```python
# -*- coding: utf-8 -*-
"""PyTorch_Net.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1VaRJXCTRYPnSrfmQcq6PVJuMZ5xxB250

<center><h1>Mini Project 3 - Convolutional Neural Network</h1>
<h4>The PyTorch File.</h4></center>

<h3>Team Members:</h3>
<center>
Yi Zhu, 260716006<br>
Fei Peng, 260712440<br>
Yukai Zhang, 260710915
</center>
"""

# Commented out IPython magic to ensure Python compatibility.
from google.colab import drive
drive.mount("/content/drive")

# %cd '/content/drive/My Drive/ECSE_551_F_2020/Mini_Project_03/'

import numpy as np
import pandas as pd
import torch
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import pickle

from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from PIL import Image

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

import torch.optim as optim

import torchvision.models as models

class MyDataset(Dataset):
    def __init__(self, img_file, label_file, transform=None, idx = None):
        self.data = pickle.load( open( img_file, 'rb' ), encoding='bytes')
        self.targets = np.genfromtxt(label_file, delimiter=',', skip_header=1)[:,1:]
        if idx is not None:
          self.targets = self.targets[idx]
          self.data = self.data[idx]
        self.transform = transform
        self.targets -= 5

    def __len__(self):
        return len(self.targets)

    def __getitem__(self, index):
        img, target = self.data[index], int(self.targets[index])
        img = Image.fromarray(img.astype('uint8'), mode='L')

        if self.transform is not None:
            img = self.transform(img)

        return img, target

img_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

train_index = np.arange(50000)
test_index = np.arange(50000, 60000)
batch_size = 32 #feel free to change it

# Read image data and their label into a Dataset class
train_set = MyDataset('./Train.pkl', './TrainLabels.csv', transform=img_transform, idx=train_index)
train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True, num_workers=2)
test_set = MyDataset('./Train.pkl', './TrainLabels.csv', transform=img_transform, idx=None)
test_loader = DataLoader(test_set, batch_size=batch_size, shuffle=True, num_workers=2)

"""# Notice: In case the code blocks are not in the right order, please run the blocks containing train() and test() functions every time after [model, criterion and optimizer] definition and before calling them.

# ResNet18
"""

resnet18 = models.resnet18()

net = resnet18
net.conv1 = nn.Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
net.fc = nn.Linear(512, 9)
# if there is a available cuda device, use GPU, else, use CPU
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
net = net.to(device)

# set criterion to cross entropy loss
criterion = nn.CrossEntropyLoss()

# set learning rate to 0.001
optimizer = optim.SGD(net.parameters(), lr=0.001)
# optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.5)

epoch = 32
train(epoch)

test()

"""# AlexNet"""

alexnet = models.alexnet()

net = alexnet
net.features[0] = nn.Conv2d(1, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
net.classifier[6] = nn.Linear(in_features=4096, out_features=9, bias=True)
# if there is a available cuda device, use GPU, else, use CPU
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
net = net.to(device)
```

```python

# set criterion to cross entropy loss
criterion = nn.CrossEntropyLoss()

# set learning rate to 0.001
# optimizer = optim.SGD(net.parameters(), lr=0.001)
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.8)

epoch = 32
train(epoch)

test()

"""# VGG16"""

vgg16 = models.vgg16()

net = vgg16
net.features[0] = nn.Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
net.classifier[6] = nn.Linear(in_features=4096, out_features=9, bias=True)
# if there is a available cuda device, use GPU, else, use CPU
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
net = net.to(device)

# set criterion to cross entropy loss
criterion = nn.CrossEntropyLoss()

# set learning rate to 0.001
optimizer = optim.SGD(net.parameters(), lr=0.001)
# optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.8)

epoch = 32
train(epoch)

test()

"""# VGG19"""

vgg19 = models.vgg19()

net = vgg19
net.features[0] = nn.Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
net.classifier[6] = nn.Linear(in_features=4096, out_features=9, bias=True)
# if there is a available cuda device, use GPU, else, use CPU
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
net = net.to(device)

# set criterion to cross entropy loss
criterion = nn.CrossEntropyLoss()

# set learning rate to 0.001
# optimizer = optim.SGD(net.parameters(), lr=0.001)
optimizer = optim.SGD(net.parameters(), lr=0.005, momentum=0.5)

# with: optimizer = optim.SGD(net.parameters(), lr=0.001)
epoch = 32
train(epoch)

test()

# with: optimizer = optim.SGD(net.parameters(), lr=0.005, momentum=0.5)
```

```python
epoch = 32
train(epoch)

test()

"""# VGG19-bn"""

vgg19bn = models.vgg19_bn()

net = vgg19bn
net.features[0] = nn.Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
net.classifier[6] = nn.Linear(in_features=4096, out_features=9, bias=True)
# if there is a available cuda device, use GPU, else, use CPU
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
net = net.to(device)

# set criterion to cross entropy loss
criterion = nn.CrossEntropyLoss()

# set learning rate to 0.001
# optimizer = optim.SGD(net.parameters(), lr=0.001)
optimizer = optim.SGD(net.parameters(), lr=0.005, momentum=0.5)

def train(num_epochs=2): # Feel free to change it
    net.train()

    running_loss = 0.0

    # Here is a piece of code that reads data in batch.
    # In each epoch all samples are read in batches using dataloader
    for epoch in range(num_epochs):
        for i, data in enumerate(train_loader):
            img, label = data

            img = img.to(device)
            label = label.to(device)

            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = net(img)

            loss = criterion(outputs, label)
            # loss = F.nll_loss(outputs, label)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            if i % 320 == 319:  # print every 320 mini-batches
                print('[%d, %5d] loss: %.3f' %
                    (epoch + 1, i + 1, running_loss / 320))
                running_loss = 0.0

                torch.save(net.state_dict(), '/model.pth')
                torch.save(optimizer.state_dict(), '/optimizer.pth')

    print('Finished Training')

def test():
    net.eval()

    correct = 0
    total = 0

    # calculate accuracy
    with torch.no_grad():
        for data in test_loader:
            images, labels = data

            images = images.to(device)
            labels = labels.to(device)

            outputs = net(images)
            # get the index of the max output
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    print('Accuracy of the network: %d %%' % (
        100 * correct / total))

# with: optimizer = optim.SGD(net.parameters(), lr=0.001)
epoch = 32
train(epoch)

test()

# with: optimizer = optim.SGD(net.parameters(), lr=0.005, momentum=0.5)
epoch = 32
train(epoch)

test()
```