

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

**Diseño e implementación de un prototipo experimental para
aprendizaje automático distribuido**

Joel Sebastián Lucero Olalla

Ingeniería en Ciencias de la Computación

Trabajo de fin de carrera presentado como requisito
para la obtención del título de
Ingeniero en Ciencias de la Computación

Quito, 17 de febrero de 2022

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

HOJA DE CALIFICACIÓN DE TRABAJO DE FIN DE CARRERA

**Diseño e implementación de un prototipo experimental para
aprendizaje automático distribuido**

Joel Sebastián Lucero Olalla

Nombre del profesor, Título académico

Ricardo Flores, PhD

Quito, 17 de febrero de 2022

© DERECHOS DE AUTOR

Por medio del presente documento certifico que he leído todas las Políticas y Manuales de la Universidad San Francisco de Quito USFQ, incluyendo la Política de Propiedad Intelectual USFQ, y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo quedan sujetos a lo dispuesto en esas Políticas.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo en el repositorio virtual, de conformidad a lo dispuesto en la Ley Orgánica de Educación Superior del Ecuador.

Nombres y apellidos: Joel Sebastián Lucero Olalla

Código: 00201685

Cédula de identidad: 1722633367

Lugar y fecha: Quito, 17 de febrero de 2022

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en <http://bit.ly/COPETheses>.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on <http://bit.ly/COPETheses>.

RESUMEN

En la actualidad, varias empresas, investigadores y desarrolladores han optado por implementar dentro de sus sistemas, algoritmos de aprendizaje automático que les sea de ayuda al momento de tomar decisiones o realizar predicciones sobre un tema en específico. Para esto es necesario el procesamiento de grandes volúmenes de datos, lo cual puede resultar desafiante. Sin embargo, esto es posible mediante la implementación de un sistema computacional distribuido. El presente trabajo analiza los fundamentos de estos sistemas, así como las diferentes opciones que se pueden encontrar dentro del ecosistema del aprendizaje automático. Para ello, se desarrolló un prototipo que utiliza TensorFlow y Keras dentro de un escenario virtualizado que permite analizar en detalle el comportamiento de estos sistemas. Finalmente, se analizan los resultados y se presentan las ventajas de utilizar un sistema de aprendizaje automático distribuido.

Palabras clave: Aprendizaje automático distribuido, redes neuronales, aprendizaje profundo, TensorFlow, Keras.

ABSTRACT

Nowadays, several companies, researchers and developers have chosen to implement machine learning algorithms within their systems to help them when making decisions or making predictions on a specific topic. For this, the processing of large volumes of data is necessary, which is possible through the implementation of a distributed computing system. This work analyzes the fundamentals of these systems, as well as the different options that can be found within the machine learning ecosystem. To do this, a prototype was developed that uses TensorFlow and Keras within a virtualized scenario that allows a detailed analysis of the behavior of these systems. Finally, the results are analyzed and the advantages of using a distributed machine learning system are presented.

Keywords: Distributed Machine Learning, Neural Networks, Deep Learning, TensorFlow, Keras.

TABLA DE CONTENIDO

TABLA DE CONTENIDO	7
ÍNDICE DE FIGURAS	9
Introducción	10
CAPÍTULO 1: MARCO TEÓRICO	11
1.1 Machine Learning	11
1.2 Técnicas de Machine Learning	11
1.2.1 Supervisado	12
1.2.2 No supervisado	12
1.2.3 Semisupervisado	12
1.2.4 Refuerzo	12
1.3 Propósitos de los algoritmos de Machine Learning	13
1.3.1 Clasificación	13
1.3.2 Clustering	13
1.3.3 Reducción de dimensiones	13
1.3.4 Análisis de regresión	14
1.4 Escalamiento	14
1.4.1 Escalamiento vertical	14
1.4.2 Escalamiento horizontal	16
1.5 Sistemas computacionales distribuidos para Machine Learning	17
1.6 Estructura de un sistema distribuido	18
1.7 Frameworks disponibles	20
1.7.1 Frameworks de computación distribuida con propósito general	20
1.7.2 Frameworks especializados en Machine Learning distribuido	20
CAPÍTULO 2: PROTOTIPO	22
2.1 Descripción general del prototipo	22
2.2 Virtual Networks Over Linux (VNX)	22
2.3 TensorFlow y Keras	24
2.3.1 MultiWorker Mirrored Strategy	25
2.4 Descripción del modelo de Deep Learning	27
2.4.1 Dataset de entrenamiento	27

2.4.2	Redes neuronales convolucionales (VGG)	27
2.4.3	Redes neuronales recurrentes.....	29
CAPÍTULO 3: RESULTADOS		32
3.1	Ejecución del prototipo.....	32
3.2	Desempeño y evaluación del modelo	33
3.3	Rendimiento del entrenamiento distribuido	35
CAPÍTULO 4: CONCLUSIONES		40
4.1	Aprendizaje automático distribuido	40
4.2	Mejoras al modelo (trabajo futuro)	41
Referencias bibliográficas.....		43

ÍNDICE DE FIGURAS

Figura 1: Estructura de un CPU vs GPU (Pandey y Silakari, 2021)	15
Figura 2: Comparación de costos al implementar escalamiento vertical y horizontal (Kozlovski, 2018)	16
Figura 3: Arquitectura de un modelo de Machine Learning distribuido (Qian et al, 2019)	18
Figura 4: Estructuras más comunes en un sistema de Machine Learning distribuido (Verbraeken et al, 2020)	19
Figura 5: Topología de red utilizada para el prototipo	23
Figura 6: Declaración de los nodos en el código y configuración inicial.....	25
Figura 7: Llamada a la función que define el modelo y lo entrena	25
Figura 8: Algunos ejemplos de imágenes que contiene el dataset	27
Figura 9: Representación gráfica del proceso de filtrado que realiza la capa de convolución (Kim, 2017)	28
Figura 10: Arquitectura del modelo VGG16.....	29
Figura 11: Arquitectura de una neurona LSTM (Graves, 2013)	30
Figura 12: Arquitectura del modelo.....	31
Figura 13: Ejecución del modelo en la red virtualizada	32
Figura 14: Gráfica del puntaje BLEU vs Epochs para el dataset de prueba	34
Figura 15: Gráfico del rendimiento del entrenamiento distribuido con anchos de banda tradicionales.....	35
Figura 16: Líneas de comando que permiten limitar el ancho de banda de una interfaz en Linux	36
Figura 17: Resultados al utilizar la herramienta Iperf para un ancho de banda de 10Mbps	36
Figura 18: Gráfico del rendimiento del entrenamiento distribuido en una red de alto rendimiento	37
Figura 19: Uso computacional promedio durante el experimento con los nodos físicos	38
Figura 20: Ejemplo de los resultados del modelo.....	39

INTRODUCCIÓN

La necesidad de crear sistemas automáticos, seguros y confiables que sean capaces de tomar decisiones sin requerir de una directa intervención humana es cada vez mayor debido a los numerosos beneficios que estos traen consigo. Desde una eficaz optimización de recursos y la simplificación de una gran variedad de procesos, sistemas como estos pueden resultar muy atractivos dentro de muchas áreas gracias a su gran potencial.

Estos sistemas, para conseguir una mayor precisión al momento de generar resultados, requieren de grandes volúmenes de datos para ser entrenados, lo cual muchas veces puede resultar un problema, ya que un modelo de aprendizaje de gran escala necesita una cantidad considerable de capacidad computacional. Esta capacidad computacional puede ser conseguida mediante un escalamiento vertical, es decir, obteniendo la máxima capacidad de cómputo de un único nodo, o también mediante escalamiento horizontal, que básicamente se traduce en agregar más nodos al modelo.

El proceso de crear un modelo de aprendizaje que se ejecuta en más de un nodo de computación se conoce como Distributed Machine Learning, por su nombre en inglés. Este modelo distribuye la carga de analizar y procesar grandes datasets de información en diferentes dispositivos, lo cual puede traer muchas ventajas en comparación a un modelo tradicional. Desde un tiempo menor de ejecución hasta una capacidad de escalamiento mucho mayor, estos y más beneficios se pueden conseguir con un modelo distribuido siempre y cuando sea configurado de una manera correcta. Hoy en día, se puede encontrar una amplia variedad de opciones las cuales facilitan la implementación de sistemas de esta naturaleza, cada una con características que satisfacen diferentes necesidades.

CAPÍTULO 1: MARCO TEÓRICO

1.1 Machine Learning

Para entender el funcionamiento de un sistema distribuido de aprendizaje automático primero es necesario comprender que es el aprendizaje automático y cuáles son sus fundamentos, puesto que de estos conceptos depende el diseño de un modelo distribuido. El aprendizaje automático o Machine Learning es una rama de las Ciencias de la Computación que tiene como objetivo diseñar algoritmos que permitan simular de la manera más precisa posible, la inteligencia y el comportamiento humano en un sistema computacional (Bonaccorso, 2017). Por tanto, a través de un algoritmo de aprendizaje automático, una computadora es capaz de aprender del entorno que la rodea para así tomar una decisión y producir un resultado. Esta primera etapa de aprendizaje se la conoce como “entrenamiento” y consiste en proporcionarle al algoritmo diversos ejemplos de la información que tendrá que procesar, así como, del resultado deseado (dependiendo si el modelo es supervisado o no) (El Naqa y Murphy, 2015). Con estos ejemplos, el sistema deberá configurarse a sí mismo para producir un resultado esperado, no solo para la información proporcionada previamente, sino también para cualquier información nueva que le sea dada. El principal objetivo de cualquier sistema de aprendizaje automático distribuido es el de mejorar el rendimiento computacional en la etapa de aprendizaje o entrenamiento.

1.2 Técnicas de Machine Learning

El proceso descrito antes generalmente es implementado siguiendo una de las cuatro técnicas descritas a continuación. Es importante recalcar que cualquiera de estas puede ser aplicada dentro de un sistema distribuido.

1.2.1 Supervisado

Para esta técnica los conjuntos de datos utilizados para entrenar el modelo están etiquetados con sus respectivos valores de salida, por lo que el objetivo del algoritmo es el de inferir una función que dé como resultado uno de estos valores de salida. Se utiliza principalmente para la clasificación de datos.

1.2.2 No supervisado

A diferencia del aprendizaje supervisado, el no supervisado utiliza datos cuyos valores de salida no han sido especificados. El propósito de este algoritmo, por lo general, es el de analizar la estructura de los datos, agruparlos por resultados, identificar patrones, entre otros.

1.2.3 Semisupervisado

Este tipo de aprendizaje puede ser definido como una combinación de los dos métodos descritos anteriormente, lo que significa que el modelo es entrenado tanto con datos etiquetados con sus valores de salida como con datos sin etiquetar (Sarker, 2021). De esta forma, el aprendizaje toma lugar de manera muy similar a lo que ocurre en la vida real, por lo que puede llegar a ser muy útil para diversas aplicaciones.

1.2.4 Refuerzo

La técnica de aprendizaje por refuerzo evalúa el comportamiento y las características de su entorno para mejorar la eficiencia de este. El estado en un determinado momento del entorno es evaluado mediante una función y dependiendo del puntaje, es recompensado o penalizado (Sarker, 2021). Este método tiene muchas implementaciones dentro de la robótica y en el desarrollo de sistemas autónomos.

1.3 Propósitos de los algoritmos de Machine Learning

Los algoritmos de Machine Learning pueden ser clasificados por el tipo de resultado que estos producen. Esta clasificación tiene una estrecha relación con el tipo de aprendizaje utilizado para el modelo. A pesar de que un sistema distribuido puede ser aplicado a varios tipos de modelos de Machine Learning, su implementación es más común en modelos que están basados en redes neuronales, las cuales son muy útiles al momento de clasificar datos o realizar predicciones.

1.3.1 Clasificación

Este tipo de algoritmo se consigue implementando la técnica de aprendizaje supervisado y consiste en clasificar la información entrante en categorías aprendidas durante la etapa de entrenamiento. Adicionalmente, existen diferentes formas de clasificar los datos: de forma binaria (cada dato puede ser clasificado en únicamente una de dos categorías), multiclases (existen más de dos categorías para la clasificación) y multi etiqueta (donde un dato puede ser clasificado en varias categorías) (Sarker, 2021).

1.3.2 Clustering

El propósito de un algoritmo de clustering es el de identificar y agrupar datos que tengan características similares entre sí dada una métrica específica. Este algoritmo implementa la técnica de aprendizaje no supervisado y generalmente se utiliza para procesar grandes cantidades de datos.

1.3.3 Reducción de dimensiones

Su objetivo es el de simplificar información mediante la eliminación de variables con poca relevancia dentro de un conjunto de datos de entrada. Esto se puede realizar de dos maneras: seleccionando únicamente las variables relevantes o reemplazando varias variables en una sola que

pueda representarlas (Sarker, 2021). Es útil ya que tiene el potencial de simplificar modelos, evitar redundancia, disminuir el costo computacional, entre otras.

1.3.4 Análisis de regresión

El análisis de regresión es capaz de predecir el valor que adquiere una variable dependiente si se cambia el valor de las demás variables del entorno (Sarker, 2021). Este algoritmo tiene importantes aplicaciones dentro de cualquier campo donde la predicción es fundamental, como la economía o la meteorología.

1.4 Escalamiento

Un algoritmo de Machine Learning puede ser escalado tanto mediante software como hardware. El escalamiento mediante software se logra al implementar métodos matemáticos como el método de gradiente estocástico descrito por Bottou et al (2018) el cual, entre otras cosas, tiene la capacidad de mejorar las tasas de convergencia, superar los efectos adversos de la no linealidad de los datos y el mal acondicionamiento. Por otro lado, un modelo de Machine Learning también puede mejorar su rendimiento y alcance realizando cambios en el sistema computacional que lo ejecuta. Dentro de estos cambios se identifican dos tipos: escalamiento vertical y escalamiento horizontal. Ambas configuraciones pueden implementarse en conjunto para obtener un máximo rendimiento.

1.4.1 Escalamiento vertical

El escalamiento vertical consiste en modificar la configuración de un único nodo computacional para mejorar su rendimiento. Uno de los métodos más comunes es el de agregar GPUs programables al sistema ya que estas unidades cuentan con un número de núcleos mucho mayor comparado con una CPU (Figura 1), lo que le permite ejecutar algoritmos de Machine

Learning con un rendimiento superior y ofrecer mejores resultados al momento de acceder a memoria y realizar operaciones aritméticas (Pandey y Silakari, 2021). Una GPU está construida con bloques conocidos como Streaming Multiprocessor (SM) y estos a su vez están formados por muchas ALUs o CUDA-Cores, como se los conoce en GPUs de la marca NVidia. Estos bloques SM se encargan de ejecutar un warp a la vez (Schlegel, 2015). Un warp es un paquete de 32 hilos en donde todos los hilos realizan la misma operación y tienen la propiedad de utilizar el máximo ancho de banda al acceder a la memoria principal si se ejecutan de forma simultánea. Adicionalmente, una GPU puede cambiar de hilos en cada ciclo de reloj, lo cual sumado a su eficiente uso de memoria explicado anteriormente le permiten a la GPU no depender de la latencia de la memoria, como un CPU, sino más bien del ancho de banda disponible (Schlegel, 2015).

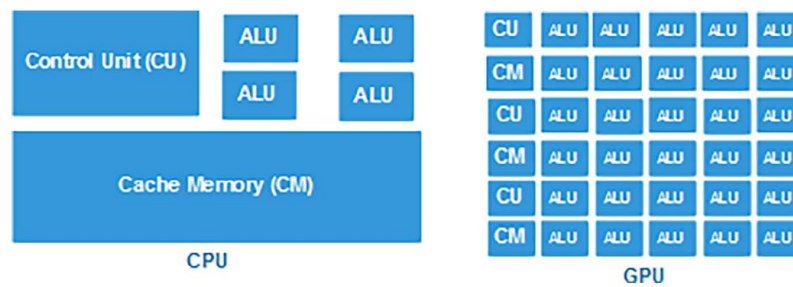


Figura 1: Estructura de un CPU vs GPU

Otra alternativa al uso de GPUs, es utilizar circuitos integrados de aplicación específica (ASICs). Estos circuitos tienen la propiedad de ofrecer un alto rendimiento acompañado de un eficiente consumo de energía y cuando son aplicados, por ejemplo, al cálculo y procesamiento de matrices se puede obtener resultados muchos mejores que al utilizar CPUs o GPUs, tal y como lo hace Google con su Tensor Processor Unit (TPU) (Verbraeken et al, 2020). La arquitectura de este tipo de circuitos está optimizada específicamente para el procesamiento de matrices y esta es diferente dependiendo de cada fabricante. Google, por ejemplo, fabricó en 2019 su Edge TPU, la

cual está formada por una arquitectura de matriz sistólica que permite realizar la multiplicación de matrices de forma muy eficiente y en gran escala, con la capacidad de realizar 4 trillones de operaciones por segundo (Hosseininoorbin, 2021). Estos circuitos no deben confundirse con los de compuertas lógicas programables (FPGAs), puesto que estos últimos tienen la propiedad de tener circuitos programables lo que les da mayor flexibilidad en cuanto al tipo de tareas que son capaces de realizar. Esta característica les da ventaja en el mercado frente a los ASIC, pero supone una desventaja en cuanto a rendimiento y consumo de energía comparado con estos (Kuon, 2007).

1.4.2 Escalamiento horizontal

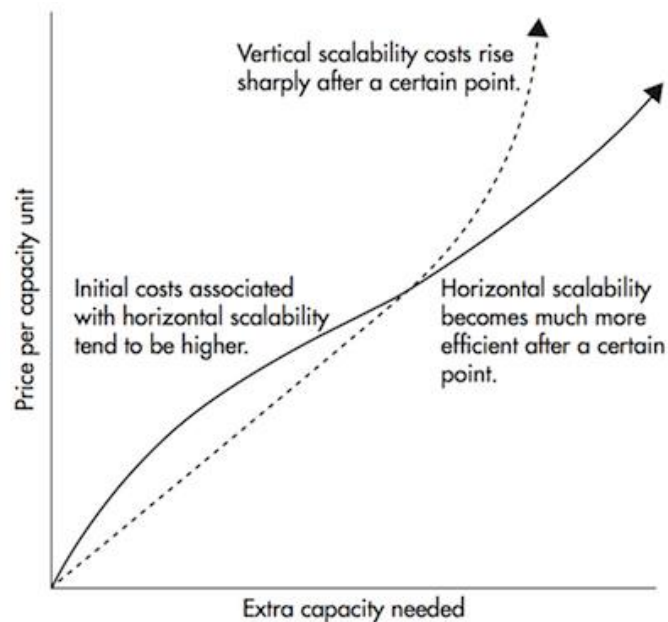


Figura 2: Comparación de costos al implementar escalamiento vertical y horizontal

Añadir más nodos computacionales a un sistema que ejecuta un algoritmo de Machine Learning se lo conoce como escalamiento horizontal. Si un algoritmo puede ser adaptado para lograr una eficiente paralelización, es posible construir un sistema distribuido que ofrezca muchos beneficios sobre uno tradicional. El escalamiento horizontal ofrece muchas ventajas como son: bajos costos de inversión y mantenimiento (dependiendo de las características del sistema), una

mejor seguridad ante fallos y un ancho de banda mayor al momento de procesar datos (Verbraeken et al, 2020). Como se observa en la Figura 2, el costo inicial asociado al escalar un sistema horizontalmente es, por lo general, mayor comparado a un escalamiento vertical. No obstante, hay un punto en donde agregar mayor capacidad computacional se vuelve económicamente más efectivo aplicando un escalamiento horizontal (Kozlovski, 2018).

1.5 Sistemas computacionales distribuidos para Machine Learning

La eficiencia de un modelo de Machine Learning depende mucho de la cantidad de datos que se disponga para su entrenamiento, hoy en día esto no es un impedimento ya que cualquier tipo de información proveniente de todo el mundo es fácilmente accesible. El factor limitante, sin embargo, se encuentra en los sistemas computacionales tradicionales y en los algoritmos de aprendizaje que son incapaces de procesar dicha cantidad de información (Peteiro-Barral, 2013). Es por esto que sistemas con más de un nodo computacional se crearon como solución a este problema, los cuales además, tienen la capacidad de ofrecer un mejor manejo de conjuntos de datos cuya naturaleza es distribuida.

Para ejecutar un algoritmo de Machine Learning distribuido es necesario que el conjunto de datos se encuentre distribuido para realizar el entrenamiento en diferentes procesos. Esta distribución puede llevarse a cabo de dos diferentes formas: horizontalmente, dividiendo los datos en subconjuntos de instancias, o verticalmente, dividiéndolo en base a los atributos de las instancias (Peteiro-Barral, 2013). Distribuir el conjunto de datos trae consigo varios beneficios como lo son un menor costo computacional al momento de procesar estos datos u ofrecer una mayor seguridad de la información. El proceso de aprendizaje en subconjuntos del conjunto original también tiene ventajas sobre un modelo tradicional, ya que se puede conseguir una mayor

precisión en los resultados, un menor costo computacional, tolerancia a ciertos tipos de fallo y una mayor escalabilidad (Peteiro-Barral, 2013).

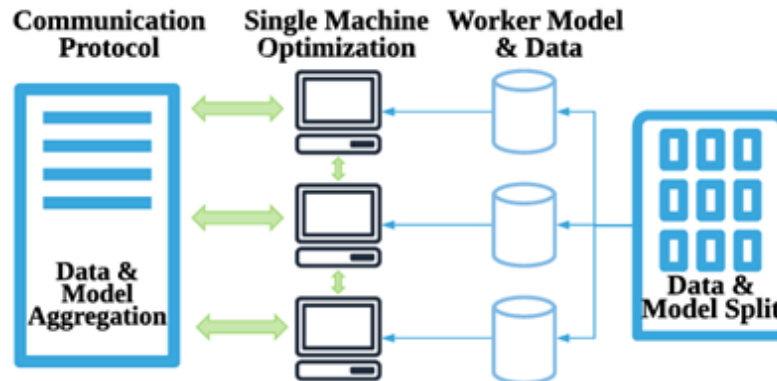


Figura 3: Arquitectura de un modelo de Machine Learning distribuido

La Figura 3 muestra el flujo general de operación de un sistema de Machine Learning distribuido. Se observa que primero, tanto el modelo como los datos son repartidos entre los diferentes trabajadores y una vez aquí, el modelo es optimizado de acuerdo con las especificaciones de hardware de cada nodo (Qian et al, 2019). Con una configuración de red correcta, las máquinas se comunican entre sí para coordinar y llevar a cabo el entrenamiento.

1.6 Estructura de un sistema distribuido

Un punto muy importante para tomar en cuenta al momento de desarrollar un sistema de Machine Learning distribuido es la arquitectura que tendrá el mismo. Para definir esto, se debe analizar varios factores tales como el tipo de algoritmo a utilizar o el nivel de distribución deseado. Los tipos de estructura más comunes son: árboles, anillos, servidor de parámetros y peer-to-peer (Figura 4).

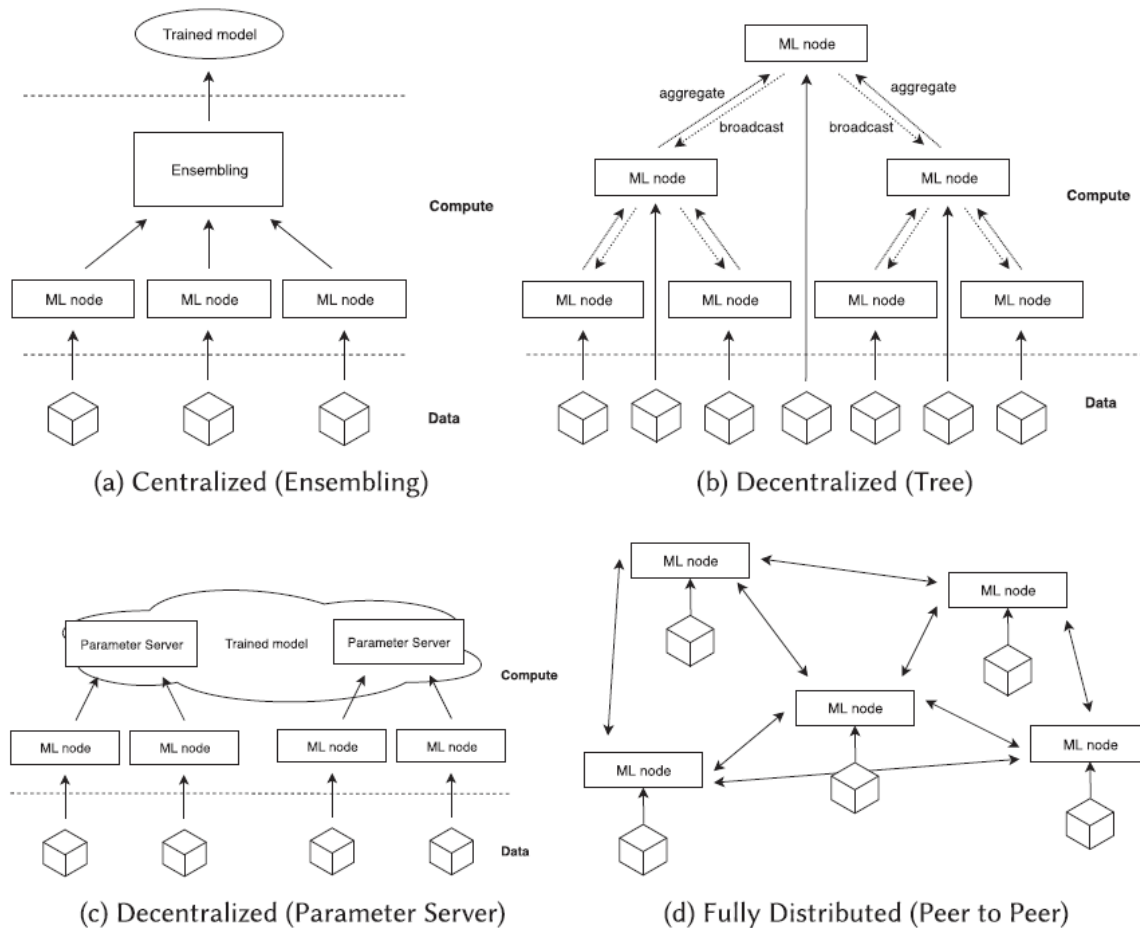


Figura 4: Estructuras más comunes en un sistema de Machine Learning distribuido

- Servidor de parámetros: En este esquema, los datos son distribuidos entre los diferentes nodos clientes mientras que los nodos servidor mantienen parámetros de la arquitectura que son compartidos globalmente (Li et al, 2013).
- Peer-to-peer: Todos los nodos se conectan directamente entre sí y todos ellos almacenan la información de los parámetros del modelo. Tiene como ventajas una fácil escalabilidad y un manejo de errores eficiente (en caso de que un nodo falle) (Verbraeken et al, 2020).
- Estructura de árboles: Basado en una arquitectura por niveles, la principal ventaja de este tipo de estructura es su capacidad de escalabilidad y administración, dado que la comunicación entre nodos se lleva a cabo entre padres e hijos.

- Anillos: La comunicación en esta estructura se da de forma simple, ya que un nodo está conectado únicamente a nodos vecinos, por lo tanto, solo puede intercambiar información con ellos. Se utiliza cuando la interacción entre nodos es mínima (Verbraeken et al, 2020).

1.7 Frameworks disponibles

Dentro del ecosistema de Machine Learning se han creado una serie de herramientas que facilitan el proceso de desarrollar un sistema distribuido. Muchas de estas herramientas fueron diseñadas con algoritmos que permiten el fácil manejo entre nodos computacionales, así como la comunicación entre ellos. Otras, sin embargo, fueron inicialmente creadas para ser ejecutadas en un único nodo, pero ante la necesidad de procesar grandes cantidades de datos se han adaptado para ser utilizadas en un sistema distribuido.

1.7.1 Frameworks de computación distribuida con propósito general

- MapReduce (Google): Modelo de programación que permite procesar grandes cantidades de información de forma paralela. Es dependiente del sistema de distribución de archivos ya que, en cada etapa, los datos son divididos en tuplas y procesados por los varios nodos para luego ser reducidos y agrupados (Google, 2021).
- Apache Spark: Extiende el modelo de MapReduce para soportar mayor variedad de procesos computacionales. Es capaz de realizar operaciones en los datos directamente desde la memoria principal, por lo cual ofrece una gran rapidez (Karau et al, 2015).

1.7.2 Frameworks especializados en Machine Learning distribuido

- Baidu AllReduce: Utiliza técnicas de computación de alto rendimiento para entrenar de manera iterativa modelos de descenso de gradientes estocásticos con diferentes segmentos pequeños del conjunto total de datos de entrenamiento (Verbraeken et al, 2020).

- Horovod: Permite implementar entrenamiento distribuido en modelos de Deep Learning con frameworks como TensorFlow, Keras y PyTorch. Esto lo hace añadiendo una capa basada en entrenamiento AllReduce que utiliza una Interfaz de Paso de Mensajes (MPI por sus siglas en inglés) y además, logra una mejor eficiencia cuando se entrena utilizando GPUs (Sergeev, 2017). Este framework, al igual que Baidu, no provee un sistema para tolerar fallo en los nodos computacionales (Verbraeken et al, 2020).
- Caffe2: Desarrollado por Facebook, este framework se fusionó con PyTorch (igualmente desarrollado por Facebook) para fusionar la flexible experiencia de usuario que este ofrece con las capacidades de Caffe2, por lo que ahora PyTorch funciona como frontend de Caffe2. De igual manera utiliza algoritmos AllReduce para distribuir el procesamiento y esto lo hace usando NCCL (NVIDIA Collective Communication Library) entre GPUs en un solo host y código personalizado entre hosts basados en la librería Gloo de Facebook (Verbraeken et al, 2020). Esta librería contiene diversos algoritmos de comunicación útiles para la creación de aplicaciones de Machine Learning (Facebook, 2017).
- TensorFlow: usa grafos de flujo de datos para representar el cálculo, estado compartido y las operaciones que mutan ese estado. Mapea los nodos de un gráfico de flujo de datos en muchas máquinas en un clúster y dentro de una máquina a través de múltiples dispositivos computacionales, incluidas CPU multinúcleo, GPU de uso general y ASIC de diseño personalizado conocidos como Unidades de Procesamiento de Tensor (TPU) (Abadi, 2016).

De los frameworks expuestos anteriormente, Tensorflow es uno de los que más se destaca en cuanto a la comunidad de soporte que se puede encontrar actualmente, aunque su redimiendo en cuanto a rapidez no sea el mejor (Qian et al, 2019).

CAPÍTULO 2: PROTOTIPO

2.1 Descripción general del prototipo

El prototipo realizado consta de dos partes fundamentales: el hardware utilizado para entrenar el modelo y el software elegido para desarrollar el modelo. En cuanto al hardware se usó una red virtualizada de computadoras utilizando la herramienta VNX y luego, tras haber completado el desarrollo del modelo y corregido varios errores, se repitió el experimento en un entorno físico para comprender más a profundidad su funcionamiento. Para la parte de software, el lenguaje utilizado fue Python y el framework escogido fue TensorFlow debido a que este es uno de los que cuenta con la mejor documentación, una amplia comunidad de soporte y una amigable experiencia para el usuario. Cada componente es explicado a detalle en las siguientes secciones.

2.2 Virtual Networks Over Linux (VNX)

VNX es una herramienta de virtualización multi propósito que permite simular redes de computadores de forma automática. La topología de dichas redes es definida por el usuario y están formadas por diferentes máquinas virtuales, cada una ejecutando su propio sistema operativo. Consta de dos partes esenciales: el programa de VNX instalado en un computador o servidor Linux y un archivo en formato XML donde el usuario define todas las características de su escenario virtual (VNX, 2020).

Esta herramienta fue utilizada para desplegar el sistema de Machine Learning distribuido siguiendo la topología definida en la Figura 5. La motivación para utilizar un entorno virtualizado para el desarrollo del prototipo se fundamenta en las siguientes razones: tener un control completo sobre la red lo que permite visualizar el comportamiento del modelo al variar ciertos parámetros,

conveniencia en cuanto a portabilidad y costo económico y por último, gran facilidad al momento de realizar pruebas y corregir errores dentro del modelo.

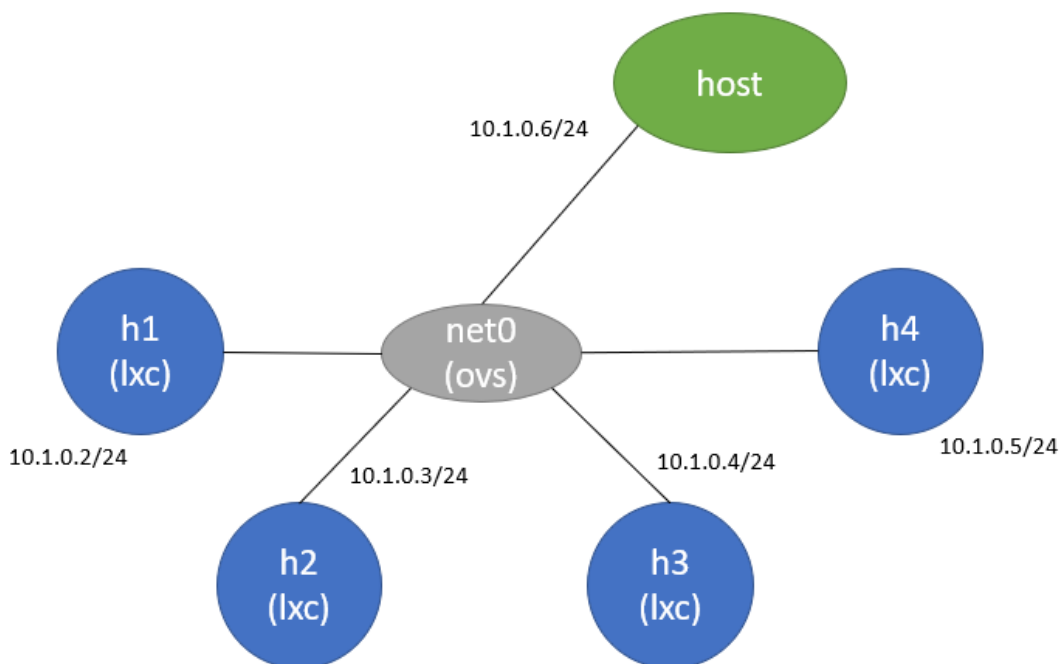


Figura 5: Topología de red utilizada para el prototipo

El escenario virtual de la Figura 5 se compone de cuatro máquinas virtuales, cada una con su propia dirección IP, interconectadas entre sí dentro de una misma red, permitiendo de esta forma que todos los nodos puedan comunicarse entre sí. Todas las máquinas virtuales están desplegadas en un contenedor Linux (lxc) y ejecutan una imagen personalizada de Ubuntu Server 20.04 (64-bits), la cual cuenta con TensorFlow y demás paquetes necesarios para el funcionamiento del prototipo. VNX también permite copiar archivos de la máquina host a las máquinas virtuales de forma automática al momento de iniciar el escenario virtual.

La transferencia de datos dentro de la red es manejada mediante un dispositivo de capa 2 open vSwitch (ovs), el cual básicamente es un switch virtual que maneja una tabla de direcciones MAC de todos los dispositivos conectados. Todas las máquinas virtuales están conectadas entre sí

mediante este dispositivo. Por otro lado, no se consideró la interconexión con entre diferentes redes ya que el prototipo no requiere recibir paquetes enrutados mediante un dispositivo de capa 3, por lo que no fue necesario la utilización de este. Esto significa que los dispositivos únicamente se pueden comunicar con otros dispositivos que se encuentren dentro de la red LAN, mas no con una red externa. Por otro lado, en cuanto al tipo de máquinas virtuales, también existía la posibilidad escoger máquinas virtuales basadas en kernel (kvm) y así lograr una virtualización completa, pero se eligieron máquinas de tipo lxc debido a que estas son muy ligeras y no tienen mucho impacto en el rendimiento del sistema.

2.3 TensorFlow y Keras

Como se analizó en la sección anterior, TensorFlow es un framework desarrollado por Google que soporta Machine Learning Distribuido de forma nativa. Para esto ofrece dos estrategias, cada una con características diferentes: `MultiWorkerMirroredStrategy` y `ParameterServerStrategy`. Ambas ofrecen una eficiente tolerancia al fallo al tener la posibilidad de guardar el progreso del entrenamiento del modelo tras cada iteración mediante llamadas a funciones que pueden ser definidas por el usuario.

Cuando se detecta múltiples dispositivos, es decir, más de un CPU o GPU, TensorFlow realiza un procedimiento conocido como “ubicación de nodos”, mediante el cual estima el costo computacional de ejecutar una operación en todos los dispositivos disponibles y así asignar tareas adecuadas para cada uno (Zhang et al, 2017).

La ejecución de TensorFlow consiste en tres componentes principales: cliente, jefe y trabajadores. El cliente se encarga de definir el grafo computacional a usar y de mantener la sesión donde se está trabajando con este grafo. Un grafo computacional o de flujo de datos es un grafo de tipo dirigido donde los vértices o nodos son operaciones, mientras que los bordes representan datos

que fluyen entre estas operaciones (Goldsborough, 2016). El cliente puede pedir una evaluación del grafo de TensorFlow a través de un objeto de sesión. Una vez recibida la petición, el servicio jefe programa y ordena a dos o más trabajadores la ejecución del grafo. Para la parte de cliente se utilizó Keras, framework que provee de bloques para la construcción de modelos de Deep Learning. Puede soportar tanto computación en CPU como en GPU (Ketkar, 2017).

```
tf_config = {
    'cluster': {
        'worker': ['10.1.0.2:50501', '10.1.0.3:50501', '10.1.0.4:50501', '10.1.0.5:50501']
    },
    'task': {'type': 'worker', 'index': int(sys.argv[1])}
}
os.environ['TF_CONFIG'] = json.dumps(tf_config)
per_worker_batch_size = 32
num_workers = len(tf_config['cluster']['worker'])
strategy = tf.distribute.MultiWorkerMirroredStrategy()
```

Figura 6: Declaración de los nodos en el código y configuración inicial

Como se puede observar en la Figura 7, es necesario realizar la definición del modelo, así como el llamado a la función que entrena el mismo, dentro del alcance de la estrategia declarada anteriormente (Figura 6). Las líneas de código mostradas en la Figura 6 y 7 son las únicas necesarias para indicar a TensorFlow que el modelo se entrenará de forma distribuida.

```
with strategy.scope():
    multi_worker_model = define_model(vocab_size, max_length)
```

Figura 7: Llamada a la función que define el modelo y lo entrena

2.3.1 MultiWorker Mirrored Strategy

Esta clase perteneciente a la API de entrenamiento distribuido de TensorFlow ofrece la posibilidad de entrenar modelos de Deep Learning en varios nodos computacionales. La

declaración de estos nodos se realiza al asignar un valor a la variable de entorno “TF_CONFIG”, la cual se almacena como formato JSON y contiene tanto las direcciones IP de los nodos, así como los puertos en donde el servidor esperará la conexión. El nodo que se declare en primer lugar será el nodo “jefe” y todos los demás serán “trabajadores”. El nodo jefe es al mismo tiempo un trabajador, sin embargo, este debe realizar más trabajo ya que también se encarga de coordinar las tareas de los demás.

Al utilizar esta clase, el entrenamiento se realiza de forma sincrónica donde cada nodo contiene una copia de los parámetros utilizados en el sistema (Zhang et al, 2017). Esto difiere de una estructura de servidor de parámetros ya que esta, al tener los parámetros del sistema almacenados en un único servidor, el entrenamiento se realiza de forma asincrónica. TensorFlow también proporciona la posibilidad de realizar entrenamiento distribuido utilizando este método con su clase ParameterServerStrategy, la cual está actualmente en continuo desarrollo.

TensorFlow realiza la comunicación entre los nodos utilizando el protocolo gRPC sobre TCP/IP y RDMA en conexiones de alto rendimiento como InfiniBand. El primero, Remote Procedure Call (gRPC, donde la g corresponde a Google), es utilizado tanto para la comunicación de tareas administrativas como para el intercambio de parámetros del modelo (Biswas et al, 2018). Este protocolo trabaja sobre Protocol Buffers, los cuales son formatos de intercambio de datos multiplataforma que se encargan de determinar la estructura y tipos de mensajes que se están enviando durante la comunicación. El núcleo de gRPC maneja esta comunicación encapsulando un canal en procesos de lectura y escritura definidas para un transporte específico, por ejemplo, TCP/IP o UDP (Biswas et al, 2018). Remote Direct Access Memory (RDMA), por otro lado, es un método para acceder a la memoria principal de un sistema sin alterar los procesos de su CPU, por lo que no necesita intervención del sistema operativo (Guo et al, 2016).

2.4 Descripción del modelo de Deep Learning

Las herramientas descritas anteriormente se utilizaron para construir un modelo de Deep Learning que implementa redes neuronales para generar subtítulos que describan los elementos que componen una imagen.

2.4.1 Dataset de entrenamiento

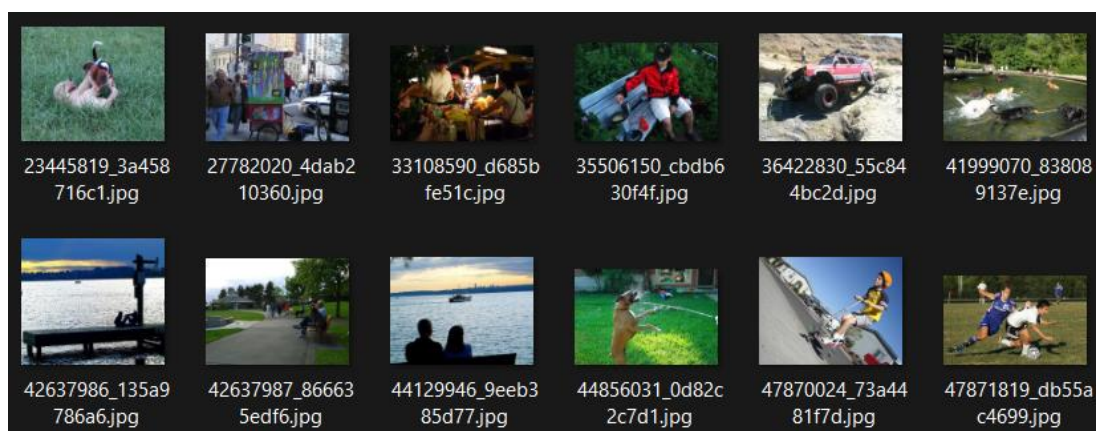


Figura 8: Algunos ejemplos de imágenes que contiene el dataset

El conjunto de datos elegidos para el entrenamiento fue Flickr8k_Dataset¹, el cual contiene alrededor de 8 000 imágenes, cada una relacionada con hasta cinco oraciones que describen claramente los eventos y las partes que componen la imagen. En la Figura 8 se puede observar que cada imagen está en formato JPG y tiene como nombre un identificador único que la diferencia de las demás. Las descripciones se encuentran en un archivo TXT independiente.

2.4.2 Redes neuronales convolucionales (VGG)

Para el procesamiento de las imágenes se emplearon redes neuronales convolucionales utilizando el modelo Grupo de Geometría Visual de Oxford (VGG). Las redes neuronales convolucionales son ampliamente utilizadas en el reconocimiento de patrones, por lo que tienen una gran utilidad para procesar imágenes y reconocimiento de voz (Kim, 2017). Dentro del

¹ Hodosh, M., Young, P., Hockenmaier J. (2013). Flickr8k_Dataset [Archivo de datos]. Obtenido de <https://academictorrents.com/details/9dea07ba660a722ae1008c4c8afdd303b6f6e53b>

prototipo se utilizaron para extraer las características de las imágenes para así poder representarlas como un vector. La forma en que este tipo de redes neuronales extraen características de una imagen es mediante la capa de convolución la cual aplica filtros a la misma para acentuar únicamente las características deseadas.

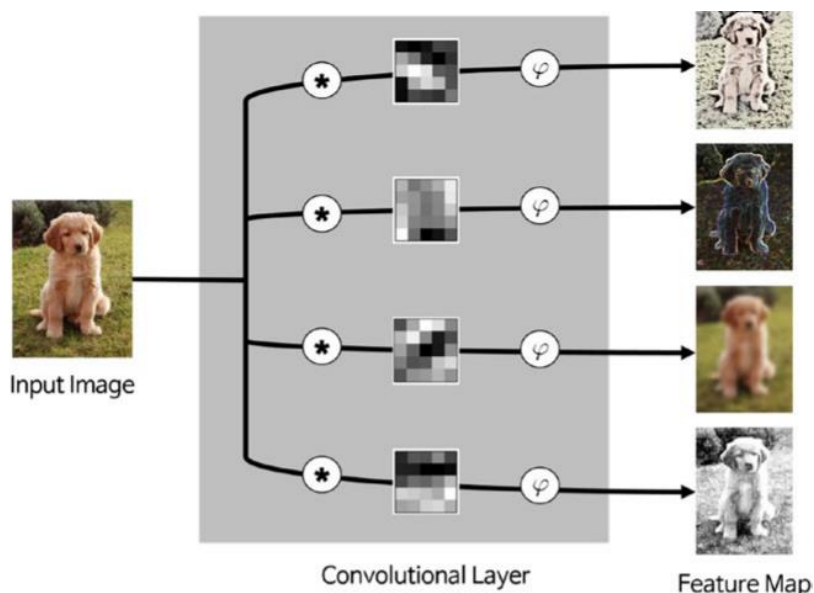


Figura 9: Representación gráfica del proceso de filtrado que realiza la capa de convolución

En la Figura 9, el símbolo ϕ representa la función de activación de las neuronas y el cuadrado con píxeles a blanco y negro representa los filtros de convolución. Estos filtros son matrices de dos dimensiones que se establecen en el proceso de entrenamiento. Después del proceso de convolución se da un proceso de agrupamiento mediante el cual se reduce el tamaño de la imagen ya que agrupa píxeles cercanos para así representarlos como un único valor.

El modelo VGG16 fue propuesto por K. Simonyan y A. Zisserman en su investigación “Very Deep Convolutional Networks for Large-Scale Image Recognition”. Contiene 16 capas las cuales varían entre capas de convolución y agrupación y se relacionan como se muestra en la Figura 10 (Perumanoor, 2021).

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 150, 150, 3)	0
block1_conv1 (Convolution2D)	(None, 150, 150, 64)	1792
block1_conv2 (Convolution2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Convolution2D)	(None, 75, 75, 128)	73856
block2_conv2 (Convolution2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Convolution2D)	(None, 37, 37, 256)	295168
block3_conv2 (Convolution2D)	(None, 37, 37, 256)	590080
block3_conv3 (Convolution2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Convolution2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Convolution2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Convolution2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Convolution2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Convolution2D)	(None, 9, 9, 512)	2359808

Figura 10: Arquitectura del modelo VGG16

2.4.3 Redes neuronales recurrentes

Las redes neuronales recurrentes son muy útiles al tratar datos que se presentan como secuencia, es decir, datos en donde el valor actual depende de los valores anteriores. Es por esto que este modelo se utiliza para predicción en series de tiempo, procesamiento de texto o reconocimiento de voz (Graves, 2013). Este tipo de redes neurales mantienen “memoria” de los datos anteriores para obtener nuevos resultados y lo hace implementado celdas de memoria. En el caso de LSTM (Long-Short Term Memory), las neuronas cuentan con cuatro compuertas que deciden la manera en que se manejará esta memoria. Estas cuatro compuertas son: Forget Gate,

para decidir si se borra la celda de memoria o no; Input Gate, para decidir si se escribe en la celda; Gate Gate, para decidir cuánto se va a escribir y Output gate, para decidir cuánto se va a revelar (Figura 11).

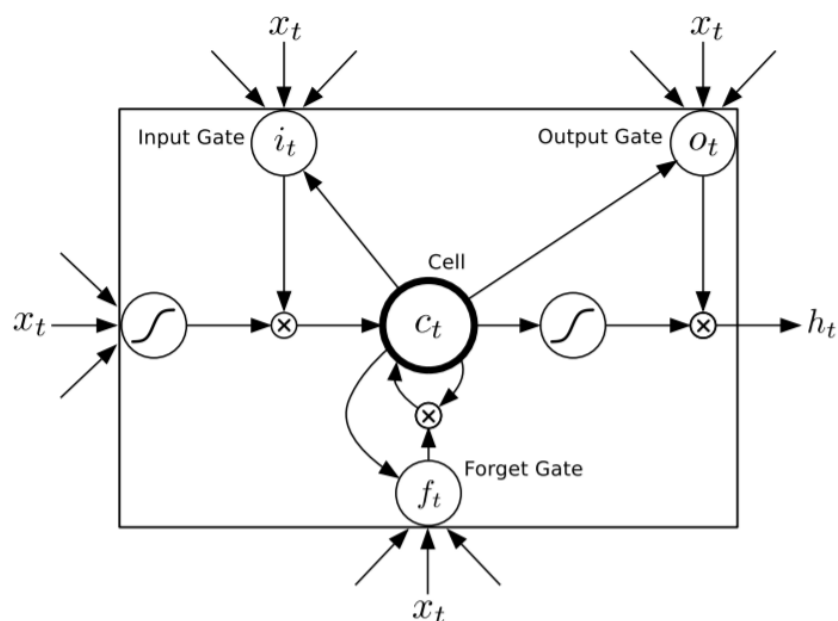


Figura 11: Arquitectura de una neurona LSTM

La Figura 12 muestra el modelo que se utilizó en el prototipo para procesar las características y las descripciones de las imágenes para poder generar descripciones de imágenes que el modelo no ha procesado anteriormente. El modelo se ramifica en dos partes, una que procesa el texto ya codificado (izquierda) y la otra que procesa las características de las imágenes extraídas con el modelo pre-entrenado VGG16 (derecha).

La rama que procesa el texto comienza implementando una capa Input la cual recibe las secuencias de los índices de las palabras de cada descripción para luego ser mapeadas por una capa Embedding la cual ignora los valores rellenos en las instancias y da como resultado un vector de 256 elementos de longitud. Posteriormente, el resultado anterior se procesa mediante una capa LSTM. Por otro lado, el proceso de procesamiento de las características de las imágenes recibe

como entrada vectores de 4096 de tamaño que son luego procesados por una capa Dense para obtener una representación de la foto. Ambas ramas del modelo implementan una capa Dropout con una regulación de 0.5 para evitar el sobreajuste (overfitting). Por último, el resultado de ambos procesos es agrupado mediante una capa Add y el resultado de esto es procesado por otras dos capas Dense para realizar la predicción.

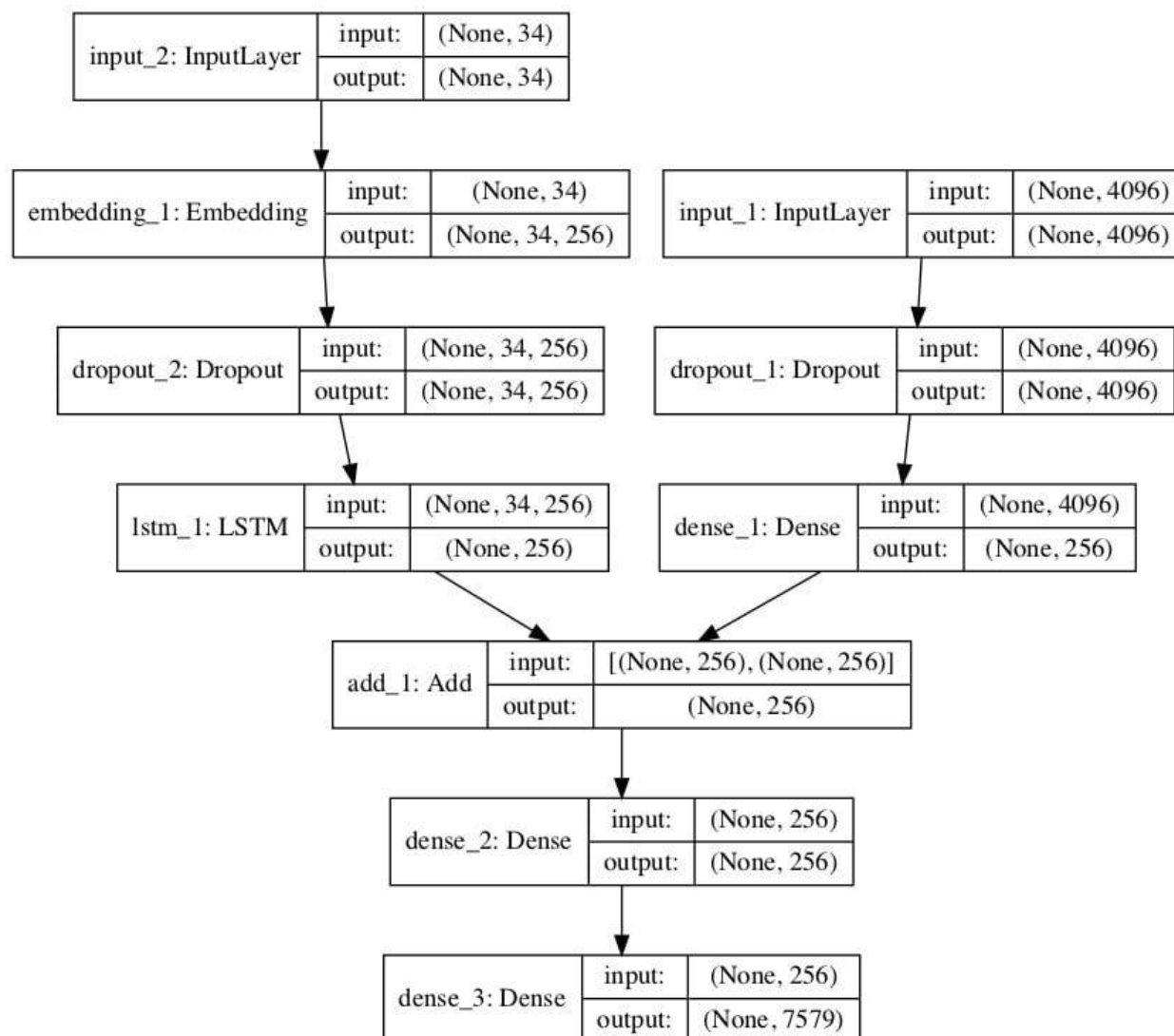
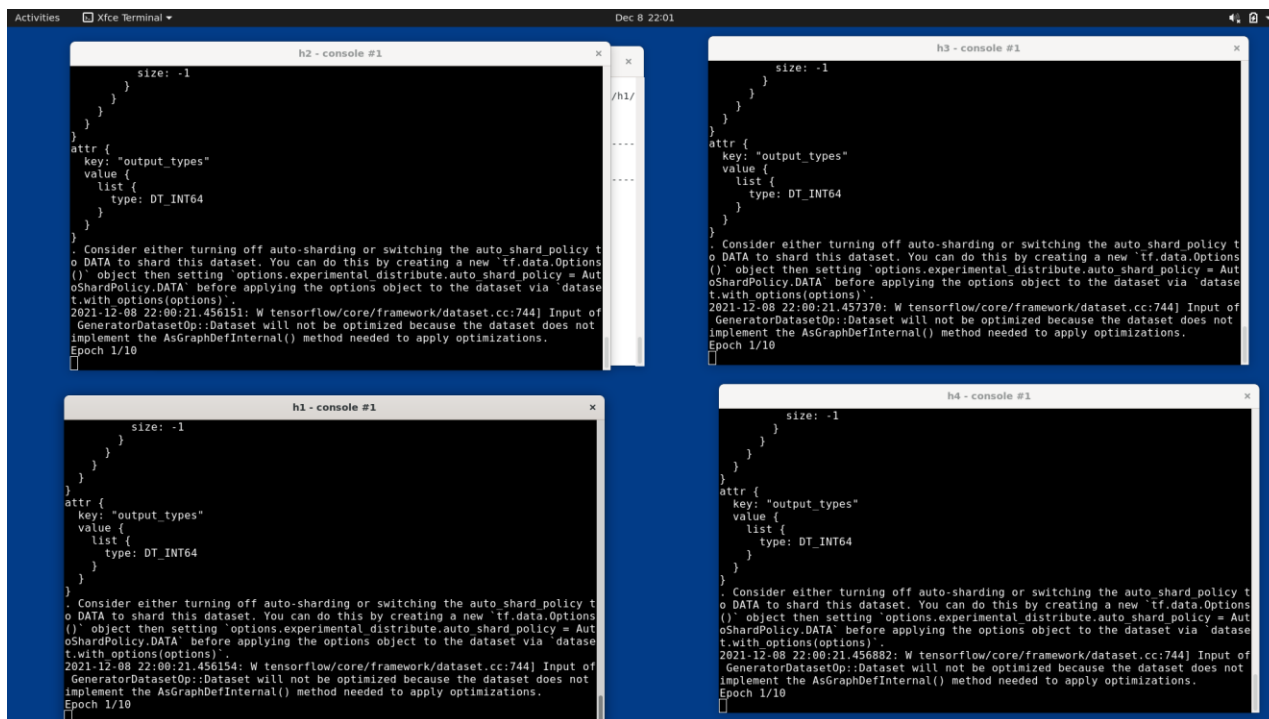


Figura 12: Arquitectura del modelo

CAPÍTULO 3: RESULTADOS

3.1 Ejecución del prototipo

Para ejecutar el prototipo es necesario tener el mismo código en todos los nodos así también como el dataset a utilizar. Posteriormente, en todas las máquinas se debe ejecutar el código escrito en Python cambiando únicamente el índice perteneciente a la dirección IP propia dentro del arreglo en la variable *TF_CONFIG*. En el prototipo desarrollado, este índice se pasa como argumento en la línea de comando al ejecutar el código. Una vez que el modelo está funcionando, el computador escucha conexiones entrantes en el puerto designado hasta que todos los demás nodos estén ejecutando el código y una vez que todos se conecten, comienza el entrenamiento. La Figura 13 muestra el modelo siendo entrenado en todos los nodos del ambiente virtualizado.



```
size: -1
  }
  }
}
attr {
  key: "output_types"
  value {
    list {
      type: DT_INT64
    }
  }
}
}
}

Consider either turning off auto-sharding or switching the auto_shard_policy t
o DATA to shard this dataset. You can do this by creating a new 'tf.data.Options
()' object then setting 'options.experimental_distribute.auto_shard_policy = Aut
oShardPolicy.DATA' before applying the options object to the dataset via 'databse
t.with_options(options)'.
2021-12-08 22:00:21.456151: W tensorflow/core/framework/dataset.cc:744] Input of
GeneratorDatasetOp::Dataset will not be optimized because the dataset does not
implement the AsGraphDefInternal() method needed to apply optimizations.
Epoch 1/10

```

Figura 13: Ejecución del modelo en la red virtualizada

3.2 Desempeño y evaluación del modelo

El desempeño de un modelo cuyo propósito es el de traducir texto o de generar texto nuevo puede ser medido mediante un puntaje conocido como Bilingual Evaluation Understudy (BLEU), métrica propuesta por Papineni et al en su investigación “BLEU: a Method for Automatic Evaluation of Machine Translation” (2002). El cálculo de este puntaje es simple ya que se basa en encontrar el número de n-gramas presentes en el texto obtenido que también están en la referencia. Por n-gramas se entiende como una única palabra (unigrama) o una agrupación de palabras (bigrama, trigrama, ...). El cálculo inicial consiste en dividir el número de n-gramas presentes en el texto resultante para el número de n-gramas totales que se encuentran en la referencia.

$$p_n = \frac{\sum_{C \in \{Candidatos\}} \sum_{n\text{-grama} \in C} Count_{clip}(n - grama)}{\sum_{C \in \{Candidatos\}} \sum_{n\text{-grama} \in C} Count(n - grama)}$$

El resultado de esta operación se conoce como precisión y esta debe ser ajustada dependiendo de si el número de n-gramas generados es mayor al número de n-gramas de la referencia. Si es mayor provoca una modificación en la precisión por lo que se debe aplicar una penitencia. El valor de esta se calcula de la siguiente manera:

$$BP = \begin{cases} 1 & \text{si } c > r \\ e^{(1-\frac{r}{c})} & \text{if } c \leq r \end{cases}$$

Con este resultado ya es posible calcular el BLEU score de la siguiente manera:

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^N w_n \log(p_n)\right)$$

Esta métrica de evaluación puede tomar valores de 0 a 1, siendo 0 una nula coincidencia entre el texto candidato y la referencia y 1 una coincidencia perfecta. Una vez aclarado el

funcionamiento de la métrica de evaluación del modelo, se procedió a obtener los resultados que se muestran en la Figura 14.

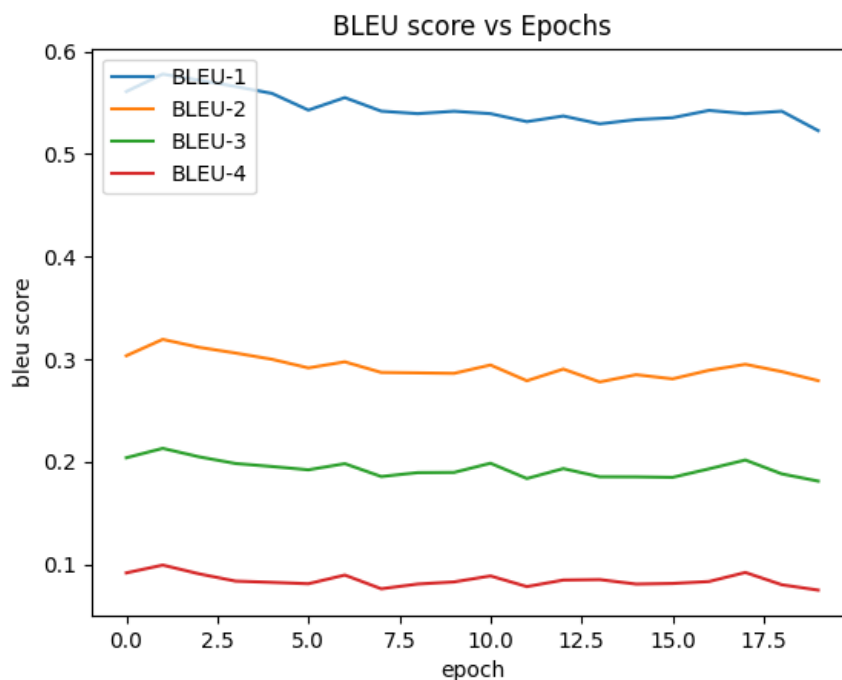


Figura 14: Gráfica del puntaje BLEU vs Epochs para el dataset de prueba

En la Figura 14 se pueden observar los resultados del desempeño del modelo para 20 épocas, calculando el puntaje BLEU para 1, 2, 3 y 4 n-gramas. A simple vista se puede notar que los valores entre las diferentes épocas no tienen mucha variación, lo cual se debe a que los textos candidatos generados tienen una corta longitud al igual que los textos de referencia. Cada texto tiene un máximo de 34 caracteres, lo que significa que pueden contener de cuatro a seis palabras. Con base en la gráfica, se puede establecer que el mejor modelo se obtiene en la época 2 ya que se logra conseguir el mejor puntaje para cada n-gramas. El puntaje BLEU fue calculado para medir la precisión del modelo que, en conjunto con los resultados con respecto al rendimiento del entrenamiento distribuido que se muestran más adelante, permiten obtener conclusiones acerca del prototipo. De acuerdo con Google (2022), un modelo con un puntaje BLEU por encima de 0.4 ya

puede ser considerado que tiene un buen rendimiento. El mejor puntaje obtenido fue de 0.578 para BLEU-1 por lo que se puede comprobar que el entrenamiento distribuido tiene la capacidad de mejorar la precisión de un modelo de Machine Learning. Esto es gracias a que la probabilidad de cometer errores en el entrenamiento disminuye a medida que se aumentan los nodos del sistema, y en el caso de uno de estos comete un error, puede potencialmente ser compensado por otro nodo (Peteiro-Barral, 2013).

3.3 Rendimiento del entrenamiento distribuido

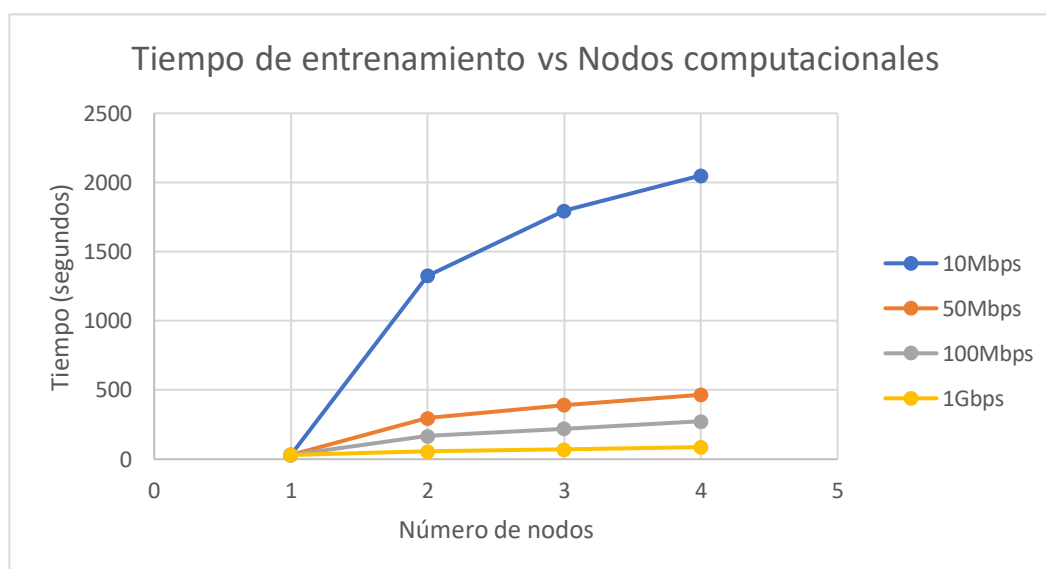


Figura 15: Gráfico del rendimiento del entrenamiento distribuido con anchos de banda tradicionales

En cuanto al rendimiento del entrenamiento distribuido ejecutado en la red virtualizada anteriormente definida, se obtuvieron los resultados mostrados en la Figura 15. Para medir el rendimiento y la eficiencia conseguida en el prototipo se evaluó la precisión del modelo, cuyos resultados fueron discutidos en la sección anterior, y también se midió el tiempo que le tomó al modelo para ser entrenado. En la gráfica se encuentra el tiempo de entrenamiento vs el número de nodos computacionales en el sistema. Cada línea representa un diferente ancho de banda, la cual

se logró modificar mediante un script de bash (Figura 16), limitando así la velocidad con la que se comunican las máquinas virtuales con sus respectivos adaptadores de red para conseguir una simulación de un entorno real. Se observa claramente como a medida que los nodos aumentan, también lo hace el tiempo de entrenamiento mientras que con el aumento de ancho de banda hace que este tiempo disminuya. Adicionalmente, el funcionamiento del script que limita el ancho de banda fue comprobado mediante la herramienta Iperf², la cual permite medir el ancho de banda entre dos nodos (Figura 17).

```

ovs-vsctl -- set Port $PORT_1 qos=@newqos -- \
--id=@newqos create qos type=linux-htb other-config:max-rate=$MAX_RATE \
  queues:0=@q0 -- \
--id=@q0 create queue other-config:max-rate=$MAX_RATE

```

Figura 16: Líneas de comando que permiten limitar el ancho de banda de una interfaz en Linux

```

root@h1:~# iperf -c 10.1.0.3
-----
Client connecting to 10.1.0.3, TCP port 5001
TCP window size: 85.0 KByte (default)
-----
[  3] local 10.1.0.2 port 53400 connected with 10.1.0.3 port 5001
[ ID] Interval      Transfer    Bandwidth
[  3] 0.0-10.2 sec  11.9 MBytes  9.75 Mbits/sec
root@h1:~#

```

Figura 17: Resultados al utilizar la herramienta Iperf para un ancho de banda de 10Mbps

Una de las propiedades fundamentales de un sistema de Machine Learning distribuido es la de reducir el tiempo en que un modelo puede ser entrenado, sin embargo, este no fue el comportamiento que se consiguió en el prototipo ya que como se explicó anteriormente, a medida que se aumentan nodos para el entrenamiento, el rendimiento disminuye. Estos primeros experimentos fueron realizados utilizando un ancho de banda que variaba entre valores que se pueden encontrar en redes tradicionales como las de un hogar o de una oficina. No obstante, sistemas como estos por lo general son implementados en ambientes adaptados para computación de alto rendimiento que cuentan con redes con velocidades mucho más altas. Es por esto que se

² ESnet, Lawrence Berkeley National Laboratory. (2020). Iperf (versión 3.9) [Software de Computadora]. Obtenido de <https://iperf.fr/>

realizó un segundo experimento en el mismo ambiente virtual, pero incrementando significativamente el ancho de banda disponible para las máquinas virtuales (resultados en la Figura 18). En cuanto al dispositivo en donde se realizaron los experimentos, era una computadora que cuenta con un Procesador Intel® Core™ i7-6500U (caché de 4 M, hasta 3,10 GHz), 12GB DDR4 RAM y Ubuntu 20.04 como sistema operativo.

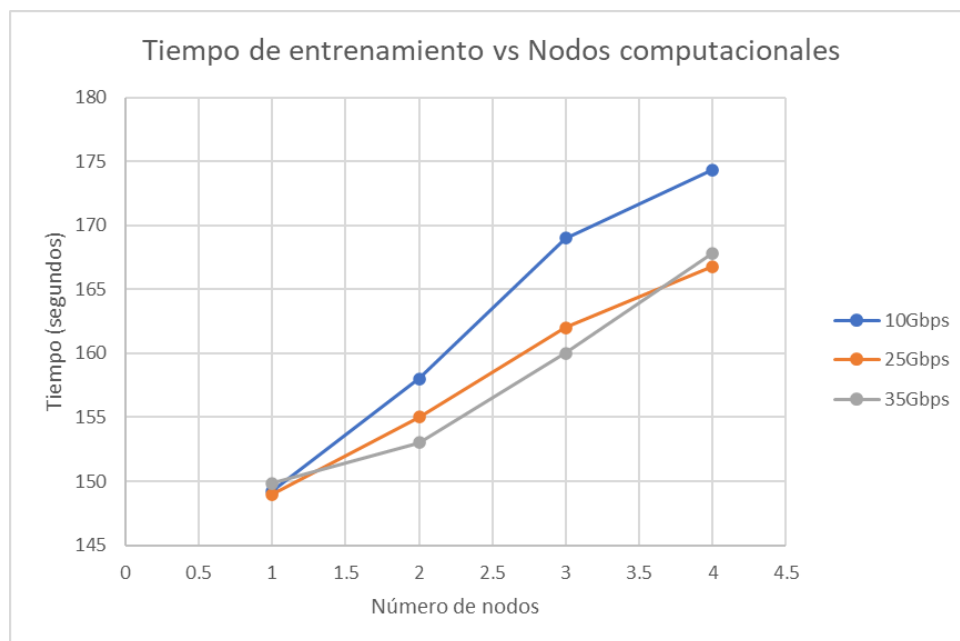


Figura 18: Gráfico del rendimiento del entrenamiento distribuido en una red de alto rendimiento

Los resultados obtenidos en el segundo experimento siguen el mismo comportamiento que los observados en el primer experimento, pero en esta ocasión se observa una diferencia mucho menor entre entrenar el modelo en un solo nodo o entre varios. Esto indica el fuerte impacto que tiene la red dentro de este tipo de sistemas.

Por otro lado, también es importante tener en cuenta que, al tratarse de un entorno virtualizado, cuando se agrega más nodos al sistema no se está agregando poder computacional real, sino que los recursos disponibles están siendo repartidos entre las diferentes máquinas virtuales. Es por esto que el experimento fue llevado a un ambiente real para comprender a más

profundidad el comportamiento del modelo y del escenario virtual. Este ambiente estuvo compuesto por dos computadores, cada uno con un Procesador Intel® Core™ i7-10870H (caché de 16 MB, hasta 5,00 GHz), una GPU NVIDIA® GeForce® GTX 1660 Ti 4GB GDDR6, con 16GB RAM DDR4 y sistema operativo Windows 10. Los computadores se encontraban conectados a una red LAN mediante Ethernet, la cual contaba con un ancho de banda máximo de 1 Gbps.

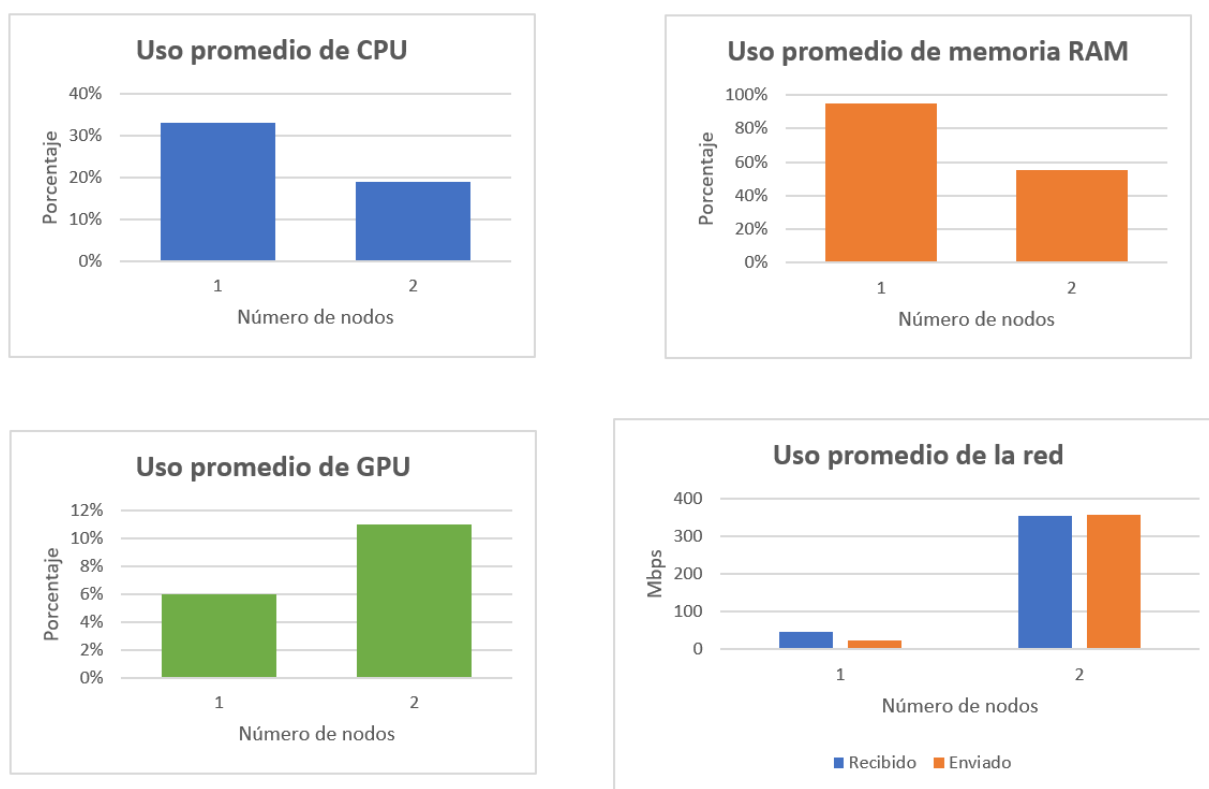


Figura 19: Uso computacional promedio durante el experimento con los nodos físicos

En el experimento con los dos computadores también se observó un comportamiento similar al del ambiente virtual, el entrenamiento fue más rápido cuando se lo realizó en un solo nodo (aproximadamente un 55% más rápido). Durante el experimento fue posible medir el costo computacional de entrenar el sistema y los resultados de esto se pueden observar en la Figura 19.

En cuanto al uso de CPU y memoria RAM se observa claramente como la carga computacional es distribuida cuando el número de nodos aumenta, no obstante, el uso de la GPU también tiene un ligero aumento por lo que este el sistema distribuido tiene la propiedad de aprovechar mejor este tipo de hardware. Por otro lado, también se observa que el uso de la red aumenta considerablemente cuando el entrenamiento se hace en más de un nodo, aunque el promedio de ancho de banda utilizado es mucho menor al disponible.

Por último, en la Figura 20 se incluye un ejemplo de la aplicación del modelo con imágenes que no constan dentro del dataset.



two children are playing in the grass

Figura 20: Ejemplo de los resultados del modelo

CAPÍTULO 4: CONCLUSIONES

4.1 Aprendizaje automático distribuido

Con base en la teoría investigada y a los experimentos realizados se concluye que un sistema de aprendizaje automático distribuido y su rendimiento dependen de los siguientes factores:

- **Topología del modelo:** Un número mayor de capas en el modelo de redes neuronales implica una comunicación más constante al momento de transmitir los gradientes, por lo que el costo de comunicación aumenta significativamente. Por otro lado, si el número de capas es menor, se requiere menos comunicación de los gradientes, pero aumenta la importancia del cálculo de los mismos, situación donde el sistema distribuido puede tener un buen rendimiento (Shi y Chu, 2018).
- **Framework utilizado:** Cada framework tiene sus ventajas y desventajas y es por esto que es necesario realizar un análisis previo para determinar cuál de estos es el que más satisface los requerimientos del proyecto. Por ejemplo, Frameworks como Horovod pueden tener un mejor desempeño en cuanto a rapidez. Sin embargo, puede que no ofrezcan tolerancia al fallo, por lo que es importante determinar cuál de estas características es más importante.
- **Dataset utilizado:** El dataset empleado para el entrenamiento del modelo también tiene impacto en su rendimiento. Quang-Hueng et al. concluyeron que mientras Tensorflow logra un mejor rendimiento al procesar datasets como el MNIST (popular dataset creado en 1998 que contiene imágenes de dígitos y letras escritas a mano para ser reconocidas por una computadora), puede conseguir un mal desempeño para procesar otros como el Beans (dataset para reconocimiento de imágenes), utilizando el mismo modelo (2020).

- Red: Los nodos deben comunicarse tanto para la transmisión de los parámetros como para tareas de planificación y es debido a esto que, la configuración de la red y su ancho de banda tienen un papel muy importante para el desempeño del sistema distribuido. Un sistema de Machine Learning distribuido tiene la propiedad de acelerar el tiempo de entrenamiento de un modelo siempre y cuando el costo de comunicación no sea muy alto (Shi y Chu, 2018). Es por esto que es recomendable el uso de redes con gran ancho de banda, por ejemplo, utilizando dispositivos InfiniBand los cuales están basados en tejidos de conmutación y ofrecen un gran ancho de banda (pueden alcanzar 56 Gbps) y baja latencia de interconexión entre dispositivos (Oracle, 2014).
- Poder computacional de cada nodo: Una máquina con un poder computacional menor al de los demás presentes en el sistema puede resultar contraproducente para la velocidad del entrenamiento. Esto se da principalmente porque cada nodo tiene que calcular el valor de los gradientes y luego pasar estos valores a los demás nodos para actualizar el modelo. Por lo tanto, si el proceso de cálculo en un nodo es lento, el tiempo de espera en la comunicación será alto.

4.2 Mejoras al modelo (trabajo futuro)

Como se puede observar en la Figura 19, el modelo realiza un buen trabajo identificando los elementos principales que componen la imagen. Sin embargo, tiene problemas para realizar decisiones acerca de detalles más específicos, como la actividad que están haciendo las personas. Una solución a este problema puede ser la utilización de un dataset más grande que cuente con una variedad mayor de imágenes, así también como descripciones más extensas. Adicionalmente, se puede considerar utilizar un modelo más grande para la extracción de imágenes como el modelo

InceptionV3, por ejemplo. Adicionalmente, se puede explorar otras topologías para el modelo de redes neuronales o variar los parámetros de entrenamiento.

Estas mejoras también pueden ser implementadas de tal forma que no solo ofrezcan resultados más precisos, sino que también permitan aprovechar el máximo rendimiento que un sistema distribuido tiene para ofrecer. Para esto, también se puede considerar replicar el modelo en un ambiente apto para computación de alto rendimiento, que ofrezcan un gran ancho de banda para la rápida comunicación en las tareas de planificación y la comunicación de los gradientes. Como se explicó anteriormente, esto puede ser conseguido implementando dispositivos InfiniBand. Adicionalmente, también se puede aumentar el poder computacional de cada nodo incorporando en su sistema GPUs de mayor potencia o circuitos ASIC que ofrezcan mejores resultados al momento de trabajar con redes neuronales.

REFERENCIAS BIBLIOGRÁFICAS

- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)* (pp. 265-283)
- Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017, August). Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)* (pp. 1-6). Ieee. doi:10.1109/icengtechnol.2017.830
- Biswas, R., Lu, X., & Panda, D. K. (2018, December). Accelerating tensorflow with adaptive rdma-based grpc. In *2018 IEEE 25th International Conference on High Performance Computing (HiPC)* (pp. 2-11). IEEE. doi:10.1109/hipc.2018.00010
- Bonaccorso, G. (2017). *Machine learning algorithms*. Packt Publishing Ltd.
- Bottou, L., Curtis, F. E., & Nocedal, J. (2018). *Optimization Methods for Large-Scale Machine Learning*. *SIAM Review*, *60*(2), 223–311. doi:10.1137/16m1080173
- El Naqa, I., & Murphy, M. J. (2015). What is machine learning?. In *machine learning in radiation oncology* (pp. 3-11). Springer, Cham.
- ESnet, Lawrence Berkeley National Laboratory. (2020). Iperf (versión 3.9) [Software de Computadora]. Obtenido de <https://iperf.fr/>
- Facebook. (2017). Gloo. Obtenido de: <https://github.com/facebookincubator/gloo>
- Goldsborough, P. (2016). A tour of tensorflow. *arXiv preprint arXiv:1610.01178*.
- Google. (2022). *Evaluating models*. Google Cloud. Obtenido de: <https://cloud.google.com/translate/automl/docs/evaluate>
- Google. (2021). *Map Reduce para App Engine*. Google Cloud. Obtenido de: <https://cloud.google.com/appengine/docs/standard/python/dataprocessing>
- Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.
- Guo, C., Wu, H., Deng, Z., Soni, G., Ye, J., Padhye, J., & Lipshteyn, M. (2016). RDMA over commodity ethernet at scale. In *Proceedings of the 2016 ACM SIGCOMM Conference* (pp. 202-215). doi:10.1145/2934872.2934908
- Hodosh, M., Young, P., & Hockenmaier, J. (2013). Framing image description as a ranking task: Data, models and evaluation metrics. *Journal of Artificial Intelligence Research*, *47*, 853-899.

- Hosseiniinoorbin, S., Layeghy, S., Kusy, B., Jurdak, R., & Portmann, M. (2021). Exploring Deep Neural Networks on Edge TPU. *arXiv preprint arXiv:2110.08826*.
- Karau, H., Konwinski, A., Wendell, P., & Zaharia, M. (2015). *Learning spark: lightning-fast big data analysis*. " O'Reilly Media, Inc."
- Ketkar, N. (2017). Introduction to keras. In *Deep learning with Python* (pp. 97-111). Apress, Berkeley, CA. doi:10.1007/978-1-4842-2766-4_7
- Kim, P. (2017). Convolutional neural network. In *MATLAB deep learning* (pp. 121-147). Apress, Berkeley, CA. doi:10.1007/978-1-4842-2845-6_6
- Kozlovski, S. (2018). *A Thorough Introduction to Distributed Systems*. FreeCodeCamp. <https://www.freecodecamp.org/news/a-thorough-introduction-to-distributed-systems-3b91562c9b3c/>
- Kuon, I., & Rose, J. (2007). Measuring the gap between FPGAs and ASICs. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 26(2), 203-215.
- Li, M., Zhou, L., Yang, Z., Li, A., Xia, F., Andersen, D. G., & Smola, A. (2013). Parameter server for distributed machine learning. In *Big Learning NIPS Workshop* (Vol. 6, p. 2).
- Oracle. (2014). *Acerca de dispositivos InfiniBand*. Obtenido de [https://docs.oracle.com/cd/E56339_01/html/E53905/eyant.html#:~:text=InfiniBand%20\(IBM\)%20es%20una%20tecnolog%C3%ADa,comunicaci%C3%B3n%20de%20host%20a%20host.](https://docs.oracle.com/cd/E56339_01/html/E53905/eyant.html#:~:text=InfiniBand%20(IBM)%20es%20una%20tecnolog%C3%ADa,comunicaci%C3%B3n%20de%20host%20a%20host.)
- Pandey, R., & Silakari, S. (2021). Investigations on optimizing performance of the distributed computing in heterogeneous environment using machine learning technique for large scale data set. *Materials Today: Proceedings*. doi:10.1016/j.matpr.2021.07.089
- Papineni, K., Roukos, S., Ward, T., & Zhu, W. J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics* (pp. 311-318).
- Perumanoor, J. (2021). *What is VGG16? — Introduction to VGG16*. Medium. Obtenido de: <https://medium.com/@mygreatlearning/what-is-vgg16-introduction-to-vgg16-f2d63849f615>
- Peteiro-Barral, D., & Guijarro-Berdiñas, B. (2013). A survey of methods for distributed machine learning. *Progress in Artificial Intelligence*, 2(1), 1-11.
- Qian, B., Jie, S., Zhenyu, W., Renyu, Y., Albert, Z. & Omer, R. (2019). Orchestrating the Development Lifecycle of Machine Learning-Based IoT Applications: A Taxonomy and Survey.

- Quang-Hung, N., Doan, H., & Thoai, N. (2020). Performance Evaluation of Distributed Training in Tensorflow 2. In *2020 International Conference on Advanced Computing and Applications (ACOMP)* (pp. 155-159). IEEE.
- Sarker, I. H. (2021). Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, 2(3), 1-21.
- Sergeev, A., Del Balso, M. (2017) Meet Horovod: Uber's Open Source Distributed Deep Learning Framework for TensorFlow. <https://eng.uber.com/horovod/>
- Shi, S., Wang, Q., & Chu, X. (2018). Performance modeling and evaluation of distributed deep learning frameworks on gpus. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)* (pp. 949-957). IEEE.
- Schlegel, D. (2015). Deep machine learning on Gpu. *University of Heidelber-Ziti*, 12.
- Verbraeken, J., Wolting, M., Katzy, J., Kloppenburg, J., Verbelen, T., & Rellermeyer, J. S. (2020). A survey on distributed machine learning. *ACM Computing Surveys (CSUR)*, 53(2), 1-33. doi:10.1145/3377454
- VNX. (2020). *About VNX*. Obtenido de: https://web.dit.upm.es/vnxwiki/index.php/Main_Page
- Zhang, K., Alqahtani, S., & Demirbas, M. (2017, July). A comparison of distributed machine learning platforms. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)* (pp. 1-9). IEEE. doi:10.1109/icccn.2017.8038464