Tarea 5 EL7008 – Primavera 2020
Clasificación de objetos usando CNNs

Profesor: Javier Ruiz del Solar
Auxiliar: Patricio Loncomilla
Ayudantes: Juan Pablo Cáceres, Hans Starke, Javier Smith, José Villagrán

Fecha enunciado: 1 de Diciembre de 2020
Fecha entrega: 14 de Diciembre de 2020

El objetivo de esta tarea es implementar un sistema de clasificación de objetos usando redes neuronales convolucionales (CNNs). En esta tarea se debe usar la librería *pytorch* para poder generar los tensores que corresponden a las imágenes y sus labels (etiquetas), además de implementar arquitecturas de red y códigos para entrenamiento y evaluación.

Preparación de Conjuntos de Entrenamiento y Test

Se usará el dataset CIFAR10, el cual contiene 10 categorías de objetos, cada uno de los cuales corresponde a una imagen RGB de 32x32 píxeles. El dataset se puede bajar desde el siguiente sitio web:

https://www.cs.toronto.edu/~kriz/cifar.html

En caso de que quiera usar un notebook de colaboratory, se recomienda bajar el archivo cifar-10-python.tar.gz usando el comando !wget, y luego descomprimirlo dentro de colaboratory.

El archivo .tar.gz, al ser descomprimido, genera los siguientes archivos:

```
batches.meta  data_batch_2  data_batch_4  readme.html
data_batch_1  data_batch_3  data_batch_5  test_batch
```

La forma de poder leer e interpretar el contenido de esos archivos se indica en el link de la página del cifar-10.

Implementación del dataset para pytorch

Para que pytorch pueda acceder a las imágenes, se debe implementar una clase `CIFAR10Train` que permita ese acceso:

```
from torch.utils.data import Dataset
class CIFAR10Train(Dataset):
    def __init__(self, path):
      # Constructor, debe leer el archivo data_batch_1 dentro de la
carpeta
      # indicada (este archivo se usará para el set de entrenamiento)
    def __len__(self):
      # Debe retornar el número de imágenes en el dataset de
entrenamiento
    def __getitem__(self, index):
      # Debe retornar un par label, image
      # Donde label es una etiqueta, e image es un arreglo de 3x32x32
```

```
        # index es un número (o lista de números) que indica cuáles
imágenes
        # y labels se deben retornar
```

Del mismo modo, se deben crear las clases `CIFAR10Val` y `CIFAR10Test`, las cuales deben leer los archivos `data_batch_2` y `test_batch` respectivamente. Dado que las imágenes están en el rango [0,255], las clases que implementan los datasets deben hacer un escalamiento lineal de esos valores para que queden en el rango [-1,1].

Creación de las arquitecturas de red

En esta tarea se usarán dos arquitecturas de red (en ambas se debe usar padding = 1). Las arquitecturas son las siguientes:

Arquitectura G (Grande)	Arquitectura P (Pequeña)
Convolución con 64 filtros de 3x3 + ReLU + BN	Convolución con 64 filtros de 3x3 + ReLU + BN
Convolución con 64 filtros de 3x3 + ReLU + BN	Max pooling
Max pooling	Convolución con 128 filtros de 3x3 + ReLU + BN
Convolución con 128 filtros de 3x3 + ReLU + BN	Max pooling
Convolución con 128 filtros de 3x3 + ReLU + BN	Convolución con 256 filtros de 3x3 + ReLU + BN
Max pooling	Max pooling
Convolución con 256 filtros de 3x3 + ReLU + BN	Convolución con 512 filtros de 3x3 + ReLU + BN
Convolución con 256 filtros de 3x3 + ReLU + BN	Max pooling
Max pooling	Fully connected con 128 unidades + ReLU + BN
Convolución con 512 filtros de 3x3 + ReLU + BN	Fully connected con 10 unidades
Convolución con 512 filtros de 3x3 + ReLU + BN	
Max pooling	
Fully connected con 128 unidades + ReLU + BN	
Fully connected con 256 unidades + ReLU + BN	
Fully connected con 512 unidades + ReLU + BN	
Fully connected con 1024 unidades + ReLU + BN	
Fully connected con 10 unidades	

Tabla 1. Arquitecturas de red que deben ser usadas en la tarea

La clase que implementa la red G debe tener la siguiente estructura:

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class MyNetBig(nn.Module):
```

```
    def __init__(self, nclasses):
        super(MyNet, self).__init__()
        self.nclasses = nclasses
        self.conv1 = nn.Conv2d(3, 64, 3, padding = 1)
        self.bn1 = torch.nn.BatchNorm2d(64)
        ...
        self.fc5 = nn.Linear(1024, self.nclasses)
    def forward(self, x):
        x = self.bn1(F.relu(self.conv1(x)))
        ...
        x = self.fc5(x)
        return x
```

Además, se debe implementar una segunda arquitectura `MyNetSmall` para la red P, cuya arquitectura se muestra en la tabla 1.

Entrenamiento de la red

Para entrenar la red, se debe crear un dataloader que use el dataset de entrenamiento, además de instanciar la red, la función de pérdida y el optimizador.

```
from torch.utils.data import DataLoader, random_split

dataset_train = CIFAR10Train('cifar-10-batches-py')

train_loader = DataLoader(dataset_train, batch_size=32, shuffle=True, n
um_w
orkers=4, pin_memory=True)

net = MyNetBig(10)

net.cuda()

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(net.parameters(), lr=1e-3)
```

Se recomienda usar la función de pérdida `nn.CrossEntropyLoss()` y el optmizador `Adam`

Además, se debe crear un código que entrene la red durante 40 épocas. El código puede basarse en:

```
for epoch in range(40):
    for i, data in enumerate(train_loader, 0): # Obtener batch
        labels = data[0].cuda()
        inputs = data[1].float().cuda()
```

```
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
    # Calcular accuracy sobre el conjunto de validación y almacenarlo
    # para hacer un plot después
```

En cada época, se debe ir calculando el *loss* sobre el conjunto de entrenamiento, almacenándolo en una lista para poder graficarlo posteriormente. Además, tras cada época, se debe calcular el *accuracy* sobre el conjunto de validación y almacenarlo también en una lista. Se recomienda usar `with torch.no_grad():` durante el proceso de validación. Dado que hay 10 clases, la predicción de la red tiene 10 canales. Se debe obtener el máximo de esos canales usando `torch.max(outputs.data, 1)`

Evaluación de la red

Para poder evaluar el desempeño de la red, se debe iterar sobre el dataloader de test. A partir de la etiqueta predicha, y la real, es posible ir llenando una matriz de confusión.

Se pide realizar los siguientes pasos y detallarlos en el informe:

1. Implementar el código para que pytorch acceda a los datasets, para el conjunto de entrenamiento, validación y prueba.
2. Implementar la código que define la red G (grande).
3. Entrenar la red G, almacenando los *losses* y *accuracies*.
4. Graficar los *losses* relacionados a la red G en función del tiempo. Graficar los *accuracies* de validación relacionados a la red G en función del tiempo. Calcular el *accuracy* y matriz de confusión normalizada relacionados a la red G sobre el conjunto de test.
5. Implementar el código que define la red pequeña.
6. Entrenar la red P, almacenando los *losses* y *accuracies*.
7. Graficar los *losses* relacionados a la red P en función del tiempo. Graficar los *accuracies* de validación relacionados a la red P en función del tiempo. Calcular el *accuracy* y matriz de confusión normalizada relacionados a la red P sobre el conjunto de test.
8. Analizar y comparar el comportamiento de las redes G y P, tanto en la velocidad de convergencia como en el *accuracy* obtenido en el conjunto de test.
9. Documentar cada uno de los pasos anteriores en el informe.

El código debe ser desarrollado en colaboratory, eligiendo un entorno de ejecución con gpu.
Los informes, los códigos y el archivo README.txt deben ser subidos a U-Cursos hasta las 23:59 horas del día 14 de diciembre.

Importante: La evaluación de esta tarea considerará el correcto funcionamiento del código, la calidad de los experimentos realizados y de su análisis, las conclusiones, así como la prolijidad y calidad del informe entregado.

Nota: El informe de la tarea debe ser subido a turnitin