Annex 3: CNN Training Code

November 30, 2018

In [ ]:
```
# -*- coding: utf-8 -*-

"""
Kaggle competition.
"""

__authors__ = "Jimmy Leroux, Nicolas Laliberte, Olivier Malenfant-Thuot,\
    Simon Dufort-Labbé"
__version__ = "1.0"
__maintainer__ = "Jimmy Leroux"
__studentid__ = "1024610, 1005803, 1012818, 0925272"

import torch
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import time
import csv


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.drop = nn.Dropout2d(p=0.05)
        self.dropfc = nn.Dropout(p=0.20)
        self.conv1 = nn.Conv2d(1, 128, 3, padding=1) #was 5 96
        self.conv2 = nn.Conv2d(128, 128, 3, padding=1)  #44
        self.conv3 = nn.Conv2d(128, 256, 3, padding=1) #22
        self.conv4 = nn.Conv2d(256, 256, 3, padding=1)#9
        self.conv5 = nn.Conv2d(256, 512, 3, padding=1)
        self.conv6 = nn.Conv2d(512, 512, 3, padding=1)
        self.conv7 = nn.Conv2d(512, 512, 3, padding=1)
        self.bn1 = nn.BatchNorm2d(128)
```

```
        self.bn2 = nn.BatchNorm2d(128)
        self.bn3 = nn.BatchNorm2d(256)
        self.bn4 = nn.BatchNorm2d(256)
        self.bn5 = nn.BatchNorm2d(512)
        self.bn6 = nn.BatchNorm2d(512)
        self.bn7 = nn.BatchNorm2d(512)
        self.bn8 = nn.BatchNorm1d(1024)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(512 * 5 * 5, 1024)
        self.fc2 = nn.Linear(1024, 31)

    def forward(self, x):
        x = self.drop((self.bn1(F.relu(self.conv1(x))))) #48
        x = self.drop(self.pool(self.bn2(F.relu(self.conv2(x))))) #22
        x = self.drop((self.bn3(F.relu(self.conv3(x)))))
        x = self.drop(self.pool(self.bn4(F.relu(self.conv4(x)))))
        x = self.drop((self.bn5(F.relu(self.conv5(x)))))
        x = self.drop(self.pool(self.bn6(F.relu(self.conv6(x)))))
        x = self.drop((self.bn7(F.relu(self.conv7(x)))))
        x = x.view(-1, 512 * 5 * 5)
        x = self.dropfc(self.bn8(F.relu(self.fc1(x))))
        x = self.fc2(x)
        return x

class kaggle_dataset(torch.utils.data.dataset.Dataset):
    def __init__(self, data, labels, transforms=None):
        self.data = data
        self.labels = labels
        self.transforms = transforms

    def __getitem__(self, index):
        dat = self.data[index]
        if self.transforms is not None:
            dat = self.transforms(dat)
        return (dat,self.labels[index])

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

def define_classes(labels):
    classes = {}
    label = []
    c = 0
    for i,j in labels:
        if j.decode('utf-8') not in classes:
            classes[j.decode('utf-8')] = c
            c += 1
        label.append(classes[j.decode('utf-8')])
```

```
    return classes, torch.LongTensor(label)

def reshape_images(images):
    train_im = []
    #noise = torch.zeros(1,100,100)
    c = 0
    #_,label = define_classes(train_labels)
    #for i in range(images.shape[0]):
    #    if label[i]==21:
    #        noise += torch.Tensor(images[i][1]).reshape((1,100,100)) / 255.
    #        c += 1.
    #noise /= c
    for i in range(images.shape[0]):
        train_im.append(torch.Tensor(images[i][1].reshape((1,40,40))) / 255.)
    return train_im

def train(images, labels):

    transform = transforms.Compose(
        [transforms.ToPILImage(), transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(),transforms.RandomApply(
        [transforms.RandomRotation(10)],0.0),transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

    trainset = kaggle_dataset(
        reshape_images(images)[:9500],define_classes(labels)[1][:9500],
            transforms=transform)

    testset = kaggle_dataset(
        reshape_images(images)[9500:],define_classes(labels)[1][9500:],
            transforms=transform)

    trainloader = torch.utils.data.DataLoader(trainset, batch_size=64,
                                          shuffle=True, num_workers=0)

    testloader = torch.utils.data.DataLoader(testset, batch_size=64,
                                         shuffle=False, num_workers=0)

    classes = [str(i) for i in range(31)]
    net = Net().to(device)
    criterion = nn.CrossEntropyLoss(weight=weight(
        define_classes(labels)[1][:9500]).to(device))
    optimizer = optim.SGD(net.parameters(), lr=0.01, weight_decay=0.025,
        momentum=0.9) #wc was 0.01

    loss_train = []
    loss_test = []
    err_train = []
```

```
err_test = []
for epoch in range(35):  # loop over the dataset multiple times
    if epoch>10:
        optimizer = optim.SGD(net.parameters(), lr=0.001,
            weight_decay=0.025, momentum=0.9) #lr 0.001
    if epoch>20: #was30
        optimizer = optim.SGD(net.parameters(), lr=0.0005,
            weight_decay=0.025, momentum=0.9)
    if epoch>30: #was40
        optimizer = optim.SGD(net.parameters(), lr=0.0001,
            weight_decay=0.025, momentum=0.9)
    correct = 0.
    total = 0.
    running_loss_train = 0.0
    running_loss_test = 0.0
    net.train()
    for i, data in enumerate(trainloader, 0):

        inputs, labels = data
        optimizer.zero_grad()
        outputs = net(inputs.to(device))
        loss = criterion(outputs, labels.to(device))
        loss.backward()
        optimizer.step()
        running_loss_train += loss.item() / 9500
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels.to(device)).sum().item()
    err_train.append(1 - correct / total)

    correct = 0.
    total = 0.
    net.eval()
    with torch.no_grad():
        for data in testloader:
            images, labels = data
            outputs = net(images.to(device))
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels.to(device)).sum().item()
            loss = criterion(outputs, labels.to(device))
            running_loss_test += loss.item() / 500
    err_test.append(1 - correct / total)

    loss_train.append(running_loss_train)
    loss_test.append(running_loss_test)
    print('Epoch: {}'.format(epoch))
    print('Train loss: {0:.4f} Train error: {1:.2f}'.format(
```

```
            loss_train[epoch], err_train[epoch]))
        print('Test loss: {0:.4f} Test error: {1:.2f}'.format(
            loss_test[epoch], err_test[epoch]))

    print('Finished Training')

    correct = 0
    total = 0
    net.eval()
    with torch.no_grad():
        for i in range(10):
            for data in testloader:
                images, labels = data
                outputs = net(images.to(device))
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels.to(device)).sum().item()

    print('Accuracy of the network on the 1000 test images: %d %%' % (
        100 * correct / total))

    class_correct = list(0. for i in range(31))
    class_total = list(0. for i in range(31))
    net.eval()
    with torch.no_grad():
        for i in range(10):
            for data in testloader:
                images, labels = data
                outputs = net(images.to(device))
                _, predicted = torch.max(outputs, 1)
                c = (predicted == labels.to(device)).squeeze()
                for i in range(c.shape[0]):
                    label = labels[i]
                    class_correct[label] += c[i].item()
                    class_total[label] += 1

    for i in range(31):
        if class_total[i]!=0:
            print('Accuracy of %5s : %2d %%' % (
                classes[i], 100 * class_correct[i]/class_total[i]))
        else:
            print('Accuracy of %5s : %2d %%' % (
                classes[i], 100 * class_correct[i]))
    return net, loss_train, loss_test, err_train, err_test

def predict(net, datas, classes):
    prediction = []
    net.eval()
```

```
    with torch.no_grad():
        for data in datas:
            image, dummy = data
            outputs = net(image.to(device))
            _, predicted = torch.max(outputs, 1)
            prediction.extend(predicted.tolist())
    with open('submission.csv', mode='w') as submission:
        writer = csv.writer(submission, delimiter=',',
            quotechar='"', quoting=csv.QUOTE_MINIMAL)
        for i in range(len(prediction)):
            prediction[i] = classes[prediction[i]]
            writer.writerow([str(i), prediction[i]])
    return prediction

def weight(labels):
    scale = torch.FloatTensor(31)
    for i in range(31):
        scale[i] = ((labels==i).sum())
    return scale.max() / scale

if __name__ == '__main__':
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    torch.cuda.manual_seed(10)
    plt.style.use('ggplot')
    plt.rc('xtick', labelsize=15)
    plt.rc('ytick', labelsize=15)
    plt.rc('axes', labelsize=15)
    classes = {3:'apple',21:'empty',23:'moustache',6:'mouth',30:'mug',19:'nail',
        13:'nose',22:'octagon',24:'paintbrush',25:'panda',26:'parrot',9:'peanut',16:'pea
        20:'pencil',18:'penguin',17:'pillow',5:'pineapple',15:'pool',10:'rabbit',
        29:'rhinoceros',1:'rifle',8:'rollerskates',12:'sailboat',2:'scorpion',27:'screwd
        0:'shovel',11:'sink',7:'skateboard',14:'skull',4:'spoon',28:'squiggle'}
    #images_train = np.load('C:/Users/Jimmy/Desktop/Kaggle/Train_images.npy',
    #    encoding='latin1')

    #train_labels = np.genfromtxt('C:/Users/Jimmy/Desktop/Kaggle/train_labels.csv',
    #    names=True, delimiter=',', dtype=[('Id', 'i8'), ('Category', 'S5')])

    #images_test = np.load('C:/Users/Jimmy/Desktop/Kaggle/test_images.npy',
    #    encoding='latin1')

    images_train = np.load(
        'C:/Users/Jimmy/Desktop/Kaggle/cleaned_images.npy',
        encoding='latin1')

    train_labels = np.genfromtxt(
        'C:/Users/Jimmy/Desktop/Kaggle/train_labels.csv',
        names=True, delimiter=',', dtype=[('Id', 'i8'), ('Category', 'S5')])
```

```
images_test = np.load('C:/Users/Jimmy/Desktop/Kaggle/cleaned_test_images.npy',
    encoding='latin1')

transform = transforms.Compose(
    [transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
images_test = reshape_images(images_test)

images_test_dataset = kaggle_dataset(
    images_test, torch.tensor([0.]*len(images_test)),transforms=transform)

test_loader = torch.utils.data.DataLoader(images_test_dataset, batch_size=64,
    shuffle=False, num_workers=0)
t1 = time.time()

net, loss_train, loss_test, e_train, e_test = train(images_train, train_labels)
prediction = predict(net, test_loader, classes)
print(time.time()-t1)

plt.figure()
plt.plot(range(1,36),loss_train, 'sk-', label='Train')
plt.plot(range(1,36),loss_test, 'sr-', label='Valid')
plt.xlabel('Epoch')
plt.ylabel('Average loss', labelpad=0)
plt.legend()

plt.figure()
plt.plot(range(1,36),e_train, 'sk-', label='Train')
plt.plot(range(1,36),e_test, 'sr-', label='Valid')
plt.xlabel('Epoch')
plt.ylabel('Error', labelpad=0)
plt.legend()

torch.cuda.empty_cache()
plt.show()
```