CNN

April 2, 2022

0.1 Creating Labels for training

[1]:
```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.optim as optim
import time
from itertools import count
import natsort
```

[2]:
```
from torch.utils.data import Dataset, DataLoader
import albumentations as A
from albumentations.pytorch import ToTensorV2
import cv2
import glob
import numpy
import random
import pandas as pd
import tqdm
from sklearn import metrics
import matplotlib.pyplot as plt
```

[3]:
```
train_transforms = A.Compose(
    [
        A.Resize(224,224),
        A.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5)),
        ToTensorV2(),
    ]
)
```

[4]:
```
train_image_paths = []
```

[5]:
```
for i in range(1,71):
    filename = '/home/zo2151/assignments/Data/Video%i'%(i,)
    train_image_paths.append(glob.glob(filename + '/*'))
```

[6]:
```
train_image_paths1 = [item for sublist in train_image_paths for item in sublist]
```

[7]:
```
train_image_paths1 = natsort.natsorted(train_image_paths1)
```

[8]:
```
df = pd.read_csv("/home/zo2151/Processed_data.csv")
df1 = df.loc[:,"Phases"].to_numpy()
```

[9]:
```
df2 = df1.tolist()
```

[10]:
```
percentile_list = pd.DataFrame(
    {'Link': train_image_paths1,
     'Label': df2,
    })
```

[11]:
```
percentile_list1 = percentile_list.sample(frac=1, random_state=1)
```

[12]:
```
train_image_paths = percentile_list1.loc[:,"Link"].to_numpy().tolist()
```

[13]:
```
labels = percentile_list1.loc[:,"Label"].to_numpy().tolist()
```

[14]:
```
train_image_paths, valid_image_paths = train_image_paths[:int(0.
 ↪8*len(train_image_paths))], train_image_paths[int(0.
 ↪8*len(train_image_paths)):]
```

[15]:
```
train_labels, valid_labels = labels[:int(0.8*len(labels))], labels[int(0.
 ↪8*len(labels)):]
```

[16]:
```
class SurgicalDataset(Dataset):
    def __init__(self, image_paths, labels, transform=False):
        super(SurgicalDataset, self).__init__()
        self.image_paths = image_paths
        self.transform = transform
        self.labels = labels

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

    def __getitem__(self, idx):
        image_filepath = self.image_paths[idx]
        image = cv2.imread(image_filepath)

        label = self.labels[idx]
        if self.transform is not None:
            image = self.transform(image=image)["image"]

        return image, label
```

[17]:
```
train_dataset = SurgicalDataset(train_image_paths,train_labels,␣
 ↪train_transforms)
```

```
val_dataset = SurgicalDataset(valid_image_paths,valid_labels, train_transforms)
```

[18]:
```
train_loader = DataLoader(
    train_dataset, batch_size=1024, shuffle=True
)

valid_loader = DataLoader(
    val_dataset, batch_size=1024, shuffle=True
)
```

[19]:
```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
cuda:0
```

0.2 The model with CNN and FC layers

[20]:
```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 53 * 53, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 14)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
net.to(device)
```

[20]:
```
Net(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
```

```
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=44944, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=14, bias=True)
)
```

[21]:
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001)
```

[23]:
```
checkpoint = torch.load("/home/zo2151/model1.pt")
net.load_state_dict(checkpoint['model_state_dict'])
optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
```

[24]:
```
for epoch in range(4):  # trained 5 epochs, after 1 epoch, connection lost
    t = time.time()
    running_loss = 0.0
    loop = tqdm.tqdm(train_loader, total = len(train_loader), leave = True)
    for img, label in loop:
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = img.to(device), label.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
            running_loss = 0.0
    d = time.time()-t
    print(d)
    torch.save({
            'epoch': 2,
            'model_state_dict': net.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            }, "/home/zo2151/model1.pt")
print('Finished Training')
```

```
100%|                                        |
169/169 [20:31<00:00,  7.29s/it]

1231.758453130722
```

```
100%|                                           |
169/169 [19:02<00:00,  6.76s/it]

1142.068077325821

100%|                                           |
169/169 [19:16<00:00,  6.84s/it]

1156.2235412597656

100%|                                           |
169/169 [19:06<00:00,  6.79s/it]

1146.8516116142273
Finished Training
```

[25]:
```
classes = [i for i in range(14)]
correct_pred = {classname: 0 for classname in classes}
total_pred = {classname: 0 for classname in classes}
pr = []
pred = []
l = []
# again no gradients needed
t = time.time()
with torch.no_grad():
    for data in valid_loader:
        images, labels = data[0].to(device), data[1].to(device)
        l.append(labels)
        outputs = net(images)
        _, predictions = torch.max(outputs, 1)
        m = F.softmax(outputs, dim=1)
        # collect the correct predictions for each class
        for label, prediction in zip(labels, predictions):
            pred.append(prediction)
            if label == prediction:
                correct_pred[classes[label]] += 1
            total_pred[classes[label]] += 1
        for p in m:
            pr.append(p)
print(time.time()-t)
print(correct_pred)
print(total_pred)
# print accuracy for each class
#for classname, correct_count in correct_pred.items():
    #accuracy = 100 * float(correct_count) / total_pred[classname]
    #print(f'Accuracy for class: {classname:5s} is {accuracy:.1f} %')
```

```
305.8759560585022
{0: 17, 1: 2, 2: 423, 3: 5243, 4: 39, 5: 268, 6: 9407, 7: 410, 8: 1810, 9: 5444,
10: 208, 11: 9659, 12: 372, 13: 1886}
```

```
{0: 52, 1: 18, 2: 587, 3: 5470, 4: 213, 5: 288, 6: 11135, 7: 2177, 8: 3001, 9:
5799, 10: 247, 11: 10405, 12: 404, 13: 3216}
```

```
[26]: for i in range(len(l)):
          l[i] = l[i].cpu()
      for i in range(len(l)):
          l[i] = l[i].data.numpy()
      l = [item for sublist in l for item in sublist]
      for i in range(len(l)):
          pred[i] = pred[i].cpu().data.numpy()
      for i in range(len(l)):
          pr[i] = pr[i].cpu().data.numpy()
```

0.3 Some Metrics

```
[27]: metrics.accuracy_score(l, pred)

[27]: 0.8180972751790198

[28]: metrics.f1_score(l, pred, average="macro")

[28]: 0.6807584625348103

[29]: metrics.precision_score(l, pred, average=None)

[29]: array([1.        , 1.        , 0.79962193, 0.8035249 , 0.8125    ,
             0.96402878, 0.9500101 , 0.98795181, 0.73220065, 0.68229101,
             0.96296296, 0.86534671, 0.88151659, 0.61937603])

[30]: metrics.recall_score(l, pred, average=None)

[30]: array([0.32692308, 0.11111111, 0.72061329, 0.95850091, 0.18309859,
             0.93055556, 0.84481365, 0.18833257, 0.60313229, 0.93878255,
             0.84210526, 0.9283037 , 0.92079208, 0.58644279])

[31]: auc = metrics.roc_auc_score(l, pr, multi_class = "ovr")

[32]: auc

[32]: 0.9842028264837943

[33]: auc = metrics.roc_auc_score(l, pr, multi_class = "ovo")

[34]: auc

[34]: 0.9753112261267696
```

[35]:
```
mem_params = sum([param.nelement()*param.element_size() for param in net.
↪parameters()])
mem_bufs = sum([buf.nelement()*buf.element_size() for buf in net.buffers()])
mem = mem_params + mem_bufs
```

[36]:
```
mem
```

[36]:
```
21630504
```