Challenge AlexNet en PyTorch y TF/Keras

Carlos Oswaldo Rodriguez Salas

Introducción

En este reporte expongo el proceso de implementacion del modelo AlexNet partiendo desde cero utilizando dos differentes paqueterias de DeepLearning: PyTorch y TF/Keras. La implementación incluye el procesamiento de los datos, la definición de la arquitectura del modelo, la configuración del pipeline de datos y la evaluación del rendimiento del modelo.

1.Visión General de AlexNet

AlexNet es una red neuronal convolucional introducida en 2012 capaz de clasificar imagenes en una gran cantidad de categorias. El modelo esta formado por varias capas convolucionales seguidas de varias capas totalmente conectadas e incorpora herramientas como MaxPooling y Dropout para aumentar su rendimiento.

El modelo original fue diseñado para aprender de imagenes de 227x227 pixeles, por lo que es necesario cambiar la primera y ultima capa para que se adapte a los datos que utilizaremos.

2.Implementación en PyTorch

La estructura en PyTorch luce de la siguiente manera:

```python
class AlexNet(nn.Module):
    def __init__(self):
        super(AlexNet, self).__init__()
        self.cn1 = nn.Conv2d(3, 96, 11, stride=4,padding=0)
        self.cn2 = nn.Conv2d(96, 256, 5, stride=1,padding=2)
        self.cn3 = nn.Conv2d(256, 384, 3, stride=1,padding=1)
        self.cn4 = nn.Conv2d(384, 384, 3, stride=1,padding=1)
        self.cn5 = nn.Conv2d(384, 256, 3, stride=1,padding=1)
        self.fc1 = nn.Linear(9216, 4096)
        self.fc2 = nn.Linear(4096, 4096)
        self.fc3 = nn.Linear(4096, 10)

    def forward(self, x):
        x = F.relu(self.cn1(x),inplace=True)
        x = F.max_pool2d(x, kernel_size=3, stride=2)
        x = F.relu(self.cn2(x),inplace=True)
        x = F.max_pool2d(x, kernel_size=3, stride=2)
        x = F.relu(self.cn3(x),inplace=True)
        x = F.relu(self.cn4(x),inplace=True)
        x = F.relu(self.cn5(x),inplace=True)
        x = F.max_pool2d(x, kernel_size=3, stride=2)
        x = x.view(-1, 9216)
        x = F.relu(F.dropout(self.fc1(x),training=self.training,inplace=False),inplace=True)
        x = F.relu(F.dropout(self.fc2(x),training=self.training,inplace=False),inplace=True)
        x = self.fc3(x)
        return x
```

Se observan las cinco capas convolucionales, las tres capas totalmente conectadas y las 3 capas de MaxPooling. Para entrenar los 58.3 millones de parametros resultantes se utilizó el dataset CIFAR-10 con 50,000 imagenes para entrenamiento y 10,000 para la evaluación, las cuales se dividen en 10 categorias.

Se eligio Entropia Cruzada como funcion de perdida y SDG sirvió como optimizador. Se hicieron 10 epocas de entrenamiento con batches de 4 y se obtuvieron los siguientes resultados:

```
Accuracy of plane : 78.00 %
Accuracy of car   : 84.50 %
Accuracy of bird  : 72.60 %
Accuracy of cat   : 69.10 %
Accuracy of deer  : 76.70 %
Accuracy of dog   : 67.90 %
Accuracy of frog  : 84.10 %
Accuracy of horse : 82.50 %
Accuracy of ship  : 87.10 %
Accuracy of truck : 89.10 %
```

El accuracy general fue de 79.16%, lo cual es bastante bueno considerando el número de clases.

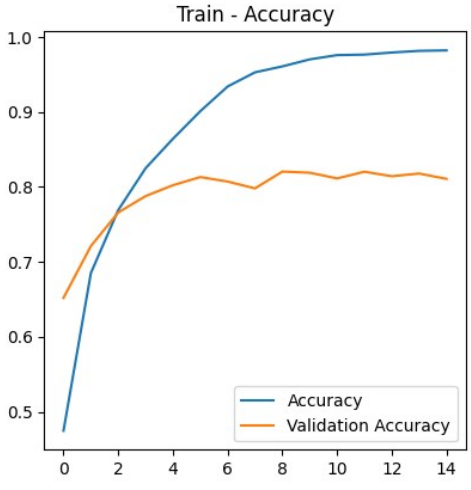
3.Implementacion en TF/Keras

La estructura general utilizada en TensorFlow es la misma, cambiando unicamente la manera de definir la red y algunos otros detalles.

```
model= tf.keras.Sequential([layers.Resizing(227,227,data_format='channels_last',input_shape=(32,32,3)),
                            layers.Conv2D(filters=96, kernel_size=11, strides=4,activation='relu',input_shape=(227,227,3)),
                            layers.MaxPooling2D(pool_size=3, strides=2),

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

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

                            layers.Flatten(),
                            layers.Dense(4096,activation='relu'),
                            layers.Dropout(0.5),

                            layers.Dense(4096,activation='relu'),
                            layers.Dropout(0.5),

                            layers.Dense(10,activation='softmax')
                            ])
```

Para el entrenamiento en TensorFlow decidí utilizar el optimizador Adam, se realizó en 15 epocas (no era necesario que fueran tantas) y en batches de 64.

El modelo obtuvo un accuracy de 81%, los resultados se observan mejor en las siguiente gráfica:



Se observa que el perforance se mantuvo constante a partir de la epoca 5, por lo que no es necesario seguir entrenando al modelo despues de esto.

4.Modelo Preentrenado en PyTorch
Por último, se cargo el modelo AlexNet preentrenado disponible en PyTorch y se comparó su desempeño con nuestros modelos.

Para esto se congelaron los pesos correspondientes a las capas convolucionales y se hizo un fine tunning de la parte clasificatoria de la red (las capas conectadas totalmente). Esto es necesario ya que el modelo original fue entrenado con un dataset de 10,000 categorias, por lo que tenemos que entrenarlo para la clasificacion de nuestras categorias de interés sin perder la capacidad adquirida de identificar caracteristicas en las imagenes.

Los pesos se congelaron con el siguiente codigo:

```
for param in model.features.parameters():
    param.requires_grad = False
```

Para entrenar la parte clasificatoria se tiene que configurar al optimizador de la siguiente manera:

```
optimizer = optim.SGD(model.classifier.parameters(), lr=0.001, momentum=0.9)
```

Los resultados obtenidos son los siguientes:

```
Accuracy of plane : 82.80 %
Accuracy of car   : 90.60 %
Accuracy of bird  : 64.40 %
Accuracy of cat   : 65.00 %
Accuracy of deer  : 80.90 %
Accuracy of dog   : 68.40 %
Accuracy of frog  : 94.60 %
Accuracy of horse : 90.00 %
Accuracy of ship  : 93.10 %
Accuracy of truck : 81.70 %
```

El modelo tuvo un accuracy general de 81.15%. Lo cual no presenta una mejora muy significativa respecto a los otros modelos.