UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

De Python a Kubeflow

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Gutiérrez Villalba, Sergio

Tutor/a: Acebrón Linuesa, Floreal

CURSO ACADÉMICO: 2023/2024

Resumen

El trabajo en el campo de la inteligencia artificial y específicamente en el del aprendizaje automático con las redes neuronales como punta de lanza, está cobrando más relevancia que nunca. La aparición de herramientas que emplean modelos entrenados como GPT-3 o GPT-4 han popularizado el uso de la inteligencia artificial a través de aplicaciones como ChatGPT o Claude.

Dentro de este contexto y del avance futuro de campos como la IA generativa o el reconocimiento del lenguaje natural, se resalta la relevancia y el impacto de los recursos educativos que permiten introducir de forma eficaz y didáctica a cualquier ingeniero en estos temas. Siguiendo esta línea, el trabajo propuesto aborda el análisis, configuración y uso de tecnologías que permiten entrenar y poner a punto modelos de redes neuronales, de forma robusta, escalable y eficiente en recursos.

Palabras clave: redes neuronales, inteligencia artificial, python, Kubeflow, TensorFlow, servicios en la nube, Vertex AI, Google Cloud Platform

Abstract

The work in the field of artificial intelligence, specifically in machine learning with neural networks as the spearhead, is gaining more relevance than ever. The emergence of tools that employ trained models like GPT-3 or GPT-4 has popularized the use of artificial intelligence through applications like ChatGPT or Claude.

Within this context and the future advancement of areas such as generative AI or natural language recognition, the significance and impact of educational resources that effectively and didactically introduce any engineer to these subjects are highlighted. Following this line, the proposed work addresses the analysis, configuration, and utilization of technologies that enable the training and fine-tuning of neural network models in a robust, scalable, and resource-efficient manner.

Keywords: neural networks, artificial intelligence, python, Kubeflow, TensorFlow, cloud services, Vertex AI, Google Cloud Platform

Tabla de contenidos

Tabla de figuras

1. Introducción

En los últimos años, el campo de la inteligencia artificial ha experimentado un crecimiento sin precedentes, impulsado principalmente por el desarrollo y la aplicación de las redes neuronales. Estas redes son modelos computacionales que se inspiran en la estructura y el funcionamiento del cerebro humano, y han demostrado su eficacia en una amplia gama de aplicaciones, como reconocimiento de voz, procesamiento de imágenes, traducción automática y más. En un mundo cada vez más impulsado por los datos y la inteligencia artificial, comprender cómo implementar y mejorar modelos de redes neuronales es de vital importancia.

En el capítulo de redes neuronales se abordarán los conceptos fundamentales y se explicará cómo una neurona funciona como una unidad básica de procesamiento, procesando entradas, aplicando pesos y produciendo salidas mediante funciones de activación. Además, se explorará cómo se realiza la combinación de estas neuronas en capas totalmente conexas, permitiendo que las redes neuronales aprendan a representar relaciones complejas en los datos.

En el siguiente apartado, se profundizará en la implementación de redes neuronales utilizando Python, un lenguaje de programación popular en el ámbito del aprendizaje automático. Se cubrirán aspectos como la configuración del entorno de trabajo y la instalación de las bibliotecas necesarias, como NumPy y TensorFlow, para llevar a cabo el desarrollo de modelos de redes neuronales. Posteriormente, se avanzará hacia el uso de la biblioteca Keras, una interfaz de alto nivel para la construcción y entrenamiento de modelos de redes neuronales en Python. A través de Keras, se puede simplificar la implementación y configuración de arquitecturas de redes neuronales, lo que permite el enfoque del desarrollador en el diseño y ajuste de hiper parámetros para mejorar el rendimiento de los modelos. También se realiza una mención a Jupyter Notebooks, un entorno de desarrollo iterativo que aporta versatilidad a cualquier proyecto mediante su interactividad e interfaz amigable y de fácil uso.

Avanzando hacia entornos escalables, se introduce Kubernetes como eje central de las tecnologías en la nube, el cual habilita la escalabilidad en el uso de recursos de una forma estandarizada. Esto sienta las bases para optimizar y automatizar el ciclo de vida de los modelos de redes neuronales.

Posteriormente, se analiza el uso de herramientas en el ámbito cloud, haciendo especial énfasis en Kubeflow, una plataforma de código abierto para el despliegue, gestión y escalado de flujos de trabajo de aprendizaje automático en entornos Kubernetes. Con Kubeflow, se da un paso más allá en la abstracción de la implementación y el centro de interés pasa a ser la escalabilidad y el despliegue de modelos, lo que permite realizar tareas de aprendizaje profundo a gran escala con una eficiencia y rendimiento mejorados.

Las ventajas que se obtienen con Kubeflow son interesantes desde el punto de vista comercial, por ello, las empresas líderes del sector se han enfocado a lanzar sus propias soluciones. Para contrastar que ofrece un proyecto privado respecto a uno de código libre, se realiza una comparación del funcionamiento y el impacto de Kubeflow versus Vertex AI, la solución ofrecida por Google Cloud (GCP) para el ámbito del aprendizaje automático.

A lo largo de este TFG, se busca destacar de forma conceptual y didáctica las ventajas e inconvenientes que ofrecen los diferentes entornos en los que se trabaja, iterando desde ámbitos de prueba básicos hasta una pila de trabajo que puede automatizar el ciclo de vida completo de un modelo de red neuronal, incluyendo su despliegue en producción. El objetivo es dotar al lector del contexto necesario para poder escoger las herramientas con las se pueden abordar problemas de aprendizaje automático de manera efectiva y mediante el uso de estas, maximizar el potencial de las redes neuronales en el análisis y procesamiento de datos de una forma eficiente.

1.1. Motivación

1.1.1. Motivación personal

Durante el transcurso de mis estudios en el Grado de Ingeniería Informática, surgió mi interés por la inteligencia artificial, especialmente por las redes neuronales. El hecho de trabajar en el campo y poder ayudar a democratizar el uso del potencial de este paradigma de programación en el día a día han sido los principales motivadores. Tras la publicación de herramientas como ChatGPT y los modelos GPT-3 y GPT-4 se ha podido demostrar que la IA ha venido para quedarse, tiene infinidad de usos y permite facilitar la vida a cualquier persona hasta límites que seguramente aún desconocemos.

Poder explorar la forma de habilitar estas herramientas para ser efectivas, incluyendo su puesta en escena de forma práctica, me resulta un objetivo sumamente interesante.

1.1.2. Motivación profesional

Mi motivación profesional principal para aprender sobre los conceptos y herramientas relacionadas con las redes neuronales y el aprendizaje automático, es la relevancia que está asumiendo este campo de la computación en el mundo laboral actual. En primer lugar, el auge de la inteligencia artificial y el aprendizaje automático ha transformado la forma en que las empresas abordan sus desafíos y toman decisiones estratégicas. Dominar estas tecnologías permite a los profesionales destacarse en su campo y estar preparados para abordar problemas complejos y resolverlos de manera más eficiente y precisa.

Además, el aprendizaje automático se ha convertido en una herramienta indispensable para el análisis y procesamiento de grandes cantidades de datos en diversos sectores, desde la medicina y la biotecnología hasta el marketing y la industria financiera. Un profesional con conocimientos en redes neuronales estará en una posición privilegiada para diseñar modelos de aprendizaje automático que impulsen la toma de decisiones fundamentadas y logren una mayor eficiencia en la resolución de problemas comerciales.

Asimismo, el aprendizaje automático es esencial para la automatización de tareas repetitivas y la mejora de la productividad en diversas áreas laborales. Con un enfoque en el desarrollo de redes neuronales y su capacidad para evolucionar y aumentar su impacto, los profesionales pueden

liberar tiempo y recursos valiosos, permitiendo que la tecnología se encargue de tareas tediosas y repetitivas, y centrarse en actividades más estratégicas y creativas.

Adicionalmente, el conocimiento en redes neuronales y aprendizaje automático es altamente demandado en el mercado laboral actual. Las empresas buscan activamente profesionales que puedan implementar y optimizar modelos de aprendizaje automático para obtener una ventaja competitiva. Aprender sobre estas tecnologías abre oportunidades de empleo en empresas líderes en tecnología, instituciones de investigación y startups innovadoras, ofreciendo una trayectoria profesional en constante crecimiento y enriquecimiento.

Por último, mi motivación profesional para aprender sobre la implementación de redes neuronales también radica en el hecho de que estas tecnologías tienen un impacto significativo en la sociedad y el bienestar de las personas. Al aplicar el aprendizaje automático de manera ética y responsable, los profesionales pueden contribuir a la resolución de problemas sociales y médicos, mejorando la atención médica, la sostenibilidad ambiental y la calidad de vida en general. Esto brinda una profunda satisfacción personal y profesional al saber que el conocimiento adquirido se utiliza para el beneficio común y el avance de la humanidad. En resumen, mi motivación profesional para aprender sobre redes neuronales y el aprendizaje automático es impulsada por la necesidad de destacar en un mercado laboral competitivo y en constante evolución, la oportunidad de contribuir a la mejora de la productividad y eficiencia en diversas áreas laborales, la demanda creciente de habilidades en este campo, la posibilidad de obtener oportunidades de empleo en empresas líderes y la capacidad de generar un impacto positivo en la sociedad.

1.2. Objetivos

Los objetivos principales de este trabajo son los siguientes:

- Explorar los fundamentos teóricos de las redes neuronales, incluyendo la comprensión de los conceptos básicos, como las neuronas, los parámetros, las funciones de activación, las épocas, las pérdidas y los lotes.

- Investigar y comparar diferentes tecnologías y bibliotecas basadas en Python que permitan implementar redes neuronales, como Keras, Tensorflow y Jupyter Notebooks.

- Investigar y comparar soluciones en la nube que permitan escalar el ciclo de vida de un modelo de red neuronal, cómo son Kubeflow y Vertex AI.

- Realizar implementaciones prácticas de redes neuronales utilizando las herramientas seleccionadas.

- Evaluar las ventajas y desventajas de cada tecnología, identificando los casos de uso más apropiados y las mejores prácticas para el desarrollo y despliegue de redes neuronales.

A lo largo de este trabajo, se abordarán los aspectos teóricos y prácticos necesarios para comprender y aplicar eficazmente las redes neuronales en Python. Se proporcionarán ejemplos y se realizarán comparaciones entre las diferentes tecnologías, con el objetivo de brindar una visión

completa y detallada de las opciones disponibles y las consideraciones clave en la implementación de redes neuronales. Al final, se presentarán las conclusiones obtenidas, resaltando las principales lecciones aprendidas y las posibles direcciones futuras en este campo en constante evolución.

2. Estado del arte

2.1. IA y redes neuronales en la actualidad, impacto

La inteligencia artificial (IA) y las redes neuronales han experimentado un rápido avance y un crecimiento exponencial en las últimas décadas. Estos campos han revolucionado la forma en que las máquinas pueden aprender, razonar y tomar decisiones, lo que ha llevado a importantes avances en una amplia gama de aplicaciones.

El auge de las redes neuronales ha sido uno de los pilares fundamentales en el desarrollo de la IA moderna. Estas estructuras inspiradas en el funcionamiento del cerebro humano han demostrado una capacidad única para modelar relaciones complejas en grandes cantidades de datos. Desde redes neuronales simples hasta modelos profundos y complejos, como las redes neuronales convolucionales y recurrentes. Estas arquitecturas han superado los resultados obtenidos por enfoques tradicionales en tareas como reconocimiento de patrones, procesamiento de imágenes y procesamiento del lenguaje natural.

En el ámbito de la visión por computador, las redes neuronales convolucionales han alcanzado un nivel de precisión sorprendente en la detección y clasificación de objetos en imágenes y vídeos. Modelos como *ResNet*, *Inception* y *EfficientNet* han batido récords [14] en competiciones de reconocimiento de imágenes y han impulsado aplicaciones prácticas en campos como la conducción autónoma y la seguridad.

En el procesamiento del lenguaje natural, los modelos de lenguaje basados en transformadores, como BERT y GPT-3, han revolucionado la forma en que las máquinas comprenden y generan texto. Estos modelos han demostrado una capacidad asombrosa para responder preguntas, realizar tareas de traducción y generar texto de manera coherente y natural.

En el campo de la salud y la medicina, las redes neuronales han sido aplicadas para diagnosticar enfermedades, predecir riesgos y personalizar tratamientos. La IA ha demostrado ser una herramienta valiosa para analizar grandes conjuntos de datos médicos y mejorar la precisión y eficiencia en el diagnóstico y el tratamiento de pacientes.

Además, el aprendizaje profundo y las redes neuronales también han tenido un impacto significativo en la industria financiera, donde se utilizan para el análisis de riesgos, la detección de fraudes y la optimización de carteras de inversión.

Sin embargo, a pesar de los impresionantes avances, la industria sigue enfrentando serios desafíos. El entrenamiento de modelos profundos puede requerir grandes cantidades de datos y una capacidad computacional significativa. Además, la interpretación de los resultados de los modelos de IA sigue siendo un punto importante a mejorar, lo que puede dificultar la confianza en las decisiones tomadas por las máquinas.

2.2. El potencial de Python y las tecnologías cloud en este campo

Python se ha convertido en el lenguaje de primera elección para el desarrollo de redes neuronales por una serie de ventajas que lo hacen especialmente adecuado para este propósito. En primer lugar, la amplia comunidad y el ecosistema de bibliotecas y marcos de trabajo especializados en IA, como TensorFlow, PyTorch y Keras, hacen que Python sea extremadamente conveniente para los desarrolladores. Estas bibliotecas proporcionan implementaciones eficientes de algoritmos de aprendizaje automático y redes neuronales, lo que permite a los desarrolladores centrarse en el diseño de modelos y en la resolución de problemas, en lugar de tener que reinventar la rueda.

Además, la naturaleza de código abierto de Python permite una colaboración y contribución activa de la comunidad. Esto ha llevado a un rápido desarrollo y mejora de las herramientas de IA en este lenguaje, lo que lo mantiene a la vanguardia de los avances en el campo de la inteligencia artificial. Se han producido una gran cantidad de tutoriales, documentación y recursos educativos para aprender y dominar las técnicas de IA en Python, lo que facilita el proceso de aprendizaje para desarrolladores e investigadores.

Otra ventaja clave de Python es su facilidad de uso y legibilidad. La sintaxis clara y expresiva de Python permite a los desarrolladores escribir código de manera más eficiente y comprensible. Esto es especialmente valioso en IA, donde los modelos y algoritmos pueden ser complejos. La legibilidad del código en Python facilita la colaboración entre equipos de desarrollo y mejora la comunicación entre los miembros del equipo que trabajan en proyectos de IA.

Como consecuencia, Python se ha consolidado como el lenguaje de primera elección [12] en el desarrollo de redes neuronales debido a su comunidad activa, su ecosistema de bibliotecas especializadas, su naturaleza de código abierto y su facilidad de uso. Estas características lo convierten en una herramienta eficaz y versátil para los desarrolladores que buscan explorar y aplicar técnicas de inteligencia artificial y aprendizaje automático en una amplia variedad de aplicaciones.

Por otro lado, el potencial de las tecnologías en la nube, como Kubeflow y Vertex AI, en el campo de la inteligencia artificial y las redes neuronales ha complementado la posición de Python como el lenguaje de primera elección. Kubeflow es una plataforma de aprendizaje automático de código abierto que aprovecha la escalabilidad y flexibilidad de Kubernetes para facilitar la implementación, el despliegue y la gestión de modelos de IA a gran escala en entornos en la nube. Al integrar Python con Kubeflow, los desarrolladores pueden llevar sus modelos entrenados a la nube, lo que les permite aprovechar los recursos de cómputo distribuido y acelerar el tiempo de entrenamiento en grandes conjuntos de datos.

Vertex AI, por su parte, ofrece una solución integral de aprendizaje automático en la nube de Google Cloud. Esta plataforma simplifica el desarrollo de modelos de IA, ofreciendo herramientas de automatización, entrenamiento distribuido y despliegue eficiente en producción. El servicio se integra perfectamente con Python y proporciona una experiencia unificada para desarrolladores, lo que facilita el ciclo de vida completo de los modelos de redes neuronales.

La naturaleza distribuida de Kubeflow y las capacidades integradas de Vertex AI abordan uno de los desafíos clave en el desarrollo de redes neuronales: la capacidad computacional. El entrenamiento de modelos profundos a menudo requiere una enorme capacidad de cómputo, y estas tecnologías en la nube proporcionan soluciones para ejecutar tareas de aprendizaje automático en clústeres de servidores, lo que permite paralelizar el proceso y reducir drásticamente el tiempo de entrenamiento.

Entre las capacidades de estos cabe destacar una característica clave, y es el uso de imágenes virtuales para ejecutar los entrenamientos, siendo esta una forma intuitiva y declarativa de definir flujos de trabajo para el entrenamiento de modelos de IA. Esto permite a los desarrolladores diseñar y organizar de manera eficiente tareas complejas, desde la adquisición de datos hasta la evaluación del modelo, lo que mejora la eficiencia y reproducibilidad de los experimentos. Con la combinación de Python y estas tecnologías en la nube, los desarrolladores pueden aprovechar el poder de la inteligencia artificial y las redes neuronales para abordar problemas complejos y transformar la forma en que interactuamos con la tecnología en el mundo moderno.

2.3. El impacto de los recursos didácticos en este campo

La disponibilidad de recursos introductorios es crucial para la popularización y democratización de estas tecnologías en el campo de la inteligencia artificial y el aprendizaje automático. Estos recursos permiten a los principiantes adentrarse en el mundo de la programación y el aprendizaje automático de manera más accesible y amigable. Al tener guías paso a paso, tutoriales interactivos y documentación detallada, los principiantes pueden superar las barreras iniciales y adquirir rápidamente habilidades fundamentales.

Los recursos introductorios también desempeñan un papel importante en la promoción de una comunidad de aprendizaje y colaboración en torno a Python y los entornos cloud mencionados anteriormente. Al proporcionar una plataforma para que los principiantes compartan conocimientos, resuelvan problemas y trabajen en proyectos prácticos, estos recursos fomentan un ambiente en el que los usuarios pueden apoyarse mutuamente y aprender de diferentes perspectivas. Esto contribuye a la creación de una comunidad activa y diversa que impulsa el desarrollo y la innovación en el campo.

Un TFG enfocado en analizar la situación actual, explicando el contexto reciente de las redes neuronales y su desarrollo en Python y en entornos en la nube puede pues, ser una potencial contribución para esta comunidad. Al seguir un enfoque evolutivo e incremental, este trabajo permite ir avanzando por pasos en cómo estas soluciones pueden ser escaladas para su uso en la resolución de problemas del mundo real y abordar desafíos complejos en diferentes industrias. Por otro lado, también puede fomentar la adopción más amplia de estas tecnologías al hacer que sean más comprensibles y accesibles para un público más amplio.

Por último, con la creación de estos trabajos también se puede contribuir a la difusión del conocimiento sobre inteligencia artificial y aprendizaje automático en general. Al mostrar cómo estas tecnologías pueden ser utilizadas para resolver problemas en diversas áreas del mundo real, se puede generar un mayor interés en el campo y alentar a más personas a dedicarse a la investigación y el desarrollo en IA.

3. Redes neuronales

Hablar de “redes neuronales”, “IA” y todo ese bagaje de conceptos suena muy bien, pero realmente, ¿qué es lo que se hace realmente al implementar en la realidad su uso?, ¿en qué se basará su puesta en funcionamiento?, ¿por qué será efectivo? En este capítulo se analizan los conceptos básicos de forma que, cuando se pase a la parte práctica, se cuente con un enfoque conceptual sólido de cómo funciona una red neuronal.

Cabe mencionar de antemano, que las redes neuronales no son artificios capaces de funcionar de forma “mágica”, necesitan ser entrenadas. Cuando se crea una red, esta no tiene conocimiento sobre los datos ni la tarea que debe realizar. Para enseñar a la red cómo realizar la tarea, es necesario facilitarle un conjunto de datos de entrenamiento que consiste en ejemplos de entrada y las salidas deseadas para estos ejemplos. Durante el proceso de entrenamiento, la red se reajusta utilizando algoritmos de optimización, que buscan minimizar la diferencia entre las predicciones y las salidas deseadas, sea cuál sea la tarea a realizar.

A lo largo del entrenamiento, aparecen diversos componentes que influyen de una forma u otra en el funcionamiento de una red, veamos cuáles son.

3.1. ¿Qué es una neurona?

Una neurona artificial es la unidad básica de procesamiento en una red neuronal, y su funcionamiento está inspirado en las neuronas biológicas presentes en el cerebro humano.

Cómo idea general, se puede imaginar que una neurona en una red neuronal artificial es una unidad de procesamiento que tomará “señales de entrada”, las combinará ponderadamente - mediante unos pesos establecidos, que indican la importancia de la información que se transmite a través de cada conexión - y las procesará mediante una función que llamaremos de “activación”, produciendo una salida. Un ejemplo concreto de esta operación neuronal es el Perceptrón, en este modelo de neurona artificial, las señales de entrada se multiplican por sus pesos correspondientes y se suman. Luego, esta suma se pasa a través de una función de activación, típicamente una función escalón unitario, que produce una salida binaria basada en si la suma ponderada excede un cierto umbral.

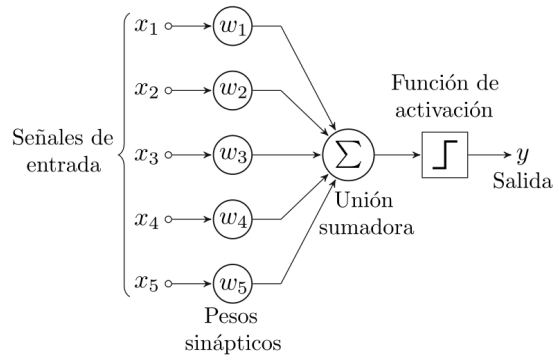


Figura 1. Diagrama de un Perceptrón con cinco señales de entrada (fuente: Alejandro Cartas, Wikipedia)

El funcionamiento descrito es conjunto entre diversas neuronas, trabajando en grupo mediante el uso de ciertas funciones que permiten realizar tareas de aprendizaje automático, como

reconocimiento de patrones, clasificación de datos y toma de decisiones, entre otros. Para que este funcionamiento se dé a cabo, las neuronas se organizan en capas dentro de una red neuronal.

En una red neuronal típica, tenemos una capa de entrada, una o varias capas ocultas y una capa de salida. Cada neurona en una capa, está conectada a todas las neuronas de la capa anterior y de la capa siguiente.

Las diversas configuraciones o estructuras específicas de las neuronas, es decir, su disposición o arquitectura, se conocen cómo “modelo” de red neuronal y su elección dota a la red de capacidades diversas con resultados variables, siendo menor o mayormente efectiva en las tareas que se le puedan encomendar.

3.2. Parámetros

Los parámetros en una red neuronal son los valores que se ajustan durante el entrenamiento del modelo para que se adapte a los datos de entrada. Los parámetros más comunes son los pesos y los sesgos. Los pesos determinan la influencia de cada entrada en la salida de cada neurona, mientras que los sesgos son valores constantes que se suman a la combinación lineal de las entradas y los pesos. Ajustar correctamente los parámetros es esencial para que la red neuronal aprenda a realizar predicciones precisas.

3.2.1. Función de activación

La función de activación es un componente crucial en las redes neuronales, pues determina su comportamiento en gran medida. Las funciones de activación se pueden imaginar como interruptores que determinan si una neurona "dispara" o no, es decir, si produce una salida activada o no activada. Estas funciones son aplicadas al resultado de la combinación ponderada de las entradas y los pesos de una neurona.

Una de las funciones de activación más utilizadas es la función sigmoide. La función sigmoide toma un valor real como entrada y lo transforma en un valor entre 0 y 1. Tiene una forma de curva en forma de "S" y es especialmente útil en problemas de clasificación binaria, donde se desea asignar una etiqueta de clase 0 o 1 a una entrada.

Otra función de activación común es la función ReLU (Rectified Linear Unit). Esta función es muy simple y efectiva. Toma un valor de entrada y si es mayor que cero, lo deja sin cambios; si es menor o igual a cero, lo convierte en cero. La función ReLU ha demostrado ser efectiva en la mayoría de los casos, ya que no sufre de gradientes que desaparezcan o exploten durante el entrenamiento (más adelante se analizará el tema de los gradientes).

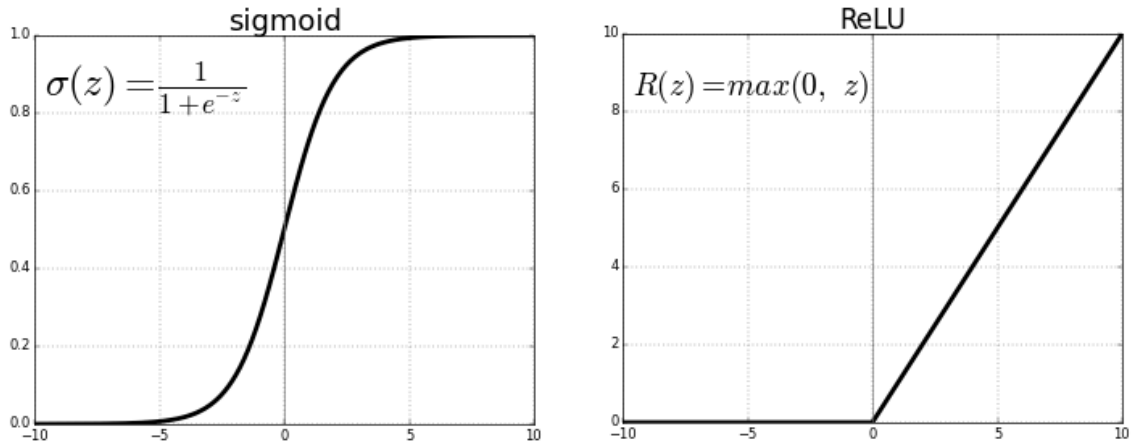


Figura 2: Funciones Sigmoide y ReLU (Fuente: adaptado de "ReLU vs. Sigmoid Function in Deep Neural Networks" Ayush Thakur et al.)

Además de estas funciones, existen otras variantes y funciones de activación específicas para diferentes necesidades. Algunas de estas incluyen la función softmax, que es útil en problemas de clasificación multiclase; la función Leaky ReLU, que permite un rango negativo no cero; y la función ELU (Exponential Linear Unit), que tiene una mejor respuesta a valores negativos.

Hay que tener en cuenta que, la función de activación es esencial para agregar no linealidad en las redes neuronales. La elección de la función de activación depende del problema y la arquitectura de la red neuronal, y cada función tiene sus propias características y propiedades matemáticas. Al experimentar con diferentes funciones de activación, los desarrolladores pueden mejorar el rendimiento y la capacidad de aprendizaje de las redes neuronales.

3.2.2. Sobre la no linealidad

Ahora bien, ¿qué es esto de la no linealidad y por qué es importante? La no linealidad es relevante en las redes neuronales porque permite a los modelos aprender y representar relaciones y patrones complejos en los datos. Sin esta característica, las redes neuronales estarían limitadas a realizar solamente transformaciones lineales, lo que las haría ineficientes en la mayoría de los problemas del mundo real.

La mayoría de los datos en el mundo real tienen una naturaleza no lineal, lo que significa que las relaciones y las interacciones entre las variables no pueden describirse simplemente mediante una función lineal. Las funciones de activación no lineales en las redes neuronales permiten modelar y capturar estas relaciones no lineales.

Al introducir no linealidad en las redes neuronales, las funciones de activación permiten a los modelos aprender características y representaciones más complejas y expresivas. Esto es especialmente importante en tareas como la visión por computador, el procesamiento del lenguaje natural, el reconocimiento de voz y muchas otras áreas donde las relaciones de los datos son inherentemente no lineales.

Debido a este mismo hecho, también juega un papel clave en la fase de entrenamiento ya que los modelos ajustan los pesos y los sesgos para minimizar la función de pérdida y hacer predicciones más precisas. Si las funciones de activación fueran lineales, el proceso de entrenamiento se reduciría a un simple ajuste de parámetros lineales, limitando la capacidad de aprendizaje de la red neuronal. La no linealidad pues, es una de las características clave que distingue a las redes neuronales de un simple modelo lineal y les permite resolver una amplia gama de problemas de aprendizaje automático de manera más efectiva.

Además, esta característica ayuda a evitar el problema del desvanecimiento del gradiente, que ocurre cuando los gradientes de error se vuelven muy pequeños a medida que se propagan hacia capas anteriores en una red neuronal profunda. Más adelante se explicará que es el descenso por gradiente y por qué es relevante.

3.2.3.Épocas, lotes y pérdida

En la fase de entrenamiento de una red neuronal, una época se refiere a una pasada completa del conjunto de entrenamiento por la red neuronal. Es decir, durante una época, todos los ejemplos del conjunto de entrenamiento se utilizan para ajustar los pesos y sesgos del modelo con el objetivo de mejorar la precisión de las predicciones.

El entrenamiento de la red generalmente se realiza a través de varias épocas, y la cantidad de épocas es un hiperparámetro que se puede ajustar durante el proceso de entrenamiento. Elegir el número adecuado de épocas es crucial, ya que un número insuficiente de épocas puede llevar a que el modelo no haya aprendido lo suficiente (sobregeneralización, *underfit*), mientras que un número excesivo de épocas puede llevar a un sobreajuste, donde el modelo se ajusta demasiado a los datos de entrenamiento y no generaliza bien a nuevos datos (sobreajuste, *overfit*).

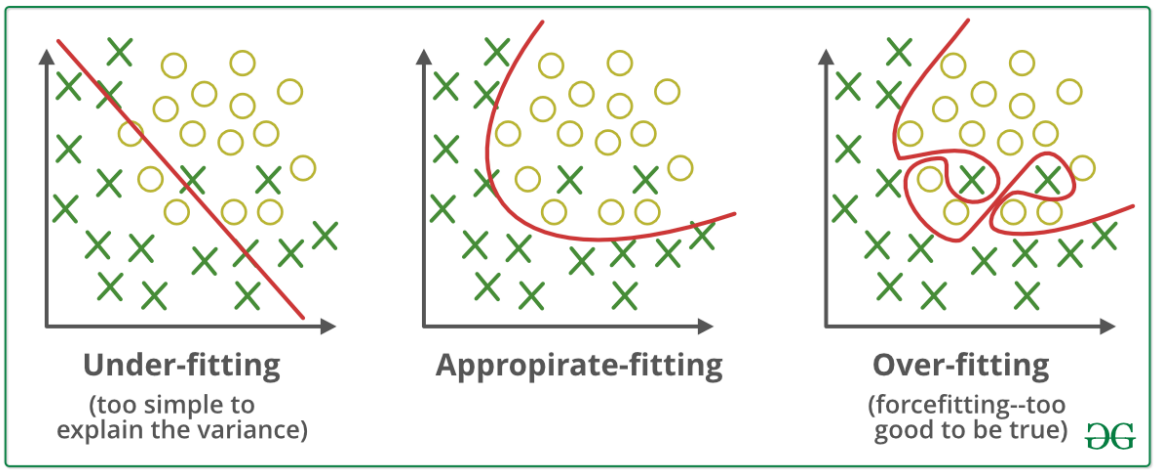


Figura 3: Fronteras de decisión de un clasificador en diferentes situaciones (fuente: GeeksForGeeks)

Por otro lado, es común realizar el entrenamiento de una red neuronal en lotes (*batches*) en vez de procesar todos los ejemplos de una sola vez. Un lote es un subconjunto de ejemplos del conjunto de entrenamiento que se utiliza para ajustar los parámetros del modelo antes de hacer una

actualización global. En otras palabras, la red neuronal procesa un lote a la vez y ajusta sus pesos y sesgos basándose en el error acumulado en ese lote.

El número de lotes utilizados en cada época se conoce como el tamaño del lote (batch size). Un tamaño de lote más grande puede acelerar el proceso de entrenamiento al procesar más ejemplos a la vez, pero también puede requerir más memoria. Por otro lado, un tamaño de lote más pequeño puede requerir más tiempo de entrenamiento debido a actualizaciones de peso más frecuentes, pero puede permitir una mejor generalización del modelo.

El uso de lotes permite aprovechar la eficiencia computacional al realizar cálculos en paralelo en lugar de manera secuencial. Esto es especialmente beneficioso cuando se trabaja con conjuntos de datos grandes, ya que se pueden realizar actualizaciones de parámetros más frecuentes y acelerar el proceso de entrenamiento. Es importante encontrar un equilibrio adecuado entre el tamaño del batch y el rendimiento del modelo. Generalmente, se realizan experimentos utilizando diferentes tamaños de lote para determinar el tamaño óptimo que maximice el rendimiento del modelo, sin sacrificar la eficiencia computacional.

Un aspecto importante al trabajar con épocas y tamaños de lote es el seguimiento del progreso del modelo durante el entrenamiento. Es común utilizar conjuntos de validación o prueba para evaluar el rendimiento del modelo en datos no vistos durante cada época. Esto permite detectar si el modelo está mejorando o si se está sobreajustando a los datos de entrenamiento, lo que guía la toma de decisiones sobre cuántas épocas son necesarias o si se deben ajustar otros hiperparámetros.

En general, el número óptimo de épocas depende del conjunto de datos y la complejidad del problema. En algunos casos, es posible que un modelo alcance un buen rendimiento con solo unas pocas épocas, mientras que en otros casos puede requerir más iteraciones para converger hacia una solución óptima.

Otro de los hiperparámetros de una red dentro de la optimización, es el conocido como "pérdida" (*loss*), que mide la discrepancia entre las predicciones realizadas por el modelo y los valores reales de los datos de entrenamiento. En otras palabras, la pérdida representa cuánto se equivoca el modelo en sus predicciones.

Una estrategia común para determinar el número adecuado de épocas es, precisamente, monitorear la función de pérdida y la precisión del modelo en un conjunto de validación separado durante el entrenamiento. Se puede detener el entrenamiento cuando se observa una disminución en el rendimiento en el conjunto de validación, lo que indica que el modelo ha alcanzado su capacidad máxima de aprendizaje.

En cada época, el modelo ajusta sus parámetros para minimizar la pérdida, lo que implica encontrar una mejor representación de los datos y capturar relaciones más precisas en los datos de entrenamiento. El proceso de ajuste se repite a lo largo de varias épocas hasta que el modelo mejora sus predicciones y logra una pérdida aceptable. Es importante recordar que el objetivo principal es que el modelo generalice correctamente a datos no vistos y sea capaz de hacer predicciones precisas en situaciones reales.

3.3. Capas en una red neuronal y capas totalmente conectadas

En una red neuronal, las capas juegan un papel fundamental en el proceso de procesamiento de información y toma de decisiones. Como se ha mencionado previamente, una capa es simplemente un conjunto de neuronas. Estas capas se organizan en una estructura jerárquica para procesar y transformar los datos de entrada a través de una serie de operaciones.

Una red neuronal típica consiste en tres tipos de capas principales: la capa de entrada, una o varias capas ocultas y la capa de salida. La capa de entrada es la primera capa de la red y recibe los datos de entrada sin procesar. Cada neurona en la capa de entrada representa una característica o atributo de los datos, y las neuronas se activan según los valores de entrada que reciben.

Las capas ocultas son las capas intermedias entre la capa de entrada y la capa de salida. Estas capas son responsables de extraer características más abstractas y complejas de los datos de entrada a medida que la información se propaga a través de la red. El número y el tamaño de las capas ocultas son factores importantes que afectan la capacidad de la red para aprender patrones y representaciones más profundas.

La capa de salida es la última capa de la red y produce las predicciones o resultados finales. Cada neurona en la capa de salida representa una clase o valor objetivo que el modelo debe predecir. Dependiendo del tipo de problema (clasificación, regresión, etc.), la cantidad de neuronas en la capa de salida variará.

Las capas totalmente conectadas, también conocidas como capas densas, son un tipo común de capa en una red neuronal. En una capa totalmente conectada, cada neurona está conectada a todas las neuronas de la capa anterior y de la capa siguiente, formando una matriz de conexiones entre capas. Esta interconexión total permite que las capas totalmente conectadas tengan una gran flexibilidad para aprender patrones complejos en los datos. Cada neurona en una capa totalmente conectada toma las salidas de todas las neuronas de la capa anterior, ponderadas por sus respectivos pesos, y produce una salida mediante la función de activación.

Las capas totalmente conectadas son altamente expresivas y pueden modelar relaciones no lineales en los datos, lo que les permite abordar una amplia gama de problemas de aprendizaje automático. Sin embargo, la cantidad de conexiones aumenta rápidamente con el tamaño de las capas, lo que puede resultar en una mayor cantidad de parámetros y requerir más recursos computacionales, esto puede llevar al sobreajuste y a un rendimiento subóptimo en conjuntos de datos pequeños. Es común utilizar una combinación de capas totalmente conectadas con otros tipos de capas para aprovechar mejor las características específicas de los datos y mejorar el rendimiento del modelo.

3.4. Descenso por gradiente

El descenso por gradiente es un algoritmo de optimización utilizado para minimizar una función de pérdida en el proceso de entrenamiento de una red neuronal. Previamente se ha mencionado que la función de pérdida mide cuánto se equivoca el modelo en sus predicciones en comparación con los valores reales del conjunto de entrenamiento. El objetivo es encontrar los

valores óptimos para los pesos de la red que minimicen la pérdida global y, por lo tanto, hagan que el modelo haga predicciones más precisas.

El descenso por gradiente comienza con valores iniciales para los pesos de la red y se realiza iterativamente en cada lote de datos del conjunto de entrenamiento. En cada iteración, el algoritmo calcula los gradientes de la función de pérdida con respecto a los pesos, lo que indica la dirección y la magnitud del cambio necesario para reducir la pérdida. Luego, los pesos se ajustan en la dirección opuesta al gradiente para disminuir la pérdida.

Es importante destacar que el descenso por gradiente puede tener diferentes variantes, como el descenso por gradiente estocástico (SGD) o el descenso por gradiente con momento, cada uno con sus ventajas y desventajas. Estas variantes permiten mejorar la velocidad de convergencia y superar obstáculos en la optimización, lo que es esencial para redes neuronales profundas y complejas.

3.5. Retropropagación (*Backpropagation*)

Uno de los algoritmos que se basa en el descenso por gradiente para ajustar los pesos de una red neuronal durante el entrenamiento es la retropropagación, o *backpropagation* en inglés. Es una técnica eficiente y con mucho potencial para calcular los gradientes de la función de pérdida con respecto a los pesos de la red, lo que permite determinar cómo los pesos influyen en la pérdida y, en última instancia, en las predicciones del modelo.

El proceso de backpropagation se realiza en dos fases: propagación hacia adelante y propagación hacia atrás. Durante la propagación hacia adelante, los datos de entrada se introducen en la red, y las salidas se calculan en cada capa a través de la combinación ponderada de las entradas y los pesos, seguido de la aplicación de la función de activación. Así, se obtienen las predicciones del modelo.

Luego, se compara la salida predicha con los valores reales del conjunto de entrenamiento, lo que resulta en la función de pérdida. En la fase de propagación hacia atrás, los gradientes de la función de pérdida con respecto a los pesos se calculan en sentido inverso, comenzando desde la capa de salida hacia la capa de entrada.

Estos gradientes indican cómo cambiar los pesos para reducir la pérdida, y se utilizan para ajustar los pesos de la red mediante el descenso por gradiente. El proceso de backpropagation se repite en múltiples épocas de entrenamiento, lo que permite que la red mejore progresivamente su rendimiento y se adapte mejor a los datos de entrenamiento.

La mención de backpropagation en los conceptos básicos no es casual, el avance del campo del aprendizaje automático ha sido significativamente impactado por la introducción y aplicación de este algoritmo. Antes de la aparición de este método en la década de 1980, el entrenamiento de redes neuronales era una tarea desafiante y poco efectiva, lo que llevó a un declive en el interés y la investigación en este campo.

Antiguamente, las redes neuronales eran entrenadas mediante métodos más simples, como el aprendizaje hebbiano, el aprendizaje competitivo y el aprendizaje por error de Widrow-Hoff [2].

Estos métodos tienen limitaciones en términos de eficiencia y capacidad de aprendizaje, lo que dificultaba el entrenamiento de redes más grandes y complejas.

Con la introducción de este algoritmo, el entrenamiento de redes neuronales se volvió mucho más eficiente y efectivo. Este algoritmo permitió ajustar los pesos de manera sistemática y automática, esto significó que las redes neuronales podían aprender a partir de datos etiquetados, lo que abrió la puerta a un enfoque más formal y estructurado para el entrenamiento.

Su aplicación en redes neuronales profundas, también conocidas como deep learning, ha sido especialmente impactante en el campo del aprendizaje automático. Antes de la popularización de las redes neuronales profundas, la mayoría de los modelos de aprendizaje automático eran más superficiales y menos capaces de aprender representaciones jerárquicas y complejas de los datos.

Se podría decir que *Backpropagation* ha sido un catalizador clave para el avance del campo del aprendizaje automático y en cierto modo, fundamental para el resurgimiento y la revolución del aprendizaje profundo, creando una nueva era en el aprendizaje automático. [3]

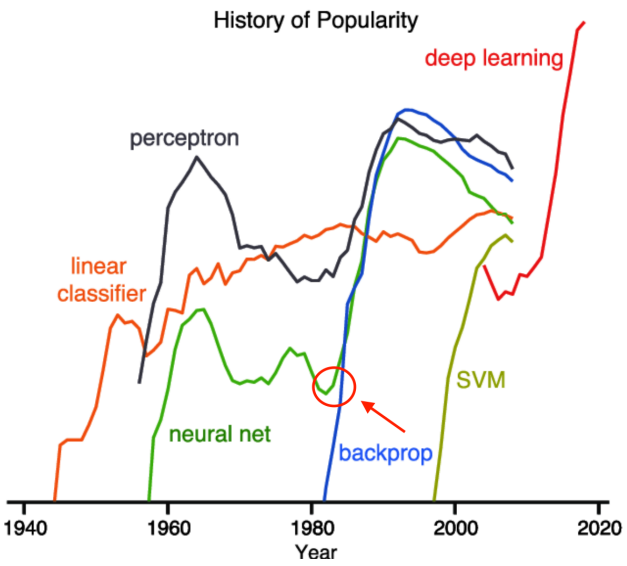


Figura 4: Popularidad de las redes neuronales en el período 1940-2020 (Fuente: "Deep learning-Using machine learning to study biological vision", Najib J Majaj, Denis G. Pelli et al.)

3.6. El ciclo de vida de una red neuronal

Otro aspecto relevante de las redes neuronales es que tienen un ciclo de vida que implica una serie de pasos a seguir para poder lograr una red neuronal eficiente y completamente operativa.

3.6.1 Recopilación de datos

La recopilación de datos es el punto de partida en el ciclo de vida de una red neuronal. Antes de que cualquier entrenamiento o modelado pueda ocurrir, es esencial obtener un conjunto de datos relevante y representativo que refleje el problema que se quiere resolver. Esto implica identificar fuentes de datos, que pueden variar desde bases de datos existentes hasta sensores, registros de usuarios o datos de redes sociales. La calidad de los datos es crítica, por lo que es necesario abordar problemas de integridad, precisión y completitud de los datos desde el principio.

Además de la calidad de los datos, es importante considerar la cantidad necesaria de datos para entrenar un modelo eficaz. En algunos casos, puede ser necesario recopilar grandes volúmenes de datos para garantizar que el modelo tenga suficiente información para generalizar correctamente. También es importante evaluar si los datos son representativos de todas las posibles situaciones o escenarios que el modelo puede encontrar en el mundo real. La recopilación de datos no es un proceso estático; a medida que el proyecto avanza, es posible que sea necesario adquirir datos adicionales o ajustar la estrategia de recopilación para abordar desafíos específicos que surjan durante el desarrollo del modelo. En última instancia, la calidad y cantidad de los datos recopilados desempeñan un papel crucial en la efectividad y la capacidad de generalización de la red neuronal.

La recopilación de datos también plantea consideraciones éticas y legales, especialmente si los datos incluyen información personal o sensible. Se deben seguir las regulaciones y políticas de privacidad correspondientes, y es importante garantizar que los datos se utilicen de manera responsable y se protejan adecuadamente. Debido a estas razones, la fase de recopilación de datos sienta las bases para todo el proyecto de aprendizaje profundo, y una planificación cuidadosa en esta etapa es esencial para el éxito del modelo.

3.6.2 Preprocesamiento de datos

El preprocesamiento de datos es la fase de preparación de los datos para su uso en el entrenamiento de la red neuronal. Implica una serie de pasos que aseguran que los datos sean adecuados y útiles para el modelo. Uno de los primeros pasos es la limpieza de datos, que implica la detección y eliminación de valores atípicos, datos faltantes o errores obvios que puedan afectar negativamente el rendimiento del modelo. Además, es común aplicar técnicas de normalización o escalado para llevar todas las características a una escala similar, lo que ayuda a acelerar el proceso de entrenamiento y evita que características con escalas grandes dominen las contribuciones.

Otro aspecto importante del preprocesamiento de datos es el manejo de desequilibrios en los datos. En muchos conjuntos de datos, puede haber una distribución desigual entre las clases o categorías que se están prediciendo. Esto puede llevar a un sesgo en el modelo, donde se presta más atención a las clases dominantes en lugar de a las clases minoritarias. Para abordar este problema, es posible aplicar técnicas de remuestreo, como el sobremuestreo de las clases minoritarias o el submuestreo de las clases dominantes, para equilibrar la distribución de clases y mejorar la capacidad de predicción en todas las categorías.

En el preprocesamiento de datos también se pueden aplicar técnicas de reducción de dimensionalidad, como el análisis de componentes principales (PCA) [4] o la selección de características, para reducir la complejidad del conjunto de datos mientras se conserva la información relevante. Esto puede ser útil en casos de conjuntos de datos de alta dimensionalidad para mejorar la eficiencia del entrenamiento y reducir la posibilidad de sobreajuste.

El procesamiento de características también es un componente clave del pre-procesamiento de datos. Esto incluye la ingeniería de características, donde se crean nuevas características a partir de las existentes para resaltar información relevante, y la codificación de características categóricas en formatos numéricos. Además, se suele dividir el conjunto de datos en conjuntos de entrenamiento, validación y prueba, para evaluar el rendimiento del modelo de manera imparcial.

Considerando estas razones, se puede decir que el preprocesamiento de datos no solo se trata de limpiar y formatear datos, sino también de abordar desafíos específicos en los datos para garantizar que la red neuronal pueda aprender de manera efectiva y generar predicciones precisas. Esta etapa es crítica, ya que asegura que los datos estén en la forma adecuada antes de que la red neuronal inicie su entrenamiento, lo que contribuye significativamente al éxito y la capacidad de generalización del modelo.

3.6.3 Diseño de la red

El diseño de la red neuronal comienza con la elección de la arquitectura de la red, que puede variar desde redes neuronales feedforward simples hasta redes neuronales convolucionales (CNN) para tareas de visión por computadora o redes neuronales recurrentes (RNN) para procesamiento de secuencias de datos. La elección de la arquitectura depende en gran medida de la naturaleza del problema que se está abordando y de la estructura de los datos. Por ejemplo, para tareas de clasificación de imágenes, las CNN suelen ser la elección preferida debido a su capacidad para capturar patrones espaciales.

El diseño de la red es una etapa que requiere un enfoque cuidadoso y una comprensión profunda del problema y de las características de los datos para crear una arquitectura eficaz y eficiente.

Además de la arquitectura, se deben determinar otros aspectos clave, como el número de capas ocultas y la cantidad de neuronas en cada capa. Estos parámetros influyen en la capacidad de representación del modelo y en su complejidad. Los investigadores deben encontrar un equilibrio entre un modelo lo suficientemente complejo como para capturar patrones importantes y lo suficientemente simple como para evitar el sobreajuste. También se deben seleccionar las funciones de activación adecuadas para cada capa y configurar hiperparámetros como la tasa de aprendizaje. El diseño de la red se basa en una comprensión profunda del problema y en la iteración y experimentación para encontrar la configuración óptima para la tarea específica.

Otro aspecto a considerar, es la inicialización de los pesos, la elección de cómo se inicializan los pesos puede afectar significativamente el proceso de entrenamiento y el rendimiento final del modelo. Las técnicas de inicialización como Xavier (*Glorot*) [5] o He se utilizan comúnmente para establecer pesos iniciales que facilitan la convergencia durante el entrenamiento.

3.6.4 Entrenamiento de la red

El entrenamiento de la red neuronal es un proceso donde el modelo aprende a partir de los datos para realizar la tarea deseada. Durante el entrenamiento, los datos de entrenamiento alimentan a la red, y los pesos y parámetros de la red se ajustan iterativamente para minimizar una función de pérdida. La optimización se realiza mediante los anteriormente mencionados algoritmos de descenso de gradiente, como el descenso de gradiente estocástico (SGD) o sus variantes, que ajustan los pesos de la red en dirección a la minimización de la pérdida.

Durante el proceso, es crucial monitorizar el rendimiento en un conjunto de validación para evitar el sobreajuste. Si el modelo se desempeña bien en el conjunto de entrenamiento pero no en el de validación, podría ser una señal de que está memorizando en lugar de generalizar. El ajuste de hiperparámetros como la tasa de aprendizaje y la cantidad de épocas de entrenamiento puede ser necesario para encontrar el equilibrio adecuado entre rendimiento en el entrenamiento y generalización.

La elección de la función de pérdida es otra consideración importante en el entrenamiento de la red. Dependiendo del tipo de tarea (clasificación, regresión, etc.), se utilizan diferentes funciones de pérdida, como la entropía cruzada para clasificación o el error cuadrático medio para regresión. Se ha de tener en cuenta que el entrenamiento de la red puede llevar tiempo, especialmente en conjuntos de datos grandes y modelos complejos, es por ello que requiere una cuidadosa atención a la configuración para garantizar un rendimiento óptimo, siendo común el empleo de hardware acelerado, como GPUs o TPUs, para agilizar el proceso.

3.6.5 Tunelado de hiperparámetros

El tunelado de hiperparámetros es la etapa relacionada con optimizar el rendimiento del modelo y encontrar la configuración más adecuada. Hemos de recordar que los hiperparámetros son parámetros que no se aprenden durante el entrenamiento, pero que influyen en la forma en que la red neuronal aprende y generaliza. El objetivo del tunelado de hiperparámetros es encontrar los valores óptimos de estos parámetros que mejoren la precisión y la capacidad de generalización del modelo.

Para realizar el tunelado de hiperparámetros de manera efectiva, se utilizan técnicas como la búsqueda en cuadrícula (grid search) o la optimización bayesiana. En la búsqueda en cuadrícula, se especifica un conjunto de valores posibles para cada hiperparámetro, y se entrena y evalúa el modelo para todas las combinaciones posibles de valores. La optimización bayesiana, por otro lado, utiliza un enfoque probabilístico para seleccionar las configuraciones de hiperparámetros más prometedoras, lo que puede ser más eficiente que la búsqueda en cuadrícula. Un detalle importante es llevar a cabo el tunelado de hiperparámetros en un conjunto de validación independiente para evitar el sobreajuste a los datos de entrenamiento.

Se ha de tener en cuenta que el tunelado de hiperparámetros es un proceso iterativo y puede llevar tiempo, pero puede marcar una diferencia significativa en el rendimiento del modelo. Una vez que se ha completado el tunelado, se obtiene una configuración óptima de hiperparámetros que se

utiliza para entrenar el modelo final en un conjunto de datos más grande. Esta fase es crucial para asegurarse de que el modelo esté afinado y listo para su implementación en producción.

3.6.6 Evaluación del modelo

La evaluación del modelo nos permite medir el rendimiento y la capacidad de generalización del modelo. Durante esta etapa, se utiliza un conjunto de datos independiente, conocido como conjunto de prueba, que no se ha utilizado previamente en el entrenamiento ni en la optimización de hiperparámetros. El objetivo es obtener una evaluación imparcial del rendimiento del modelo en condiciones del mundo real.

Existen diversas métricas de evaluación que dependen del tipo de problema que se esté abordando. Para tareas de clasificación, se suelen utilizar métricas como la precisión, la sensibilidad, la especificidad, la F1-score y la matriz de confusión, entre otras. En problemas de regresión, métricas como el error cuadrático medio (MSE) o el coeficiente de determinación (R-cuadrado) son comunes. La elección de las métricas adecuadas es esencial para comprender cómo el modelo se desempeña en la tarea específica y si cumple con los requisitos del problema.

La evaluación no se limita a métricas numéricas, sino que también puede incluir análisis de visualización de resultados, como gráficos de curvas ROC o de precisión-recall, para comprender mejor el rendimiento del modelo en diferentes umbrales de decisión. Además, es importante tener en cuenta el contexto del problema y las implicaciones prácticas de los resultados. La evaluación del modelo proporciona información valiosa para tomar decisiones sobre la implementación en producción y para identificar áreas de mejora y ajuste en el modelo. Por ello, la fase de evaluación del modelo se considera esencial para determinar si el modelo cumple con los estándares de rendimiento y generalización requeridos y si está listo para enfrentar desafíos del mundo real.

3.6.7 Ajuste y optimización en producción

Una vez que un modelo de red neuronal ha pasado por las etapas de entrenamiento, validación y evaluación, es necesario llevar a cabo el ajuste y la optimización en el entorno de producción para asegurar un rendimiento óptimo en el mundo real. El entorno de producción puede presentar desafíos diferentes a los del entorno de desarrollo y pruebas, por lo que es importante que el modelo esté listo para enfrentarlos. Esto implica una serie de tareas y consideraciones específicas.

En primer lugar, el ajuste y la optimización en producción pueden incluir la reevaluación de hiperparámetros y configuraciones del modelo en función de las demandas del entorno en tiempo real. Por ejemplo, se pueden ajustar los umbrales de decisión o las tasas de aprendizaje para adaptarse a cambios en la distribución de datos o en los patrones de entrada. Además, se debe considerar la eficiencia computacional y la escalabilidad para garantizar que el modelo pueda manejar la carga de trabajo esperada sin problemas de rendimiento.

La monitorización continua del modelo en producción implica el seguimiento constante del rendimiento del modelo, la detección de posibles degradaciones o cambios en la precisión y la identificación de problemas de deriva de datos que puedan afectar el rendimiento. Se pueden implementar estrategias de reentrenamiento automático o semiautomático para mantener el modelo actualizado y adaptado a las condiciones cambiantes. El ajuste y la optimización en producción son pasos críticos para garantizar que el modelo de red neuronal siga siendo efectivo y eficiente en el entorno del mundo real y que continúe proporcionando resultados precisos y útiles a lo largo del tiempo.

3.6.8 Monitorización y mantenimiento

La monitorización y el mantenimiento son especialmente útiles cuando el modelo se encuentra en producción. La monitorización implica el seguimiento constante del rendimiento del modelo en tiempo real. Esto incluye la evaluación de métricas clave, como la precisión, la sensibilidad o el error, para detectar cualquier degradación en el rendimiento. La detección temprana de problemas es crucial para tomar medidas correctivas de manera oportuna.

Otra parte importante en este proceso es la supervisión de los cambios en los datos de entrada. Los modelos de aprendizaje profundo pueden ser sensibles a desviaciones en la distribución de datos, lo que se conoce como deriva de datos. La monitorización de la deriva de datos (sesgo) permite identificar cambios en los patrones de entrada y tomar medidas para adaptar el modelo en consecuencia, ya sea mediante la actualización de los datos de entrenamiento o la recalibración de hiperparámetros.

El mantenimiento de la red neuronal también implica tareas como la actualización de software y bibliotecas, la gestión de recursos computacionales y la optimización de la infraestructura de producción. Además, se debe mantener un proceso de reentrenamiento planificado para asegurarse de que el modelo se mantenga actualizado con nuevos datos y condiciones cambiantes. Esta parte del ciclo de vida se encarga pues, de garantizar que el modelo continúe siendo efectivo, preciso y seguro en su entorno de implementación a lo largo del tiempo y para abordar los desafíos que surgen en el mundo real.

4. Una primera implementación

Una vez analizados los conceptos más importantes que están relacionados con las redes neuronales, se ha sentado el precedente que permite pasar a trabajar en la implementación real en Python.

Python es un lenguaje de programación de alto nivel, interpretado, de propósito general y de código abierto, fue lanzado en 1991. Desde entonces, ha ganado una enorme popularidad debido a su simplicidad, legibilidad y versatilidad. Python es conocido por su sintaxis clara y expresiva, lo que facilita a los desarrolladores escribir código de manera más eficiente y comprensible. Esto ha llevado a que se considere un lenguaje fácil de aprender y usar, especialmente para principiantes en programación.

La elección de Python para esta tarea no es aleatoria ni una mera casualidad, ya que este lenguaje:

- Cuenta con una comunidad activa y colaborativa que ha desarrollado una gran cantidad de bibliotecas y marcos de trabajo específicos para IA y aprendizaje automático.

- Es ideal para el rápido prototipado y desarrollo iterativo de algoritmos de IA. Su sintaxis simple y su facilidad para escribir código permiten a los investigadores probar y modificar rápidamente diferentes enfoques y técnicas sin invertir mucho tiempo en aspectos técnicos de bajo nivel.

- Es conocido por su capacidad para integrarse fácilmente con otras tecnologías y lenguajes de programación. Esto permite utilizar bibliotecas y herramientas desarrolladas en otros lenguajes, como C++ o Java, cuando sea necesario para tareas específicas de IA.

- Es ampliamente usado en entornos didácticos y recursos educativos para aprendizaje automático, lo que ha creado un entorno que facilita el aprendizaje gracias a la riqueza de información y tutoriales accesibles a cualquier interesado en ellos, siendo este trabajo un ejemplo de ello.

Por otro lado, la red neuronal que se implementa está destinada a reconocer dígitos manuscritos del 0-9, siendo el conjunto de datos, el ofrecido por el MNIST. La propuesta consiste en entrenar una red efectiva para este fin, iterando la capacidad de mejorarla, entrenarla y extender su soporte de mantenimiento en un ámbito de trabajo local.

4.1. Configuración del entorno de trabajo

Antes de comenzar a trabajar, es importante configurar adecuadamente el entorno de trabajo y comprender los aspectos básicos de la configuración necesaria. A continuación, se describen una lista de pasos clave para establecer un entorno propicio para el desarrollo de redes neuronales:

Entre las herramientas a usar para establecer un entorno propicio para el desarrollo de redes neuronales necesitaremos Python 3.10.9, Git, pip, los paquetes Python correspondientes (numpy 1.13.9, scikit-learn 0.19.0, scipy 0.19.1, theano 0.7.0). Opcionalmente se puede usar miniconda y crear un entorno específico para poder interactuar con él o cambiar a otro de forma que se puedan tener diferentes versiones de Python y sus paquetes por comodidad y facilidad de implementación. Por último es necesario clonar el repositorio de código sobre el que se basarán los ejemplos [6]

El código fuente proporcionado se ha probado en los siguientes entornos:

- Ubuntu: versiones 20 a 23
- MacOSX Sonoma
- Windows 11

La forma de avanzar en las implementaciones va a ceñirse al siguiente hilo argumental:

1. Se explica la implementación a nivel conceptual.
2. Se explican las ventajas/desventajas y posibles puntos de interés adicionales.

4.2 Implementación básica en Python

Código fuente de referencia: src/network3.py, src/entry3.py

En esta implementación contamos con la clase Network, la cuál se instancia especificando las capas y el tamaño del lote. Adicionalmente, contamos con un método `SGD` que acepta los datos de entrenamiento, prueba y validación, las épocas y el ratio de aprendizaje. Este método implementa el descenso por gradiente estocástico.

De esta implementación podemos observar que mediante las librerías *numpy* y *theano* más la propia lógica del código, se realiza el uso de capas convolucionales `(ConvPoolLayer)`, totalmente conectadas `(FullyConnectedLayer)` y que aplican funciones de activación específicas `(SoftmaxLayer)`.

El cálculo del coste, la precisión y la aceleración por GPU para los cálculos también o bien se implementan o bien se ofrecen a través de estas librerías. Esta red funciona relativamente bien, pues consigue una precisión de un 97.8% en el reconocimiento de dígitos manuscritos.

Desde el punto de vista práctico, se destacan una serie de desventajas:

- No hay soporte para entrenamientos distribuidos, esto es, entrenamientos donde se reparten los esfuerzos de cálculos. Esta es la base para poder escalar el entrenamiento cuándo hablamos de modelos con millones o incluso billones de parámetros.

- No se cuenta con soporte a nivel de desarrollo más allá de las capacidades propias del desarrollador. ¿Se romperá algo cuando actualicemos de versión de Python? ¿Qué garantías hay de que el código está ejecutándose con un alto grado de eficiencia?

- Es necesario implementar cada nuevo tipo de capa, cada nuevo tipo de algoritmo de optimización, etc para poder usarlo. Se crea una fuerte dependencia del conocimiento que se tenga sobre conceptos o detalles de implementación de bajo nivel.

Idealmente se busca poder desarrollar centrando el foco de los esfuerzos en la mejora del modelo y su eficacia, más que en detalles de implementación de bajo nivel. Para resolver este problema, se propone el uso de la librería Tensorflow.

4.3 Implementación mediante el uso de Tensorflow

Código fuente de referencia: src/tf (directorio al completo)

TensorFlow es una de las bibliotecas de código abierto más populares y ampliamente utilizadas para el aprendizaje profundo y el aprendizaje automático. Fue desarrollado por Google Brain y lanzado como un proyecto de código abierto en 2015. Esta librería proporciona un entorno flexible y eficiente para desarrollar y entrenar modelos de redes neuronales, así como para llevar a cabo tareas de inferencia en producción.

Los cambios a destacar principalmente en esta versión, son:

- Delegación de la mayoría de operaciones matemáticas a Tensorflow, en detrimento de numpy o implementaciones hechas a mano.

Por ello se obtienen como principales ventajas:

- **Eficiencia computacional:** TensorFlow está altamente optimizado para la ejecución eficiente de operaciones numéricas en hardware acelerado, como GPUs y TPUs. Esto significa que TensorFlow puede entrenar modelos más rápido que implementaciones personalizadas en NumPy y Theano.

- **Escalabilidad:** TensorFlow puede aprovechar fácilmente la distribución de tareas y la paralelización en sistemas de cómputo distribuido.

- **Comunidad activa:** TensorFlow tiene una comunidad grande y activa, lo que significa que hay una gran cantidad de recursos disponibles, incluyendo documentación, tutoriales y bibliotecas adicionales.

- **Facilidad de implementación en producción:** TensorFlow ofrece herramientas como TensorFlow Serving y TensorFlow Lite que simplifican la implementación de modelos en producción, lo que es crucial para aplicaciones empresariales.

- **Compatibilidad multiplataforma:** TensorFlow es compatible con una variedad de plataformas, lo que permite implementar modelos en una amplia gama de dispositivos, desde servidores hasta dispositivos móviles y dispositivos de borde.

- **Soporte y actualizaciones:** TensorFlow es mantenido por Google, lo que significa que recibe un alto nivel de soporte y actualizaciones regulares. Esto garantiza que la biblioteca esté al día con las últimas técnicas y avances en el campo del aprendizaje profundo.

- **Herramientas de visualización:** TensorFlow incluye herramientas de visualización integradas, como TensorBoard, que facilitan el seguimiento y la visualización del progreso del entrenamiento de modelos.

Sin embargo, se siguen encontrando algunas desventajas importantes, cómo:

- El código sigue teniendo cierta complejidad inherente de todas las operaciones a realizar. Aunque TensorFlow optimiza los cálculos, sigue teniendo ciertos detalles de bajo nivel que pueden resultar tediosos o costosos de mantener o extender.

- El código está acoplado a Tensorflow, lo que implica que se depende del rumbo que decida tomar el equipo de desarrolladores que mantiene esta librería para ampliar la funcionalidad de nuestro modelo.

Ahora que se cuenta con un código con mejor rendimiento y más mantenible, ¿cómo se puede hacer más sencillo y romper esa dependencia de TensorFlow mientras se mantienen sus ventajas? Es en este punto donde se introduce Keras.

4.4 Implementación en Keras

Código fuente de referencia: src/backend/keras.py

Keras es una biblioteca de código abierto de alto nivel diseñada para facilitar y acelerar el desarrollo de modelos de redes neuronales y aprendizaje profundo. Fue desarrollada originalmente por François Chollet y se ha convertido en parte integral de TensorFlow desde TensorFlow 2.0.

Los cambios a destacar principalmente en esta versión son:

- Delegación a Keras de todos los detalles de la implementación de la red.

Por ello se obtiene como principales ventajas:

- **Facilidad de uso:** Keras se centra en la simplicidad y la facilidad de uso. Proporciona una API simple y coherente que permite a los desarrolladores definir, compilar y entrenar modelos de manera intuitiva. Esto hace que sea una excelente opción para aquellos que desean crear modelos de aprendizaje profundo sin preocuparse demasiado por los detalles técnicos.

- **Modularidad**: Keras permite a los desarrolladores construir modelos de manera modular mediante la combinación de capas. Puedes crear redes neuronales convolucionales (CNN), redes neuronales recurrentes (RNN), modelos generativos, y otros tipos de modelos, todo ello utilizando una interfaz coherente y modular.

- **Compatibilidad con múltiples motores de aprendizaje automático**: Keras fue diseñado originalmente para ser agnóstico a la librería que realiza muchos de los detalles de bajo nivel, lo que significa que es compatible con varios motores, incluyendo Theano y TensorFlow. Desde TensorFlow 2.0, Keras se ha integrado directamente en TensorFlow como su API de alto nivel, aunque se sigue pudiendo usar con otros motores diferentes.

- **Extensibilidad:** Keras permite la personalización y extensión de capas y funciones, lo que facilita la creación de modelos personalizados y la experimentación con arquitecturas únicas.

- **Gran comunidad y documentación:** Keras también tiene una comunidad activa de desarrolladores y una amplia documentación, que incluye tutoriales y ejemplos.

- **Portabilidad**: Debido a su modularidad y compatibilidad con múltiples motores, los modelos entrenados en Keras son altamente portables y pueden ser implementados en una variedad de plataformas y entornos, lo que facilita la transición de modelos desde el desarrollo hasta la producción.

Como se puede observar en el código fuente, las funciones de Keras son de un relativo alto nivel, requiriendo unos conocimientos prácticamente conceptuales para poder construir modelos de forma declarativa. Se mantienen pues, las ventajas de TensorFlow en cuánto a rendimiento y además se consigue un marco de trabajo mucho más sencillo, dónde la productividad y el tiempo del investigador se puede centrar en experimentar iterando con diferentes modelos y parámetros sin preocuparse demasiado de lo que ocurre ‘por debajo’.

4.5 Trabajando con Jupyter Notebooks

A la hora de desarrollar una red neuronal, las tecnologías a emplear impactan directamente en el resultado, tanto así sucede con el entorno. Por ello, se realiza una mención especial a Jupyter Notebooks.

Esta herramienta ofrece una serie de ventajas a la hora de trabajar, las cuales enumeramos a continuación:

- **Interactividad**: Es ideal para el desarrollo interactivo. Se pueden ejecutar fragmentos de código de manera incremental y ver los resultados de inmediato en el mismo documento. Esto es especialmente útil para la exploración de datos y la experimentación, ya que permite iterar rápidamente en el análisis y la visualización.

- **Documentación y narrativa**: Los Jupyter Notebooks permiten combinar código, texto, imágenes y visualizaciones en un solo documento. Esto facilita la creación de informes y tutoriales interactivos donde se puede explicar el razonamiento detrás del código, lo que es útil para la comunicación y la documentación.

- **Visualizaciones integradas**: Los gráficos y visualizaciones se pueden generar directamente en el Notebook, lo que facilita la creación y la exploración de datos. Se pueden referenciar gráficos en línea con el código y la narrativa relacionada, lo que ayuda a comprender mejor los resultados.

- **Facilidad de compartir**: Jupyter Notebooks se pueden compartir fácilmente con otros a través de archivos .ipynb o servicios en la nube como Google Colab o GitHub. Esto permite la colaboración y la revisión de código de manera más sencilla en comparación con compartir archivos de código de un entorno tradicional.

- **Soporte para múltiples lenguajes**: Además de Python, Jupyter Notebooks admite varios lenguajes de programación, lo que es útil si se trabaja con proyectos que involucran múltiples lenguajes.

- **Flexibilidad y extensibilidad:** Los Notebooks se pueden extender con complementos que agregan funcionalidad adicional, como soporte para gráficos interactivos, tablas dinámicas y más. Esto permite personalizar y adaptar el entorno según necesidades específicas.

- **Educación y enseñanza:** Los Jupyter Notebooks son ampliamente utilizados en entornos educativos y académicos para enseñar programación y conceptos de ciencia de datos. Los estudiantes pueden seguir los ejemplos y realizar experimentos de manera interactiva.

- **Comunidad activa y recursos disponibles:** Jupyter cuenta con una comunidad activa de usuarios y desarrolladores, lo que significa que hay una amplia cantidad de recursos en línea, tutoriales y bibliotecas disponibles para su uso.

5. Introducción a Kubernetes

Kubernetes, comúnmente abreviado como K8s, es un sistema de orquestación de contenedores de código abierto que ha revolucionado la gestión de aplicaciones en entornos en la nube. La historia de Kubernetes comenzó en Google, donde se desarrolló como un proyecto interno llamado "Borg" a principios de la década de 2000 para gestionar contenedores en sus vastos centros de datos. En 2014, Google anunció Kubernetes como un proyecto de código abierto, y desde entonces ha sido adoptado y respaldado por una amplia comunidad de desarrolladores y empresas de tecnología líderes.

Una de las principales bondades de Kubernetes es su capacidad para automatizar la implementación, el escalado, la gestión y la operación de aplicaciones en contenedores de manera eficiente y consistente. Permite a las organizaciones desplegar aplicaciones de manera rápida y confiable, independientemente de la infraestructura subyacente. Kubernetes ofrece una abstracción de la infraestructura, lo que significa que los desarrolladores pueden centrarse en escribir código y definir cómo debería ejecutarse la aplicación, sin preocuparse por los detalles de la infraestructura.

Kubernetes se ha convertido en un estándar de facto en la gestión de contenedores y es ampliamente utilizado en entornos en la nube por varias razones. Primero, es altamente escalable y puede administrar aplicaciones desde pequeños clústeres locales hasta grandes despliegues en la nube. Segundo, es agnóstico en cuanto a la plataforma, lo que significa que se puede ejecutar en una variedad de entornos de nube pública (como AWS, Azure, GCP) y privada (on-premises o en data centers). Tercero, su arquitectura modular permite la extensibilidad y la personalización, lo que lo hace adaptable a una amplia gama de casos de uso.

Esta herramienta proporciona características robustas para la gestión de contenedores, como la automatización de implementación y actualizaciones, el escalado automático, el equilibrio de carga, la autenticación y la autorización, el monitoreo y la recuperación ante fallos. También admite la gestión de aplicaciones de múltiples contenedores a través de conceptos como "Pods" (grupos de contenedores), servicios y volúmenes compartidos.

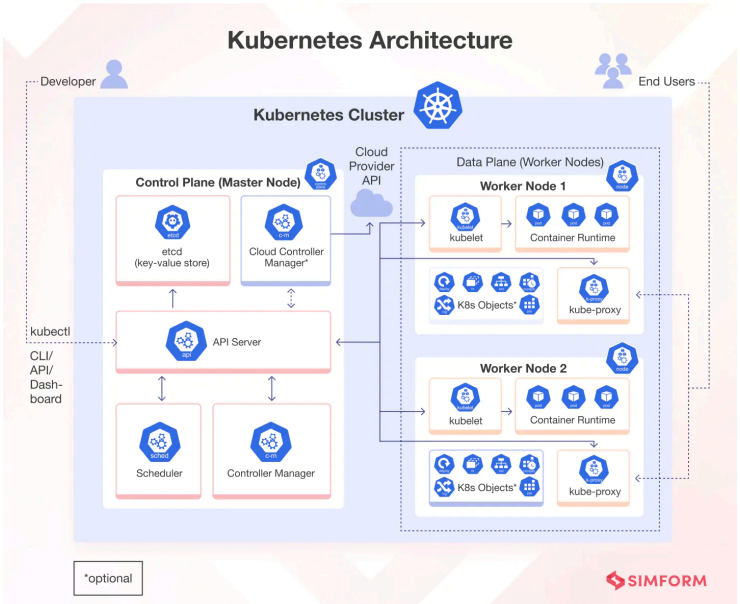


Figura 5: Arquitectura Kubernetes (fuente: sinform.com)

Además, en torno a Kubernetes se ha fomentado un ecosistema rico de herramientas y servicios complementarios que lo hacen aún más poderoso y versátil.

Se ha convertido en una base sólida para las implementaciones de microservicios y aplicaciones nativas de la nube, lo que permite

a las empresas adoptar arquitecturas modernas y escalables. Su popularidad se debe a su capacidad para ofrecer una gestión de aplicaciones altamente automatizada y eficiente, lo que reduce la complejidad operativa y los costos. Esto es especialmente beneficioso en entornos en la nube, donde la agilidad y la escalabilidad son esenciales para responder a las demandas cambiantes.

Empresas líderes de tecnología, como Amazon, Microsoft, Google, IBM y muchas más, ofrecen servicios gestionados de Kubernetes en sus respectivas plataformas, lo que facilita aún más la adopción y el uso de Kubernetes en entornos en la nube. Esto ha llevado a que Kubernetes sea una herramienta esencial en la gestión de aplicaciones y un componente fundamental de la infraestructura moderna. La historia de Kubernetes está marcada por su éxito en la simplificación y automatización de la gestión de aplicaciones en contenedores, convirtiéndo en una tecnología ampliamente utilizada y esencial en entornos en la nube a nivel global.

Otra ventaja clave de Kubernetes es su comunidad activa y su compromiso con el código abierto. Esto ha resultado en una rápida innovación y desarrollo de características, así como en una amplia disponibilidad de recursos educativos y de soporte. La documentación, las conferencias y los grupos de usuarios de Kubernetes son abundantes, lo que facilita la adopción y el aprendizaje continuo.

Dentro de las herramientas que Kubernetes ofrece en su ecosistema, se destaca una muy útil con relación a los entrenamientos de redes neuronales, los Jobs, los cuáles ejecutan un trabajo que puede tener cierta regularidad programada (*CronJob*) o requiera de una reparto de tareas o paralelización de algún tipo, existiendo uno especializado en Tensorflow (*TFJob*). Dado que el uso de contenedores es la base de Kubernetes, y en estos mismos contenedores se pueden ejecutar procesos de entrenamiento de redes neuronales, se encuentra en el concepto de ‘Job’, la simbiosis entre los dos mundos. Esta utilidad y una derivación de ella, los *TFJobs*, permitirán establecer las bases para escalar en cómputo y capacidad los procesos de entrenamiento, la realimentación y la clasificación de los modelos que se empleen.

Para ejecutar un Job se tiene que definir la imagen sobre la que se ejecuta el contenedor que el trabajo va a ejecutar, el comando de inicio (si procede, ya que hay imágenes que tienen su propio comando de inicio) y otros comportamientos personalizados que puedan resultar interesantes para el desarrollador fuera de los que tenga por defecto como si se ha de reiniciar en caso de error o los recursos máximos a emplear, entre otros.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: mi-trabajo
spec:
  template:
    spec:
      containers:
      - name: mi-contenedor
        image: ubuntu
        command: ["echo", "Hola mundo"]
      restartPolicy: Never
```

En el caso de que se quiera ejecutar un *CronJob*, basta con cambiar el tipo de entidad (en este caso de *Job* a *CronJob*) y definir su frecuencia de ejecución, Kubernetes hará el resto:

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: mi-cronjob
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: mi-contenedor
            image: ubuntu
            command: ["echo", "Hola mundo"]
          restartPolicy: Never
```

Como se puede observar, la regularidad se especifica mediante el formato cron, un estándar muy empleado que define la regularidad de la siguiente forma:

```
* * * * *
| | | | |
| | | | + → Día de la semana (0-6, donde 0 es domingo)
| | | +   → Mes (1-12)
| | +     → Día del mes (1-31)
| +       → Hora (0-23)
+         → Minuto (0-59)
```

De forma que en el ejemplo anterior, donde se usa “*/1 * * * *”, el trabajo se ejecutará “cada minuto” durante todos los días del año.

6. Saltando a la nube con Kubeflow

Se ha mencionado que el ciclo de vida de las redes neuronales tiene múltiples fases, previamente mencionadas: la recopilación de datos, el preprocesamiento de estos, el diseño de una arquitectura de la red, el posterior entrenamiento, la validación y ajuste (tunelado) de hiperparámetros, la evaluación del modelo, su ajuste y optimización para su despliegue en producción y la monitorización y mantenimiento de su rendimiento.

Dentro de todas estas necesidades, nace Kubeflow cómo una plataforma de código abierto diseñada para facilitar el despliegue, la gestión y el escalado de flujos de trabajo de aprendizaje automático (Machine Learning, ML) en entornos Kubernetes. Su historia comienza en el año 2017 cuando fue lanzado por Google como un proyecto de código abierto. La razón detrás de la creación de Kubeflow fue abordar los desafíos y las complejidades que enfrentan los equipos de desarrollo y los científicos de datos al implementar soluciones de ML a gran escala.

Los orígenes de Kubeflow se basan en la observación de que el aprendizaje automático se estaba volviendo cada vez más importante en la industria, pero la gestión de modelos y flujos de trabajo de ML en entornos de producción era compleja. Los equipos necesitaban una manera más eficiente y escalable de implementar y gestionar los modelos, y aquí es donde Kubeflow entró en juego.

Kubeflow se desarrolló como una respuesta a la necesidad de estandarizar y simplificar la implementación de aplicaciones de ML en Kubernetes, que ya había demostrado ser una plataforma sólida y escalable para orquestar contenedores. En lugar de forzar a los desarrolladores y científicos de datos a aprender varias herramientas y sistemas, Kubeflow proporcionó una plataforma unificada que aprovechaba las capacidades de Kubernetes.

Las bases de Kubeflow son:

- **Automatización y estandarización**: Kubeflow automatiza gran parte del proceso de implementación y gestión de modelos de ML, lo que ayuda a estandarizar las mejores prácticas y ahorra tiempo y esfuerzo.

- **Escalabilidad**: Permite el escalado horizontal de aplicaciones de ML en clústeres de Kubernetes, lo que facilita el procesamiento de grandes conjuntos de datos y modelos complejos.

- **Reproducibilidad**: Kubeflow facilita la reproducibilidad de experimentos de ML, lo que es esencial en la investigación y el desarrollo de modelos.

- **Colaboración**: Facilita la colaboración entre equipos de desarrollo, operaciones y científicos de datos al proporcionar una plataforma común para todos.

- **Flexibilidad**: Kubeflow es extensible y permite la integración con una variedad de bibliotecas y herramientas de ML, lo que lo hace versátil para diferentes casos de uso y entornos.

Este proyecto trata de aplicar esas ventajas sobre las fases del ciclo de vida de una red - exceptuando la recopilación de datos - mediante un conjunto de proyectos unificados bajo el mismo paraguas y orientados a resolver y automatizar los procesos requeridos para cada fase.

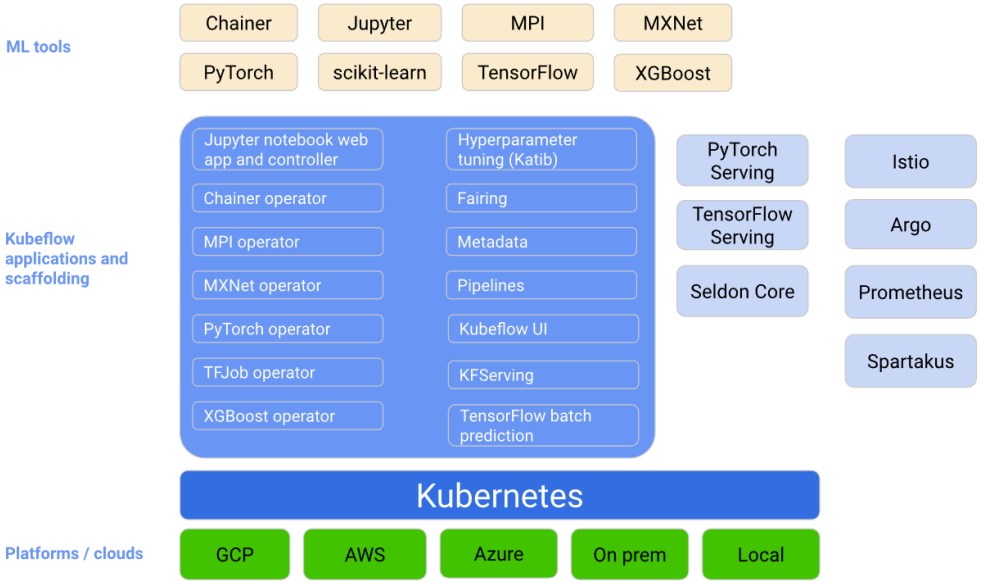


Figura 6: Diagrama de componentes de Kubeflow (fuente: kubeflow.org)

6.1 Kubeflow Pipelines para el preprocesamiento de los datos

Kubeflow Pipelines es una herramienta poderosa para abordar el preprocesamiento de datos en el ciclo de vida de una red neuronal. Una de sus principales bondades es su capacidad para diseñar y ejecutar flujos de trabajo de manera eficiente y escalable. Esto es esencial para tareas de preprocesamiento de datos que a menudo implican una serie de pasos complejos, como la limpieza de datos, la normalización, la codificación de características y la división de conjuntos de datos. Con Kubeflow Pipelines, puedes definir un flujo de trabajo que automatiza cada uno de estos pasos, lo que garantiza que los datos de entrada estén listos y en la forma adecuada para alimentar a tu red neuronal.

Esta herramienta permite definir esos flujos de trabajo mediante su SDK, programable en Python (*kfp*), permite definir cómo código cada trabajo que forma parte del flujo, teniendo dos opciones:

- Insertar de forma literal el código Python que realiza el trabajo, incluyendo las librerías importadas.

- Usar una imagen de Docker que contiene el código fuente a ejecutar que realizará el trabajo, permitiendo que los trabajos sean agnósticos al lenguaje de programación empleado, sacando el máximo partido al hecho de usar Kubernetes como mecanismo subyacente para la creación de instancias (‘pods’).

Además, Kubeflow Pipelines ofrece un entorno colaborativo y de gestión a través del panel de control central que facilita la colaboración entre equipos de científicos de datos y desarrolladores. Se pueden compartir y reutilizar flujos de trabajo, lo que promueve la estandarización y la eficiencia en la implementación del preprocesamiento de datos en tu proyecto. Adicionalmente, la herramienta permite la supervisión y la programación de flujos de trabajo, lo que garantiza que el preprocesamiento de datos se realice de manera consistente y programada de acuerdo a las necesidades del proyecto.

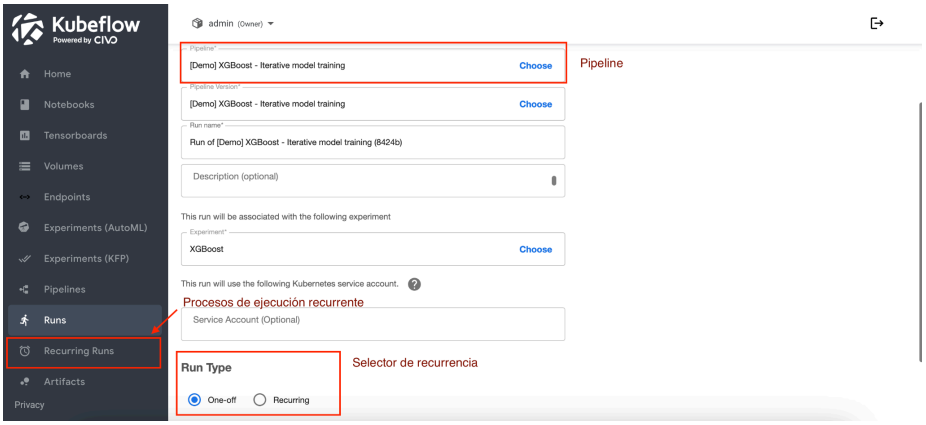


Figura 7: Sección Runs, panel central de Kubeflow (Fuente: Elaboración propia)

Esta herramienta simplifica y automatiza la fase crítica de preprocesamiento de datos en el ciclo de vida de una red neuronal, lo que reduce los errores, acelera el proceso y promueve la colaboración y la gestión eficiente de flujos de trabajo de datos complejos.

6.2 Katib para el diseño de la arquitectura de la red y el tunelado

Kubeflow Katib desempeña un papel crucial en la fase de diseño de la arquitectura de una red neuronal al ofrecer una solución sofisticada para la optimización de hiperparámetros, siendo esta igual de útil para la fase de tunelado. En proyectos de aprendizaje profundo, donde la elección de la arquitectura adecuada es esencial, Katib brilla al automatizar la búsqueda de configuraciones óptimas. Las redes neuronales profundas pueden tener numerosas capas y hiperparámetros, lo que dificulta la tarea de ajustar manualmente cada parámetro. Katib simplifica este proceso al permitir a los científicos de datos y a los ingenieros de machine learning explorar de manera eficiente una amplia variedad de combinaciones de hiperparámetros.

Su uso es sencillo y directo, mediante el concepto de ‘Experimento’, se define:

- Si es mediante la interfaz:

    a. Se seleccionan los ‘objetivos’, es decir, los parámetros a optimizar. Por ejemplo, es muy común especificar que para un Experimento con entrenamientos, se maximice - se puede maximizar o minimizar - el learning rate hasta cierto valor. Se pueden añadir métricas personalizadas, siempre y cuando se les asigne un alias.

b. Se seleccionan los parámetros a probar, incluyendo sus rangos, que pueden ser de tipo número (por ejemplo, ratio de aprendizaje comprendido entre 0.01 y 0.001 con incrementos de 0.005 entre cada prueba) o de tipo cadena (por ejemplo, tipo de optimizador a emplear en la ejecución: adam, sgd, etc). Una vez seleccionados.

c. Se seleccionan la cantidad de 'Trials' (pruebas) a realizar en paralelo y opcionalmente, los recursos que ha de tener cada proceso disponibles (RAM, CPU, GPU, si aplica), el criterio para parar el experimento (máximo límite de procesos fallidos, duración…)

d. Se selecciona el algoritmo de búsqueda (el criterio que se aplica sobre el orden de aplicación de las diferentes combinaciones de parámetros que generan los rangos proporcionados), de entre los disponibles:

    - **Random Search:** Este algoritmo realiza una búsqueda aleatoria en el espacio de hiperparámetros. Genera configuraciones de manera aleatoria y evalúa su rendimiento. Aunque puede ser menos eficiente que otros algoritmos, a veces puede encontrar configuraciones sorprendentemente buenas.

    - **Grid Search:** *Grid Search* explora un conjunto predefinido de valores para cada hiperparámetro. Es una búsqueda exhaustiva que evalúa todas las combinaciones posibles de hiperparámetros dentro del espacio especificado.

    - **Bayesian Optimization:** Este algoritmo utiliza técnicas de optimización bayesiana para modelar el espacio de hiperparámetros y determinar cuáles configuraciones tienen más probabilidades de mejorar el rendimiento. Es más eficiente que Grid Search y Random Search, ya que se adapta y enfoca en las áreas más prometedoras del espacio de búsqueda.

    - **Hyperband:** *Hyperband* es un algoritmo de búsqueda basado en bandas que explora diferentes configuraciones de hiperparámetros utilizando una estrategia de eliminación temprana. Proporciona un equilibrio entre la exploración y la explotación.

    - **TPE** (*Tree-structured Parzen Estimator*): TPE es otro algoritmo de optimización bayesiana que modela el espacio de hiperparámetros utilizando un modelo probabilístico y selecciona configuraciones en función de sus probabilidades de mejora.

    - **SMAC** (*Sequential Model-Based Algorithm Configuration*): SMAC es un algoritmo de optimización basado en modelos secuenciales que utiliza modelos de regresión para predecir el rendimiento de las configuraciones de hiperparámetros.

e. Opcionalmente, se ofrece usar *Median Stopping Rule* como algoritmo de parada 'temprana', esto es, para tratar de evitar que el modelo se sobreentrene,

adaptándose demasiado a los datos de prueba y perdiendo ese factor de aleatoriedad que le permita ofrecer un buen rendimiento con datos reales.

f. Finalmente se escogen diferentes colectores de métricas. Dado que Katib funciona igual que Pipelines, mediante imágenes de Docker / código Python embebido, ¿cómo comunican los procesos a Kubeflow que valor de los parámetros a optimizar han obtenido al terminar la prueba? Existen varias formas, siendo la más común y simple el uso de la salida estándar del proceso que se ejecuta. El proceso que realiza por ejemplo, un entrenamiento, debe emitir por la salida estándar estos valores a optimizar siguiendo un formato específico. Existen otras opciones cómo la escritura de esos valores en un archivo de una ruta específica que crea Kubeflow dentro del contenedor que ejecuta el trabajo, usando el formato de *Prometheus* o emitiendo un *TensorFlow Event*, que son otros protocolos más específicos. Es en este punto donde los procesos que se ejecutan deben calcular y comunicar las métricas personalizadas que hayamos añadido en el punto a.

- Si es mediante API, se enviaría un YAML con la declaración de los mismos parámetros mencionados via UI, mediante el uso de alguna herramienta como *kubectl*. La UI también genera un YAML - que da opción a visualizar / modificar / descargar antes de proceder con la creación del Experimento - conforme se van escogiendo opciones, siendo en el fondo, una forma más agradable / de más alto nivel de hacer lo mismo.

Tras crear el experimento, conforme los procesos se vayan ejecutando, el colector de métricas irá recolectando los resultados de cada prueba, los cuáles serán accesibles vía UI, al igual que el estado de cada prueba (exitosa/en progreso/fallida), su duración, los registros (logs) emitidos por cada proceso y la combinación de parámetros empleados que se han empleado, siempre contenidos dentro del espacio de parámetros que se le han pasado a Katib.

Cabe decir que para que todo esto sea posible, el código fuente ha de estar preparado para recibir los parámetros y emitir los resultados en el formato esperado, de lo contrario la comunicación con Katib no se realizará con éxito.

Por otro lado, se destaca el valor en especial de esta opción, ya que esta automatización no sólo acelera el proceso de diseño, sino que también aumenta las posibilidades de obtener modelos altamente precisos y efectivos en el aprendizaje profundo.

6.3 Kubeflow Training Operator para el entrenamiento

Kubeflow Training Operator simplifica la ejecución y el monitoreo de los entrenamientos de modelos de machine learning, lo que es esencial cuando se trabaja con conjuntos de datos grandes o modelos complejos. Este componente también se encarga de ejecutar cada prueba de Katib.

Es propicio recordar que la infraestructura de Kubernetes ofrece una escalabilidad excepcional, lo que significa que se pueden entrenar modelos en paralelo en múltiples nodos para acelerar el proceso de entrenamiento y gestionar fácilmente los recursos. Aquí es donde inciden los conceptos

de Job y *TFJob* entre otros, ya que son los encargados de ejecutar esos entrenamientos. Hay otras opciones como MPI o *XGBoost Training*, los cuáles son tipos de trabajos específicos para esas tecnologías. Siendo Job el más genérico - y por tanto el más limitado en temas de paralelismo -, requiere exclusivamente de una imagen de Docker que ejecutar y un límite de los recursos a destinar a cada ejecución. *TFJob* está preparado para entender y ejecutar trabajos distribuidos de TensorFlow, luego si el proceso a ejecutar está preparado para ello, esta sería la opción ideal, especialmente en ambientes con múltiples GPUs disponibles.

Esta herramienta simplifica significativamente la gestión de proyectos de machine learning, permitiendo a los equipos de trabajo colaborar y realizar un seguimiento eficiente del rendimiento de sus modelos. Además, la herramienta se integra bien con otras funcionalidades de Kubeflow, como el registro de metadatos y la monitorización, para proporcionar una visión completa y unificada del proceso de entrenamiento.

6.4 Kubeflow Pipelines UI para la evaluación del modelo

Aunque Kubeflow Pipelines se enfoca principalmente en la orquestación y automatización de flujos de trabajo de machine learning, puede ser complementado con herramientas y componentes adicionales para la evaluación y visualización de datos de rendimiento de un modelo, como el ROC (*Receiver Operating Characteristic*) y la matriz de confusión.

Aquí hay algunas características y consideraciones importantes sobre Kubeflow Pipelines UI y cómo se puede utilizar para evaluar y visualizar el rendimiento de un modelo:

- **Uso de componentes personalizados**: Se pueden desarrollar componentes personalizados que realicen tareas de evaluación específicas, como calcular métricas de rendimiento (por ejemplo, ROC AUC, precisión, recall) o generar visualizaciones como matrices de confusión. Luego, se pueden integrar estos componentes personalizados en tu pipeline. Estos componentes dan soporte al uso de tecnología como:
    a. Markdown
    b. HTML:
        i. Se puede especificar un archivo HTML que crea el componente y la interfaz de usuario de Kubeflow Pipelines representa ese HTML en la página de salida. El archivo HTML debe ser autónomo, sin referencias a otros archivos del sistema de archivos. El archivo HTML puede contener referencias absolutas a archivos de la web. El contenido que se ejecuta dentro del archivo HTML está aislado en un iframe y no puede comunicarse con la UI de Kubeflow Pipelines.

- **Visualización de resultados**: Una vez que el pipeline se haya ejecutado, se puede acceder a los resultados almacenados a través de Kubeflow Pipelines UI, a través de esta interfaz se pueden ver las métricas de rendimiento, las visualizaciones y otros datos relacionados con la evaluación del modelo.

- **Integración con herramientas de visualización adicionales**: Además de las funcionalidades integradas de Kubeflow Pipelines UI, se pueden integrar otras herramientas de visualización de datos y métricas, como TensorBoard o herramientas de visualización de datos personalizadas, para un análisis más detallado del rendimiento del modelo.

6.5 KServe para el despliegue

KServe permite la inferencia en Kubernetes y proporciona interfaces de alto rendimiento y abstracción para marcos de trabajo de aprendizaje automático (ML) como TensorFlow, XGBoost, scikit-learn, PyTorch y ONNX para resolver casos de uso de servicio de modelos de producción.

Basta con proveer una definición de recursos personalizada de Kubernetes (CRD) para ofrecer modelos de aprendizaje automático en las tecnologías previamente mencionadas. Esta herramienta tiene como objetivo encapsular la complejidad del escalado automático, las redes, la verificación del estado y la configuración del servidor mientras se ofrecen funciones como el escalado automático de GPU, el escalado a cero y los despliegues canarios a las implementaciones de aprendizaje automático.

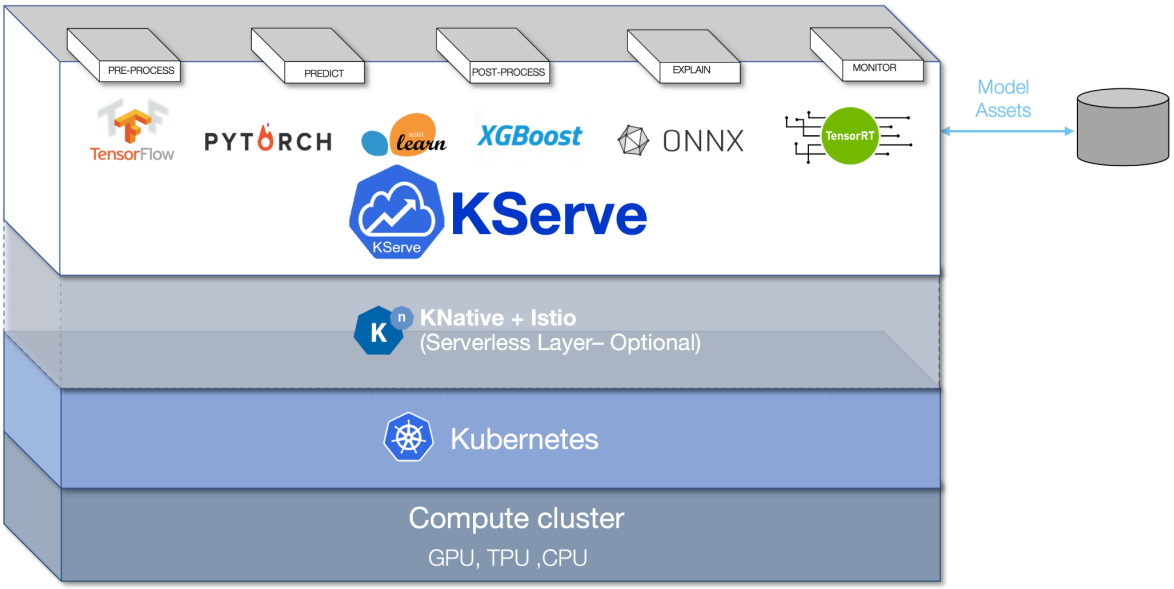


Figura 8: Diagrama de arquitectura de KServe (Fuente: KServe)

La definición (la 'forma' de pedírselo a KServe) del despliegue del modelo varía ligeramente dependiendo de la tecnología subyacente con la que se quiere trabajar, no obstante, la dinámica es sencilla y funciona de forma declarativa, en la línea típica de Kubernetes:

```
apiVersion: "serving.kserve.io/v1beta1"
kind: "InferenceService"
metadata:
  name: "cifar-10"
```

```
  …
spec:
  predictor:
    tensorflow:
      storageUri: "gs://models/my-model"
```

Básicamente se escoge el tipo de predictor (la tecnología subyacente) y se define dónde está el modelo a desplegar. Ese modelo ha de ser guardado en el formato que cada tecnología entiende y podría haber sido originado en un entrenamiento que ha terminado y ha subido ese modelo tras terminar, continuando el ciclo de operaciones que comprende el ciclo de vida de una red neuronal.

6.6 *Alibi Detect* para la monitorización del modelo

Alibi Detect es una biblioteca de Python que se utiliza para el monitoreo y la detección de anomalías en modelos de aprendizaje automático. Viene integrada de forma nativa en *KServe*, luego basta con hacer uso de él para empezar a obtener información de los modelos desplegados por KServe. Dentro de las opciones que Alibi ofrece encontramos:

- **Detección de anomalías en las predicciones:** Se puede realizar un seguimiento en tiempo real de las predicciones realizadas por el modelo desplegado en KServe. Este componente puede detectar anomalías en las predicciones, como valores inusuales o fuera de lo común, y generar alertas cuando se detectan esas anomalías, lo que ayuda a identificar problemas en el modelo o en los datos de entrada.

- **Monitoreo de la distribución de datos:** Alibi Detect también puede supervisar la distribución de los datos de entrada que se están utilizando para realizar predicciones. Esto es útil para detectar cambios en la distribución de datos que pueden indicar problemas en el entorno de producción o en la calidad de los datos.

- **Evaluación de la calidad del modelo:** Se puede evaluar la calidad del modelo en tiempo real mediante métricas de rendimiento y detección de anomalías. Esto puede resultar de ayuda para identificar si el modelo está funcionando de manera óptima o si necesita ser recalibrado o ajustado.

- **Generación de informes y alertas:** También existe la posibilidad de generar informes y alertas en tiempo real para que los equipos de operaciones y desarrollo puedan tomar medidas rápidas en caso de problemas con el modelo desplegado en KServe.

Cabe mencionar que esta herramienta está disponible en código, no vía interfaz, por ello para empezar a usarla sería necesario programar en Python que es lo que queremos monitorizar y cómo, y en base a eso, lanzar una alerta. Esto es relativamente fácil de integrar creando una imagen Docker que ejecute este programa de forma recurrente mediante *Recurrent Runs*.

6.7 *Recurrent Runs* para el mantenimiento

Las ventajas de poder usar imágenes Docker para realizar ejecuciones y la posibilidad de poder programar su recurrencia de ejecución hacen de Recurrent Runs la herramienta ideal para configurar cualquier mantenimiento que sea necesario. La flexibilidad ofrecida por esta funcionalidad en combinación con Docker es amplia, ya que podemos hablar con cualquier API de Kubeflow mediante código fuente para poder configurar cualquier tarea de mantenimiento que pueda resultar interesante.

Por mencionar un ejemplo, si Alibi detectase un sesgo en el modelo o si se contase con X información nueva o queremos simplemente re-entrenar el modelo con cierta frecuencia o ajustar los parámetros de nuevo mediante tunelado, podríamos simplemente configurar un trabajo recurrente mediante esta funcionalidad y usar las APIs ofrecidas por Kubeflow para extraer la información necesaria, cargar nuevos datos o ejecutar cualquier tarea mediante el uso de la librería *kfp*.

Para configurar un trabajo recurrente, se recurre a la interfaz del panel central de Kubeflow, seleccionando un Pipeline que esté vinculado a la imagen cuyo programa se quiere ejecutar y se selecciona una frecuencia de uso. Cabe mencionar que todas las ejecuciones se vinculan a un experimento y a un pipeline, lo que significa que cada ejecución aparecerá en esa sección también, permitiendo la visualización de cualquier registro que el programa haya realizado, cuántos recursos ha consumido, el tiempo que ha tardado y su estado en tiempo real.

Las opciones de recurrencia son variadas, existiendo la posibilidad de escoger detalles cómo la fecha de comienzo, la fecha de fin, cada cuánto se ejecuta, cuántas ejecuciones pueden existir a la vez (en caso de que duren mucho, podrían solaparse)

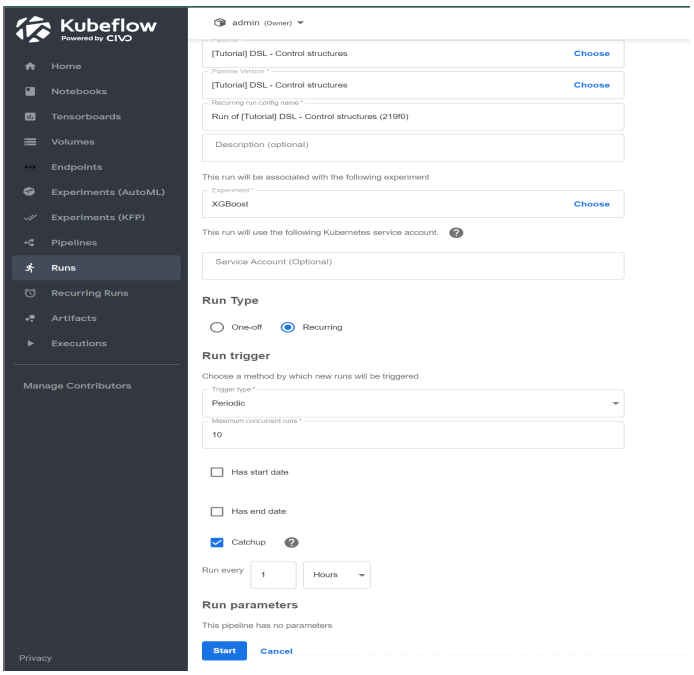


Figura 9: Configuración de una ejecución en Kubeflow (Fuente: elaboración propia)

7. Vertex AI

Dentro de las soluciones comerciales en la nube, que son ‘gestionadas’ para el campo del Machine Learning, se destaca Vertex AI cómo una herramienta eficaz y competente en este campo.

Vertex AI es una plataforma de inteligencia artificial desarrollada por Google Cloud Platform (GCP) que nace con el objetivo de simplificar y acelerar el desarrollo, entrenamiento y despliegue de modelos de machine learning a nivel empresarial. La historia de Vertex AI se basa en la creciente demanda de soluciones de IA y machine learning en la industria y en la experiencia de Google en el campo de la inteligencia artificial.

1. **Demanda creciente de IA:** La creciente demanda de aplicaciones de IA y machine learning en empresas de todos los tamaños generó la necesidad de una plataforma integral y fácil de usar que permitiera a las organizaciones aprovechar todo el potencial de estas tecnologías. Las empresas buscaban una solución que les permitiera implementar modelos de manera eficiente y a escala, y que proporcionará herramientas para el desarrollo colaborativo de modelos.

2. **Experiencia de Google en IA**: Google ha estado a la vanguardia de la investigación y desarrollo en inteligencia artificial durante años. La empresa ha desarrollado herramientas y tecnologías de IA de vanguardia, como TensorFlow, y ha utilizado la IA en sus productos y servicios, como Google Search y Google Photos. La experiencia acumulada en IA permitió a Google entender las necesidades de las empresas en este campo y diseñar una plataforma que las abordara de manera efectiva.

3. **La evolución de GCP**: Vertex AI se construyó sobre la base de Google Cloud Platform (GCP), que ya era una plataforma líder en la nube utilizada por numerosas empresas en todo el mundo. La integración de Vertex AI con GCP permitió a las organizaciones aprovechar la infraestructura y la escalabilidad de Google Cloud, lo que facilitó el desarrollo y la implementación de aplicaciones de IA en entornos empresariales.

Dentro de este contexto, la plataforma está planteada para cubrir las necesidades de Machine Learning, lo que engloba el ciclo de vida de un modelo de red neuronal. ¿Con qué herramientas cuenta esta plataforma para proveer una solución a las diferentes etapas?

7.1 *Data Labeling* para el preprocesamiento de los datos

Esta herramienta proporciona una plataforma completa para etiquetar y anotar conjuntos de datos de manera eficiente. El proceso de etiquetado es esencial para garantizar que los datos de entrenamiento sean adecuados y estén en el formato correcto para alimentar a los modelos de machine learning. Entre los tipos de datos que permite etiquetar se encuentran las imágenes, el texto o incluso videos.

Vertex AI Data Labeling ofrece capacidades de etiquetado manual y automático, lo que permite a los equipos etiquetar datos a gran escala y reducir el tiempo necesario para preparar los datos. El modo manual permite subir imágenes directamente a la interfaz y poder vincular las etiquetas previamente creadas.

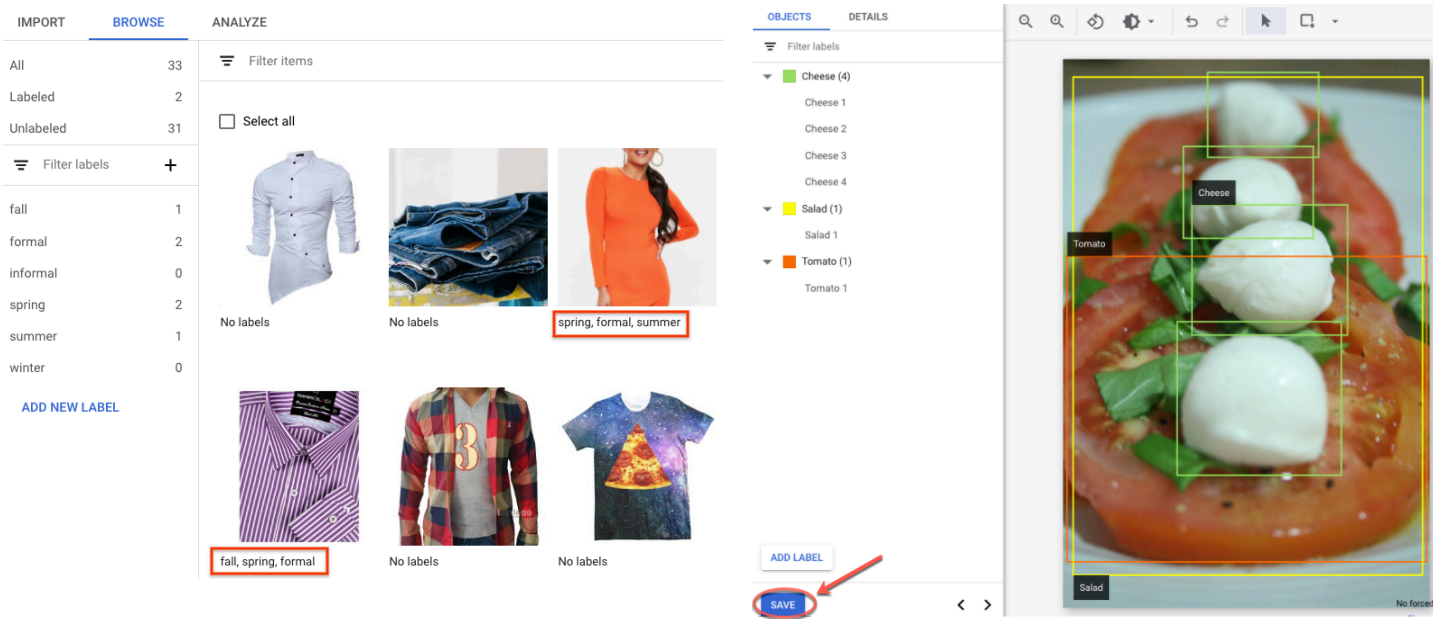


Figura 10: Etiquetado de muestras en Vertex AI (Fuente: Google)

Dentro de esta interfaz, se ofrece la posibilidad colaborativa, permitiendo a múltiples usuarios colaborar en el etiquetado de datos. Los equipos pueden definir flujos de trabajo de etiquetado personalizados, revisar y aprobar etiquetas, y gestionar de manera eficiente conjuntos de datos etiquetados. Esto promueve la coherencia en la calidad de los datos y acelera el proceso de preprocesamiento.

Por la parte del etiquetado automático, encontramos la posibilidad de dibujar un área dentro de cada imagen, y la plataforma tratará de "averiguar" qué etiqueta de las propuestas encaja mejor con las zonas seleccionadas.

7.2 *Neural Architecture Search (NAS)* para el diseño de la arquitectura de la red

Con la búsqueda de arquitectura neuronal de Vertex AI, se pueden buscar arquitecturas neuronales óptimas en términos de precisión, latencia, memoria, una combinación de estas o una métrica personalizada. Es una herramienta de optimización de alto nivel que se usa para encontrar mejores arquitecturas neuronales en términos de precisión con o sin restricciones. El espacio de búsqueda de posibles opciones de arquitectura neuronal puede ser de hasta 10^20. El sistema se basa en una técnica, que generó de forma correcta varios modelos de visión artificial de vanguardia en los últimos años, como *Nasnet* o *MNasnet.*, *EfficientNet*, *NAS-FPN* y *SpineNet*.

Las desventajas de este modelo tan 'automático' son advertidas por Google en su documentación:

- No es una solución en la se pueden incorporar los datos y esperar un buen resultado sin experimentación. Es una herramienta de experimentación.

- No sirve para ajuste de hiperparámetros, como pueden ser el ajuste de la tasa de aprendizaje o la configuración del optimizador.

- No se recomienda su uso con datos de entrenamiento limitados o conjuntos de datos altamente desequilibrados en los que algunas clases son muy poco frecuentes, si ya se usan magnificaciones importantes para el entrenamiento del modelo de referencia debido a la falta de datos, esta herramienta no se recomienda.

- No se usa un enfoque de supernet (*onehot-NAS* o NAS basado en uso compartido de la ponderación) en el que solo basta con proporcionar los propios datos. No es trivial - requiere meses de esfuerzo - personalizar una superred y a diferencia de esta, NAS es altamente personalizable para definir espacios de búsqueda y recompensas personalizadas. La personalización se puede realizar en aproximadamente uno o dos días.

- Esta funcionalidad se ofrece en 8 regiones alrededor del mundo, luego tampoco está disponible en todas las zonas.

Google estima [7] que si se tiene un espacio de búsqueda en mente, esta funcionalidad es muy valiosa y puede reducir al menos seis meses de tiempo de ingeniería para explorar un espacio de búsqueda de hasta las susodichas 10^20 opciones de arquitectura.

Se puede observar que esta herramienta está aún en sus primeras etapas y aunque es una opción disponible en producción y comercialmente, requiere de una capacidad de cómputo ingente, pues de momento se dirige a clientes empresariales que se puedan permitir gastar varios miles de euros en un experimento.

En un hipotético caso de uso proporcionado en la documentación, el coste de encontrar una arquitectura idónea para un modelo de red neuronal convolucional optimizada para ser usada en dispositivos móviles (*MnasNet,* [8]) en el plazo de un mes, se divide en 2 etapas, alcanzando un costo de 22.000€ aproximadamente, requiriendo la realización de un experimento de 2000 pruebas.

El sistema funciona con un mecanismo de recompensa donde a medida que aumentan la cantidad de pruebas, el controlador del experimento encuentra mejores modelos, recibiendo cada vez mayores recompensas, hasta que se alcanza un punto de convergencia donde la recompensa comienza a disminuir. Google estima el punto de convergencia en alrededor de 2000 pruebas, de ahí que la estimación de costes se base en ese supuesto.

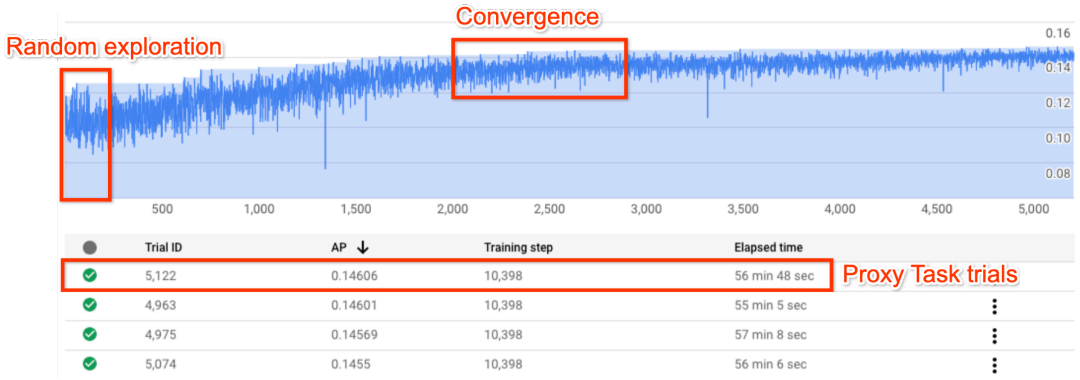


Figura 11: Convergencia en un búsqueda de arquitectura de red neuronal en Vertex AI (Fuente: Google)

7.3 *Training* para el entrenamiento

Vertex AI Training está diseñado para simplificar y optimizar el proceso de entrenamiento, permitiendo a los equipos de científicos de datos y desarrolladores entrenar modelos de manera eficiente y escalable.

Uno de los principales beneficios de Vertex AI Training es su capacidad para gestionar y orquestar el entrenamiento de forma que sea posible escalar horizontalmente el proceso, lo que acelera significativamente la velocidad y permite manejar conjuntos de datos grandes y modelos complejos de manera efectiva.

Los entrenamientos permiten definir la cantidad de recursos, el tipo (único o distribuido), la frecuencia - si aplica - y el tipo de hardware a emplear, incluyendo CPUs, GPUs y TPUs (añadir referencia aquí). Existe una compatibilidad totalmente integrada con algunos de los marcos de trabajo más famosos en aprendizaje automático cómo:

- PyTorch
- TensorFlow
- scikit-learn
- XGBoost

Entre las ventajas de este servicio también se pueden mencionar:

- **La seguridad empresarial:** Se pueden usar redes virtuales privadas (VPCs) para limitar el acceso a la red y mitigar riesgos de robo de datos en las ejecuciones de esos entrenamientos. El control del acceso e identidad de Google también están integrados y son totalmente compatibles, en caso de que los entrenamientos necesiten acceder a recursos restringidos o que requieren de una autorización específica.

- **La infraestructura de procesamiento está totalmente administrada:** El coste de los entrenamientos es por uso de cómputo, luego la necesidad de aprovisionar o administrar servidores desaparece. El control de los trabajos, el registro de lo que ocurre durante su

ejecución y la supervisión de su disponibilidad y funcionamiento es responsabilidad de la plataforma y por tanto, un dolor de cabeza menos.

Se destaca el hecho de que este componente sea capaz de integrarse con otras herramientas de la plataforma, facilitando el guardado de contenido generado por el entrenamiento (en el almacenamiento de Google), la interconexión con el tunelado de hiperparámetros, la extracción de los datos a usar de *BigQuery* o NFS, entre otros.

Esta herramienta admite el uso de imágenes Docker cómo base de la ejecución del entrenamiento, de forma que, se vuelve agnóstico a la tecnología empleada para realizarlo, ampliando el rango de posibilidades. No obstante, en el caso de Python, Google también ofrece imágenes compiladas previamente con toda la configuración realizada para ejecutar el trabajo, en el caso de algunas tecnologías como Tensorflow. De esta forma, basta escoger la versión de la imagen que interese y olvidarse de configuraciones adicionales. Si funciona en local con la versión X.Y, escoger la imagen de la versión X.Y garantiza que el código fuente funcionará sin problemas, e incluirá todo el soporte que pueda requerirse en caso de trabajos distribuidos y aceleración por GPU/TPU.

7.4 *HyperTune* para la validación y el tunelado de hiperparámetros

Vertex AI *HyperTune* aborda uno de los desafíos críticos en la construcción de modelos: encontrar las configuraciones de hiperparámetros óptimas que maximicen el rendimiento del modelo. Una de las ventajas clave es la capacidad de este componente para automatizar este proceso. Utiliza técnicas avanzadas, como la optimización bayesiana, para explorar de manera inteligente el espacio de hiperparámetros y encontrar las configuraciones que producen los mejores resultados. Esto ahorra tiempo y recursos que de otra manera se emplearían en ajustes manuales de hiperparámetros.

Este servicio permite a los usuarios definir métricas de evaluación personalizadas para que la herramienta seleccione las configuraciones de hiperparámetros en función de criterios específicos del proyecto. Esto brinda flexibilidad para optimizar el modelo según los objetivos y requisitos específicos.

Para ejecutar un experimento de tunelado, basta con definir una imagen de Docker a usar como base. Al igual que en Kubeflow, el proceso que ejecuta la imagen debe estar preparado para recibir los parámetros a probar en el espacio de parámetros en el que se busca encontrar la solución óptima.

Dentro de las ejecuciones a realizar, existe la posibilidad de escoger la cantidad de recursos y el tipo de cómputo, observar su uso en tiempo real y obtener perspectivas precisas de lo que sucede en cada momento. Dado que Google integra diferentes herramientas de otros servicios para su reutilización, las métricas del uso o la exploración de los registros que pueda emitir cada proceso se pueden visualizar en la interfaz de la misma forma que siempre se hace en otras partes de la plataforma, esto es, usando las mismas interfaces con las mismas opciones y capacidades. Esto hace más sencilla la adaptación a principiantes en el uso de esta sección, pero usuarios experimentados en la plataforma, ya que ofrece la misma capacidad de personalización con la que ya están familiarizados.

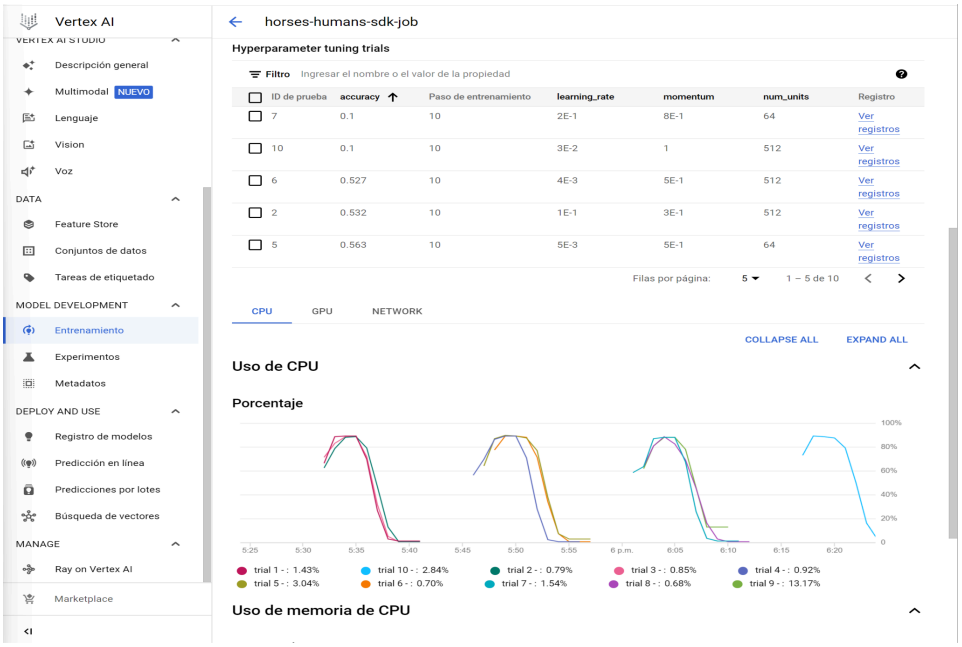


Figura 12: Sección Training en Vertex AI (Fuente: elaboración propia)

7.5 *Deploy & Use* para la evaluación del modelo y su despliegue

Esta fase es crítica para asegurar que los modelos entrenados sean utilizables y accesibles para aplicaciones y usuarios finales. El uso de esta herramienta se ofrece en 3 subcomponentes que cubren las distintas necesidades de esta etapa:

- **Registro de modelos:** Empezando por esta etapa, el primer paso obvio es preparar el modelo a desplegar. La forma de cargarlo es sencilla, basta con guardar el modelo generado tras los entrenamientos en un almacenamiento que o bien sea público, o bien sea dentro del Object Storage de Google de forma que este componente sea capaz de descargarlo y prepararlo para su uso. Existe soporte integrado para todas las tecnologías de entrenamiento mencionadas en el training, por ejemplo en el caso de Tensorflow, con guardar el modelo en formato *SavedModel* y subirlo a esta fuente de almacenamiento, sería suficiente. Posteriormente la plataforma empezará a analizar el modelo en segundo plano, y tras un tiempo, que en el caso de modelos sencillos es de 5 minutos, el modelo estará listo para conectarlo al despliegue. Esta herramienta permite copiar los modelos en diferentes regiones y desplegarlos en un sólo clic haciendo uso de la predicción en línea, también se permite el versionado del modelo. La evaluación de este, acepta modelos de imágenes, video y texto y se puede realizar de la siguiente manera:

    - Realiza un trabajo de predicción por lotes para el modelo entrenado a evaluar.

    - Proporciona unos datos de prueba que sean la verdad fundamental, es decir, que estén etiquetados de forma correcta según lo que sería un etiquetado 100%

correcto. El formato de estos datos suele ser el mismo que se usó durante la etapa de entrenamiento.

    - Ejecuta el trabajo de evaluación, el cuál usará la comparación del trabajo de predicción por lotes y los datos de verdad fundamentales para calcular las métricas de exactitud del modelo.

    - Itera el modelo ejecutando varios trabajos de evaluación tras realizar cambios y comparar los resultados de diferentes versiones del modelo que se han ido añadiendo al realizar esos cambios.

- **Predicción en línea:** Una vez con el modelo en el registro, esta sección permite seleccionarlo como fuente de un despliegue, escogiendo en qué zona estará disponible y si requerirá de una conexión privada o una estándar. En ambos casos, se desplegará una API HTTP que permitirá hacer peticiones enviando en el cuerpo de la petición la información en el formato que el modelo acepte. En el caso de un modelo de reconocimiento de imágenes que acepte por ejemplo imágenes en formato 32x32 en RGB, para una foto dada que quisiéramos clasificar, el conjunto de bits de la imagen, después de redimensionarla a 32x32 y normalizar sus colores a RGB. Cabe destacar que al configurar desde la interfaz este despliegue, se permite servir desde el mismo lugar múltiples modelos, especificando qué porcentaje de peticiones irá a cada modelo y la cantidad mínima de nodos de procesamiento por modelo, esto es, los recursos destinados a este despliegue, incluyendo el tipo de máquina (con/sin GPU, más o menos RAM/CPU, etc). El tema del porcentaje de peticiones es extremadamente útil si se busca desplegar una nueva versión del modelo y se quiere analizar su rendimiento en un entorno controlado, pues se puede redirigir, digamos, un 5% de las peticiones, de forma que se obtengan datos realistas de su desempeño en un ámbito real, sin poner el riesgo el sistema de predicción al completo o cómo prueba de concepto que pueda mejorar el uso de recursos, el tiempo de predicción o un mayor acierto.

- **Predicciones por lotes:** Este componente está orientado a predicciones que no requieren respuesta inmediata o incluso que su respuesta se pueda guardar en otro lugar. Un caso de uso muy común es para el análisis del modelo, ya que se pueden guardar los resultados en almacenes de datos como BigQuery, para realizar consultas posteriores.

Teniendo en cuenta estos 3 componentes, se concluye que el uso de esta herramienta simplifica la fase de despliegue de modelos de machine learning al proporcionar una solución escalable y gestionada que garantiza el funcionamiento eficiente de los modelos en entornos de producción. Su integración con herramientas de monitorización y su capacidad para gestionar múltiples versiones de modelos hacen que sea una herramienta valiosa para implementar modelos de manera confiable en aplicaciones empresariales.

7.6 *Model Monitoring* para el mantenimiento y la monitorización

Vertex AI Model Monitoring desempeña un papel esencial en las fases de mantenimiento y monitorización de modelos de machine learning en producción. Estas fases son cruciales para garantizar un rendimiento continuo y confiable de los modelos en entornos del mundo real.

Este componente también facilita la generación de alertas y notificaciones automáticas. Esto asegura que los equipos puedan tomar medidas inmediatas en caso de que se detecte un problema en el rendimiento del modelo. Además, la herramienta se integra con otros componentes de Vertex AI, como Vertex AI Model Deployment, lo que permite una gestión holística de modelos en producción.

Como base, un modelo implementado en producción tiene un mejor rendimiento en los datos de entrada de predicción que son similares a los datos de entrenamiento. Cuando los datos de entrada difieren de los datos que se usan para entrenar el modelo, el rendimiento del modelo puede disminuir, incluso si el modelo en sí, no cambia. Para mantener el rendimiento de un modelo, Model Monitoring puede supervisar los datos de entrada de predicción del modelo para detectar el sesgo y el desvío de atributos:

- **El sesgo entre el entrenamiento y la entrega** se produce cuando la distribución de datos de atributos en la producción difiere de la distribución de datos de atributos que se usaron para entrenar el modelo. Si los datos de entrenamiento originales están disponibles, se puede habilitar la detección de sesgos a fin de supervisar los modelos.

- **El desvío de la predicción** ocurre cuando la distribución de datos de atributos en la producción cambia de manera significativa con el tiempo. Si los datos de entrenamiento originales no están disponibles, se puede habilitar la detección de desvío para supervisar las entradas de datos en busca de cambios con el tiempo.

Esta detección de desvío y sesgo cuenta con soporte para atributos categóricos y numéricos:

- **Los atributos categóricos** son datos limitados por la cantidad de valores posibles, comúnmente agrupados por propiedades cualitativas. Por ejemplo, las categorías como tipo de producto, país o tipo de cliente.

- **Los atributos numéricos** son datos que pueden corresponder a cualquier valor numérico. Por ejemplo, peso y altura.

Una vez que el sesgo o desvío del atributo de un modelo excede el umbral de alertas que se establece, Model Monitoring puede enviar una alerta por correo electrónico. Otra ventaja es que también se pueden ver las distribuciones para cada atributo con el paso del tiempo para evaluar si el modelo necesita volver a entrenarse.

En el siguiente ejemplo, se muestra un sesgo o un desvío entre el modelo de referencia y las distribuciones más recientes de un atributo numérico:

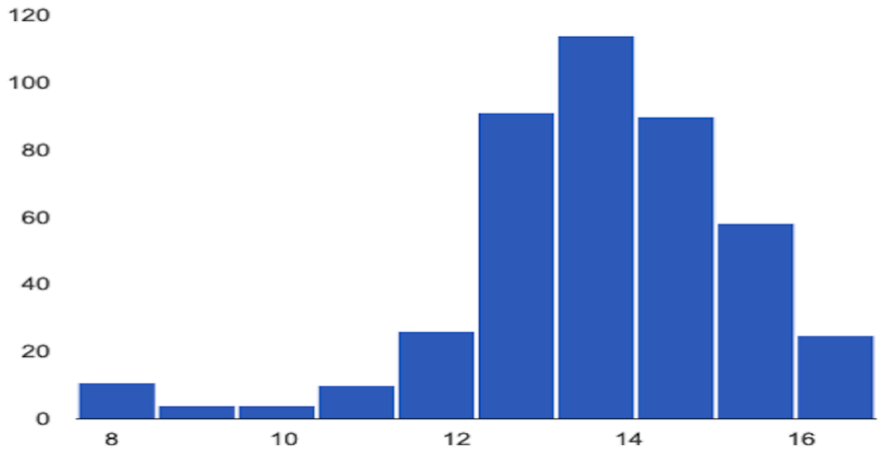


Figura 13: Distribución de un atributo numérico para un modelo sesgado (Fuente: Google)

Se observa en este caso que hay un sesgo hacia mayores valores, pues mayor cantidad de predicciones (eje Y) se dan en el espectro de valores >12 (eje X). Este tipo de información está disponible en el monitoreo gracias a la captura de esa información por parte del servicio.

Por otro lado, dentro de las opciones disponibles para configurar la monitorización, encontramos:

- **En el ámbito de los costes:** se puede configurar una tasa de muestreo de solicitudes de predicción, a fin de supervisar un subconjunto de las entradas de producción en un modelo.

- **En el ámbito de la frecuencia del análisis:** se puede establecer la frecuencia con la que se supervisarán las entradas registradas recientemente de un modelo implementado para detectar sesgos o desvíos. Se permite establecer el período (cada semana, por ejemplo) y el tamaño del período de supervisión (durante 1 hora por ejemplo).

- **En el ámbito de las alertas:** Se puede especificar el umbral para cada función que se supervisa, esto es, se registrará la alerta cuando la distancia estadística entre la distribución de los atributos de entrada y el modelo de referencia correspondiente supera el umbral especificado.

8. Desplegando un modelo para CIFAR-10

Una vez con el conocimiento necesario de lo que ofrece cada opción, cabe probar un pequeño caso práctico en el que se entrene, pruebe y despliegue un modelo que use CIFAR-10 como fuente de datos, realizando este proceso mediante ambas plataformas. Como referencia, el conjunto de datos CIFAR-10 contiene 60,000 imágenes 32x32 a color de 10 clases distintas: avión, coche, pájaro, gato, ciervo, perro, rana, caballo, barco y camión. Hay 6000 imágenes por cada clase.

La arquitectura de red neuronal que se usará es un modelo de red convolucional (CNN) diseñada para el procesamiento de imágenes. Este modelo es relativamente complejo y se inspira en arquitecturas conocidas como VGG, dada su secuencia de capas convolucionales profundas seguidas de capas de normalización por lotes, capas de agrupamiento máximo (*MaxPooling*), capas de abandono (*Dropout*), y capas densas al final. La arquitectura está estructurada para manejar imágenes de entrada de tamaño 32x32x3.

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(128, (3,3), activation='relu', padding='same', input_shape=(32, 32, 3)),
    tf.keras.layers.Activation('relu'),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu', padding='same', input_shape=(32, 32, 3)),
    tf.keras.layers.Activation('relu'),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu', padding='same', input_shape=(32, 32, 3)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),

    tf.keras.layers.Conv2D(256, (3,3), activation='relu', padding='same', input_shape=(32, 32, 3)),
    tf.keras.layers.Activation('relu'),
    tf.keras.layers.Conv2D(256, (3,3), activation='relu', padding='same', input_shape=(32, 32, 3)),
    tf.keras.layers.Activation('relu'),
    tf.keras.layers.Conv2D(256, (3,3), activation='relu', padding='same', input_shape=(32, 32, 3)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),

    tf.keras.layers.Conv2D(256, (3,3), activation='relu', padding='same', input_shape=(32, 32, 3)),
    tf.keras.layers.Activation('relu'),
    tf.keras.layers.Conv2D(256, (3,3), activation='relu', padding='same', input_shape=(32, 32, 3)),
    tf.keras.layers.Activation('relu'),

    tf.keras.layers.Conv2D(384, (3,3), activation='relu', padding='same', input_shape=(32, 32, 3)),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Flatten(),

    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1024, 'relu'),
    tf.keras.layers.Dense(512, 'relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),

    tf.keras.layers.Dense(10, activation='softmax')
  ])
```

Algunas características de esta arquitectura:

- **Capas convolucionales profundas:** Comienza con capas convolucionales (Conv2D) que tienen filtros de 128, 256 y luego 384 unidades con un tamaño de kernel de (3x3) y activación ReLU. Estas capas están diseñadas para extraer características de alto nivel a medida que la información avanza a través de la red.

- **Normalización por lotes**: Se utiliza después de algunas capas convolucionales para estabilizar y acelerar el entrenamiento, normalizando la salida de la capa anterior.

- **Pooling**: Se emplea para reducir la dimensionalidad espacial de las salidas de las capas convolucionales, ayudando a disminuir la cantidad de parámetros y la carga computacional, y también a controlar el sobreajuste.

- **Capas de abandono (*Dropout*)**: Se utilizan para regularizar el modelo, ayudando a prevenir el sobreajuste al "abandonar" aleatoriamente un número de características durante el entrenamiento.

- **Aplanamiento (*Flatten*)**: Transforma la matriz de características multidimensional en un vector antes de las capas densas, permitiendo que las características extraídas por las capas convolucionales y de pooling sean utilizadas por capas densas.

- **Capas densas:** Después del aplanamiento, la red utiliza capas densas con 1024 y 512 unidades y activación ReLU, seguidas de normalización por lotes y abandono, finalizando con una capa densa de 10 unidades con activación softmax para la clasificación final entre 10 categorías.

Una vez con el modelo listo y los conocimientos adquiridos, se puede proceder a realizar los entrenamientos.

8.1. En Vertex AI

8.1.1. Tunelado

Para realizar el tunelado en VertexAI se ha de definir cómo realizar la ejecución del experimento en HyperTune. En este caso práctico se usa la API de Google Cloud, pues permite definir la generación de los experimentos de una forma consistente.

Para ello, primeramente se instalan los SDKs de Google para Python:

```
pip3 install --upgrade --quiet google-cloud-aiplatform
```

También es necesario especificar qué proyecto de Google Cloud queremos que ejecute el trabajo de tunelado:

```
gcloud config set project {PROJECT_ID}
```

Una vez se cuenta con las librerías disponibles y el proyecto especificado, se ha de hacer uso de las librerías de Google en el script Python, lo cuál se consigue añadiendo las siguientes líneas:

```
import argparse
import hypertune

def get_args():
  '''Parses args. Must include all hyperparameters you want to tune.'''

  parser = argparse.ArgumentParser()
  parser.add_argument(
```

```
      '--learning_rate',
      required=True,
      type=float,
      help='learning rate')
  parser.add_argument(
      '--momentum',
      required=True,
      type=float,
      help='SGD momentum value')
  parser.add_argument(
      '--num_units',
      required=True,
      type=int,
      help='number of units in last hidden layer')
  args = parser.parse_args()
  return args

def create_model(num_units, learning_rate, momentum):
       model  = (ya está definido al principio del capítulo)
       model.compile(
             loss=tf.keras.losses.sparse_categorical_crossentropy,
             optimizer=tf.keras.optimizers.SGD(
                 learning_rate=learning_rate,
                 momentum=momentum
             ),
       metrics=['accuracy'])
  return model

def scale(image, label):
    image = tf.cast(image, tf.float32)
    image /= 255.0
    return image, label

def create_dataset():
  '''Loads Cifar10 dataset and preprocesses data.'''

  data, info = tfds.load(name='cifar10', as_supervised=True, with_info=True)

  BUFFER_SIZE=10000
  BATCH_SIZE=64

  # Create train dataset
  train_data = data['train'].map(scale)
  train_data = train_data.shuffle(BUFFER_SIZE)
  train_data = train_data.batch(BATCH_SIZE)

  # Create validation dataset
  validation_data = data['test'].map(scale)
  validation_data = validation_data.batch(BATCH_SIZE)

  return train_data, validation_data

def main():
  args = get_args()
  train_data, validation_data = create_dataset()
  model = create_model(args.num_units, args.learning_rate, args.momentum)
  history = model.fit(train_data, epochs=NUM_EPOCHS, validation_data=validation_data)

  # DEFINE METRIC
  hp_metric = history.history['val_accuracy'][-1]

  hpt = hypertune.HyperTune()
  hpt.report_hyperparameter_tuning_metric(
      hyperparameter_metric_tag='accuracy',
      metric_value=hp_metric,
      global_step=NUM_EPOCHS)
```

```
if __name__ == "__main__":
    main()
```

Este código sirve para el tunelado en ambas plataformas excepto por una ligera diferencia, el uso de la librería HyperTune para enviar el trabajo a GCP. Este programa está preparado para ser parametrizado, de forma que el tunelado pueda probar con distintas variaciones de éstos con el fin de maximizar la precisión (`metric_value=history.history['val_accuracy'][-1]`) .

Una vez con el programa listo, se debe 'Dockerizar', esto es, definir su ejecución dentro de Docker, de modo que se genere una imagen que sirva de base para cada prueba. Una posible definición de la imagen es esta:

```
FROM gcr.io/deeplearning-platform-release/tf2-gpu.2-8

WORKDIR /

# Installs hypertune library
RUN pip install cloudml-hypertune

# Copies the trainer code to the Docker image.
COPY script.py script.py

# Sets up the entry point to invoke the trainer.
ENTRYPOINT ["python", "script.py"]
```

Teniendo la definición, el siguiente paso es subir la imagen al repositorio de imágenes de Google mediante la creación de un repositorio en la zona donde se realiza el entrenamiento y subiendo la imagen a ese lugar:

```
gcloud artifacts repositories create <repo_name> --repository-format=docker \
--location=<gcp_region> --description="Docker repository"

IMAGE_URI = <gcp_region>-docker.pkg.dev/<project_id>/<repo_name>/cifar10:latest

docker build ./ -t $IMAGE_URI && docker push $IMAGE_URI
```

Por último, falta definir el hardware que se va a emplear y el rango de parámetros que se desea probar, lo cuál se puede hacer, de nuevo, mediante Python:

```
import os

import google.cloud.aiplatform as aiplatform
from google.cloud.aiplatform import hyperparameter_tuning as hpt

worker_pool_specs = [
    {
        "machine_spec": {
            "machine_type": "n1-standard-4",
            "accelerator_type": "NVIDIA_TESLA_T4",
            "accelerator_count": 1,
        },
        "replica_count": 1,
        "container_spec": {
            "image_uri": f"{IMAGE_URI}"
        },
    }
```

```
]

parameter_spec = {
    "learning_rate": hpt.DoubleParameterSpec(min=0.001, max=1, scale="log"),
    "momentum": hpt.DoubleParameterSpec(min=0, max=1, scale="linear"),
    "num_units": hpt.DiscreteParameterSpec(values=[64, 128, 512], scale=None),
}

metric_spec = {"accuracy": "maximize"}

my_custom_job = aiplatform.CustomJob(
    display_name="cifar10-job",
    worker_pool_specs=worker_pool_specs,
)

hp_job = aiplatform.HyperparameterTuningJob(
    display_name="cifar10-job",
    custom_job=my_custom_job,
    metric_spec=metric_spec,
    parameter_spec=parameter_spec,
    max_trial_count=10, # definimos cuantas pruebas como máximo se realizan
    parallel_trial_count=3, # definimos cuántas pruebas se realizan en paralelo
)

hp_job.run()
```

Si todo va bien aparecerá en la interfaz de GCP el trabajo de tunelado, su duración y su progreso -si aún no ha finalizado -:

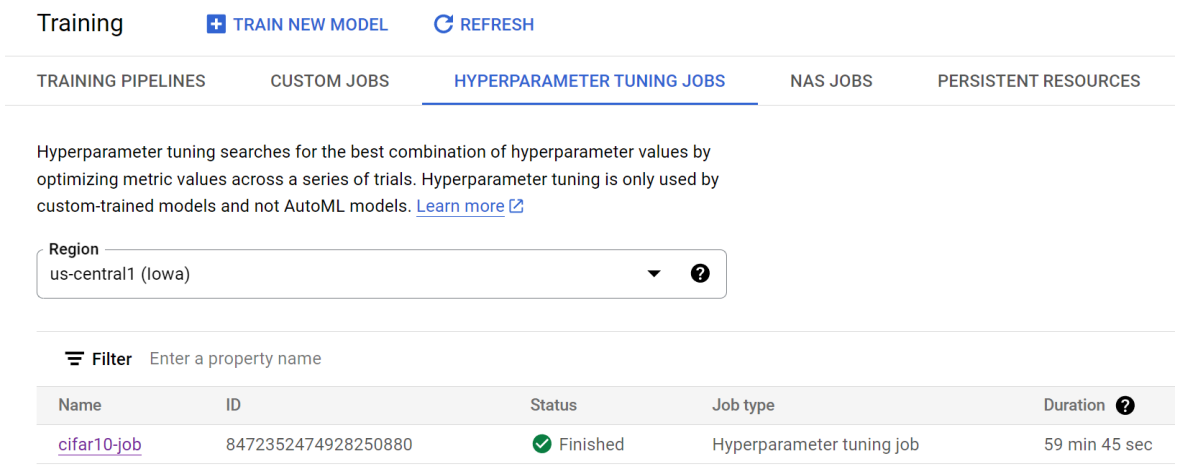


Figura 14: Lista de trabajos de tunelado en Vertex AI (fuente: elaboración propia)

Existe la posibilidad de ahondar en más detalles mediante la misma interfaz, pudiendo ver los registros de cada contenedor, su consumo de recursos y el resultado para cada combinación que se haya probado

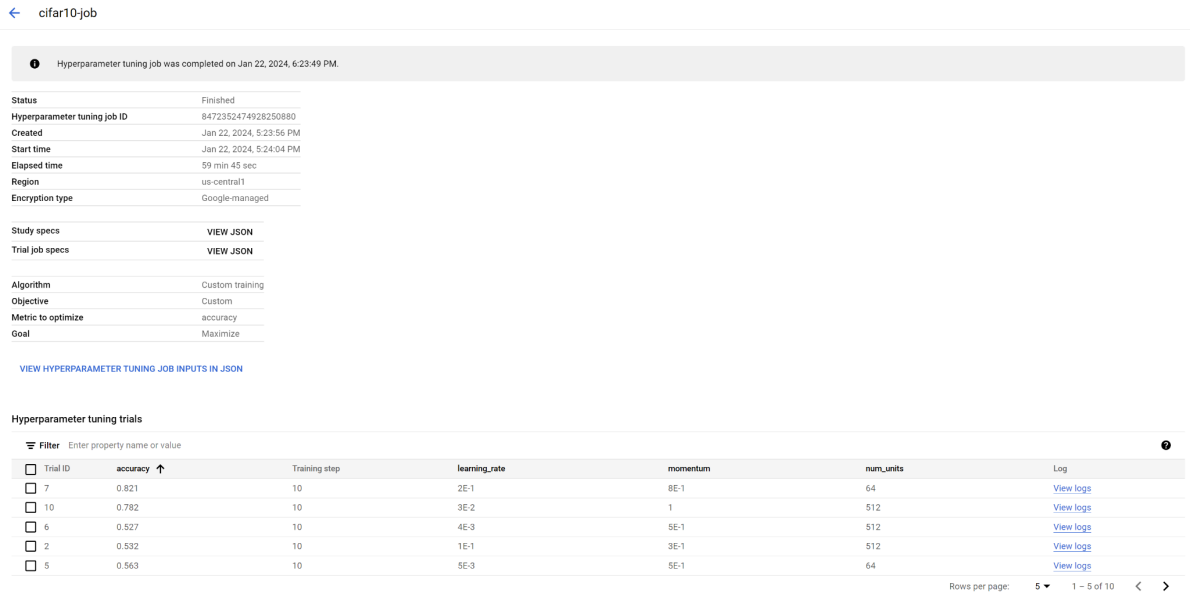
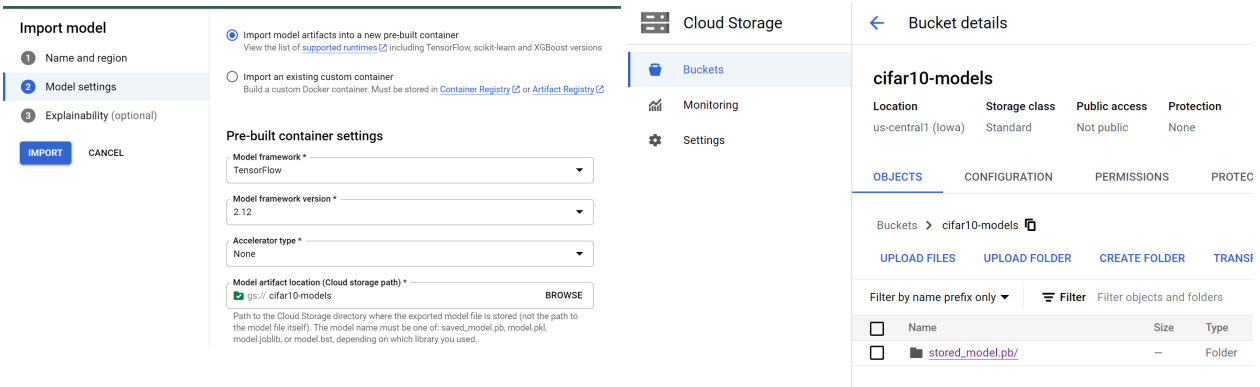


Figura 15: Panel de estadísticas sobre un trabajo de tunelado en Vertex AI

8.1.2. Despliegue e Inferencia

Una vez con el tunelado realizado y contando con los resultados, se puede proceder a servir el modelo usando la configuración más prometedora obtenida en el tunelado, para ello, se necesita exportar el modelo a formato SavedModel:

Una vez exportado a SavedModel, se han de subir los archivos generados al almacenamiento de objetos (*Object Storage*) y vincular ese 'bucket' (donde se guarda el contenido, en nuestro caso se llama cifar10-models) al despliegue del modelo en Vertex AI > Deploy & Use > Model Registry > Import. Tras unos minutos el modelo habrá sido procesado por la plataforma y estará listo para su despliegue.



Figuras 16 e 17: Importación un modelo disponible en el almacenamiento y bucket de almacenamiento que contiene el modelo entrenado con CIFAR-10 en formato Saved Model (fuente: elaboración propia)

Una vez con el modelo importado, basta ir a la sección *Deploy & Test* y pulsar en *Deploy to Endpoint.* Esta opción ofrecerá configuraciones cómo el hardware que se va a usar para el despliegue y la separación del tráfico en porcentaje si se desean usar diferentes versiones del modelo a la vez (70% a la versión 1 y 30% a la versión 2, por ejemplo).

Tras unos minutos el modelo estará desplegado y listo para realizar cualquier inferencia de imágenes.

8.2. En Kubeflow

8.2.1. Tunelado

Para realizar el tunelado en Kubeflow basta con usar el mismo script Python empleado en Vertex AI con una ligera diferencia: en vez de usar la librería de HyperTune, el programa generará registros por la salida estándar en el formato que un TFJob entiende. Para ello, se ha de añadir una función que se ejecutará tras el final de cada época y escribirá los datos relevantes para Katib por la salida estándar:

```
class KatibMetricLogger(Callback):
    def on_epoch_end(self, epoch, logs=None):
        logs = logs or {}
        print('logs', logs)
        train_accuracy = logs.get('accuracy')
        val_accuracy = logs.get('val_accuracy')
        if train_accuracy is not None:
            print(f"Epoch[{epoch}] Train-accuracy={train_accuracy}\n")
        if val_accuracy is not None:
            print(f"Epoch[{epoch}] Validation-accuracy={val_accuracy}\n")
...
model.compile ..
model.fit(
    ...
    callbacks=[KatibMetricLogger()],
)
```

Una vez realizada esta modificación, de nuevo el programa ha de ser 'Dockerizado', usando la misma definición que se usa en Vertex, aunque se puede eliminar la línea que instala la librería *hypertune*, pues en este caso no es necesaria. Se realiza la construcción y subida de la imagen, igual que antes:

```
docker build . -t cifar10:latest && docker push gvsergio/cifar10:latest
```

Se recomienda usar DockerHub como repositorio para almacenar la imagen, tanto por ser gratuito como por ser público, siendo además el lugar que prácticamente todos los clusters de Kubernetes 'conocen', esto significa que no necesita de ninguna configuración adicional para que el cluster sepa de donde descargar la imagen, más allá de su nombre.

Una vez con la imagen disponible, se define el tunelado en Katib, referenciando la imagen y los parámetros dinámicos que se le pasarán:

```
apiVersion: kubeflow.org/v1beta1
kind: Experiment
metadata:
```

```
  namespace: admin
  name: cifar10
spec:
  objective:
    type: maximize
    goal: 0.99
    objectiveMetricName: Validation-accuracy
    additionalMetricNames:
      - Train-accuracy
  algorithm:
    algorithmName: random
  parallelTrialCount: 3
  maxTrialCount: 12
  maxFailedTrialCount: 3
  parameters:
    - name: lr
      parameterType: double
      feasibleSpace:
        min: "0.001"
        max: "0.1"
    - name: epochs
      parameterType: int
      feasibleSpace:
        min: "10"
        max: "100"
    - name: batch-size
      parameterType: int
      feasibleSpace:
        min: "10"
        max: "40"
    - name: num-layers
      parameterType: int
      feasibleSpace:
        min: "2"
        max: "5"
    - name: optimizer
      parameterType: categorical
      feasibleSpace:
        list:
          - sgd
          - adam
          - ftrl
  trialTemplate:
    primaryContainerName: training-container
    trialParameters:
      - name: learningRate
        description: The learning rate
        reference: lr
      - name: epochs
        description: The number of epochs
        reference: epochs
      - name: batchSize
        description: The number of epochs
        reference: batch-size
      - name: numberLayers
        description: Number of training model layers
        reference: num-layers
      - name: optimizer
        description: Training model optimizer (sdg, adam or ftrl)
        reference: optimizer
    trialSpec:
      apiVersion: batch/v1
      kind: TFJob
      spec:
        template:
          metadata:
            annotations:
```

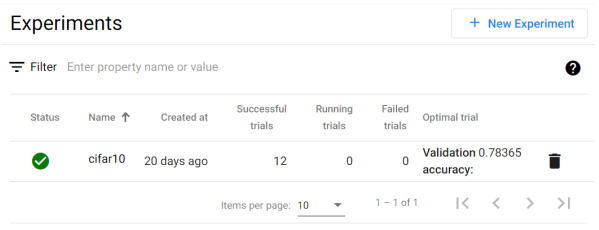
```
      sidecar.istio.io/inject: "false"
  spec:
    containers:
      - name: training-container
        image: gvsergio/cifar10:latest
        command:
          - "python"
          - "script.py"
          - "--lr=${trialParameters.learningRate}"
          - "--epochs=${trialParameters.epochs}"
          - "--batch-size=${trialParameters.batchSize}"
          - "--num-layers=${trialParameters.numberLayers}"
          - "--optimizer=${trialParameters.optimizer}"
        resources:
          limits:
            memory: "32Gi"
            cpu: "8"
    restartPolicy: Never
```

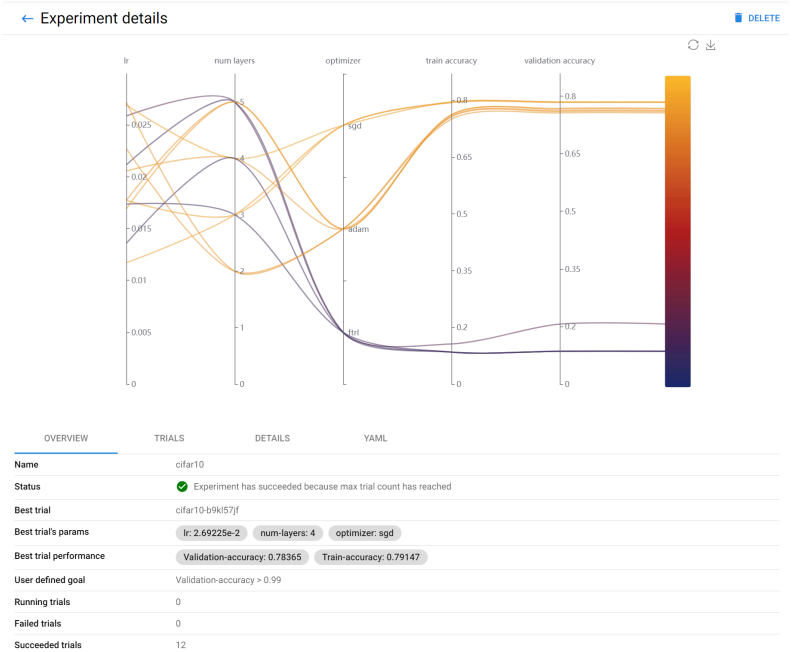
Nótese que en este archivo YAML se define todo lo necesario para el tunelado: parámetros y sus rangos de valores, recursos disponibles, comportamiento en caso de error, imagen a emplear, número máximo de pruebas, cantidad de pruebas en paralelo y variable objetivo y que se busca de su valor, en este caso, se busca maximizar la precisión.

Dicho esto, se aplica la definición del tunelado mediante *kubectl*:

```
kubectl apply -f cifar10-katib-experiment.yml
```

Tras aplicar el archivo YAML, aparecerá el trabajo en acción en el panel central de Kubeflow, al igual que con Vertex, se pueden observar estadísticas del proceso y ver los registros de cada ejecución, aunque en este caso, no se puede ver el uso de recursos empleado durante el tunelado.





Figuras 18 y 19: Ejecución de un experimento en Katib y estadísticas sobre este

8.2.2. Despliegue e Inferencia

En el caso del despliegue para su posterior inferencia se referencia el apartado 6.5, pudiéndose usar el mismo procedimiento de almacenamiento de Google ya descrito o cualquier otro para subir el modelo en formato SavedModel u otro de los soportados por Kubeflow especificados en la referencia [15].

Una vez servido el modelo, se puede realizar la inferencia mediante el balanceador de carga que viene disponible en Kubeflow, generalmente gestionado por *Istio*, el cuál se puede encontrar mediante *kubectl*:

```
kubectl get svc istio-ingressgateway -n istio-system

NAME                   TYPE           CLUSTER-IP       EXTERNAL-IP      PORT(S)   AGE
istio-ingressgateway   LoadBalancer   172.21.109.129   130.211.10.121   ...       17h
```

Una vez se encuentra el balanceador, el formato de la ruta es el siguiente:

```
http://${EXTERNAL-IP}:${PORT}/v1/models/cifar10:predict
```

Basta entonces con realizar peticiones POST en un objeto JSON con el formato:

```
{ "instances": [<imagen_en_el_formato_que_el_modelo_entiende>] }
```

Para obtener una respuesta con las predicciones, donde cada elemento es la probabilidad de que la imagen pertenezca a la clase en un orden específico (posición 0: coche, posición 1: avión, etc):

```
{ "predictions": [0.145, 0.563, 0.123, …] }
```

9. Comparativa

Una vez se han analizado las herramientas que ofrece cada tecnología, incluído un caso práctico, surgen algunos puntos importantes de contraste entre ambas plataformas dependiendo del ámbito en el que se establezca la comparación. Para ello, se plantea asumir el rol de dos enfoques que pueden llegar a tener - aunque no necesariamente - necesidades muy distintas en el contexto práctico: una gran empresa que quiere migrar sus procesos existentes de aprendizaje automático a los marcos de trabajo que cubren todo el ciclo de vida ofrecidos tanto por Kubeflow cómo por Vertex AI y una pequeña empresa que necesita integrar el uso de redes neuronales en un producto que va a lanzar al mercado.

9.1. Sencillez de uso

Ambas herramientas tratan de hacer la vida más fácil al usuario en términos de automatizar lo máximo posible de la forma más eficiente, no obstante, cabe recordar que su origen y naturaleza son diferentes, ya que Kubeflow es un proyecto Open Source y Vertex AI es una solución comercial.

Teniendo estos detalles en cuenta, si se empieza por el primer paso, esto es, migrar los procesos actuales o crear procesos de entrenamiento nuevos, Vertex AI ofrece un enfoque integrado, ya que tiene compatibilidad nativa con otras herramientas del Cloud de Google, luego en el ciclo de vida de una red neuronal:

- Existe la posibilidad de migrar los datos a *Big Query* por comodidad, o bien, usar cualquier otra fuente de datos.

- Se puede pre-procesar bien con Data Labelling o bien mediante ejecuciones de los procesos actuales de forma virtualizada, usando máquinas de cómputo de GCP o incluso creando trabajos de training cuya etapa previa sea ese pre-procesamiento. Destacar que existe soporte para consumir esos datos desde Big Query usando las librerías oficiales de Google, las cuáles están disponibles en 8 lenguajes diferentes (C#, C++, Java, JavaScript, Go, Python, Ruby, PHP). Si no se puede/quiere trabajar con estos lenguajes, las APIs para integrarse con cualquier componente se pueden consumir vía HTTP por cualquier lenguaje de programación a elección del desarrollador.

- Si cuenta con algún proceso de re-entrenamiento ya existente, se puede integrar en el Training y programar su recurrencia sin problema alguno, ya que de nuevo, se recurre a la virtualización para integrar cualquier proyecto en cualquier lenguaje de programación, incluyendo cualquier consumo de recursos, aunque sea ingente - con el consecuente coste -.

- El tunelado de hiperparametros tampoco tendría ninguna implicación seria a nivel de integración siempre y cuando se ejecute usando las tecnologías que Google acepta.

Actualmente Vertex acepta 4 tipos de modelos diferentes: los generados mediante XGBoost, scikit-learn, PyTorch y TensorFlow.

- Desplegar el modelo en producción es tarea sencilla dado que con que contar con ese modelo ya registrado en la plataforma (para ello, de nuevo, tiene que estar en un formato que Google acepte) se ofrece la posibilidad de despliegue en 2 clics, especificando los recursos a dedicar, contando además con la observabilidad inmediatamente disponible para su uso.

Una vez considerado el caso más problemático para Vertex AI, el de una migración, ¿qué puede ofrecer Kubeflow en esta situación desde el punto de la sencillez de uso?

Kubeflow es un proyecto Open Source, y de hecho, es un proyecto que integra otros proyectos de código libre cómo Katib, por ejemplo. Ello tiene ciertas implicaciones para las empresas que quieran migrar, pues hay una curva de aprendizaje relativamente más pronunciada que en Vertex. Al ser un proyecto abierto, la documentación existente es más técnica y se ofrece en un menor nivel y mayor brevedad que la que aporta Google. La documentación de Vertex acompaña al lector con ejemplos más completos y explicaciones más detalladas, lo que es un punto a considerar para cualquier interesado en contrastar ambos servicios, ya que integrarse con Kubeflow dá más flexibilidad para configuraciones, pero requiere más tiempo de dedicación.

Respecto a la integración con herramientas o procesos ya existentes:

- Podemos usar cualquier fuente de datos.

- El pre-procesamiento ha de venir elaborado en un proceso ya existente para integrarlo en Pipelines, no existe alternativa gestionada u ofrecida por Kubeflow.

- Gracias a la virtualización, Kubeflow es agnóstico a la tecnología, por ello, se pueden emplear programas ya existentes, siempre y cuando estén integrados mediante Docker para su uso y ejecución por parte de Kubeflow. Existe una librería llamada kfp para generar definiciones de entrenamientos con código, no obstante, sólo está disponible en Python. Otros proyectos como Katib también cuentan con librerías en Python, luego se pueden ‘integrar’ ambas cosas - un re-entrenamiento o un pre-procesamiento diferente de los datos antes de ejecutar un nuevo experimento de tunelado, por ejemplo -, aunque no es el objetivo principal de su existencia.

- El tunelado de hiperparámetros no requeriría ningún esfuerzo extra más allá del requerido para ejecutar un entrenamiento en Pipelines, ya que este mismo tunelado usa como gestor de ejecuciones subyacente, a Pipelines, limitándose su papel a añadir la capa de parametrización dinámica basada en múltiples criterios a escoger.

- Desplegar el modelo en producción, aunque pueda requerir un YAML, es trivial aunque requiere algo de conocimiento técnico sobre Kubernetes. Se admiten 11 tipos de tecnologías diferentes con sus formatos específicos de modelo [15]

Se puede concluir pues, que desde el punto de vista de la facilidad de uso, Vertex y Kubeflow tienen varios puntos en común, no obstante se destaca la necesidad de conocimientos requeridos para realizar ciertas operaciones cómo el tunelado y el despliegue en Kubeflow, donde Google tiene la accesibilidad más trabajada, aunque cabe mencionar, a costa de no ser tan flexible en las configuraciones disponibles. Por lo tanto, si se busca flexibilidad y mayor nivel de configurabilidad a costa de la necesidad de contar con ciertos conocimientos técnicos, Kubeflow puede resultar más interesante. Si por lo contrario, se busca agilidad y poder dedicar más tiempo al ciclo de vida del modelo despreocupándose por detalles de bajo nivel, Vertex AI puede ser más adecuado.

9.2. Costes

En cuanto a los costes:

- Para VertexAI: se usan los precios de referencia públicos [13] a fecha de 1 de Febrero de 2024 para la zona *us-central1* en Google (Iowa) en cuánto a cualquier mención de precios por cómputo. Vertex sigue un modelo de precios basados en pago por uso, lo que significa que los costos se relacionan directamente con los recursos consumidos y los servicios utilizados.

- Para Kubeflow: Al usar Kubernetes, lo que se paga si se monta 'a mano', esto es, si se realiza la instalación en un cluster desde 0, varía dependiendo del proveedor de servicios en la nube que se escoja. Hay que tener en cuenta que muchos proveedores cobran por uso de ancho de banda en cada instancia que forma parte del cluster, a parte del precio/hora o precio/mes por capacidad de cómputo. Por practicidad, claridad y en pro de realizar una comparativa lo más precisa posible, se usarán los precios del cómputo en Google también, de forma que usemos el mismo hardware y se pueda ver dónde radica la diferencia de costes, si es que la hay.

- Se usan como base 3 máquinas de tipo n1-standard-32 (32 vCPUs, 120GB RAM)

Ambas empresas buscan poder recorrer las fases del ciclo de vida hasta su puesta en producción de la forma más ágil posible, por ello, montan un cluster Kubeflow con 3 instancias n1-standard-32:

- La instalación, despliegue y configuración requiere un tiempo de trabajo que se estima entre 3 días y 1 semana, por parte de un perfil de nivel medio en Kubernetes.

- Se garantiza un SLA del 99.95%

- Ambas empresas tardan un mes en contar con un modelo listo para su uso en un entorno real. Los costes ascienden a 2.404,35$

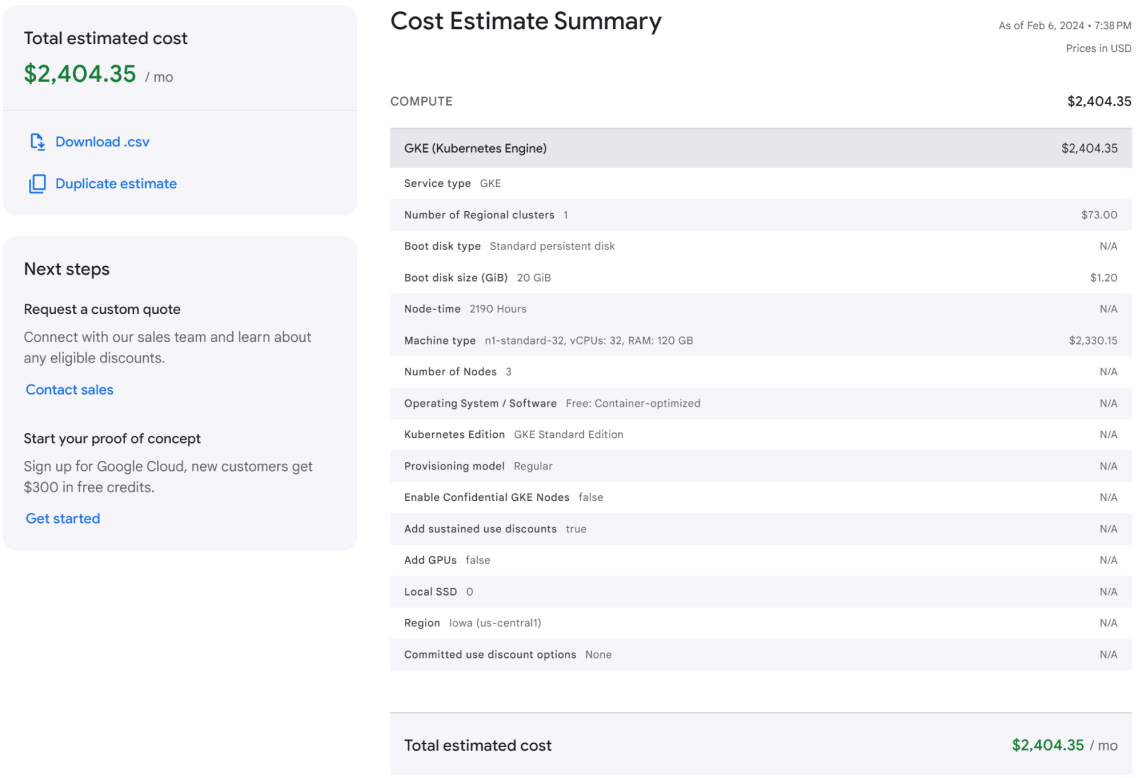


Figura 20: Estimación de coste para un clúster Kubernetes en GKE (Fuente: elaboración propia)

Destacamos que dentro de este periodo facturado, se han podido ejecutar tantos entrenamientos y predicciones como haya dado el tiempo que contiene el mes y los recursos contratados, esto es, se cuenta con la ventaja de obtener un precio estable a costa de tener un coste fijo, predecible y esperable. Dentro de los recursos disponibles, Kubernetes necesita reservar una parte para la propia gestión del cluster, en el caso de GKE (el servicio de Kubernetes de Google) para nuestro cluster se reservará un 8% de RAM y un 1% de CPU [9].

También se ha de tener en cuenta que se habrían de realizar pruebas previas para saber que tipo de instancia base es la más eficiente por $ invertido, independientemente de que todas ellas se puedan escalar lo que sea necesario, lo que implicará la instalación de Kubeflow posiblemente más de una o dos veces.

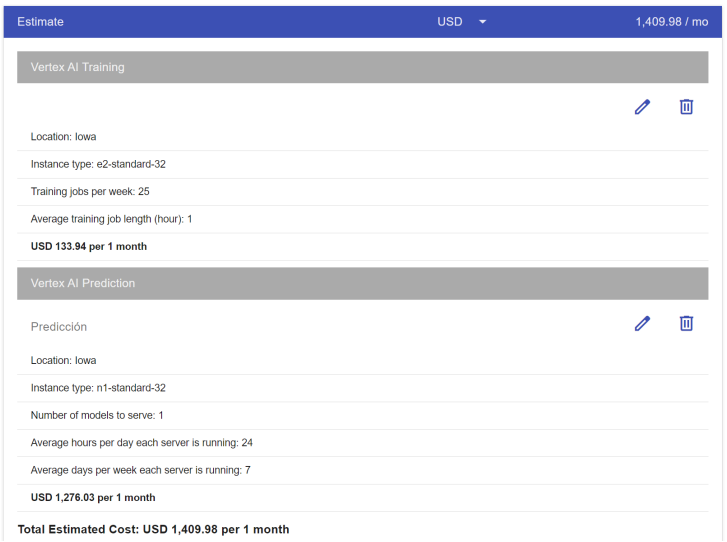
También se ha de tener en cuenta que adicionalmente a la monitorización del modelo que se configure usando Alibi, necesitamos instalar un sistema de monitorización para poder observar el rendimiento del propio cluster Kubernetes en sí, ya que Kubeflow depende directamente de este último para poder funcionar correctamente.

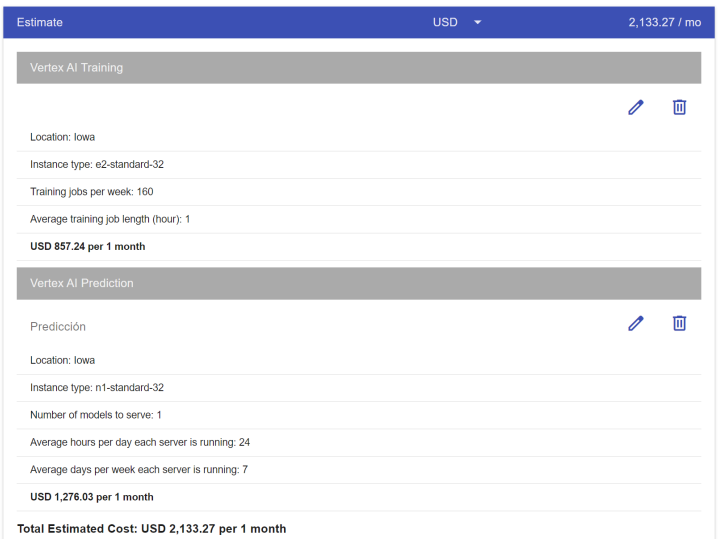
Por ello, aparejado al coste del hardware, se requeriría un perfil de Operaciones/Infraestructura que instale, controle y mantenga estas herramientas mientras mantiene una vigilancia semi-pasiva (gracias a las alertas) de que el rendimiento es el adecuado, además de que opcionalmente reajuste la instancia base del cluster en base a los datos que se puedan recolectar sobre el rendimiento de las máquinas.

Tras tener en cuenta las consideraciones de Kubeflow, pasamos a explorar la situación pero esta vez, usando Vertex:

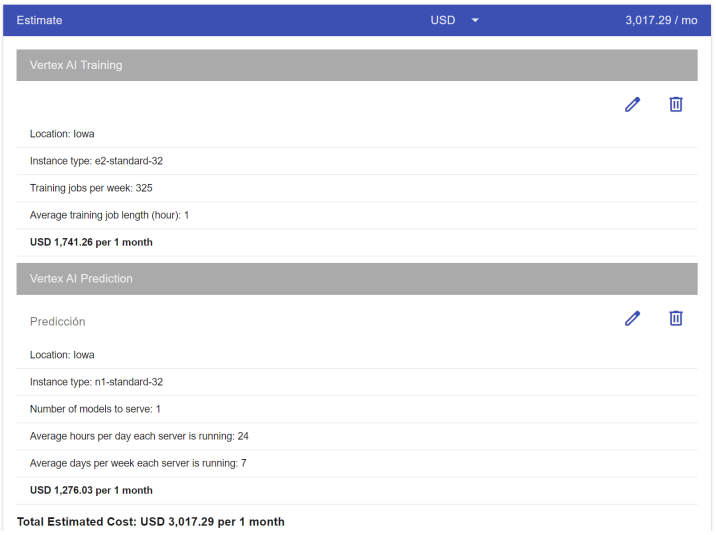
- Para realizar este supuesto y en pro de ver el valor por la variación hacemos cálculos en 3 situaciones diferentes:
    - El primer caso asume 100 entrenamientos al mes en total (10 para preprocesamiento y pruebas varias, 90 para ejecutar un tunelado con 90 combinaciones de parámetros diferentes). Esta cantidad permite probar 3 combinaciones de valores para 4 parámetros como pueden ser el número de épocas, el tamaño de lote, el ratio de aprendizaje y el número de capas.
    - El segundo caso asume 640 entrenamientos al mes en total (10 para preprocesamiento y pruebas varias, 640 para ejecutar un tunelado con 640 combinaciones de parámetros diferentes). Esta cantidad permite probar 4 combinaciones de valores para 4 parámetros como pueden ser el número de épocas, el tamaño de lote, el ratio de aprendizaje y el número de capas.
    - El tercer caso asume 1300 entrenamientos al mes en total (10 para preprocesamiento y pruebas varias, 630 para ejecutar un tunelado con 630 combinaciones de parámetros diferentes). Esta cantidad permite probar 5 combinaciones de valores para 4 parámetros como pueden ser el número de épocas, el tamaño de lote, el ratio de aprendizaje y el número de capas.

En todos los casos se usa 1 hora como duración de los entrenamientos y se especifica que el despliegue del modelo funciona 24 horas los 7 días de la semana.





Figuras 21 y 22: Cálculo de costes para entrenamiento y predicción en Vertex AI - Casos 1 y 2 (Fuente: elaboración propia)



A priori se observa que los costes de las predicciones vienen fijos por hora y es en los entrenamientos donde la parte variable se puede disparar.

Según la cantidad de pruebas que se busque realizar, impactando directamente en la cantidad de combinaciones de parámetros que se empleen en el tunelado, los costes pueden llegar a ser mayores o menores que disponer de un clúster con Kubeflow.

Figura 23: Cálculo de costes para entrenamiento y predicción en Vertex AI - Caso 3 (Fuente: elaboración propia)

Para facilitar la comparativa, se calcula un coste fijo de 1,77$/h para la parte de inferencia y realización de predicciones, y una parte variable en las fases combinadas de pre-procesamiento y entrenamiento que se sitúa en la horquilla de 0,18$/h-2,41$/h.

Por ello, se puede concluir que en este apartado, Kubeflow puede resultar más interesante si se cuenta con un equipo de profesionales cualificados para el mantenimiento y configuración del cluster y se requiere una total flexibilidad en la cantidad de procesos a ejecutar para que el equipo que trabaja en la elaboración del modelo de la red pueda trabajar sin límites más allá del propio tiempo disponible en el mes y la capacidad de cómputo, que en ambos casos, es la misma.

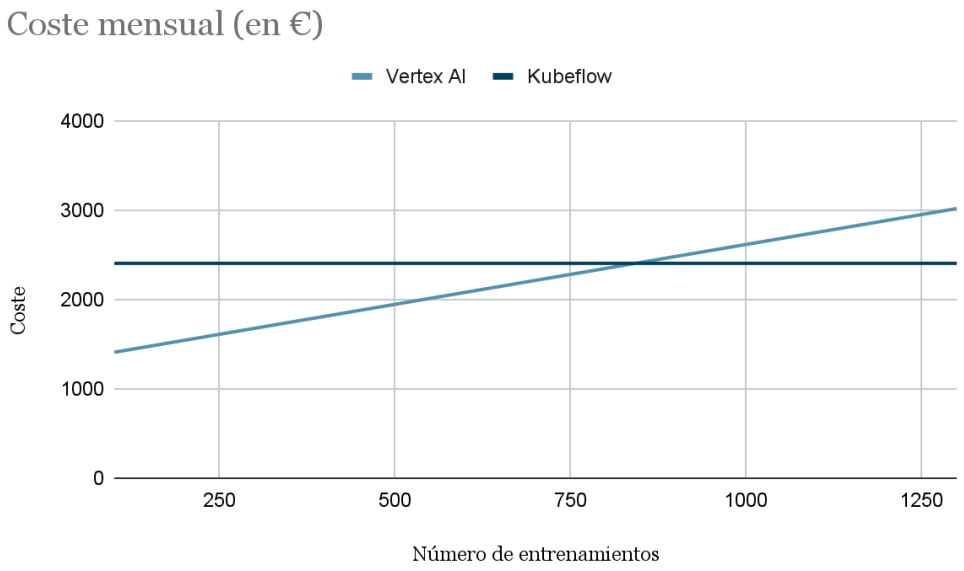


Figura 24: Gráfico comparativo de costes VertexAI - Kubeflow en GCP (Fuente: elaboración propia)

9.3. Escalabilidad

Un punto clave a la hora de ofrecer un modelo de red neuronal es considerar cuánta carga puede soportar mientras mantiene una alta disponibilidad y un rendimiento competitivo. Google es muy claro respecto a los límites de Vertex AI por proyecto, esto es, lo que se puede soportar por cada región que existe en Google. [10]

Tipo de solicitud	Peticiones por minuto
Peticiones de gestión de recursos (crear, eliminar, actualizar, borrar)	600
Peticiones de predicción en vivo	30000
Tráfico de peticiones de predicción en vivo	1,5GB

Tipo de solicitud	Valor
Número de entrenamientos concurrentes	2000
Número de instancias N1 activas para entrenamientos por región	2200

Son límites bastante altos, no obstante dado que Google cuenta con 49 regiones y se podrían balancear las peticiones por región, estos límites son aún más anchos si cabe. Esta escalabilidad aunque considerable, depende directamente del tipo de instancia escogida, esto es, aunque Vertex permite 30000 peticiones por minuto, eso no significa que exista una capacidad de soportar esa carga de por sí. Más bien significa que, empleando el tipo de máquina para cubrir las necesidades de cómputo para satisfacer por ejemplo, 40000 peticiones por minuto, cómo máximo se puede responder a 30000, debido a ese límite existente.

Dado que se cuenta con gráficos de consumo de recursos para las predicciones realizadas con el modelo desplegado, basta con cambiar el tipo de instancia para optimizar costes, conforme se vaya observando la evolución del uso de recursos para los diferentes tipos de máquinas disponibles..

En resumidas cuentas, Vertex permite despreocuparse por la escalabilidad del modelo desplegado, más allá de ir regulando qué tipo de máquina es escogida como base para realizar las predicciones en pos de la eficiencia económica.

Ahora bien, ¿qué sucede en el caso de Kubeflow? Teniendo en cuenta que se basa en el uso de Kubernetes, se obtienen de partida todas las ventajas del escalado con las que Kubernetes ya cuenta de forma nativa. La principal problemática es la necesidad de tener clústers en diferentes zonas en caso de querer balancear a nivel global la carga de peticiones que se pueda recibir, ya que los clusters cómo mucho, son capaces de dar soporte en una región. Como consecuencia, las latencias pueden ser significativas en caso de centralizar todas las predicciones en una sola zona, la cuál puede que esté considerablemente lejos de la aplicación cliente que la solicita.

Por otro lado, Kubernetes tiene sus propios límites, entre los que conviene resaltar:

Límite	Valor
Réplicas (pods) por nodo (máquina)	110
Total de nodos por cluster	5000
Total de réplicas por cluster	150000
Total de contenedores por cluster	300000

En el caso de Google, han extendido estos límites un poco más allá para su servicio GKE [10]:

Límite	Valor
Réplicas (pods) por nodo (máquina)	256
Total de nodos por cluster	15000
Total de réplicas por cluster	200000
Total de contenedores por cluster	400000
Nodos por grupo de nodos en una zona	1000

De nuevo, estos límites son considerables y en ningún momento se especifica límite de peticiones por minuto, lo que implica que si somos capaces de acomodar los recursos necesarios con el máximo de nodos permitidos, podríamos manejar más de 30000 peticiones por minuto.

No obstante, manejar un cuantioso número de réplicas y nodos tiene implicaciones en la estabilidad y rendimiento de Kubernetes. La red también puede enfrentar desafíos en términos de escalabilidad y latencia, afectando la comunicación entre los nodos. Además, la gestión de la configuración y los secretos a esta escala puede volverse compleja, y las actualizaciones y el despliegue de aplicaciones requieren una planificación cuidadosa para evitar interrupciones.

9.4. Sumario

Finalmente, se presentan unas tablas comparativas con los pros y los contras a modo de resumen con el fin de ofrecer una descripción general condensada.

	Vertex AI	Kubeflow
Curva de aprendizaje	Media	Alta
Autogestionado	Sí	No
Coste fijo	1,77$/h	3,38$/h (según proveedor)

Coste variable	Sí, dependiendo de la cantidad de entrenamientos	No
Límite de peticiones por minuto	30000 (por región)	-
SLA	>= 99,9% (entrenamiento), >= 99,5% (predicción en tiempo real)	>= 99,95%
Requiere personal técnico en infraestructura (mantenimiento)	No	Sí
Ofrece modelos ya entrenados	Sí	No
Ofrece NAS	Sí	Sí (versión Alpha)
Ofrece SDK	Sí, en 11 lenguajes de programación	Parcialmente, en Python
Ofrece soporte prioritario	Sí	No
Altamente configurable	No	Sí

10. Trabajo futuro

En este trabajo de fin de grado se han revisado los fundamentos teóricos de las redes neuronales y se ha explorado la implementación de redes neuronales utilizando diversas tecnologías basadas en Python y cómo portarlas a la nube. Dentro del análisis se ha abordado tanto la implementación en Python en general como la implementación específica en tecnologías especializadas y se han revisado los fundamentos teóricos de las redes neuronales, desde las neuronas individuales hasta conceptos como parámetros, funciones de activación, épocas, pérdidas y lotes.

Por otro lado, también ha explorado exhaustivamente las capacidades y limitaciones de dos plataformas para trabajar con inteligencia artificial y aprendizaje automático en la nube, cómo son Kubeflow y Vertex AI y cómo pueden ser aplicadas eficazmente para resolver problemas complejos. A través de la experimentación práctica y el análisis comparativo, se ha podido demostrar que, aunque cada plataforma ofrece ventajas únicas en términos de escalabilidad, eficiencia de costos y facilidad de uso, la selección adecuada depende en gran medida de las necesidades específicas del proyecto y del contexto de implementación.

Hemos encontrado que Kubeflow sobresale en entornos donde la flexibilidad y el control detallado sobre el proceso de entrenamiento de modelos son primordiales, mientras que Vertex AI ofrece una experiencia más integrada y gestionada, ideal para aquellos que buscan simplificar el despliegue y mantenimiento de modelos a gran escala. Además, este estudio ha identificado desafíos clave relacionados con la gestión de recursos, que deben ser considerados cuidadosamente durante la fase de planificación de cualquier proyecto de IA en la nube.

Dentro del uso de estas tecnologías y dado que ahora más que nunca, el aprendizaje automático está en boga y está generando cada vez mayor impacto en la sociedad a través de sus distintas aplicaciones, es probable que nuevas herramientas cada vez más sotifisticadas aparezcan con el transcurso de los años, teniendo en cuenta que las grandes tecnológicas (Amazon, Microsoft, OpenAI, Meta) están realizando fuertes inversiones en sus respectivas propuestas. Por ello, dentro del contexto de la inteligencia artificial, se proponen varias direcciones para posibles investigaciones futuras:

- **Evaluación de nuevas plataformas y tecnologías:** A medida que evolucionan las tecnologías de IA y aprendizaje automático, es crucial realizar evaluaciones periódicas de nuevas herramientas y plataformas, para entender mejor sus capacidades y cómo se comparan con las soluciones existentes.

- **Optimización de modelos de IA para la nube:** Desarrollar técnicas más avanzadas para optimizar modelos de IA específicamente para entornos en la nube, lo cual podría incluir la mejora de la eficiencia computacional y la reducción de costos operativos. Cabe destacar que Meta y Google ya están sacando sus propios modelos entrenados para uso comercial de forma que la parte de entrenamiento y mantenimiento vienen dados, permitiendo centrar el foco en hacer accesibles los modelos mediante integraciones en aplicaciones multiplataforma.

- **Ejecución de modelos en local:** Puede resultar especialmente interesante la ejecución de modelos en dispositivos locales (cómo teléfonos móviles) que han sido pre-optimizados para este fin y que vienen integrados en sus sistemas operativos de una forma natural y no invasiva, convirtiéndose en un 'asistente' más proactivo e inteligente que los asistentes de voz tradicionales.

- **Estudios de casos en industrias específicas:** Aplicar las tecnologías y plataformas evaluadas en este trabajo a problemas específicos de la industria, como la salud, la logística y la financiación, para demostrar su valor práctico y operativo en contextos del mundo real. Mencionar que una aplicación muy útil en España sería el uso del procesamiento de lenguaje natural en las consultas para redactar los detalles en la atención al paciente, aumentando la eficiencia de los hospitales públicos en materia de atención primaria, un problema acuciante actualmente.

- **Abordar desafíos éticos y de privacidad:** Profundizar en el estudio de las implicaciones éticas y de privacidad asociadas con el uso de IA y aprendizaje automático en la nube, desarrollando marcos de trabajo y mejores prácticas para mitigar potenciales riesgos. ¿Cómo se filtra qué información se puede o no usar para re-entrenar un modelo? ¿Qué permisos hay que solicitarle a los usuarios para poder hacer más o menos efectivos estos modelos mientras se informa de las implicaciones?

- **Fomentar la colaboración interdisciplinaria:** Promover proyectos de investigación que involucren la colaboración entre expertos en tecnología, ética, regulación y dominios de aplicación específicos, para asegurar que el desarrollo de la IA en la nube sea responsable, equitativo y beneficioso para la sociedad.

Estas áreas de trabajo futuro no solo prometen avanzar en el conocimiento y la aplicación de la IA en la nube, sino también abordar algunos de los desafíos más apremiantes que enfrentamos hoy en día, asegurando que estas tecnologías se desarrollen de una manera sostenible, ética y ampliamente accesible.

11. Anexo

11.1. Redes neuronales convolucionales

Las redes convolucionales, o CNN por sus siglas en inglés (*Convolutional Neural Networks*), son un tipo especializado de red neuronal artificial diseñada principalmente para procesar datos que tienen una estructura de cuadrícula, como imágenes. Estas redes son muy eficaces en tareas relacionadas con la visión por computador, como la clasificación de imágenes, la detección de objetos y el reconocimiento facial, entre otros.

Para comprender cómo funcionan las redes convolucionales, es útil entender su arquitectura y los conceptos clave que las impulsan:

- **Convoluciones:** La convolución es una operación matemática que aplica un filtro (también llamado kernel) a una región específica de una imagen. Este filtro se desliza a lo largo de la imagen y realiza una operación de multiplicación y suma de los valores de los píxeles que cubre. Esto permite extraer características importantes de la imagen, como bordes, texturas o patrones.

- **Capas convolucionales:** En una CNN, las capas convolucionales son responsables de aplicar convoluciones a la entrada (generalmente una imagen) utilizando múltiples filtros para extraer características. Cada filtro detecta diferentes características en la imagen, como bordes en diferentes direcciones o texturas. Estas capas son fundamentales para aprender representaciones significativas de los datos de entrada.

- **Capas de agrupación** (*Pooling*): Después de las capas convolucionales, es común utilizar capas de agrupación para reducir la dimensionalidad de las características extraídas y hacer que la red sea más robusta ante pequeñas variaciones en la posición de los objetos en la imagen. La operación de agrupación reduce el tamaño de la imagen al resumir localmente regiones de la misma.

- **Capas totalmente conectadas:** Después de una serie de capas convolucionales y de agrupación, las características extraídas se alimentan a una o más capas totalmente conectadas, que funcionan como en una red neuronal estándar. Estas capas finales interpretan las características extraídas y las utilizan para realizar la tarea específica para la que se diseñó la red, como clasificar objetos en una imagen.

- **Funciones de activación:** En cada capa de una CNN, después de realizar operaciones de convolución y agrupación, se aplica una función de activación no lineal, como ReLU (*Rectified Linear Unit*). Estas funciones son cruciales para introducir la no linealidad en la red y permitirle aprender relaciones complejas entre las características de entrada y las salidas deseadas.

11.2. Redes neuronales recurrentes

Las redes recurrentes (RNN por sus siglas en inglés, *Recurrent Neural Networks*) son un tipo de arquitectura de redes neuronales diseñadas para procesar secuencias de datos, donde la salida en un momento dado depende de las entradas anteriores y, en algunos casos, también de las salidas anteriores. Este tipo de redes cuenta con unas características específicas que las hacen particularmente útiles para modelar datos secuenciales como texto, audio o series temporales:

- **Estructura recurrente:** A diferencia de las redes neuronales tradicionales, donde la información fluye en una sola dirección (de entrada a salida), en las RNN, hay conexiones recurrentes que permiten que la información persista en la red y se utilice en momentos posteriores. Esto podría verse como una especie de "memoria" interna que tiene la red y que le permite recordar y utilizar información de pasos de tiempo anteriores.

- **Celdas recurrentes:** La unidad básica de una RNN es la celda recurrente. Esta celda toma como entrada tanto la entrada en el paso de tiempo actual como la salida de la celda en el paso de tiempo anterior, lo que permite que la red mantenga un estado interno o memoria a medida que procesa la secuencia.

- **Procesamiento secuencial:** Las RNN procesan las secuencias de datos de manera secuencial, paso a paso. En cada paso de tiempo, la celda recurrente calcula una salida basada en la entrada actual y la salida anterior. Esto permite que la red capture dependencias temporales y contextuales en los datos.

- **Problema de desvanecimiento y explosión del gradiente**: Uno de los desafíos principales en el entrenamiento de RNNs es el problema de desvanecimiento o explosión del gradiente. Debido a las conexiones recurrentes, los gradientes que se retro propagan a través del tiempo pueden volverse extremadamente pequeños o grandes, lo que dificulta el entrenamiento efectivo de la red. Para abordar este problema, se han desarrollado arquitecturas más avanzadas como LSTM (*Long Short-Term Memory*) y GRU (*Gated Recurrent Unit*).

- **Arquitecturas LSTM y GRU:** Estas son extensiones de las RNN estándar diseñadas para abordar el problema de desvanecimiento y explosión del gradiente. Introducen mecanismos de puertas que controlan el flujo de información dentro de la celda recurrente, lo que permite que la red aprenda a retener o descartar información en función del contexto.

- **Aplicaciones:** Las RNN y sus variantes encuentran aplicación en una amplia gama de tareas, como el procesamiento del lenguaje natural (traducción automática, generación de texto, análisis de sentimientos), la generación de música, el modelado de series temporales (predicción del clima, finanzas), el reconocimiento de voz y mucho más.

11.3. Instalación de Kubeflow en un entorno de desarrollo

Contar con una versión de Kubeflow en local puede ser extremadamente útil para realizar pruebas de concepto, nuevos desarrollos de procesos que posteriormente se pueden escalar o simplemente aprender a usar las herramientas nuevas que puedan aparecer con cada actualización.

Para poder empezar a trabajar con Kubeflow en local, es necesario primero contar con un cluster de Kubernetes totalmente operativo sobre el que instalar Kubeflow. Una herramienta frecuentemente usada para este fin es *microk8s*.

Para instalar el cluster basta con ir a la página oficial [19] y escoger la versión para nuestro sistema operativo, estando disponible para Ubuntu, Windows 10/11 y macOS (a partir de la versión *Yosemite*). Una vez terminada la instalación se comenzará a crear el cluster en segundo plano de forma automática.

Antes de instalar Kubeflow es necesario contar con algunas funcionalidades extra de *microk8s* activas, como un servicio de DNS para que los componentes se puedan encontrar entre ellos, un soporte para el almacenamiento y un balanceador de carga, entre otros. Se pueden habilitar estas funcionalidades mediante el siguiente comando:

```
microk8s enable dns hostpath-storage ingress metallb:10.64.140.43-10.64.140.49 rbac
```

Tras ejecutar el comando deberemos esperar unos 5 minutos aproximadamente, se puede confirmar que todo está listo mediante el comando:

```
microk8s status
```

Este comando muestra el estado del cluster y que complementos de *microk8s* están habilitados. En esta lista se deben poder ver las funcionalidades que hemos solicitado habilitar bajo la categoría *enabled*:

```
microk8s is running
high-availability: no
  datastore master nodes: 127.0.0.1:19001
  datastore standby nodes: none
addons:
  enabled:
    dns
    ha-cluster
    hostpath-storage
    ingress
    metallb
    storage
    rbac
  disabled:
...
```

Una vez se cuenta con el cluster y sus correspondientes elementos habilitados, la instalación de Kubeflow puede comenzar. Hay dos formas de conseguir el mismo objetivo llegados a este punto:

- Realizar la instalación mediante herramientas semi-automatizadas como juju (Canonical, Ubuntu [16] o deployKF [17]

- Aplicar las definiciones de los componentes [18] de Kubeflow en Kubernetes mediante herramientas como *kustomize*, que nos permiten configurar algunos parámetros.

Personalmente, suele ser más ventajoso usar esas herramientas semi-automatizadas, pues hay un interés por parte de los proyectos que los sostienen de brindar instalaciones compactas y estables, actualizables posteriormente y mediante sencillos comandos, de forma que el mantenimiento y la cantidad de incidentes durante ese mismo proceso se reduzca al mínimo.

Por el otro lado, si necesita un nivel de granularidad o configuración muy detallado, se puede optar por la segunda opción. Esta opción sólo requiere la aplicación de dos pasos que funcionan tanto en Ubuntu cómo en MacOS, fácilmente adaptable a la versión en Windows:

- La instalación de *kustomize*:

    - Ubuntu/Debian: `curl -s "<https://raw.githubusercontent.com/kubernetes-sigs/kustomize/master/hack/install_kustomize.sh>" | bash && sudo install kustomize /usr/local/bin/kustomize`

    - MacOS: `brew install kustomize`

- Descargar las definiciones de Kubeflow del repositorio público y aplicarlas: `git clone https://github.com/kubeflow/manifests.git && cd manifests && while ! kustomize build example | awk '!/well-defined/' | microk8s kubectl apply -f -; do echo "Retrying to apply resources"; sleep 10; done`

Tras seguir estos dos sencillos pasos se puede observar con el comando *kubectl* (que viene integrado en *microk8s*) cómo va evolucionando la instalación:

```
microk8s kubectl get pods -A
```

Tanto en este caso cómo en el de las herramientas semi-automatizadas habrá que vincular el servicio dónde se halla el panel de central de Kubernetes dentro del cluster a un puerto de nuestro ordenador, para poder crear un fácil acceso mediante el navegador:

```
microk8s kubectl port-forward svc/istio-ingressgateway -n istio-system 8080:80
```

Al ejecutar este comando se puede acceder al panel desde el navegador abriendo una pestaña en la url *http://localhost:8080*

Las credenciales por defecto en las pruebas realizadas son *user@example.com* como usuario y *12341234* como contraseña.

En el caso de decidir usar *juju*, una herramienta para Ubuntu/Debian, el despliegue de los componentes se realizaría de la siguiente forma:

```
juju add-model kubeflow && juju deploy kubeflow --trust
```

Estos comandos se encargan de descargar y aplicar lo necesario para instalar los componentes. De nuevo, se puede monitorear que todo está correctamente funcionando bien mediante el uso del comando anteriormente mencionado (`microk8s kubectl get pods -A`) o bien mediante el propio programa *juju*:

```
watch -c juju status --color
```

En este caso antes hace falta configurar la url de acceso al panel central, de nuevo mediante *juju* es posible hacerlo. Dado que se usa *microk8s* y *metallb*, la ip siempre será *http://10.64.140.43.nip.io*

```
juju config dex-auth public-url=http://10.64.140.43.nip.io
juju config oidc-gatekeeper public-url=http://10.64.140.43.nip.io
```

En caso de que se use una instancia remota, es necesario configurar un proxy SOCKS a la máquina por ssh especificando el puerto en el que se establece esa conexión:

```
ssh -i -D 9999 <username>@<hostname>
```

En el ejemplo se ha establecido un proxy SOCKS en el puerto 9999 de la máquina local, lo que permitirá acceder a la url *http://10.64.140.43.nip.io* desde el navegador. Cabe mencionar que este proxy hará que todas las conexiones de internet locales para la interfaz por defecto lo usen como 'puente' a Internet. Esto podría generar tráfico en la máquina de destino y/o ralentizar la conexión a Internet del equipo local o incluso que ciertas páginas web piensen que el ordenador que se use esté teniendo un comportamiento malicioso o lo identifiquen erróneamente como un programa informático.

Tras tener el proxy configurado, se procede a establecer las credenciales de acceso al panel central:

```
juju config dex-auth static-username=<usuario>
juju config dex-auth static-password=<contraseña>
```

11.4. Instalación de Kubeflow en la nube

A la hora de obtener un entorno en la nube donde tener Kubeflow, hay variantes según diferentes factores a tener en cuenta como la escalabilidad, el precio y el mantenimiento, por ello se proponen 3 soluciones distintas:

- **Una sóla máquina** que el proveedor de servicios en la nube permita pagar de forma fija e incluso reservada (aprovisionamiento) pero con los recursos suficientes para tener cierto margen de escalado o simplemente para tareas que no son críticas, como querer usar las ventajas del entrenamiento pero realizar el despliegue mediante otros métodos (entrenar el modelo en Kubeflow y desplegarlo en Vertex AI, por ejemplo). Dado que muchos proveedores no permiten el pago mensual de máquinas que forman parte de un cluster Kubernetes, forzando el pago por hora, puede resultar más interesante contar con una máquina de por ejemplo 30GB RAM, 9 vCPU de pago mensual que 3 nodos con 10GB y 3 vCPU cada uno con un pago horario, que es por lo general, más caro. La idea detrás de esta

proposición es pagar un coste fijo que salga más barato que 3 nodos de uso horario, contando en ambas situaciones con los mismos recursos. Esto tiene sentido para un cluster pequeño, ya que para contar con alta disponibilidad es recomendable tener al menos un número impar de nodos mayor que 1, es decir, 3. Por otro lado, si sólo se necesita un despliegue modesto para tareas dónde se puede permitir esperar un tiempo con el correspondiente ahorro o para reducir los costes, usar una instancia puede ser ideal. En cualesquiera de los casos, la instalación se realizaría idénticamente a las formas explicadas con detalle en el punto 10.3, con el consecuente ajuste del firewall y las credenciales del panel central para una seguridad reforzada propia de un entorno de producción.

- **Una solución totalmente gestionada**: contar con una solución cuyo soporte recae sobre la empresa que lo ofrece, es decir se obtiene Kubeflow como servicio (*Kubeflow as a Service, KFaaS*) a usar, totalmente preparado en todo momento y configurado, actualizado y mantenido por un tercero. En caso de no contar con un equipo para realizar, mantener o actualizar Kubeflow, esta idea puede ser interesante, a costa de un pequeño sobrecoste que puede variar según el proveedor. En este caso la instalación no es necesaria pero se genera una dependencia directa con el distribuidor de la solución, que puede no mantener las últimas versiones de Kubeflow al día o con un retraso que afecte a la disponibilidad del uso de las últimas herramientas y mejoras que se ofrezcan en posteriores versiones. Por otro lado, en caso de incidencias, de nuevo se depende completamente del proveedor y su tiempo de respuesta para obtener una solución. Generalmente no se ofrece acceso directo al cluster en este casos, lo que implica que ciertas herramientas que inicialmente sólo se pueden usar vía definición de Kubernetes (usualmente, YAMLs) no podrían emplearse hasta que fuesen añadidas al panel central, que es el único punto de acceso a Kubeflow que se suele otorgar en estos casos.

- **Un cluster Kubernetes desplegado en cualquier proveedor**: En este caso los métodos ofrecidos en el punto 10.3 sirven también, con la diferencia de que se quita de la ecuación el uso de *microk8s*, pero se sigue usando *kubectl*, es decir, se empezaría la instalación de Kubeflow directamente. Dado que cada proveedor en la nube es un mundo y tiene sus propias medidas de seguridad, configuraciones propias y servicios integrados, es recomendable seguir las guías oficiales de Kubeflow [11] para instalaciones en clústeres de proveedores como Google Cloud, AWS, IBM, Azure, Oracle o VMWare los cuáles las han elaborado y las mantienen activamente. También se ofrece una instalación para un clúster de Kubernetes genérico, si bien, puede requerir de detalles o pasos adicionales dependiendo del proveedor que se decida usar. Por otro lado, se puede incurrir en costes adicionales complejos de cuantificar de forma predecible, como el coste por uso del tráfico de cada instancia o de soluciones propias que se requieran para sustituir algún componente interno, como es el caso de *Anthos* para sustituir a *Istio* cómo malla de servicios, en el caso de Google Cloud.

11.5. Objetivos de desarrollo sostenible

Objetivos de Desarrollo Sostenible	Alto	Medio	Bajo	No procede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**		X		
ODS 4. **Educación de calidad.**	X			
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**	X			
ODS 9. **Industria, innovación e infraestructuras.**		X		
ODS 10. **Reducción de las desigualdades.**				X
ODS 11. **Ciudades y comunidades sostenibles.**				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

ODS 3. Salud y bienestar

Las tecnologías y metodologías desarrolladas y descritas en el trabajo tienen el potencial de contribuir a este objetivo ya que las aplicaciones de la inteligencia artificial y el aprendizaje automático en el sector de la salud son vastas y van desde el diagnóstico médico asistido por IA, la personalización de tratamientos, hasta la gestión de sistemas de salud pública. Estas tecnologías pueden mejorar la precisión y eficiencia en el diagnóstico y tratamiento de enfermedades, facilitar la investigación y desarrollo de nuevos medicamentos, y optimizar la gestión y el acceso a los servicios de salud, contribuyendo así a mejorar la salud y el bienestar global.

ODS 4. Educación de calidad

Las tecnologías discutidas en este trabajo documento pueden usarse para mejorar significativamente la calidad y accesibilidad de la educación. La inteligencia artificial y el aprendizaje automático pueden personalizar la experiencia educativa, adaptándose a las necesidades de aprendizaje individuales, proporcionar plataformas educativas más interactivas y accesibles, y mejorar los métodos de evaluación y seguimiento del progreso del estudiante. Estas tecnologías ofrecen oportunidades para superar barreras geográficas y socioeconómicas, promoviendo una educación inclusiva, equitativa y de calidad para todos.

Por otro lado, este trabajo presenta un enfoque didáctico e iterativo que aspira a contribuir a la base de conocimiento ya existente sobre las herramientas disponibles para entrenar modelos de redes neuronales, a la par que incentivar su uso mediante la proposición de casos prácticos presentados de forma evolutiva y detallada con el fin de facilitar su aprendizaje.

ODS 8. Trabajo decente y crecimiento económico

El trabajo relacionado con las redes neuronales, como se describe en este trabajo, puede contribuir significativamente al Objetivo 8: Trabajo Decente y Crecimiento Económico de la Agenda 2030. Esto se debe a que el desarrollo y aplicación de estas tecnologías fomentan la innovación y la competitividad en diversos sectores económicos. Pueden crear nuevas oportunidades de empleo en el campo de la tecnología y mejorar la eficiencia y productividad en industrias existentes, lo cual es esencial para el crecimiento económico sostenible. Además, la aplicación de estas herramientas puede ayudar en la formación de habilidades y en la educación continua, preparando a la fuerza laboral para los trabajos del futuro y promoviendo el trabajo decente para todos.

ODS 9. Industria, innovación e infraestructuras

Este objetivo busca construir infraestructuras resilientes, promover la industrialización inclusiva y sostenible, y fomentar la innovación. La investigación y desarrollo en estas áreas tecnológicas son fundamentales para la creación de nuevas infraestructuras tecnológicas, la mejora de las existentes y la promoción de la innovación en diversos sectores industriales.

Por un lado, la implementación de sistemas basados en el uso de redes neuronales puede optimizar procesos industriales, hacer más eficientes las cadenas de suministro, y mejorar la calidad de los productos y servicios, contribuyendo así a la industrialización sostenible. Por otro lado, el avance en estas tecnologías fomenta el desarrollo de nuevas infraestructuras digitales, indispensables para la economía del conocimiento. Estas innovaciones tecnológicas no sólo impulsan el crecimiento económico, sino que también pueden hacer que las industrias sean más sostenibles y menos perjudiciales para el medio ambiente, alineándose con los principios de sostenibilidad y resiliencia promovidos por el ODS 9.

12. Bibliografía

[1] *Michael Nielsen et al., "Neural Networks and Deep Learning": http://neuralnetworksanddeeplearning.com/*

[2] *Universidad de Berlín, "Widrow-Hoff learning rule": https://www.geo.fu-berlin.de/en/v/soga-py/Machine-learning/Artifical-Neural-Networks/Widrow-Hoff-rule/index.html*

[3] *Najib J Majaj, Denis G. Pelli et al., "Deep learning—Using machine learning to study biological vision": https://www.researchgate.net/publication/329380147_Deep_learning-Using_machine_learning_to_study_biological_vision*

[4] *Wikipedia, "Principal Component Analysis": https://en.wikipedia.org/wiki/Principal_component_analysis*

[5] *Xavier Glorot, Yoshua Bengio et al., "Understanding the difficulty of training deep feedforward neural networks": https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf*

[6] *Repositorio de código con código fuente del trabajo: github.com/sg-gs/tfg.git*

[7] *Google, "About Vertex AI Neural Architecture Search": https://cloud.google.com/vertex-ai/docs/training/neural-architecture-search/overview*

[8] *Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, Quoc V. Le et al., "MnasNet: Platform-Aware Neural Architecture Search for Mobile"*

[9] *GKE pricing: https://cloud.google.com/kubernetes-engine/docs/concepts/plan-node-sizes#memory_and_cpu_reservations*

[10] *GKE quotas: https://cloud.google.com/kubernetes-engine/quotas?hl=es-419*

[11] *Installing Kubeflow: https://www.kubeflow.org/docs/started/installing-kubeflow/*

[12] *PYPL PopularitY of Programming Language: https://pypl.github.io/PYPL.html*

[13] *Precios en Vertex AI: https://cloud.google.com/vertex-ai/pricing*

[14] *ImageNet benchmark: paperswithcode.com/sota/image-classification-on-imagenet*

[15] *KServe Serving: kserve.github.io/website/0.10/modelserving/v1beta1/serving_runtime/*

[16] *Charmed Kubeflow, Canonical: https://charmed-kubeflow.io/docs/install*

[17] *DeployKF: deploykf.org*

[18] *Kubeflow manifests: github.com/kubeflow/manifests*

[19] *MicroK8S: microk8s.io/docs/install-alternative*