Intro to ML Final Project

Daniel Sun

December 12, 2022

1 Method

1.1 Import Packages

[1]:
```
import torch
import torch.nn as nn
from torch.utils.data import Dataset
from torchvision import transforms
import numpy as np
import os
import cv2
import pickle as pkl
from torchvision.models import resnet50, ResNet50_Weights
```

1.2 Define Device to use

[2]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

1.3 Load and Normalize Data

1.3.1 Define Lazy Loader Class

We only do transformation and normalization on the imgs. Doing things on depth seems to have negative effect on result.

[3]:
```
class LazyLoadDataset(Dataset):
  def __init__(self, path, train=True, transform=None):
    self.transform = transform
    path = path + ("train/" if train else "test/")

    self.pathX = path + "X/"
    self.pathY = path + "Y/"

    self.data = os.listdir(self.pathX)
    self.train = train

  def __getitem__(self, idx):
    f = self.data[idx]
    img0 = cv2.imread(self.pathX + f + "/rgb/0.png")
    img1 = cv2.imread(self.pathX + f + "/rgb/1.png")
    img2 = cv2.imread(self.pathX + f + "/rgb/2.png")

    if self.transform is not None:
      img0 = self.transform(img0)
      img1 = self.transform(img1)
      img2 = self.transform(img2)

    depth = np.load(self.pathX + f + "/depth.npy")

    field_id = pkl.load(open(self.pathX + f + "/field_id.pkl", "rb"))

    if self.train:
      Y = np.load(self.pathY + f + ".npy") * 1000
      return (img0, img1, img2, depth), Y
    else:
      return (img0, img1, img2, depth)

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

1.3.2 Define transformation and normalization functions

Here we use `transform` from `torchvision` to do transformation and normalization.

[4]:
```
data_transforms = {
    'train': transforms.Compose([
        transforms.ToPILImage(),
        transforms.RandomRotation(45),
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
    ]),
    'test': transforms.Compose([
        transforms.ToPILImage(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
    ]),
}
```

1.3.3 Define train dataseet and dataloader

```
[5]: train_dataset = LazyLoadDataset("./lazydata/",␣
      ↪transform=data_transforms['train'])
     train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=30,␣
      ↪shuffle=False)
```

1.4 Training the model

1.4.1 Define the training function

This is a function to run the training for one epoch

```
[6]: def train(epoch, model, optimizer, permute_pixels=None, permutation_order=None):
         model.train()
         total_loss = 0
         for batch_idx, ((img0, img1, img2, depth), target) in␣
      ↪enumerate(train_loader):
             # send to device
             concate = torch.cat((img0, img1, img2, depth), dim=1)
             data, target = concate.to(device), target.to(device)

             # permute pixels
             if permute_pixels is not None:
                 data = permute_pixels(data, permutation_order)

             optimizer.zero_grad()
             model = model.to(device)
             output = model(data)
             lossFn = nn.MSELoss()
             loss = lossFn(output.float(), target.float())
             total_loss += loss
             loss.backward()
             optimizer.step()

         # print average loss
         print('Train Epoch: {} \tAvg Loss: {:.6f}'.format(
                   epoch, total_loss/len(train_loader.dataset)))
```

1.4.2 Define the Training Model

I use the `resnet50` model as a base. And modify the `fc` and `conv1` layer to accommendate the training data.

```
[7]: model = resnet50(weights=ResNet50_Weights.DEFAULT)
     model = model.to(device)
     model.eval()
     model.float()
     model.fc = nn.Linear(2048, 12)
```

```
weight = model.conv1.weight.clone()
model.conv1 = nn.Conv2d(12, 64, kernel_size=7, stride=2, padding=3, bias=False)
with torch.no_grad():
    model.conv1.weight[:, :3] = weight
    model.conv1.weight[:, 3] = model.conv1.weight[:, 0]
```

1.4.3 Run Tranning

I use SGD optimizer and run the model for 40 epoches

[8]:
```
# define the optimizer
optimizer = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.9)

# train the model for 40 epoches
for epoch in range(0, 40):
    train(epoch, model, optimizer)
```

1.4.4 Export Test Prediction

Mainly using the provided script but transform the input

[9]:
```
import pandas as pd

outfile = 'submission.csv'

output_file = open(outfile, 'w')

titles = ['ID', 'FINGER_POS_1', 'FINGER_POS_2', 'FINGER_POS_3', 'FINGER_POS_4',␣
↪'FINGER_POS_5', 'FINGER_POS_6',
         'FINGER_POS_7', 'FINGER_POS_8', 'FINGER_POS_9', 'FINGER_POS_10',␣
↪'FINGER_POS_11', 'FINGER_POS_12']
preds = []

test_data = torch.load('./test/test/testX.pt')
file_ids = test_data[-1]
rgb_data = test_data[0]
depth = test_data[1]
model.eval()

for i, (img0, img1, img2) in enumerate(rgb_data):

    img0 = data_transforms['test'](img0)
    img1 = data_transforms['test'](img1)
    img2 = data_transforms['test'](img2)

    data = torch.cat((img0, img1, img2, depth[i]), dim=0)
    data = torch.unsqueeze(data, 0)
    output = model(data.to(device)) / 1000
    preds.append(output[0].cpu().detach().numpy())

df = pd.concat([pd.DataFrame(file_ids), pd.DataFrame.from_records(preds)], axis␣
 ↪= 1, names = titles)
df.columns = titles
df.to_csv(outfile, index = False)
print("Written to csv file {}".format(outfile))
```

2 Experiment Result

The highest public score of the prediction on Kaggle is 0.00372.
However, every time the model run, the score varies from 0.00372 to 0.00628.

3 Discussion

3.1 Loading

- Lazy loading can save a lot of time
- It is also memory friendly
- Large batch size can result in memory shortage so decreases batch size to save GPU

3.2 Normalization

- Normalization on RGB images can improve the performance of the model tremendously
- Possibly because some of the data are very large compare to the other ones
- It is very useful to multiple the Y by 1000 when loading the data as the Y is smaller than X
- Transform depth image doesn't improve the performance much

3.3 Model

- Using pre-trained model(like resnet50 in this case) can provide a good start
- Need to modify the fc layer of the model to accomadate the data
- SGD optimizer in my case perform better than Adam optimizer
- Running multiple epoches can improve the model tremendously

4 Future Work

- Trying different normalization methods to see if we can achieve a better result
- Trying more optimizers or modifying optimizers' parameters like learning rate
- Using other pre-trained or self-defined models
- Running the model for more epoches