Análisis de tiempos de ejecución para modelos de machine learning entrenados localmente y en clusters de Spark

Juan Pablo Jaramillo Tobon
Daniel Alejandro Higuita Usuga

Monografía presentada para optar al título de Especialista en Analítica y Ciencia de Datos

Asesor

Efraín Alberto Oviedo

Universidad de Antioquia
Facultad de Ingeniería
Especialización en Analítica y Ciencia de Datos
Medellín, Antioquia, Colombia
2024

Cita	(Jaramillo Tobon & Higuita Usuga, 2024)
Referencia **Estilo APA 7 (2020)**	Jaramillo Tobon, J. P. Higuita Usuga, D. A. (2024). *Predicción de resultados de pruebas saber del 2024-1 en base a los resultados históricos de las pruebas, usando la información del estudiante, colegio y familia*. Universidad de Antioquia, Medellín, Colombia.

Especialización en Analítica y Ciencia de Datos, Cohorte V.

Centro de Investigación Ambientales y de Ingeniería (CIA).

UNIVERSIDAD DE ANTIOQUIA
Vicerrectoría de Docencia

Sistema de Bibliotecas

Centro de Documentación Ingeniería (CENDOI)

Repositorio Institucional: http://bibliotecadigital.udea.edu.co

Universidad de Antioquia - www.udea.edu.co

Rector: John Jairo Arboleda Céspedes.
Decano: Julio Cesar Saldarriaga Molina
Jefe departamento: Diego José Luis Botia Valderrama

Dedicatoria

A nuestras familias, sin su apoyo esto no sería posible

Agradecimientos

Al docente David Manuel Villanueva por su guía durante el desarrollo del trabajo

Tabla de contenido

Lista de tablas

Lista de figuras

Siglas, acrónimos y abreviaturas

ML	Machine Learning
TPR	True Positive Rate
MLPC	Multilayer Perceptron Classifier
AWS	Amazon Web Services
GCP	Google Cloud Platform
CNN	Convolutional Neural Network
SVM	Support Vector Machine

Resumen

El entrenamiento de modelos de machine learning es un proceso iterativo que consume altos recursos computacionales y tiempo de ejecución, dependiendo del hardware disponible, estos entrenamientos pueden ser más o menos eficientes respecto al tiempo de ejecución, lo que puede llevar a un gran costo energético e inversiones de tiempo, cuyo resultados pueden ser un modelo que no sea óptimo para el objetivo propuesto. Debido a las alternativas que existen para el entrenamiento de modelos, se necesita hacer un análisis sobre los tiempos de ejecución que lleva entrenar modelos que retornen una buena hipótesis, capaz de predecir con accuracy mayor al 70% y comparar la ejecución clásica en un pc vs clustering aplicando distintas configuraciones. Este proyecto utilizará modelos de aprendizaje automático para distinguir un pulmón sano de uno con neumonía, estos modelos serán entrenados en: CPU, GPU y ejecución distribuida en cluster, y se calculará cual recurso que en promedio sea más eficiente respecto al porcentaje de mejora en tiempos de ejecución en el entrenamiento de los modelos.

Palabras clave: Machine Learning, CPU, GPU, Cluster, Entrenamiento, Clasificación

Repositorio Github: https://github.com/juanjaramillo98/Training_Comparison/tree/master

Abstract

Training machine learning models is an iterative process that consumes high computational resources and execution time. Depending on the available hardware, these trainings can be more or less efficient with respect to execution time, which can lead to a large energy cost and time investments, the results of which may be a model that is not optimal for the proposed objective. Due to the alternatives that exist for training models, it is necessary to do an analysis of the execution times it takes to train models that return a good hypothesis, capable of predicting with accuracy greater than 70% and compare the classic execution on a PC vs. clustering applying different configurations. This project will use machine learning models to distinguish a healthy lung from one with pneumonia. These models will be trained on: CPU, GPU and distributed execution in a cluster, and it will be calculated which resource is most efficient on average with respect to the percentage of improvement in time. of execution in training the models.

Keywords: Machine Learning, CPU, GPU, Cluster, Training, Classification

Github Repository: https://github.com/juanjaramillo98/Training_Comparison/tree/master

1. Descripción del problema

1.1. Problema de negocio

Entrenar y mantener modelos de machine learning en la nube puede ser costoso, por ejemplo Stability AI, los creadores del generador de imágenes Stable Diffusion, pagan en promedio 99 millones de dólares al año a AWS por sus modelos en la nube [1]. Se estima que para entrenar GPT-3, un modelo con 175 millones de parámetros, tuvo un costo de 4.6 millones de dólares[2].

Se requiere comparar el rendimiento, respecto a tiempo de entrenamiento y costos, de entrenar modelos que identifiquen si una persona tiene neumonía dado su radiografía de pulmón, utilizando CPU, GPU, de manera distribuida con contenedores y en la nube. Se utilizaran imágenes de radiografías donde se ha o no diagnosticado la enfermedad.

Plataformas en la nube como AWS, Azure y GCP ofrecen servicios para desplegar clusters como EMR AWS, Azure Databricks y Dataproc para GCP, en este proyecto se eligió AWS para desplegar los clusters en EMR, ya que se trabajó con esta herramienta en cursos de la especialización y la curva de aprendizaje era menor que las de Azure y GCP.

Además los modelos deben dar hipótesis útiles para la clasificación y se debe tener en cuenta el tiempo que toma entrenarlos en cada recurso, por lo que las métricas serán: el porcentaje de mejora en tiempo de ejecución respecto al tiempo de entrenamiento en CPU, este tiempo solo incluye el entrenamiento y no el preprocesamiento de imágenes, ya que todos los modelos se van a entrenar con los mismo datos procesados, el objetivo principal es medir los tiempos de ejecución por lo que para considerar que el tiempo invertido en entrenamiento dio un buen modelo, se utilizara la medida de accuracy, si el modelo tiene un accuracy mayor del 70% se acepta, esta medida es la base que deben tener todos los modelos para ser aceptados.

1.2. Aproximación desde la analítica de datos

Para identificar si una persona tiene neumonía, se implementarán modelos de clasificación binaria que a partir de la radiografía de pulmón del individuo, clasifique si tiene la enfermedad. Los modelos se entrenarán con los datos de imágenes donde se ha dado el diagnóstico o el pulmón no tenga la enfermedad. Tendrán la capacidad de encontrar los patrones de la enfermedad dada la radiografía y con dicha información se puede soportar un diagnóstico, el modelo no reemplaza el diagnóstico de profesionales sólo es una herramienta que da información sobre la posibilidad de que tenga la enfermedad. Además se podrá ver si el entrenamientos del modelo es viable con hardware local antes de ir a la nube.

1.3. Origen de los datos

Los datos provienen de repositorios con los rayos x tomados a pacientes donde se dividen entre los que tienen neumonía y los que no, estos son: Chest X-ray Images, Chest X-Ray Images (Pneumonia), Chest X-ray (Covid-19 & Pneumonia) y Random Sample of NIH Chest X-ray . Los datos se encuentran en la plataforma de kaggle y son abiertos, no hay restricciones para su descarga y la plataforma tiene alta disponibilidad, en promedio pesan 4GB por repositorio, se debe tener espacio suficiente para almacenamiento [3]

Tabla 1

Fuentes con su fecha de generación y última actualización

Fuente	Fecha de Generación	Año última actualización
Chest X-ray Images	2018-06-01	2020
Chest X-Ray Images (Pneumonia)	2018-01-06	2018
Chest X-ray (Covid-19 & Pneumonia)	2020-10-12	2023
Random Sample of NIH Chest X-ray	2017-09-27	2017

1.4. Métricas de desempeño

Para comparar el desempeño en los tiempos de ejecución invertidos en el entrenamiento de los modelos, utilizando varios recursos: CPU, GPU, contendores y EMR en AWS, se utilizará el porcentaje de mejora en tiempo de ejecución

Figura 1
Fórmula de mejora en rendimiento respecto al tiempo de ejecución en CPU

$$\%rendimiento = ((tiempo\ ejecución\ recurso\ x / tiempo\ ejecución\ CPU) - 1) * 100$$

La fórmula para el cálculo se puede ver **Figura 1**, aquí se calcularán los tiempos de ejecución en cada recurso y se dividirán los tiempos obtenidos para comparar qué tanto mejora un tipo de ejecución respecto a otra los tiempos de ejecución, aquí solo se tienen en cuenta los tiempos de entrenamiento y no los de procesamiento de datos, ya que los modelos se entrenan con los mismos datos, el objetivo de la mejora en el tiempo es que sea mayor al 20% . Para las métricas de los modelos se espera que los modelos entrenados den un accuracy mayor a 70%

2. Objetivos

2.1.Objetivo general

Analizar el rendimiento del tiempo de ejecución en entrenamientos de modelos de machine learning entrenados localmente y en clusters de Spark.

2.2.**Objetivos específicos**

- Realizar un análisis exploratorio de los datos y ajustar el tamaño de las radiografías para que tengan un tamaño estándar.
- Crear una representación numérica de las imágenes
- Desplegar ambientes de ejecución en local y en clusters de spark que permitan ejecutar los modelos
- Medir los tiempos de ejecución de cada recurso utilizado con su respectivo modelo
- Comparar los tiempos de ejecución de cada modelo desplegado

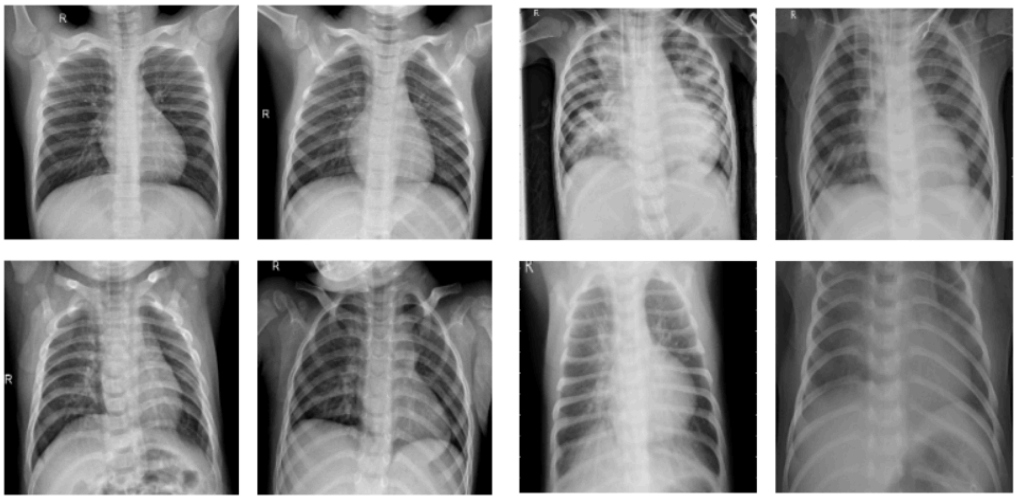
3. Datos

3.1. Datos originales

Los datos se recolectaron de la plataforma de kaggle, los datos están conformados por cuatro repositorios:

- Chest X-ray Images: contiene 5856 imágenes que fueron seleccionadas de cohortes de pacientes pediátricos de edades entre uno a cinco años del centro médico Guangzhou
- Chest X-Ray Images (Pneumonia): contiene 5863 imágenes que fueron seleccionadas de cohortes de pacientes pediátricos de edades entre uno a cinco años del centro médico Guangzhou como parte de la rutina clínica
- Chest X-ray (Covid-19 & Pneumonia): contiene 6432 imágenes que fueron tomadas de sitios públicos
- Random Sample of NIH Chest X-ray: contiene 5606 imágenes que se crearon a partir de las radiografías originales.

Figura 2
Radiografías de pulmones sanos y con neumonía



Pulmones Sanos **Pulmones con Neumonía**

Cada repositorio tiene un tamaño aproximado de 4GB y los datos que contiene son imágenes en formato PNG, y están divididos entre pulmones sanos y con la enfermedad como se ve en la ***Figura 2***

Los datos pueden descargarse de la plataforma de kaggle[4], se puede acceder a cualquiera de los cuatros repositorios sin restricción

Tabla 2

Datos de etiquetas en los repositorios

Dataset	Etiquetas
Chest X-ray Images	NORMAL, PNEUMONIA
Chest X-Ray Images (Pneumonia)	NORMAL, PNEUMONIA
Chest X-ray (Covid-19 & Pneumonia)	NORMAL, PNEUMONIA. COVID19
Random Sample of NIH Chest X-ray	NORMAL, PNEUMONIA

Además en cada repositorio se dividen las imagenes por directorio, donde el nombre el directorio indica la etiqueta de la imagen, las etiquetas de cada repositorio pueden verse en la **Tabla 2**

3.2. Datasets

Las imágenes van a estandarizarse para que tengan un tamaño de 128x128 pixeles, y se codifican para tener una representación vectorial, para esto se utiliza la librería de tensor flow, decode jpeg, que permite ajustar tamaño de imágenes y codificarlas

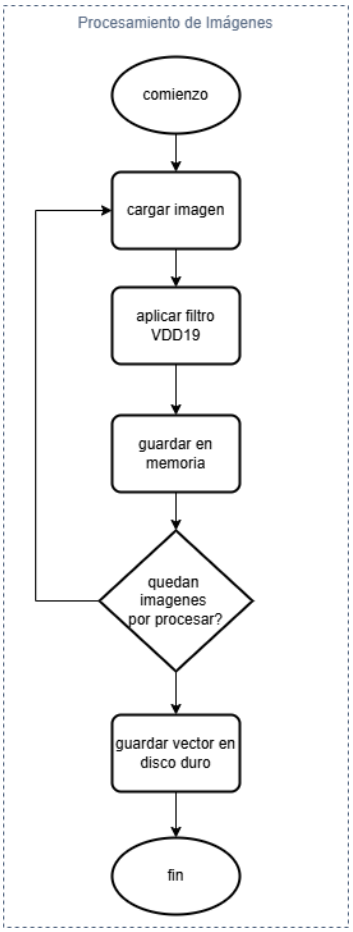
Los datos de tipo imagen deben ser codificados y tener un tamaño estándar para que puedan ser usados por modelos de ML, para este proyecto se utilizan tres tipos de transformaciones:

- **Decode:** transforma una imagen a tensor.
- **Resize:** ajusta la imagen transformada a un tamaño estándar
- **Convolución VGG19:** extrae las características más importantes de las imágenes para reducir la dimensionalidad

4. Data Pipeline

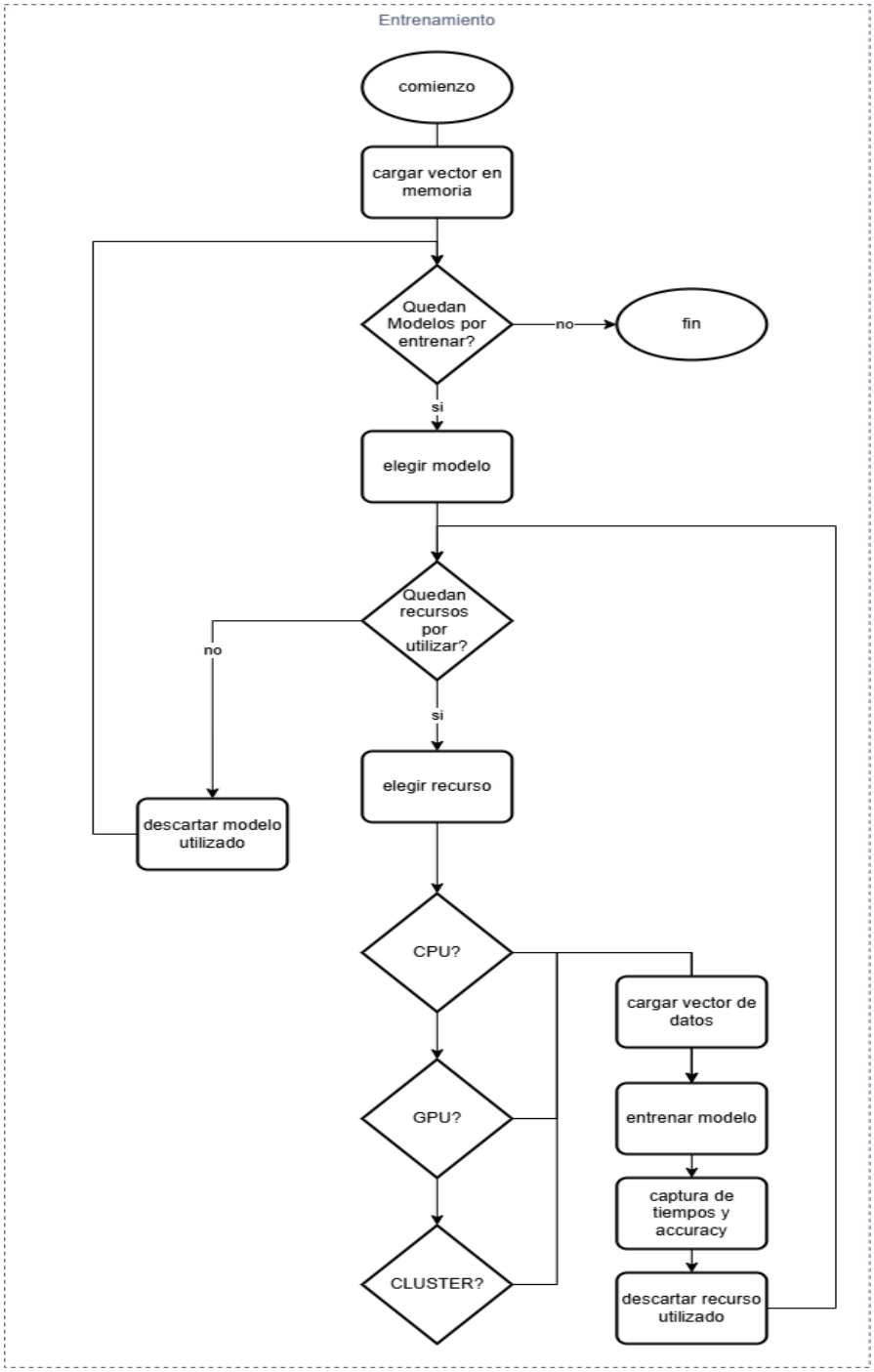
El pipeline comienza con un procesamiento de datos donde se crea un vector a partir de las imágenes utilizando VGG19

Figura 3
Diagrama de flujo procesamiento de imágenes



En la **Figura 3** se observa el flujo utilizado para transformar las imágenes en vectores, el flujo comienza con la carga de imágenes en memoria, este es un proceso intensivo en memoria por lo que es necesario tener una ram de mínimo 16 GB

Figura 4
Diagrama de entrenamiento de modelos por recurso



En la **Figura 4** se observa cómo se hizo el entrenamiento de modelos utilizando el vector procesado en la **Figura 3**, el entrenamiento se repite hasta que los modelos den una hipótesis con el rendimiento esperado mayor al 70%

5. Metodología

5.1 Primera Iteración

Para estandarizar los tiempos de máquina local se escogió el computador de uno de los integrantes del equipo, donde se van a realizar todos los experimentos

Tabla 3

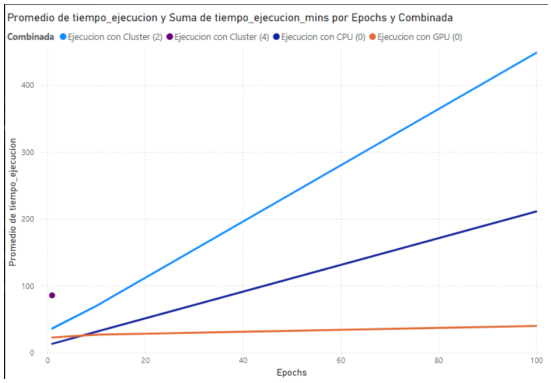
Especificaciones máquina local con su costo en dólares

Recurso	Costo
11th Gen Intel(R) Core(TM) i9-11900 @ 2.50GHz 2.50 GHz	$456.98
NVIDIA GeForce RTX 3070 Ti PCIe 4.0, 8 GB GDDR6X	$840.01

En la **Tabla 3** se encuentran las especificaciones del computador, la CPU y GPU que tiene y el costo en dólares de cada recurso

Decidimos solo entrenar una red neuronal VGG19 con las radiografías, y a partir de ahí, medir los tiempos de ejecución, esto trajo varios inconvenientes, no se pudieron aplicar tratamiento de datos, debido a que VGG19 tiene en sus capas su propia manera de transformar datos, por lo que no se pudo hacerles una intervención, por esto solo se midieron los tiempos de ejecución

Figura 5
Comparación de tiempos primera ejecución en máquina local



Como se evidencia en la **Figura 5** el tiempo de ejecución más bajo fue con GPU y el más alto fue con clustering, el bajo rendimiento del clustering pudo deberse a que la red VGG19 no se ejecutó de manera paralelizable, por esto, los tiempos pudieron ser penalizados.

Figura 6
Medición de tiempos en la primer ejecución de modelos en local

Tipo Ejecucion	Metodo	Steps_per_epoch	Epochs	tiempo_ejecucion
Ejecucion con CPU	VGG19	1	1	13.184482
			10	31.105596
			100	211.011338
Ejecucion con Cluster	VGG19	1	1	44.362372
			10	69.616173
			100	447.998702
Ejecucion con GPU	VGG19	1	1	22.663502
			10	26.856227
			100	39.893751

En la **Figura 6** están los resultados de los entrenamientos hechos, donde se ve el recurso utilizado, el modelo entrenado y las configuraciones usadas para entrenar, como ya se evidenciaba en la **Figura 5**, clustering tuvo el peor desempeño en todas ejecuciones.

Al final de esta iteración se decidió hacer un cambio a las métricas, el objetivo no solo es tener tiempos bajos, sino también tener modelos con un accuracy aceptable y ver cuanto tiempo de entrenamiento se invertiría en la obtención de dicho modelo por lo que se hizo un ajuste para medir el rendimiento

Tabla 4

Ejemplo de la fórmula de rendimiento aplicada a las mediciones de tiempos

Tipo Ejecución Cross Validation	Método	Epochs	Tiempo Ejecución (Segundos)	Accuracy	Mejora
CPU	MLPC	1	10.00	0.74	
GPU	MLPC		15.00	0.74	-33.33%
Cluster Docker	MLPC		5	0.74	200.00%

En la **Tabla 6** se encuentra un ejemplo del cálculo del rendimiento utilizando la fórmula de la **Figura 1**, en este caso la GPU tuvo un rendimiento inferior a los tiempos de la CPU y el cluster los mejoro.

5.2 Segunda Iteración

En la segunda iteración se decidió hacer la transformación de las imágenes a una representación numérica, utilizando solo el layer convolucional de VGG19 para extraer las características más relevantes de las imágenes, además se escogieron tres modelos de clasificación: regresión logística, naive bayes, clasificador MLP (multi layer perceptron), cada modelo se entrenó con CPU, GPU, distribuido en contenedores y se midió su tiempo de ejecución durante el entrenamiento

Tabla 5

Comparación tiempos de ejecución en máquina local segunda iteración

Tipo Ejecución	Método	Epochs	Tiempo Ejecución (Segundos)	Accuracy	Mejora
CPU	Logistic Regression		1.39	0.74	
	MLPC	20	38.70	0.74	
	Naive Bayes		0.18	0.37	
GPU	Logistic Regression		0.20	0.74	595.00%
	MLPC	20	38.05	0.74	1.71%
	Naive Bayes		0.05	0.37	260.00%
Cluster Docker	Logistic Regression		79.68	0.67	-98.26%
	MLPC	20	29.04	0.68	33.26%
	Naive Bayes		11.71	0.68	-98.46%

En la **Tabla 5** se encuentran las configuraciones de cada ejecución y los resultados obtenidos en tiempo y accuracy, para la regresión logística y bayes, la GPU tuvo el mejor rendimiento, mientras que para el modelo de MLCP fue mejor el cluster de docker, se resalta que todos los modelos excepto MLPC no usan épocas

Al final de esta iteración se analizó las razones del bajo rendimiento del cluster, las cuales se listan a continuación:

- **Metodología de Entrenamiento:** en esta iteración no se usó cross validation, por lo que no hubo una gran cantidad de carga para los recursos, por esto en la próxima iteración se harán entrenamientos con cross validation
- **Complejidad de los modelos:** los modelos utilizados como logistic regression y naive bayes no exigieron una carga computacional muy alta, por lo que las operaciones del clúster en la distribución del trabajo y consolidación de resultados van a penalizar aún más sus tiempos, por esto se va a conservar sólo el modelo de MLPC.

5.3 Tercera Iteración

El enfoque del proyecto estaba centrado en el análisis de optimización de tiempos y se estaba dejando de lado las métricas de machine learning, pero se aclaró que buenas métricas de machine learning son lo esencial y se debe tener el tiempo asociado a entrenar buenos modelos, en cuanto a los modelos entrenados, se descartaron los modelos más simples (regresión logística y naive bayes) para entrenar modelos más complejos como los MLPC, utilizando cross validation

Tabla 6

Comparación tiempos de ejecución en máquina local tercera iteración

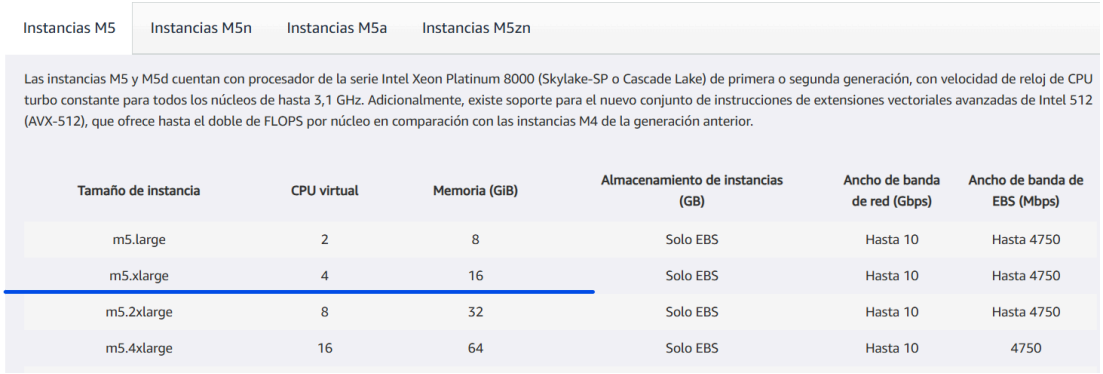
Tipo Ejecución Cross Validation	Método	Epochs	Tiempo Ejecución (Segundos)	Accuracy	Mejora
CPU	MLPC	20	163.16	0.74	
GPU	MLPC		332.26	0.74	-50.89%
Cluster Docker	MLPC		435.46	0.74	-62.53%

En la **Tabla 6** se muestran los resultados obtenidos, la GPU y el cluster en docker no mejoraron el rendimiento de la CPU. En comparación a los resultados obtenidos en la segunda iteración **Tabla 5**, aplicar cross validation al modelo MLPC aumentó de manera significativa los tiempos de ejecución de todos los recursos, analizando las razones que podrían estar causando el bajo rendimiento del cluster, se llegó a la conclusión que la virtualización hecha por Docker podía ser el problema, por esto, se decidió para la próxima iteración desplegar el cluster en AWS.

5.4 Cuarta Iteración

En esta iteración se abordó el despliegue de cluster en la nube de AWS para entrenar los modelos de MLPC utilizando ejecución distribuida, el cluster se desplegó en el servicio de AWS EMR, es una plataforma de clústeres gestionada que simplifica la ejecución de marcos de big data, como Apache Hadoop y Apache Spark[5], la prueba se hizo en una cuenta de AWS estudiantil[6].

Figura 7
Ficha técnica de las instancias M AWS[7]



Instancias M5 | Instancias M5n | Instancias M5a | Instancias M5zn

Las instancias M5 y M5d cuentan con procesador de la serie Intel Xeon Platinum 8000 (Skylake-SP o Cascade Lake) de primera o segunda generación, con velocidad de reloj de CPU turbo constante para todos los núcleos de hasta 3,1 GHz. Adicionalmente, existe soporte para el nuevo conjunto de instrucciones de extensiones vectoriales avanzadas de Intel 512 (AVX-512), que ofrece hasta el doble de FLOPS por núcleo en comparación con las instancias M4 de la generación anterior.

Tamaño de instancia	CPU virtual	Memoria (GiB)	Almacenamiento de instancias (GB)	Ancho de banda de red (Gbps)	Ancho de banda de EBS (Mbps)
m5.large	2	8	Solo EBS	Hasta 10	Hasta 4750
m5.xlarge	4	16	Solo EBS	Hasta 10	Hasta 4750
m5.2xlarge	8	32	Solo EBS	Hasta 10	Hasta 4750
m5.4xlarge	16	64	Solo EBS	Hasta 10	4750

La cuenta estudiantil restringe los tipos de instancias que se pueden usar, por resto la instancia subrayada en la **Figura 7** es la más cercana a la máquina local **Tabla 3,** donde se hicieron las ejecuciones

Tabla 7

Ejecución en cluster AWS cuenta de estudiante

Tipo Ejecución Cross Validation	Workers	Paralelismo	Método	Epochs	Tiempo Ejecución (Segundos)	Accuracy
Cluster AWS	4	2	MLPC	20	279.89	0.74

El resultado de la ejecución en el cluster de EMR se observa en la **Tabla 7** aquí se ven ejecución y los resultados de tiempo y accuracy obtenidos utilizando el cluster de EMR

Tabla 8

Comparación de tiempos con ejecución en cluster AWS cuenta de estudiante

Tipo Ejecución Cross Validation	Método	Epochs	Tiempo Ejecución (Segundos)	Accuracy	Mejora
CPU	MLPC	20	163.16	0.74	
GPU	MLPC		332.26	0.74	-50.89%
Cluster Docker	MLPC		501.95	0.74	-67.49%
Cluster AWS	MLPC		279.89	0.74	-41.71%

En la **Tabla 8** se comparan los tiempos de ejecución en el cluster desplegado en la cuenta de estudiante de AWS, se resalta que para el modelo MLCP, ninguno de los recursos mejoró a la cpu, pero los tiempos de cluster mejoraron a los obtenidos en la tercera iteración **Tabla 6**

Al final de la iteración se hizo un análisis de las instancias disponibles en AWS y se compararon con la utilizada en la **Figura 7,** se encontró que las instancias M están optimizadas para procedimientos de alto rendimiento en memoria, por lo que se decidió hacer un cambio a instancias enfocadas en computación, además por las limitaciones de la cuenta de estudiante de AWS para la próxima iteración se van a hacer los experimentos en una cuenta personal de AWS para poder tener acceso a todas las instancias.

5.5 Quinta Iteración

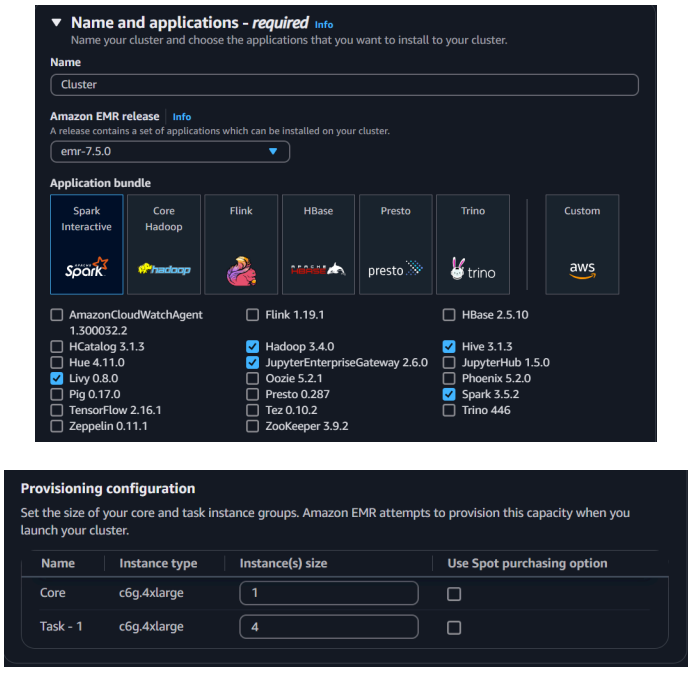
Esta fue la última iteración del proyecto, aquí se utilizó una cuenta paga de AWS para hacer las ejecuciones ya que la cuenta estudiantil tiene restricciones sobre el tipo de instancias que se puede usar y se buscó utilizar instancias orientadas a procesamiento y no memoria como las M.

Figura 8
Ficha técnica de las instancias C AWS[8]

Tamaño de la instancia	CPU virtual	Memoria (GiB)	Almacenamiento de instancias (GB)	Ancho de banda de red (Gbps)	Ancho de banda de EBS (Mbps)
c6g.medium	1	2	Solo EBS	Hasta 10	Hasta 4750
c6g.large	2	4	Solo EBS	Hasta 10	Hasta 4750
c6g.xlarge	4	8	Solo EBS	Hasta 10	Hasta 4750
c6g.2xlarge	8	16	Solo EBS	Hasta 10	Hasta 4750
c6g.4xlarge	16	32	Solo EBS	Hasta 10	4750

La instancia escogida para las ejecuciones es c6g.4xlarge, esta instancia tiene las mismas especificaciones que el computador utilizado, **Tabla 3,** la ficha técnica se ve a en la **Figura 8**

Figura 9
Configuraciones de EMR AWS cuenta personal



En la **Figura 9** están las configuraciones usadas para desplegar el cluster en emr

Sin embargo, durante el aprovisionamiento no creó el cluster completo, esto se debe a que las cuotas para cpus virtuales , hay un número máximo de vCPU de Amazon EC2 que puede aprovisionar para instancias bajo demanda en una región. Si intenta iniciar una instancia en una región y esta solicitud provoca que su uso supere esta cuota, se producirá un error en la solicitud. Si esto ocurre, puede reducir el uso de recursos o solicitar un aumento de cuota[9].

Figura 10

Solicitud de incremento de cuotas para EC2

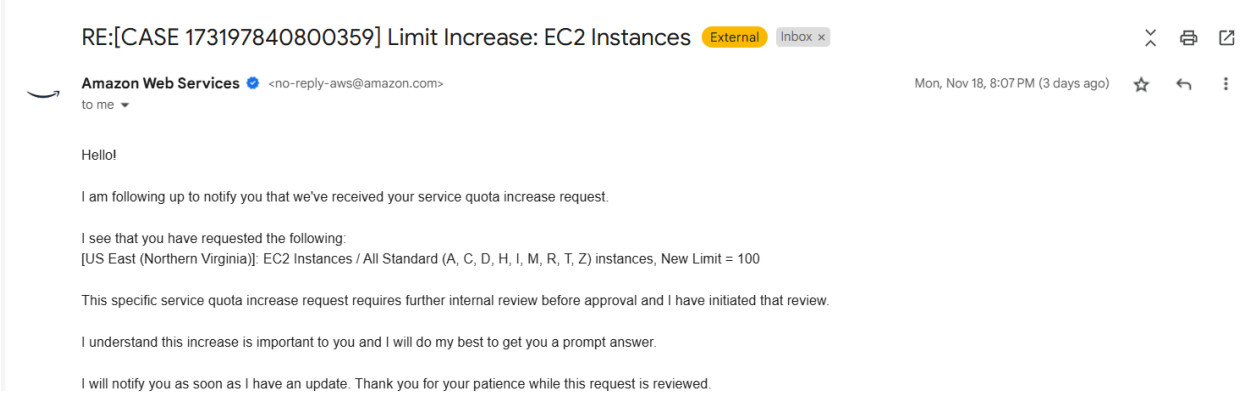
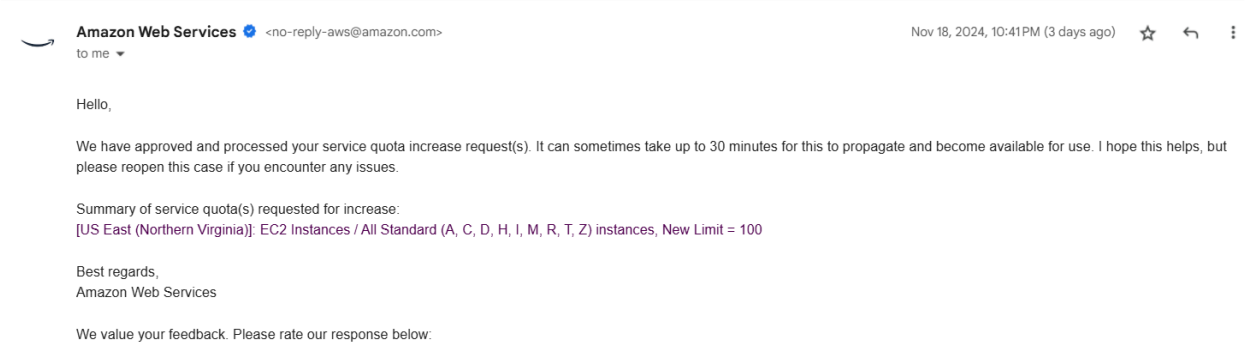


Figura 11

Respuesta del incremento de cuotas para EC2



Se hizo la solicitud de aumento de cuotas en la **Figura 10** y el tiempo de respuesta fue de aproximadamente 2 horas como se evidencia en la **Figura 11**

Tabla 9

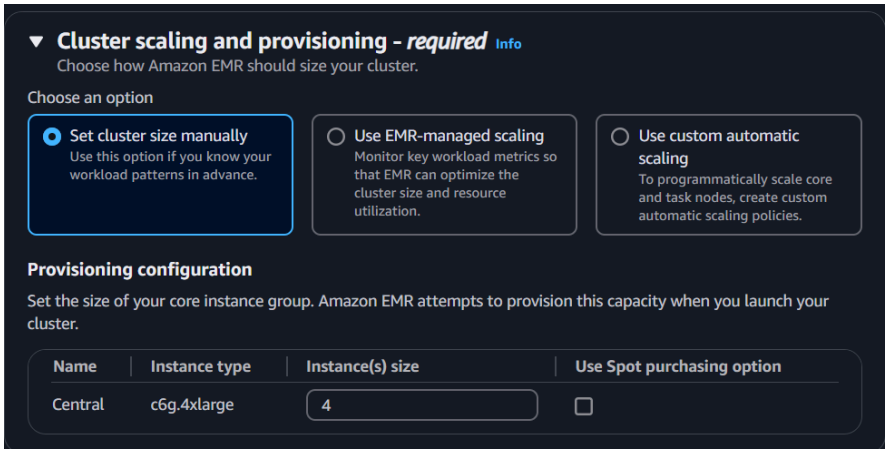
Comparación de tiempos con ejecución en cluster AWS 4 tasks

Tipo Ejecución Cross Validation	Método	Epochs	Tiempo Ejecución (Segundos)	Accuracy	Mejora
CPU	MLPC	20	163.16	0.74	
GPU	MLPC		332.26	0.74	-50.89%
Cluster Docker	MLPC		435.46	0.74	-23.70%
Cluster AWS 4 Tasks	MLPC		131.91	0.74	23.69%

Después de que aprobaran el incremento, ya se pudo crear el cluster completo con las configuraciones de la **Figura 9**, los resultados de la ejecución se pueden ver en la **Tabla 9**, como se evidencia el cluster de AWS mejorar el rendimiento de la CPU por encima del 20%

Figura 12

Configuración EMR con 4 Core sin Tasks



Con estos resultados se hicieron otras ejecuciones con configuraciones de cluster distintas, esto debido a que cambiando los nodos a central se podían obtener tiempos aún mejores, las configuraciones solo variaron respecto a la **Figura 9** de la siguiente manera, en la **Figura 12** se encuentre la configuración, solo se utilizan nodos centrales.

Tabla 10

Comparación de tiempos con ejecución en cluster AWS 4 cores

Tipo Ejecución Cross Validation	Método	Epochs	Tiempo Ejecución (Segundos)	Accuracy	Mejora
CPU	MLPC	20	163.16	0.74	
GPU	MLPC		332.26	0.74	-50.89%
Cluster Docker	MLPC		435.46	0.74	-23.70%
Cluster AWS 4 Tasks	MLPC		131.91	0.74	23.69%
Cluster AWS 4 Cores	MLPC		134.36	0.74	21.44%

Como se ve en la **Tabla 10** esta nueva configuración mejora el tiempo de CPU pero su mejora es inferior respecto a tener 4 tasks

Por último se hizo un experimento con las mismas configuraciones anteriores, pero utilizando cache desde el código para que los dataframe queden en memoria

Tabla 11

Comparación de tiempos con ejecución en cluster AWS 4 cores con cache

Tipo Ejecución Cross Validation	Método	Epochs	Tiempo Ejecución (Segundos)	Accuracy	Mejora
CPU	MLPC	20	163.16	0.74	
GPU	MLPC		332.26	0.74	-50.89%
Cluster Docker	MLPC		435.46	0.74	-23.70%
Cluster AWS 4 Tasks	MLPC		131.91	0.74	23.69%
Cluster AWS 4 Cores	MLPC		134.36	0.74	21.44%
Cluster AWS 4 Cores Cache	MLPC		139.85	0.74	16.67%

Los resultados pueden verse en la **Tabla 11** se siguen mejorando los tiempos respecto a la CPU pero se obtienen rendimientos inferiores respecto a tasks y cores sin caché.

En general, las configuraciones que dieron el mejor porcentaje de mejora y que fueron superiores a la meta del 20% y generaron modelos buenos, por encima del 70% de accuracy, fueron el cluster desplegado en EMR en la cuenta personal de AWS

Tabla 12

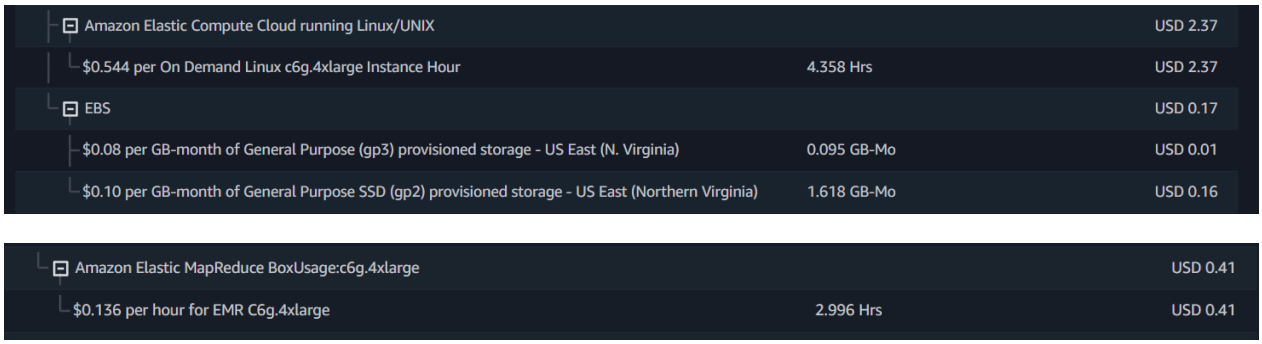
Configuración con los mejores resultados

Instancia	Tipo Ejecución Cross Validation	Método	Epochs	Tiempo Ejecución (Segundos)	Accuracy	Mejora
c6g.4xlarge	Cluster AWS 4 Tasks	MLPC	20	131.91	0.74	23.69%

En la **Tabla 12** se observan las configuraciones que dieron los mejores resultados y se mantuvieron dentro de los límites mínimos establecidos, en la sección de métricas,

Figura 13

Configuración EMR con 4 Core



Finalmente Los costos asociados a los experimentos en AWS pueden verse en la **Figura 13**, el costo total de los experimentos fue de 2.95 dólares estadounidenses

6. Herramientas

Tabla 13

Herramientas usadas en el desarrollo

Herramienta	**Descripción**
Python	Lenguaje de programación
Kubernetes	Plataforma que permite administrar contenedores
Docker	Plataforma que permite administrar procesos y recursos computacionales
Tensor Flow	Librería para entrenar modelos basados en redes neuronales
Spark	Herramienta que permite el procesamiento distribuido de grandes volúmenes de datos
AWS EMR	Servicio de AWS que permite montar clusters de spark
Github	Herramienta para el control de versiones
Rapids Cuda	Librería que ejecuta procesos como transformación de imágenes en GPU
Sklearn	Librería con implementaciones de modelos de machine learning

En la **Tabla 13** se ven las herramientas y una breve descripción de cada una de ellas.

7. Conclusiones

El rendimiento de tiempos en la nube fue superior a los de los recursos de la máquina local obteniendo en todos los entrenamientos un modelo consistente con un accuracy aproximadamente del 74% para todos, respecto a la virtualización de cluster en docker tuvo una penalización alta en tiempos a comparación de la hecha por AWS con sus instancias EC2, además el modelo de costo por pago por uso de AWS fue más barato que el costo por adelantado del pc local.

En cuanto a los rendimientos de los cluster en AWS, el mejor fue el que se entrenó para modelos MLCP con cuatros tasks, los resultados con cuatro cores y haciendo caché en memoria no mejoraron el rendimiento final.

Hay oportunidad de mejora en la cantidad de datos usados para el entrenamiento, con esto aparte de depender de la capacidad de procesamiento de los recursos se pueden hacer pruebas donde se deban alojar grandes cantidades de memoria en RAM y se deban mover datos entre memoria y disco duro, lo que puede penalizar aún más los tiempos de entrenamiento por operaciones I/O

El modelo de MLCP con las configuraciones de cluster emr en aws, instancia c6g.4xlarge y con cuatros tasks dio el mejor rendimiento 23.69% con accuracy del 74%, los cuales estuvieron dentro de los límites mínimos establecidos de rendimiento, mayor a 20%, y accuracy,mayor al 70%, para oportunidades de mejora se recomienda realizar los entrenamientos con modelos aún más complejos que el MLCP por ejemplo CNN, SVM y XG Boost

8. Recomendaciones

Para futuros experimentos se pueden hacer comparaciones con RAGS para determinar configuraciones con las que el entrenamiento de modelos puede compararse a los obtenidos en la nube.

Se pueden hacer estudios de mercado donde se hagan estimaciones de costos respecto a donde es más óptimo económicamente entrenar modelos.

Se pueden hacer estudios donde se busque mejorar la manera de distribuir el trabajo entre los recursos disponibles para hacer los entrenamientos más óptimos respecto al tiempo, ya que en lugar de tener máquinas cada vez más potentes se podrían consumir menos recursos con mejores algoritmos distribuidos.

Referencias

[1]Mann, T. (2024) Stability AI reportedly ran out of cash to pay its AWS Bills, Available at: https://www.theregister.com/2024/04/03/stability_ai_bills/

[2] Meyer, D. (2024) *Why the cost of training AI could soon become too much to bear*, Available at:https://fortune.com/2024/04/04/ai-training-costs-how-much-is-too-much-openai-gpt-anthropic-microsoft/

[3] *A survey on machine learning: Concept, algorithms* Available at: https://www.smec.ac.in/assets/images/committee/research/17-18/281

[4](2024) *Pneumonia detections using Deep Learning*, *Kaggle*. Available at: https://www.kaggle.com/code/chanchal24/pneumonia-detections-using-deep-learning/input?select=chest_xray

[5]*¿Qué es Amazon Emr?*Available at: https://docs.aws.amazon.com/es_es/emr/latest/ManagementGuide/emr-what-is-emr.html

[6]*AWS educate - cloud skills for education- AWS*. Available at: https://aws.amazon.com/education/awseducate/

[7] *Amazon EC2 M5 instances - general purpose compute workloads*. Available at: https://aws.amazon.com/ec2/instance-types/m5/

[8] *Amazon EC2 C5 instances — Amazon Web Services (AWS)*. Available at: https://aws.amazon.com/ec2/instance-types/c5/

[9] *Cuotas de Servicio de Amazon EC2 - amazon elastic compute cloud*. Available at: https://docs.aws.amazon.com/es_es/AWSEC2/latest/UserGuide/ec2-resource-limits.html