AlexNet challenge

Alumnos: Mar Bazúa y Néstor Medina

Decidimos realizar el proyecto utilizando Google Colab por practicidad, eficiencia y temas de compatibilidad con los chips Apple Silicon que necesitan una configuración especial en TensorFlow (WIP),

Para realizar el “challenge” se realizó lo siguiente:

- Recrear AlexNet desde ceros utilizando pytorch y tensorflow/keras. La red debe tener 5 capas convolucionales, 3 capas con max pooling y 3 capas densas. Dando un total de 11 capas.
- Usar la base CIFAR-10 para comparar el rendimiento entre modelos realizados desde ceros vs pre-entrenados con TensorFlow y PyTorch.

El Código completo se encuentra en dos cuadernos separados por framework *AlexNetChallengeFinal-pytorch.ipynb y AlexNetChallengeFinal-tensorflow.ipynb*

Para utilizar las imágenes de CIFAR-10 es importante descargarlos y realizar algunos ajustes para que puedan ser entendidos por los modelos configurados.

1.- Recrear AlexNet desde cero.

Para recrear el modelo desde cero, es necesario definir cada una de las capas, en la cual se puede jugar con algunas variantes (como el uso de padding en la convolución).

1.1 AlexNet desde cero, utilizando PyTorch
La estructura de la red es:

```
    self.features = nn.Sequential(
          nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),  # 1
          nn.ReLU(inplace=True),
          nn.MaxPool2d(kernel_size=3, stride=2),  # 2

         nn.Conv2d(64, 192, kernel_size=5, padding=2),  # 3
          nn.ReLU(inplace=True),
          nn.MaxPool2d(kernel_size=3, stride=2),  # 4

          nn.Conv2d(192, 384, kernel_size=3, padding=1),  # 5
          nn.ReLU(inplace=True),

           nn.Conv2d(384, 256, kernel_size=3, padding=1),  # 6
          nn.ReLU(inplace=True),

          nn.Conv2d(256, 256, kernel_size=3, padding=1),  # 7
          nn.ReLU(inplace=True),
```

AlexNet challenge

```
            nn.MaxPool2d(kernel_size=3, stride=2)   # 8
      )

      self.classifier = nn.Sequential(
            nn.Linear(256 * 6 * 6, 4096),  # 9
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),  # 10
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, num_classes)  # 11
      )
```

Utilizamos 10 épocas y obtuvimos un accuracy de 76.08% para los datos de entrenamiento (train) y de 61.59% para los datos de prueba (test). El desempeño del modelo es bajo.

1.2 AlexNet desde cero, utilizando TensorFlow
La estructura de la red es:

```
       Conv2D(96, kernel_size=11, strides=4, activation='relu',
input_shape=(227, 227, 3)),
       MaxPooling2D(pool_size=3, strides=2),

       Conv2D(256, kernel_size=5, padding='same', activation='relu'),
       MaxPooling2D(pool_size=3, strides=2),

       Conv2D(384, kernel_size=3, padding='same', activation='relu'),
       Conv2D(384, kernel_size=3, padding='same', activation='relu'),
       Conv2D(256, kernel_size=3, padding='same', activation='relu'),
       MaxPooling2D(pool_size=3, strides=2),

       Flatten(),
       Dense(4096, activation='relu'),
       Dropout(0.5),
       Dense(4096, activation='relu'),
       Dropout(0.5),
       Dense(10, activation='softmax')
```

Utilizamos 10 épocas y obtuvimos un accuracy de 96.32% para train y 80.34% para test. Mucho mejor que con PyTorch.

AlexNet challenge

2.- Utilizando los modelos preentrenados

Pasando a la segunda parte, se ajustaron los modelos pre entrenados tanto con PyTorch como con TensorFlow. En el caso de TensorFlow no existe un modelo AlexNet pre entrenado por lo que utilizamos ResNet50 como base.

2.1.- AlexNet pre-entrenado utilizando PyTorch
El modelo que utilizamos fue

```
# 2. Descargar AlexNet preentrenado
model_pretrained = models.alexnet(pretrained=True)
model_pretrained.to(device)
```

Con el modelo pre entrenado obtuvimos un accuracy del 93%.

2.2.- AlexNet pre-entrenado utilizando TensorFlow (utilizando ResNet50)

```
# Cargar ResNet50 pre-entrenado
base_model = ResNet50(weights='imagenet', include_top=False,
input_tensor=input_tensor)

# Descongelar solo las últimas 15 capas para reducir uso de memoria
for layer in base_model.layers[:-15]:
   layer.trainable = False
for layer in base_model.layers[-15:]:
   layer.trainable = True

# Añadir capas personalizadas
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu', dtype='float32')(x)  # Forzar capa
en float32
x = Dropout(0.5)(x)
x = Dense(512, activation='relu', dtype='float32')(x)
x = Dropout(0.3)(x)
output_layer = Dense(10, activation='softmax', dtype='float32')(x)

# Crear modelo
model = Model(inputs=base_model.input, outputs=output_layer)
#model = Model(inputs=[base_model.input], outputs=output_layer)
```

Con el modelo pre entrenado obtuvimos un accuracy del 67%.

AlexNet challenge

3.- Comparación de los modelos y conclusiones

La siguiente tabla presenta el accuracy de test de los modelos ajustados en las secciones 1 y 2.

Modelo AlexNet	Accuracy
Desde cero con PyTorch	61.59%
Desde cero con TensorFlow	80.34%
Preentrenado con PyTorch	93%
Preentrenado con TensorFlow (ResNet50)	67%

Claramente los resultados obtenidos en nuestras pruebas muestran diferencias significativas en el rendimiento de los modelos implementados:

Conclusiones

1. Diferencias entre frameworks: Se observa una notable diferencia en el rendimiento entre las implementaciones desde cero con PyTorch (61.59%) y TensorFlow (80.34%). Esto sugiere que las implementaciones predeterminadas, optimizaciones internas y configuraciones de cada framework pueden tener un impacto sustancial en el rendimiento final, incluso cuando la arquitectura de la red es similar.

2. Ventaja de modelos preentrenados: El modelo preentrenado de AlexNet en PyTorch alcanzó un accuracy significativamente mayor (93%) que su contraparte implementada desde cero (61.59%), demostrando el valor de los pesos preentrenados en conjuntos de datos más grandes como ImageNet.

3. Diferencias arquitectónicas: Es importante señalar que para TensorFlow, al no existir un modelo AlexNet preentrenado oficial, se utilizó ResNet50, una arquitectura más moderna pero diferente. A pesar de ser teóricamente más avanzada, obtuvo un rendimiento inferior (67%) al AlexNet implementado desde cero en el mismo framework (80.34%), lo que podría indicar que no todas las arquitecturas más complejas son necesariamente mejores para todos los conjuntos de datos.