una de las mejoras que se pueden implementar sería la sustitución del algoritmo Bayesiano TPE por otro algoritmo de optimización de caja negra, esto podría permitir una mejora en la generación de modelos. Otra área puede ser la ampliación del espacio de búsqueda para diferentes tareas, pueden ser procesamiento de lenguaje natural, identificación de objetos o reducción de dimensionalidad. Otra mejora a realizar, puede ser un control de sobre-ajuste y sub-ajuste para evitar que los modelos de optimización obtengan resultados erróneos.

Para el apartado de pronóstico de series de tiempo de imágenes, algunas mejoras pueden consistir en la implementación de algoritmos de codificación diferentes. También pueden mejorarse los modelos de series de tiempo implementados, o incluso utilizar una estrategia de optimización automatizada para una generación especifica de cada modelo. Otra característica que se puede mejorar es la implementación de una diferente estrategia de pronóstico a múltiples pasos hacia el futuro. En el caso de este proyecto se utiliza una estrategia recursiva, pero también puede ser implementada una estrategia directa, MIMO, DirRec o DIRMO.

Bibliografía

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M. et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (page 51).

Aliani, H., Malmir, M., Sourodi, M. & Kafaky, S. B. (2019). Change detection and prediction of urban land use changes by CA–Markov model (case study: Talesh County). *Environmental Earth Sciences*, *78*(17), 1-12 (page 59).

Alonso-Montesinos, J., Polo, J., Ballestrın, J., Batlles, F. & Portillo, C. (2019). Impact of DNI forecasting on CSP tower plant power production. *Renewable Energy*, *138*, 368-377 (page 59).

Ayet, A. & Tandeo, P. (2018). Nowcasting solar irradiance using an analog method and geostationary satellite images. *Solar Energy*, *164*, 301-315 (page 59).

Benardos, P. G. & Vosniakos, G. C. (2007). Optimizing feedforward artificial neural network architecture. *Engineering applications of artificial intelligence, 20(3), 365-382.* (page 38).

Benediktsson, J. A., Swain, P. H. & Ersoy, O. K. (1990). Neural network approaches versus statistical methods in classification of multisource remote sensing data (page 1).

Bengio, I., Y.and Goodfellow & Courville, A. (2017). *Deep learning (Vol. 1)*. Massachusetts, USA: MIT press. (Pages 12, 18, 46).

Bergstra, J., Bardenet, R., Bengio, Y. & Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, *24* (page 41).

Berhan, G., Hill, S., Tadesse, T. & Atnafu, S. (2013). Drought prediction system for improved climate change mitigation. *IEEE Transactions on Geoscience and Remote sensing*, *52*(7), 4032-4037 (page 59).

Bose, P., Kasabov, N. K., Bruzzone, L. & Hartono, R. N. (2016). Spiking neural networks for crop yield estimation based on spatiotemporal analysis of image time series. *IEEE Transactions on Geoscience and Remote Sensing*, *54*(11), 6563-6573 (page 59).

Brownlee, J. (2018). *Deep learning for time series forecasting: predict the future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery. (Page 57).

Burkov, A. (2019). The hundred-page machine learning book (Vol. 1). (Pages 11-13, 19).

Burns, B. (2018). *Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services*. O'Reilly Media, Inc. (Pages 47, 48).

Carriere, T., Vernay, C., Pitaval, S. & Kariniotakis, G. (2019). A novel approach for seamless probabilistic photovoltaic power forecasting covering multiple time frames. *IEEE Transactions on Smart Grid*, *11*(3), 2281-2292 (page 59).

Cheng, H.-Y. & Yu, C.-C. (2016). Solar irradiance now-casting with ramp-down event prediction via enhanced cloud detection and tracking. *2016 IEEE International Conference on Multimedia and Expo (ICME)*, 1-6 (page 59).

Chow, C. W., Urquhart, B., Lave, M., Dominguez, A., Kleissl, J., Shields, J. & Washom, B. (2011). Intra-hour forecasting with a total sky imager at the UC San Diego solar energy testbed. *Solar Energy*, *85*(11), 2881-2893 (page 58).

Conway, D. & White, J. (2012). *Machine learning for hackers*. "O'Reilly Media, Inc." (Page 11).

Cornejo-Bueno, L., Borge, J. N., Alexandre, E., Hessner, K. & Salcedo-Sanz, S. (2016). Accurate estimation of significant wave height with support vector regression algorithms and marine radar images. *Coastal Engineering*, *114*, 233-243 (page 59).

Coulouris, G., Dollimore, J., Kindberg, T. D. & G., B. (2012). *Distributed Systems: Concepts and Design Edition 5*. System, 2(11), 15. (Pages 47, 48).

Dambreville, R., Blanc, P., Chanussot, J. & Boldo, D. (2014). Very short term forecasting of the global horizontal irradiance using a spatio-temporal autoregressive model. *Renewable Energy*, *72*, 291-300 (page 59).

Das, M. & Ghosh, S. K. (2016). Deep-STEP: A deep learning approach for spatiotemporal prediction of remote sensing data. *IEEE Geoscience and Remote Sensing Letters*, *13*(12), 1984-1988 (page 59).

Deb, C., Zhang, F., Yang, J., Lee, S. E. & Shah, K. W. (2017). A review on time series forecasting techniques for building energy consumption. *Renewable and Sustainable Energy Reviews*, *74*, 902-924 (page 57).

Dong, Z., Yang, D., Reindl, T. & Walsh, W. M. (2014). Satellite image analysis and a hybrid ESSS/ANN model to forecast solar irradiance in the tropics. *Energy Conversion and Management*, *79*, 66-73 (page 59).

Falkner, S., Klein, A. & Hutter, F. (2018). BOHB: Robust and efficient hyperparameter optimization at scale. *International Conference on Machine Learning*, 1437-1446 (pages 55, 56).

Gamboa, J. C. B. (2017). Deep learning for time-series analysis. *arXiv preprint arXiv:1701.01887* (page 57).

Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media. (Pages 31, 47).

Goodfellow, I., Bengio, Y., Courville, A. & Bengio, Y. (2016). Deep learning (Vol. 1, No. 2). (Pages 17, 21, 22, 26, 28, 33, 34).

Grinberg, M. (2021). Flask-socketio documentation. *línea]. Disponible en: https://flask-socketio. readthedocs. org/en/latest/.[Último acceso: 2015]* (page 53).

Hao, P., Löw, F. & Biradar, C. (2018). Annual cropland mapping using reference Landsat time series—a case study in Central Asia. *Remote Sensing*, *10*(12), 2057 (page 59).

He, X., Zhao, K. & Chu, X. (2021). AutoML: A Survey of the State-of-the-Art. *Knowledge-Based Systems*, *212*, 106622 (pages 1, 35-39, 56, 70).

Hutter, F., Kotthoff, L. & Vanschoren, J. (2019). *Automated machine learning: methods, systems, challenges*. Springer Nature. (Pages 35, 37-40).

Ionescu, V. M. (2015). The analysis of the performance of RabbitMQ and ActiveMQ. *2015 14th RoEduNet International Conference-Networking in Education and Research (RoEduNet NER)*, 132-137 (pages 51, 52).

Jin, H., Song, Q. & Hu, X. (2019). Auto-keras: An efficient neural architecture search system. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1946-1956 (page 56).

Joshi, A. V. (2020). *Machine learning and artificial intelligence*. Springer. (Pages 8, 10, 11, 23, 29, 32, 43, 45).

Kohavi, R. & John, G. H. (1995). Automatic parameter selection by minimizing estimated error. *In Machine Learning Proceedings 1995 (pp. 304-312). Morgan Kaufmann.* (page 35).

Kruse, R., Borgelt, C., Braune, C., Mostaghim, S. & Steinbrecher, M. (s.f.). Computational Intelligence-A Methodological Introduction, 2016. (Page 15).

LeCun, Y. A., Bottou, L., Orr, G. B. & Müller, K.-R. (2012). Efficient backprop. *Neural networks: Tricks of the trade* (pp. 9-48). Springer. (Page 69).

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A. & Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research, 18(1), 6765-6816.* (pages 37, 55, 56).

Lin, M., Chen, Q. & Yan, S. (2013). Network in network. *arXiv preprint arXiv:1312.4400* (page 76).

Ma, X., Zhang, J., Du, B., Ding, C. & Sun, L. (2018). Parallel architecture of convolutional bi-directional LSTM neural networks for network-wide metro ridership prediction. *IEEE Transactions on Intelligent Transportation Systems*, *20*(6), 2278-2288 (pages 59, 60).

Mafi-Gholami, D., Zenner, E. K., Jaafari, A. & Bui, D. T. (2020). Spatially explicit predictions of changes in the extent of mangroves of Iran at the end of the 21st century. *Estuarine, Coastal and Shelf Science*, *237*, 106644 (page 59).

Magnone, L., Sossan, F., Scolari, E. & Paolone, M. (2017). Cloud motion identification algorithms based on all-sky images to support solar irradiance forecast. *2017 IEEE 44th Photovoltaic Specialist Conference (PVSC)*, 1415-1420 (page 57).

Mahalakshmi, G., Sridevi, S. & Rajaram, S. (2016). A survey on forecasting of time series data. *2016 International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE'16)*, 1-8 (page 43).

Marquez, R., Pedro, H. T. & Coimbra, C. F. (2013). Hybrid solar forecasting method uses satellite imaging and ground telemetry as inputs to ANNs. *Solar Energy*, *92*, 176-188 (page 59).

McCarthy, J. (1998). What is artificial intelligence? (Page 8).

Meng, J., Zhang, X., Zhang, X., Xie, Y. & Yuan, B. (2011). Analysis and prediction of land use and land cover change: A case study of Minjiang River, China. *2011 19th International Conference on Geoinformatics*, 1-4 (page 59).

Microsoft. (2021). A tour of the C# language [[Online; accesado 31-mayo-2021]]. (Page 52).

Mitchell, T. M. et al. (1997). Machine learning (pages 11, 24).

Montgomery, D. C., Jennings, C. L. & Kulahci, M. (2015). *Introduction to time series analysis and forecasting*. John Wiley & Sons. (Page 42).

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press. (Page 13).

Naik, N. (2017, October). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. *2017 IEEE international systems engineering symposium (ISSE) (pp. 1-7). IEEE* (page 49).

Nduati, E., Sofue, Y., Matniyaz, A., Park, J. G., Yang, W. & Kondoh, A. (2019). Cropland Mapping Using Fusion of Multi-Sensor Data in a Complex Urban/Peri-Urban Area. *Remote Sensing*, *11*(2), 207 (page 59).

Negnevitsky, M. & Intelligence, A. (2005). A guide to intelligent systems. *Artificial Intelligence, 2nd edition, pearson Education* (pages 8-10, 14-17, 25).

Norman, L. M., Feller, M. & Guertin, D. P. (2009). Forecasting urban growth across the United States–Mexico border. *Computers, Environment and Urban Systems*, *33*(2), 150-159 (page 59).

Oliphant, T. E. (2007). Python for scientific computing. *Computing in Science Engineering, 9(3), 10-20.* (page 50).

Pal, S., Chowdhury, S. & Ghosh, S. K. (2016). DCAP: a deep convolution architecture for prediction of urban growth. *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, 1812-1815 (page 59).

Patterson, J. & Gibson, A. (2017). *Deep learning: A practitioner's approach*. .ºReilly Media, Inc." (Pages 13-15, 17, 19-32, 44).

Pelikan, M., Goldberg, D. E. & Cantú-Paz, E. (1999, July). BOA: The Bayesian optimization algorithm. *In Proceedings of the genetic and evolutionary computation conference GECCO-99 (Vol. 1, pp. 525-532)* (page 41).

Rafiee, R., Mahiny, A. S., Khorasani, N., Darvishsefat, A. A. & Danekar, A. (2009). Simulating urban growth in Mashad City, Iran through the SLEUTH model (UGM). *Cities*, *26*(1), 19-26 (page 59).

Springenberg, J. T., Dosovitskiy, A., Brox, T. & Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806* (page 77).

Suganuma, M., Shirakawa, S. & Nagao, T. (2017). A genetic programming approach to designing convolutional neural network architectures. *Proceedings of the genetic and evolutionary computation conference*, 497-504 (page 56).

Wen, H., Du, Y., Chen, X., Lim, E., Wen, H., Jiang, L. & Xiang, W. (2020). Deep Learning Based Multistep Solar Forecasting for PV Ramp-Rate Control Using Sky Images. *IEEE Transactions on Industrial Informatics*, *17*(2), 1397-1406 (page 59).

Xu, Z., Du, J., Wang, J., Jiang, C. & Ren, Y. (2019). Satellite Image Prediction Relying on GAN and LSTM Neural Networks. *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, 1-6 (page 59).

Yan, W., Liu, J., Zhang, M., Hu, L. & Chen, J. (2017). Outburst flood forecasting by monitoring glacier-dammed lake using satellite images of Karakoram Mountains, China. *Quaternary International*, *453*, 24-36 (page 59).

Yang, L., Gao, X., Li, Z., Jia, D. & Jiang, J. (2019). Nowcasting of surface solar irradiance using FengYun-4 satellite observations over China. *Remote Sensing*, *11*(17), 1984 (page 59).

Yuan, Y., Lin, L., Huo, L.-Z., Kong, Y.-L., Zhou, Z.-G., Wu, B. & Jia, Y. (2020). Using An Attention-Based LSTM Encoder–Decoder Network for Near Real-Time Disturbance Detection. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *13*, 1819-1832 (page 59).

Zambrano, F., Vrieling, A., Nelson, A., Meroni, M. & Tadesse, T. (2018). Prediction of drought-induced reduction of agricultural productivity in Chile from MODIS, rainfall estimates, and climate oscillation indices. *Remote sensing of environment*, *219*, 15-30 (page 59).

Zhang, G., Zhou, H., Wang, C., Xue, H., Wang, J. & Wan, H. (2020). Forecasting Time Series Albedo Using NARnet Based on EEMD Decomposition. *IEEE Transactions on Geoscience and Remote Sensing*, *58*(5), 3544-3557 (page 59).

Zhang, W., Chen, E., Li, Z., Zhao, L., Ji, Y., Zhang, Y. & Liu, Z. (2018). Rape (Brassica napus L.) growth monitoring and mapping based on radarsat-2 time-series data. *Remote Sensing*, *10*(2), 206 (page 59).

Zoph, B. & Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (pages 1, 55, 56).