IELE-4014: Machine Learning - Tarea 6

Juan Pablo Naranjo Cuellar
201730006

Isabella Salgado Pinzón
201730418

10 de diciembre de 2021

Modelo de clasificación de imágenes

El objetivo de esta tarea fue aprender a utilizar la herramienta *Edge Impulse* para entrenar y evaluar una red neuronal convolucional capaz de predecir el objeto representado en una imagen de 96x96 píxeles. En este caso, se quiso diseñar una red convolucional capaz de distinguir entre dos objetos mostrados en imágenes: una lámpara o un esfero. La herramienta de *Edge Impulse* es muy versátil en el sentido en que le permite al usuario seleccionar los parámetros de la red neuronal que desea entrenar y además le permite usar los datos de entrenamiento que este desee. En este caso, los datos fueron recolectados a partir de la cámara de un smartphone. Se tomaron 50 fotografías de una lámpara, 50 fotografías de un esfero (en diferentes ángulos y acercamientos) y 50 fotografías de objetos que no eran ni una lámpara ni un esfero. De esta forma, la herramienta *Edge Impulse* se encargó de realizar la partición de los datos en conjuntos de entrenamiento y de prueba.

El objetivo de esta red neuronal convolucional fue entonces predecir si la imagen que se le presentaba era una lámpara, un esfero o ninguno de los anteriores. El entrenamiento se llevó a cabo en los servidores de la nube de *Edge Impulse* y los resultados se mostraron en el *dashboard* de la herramienta. Otra ventaja de usar *EI* es el hecho de que esta permite generar un cuaderno de Jupyter con los datos tomados y el modelo de red convolucional entrenado usando *Tensorflow* y *Keras*. Para aprovechar esta gran ventaja, se realizó una comparación entre el rendimiento del entrenamiento en la nube de *EI* con el entrenamiento hecho localmente en un computador.

Las redes neuronales convolucionales son ampliamente utilizadas en procesamiento de imágenes por su alta precisión. El proceso de clasificación de imágenes con una red convolucional se muestra en la figura 1. Para entrenar dicha red se utilizará *transfer learning*, un método de Machine Learning donde se toma un modelo existente y se aplica en una nueva tarea. Lo anterior permite un gran ahorro de tiempo y menor complejidad, pero con una alta seguridad de que se tendrá una precisión adecuada.

Resultados

En las figuras 2 y 3 se presentan los resultados de precisión del modelo de red convolucional en los datos de prueba. Es claro que la precisión del modelo varía dependiendo si se

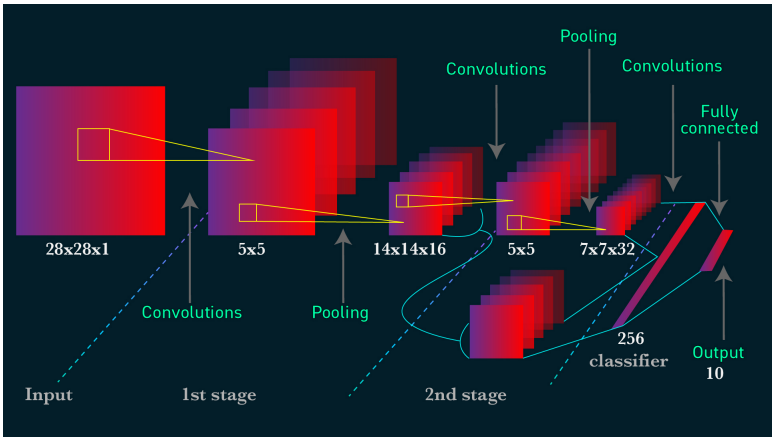

Figura 1: Redes neuronales convolucionales en procesamiento de imágenes [1]

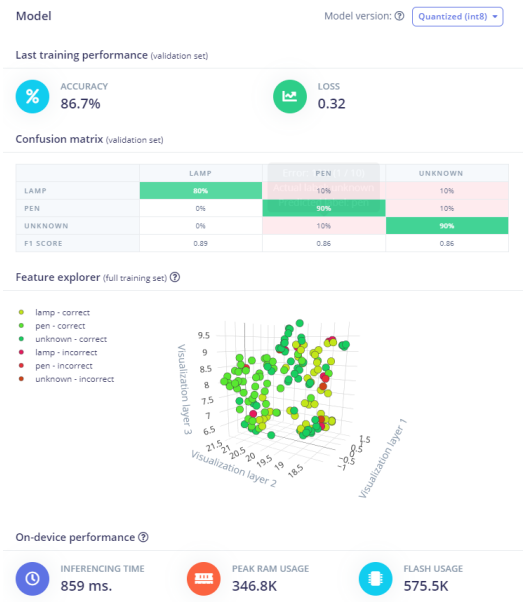

Figura 2: Resultados obtenidos en *EI* (quantized)

elige una versión del modelo tipo *quantized* o *unoptimized*. Sin embargo, se puede notar que en ambos casos la precisión del modelo fue bastante buena, considerando que la cantidad de datos de entrenamiento fue muy baja y que en general las redes neuronales convolucionales necesitan muchos más datos de entrenamiento para obtener una precisión tan buena.

Por otro lado, se ejecutó el entrenamiento del modelo en un cuaderno de Jupyter en nuestra máquina local. Para hacer esto, se usó *Keras*, que permite definir redes convolucionales utilizando un modelo conocido como "secuencial", que permite agregarle capas a la red convolucional según la arquitectura definida.

Para el caso de esta tarea, la arquitectura definida fue la arquitectura *MobileNetV2 96X96 0.35*. Al realizar el entrenamiento de esta red convolucional en nuestra máquina local con esta arquitectura, se siguieron los siguientes pasos:

1. Se agregó una capa fully-connected/Dense con 8 neuronas y función de activación

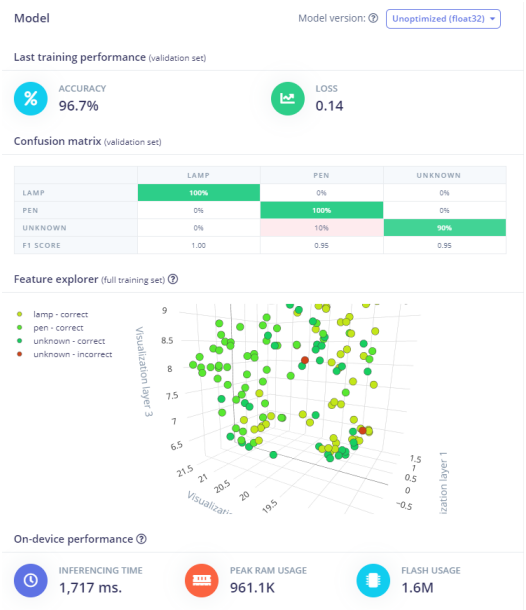

Figura 3: Resultados obtenidos en *EI* (unoptimized)

ReLu.

2. Se agregó una capa de Dropout con probabilidad del 10 %.
3. Se convirtió el resultado matricial a un vector.
4. Se agregó una capa fully-connected/Dense con 3 neuronas (se tenían 3 clases: lámpara, esfero, desconocido) y función de activación Softmax.
5. Se compiló la versión final del modelo con los parámetros
   - Optimizador Adam con tasa de aprendizaje 0.0005
   - Función de pérdida *categorical cross-entropy*
   - Métricas: *accuracy*

En la figura 4 se pueden encontrar los resultados de precisión (accuracy) obtenida por el modelo de red convolucional descrito previamente tanto para los datos de entrenamiento como para los datos de validación/prueba, para diferentes ciclos (epochs). Los *epochs* o ciclos en Machine Learning se pueden definir como el número de veces que los datos de entrenamiento pasan por la red neuronal. Otra forma de verlo es como la cantidad de veces que se actualizan los pesos de la red neuronal; es decir, la cantidad de veces que se ejecutan los algoritmos de *forwardpropagation* y *backpropagation* [2]. Como se puede notar, se evaluó el modelo para un número considerablemente bajo de ciclos e igual se obtuvo una precisión muy buena sobre los datos de entrenamiento y sobre los datos de validación/prueba.

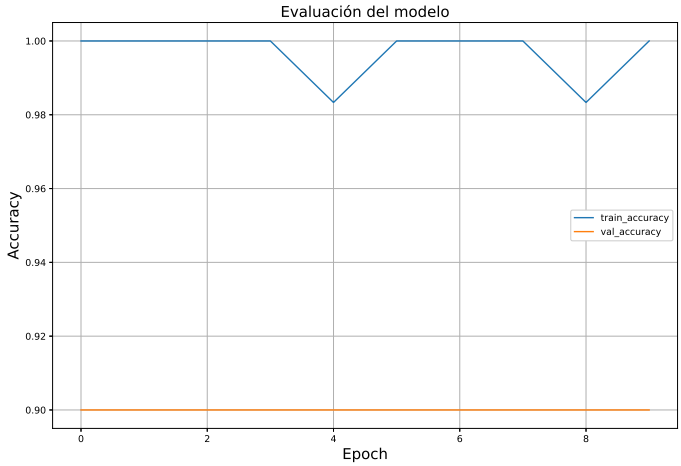

Figura 4: Resultados obtenidos localmente

Análisis de resultados

A partir de los resultados presentados en la sección anterior, se puede afirmar que tanto para el entrenamiento en la nube de *EI* como para el entrenamiento local se obtuvieron resultados muy buenos de precisión del modelo en los datos de prueba. Para el caso del entrenamiento en *EI* y versión de modelo tipo *quantized* se obtuvo un accuracy de 86.7 %, para la versión de modelo tipo *unoptimized* se obtuvo un accuracy de 96.7 %. Mientras tanto, para el entrenamiento local se observó una precisión promedio de 90 % para diferente número máximo de epochs (ciclos) del algoritmo de entrenamiento. Teniendo en cuenta que el entrenamiento de esta red convolucional se hizo con pocos datos de entrenamiento, se puede decir que el resultado de precisión obtenido fue muy bueno, pues en general para obtener buenos resultados de precisión en un modelo de ML es necesario contar con miles o cientos de miles de datos de entrenamiento. Esto demuestra el hecho de que la red neuronal convolucional es muy eficaz especialmente para la tarea de clasificación de imágenes y detección de objetos, gracias al hecho de que la operación de convolución es muy útil y eficaz para realizar extracción de características en imágenes 2D.

Quantized vs. Unoptimized

En primer lugar, se discute la diferencia en los resultados de precisión para la versión del modelo *quantized* y *unoptimized*. La cuantificación (*quantization*) en *deep learning* es el proceso de aproximar una red neuronal que utiliza números de punto flotante mediante una red neuronal de números de ancho de bits bajo. El modelo *unoptimized* utiliza número de punto flotante representados en 32 bits, mientras que el *quantized* usa números de tipo *int* con 8 bits. La cuantificación es importante ya que reduce el costo computacional y el requerimiento de memoria, lo cual es clave en las redes neuronales que se usan en *deep learning*, ya que suelen ser bastante densas [3]. De lo observado en las figuras 2 y 3, se evidencia que la precisión del modelo *unoptimized* es mayor. Lo anterior se puede explicar

ya que en el modelo *quantized* se reduce la precisión de los números utilizados. Esto es una clara desventaja frente al modelo *unoptimized*, sin embargo, la complejidad del modelo es menor, haciendo que sea más fácil trabajar sobre él sin mayores costos computacionales. De igual forma, la precisión sigue siendo bastante alta, por lo cual se tendrá un número bajo de errores en la clasificación de las imágenes. Para saber si utilizar *quantization* en un modelo o no, se debe verificar cuanta precisión se está dispuesto a sacrificar, teniendo en cuenta el propósito del modelo y el problema en el que se aplica.

Nube vs. Local

En general, es claro que los resultados de precisión para el entrenamiento hecho en la nube de *EI* y el entrenamiento hecho localmente fue muy similar. Las pequeñas diferencias se pueden deber a que la partición de los datos entre los conjuntos de entrenamiento y prueba fue diferente en ambos casos. Sin embargo, hacer el entrenamiento localmente o en la nube no tiene ningún efecto sobre la precisión del modelo, pues ambos entrenamientos se realizaron usando el mismo modelo de red neuronal convolucional y los mismos híperparámetros de la *MobileNet*.

La diferencia más grande entre el entrenamiento en la nube y el entrenamiento local es el tiempo que tardó cada proceso. Aunque las gráficas presentadas no demuestran este tiempo de cómputo, se encontró que el entrenamiento de la red convolucional en la nube de *EI* tardó alrededor de unos 4 minutos. Por otro lado, el entrenamiento local en el cuaderno de Jupyter se tomó unos 20 segundos. Esta diferencia tan grande se puede deber al hecho de que los datos no deben viajar hasta un servidor remoto para realizar el entrenamiento y evaluación y regresar al dispositivo donde se está realizando el despliegue.

Para cualquier aplicación de ML va a ser importante tener en cuenta esta diferencia de tiempos de entrenamiento y cómputo: dependiendo de cuál sea el objetivo de la aplicación, se va a necesitar un tiempo de cómputo más rápido o se le puede dar menos importancia al mismo. El ejemplo clásico es el vehículo autónomo. Este va a necesitar un tiempo de cómputo casi instantáneo para sus tareas de detección de obstáculos, por ejemplo, y por eso es mucho más recomendable que sus modelos de ML se realicen localmente que en la nube, pues esto puede evitar accidentes automovilísticos y puede ser la diferencia entre la vida y la muerte. Sin embargo, la aplicación desarrollada en esta tarea (detección de objetos) no tiene un requerimiento de latencia alto y por esta razón un procesamiento en la nube no tiene un efecto grave sobre los resultados obtenidos.

Conclusiones

- Usar convoluciones para extraer características sirve para extraer varias características (features) diferentes de una imagen, por lo que se pueden generar bastantes características para una imagen pequeña, como las de esta tarea, que eran de 96x96 píxeles.
- Las redes neuronales convolucionales encuentran una gran utilidad para el proceso de clasificación de imágenes por el hecho de que convolucionar una imágen 2D con una máscara de diferentes tipos puede ayudar para extraer diferentes características

(features) de una imagen, como las esquinas, las partes redondas, la silueta de los objetos, etc.

- El entrenamiento de una red convolucional se lleva a cabo más rápido en una máquina local que en los servidores de *Edge Impulse* por el hecho de que los datos no deben viajar hasta un servidor remoto para realizar el entrenamiento y evaluación: todo se hace en la misma máquina, así reduciendo el tiempo en que el procesamiento de los datos se lleva a cabo.

- Al usar el modelo *quantized* (con los datos cuantizados) se reduce la precisión de los números utilizados. Esto es una clara desventaja frente al modelo *unoptimized*. Sin embargo, la complejidad del modelo se reduce, haciendo que sea más fácil trabajar sobre él sin mayores costos computacionales.

- Existe cierta cantidad de precisión que se puede ganar o perder dependiendo si se quiere usar un modelo *quantized* o *unoptimized* de los datos (hay un tradeoff de precisión). De igual forma, se observó una precisión alta de clasificación en ambos casos.

- Dependiendo de la aplicación que un usuario quiera darle a su red neuronal convolucional, o a su modelo de *deep learning* en general, se va a necesitar un tiempo de cómputo más rápido o se le puede dar menos importancia al mismo.

Referencias

[1] "Overview of convolutional neural network in image classification," Jan 2018. [Online]. Available: https://analyticsindiamag.com/convolutional-neural-network-image-classification-overview/

[2] V. Rodríguez, "Conceptos básicos sobre redes neuronales," Oct 2018. [Online]. Available: https://vincentblog.xyz/posts/conceptos-basicos-sobre-redes-neuronales

[3] J. Nicholls, "Quantization in deep learning," Aug 2018. [Online]. Available: https://medium.com/@joel_34050/quantization-in-deep-learning-478417eab72b