Reporte de: AlexNet Challenge

Por: Fausto Morales ffmogbaj@gmail.com

- Descripción del trabajo realizado

En este trabajó se completó el challange de la presentación Image Processing in PyTorch pp 81.

Para poder completar las actividades se definió código reutilizable para todos los modelos los cuales consistían en las siguientes actividades:

- Importar librerías
- Validar uso de GPU
- Obtención de datos y separación de conjunto de entrenamiento y prueba
- Definición de las clases del entrenamiento
- Definición de tamaño de batches a usar y épocas
    - Se decidió usar 64 batches para no saturar el equipo.
    - Así como 30 épocas para intentar tener una mejor precisión con los modelos desde Scratch y 10 épocas gastando aproximadamente 5 minutos por época, dando un total aproximado de 2h 30 min para los procesos desde scratch y de 50 min para los procesos pre entrenados.
        - Fue necesario dejar la máquina corriendo en la noche.
- Limpieza de caché

Las cuales se describieron en la 1ª celda del Notebook para pytorch y para tensorflow.

Así mismo, existen tareas relacionadas con el proceso de la ejecución del modelo muy específicas cómo lo son:

- Definir optimizador y criterio de pérdida.
- Entrenar con datos de entrenamiento.
- Guardar el modelo como opcional.
- Evalar la precisión del modelo con datos de prueba por época, final por clase y final global.

Los cuales fueron desarrollados en la función: train_model_tf y train_model_torch para tensorflow y pytorch, respectivamente.

Una vez teniendo estos pasos se desarrollaron las funciones que definen la arquitectura del modelo, basándome en los ejemplos vistos en clase de “LeNet from scratch” para transformarlo a AlexNet, cómo en la diapositiva número 34 de la presentación de referencia:

Sin embargo, para poderlo adaptar al dataset de CIFAR-10 al modelo de AlexNet o VGG16 fue necesario realizar modificaciones:

- Incluir una capa densa para poder recuperar de las 1000 categorías a sólo 10 categorías.
- Hacer un resize ya sea como layer o cómo parte del modelo para cambiar del input de: 224, 224

Para poder lograr las actividades se realizó la definición de arquitectura de 4 modelos distintos:

- AlexNet en TensorFlow desde cero a través de la función: AlexNet_tf
- AlexNet en Pytorch desde cero a través de la clase: AlexNet_torch
- Alexnet pre-entrenado en Pytorch desde el modelo: model_AlexNet_pretrained
- VGG16 pre-entrenado en TensorFlow desde la función: vgg16_tf_pretrain

- Resultados obtenidos

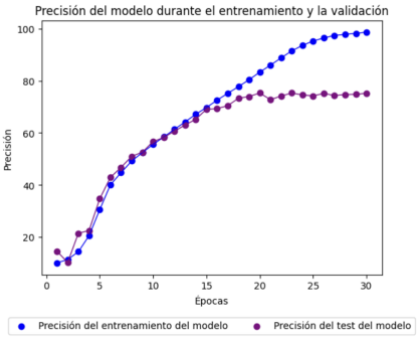
Gracias a la abstracción del problema y segmentación de las partes se optimizó el código para usarse una única vez a lo largo del notebook y se implementaron librerías para poder tener trazabilidad del % de avance del proceso para pytorch.

Se lograron desarrollar funciones que pueden llegar a guardar los modelos entrenados según las necesidades y que a su vez realizan la evaluación del desempeño evaluando la precisión de estos por época, por clase y total despúes del entrenamiento.

Se obtuvo el entrenamiento de los 4 modelos, así como su precisión, estos se detallan a continuación:

AlexNet desde Pytorch

- Parametros entrenables: 62,388,354
- Épocas empleadas: 30
- Tiempo empleado de computador: 11,577.01 seg
- Diagrama de precisión por época:



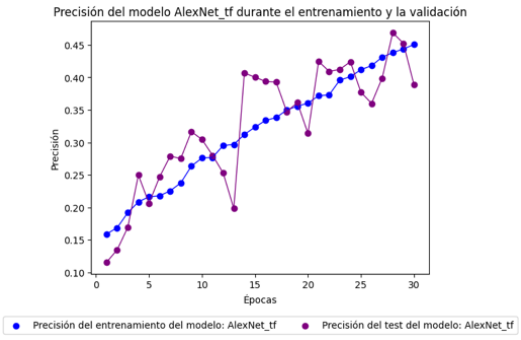
- Precisión de la época final 75.23%, De la cual:

Precisión para la clase 0, 82.10%
Precisión para la clase 1, 90.90%
Precisión para la clase 2, 62.70%
Precisión para la clase 3, 58.10%
Precisión para la clase 4, 75.80%
Precisión para la clase 5, 64.70%
Precisión para la clase 6, 75.90%
Precisión para la clase 7, 79.40%
Precisión para la clase 8, 86.10%
Precisión para la clase 9, 76.60%

- Mejor precisión en época 23 por: 75.3%

AlexNet desde TensorFlow

- Parámetros entrenables: 21,624,906
- Épocas empleadas: 30
- Tiempo empleado de computador: 1,648.81 seg
- Diagrama de precisión por época:



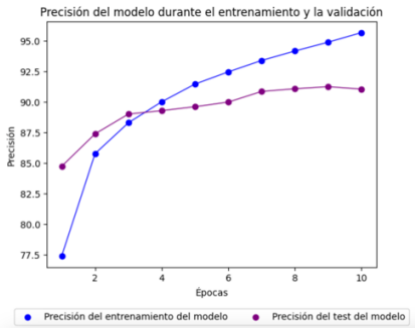
- Precisión de la época final 38.86%%, De la cual:

Precisión para la clase 0: 45.20%,
Precisión para la clase 1: 20.50%
Precisión para la clase 2: 24.60%
Precisión para la clase 3: 0.00%
Precisión para la clase 4: 66.00%
Precisión para la clase 5: 86.90%
Precisión para la clase 6: 35.10%
Precisión para la clase 7: 20.10%
Precisión para la clase 8: 41.90%
Precisión para la clase 9: 48.30%

- Mejor precisión en época 28 por: 46.88%

AlexNet pre entrenado desde Pytorch

- Parámetros entrenables: 57,044,810
- Épocas empleadas: 10
- Tiempo empleado de computador: 2652.99 seg.
- Diagrama de precisión por época:



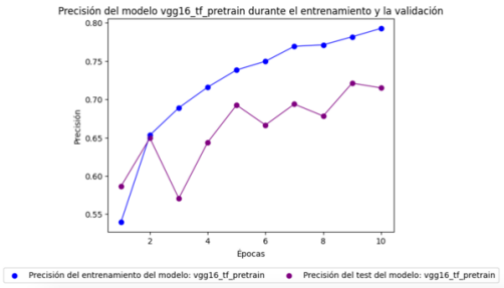
- Precisión de la época final 91%, De la cual:

Precisión para la clase 0, 88.60%,
Precisión para la clase 1, 96.30%,
Precisión para la clase 2, 91.20%,
Precisión para la clase 3, 72.30%,
Precisión para la clase 4, 92.50%,
Precisión para la clase 5, 89.40%,
Precisión para la clase 6, 95.30%,
Precisión para la clase 7, 94.80%,
Precisión para la clase 8, 95.60%,
Precisión para la clase 9, 94.40%,

- Mejor precisión en época 9 por: 95.7%

VGG16 preentrenado en TensorFlow

- Parámetros entrenables: 12,850,698
- Épocas empleadas: 10
- Tiempo empleado de computador: 14541.24 seg
- Diagrama de precisión por época:



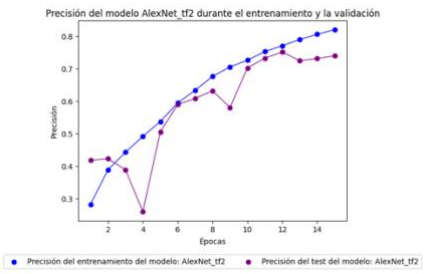
- Precisión de la época final 71.5%, De la cual:

Precisión para la clase 0: 68.20%,
Precisión para la clase 1: 81.10%,
Precisión para la clase 2: 56.90%,
Precisión para la clase 3: 62.90%,
Precisión para la clase 4: 78.60%,
Precisión para la clase 5: 23.70%,
Precisión para la clase 6: 88.50%,
Precisión para la clase 7: 81.30%,
Precisión para la clase 8: 91.80%,
Precisión para la clase 9: 82.00%

- Mejor precisión en época 9 por: 72.12%

- Ejercicio complementario:

Se probó realizar mejoras variando la arquitectura de AlexNet desde TensorFlow



Donde se logró subir al 74.01% de precisión en 15 épocas al ajustar las capas densas más pequeñas.

- Conclusiones.

Gracias a este ejercicio pude:

- Sintetizar los pasos del proceso de modelado de categorías a partir de imágenes, generando un código reutilizable a lo largo del proyecto.
- Desarrollar un proceso de entrenamiento estándar para el problema en pytorch y tensorflow.
- Tener experiencia en la implementación de modelos desde 0 y pre entrenados.
- Realizar ajustes a la arquitectura del modelo cuando no se convergía o buscar estrategias para optimizar la precisión.
- Tratar de implementar algoritmos que permitan el entorno de MAC y CUDA.
- Darme una idea del largo tiempo de ejecución que requieren los modelos y el tipo de infraestructura necesaria para realizarlos.

Así como concluir que:

- Los modelos pre entrenados de un problema ya trabajado permiten optimizar tiempos y lograr modelos más eficientes más rápidamente (Similar a lo sucedido con los modelos de DeepSeek)
- En mi caso, el módelo que tuvo un mejor rendimiento fue: AlexNet pre entrenado desde Pytorch con una precisión de: 91% (época 10) o 95.7% (época 9) y un tiempo de ejecución de: 2652.99 segundos.