

Análisis de imágenes multiespectrales para detección de patrones en  
Agricultura con grandes volúmenes de datos.

Arcia Hernández Jesús, Fajardo Becerra Daian, Salazar Escobar Carlos, & Sepúlveda  
Jimenez Hernán.

Universidad EAFIT.

Maestría en ciencia de los datos y analítica.

Minería de datos para grandes volúmenes de datos.

2021.

## Pregunta de Investigación

¿Es posible entrenar un algoritmo de aprendizaje automático para la clasificación de imágenes con un data set tan grande que no es posible entrenarlo en memoria de manera convencional?

## Objetivo General

Diseñar e implementar un algoritmo con arquitectura en la nube, orientada al manejo de grandes volúmenes de datos como solución a la clasificación de imágenes de cultivos.

## Objetivos Específicos

1. Diseñar una arquitectura de servicios en la nube orientada al entrenamiento, validación y testeo de algoritmos de aprendizaje automático y profundo.
2. Identificación de los métodos que se utilizan para el manejo de grandes volúmenes de datos.
3. Probar diferentes técnicas de paralelismo en los datos y en el entrenamiento del modelo.
4. Evaluar el rendimiento y costo de la arquitectura y algoritmos propuestos.
5. Implementación de la arquitectura y el algoritmo seleccionado en producción.

## Revisión de la literatura, estado del arte

Dentro de la arquitectura de este proyecto, se tomo para el procesamiento de imágenes *Tensor Flow* ya que es una librería de código libre para machine learning a través de un rango de tareas. Tensor Flow fue desarrollado por Google con el fin de satisfacer las necesidades de redes neuronales, y debido a esto, se volvió una herramienta muy popular para Machine Learning.

El uso de Tensor Flow consta de *Placeholders* que son las entradas y salidas que son alimentadas con datos, variables y/o constantes, error, algoritmo de aprendizaje (descenso del gradiente) y finalmente la predicción.

Esta herramienta ha tenido diferentes aplicaciones dentro del mundo de la ciencia de los datos, entre los cuales se puede destacar la segmentación de imágenes, donde se busca conocer en donde se encuentra un objeto dentro de una imagen, por lo que se quiere asignar una etiqueta a cada píxel, Por lo tanto, la tarea de la segmentación de la imagen consiste en entrenar una red neuronal para que emita una máscara de la imagen en función de los píxeles.

Otra aplicación es la clasificación de imágenes, y dentro de este proyecto se revisará la construcción de modelos con Keras. *Keras* ha sido parte integral de la API de Tensor Flow a pesar de que es un software independiente, adicionalmente es una librería de código abierto con el cual se puede construir redes neuronales.

Keras se desempeña a nivel de modelo y se proporciona en bloques modulares en el cual trabajan los modelos de Machine Learning. Para el uso de Keras se debe comprender que se toman los datos almacenados en una carpeta y cada subcarpeta se vuelve una clase individual, indicando que se redimensiona las imágenes y crea los lotes automáticamente, y finalmente las etiquetas generadas se crean a partir de los nombres de la subcarpeta.

Una de las características que mas se resalta en Keras es que puede deducir automáticamente la forma de los tensores entre capas, por lo que se debe alimentar la primera capa con los datos, indicar el número de nodos que debe tener cada capa y establecer la función de activación. Se espera que la capa de salida tenga tantas neuronas como clases (Exceptuando la clasificación binaria).

Keras ha tenido diferentes aplicaciones para resolver problemas complejos con datos no estructurados, desempeñándose como framework, facilita la construcción de modelos de aprendizaje en un tiempo muy competitivo.

Dentro del Pipeline definido se requiere guardar las imágenes con el formato *TFRecords*. Este formato almacena las imágenes de forma binaria por lo que el entrenamiento es más rápido especialmente para grandes conjuntos de datos. Los *TFRecords* puede lograr un mejor rendimiento si es utilizar TPUs ya que son mucho más rápidas que las GPUs.

Las *TPUs* fueron diseñado por Google en 2016 específicamente para TensorFlow (Tensor Processing Unit) es utilizado para Inteligencia Artificial.

También es necesario realizar *paralelización* con el fin de mejorar el rendimiento de *CNN*. Las aproximaciones que son utilizadas son:

- La de dimensión de datos, en la cual diferentes núcleos procesan diferentes ejemplos de datos concurrentemente
- La de la dimensión de modelo, en la cual diferentes núcleos trabajan diferentes partes del modelo, es decir, varios núcleos comparten el trabajo computacional de una entrada.

El que una aproximación sea mejor que la otra depende de la arquitectura del modelo y de que tan bien escala el número de unidades de procesamiento, aunque es común sacar provecho de ambas. En general la aproximación de dimensión del modelo es eficiente con cantidades altas de cómputo por neurona, mientras que la aproximación de dimensión de datos es eficiente cuando la cantidad de computación por pesos es alta.

Dado que en las CNNs las capas convolucionales contienen cerca del 90-95% de la computación con cerca del 5% de los parámetros, mientras que las capas completamente conectadas contienen cerca del 5 al 10% del cómputo con cerca del 95% de los parámetros, se muestra más apropiado que se use la aproximación de datos para las primeras (Convolutional layers) y la aproximación de modelo para las últimas (Fully-connected layers).

Otro aspecto importante a tener en cuenta es el del tamaño de Batch, ya que se puede pensar que un tamaño de Batch mayor haría más efectiva la paralelización, sobre todo en la aproximación de datos. Sin embargo, tamaños grandes de Batch disminuyen la convergencia del modelo e incluso pueden afectar negativamente la calidad del resultado final.

#### *Estrategias de paralelización:*

La secuencia natural del algoritmo de descenso estocástico por gradientes para la actualización de los pesos se constituye en una complicación para el uso de plataformas de multiproceso. Sin embargo, existen varias estrategias de implementación, dentro de las cuales reseñamos algunas.

- Estrategia híbrida (Hybrid). Usa ambas aproximaciones. La de dimensión de datos en las capas densas y la de dimensión del modelo en las capas completamente conectadas.
- Gradiente Estocástico promedio (Averaged stochastic gradient). Cada núcleo procesa una porción de los datos y calcula los respectivos pesos gradientes devolviéndolos al máster, quien los consolida y actualiza los nuevos pesos que son nuevamente enviados a los núcleos para una nueva iteración. La convergencia de esta estrategia es un poco peor que la de la aproximación secuencial, pero el tiempo de entrenamiento es sustancialmente inferior.
- Gradiente Estocástico Retardado (Delayed stochastic gradient).

## Metodología de investigación

La metodología utilizada para este proyecto es Team Data Science Process (TDSP), que es una metodología ágil e iterativa de ciencia de datos para entregar soluciones de análisis predictivo y aplicaciones inteligentes de manera eficiente. TDSP tiene 4 componentes principales:

Componentes	Subcomponentes	Donde encontrarlo
Data Science LifeCycle	Bussiness Understanding	Pregunta de investigación y objetivos del proyecto
	Data Acquisition and understanding	Análisis de los datos
	Modelling and Deployment	Uso y herramientas de big data
Standardized Project Structure		Entregables
Infraestructure Resources		Uso y herramientas de big data; Entregables
Tools and Utilities		Uso y herramientas de big data; Entregables

## Análisis de los datos

Gracias al procesamiento que fue realizado con TensorFlow y la lectura de las imágenes por medio de TFRecords se pudo encontrar que las etiquetas con las cuales se deben trabajar las clasificaciones de imágenes son:

	Class	label
0	storm_damage	0
1	drydown	1
2	endrow	2
3	water	3
4	nutrient_deficiency	4
5	double_plant	5
6	waterway	6
7	weed_cluster	7
8	planter_skip	8

Table 1: Etiquetas para Clasificación

Análisis de imágenes multiespectrales para detección de patrones en Agricultura

Gracias a la carga y procesamiento que se realizó con Tensor Flow, se puede realizar una visualización de las etiquetas presentadas en la tabla 1:

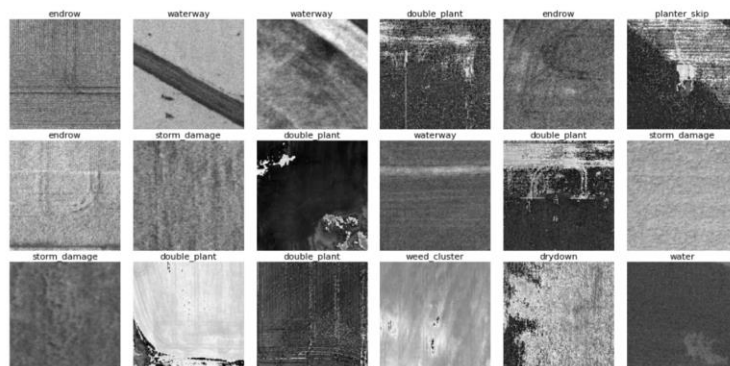



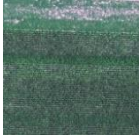
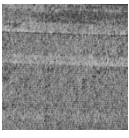


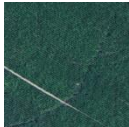
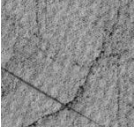


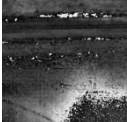



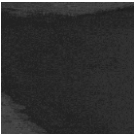

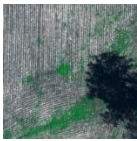
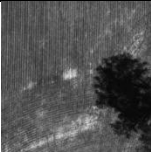

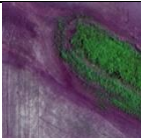
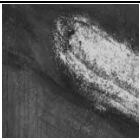

Figure 1: Visualización de imágenes de acuerdo a sus etiquetas

Para cada sección del campo agrícola fotografiado se presentan cinco variantes de la misma imagen distribuidas en los siguientes grupos:

1. **RGB**: Clasificada en el grupo basado en la adicción de colores lumínicos primarios
2. **NIR**: Near infrared, correspondiente a la imagen del espectro infrarrojo.
3. **Field labels**: Contiene cuadriláteros demarcando el área de la fotografía que presenta una afectación característica en particular.
4. **Field bounds**: Contiene la imagen bicromática que discrimina que porción del área de la fotografía corresponde a cultivo y cual a algo diferente. (i.e. Una Carretera)
5. **Field masks**: Contiene imagen bicromática que determina si hay pixeles inválidos en la imagen.

Algunos ejemplos son:

Característica: Doble plantación				
field_labels/double_plant/	field_images/rgb/	field_images/nir/	field_bounds/	field_masks/
Característica: Desección				
field_labels/drydown/	field_images/rgb/	field_images/nir/	field_bounds/	field_masks/
Característica: Surco				
field_labels/endrow/	field_images/rgb/	field_images/nir/	field_bounds/	field_masks/

Característica: Falta de nutrientes				
nutrient_deficiency/	field_images/rgb/	field_images/nir/	field_bounds/	field_masks/
				
Característica: Salto de siembra				
field_labels/planter_skip/	field_images/rgb/	field_images/nir/	field_bounds/	field_masks/
				
Característica: Daño por tormenta				
field_labels/storm_damage/	field_images/rgb/	field_images/nir/	field_bounds/	field_masks/
				
Característica: Agua estancada				
field_labels/water/	field_images/rgb/	field_images/nir/	field_bounds/	field_masks/
				
Característica: Malezas				
field_labels/weed_cluster/	field_images/rgb/	field_images/nir/	field_bounds/	field_masks/
				
Característica: Camino del agua				
field_labels/waterway/	field_images/rgb/	field_images/nir/	field_bounds/	field_masks/
				

## Uso de herramientas de Big Data

Para empezar con el proyecto, se realizó un diseño de arquitectura de cómo se realizaría el flujo de los datos dentro de AWS Educate, como se puede apreciar en la figura 2:

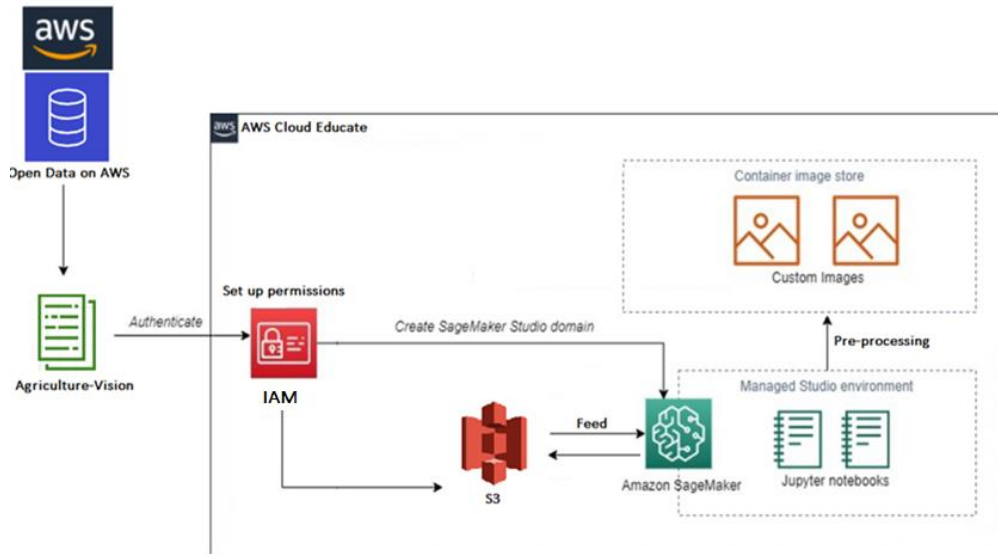


Figure 2: Diseño arquitectura AWS

Una de las primeras estrategias para el uso trabajar con big data es por medio de La paralelización. La paralelización es un tipo de computación en la cual muchos cálculos o procesos se ejecutan simultáneamente. La eficiencia de estas estrategias consiste en maximizar el trabajo de cada núcleo o trabajador al que el máster le asigna una porción del trabajo total.

Dos aproximaciones de paralelismo son utilizadas en el objetivo de mejorar el rendimiento de las CNNs, a saber:

- La de dimensión de datos, en la cual diferentes núcleos procesan diferentes ejemplos de datos concurrentemente
- La de la dimensión de modelo, en la cual diferentes núcleos trabajan diferentes partes del modelo, es decir, varios núcleos comparten el trabajo computacional de una entrada.

El que una aproximación sea mejor que la otra depende de la arquitectura del modelo y de que tan bien escala el número de unidades de procesamiento, aunque es común sacar provecho de ambas.

En general la aproximación de dimensión del modelo es eficiente con cantidades altas de cómputo por neurona, mientras que la aproximación de dimensión de datos es eficiente cuando la cantidad de computación por pesos es alta.

Dado que en las CNNs las capas convolucionales contienen cerca del 90-95% de la computación con cerca del 5% de los parámetros, mientras que las capas completamente conectadas contienen cerca del 5 al 10% del cómputo con cerca del 95% de los parámetros, se muestra más apropiado que se use la

aproximación de datos para las primeras (Convolutional layers) y la aproximación de modelo para las últimas (Fully-connected layers).

Otro aspecto importante a tener en cuenta es el del tamaño de Batch, ya que se puede pensar que un tamaño de Batch mayor haría más efectiva la paralelización, sobre todo en la aproximación de datos. Sin embargo, tamaños grandes de Batch disminuyen la convergencia del modelo e incluso pueden afectar negativamente la calidad del resultado final.

La secuencia natural del algoritmo de descenso estocástico por gradientes para la actualización de los pesos se constituye en una complicación para el uso de plataformas de multiproceso. Sin embargo, existen varias *estrategias de implementación*, dentro de las cuales reseñamos algunas.

1. Estrategia híbrida (Hybrid). Usa ambas aproximaciones. La de dimensión de datos en las capas densas y la de dimensión del modelo en las capas completamente conectadas.
2. Gradiente Estocástico promedio (Averaged stochastic gradient). Cada núcleo procesa una porción de los datos y calcula los respectivos pesos gradientes devolviéndolos al máster, quien los consolida y actualiza los nuevos pesos que son nuevamente enviados a los núcleos para una nueva iteración. La convergencia de esta estrategia es un poco peor que la de la aproximación secuencial, pero el tiempo de entrenamiento es sustancialmente inferior.
3. Gradiente Estocástico Retardado (Delayed stochastic gradient). Sugiere la actualización de los parámetros de peso mediante un algoritmo. Usando este algoritmo se separan las muestras por el número de hilos, y cada hilo trabaja en su propio subconjunto de muestras sólo compartiendo un vector común de pesos. A los hilos sólo se les permite actualizar el vector de pesos según la estipulación del algoritmo, y por tanto, cada actualización debe ser retardada. Un ejemplo de este algoritmo es el de Round robin, que es un proceso muy utilizado en redes de comunicación y sistemas de operaciones de carga balanceada.

Se puede pensar que un *tamaño de Batch* mayor haría más efectiva la paralelización, sobre todo en la aproximación de datos. Sin embargo, tamaños grandes de Batch disminuyen la convergencia del modelo e incluso pueden afectar negativamente la calidad del resultado final.

Los *generadores* son una clase especial de función que retorna un iterador perezoso o “lazy iterator”. Estos son objetos sobre los que se pueden efectuar loops como una lista. Sin embargo, a diferencia de las listas, los “lazy operators” no guardan su contenido en memoria.

Su conformación es similar a una típica definición función, excepto por la expresión de python “yield” con que normalmente terminan. “Yield” indica cuando un valor es regresado al llamador, pero a diferencia de return, no abandona la función y, en su lugar, el estado de la función es recordado para continuar en donde iba cuando vuelve a llamarse.

Un ejemplo ilustrador de la utilidad del uso de generadores, es que si el dataset que alimenta el modelo proviene de una función, esta carga en memoria toda la data, pero si se ejecuta con un generador al que se le establece un batch de carga, este retorna al modelo los  $n = \text{batch}$  elementos primeros, el modelo los procesa y vuelve a llamar la función, retornándole esta los siguiente  $n = \text{batch}$  elementos y así sucesivamente, lo que implica que solo carga en memoria una porción del número total de elementos del dataset equivalente al batch.



## Función

Python

```
def csv_reader(file_name):
    file = open(file_name)
    result = file.read().split("\n")
    return result
```

## Generador

Python

```
def csv_reader(file_name):
    for row in open(file_name, "r"):
        yield row
```

El uso yield resulta en un objeto generador

- Comprima imágenes y archivos
  - o Normalmente el ahorro por ancho de banda supera los costos de decodificación con CPU.
- Maneje estructuras de datos eficientes
  - o Un ejemplo son los TFRecords que disminuyen tamaño de datos y hacen más eficiente el cómputo, como se verá mas adelante.
- Pruebe con modelos pequeños (Toy Models)
  - o Es importante hacer ensayos con poca cantidad de datos e ir incrementando a medida que se tiene certeza del funcionamiento apropiado.
- Utilizar Data Augmentation es una forma de regularización
  - o Sin embargo, se debe tener en cuenta que en general conlleva a tener mayor accuracy testeo acosta de alguna disminución en el accuracy de entrenamiento
- Finalmente se debe recordar que Redes Neuronales está casi siempre asociado a grandes volúmenes de datos.

TFRecord es un formato de datos personalizado de TensorFlow. Los archivos son soportados de forma nativa por la API tf.data, soportan conjuntos de datos distribuidos y aprovechan la paralelización de E/S; Internamente utiliza Protocol Buffers[1] para serializar los datos y almacenarlos en bytes, así ocupa menos espacio que permite mantener una amplia cantidad de datos y no solo para almacenarlos si no también para transferirlos.

### La estructura de TFRecord

Los TFRecords(o las grandes matrices numpy) son útiles cuando queremos cargar los datos y estos se tardan demasiado tiempo, entonces en lugar de almacenar los datos dispersos, obligando a los discos a saltar entre bloques, simplemente almacenamos los datos en una disposición secuencial. Podemos visualizar este concepto de la siguiente manera:



Figure 3: Estructura TFRecords

El archivo TFRecord puede verse como un paquete de todas las muestras de datos individuales. Cada muestra de datos se llama Ejemplo, y es esencialmente un diccionario que almacena el mapeo entre una clave y nuestros datos reales.

Ahora, para escribir los datos en TFRecords, primero se tiene que convertir los datos en una Característica. Estas características son entonces los componentes internos de un Ejemplo:

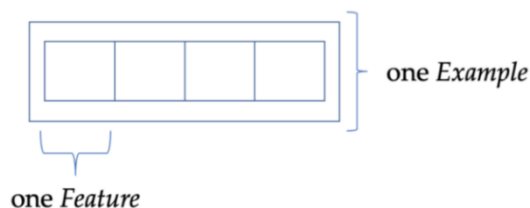


Figure 4: Estructura de un Ejemplo

¿Cuál es la diferencia de almacenar los datos en un array numpy comprimido, o en un archivo pickle? Primero, el archivo TFRecord se almacena de forma secuencial, lo que permite un streaming rápido debido a los bajos tiempos de acceso. En segundo lugar, los archivos TFRecord están integrados de forma nativa en la API tf.data de TensorFlow, lo que permite fácilmente la agrupación por lotes, el barajado, el almacenamiento en caché entre otros.

En caso de que el TFRecord sea demasiado grande, se recomienda particionar el TFRecord en varios archivos entre 10 y 100 MB, esto también sirve para paralelizar la E/S. Como ventaja, para hacer un entrenamiento multitrabajador(multi-worker), se puede distribuir el conjunto de datos entre las máquinas.

Las imágenes son un dominio común en el aprendizaje profundo, hay multitud de formas de llevar las imágenes del disco al modelo: escribiendo un generador personalizado, usando las herramientas incorporadas de Keras, o cargándolas desde un array de numpy. Para hacer eficiente la carga y el análisis de los datos de las imágenes podemos recurrir a TFRecords como formato de archivo subyacente.

### Entrenamiento del modelo.

Implementamos un clasificador de imágenes apoyados de la red neuronal de transfer learning NASNetLarge por la particularidad únicas de las imágenes del dataset de Agriculture-Vision, también se implemento la arquitectura resnet50 y mobilenet donde estas no dieron buenos resultados.

La estrategia que implementamos para el entrenamiento de los modelos fue que utilizamos un dataset en formato de directorio y un formato tipo tfrecord para ver las ventajas de uno con respecto al otro bajo el entrenamiento de diferentes mecanismos físicos (CPU – GPU).

FORMATO DATASET DIRECTORIO	FORMATO DATASET TFRECORD
<ul style="list-style-type: none"> <li>Imágenes-Clasificación (Carpeta). <ul style="list-style-type: none"> <li>Dataset (Carpeta). <ul style="list-style-type: none"> <li>Entrenamiento (Carpeta). <ul style="list-style-type: none"> <li>Imagen categoría 1 (Carpeta). <ul style="list-style-type: none"> <li>Imagen 1.jpg</li> <li>Imagen 2.jpg</li> <li>Imagen 3.jpg</li> </ul> </li> </ul> </li> </ul> </li> </ul> </li> </ul>	

<ul style="list-style-type: none"> <li>○ ...</li> <li>• Imagen categoría 2 (Carpeta). <ul style="list-style-type: none"> <li>○ Imagen 1.jpg</li> <li>○ Imagen 2.jpg</li> <li>○ Imagen 3.jpg</li> <li>○ ...</li> </ul> </li> <li>▪ Prueba (Carpeta). <ul style="list-style-type: none"> <li>• Imagen categoría 1 (Carpeta). <ul style="list-style-type: none"> <li>○ Imagen 1.jpg</li> <li>○ Imagen 2.jpg</li> <li>○ Imagen 3.jpg</li> <li>○ ...</li> </ul> </li> </ul> </li> </ul>	
--	--

En la primera iteración comparamos el tiempo de entrenamiento del modelo bajo el dataset en formato de directorio contra el formato tfrecord en una CPU.

En la segunda iteración también comparamos el tiempo de entrenamiento del modelo bajo el dataset en formato de directorio contra el formato tfrecord en una GPU.

Dando como resultado la siguiente tabla:

TIEMPOS ENTRENAMIENTO			
Épocas de entrenamiento	tipo de dataset	CPU (Horas)	GPU ( Minutos )
10	dataset_from_directory	12 ...	12
	dataset_tfrecord	7 ...	5,23
50	dataset_from_directory	20 horas ...	57.29
	dataset_tfrecord	12 horas ...	26.36

Nos damos cuenta de que al trabajar el entrenamiento de la red neuronal con el dataset en formato tfrecord es mas optimo y sobrepasa por mucho tiempo al formato tipo directorio. También notamos que al trabajar bajo diferentes mecanismos físicos en este caso CPU y GPU es más adecuado trabajar este tipo de paradigmas en GPU.

En la tercera iteración implementamos una estrategia de entrenamiento distribuido por medio de la API de tensorflow “tf.distribute.Strategy”. Esta nos proporciona una abstracción para distribuir su entrenamiento a través de múltiples unidades de procesamiento. El objetivo es permitir habilitar el entrenamiento distribuido utilizando las GPU disponibles en los servidores donde se entrenan los modelos, básicamente, copia todas las variables del modelo a cada procesador.

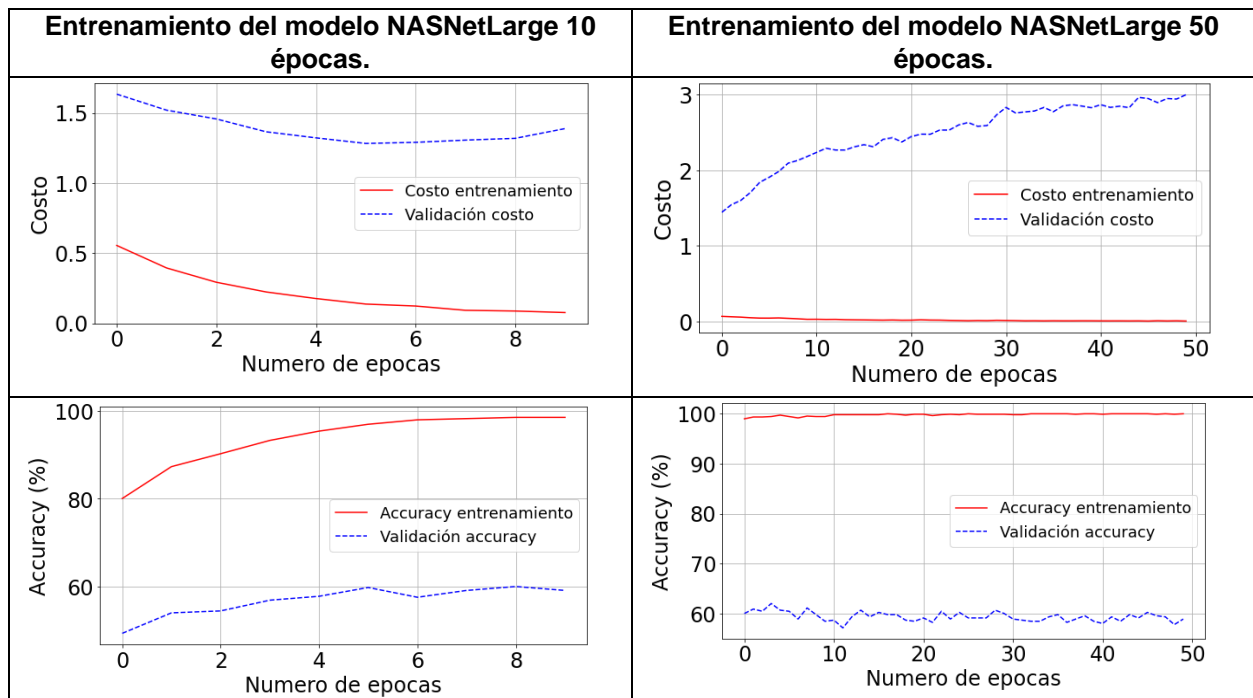
En nuestro caso utilizamos una estrategia llamada “MirroredStrategy” que es una de muchas estrategias de distribución disponible de la api de tensorflow. Cuando se entrena un modelo con múltiples GPUs, se puede utilizar la potencia de cálculo adicional de forma eficaz aumentando el tamaño del lote. En general, utilizando el mayor tamaño de lote que se ajuste a la memoria de la GPU y ajuste la tasa de aprendizaje en consecuencia del modelo.

Los resultados fueron los siguientes:

TIEMPOS ENTRENAMIENTO EN MINUTOS		
Épocas de entrenamiento	Tipo de dataset	GPU strategy ( Minutos )
50	dataset_from_directory	62,5
	dataset_tfrecord	55

Efectivamente al trabajar el entrenamiento del modelo en el formato tfrecord supera al formato de directorio, en nuestro caso solo utilizamos una sola GPU, pero al trabajar como mas estos tiempos se disminuyen.

A continuación, mostramos las gráficas del comportamiento de la función de costo y accuracy del modelo en el momento del entrenamiento y validación de este.



Notamos que esta arquitectura funciona muy bien para pocas épocas evitando el sobre ajuste del modelo y llegando a un accuracy alrededor de 60 % de precisión.

## Entregables y su descripción

Nuestro proyecto puede ser consultado en el siguiente repositorio de GitHub:

<https://github.com/chucho111/Mineria-de-datos-para-grandes-volumenes-de-informaci-n>.

## Conclusiones

1. TFRecord más rápido que directorio por ser binario y esta soportado nativamente por la API.
2. El uso de TFRecords Particionado es mejor debido a que utiliza paralelización de entrada y salida.
3. El uso de la GPU para entrenamiento de modelos de Deep Learning es lo ideal porque se logra llevar a cabo miles de operaciones en paralelo.
4. El uso de una estrategia distribuida para entrenar los modelos demuestra ser competitivas para la aceleración de entrenamiento del modelo.
5. La infraestructura es costosa y de configuración minuciosa, por lo tanto, AWS Educate no soporto este proyecto.
6. Al pasar el Dataset de los datos al formato TFRecord es costoso a nivel de cómputo, sin embargo, vale la pena ya que los resultados se ven al entrenar el modelo.

## Trabajos Futuros

1. Entrenar los modelos con otras estrategias disponibles en la API de Tensor Flow utilizando más de una GPU.
2. Entrenar los modelos en una o varias TPU utilizando las estrategias disponibles de la API de Tensor Flow.

## Ejecución del plan

Dentro del anteproyecto inicial se planeo una ejecución del plan el cual se ejecuto de la siguiente manera:

Plan Propuesto	Plan Real - Comentarios	Lecciones Aprendidas
Planteamiento del problema	Se realizo una investigación sobre los problemas que podían existir en los cultivos. Fue necesario realizar investigación de cómo manejar imágenes RGB, NIR y cómo es posible combinar estas. También se realizó una investigación de metodologías para el procesamiento de imágenes.	
Exploración de datos	Dentro de la exploración del Dataset, se cumplió con el tiempo establecido, tambien se investigaron técnicas de exploración de un Dataset de imágenes.	
Diseño Arquitectura	Se cambio esta fase con la investigación de herramientas, ya que para el test de algunas herramientas era necesario crear estructura de carga en S3 para poder alimentar las herramientas. Sin embargo, se realizó una propuesta de la arquitectura en AWS con el fin de poder tener un plan de investigación de herramientas necesarias.	Entender las limitantes del ambiente en el que se trabaja, como lo es aws Educate.

Investigación de herramientas	Esta fase estuvo combinada con el diseño de la arquitectura, ya que era necesario entender que herramientas podían ser utilizadas en el ambiente educare y sus posibles limitantes. Esta fase se tomó alrededor de 3 semanas adicionales, ya que a medida que se leían los trabajos más recientes se veía que existían más técnicas y herramientas que ayudarían a este proyecto.	Contar con SME del procesamiento de imágenes hubiera ayudado a que esta fase fuera más productiva.
Entrenamiento y despliegue	Dentro del entrenamiento existieron varios problemas ya que dentro del plan principal no se tenía el uso de TFRecords para procesamiento de imágenes, por lo que se estaban generando problemas con la carga de todas las imágenes.	

## Implicaciones éticas

La aplicación de tecnologías en el ambiente de la agricultura puede dar riesgos socio éticos como es la dependencia de la tecnología, monocultivo, aumento en la brecha digital, posible concentración y manipulación de datos, incluido la dependencia por parte de los agricultores de insumos externos suministrados por proveedores de tecnologías, amenaza contra la sostenibilidad de los pequeños agricultores locales, control y prácticas desleales.

Entre los principales riesgos éticos asociados en la inclusión de tecnologías en el ámbito de la agricultura esta la seguridad alimentaria y las consideraciones ecológicas, desarrollo sostenible, la necesidad de tener en cuenta las necesidades de las generaciones futuras, cambios sociales que se puedan desencadenar por la adopción nuevas tecnologías, la brecha social entre pequeños y grandes agricultores, tanto dentro del país como fuera sobre todo en los países en vía de desarrollo dado el costo de incluir estas tecnologías, el equilibrio entre el aumento de la productividad y la eficiencia para la agricultura tradicional y la sostenibilidad del medio ambiente.

## Aspectos legales y comerciales

Lo primero a tener en cuenta para poder realizar este proyecto son los términos de uso especificados para el uso del Dataset Agriculture-vision, en el cual se especifica que el conjunto de datos puede ser descargado y utilizado siempre y cuando no se utilice con fines comerciales. Tampoco se puede vender, transferir o licenciar el Dataset a terceros (excepto a colegas de investigación), ni crear ningún trabajo derivado del Conjunto de Datos que no sea el desarrollo de aplicaciones de investigación no comerciales

Gracias a los drones en la agricultura, se puede verificar las características de las plantaciones a través del análisis de imágenes multiespectrales, donde se representan niveles de reflectancia de luz obteniendo mapas de colores que indican las diferentes características de los cultivos y que permiten:

1. Explorar los campos en menos tiempo.
2. Tomar decisiones por medio de los mapas de color generados por los drones.
3. Conocer si el campo a mejorado o desmejorado desde su última revisión.

Se conoce que el sector agropecuario ha tenido un gran crecimiento en Colombia donde contribuye con el 6.8% del PIB, por lo que el uso de drones para detectar anomalías en las características fundamentales que puedan afectar el producto final es crucial en los mercados de hoy en día, permitiendo así mejorar las

prácticas de manejo del cultivo y dando la posibilidad de tener reacciones rápidas que eviten pérdidas y mejoren productividad.

### Bibliografía

1. Google (2021). *Data Preprocessing for ML with Tensor Flow [Handbook]*. Recuperado de: <https://cloud.google.com/architecture/data-preprocessing-for-ml-with-tf-transform-pt2>
2. Tensor Flow (2021). *Images segmentations [Tutorials]*. Recuperado de: <https://www.tensorflow.org/tutorials/images/segmentation>
3. Karthikeyan Vijayan (2020). *Image classification using tfrecords [Python Notebook]*. Recuperado de: <https://www.kaggle.com/karthikeyanvijayan/image-classification-using-tfrecord>
4. Yan Gobeil (2021). *What Is the Best Input Pipeline to Train Image Classification Models with tf.keras? [Magazine]*. Recuperado de: <https://towardsdatascience.com/what-is-the-best-input-pipeline-to-train-image-classification-models-with-tf-keras-eb3fe26d3cc5>
5. Universidad Campus Nord (2018). *Deep learning inteligencia artificial con Keras [Libro]*. Recuperado de: [https://torres.ai/deep-learning-inteligencia-artificial-keras/#Redes\\_densamente\\_conectadas\\_en\\_Keras](https://torres.ai/deep-learning-inteligencia-artificial-keras/#Redes_densamente_conectadas_en_Keras)
6. DataCamp (2021). *Deep learning with Keras Python [Curso]*. Recuperado de: <https://learn.datacamp.com/courses/deep-learning-with-keras-in-python>
7. Tensor Flow (2021). *TfRecords Dataset [Documentation]*. Recuperado de: [https://www.tensorflow.org/api\\_docs/python/tf/data/TFRecordDataset](https://www.tensorflow.org/api_docs/python/tf/data/TFRecordDataset)
8. Tensor Flow (2021). *tfRecords train example [Tutorials]*. Recuperado de: [https://www.tensorflow.org/tutorials/load\\_data/tfrecord](https://www.tensorflow.org/tutorials/load_data/tfrecord)
9. Lesly Sandra (2019). *Diferencia entre CPU, GPU y TPU [Blog]*. Recuperado de: <https://leslysandra.medium.com/cpu-gpu-tpu-e4a686d3dbc9>
10. Developers (2021). *Protocol Buffer Basics: Python [Documentación]*. Recueprado de: <https://developers.google.com/protocol-buffers/docs/pythontutorial>
11. Tensor (2021). *TFRecord train example [handbook]*. Recuperado de: [https://www.tensorflow.org/tutorials/load\\_data/tfrecord](https://www.tensorflow.org/tutorials/load_data/tfrecord)
12. Alex Krizhevsky (2014). *One weird trick for parallelizing convolutional neural networks [Paper]*
13. Data Science blog (2019). *How to make two parallel convolutional neural networks in Keras? [Blog]*. Recuperado de: <https://datascience.stackexchange.com/questions/39407/how-to-make-two-parallel-convolutional-neural-networks-in-keras>
14. Niloy Purkait (2018). *How to train your Neural Networks in parallel with Keras and Apache Spark [Article]*. Recuperado de: <https://towardsdatascience.com/how-to-train-your-neural-networks-in-parallel-with-keras-and-apache-spark-ea8a3f48cae6?gi=ad9f6a57ad9>
15. Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna (2015) *Rethinking the Inception Architecture for Computer Vision. [Article]*. Recuperado de: <https://arxiv.org/abs/1512.00567>