Reporte: Comparación de Modelos AlexNet desde Cero y Preentrenados utilizando CIFAR-10

Nombre: Bustos Benítez Sonia Valeria
Fecha de Entrega: 03/07/2025

---

1. Introducción

El objetivo de este trabajo es implementar una versión personalizada de AlexNet desde cero y comparar su desempeño con modelos preentrenados adaptados para el conjunto de datos CIFAR-10. Esto se llevó a cabo utilizando PyTorch y TensorFlow/Keras. La versión creada cuenta con 11 capas: 5 convolucionales, 3 de max pooling y 3 densas. A partir de los resultados, se evalúan las ventajas y limitaciones del aprendizaje desde cero frente al aprendizaje por transferencia.

---

2. Implementación

2.1. Conjunto de Datos: CIFAR-10

CIFAR-10 es un conjunto de datos compuesto por imágenes de 32x32 en 10 clases. El preprocesamiento incluyó:

- Normalización.
- Aumentación de datos (rotaciones, desplazamientos horizontales/verticales, volteo horizontal).

2.2. Arquitectura AlexNet

Ambos modelos creados desde cero compartieron la misma arquitectura:

- **Capas Convolucionales (5):** Extraen características visuales.
- **Capas de Max Pooling (3):** Reducen las dimensiones espaciales.
- **Capas Densas (3):** Con 4096 unidades en las dos primeras y 10 en la salida (una por clase).

2.3. Implementación de Modelos

Desde Cero:

- **PyTorch:** Se utilizó `torch.nn.Module` para construir la red desde cero. El entrenamiento empleó Adam como optimizador y entropía cruzada categórica como pérdida.
- **TensorFlow/Keras:** Se usó `tf.keras.Sequential` para construir el modelo desde cero. El flujo también incluyó aumentación de datos y callbacks para dinamizar el aprendizaje.

Preentrenados:

- **PyTorch:** AlexNet preentrenado con ImageNet.
- **TensorFlow/Keras:** VGG16 como alternativa a AlexNet, adaptado para 10 clases de CIFAR-10.

---

3. Resultados

3.1. Modelos desde Cero

PyTorch:

- **Tiempo por Época:** 5:33 minutos.
- **Precisión en el Conjunto de Prueba:** 70.93%.
- **Resultados del Entrenamiento:**

```
Epoch 10, Loss: 0.6899, Accuracy: 76.12%
Test Accuracy: 70.93%
```

TensorFlow/Keras:

- **Tiempo por Época:** ~7 minutos.
- **Precisión en el Conjunto de Prueba:** 73.02%.
- **Resultados del Entrenamiento:**

```
Epoch 10, Loss: 0.8467, Accuracy: 70.63%
Test Accuracy: 73.02%
```

3.2. Modelos Preentrenados

PyTorch:

- **Tiempo por Época:** 18 minutos.
- **Precisión en el Conjunto de Prueba:** 10.00%.
- **Resultados del Entrenamiento:**

```
Epoch 10, Loss: 2.3031, Accuracy: 9.84%
Test Accuracy: 10.00%
```

TensorFlow/Keras:

- **Tiempo por Época:** ~53 minutos.
- **Precisión en el Conjunto de Prueba:** 81.72%.
- **Resultados del Entrenamiento:**

```
Epoch 10, Loss: 0.8739, Accuracy: 72.88%
Test Accuracy: 81.72%
```

4. Análisis y Discusión

1. **Modelos desde Cero:**
    - El modelo desarrollado desde cero en PyTorch mostró un desempeño aceptable (70.93%), aunque fue ligeramente superado por la versión en TensorFlow/Keras (73.02%). Este comportamiento puede deberse a diferencias en la optimización y manejo de regularización.
2. **Modelos Preentrenados:**
    - La versión preentrenada en PyTorch no logró converger (10.00%). Esto sugiere un problema de configuración, como tasas de aprendizaje inadecuadas o mal manejo de los pesos congelados.
    - La versión preentrenada en TensorFlow/Keras alcanzó una precisión significativamente mayor (81.72%), evidenciando las ventajas del aprendizaje por transferencia en tareas de clasificación.
3. **Impacto del Conjunto de Datos:**
    - CIFAR-10, por ser un conjunto relativamente pequeño, limita el desempeño de los modelos desde cero en comparación con los preentrenados que aprovechan características extraídas de conjuntos masivos como ImageNet.
4. **Eficiencia de Entrenamiento:**
    - Los modelos desde cero entrenaron más rápido (5-7 minutos por época) que los preentrenados en TensorFlow/Keras (53 minutos por época).
    - Sin embargo, la precisión de los preentrenados valida su costo computacional en tareas de alta importancia.

---

5. Conclusiones

1. **Modelos Desde Cero:** Ofrecen una buena base para entender la arquitectura y entrenar en conjuntos pequeños, pero su desempeño no compite con los preentrenados.
2. **Modelos Preentrenados:** En TensorFlow/Keras, estos modelos lograron un aprendizaje más rápido y mejores resultados (81.72%). Sin embargo, el modelo en PyTorch requiere ajustes para converger.
3. **Recomendación:** Para tareas prácticas, priorizar modelos preentrenados con fine-tuning, ya que maximizan precisión y generalización.