AlexNet en PyTorch y TensorFlow

Joseph Tuffit Hadad Piña

7 de marzo de 2025

1. Introducción

En el presente reporte presentamos los resultados de crear AlexNet desde cero y compararla con versiones pre-entrenadas ya existentes en PyTorch y TensorFlow.

AlexNet es una arquitectura de redes neuronales convolucionales diseñada por Alex Krizhevsky en colaboracion con Ilya Sutskever y Geoffrey Hinton. La red tiene un total de 60 millones de parametros y 650,000 neuronas.

1.1. Arquitectura

AlexNet contiene 8 capas: Las primeras 5 capas son convolucionales, algunas seguidas por max pooling y las ultimas 3 son fully conected. Tenemos lo siguiente

$$(CNN \to RN \to MP)^2 \to (CNN^3 \to MP) \to (FC \to DO)^2 \to Linear \to Softmax$$

2. Metodología

Realizamos 4 pruebas de AlexNet

1. **PyTorch desde cero**: Creamos la arquitectura de AlexNet desde cero usando la libreria Pytorch.
2. **PyTorch pre-entrenado**: Usamos el modelo pre entrenado de Pytorch para AlexNet
3. **TensorFlow desde cero**: AlexNet desde cero usando TensorFlow Keras.
4. **TensorFlow pre-entrenado**: Modelo VGG16 pre-entrenado en ImageNet como sustituto, Para AlexNet.

Es importante realizar la aclaracion que no existe un modelo pre entrenado de AlexNet en TensorFlow/Keras. Para motivos de comparaciones hemos usado el pre entrenado de VGG 16/

Los modelos fueron entrenados bajo las siguientes condiciones:

- **Conjunto de datos**: CIFAR-10
- **Optimizador**: SGD con momentum 0.9 y weight decay 5e-4
- **Tasa de aprendizaje**: 0.01 con programación de disminución cada 5 épocas
- **Función de pérdida**: Cross Entropy
- **Aumento de datos**: Data Augmentation para evitar sobre ajuste.

3. Implementaciones

3.1. AlexNet desde cero en PyTorch

El modelo es de:

- 5 Capas convolucionales.
- 3 Capas de MaxPooling para reducción de dimensión
- 3 Capas densas.

Hemos agregado una capa dropout para evitar el sobre ajuste. Tambien hemos agregado cientras transforms a las imagenes para generar un data augmentations como es cortar imagne y espejo. Esto para evitar un sobre ajuste que estabamos presetnado.

3.2. AlexNet desde cero en TensorFlow

La implementación sigue una estructura similar a la versión de PyTorch,de Keras con:

- Capas Conv2D con activación ReLU
- Capas MaxPooling2D
- Capas Dense con dropout para reducir el sobreajuste

En esta version nuevamente realizamos data augmentation para evitar un sobre ajuste.

3.3. Versiones pre entrenadas

- **PyTorch**: Se utilizó el modelo AlexNet pre-entrenado de torchvision, ajustando el clasificador para CIFAR-10.
- **TensorFlow**: Se utilizó VGG16 pre entrenado. Ya que TensorFlow/Keras no presentar el pre entrenado de AlexNet.

4. Resultados

Realizamos 2 pruebas distintas. La primera es correr los dos modelos from scratch por 50 epocas que es un numero elevado de epocas para ver que accuracy podiamos alcanzar en ambos. La segunda prueba fue correr los modelos pre entrenados y from scratch a 10 epocas y compararlos. El motivo de esta division es que entrenar el AlexNet from scratch a 50 epocas tomaba demasiado tiempo que no teniamos los recursos para dejarlo tanto tiempo. Entonces Para poder realizar una comparativa justa se dejo a entrenar los 4 modelos por 10 epocas.

4.1. Entrenamiento a 50 épocas (modelos from scratch)

Los modelos desde cero fueron entrenados durante 50 épocas:
PyTorch desde cero:

- Precisión final en prueba: 89.16 %
- La pérdida disminuyó de 2.140 a 0.091

TensorFlow desde cero:

- Precisión final en prueba: 87.46 %
- La pérdida disminuyó de 1.964 a 0.088

4.2. Entrenamiento a 10 épocas

La siguiente

Modelo	Precisión Final (%)	Tiempo (s)	Pérdida Final
PyTorch from scratch	71.13	246.15	0.822
PyTorch pre entrenado	90.40	1,261.76	0.153
TensorFlow from scratch	81.18	271.87	0.568
TensorFlow pre entrenado	64.15	272.04	1.083

Cuadro 1: Comparación de rendimiento de los modelos en un entrenamiento a 10 épocas

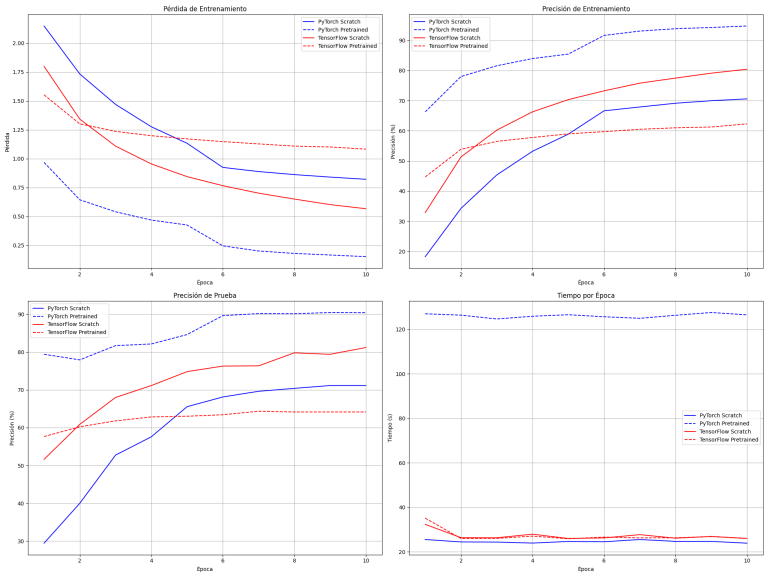

Figura 1: Comparación de precisiones y tiempos para los distintos modelos.

5. Análisis de Resultados

1. **Precisión**:
   - El modelo pre entrenado de PyTorch obtuvo la mejor precisión 90.40 %, seguido por TensorFlow desde cero 81.18 %.
   - El modelo pre entrenado de TensorFlow (VGG16) obtuvo el peor rendimiento, posiblemente debido a problemas de adaptación al tamaño de imagen pequeño. Y que obviamente se trata de un VGG 16 y no un AlexNet
   - El modelo de TensorFlow a 50 epocas tuvo una mayor precision que el Pytorch a 50 epocas.
2. **Tiempo de entrenamiento**:
   - Los modelos de PyTorch y TensorFlow from scratch tuvieron tiempos de entrenamiento similares y no muy altos a pesar del alto numero de epocas.
   - El modelo pre entrenado de PyTorch fue significativamente más lento, posiblemente debido al mayor tamaño de la red.
3. **Velocidad de convergencia**:
   - El modelo pre entrenado de PyTorch alcanzo una muy alta precision desde epocas muy tempranas.

- TensorFlow from scratch mostró una convergencia a una precision alta más rápida que en PyTorch.

6. Conclusiones

En el paper actual comparamos los modelos de AlexNet from scratch a los pre entrenados existentes tanto en TensorFlow como en Pytorch. De las lecciones aprendidas podemos sobresaltar la importancia del data augmentation en los modelos from scratch y el agregar capas dropout. Realizar esta implementación nos permitió alcanzar una muy alta precision en un tiempo de ejeccucion considerable. Antes de realizar estas mejoras teniamos una precision que se aproximaba mejor al 50. EL modelo pre entrenado de AlexNet de Pytorch tuvo un desempeño increible, su tiempo de ejeccucion para 10 epocas fue mayor que el resto pero su precisión fue la mejor de las 4. Lamentablemente TensorFlow no cuenta con un AlexNet pre entrenado para poder realizar una justa comparación. Usamos VGG 16 pero solo fue para fines de llenar el vació, el modelo performeo de peor manera que los 4.