Image2 Circuit - Clasificador de elementos electrónicos usando una CNN mediante Tensorflow

Proyecto Final. Introducción a la programación en paralelo con MPI, OpenMP y CUDA.

Oscar Andres Rosas Hernandez
SoyOscarRH@ciencias.unam.mx
Facultad de Ciencias, UNAM

ABSTRACT

Las redes neuronales convoluciones (CNN) son la herramienta por excelencia en la actualidad para crear un sistema automático que analiza imágenes. En este proyecto buscaré investigar la fiabilidad y los resultados de usar una CNN para poder clasificar imágenes tanto fotografías como imágenes esquemáticas creadas por computadora de diversos elementos eléctricos (se piensa trabajar como mínimo con los 4 elementos básicos: resistores, capacitores e inductores y fuentes de voltaje) usando la librería TensorFlow y haremos una comparativa de la diferencia que causa usar GPU's en el desarrollo de la misma.

I. OBJETIVOS

Se dará una breve introducción a las redes neuronales convoluciones y términos como data aumentation y drop out. Una vez entendida la teoría, se procederá a generar la implementación de un modelo en TensorFlow para Python, medir su rendimiento así como la influencia que tiene usar una GPU (o varias en los tiempos de entrenamiento del mismo. Hablaremos mas sobre la idea de hacer un entrenamiento distribuido usando tensorflow. Finalmente se mostrarán los resultados de evaluación del modelo propuesto aquí.

II. MARCO TEÓRICO

II-A. Machine learning

Definimos al machine Learning como el área que estudia como hacer que las computadoras puedan aprender de manera automática usando experiencias (data) del pasado para predecir el futuro.

La mayoría del aprendizaje automático práctico utiliza aprendizaje supervisado. El aprendizaje supervisado es donde se tiene variable(s) de entrada (x) y una variable de salida (Y) y se utiliza un algoritmo para "aprender" la función que mapea la entrada a la salida.

$$Y = f(X) \tag{1}$$

El objetivo es aproximar la función tan bien que cuando tenga nuevos datos de entrada (X) pueda predecir las variables de salida (Y) para esos datos.

Se llama aprendizaje supervisado porque el proceso de un algoritmo que aprende del conjunto de datos de capacitación puede verse como un profesor que supervisa el proceso de aprendizaje. Conocemos las respuestas correctas, el modelo realiza predicciones de forma iterativa sobre los datos de entrenamiento y es corregido por el profesor.

Durante el entrenamiento, un algoritmo de clasificación recibirá una serie de datos con una categoría asignada. El trabajo de un algoritmo de clasificación es tomar un valor de entrada y asignarle una clase o categoría que se ajuste según los datos de entrenamiento proporcionados. [1]

II-A1. Accuracy: Esta dada por "de todos nuestros datos, que tanto % clasificamos correctamente" En otra palabras:

$$accuracy := \frac{true\_negative + true\_positive}{all} \tag{2}$$

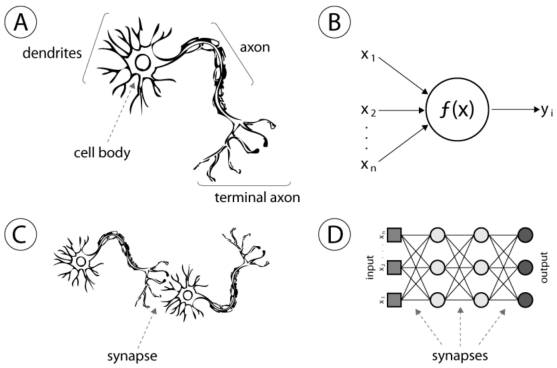
Ahora, el "accuracy" no es siempre la mejor métrica, sobre todo cuando nuestros datos no están balanceados, es decir cuando en cada una de nuestras clases tenemos una cantidad diferente de datos.

II-B. Redes Neuronales

Es una práctica común conceptualizar una red neuronal artificial como una neurona biológica. Técnicamente hablando, una red neuronal artificial, como lo expresó Ada Lovelace, es el "cálculo del sistema nervioso". Una versión muy, muy simplificada del sistema nervioso. Tanto es así que algunos neurocientíficos probablemente han perdido las esperanzas tan pronto como vieron lo sencilla y alejada de la realidad que es.

Las redes neuronales son un subgrupo de la Inteligencia Artificial basado en un sistema de aprendizaje y procesamiento automático inspirado en el funcionamiento del sistema nervioso humano. Más concretamente, pertenecen al subgrupo del Machine Learning, una tecnología basada en crear sistemas que puedan aprender automáticamente, es decir, pueden descubrir patrones complejos enterrados en grandes conjuntos de datos sin la necesidad de interferencia humana.

Las redes neuronales consisten en una capa de entrada, una o varias capas ocultas y una capa de salida. Hay que tener en cuenta que las capas se cuentan desde la primera capa oculta. Cuantas más capas ocultas haya, más compleja es la red.



[3]

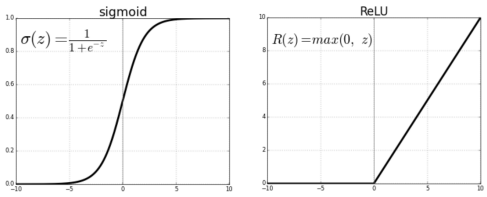
Las sinapsis toman el valor de la entrada almacenada de la capa anterior y la multiplican por un peso (w). Entonces podemos decir que la tarea principal dentro del aprendizaje profundo es calibrar (o encontrar) los pesos a un valor específico para obtener un resultado preciso.

"Es como cuando te estás bañando y hay que mover las dos manijas para encontrar el punto perfecto donde esta la temperatura ideal."

Posteriormente, las sinapsis propagan hacia adelante sus resultados a las capas siguientes. En general se pueden ver como:

$$Y = b + \sum w_i x_i \tag{3}$$

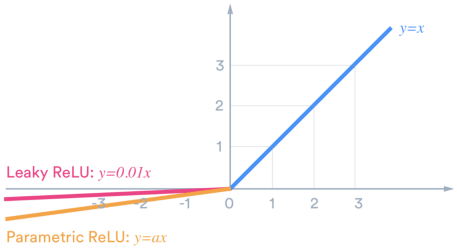
Finalmente tras realizar este calculo y para lograr la no linearlidad que generalmente buscamos se aplican funciones de activación, entre las mas famosas estan la ReLU (y su prima la Leaky ReLU) y la sigmoide.



[3]

[3]

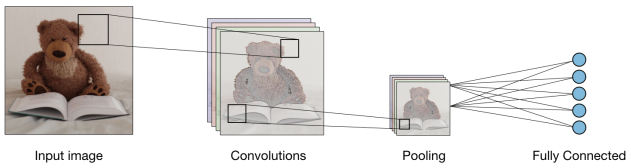
Las redes neuronales se diferencian por topología, dependiendo de las características con las que cuentan: perceptrón monocapa, la más sencilla; perceptrón multicapa, donde se desarrollan capas ocultas y es capaz de eliminar información irrelevante; Red Neuronal recurrente, donde cobra un papel importante la temporalidad y, con ella, se dota a la red neuronal de memoria y, por último, las Redes Neuronales Convolucionales; las más completas.



II-C. Redes Neuronales Convolucionales

Cada parte de una red neuronal convolucional está entrenada para realizar una tarea, por lo que el entrenamiento de cada una de las partes se desarrolla de manera individual y se efectúa más rápido. Estas redes se utilizan en especial para el análisis de imágenes.

Las CNN estan formadas por 3 partes generalmente:



- Las capas convolucionales usa filtros que realizan operaciones de convolución mientras se escanea / pasa la "imagen" entrada, sus hiperparámetros incluyen el tamaño del filtro y el salto que se da, así como que es lo que pasa en las orillas de la imagen de entrada, la salida se llama mapa de características o mapa de activación.
- Agrupación (pooling): Es una operación que busca la disminución de la resolución, generalmente se aplica después de una capa de convolución; produce cierta invariancia espacial. En particular, el max y mean pooling son tipos especiales de pooling donde se toman el valor máximo y promedio, respectivamente.
- Lineares: Totalmente conectada: Funciona en una entrada aplanada donde cada entrada está conectada a todas las neuronas, para esto tenemos que "espagetificar" nuestra entrada, estas si están presentes, se encuentran hacia el final de la arquitectura y se pueden usar para optimizar objetivos como las probabilidades de una clase.

Cada parte de una red neuronal convolucional está entrenada para realizar una tarea, por lo que el entrenamiento de cada una de las partes se desarrolla de manera individual y se efectúa más rápido. Estas redes se utilizan en especial para el análisis de imágenes.

II-D. Dropout

En pocas palabras, el dropout se refiere a ignorar unidades (es decir, neuronas) durante la fase de entrenamiento de cierto conjunto de neuronas que se elige al azar.

Por "ignorar", quiero decir que estas unidades no se consideran durante un pase particular hacia adelante o hacia atrás.

Más técnicamente, en cada etapa de entrenamiento, los nodos individuales se eliminan de la red con probabilidad 1-p o se mantienen con probabilidad p, de modo que queda una red reducida; Los bordes entrantes y salientes a un nodo abandonado también se eliminan.

II-D1. ¿Por qué lo necesitamos?: ¿Por qué necesitamos literalmente apagar partes de una red neuronal? La respuesta a estas preguntas es para evitar el overfitting.

Una capa totalmente conectada ocupa la mayoría de los parámetros y, por lo tanto, las neuronas desarrollan una codependencia entre ellas durante el entrenamiento, lo que frena el poder individual de cada neurona, lo que lleva a un ajuste excesivo de los datos de entrenamiento.

Para esto es que el dropout cae como anillo al dedo.

III. TensorFlow

TensorFlow es una biblioteca de software de código abierto para computación numérica, que utiliza gráficos de flujo de datos. Los nodos en las gráficas representan operaciones matemáticas, mientras que los bordes de las gráficas representan las matrices de datos multidimensionales (tensores) comunicadas entre ellos.

TensorFlow es una gran plataforma para construir y entrenar redes neuronales, que permiten detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos.

La arquitectura flexible de TensorFlow le permite implementar el cálculo a una o más CPU o GPU en equipos de escritorio, servidores o dispositivos móviles con una sola API. TensorFlow fue desarrollado originalmente por investigadores e ingenieros que trabajaban en el equipo de Google Brain Team, dentro del departamento de investigación de Machine Intelligence, con el propósito de llevar a cabo el aprendizaje automático y la investigación de redes neuronales profundas.

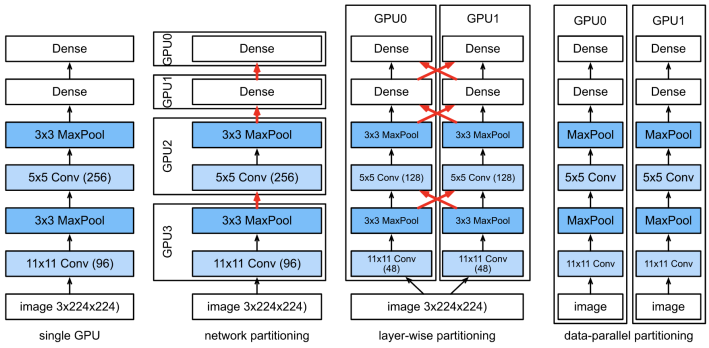
Sin embargo, el sistema es lo suficientemente general como para ser aplicable a una amplia variedad de otros dominios igualmente.

TensorFlow se construyó pensando en el código abierto y en la facilidad de ejecución y escalabilidad. Permite ser ejecutado en la nube, pero también en local. La idea es que cualquiera pueda ejecutarlo. Se puede ver a personas que lo ejecutan en una sola máquina, un único dispositivo, una sola CPU (o GPU) o en grandes clústeres.

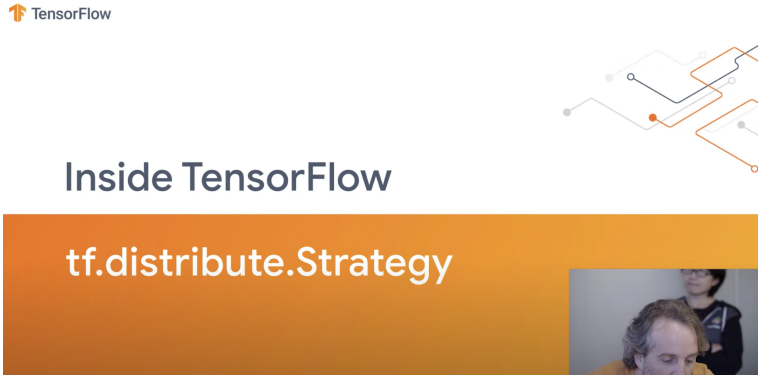
IV. TensorFlow Distribuido con muchas GPUs

Generalmente hay dos formas de distribuir la computación en múltiples dispositivos:

- Paralelismo de datos,
  Donde un solo modelo se replica en múltiples dispositivos o múltiples máquinas. Cada uno de ellos procesa diferentes lotes de datos, luego fusionan sus resultados. Existen muchas variantes de esta configuración, que difieren en la forma en que las diferentes réplicas del modelo combinan los resultados, en si permanecen sincronizadas en cada lote o si están acopladas más libremente, etc.
- Paralelismo de modelos,
  Donde diferentes partes de un solo modelo se ejecutan en diferentes dispositivos, procesando un solo lote de datos juntos. Esto funciona mejor con modelos que tienen una arquitectura paralela natural, como los modelos que cuentan con múltiples ramas.



Recomiendo mucho ver este video para entender mejor este vídeo:



https://youtu.be/jKV53r9-H14

IV-A. Map Reduce

El paradigma de programación de MapReduce se basa en el concepto de pares clave-valor. También proporciona paradigmas poderosos para el procesamiento de datos en paralelo. Para procesar datos en MapReduce, debe poder asignar una entrada dada y la salida esperada en el paradigma MapReduce, es decir, tanto la entrada como la salida deben asignarse en el formato de múltiples pares clave-valor. Un único par de valores clave también se conoce como registro.

Por ejemplo, tiene un archivo de texto 'input.txt' con 100 líneas de texto y desea averiguar la frecuencia de aparición de cada palabra en el archivo. Cada línea en el archivo input.txt se considera como un valor y el desplazamiento de la línea desde el inicio del archivo se considera como una clave, aquí (desplazamiento, línea) es un par clave-valor de entrada. Para

contar cuántas veces se produjo una palabra (frecuencia de palabra) en input.txt, una sola palabra se considera como una clave de salida y la frecuencia de una palabra se considera como un valor de salida. Nuestro valor-clave de entrada es (desplazamiento de una línea, línea) y el valor-clave de salida es (palabra, frecuencia de palabra).

Un trabajo Map-Reduce se divide en cuatro fases simples:

- Fase de mapa
- Fase de combinación
- Fase aleatoria
- Fase de reducción.

En nuestro ejemplo de conteo de palabras, la fase combinar y reducir realiza la misma operación de agregar frecuencia de palabras.

Está inspirado en el map y reduce; las funciones comúnmente utilizadas en la programación funcional, aunque su propósito en el marco de MapReduce no es el mismo que en sus formas originales.

IV-A1. Como funciona: En cada paso del entrenamiento:

El lote actual de datos (llamado lote global) se divide en n (supongamos 8 por el resto del ejemplo) sub-lotes diferentes (llamados lotes locales).

Por ejemplo, si el lote global tiene 512 muestras, cada uno de los 8 lotes locales tendrá 64 muestras. Cada una de las 8 réplicas procesa de forma independiente un lote local: ejecutan un paso hacia adelante, luego un paso hacia atrás, generando el gradiente de los pesos con respecto a la pérdida del modelo en el lote local.

Las actualizaciones de peso originadas en gradientes locales se fusionan de manera eficiente en las 8 réplicas. Como esto se hace al final de cada paso, las réplicas siempre permanecen sincronizadas.

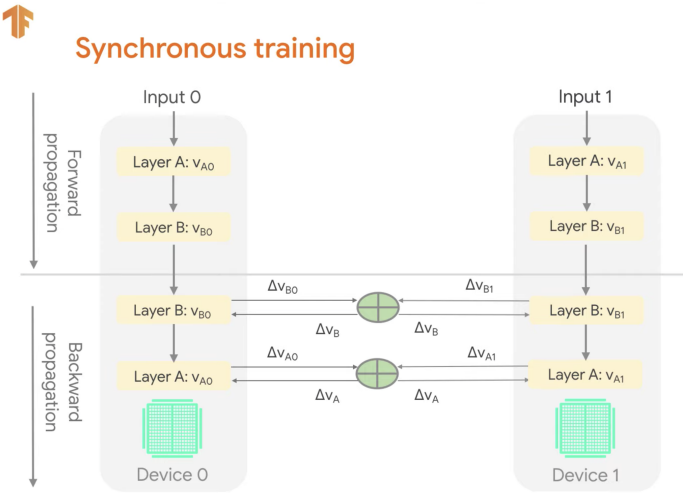
En la práctica, el proceso de actualización sincrónica de los pesos de las réplicas del modelo se maneja a nivel de cada variable de peso individual. Esto se hace a través de un objeto variable reflejado.

IV-B. Mirror

`tf.distribute.MirroredStrategy` admite entrenamiento distribuido sincrónico en múltiples GPU en una máquina. Este crea una réplica por GPU. Cada variable en el modelo se refleja en todas las réplicas. Juntas, estas variables forman una sola variable conceptual llamada `MirroredVariable`.

Estas variables se mantienen sincronizadas entre sí mediante la aplicación de actualizaciones idénticas. Se utilizan algoritmos eficientes de reducción total para comunicar las actualizaciones variables a través de los dispositivos. Reduzca todos los tensores de agregados en todos los dispositivos agregándolos y los pone a disposición en cada dispositivo.

Es un algoritmo fusionado que es muy eficiente y puede reducir significativamente la sobrecarga de sincronización. Hay muchos algoritmos e implementaciones de reducción total disponibles, dependiendo del tipo de comunicación disponible entre dispositivos. Por defecto, usa `NVIDIA NCCL` como la implementación de reducción total. Se puede elegir entre algunas otras opciones que ofrecemos, o escribir la nuestra.



Hay varias opciones que podemos usar a la hora de hacer un reduce:

- `tf.distribute.HierarchicalCopyAllReduce` Se reduce a una GPU a lo largo de los bordes en alguna jerarquía y se transmite a cada GPU a lo largo de la misma ruta.
  Antes de realizar la reducción total, los tensores serán reempaquetados o agregados para un transporte más eficiente entre dispositivos.
  Esta es una reducción creada para Nvidia DGX-1 que supone que las GPU se conectan así en la máquina DGX-1. Si tiene diferentes interconexiones de GPU, es probable que sea más lento que tf.distribute.ReductionToOneDevice.
- `tf.distribute.ReductionToOneDevice` Always do reduction to one device first and then do broadcasting.
- `tf.distribute.NcclAllReduce` Reducción usando NCCL all-reduce.

V. El problema

Para este proyecto buscaré investigar la fiabilidad y los resultados de usar una red neuronal profunda convolucional (CNN) para poder clasificar imagenes (tanto fotografías como imagenes esquemáticas readas por computadora) de diversos elementos eléctricos (se piensa trabajar como mínimo con los 3 elementos básicos: resistores, capacitores e inductores) con un alto grado de confiabilidad (esperamos que dado un dataset balanceado de prueba obtener al menos un 85 % de exactitud).

Para experimentar y llegar un modelo final en este trabajo se usará la librería Tensor Flow, y Google Colab (dadas las limitaciones de conexion y de los equipos de cómputo que tengo en el momento) para poder iterar, durante este y dado el largo proceso de entrenamiento que es común que dure de

20 minutos a una hora para 40 iteraciones sobre un dataset pequeño buscaré usar las capacidades de tensor flow para realizar esta tarea en paralelo así como medir diferencias en el tiempo de entrenamiento usando sólo CPUs, usando GPUs y usando multiples GPUs.

V-A. Motivación

Este trabajo servirá como pivote para poder empezar con el plan a más largo plazo de mi trabajo terminal, medio por el cual me titulare de la carrera de Ingeniería en Sistemas Computacionales de la ESCOM IPN, esto dado que estoy haciendo ambas carreras de manera simultánea (junto con Ciencias de la Computación en la Facultad de Ciencias), pero en la primera me encuentro ya en la recta final.

El objetivo final de dicho trabajo terminal será ocupar el modelo que estoy proponiendo y que apoyado con otros dos sistemas: uno que permita encontrar componentes de una imagen, basado en la arquitectura de YOLO y otro que permita encontrar como se relacionan (si existe conexion directa entre cualesquiera dos elementos), junto con el modelo que estoy creando que permita clasificar con fiabilidad a los componentes electrónicos para poder crear un grafo abstracto de conexion y usarlo para poder crear automáticamente esquemáticos a partir de la imagen de entrada o crear un archivo de simulación (por ejemplo para abrirlo en proteus) de manera automática.

V-B. Justificación

Cualquier disposición de las diversas fuentes de energía eléctrica junto con los diferentes elementos del circuito se denomina red eléctrica (circuito eléctrico / electrónico), los circuitos electrónicos son representados esquemáticamente con diferentes estándares tales como el ANSI o el IEEE.

Estos esquemas se ocupan en los diferentes simuladores que hay en el mercado los cuales nos permiten visualizar los resultados generados por dicha red.

Además, generalmente es necesario generar un reporte técnico y para realizarlo se utilizan desde los mismos simuladores hasta aplicaciones de dibujo para crear esquemas. Buscamos crear una herramienta de apoyo a la hora de digitalizar circuitos electrónicos minimizando el tiempo que el usuario dedica.

Existen al menos 12 carreras de licenciatura en el Instituto Politécnico Nacional en las que tienen mínimo una unidad de aprendizaje relacionada directamente con circuitos electrónicos, al realizar diseños para dichas UA suele ser necesario simular los resultados o generar un reporte técnico con esquemas, este proceso donde se involucra la escritura a mano, el realizar la simulación y el reporte suele ser ocupar varias horas de trabajo, dependiendo de la experiencia con el simulador.

Generar un sistema para ayudar a la digitalización de dicho circuito nos minimizará tiempos de realización de prácticas y la posibilidad de realizar una simulación y obtener resultados en un menor tiempo.

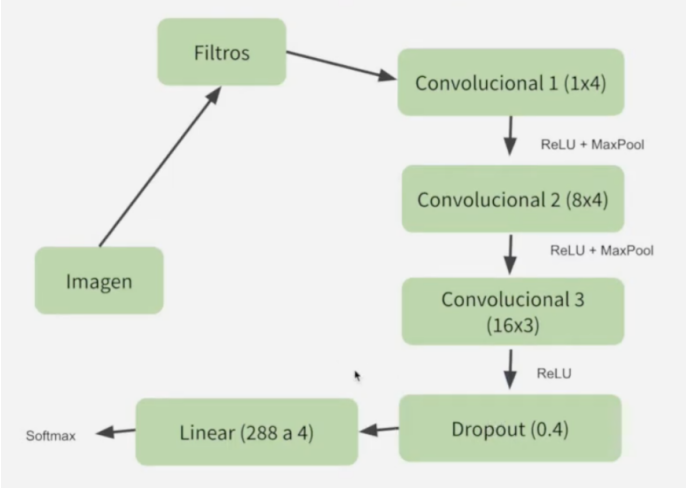
V-C. El dataset

Planeo usar como base el dataset que existe en Kaggle: https://www.kaggle.com/xuowen/circut-components/kernels eso si, hagase notar que no existe al parecer otro dataset disponible con las caracteristicas necesarias para desarrolar el projecto, por lo que se puede usar este como base apoyarnos con el mismo pero aumentarlo para tener mas datos (o mas clases incluso).

De hecho se uso la técnica conocida como data aumentation para crear un modelo mas resiliente.

V-D. El modelo que se uso para resolverlo

La idea para este proyecto fue tomar las ideas que he estado realizando durante las últimas semanas en pytorch y pasarlos a TensorFlow con Keras, replicar la arquitectura que ya existe que es:



Esta es una arquitectura bastante basica y clasica para cnn:

```
model = Sequential([
  Conv2D(8, 4, activation=tf.nn.relu, input_shape=(100, 100, 1)),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(16, 4, activation=tf.nn.relu),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(32, 3, activation=tf.nn.relu),
  Flatten(),
  Dropout(0.4),
  Dense(32, activation=tf.nn.leaky_relu),
  Dense(4, activation=tf.nn.softmax),
])

loss = tf.keras.losses.CategoricalCrossentropy()
model.compile(optimizer='Nadam', loss=loss, metrics=['accuracy'])
model.summary()
```

V-E. Experimientos y resultados

Finalmente veamos que es lo que ha pasado.

Hemos creado 3 entrenamientos para el sistema, debido a las limitaciones del sistema, no fui capaz de probarlo como me gustaria en un sistema conv arias gpus, asi que tuve que usar gpus virtuales, con esto esperamos un resultado similiar en los primeros dos experimentos.

Usando muchas GPUs:

```
import time

strategy = tf.distribute.MirroredStrategy(
  cross_device_ops=tf.distribute.ReductionToOneDevice())
print('Number of devices: {}'.format(strategy.num_replicas_in_sync))

with strategy.scope():
  model = create_model()

start = time.time()
model.fit(train_data, epochs=35)
end = time.time()

distribute = end - start
print(f"Elapsed (distribute) = {distribute}")
```

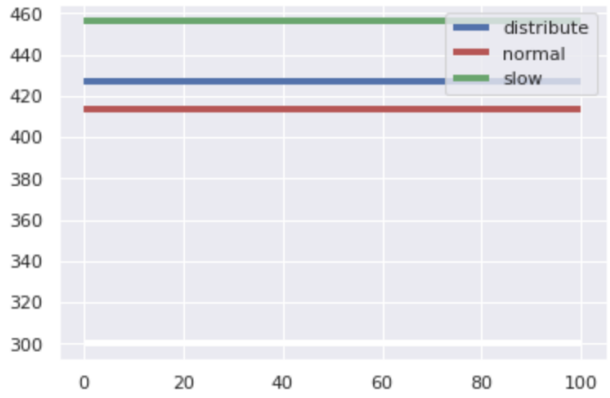
Usando una GPU:

```
model = create_model()

start = time.time()
model.fit(train_data, epochs=35)
end = time.time()

normal = end - start
print(f"Elapsed (normal) = {normal}")
```

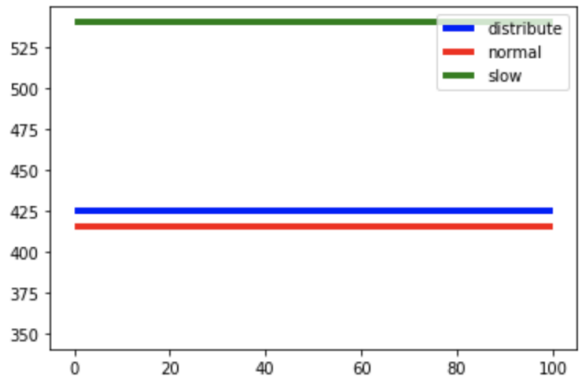
Usando una CPU:

```
with tf.device('/CPU:0'):
  model = create_model()
  start = time.time()
  model.fit(train_data, epochs=35)
  end = time.time()

slow = end - start
print(f"Elapsed (slow) = {slow}")
```

Este es el resultado de los 3 metodos de ejecución USANDO usar GPU para hacer el preprocesamiento:



Este es el resultado de los 3 metodos de ejecución sin usar GPU para hacer el preprocesamiento:



Al final todo el codigo del modelo puede encontrarse en:

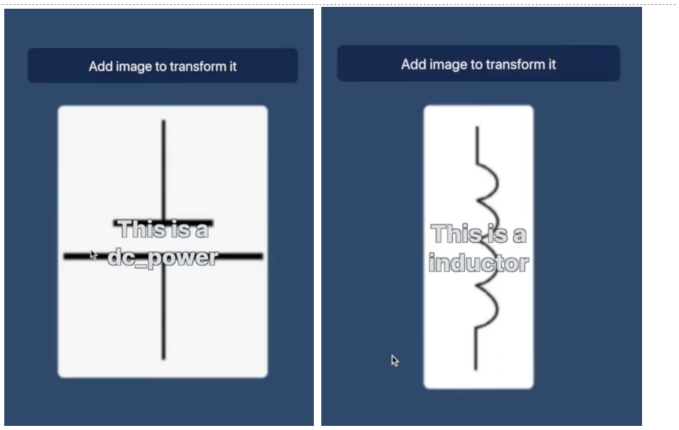
https://colab.research.google.com/drive/1qH6eWiQR4_0A-tzwD5LVAmN1BpcpuSh4?usp=sharing

V-F. Modelo final y deploy

El resultado esperado es ocurrio, pudimos crear un modelo que pueda ser usado para predecir correctamente la clase a la que pertenece un componente electrónico con más de un 80 % de exactitud (dado un dataset de pruebas balanceado y parecido al que usamos durante el entrenamiento pero si repetir ninguna imagen en ambos).

Se pudo hacer deploy de este modelo usando un sistema cliente servidor muy básico o incluso usando tensorflow.js poder crear una pwa (progressive web app) que no requiera de un servidor para ejecutarse.

Al final todo el codigo del sistema de deploy final puede encontrarse en:



https://github.com/SoyOscarRH/Image2Circuit

V-F1. Discusión: Actualmente no existe una forma de hacer procesamiento en paralelo para la data aumentation, pero eso será un gran siguiente paso y algo que me alegra mucho de ver que estan trabajando.

Finalmente es impresionante la forma en la que usar una GPU afecta a la velocidad de entrenamiento, haciendola muchas veces mas veloz, se esperaria ver un cambio similiar al usar varias gpus pero me temo que sin el equipo necesario para probarlo esta idea no fue ni probada ni descartada, solo se hace la recomendacion a los siguientes estudiantes que le echen un ojo a el sistema distruido de tensorflow.

Referencias

[1] *Aidan Wilson*, Sep 29, 2019
https://towardsdatascience.com/a-brief-introduction-to-supervised-learning
[2] June 1, 2020
https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks
[3] June 1, 2020
https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
[4] Multi-GPU and distributed training
https://keras.io/guides/distributed_training