Final Project Report

Method

The libraries that I used:

```
import os
from torchvision.io import read_image
import numpy as np
from torch.utils.data import Dataset
import torchvision
import torchvision.transforms as transforms
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import pickle as pkl
import itertools
import timm
import matplotlib.pyplot as plt
import pandas as pd
```

I used deep learning methods for my final project. The tool package I mainly use is pytorch. When doing this project, I read the data files by using the lazy loading method provided by the TAs and professor. Then, I build a dataset class to organize data from files into tensor types. We cannot further build the deep learning model without converting the data into tensor type.

```
class RoboticDataset(Dataset):
    def __init__(self, img_dir, val=False,train_size=0.8,load_y=True,transform=None,d_transform=None):
        self.img_idx_list = os.listdir(os.path.join(img_dir,'X'))
        if val:
            self.img_idx_list = self.img_idx_list[int(len(self.img_idx_list) * train_size):]
        else:
            self.img_idx_list = self.img_idx_list[:int(len(self.img_idx_list) * train_size)]
        self.img_dir = img_dir
        self.transform = transform
        self.d_transform = d_transform
        self.load_y = load_y

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

    def __getitem__(self, idx):
        img_path = os.path.join(os.path.join(self.img_dir,'X'), self.img_idx_list[idx])
        if self.load_y:
            y_path = os.path.join(os.path.join(self.img_dir,'Y'), self.img_idx_list[idx] + ".npy")
            coordinates = torch.tensor(np.load(y_path))
        d_data = torch.tensor(np.load(os.path.join(img_path,"depth.npy")))
        rgbd_img = torch.empty(3,4,224,224)
        rgbd_img[:,3,:,:] = d_data
        for i in range(3):
            rgbd_img[i,:3,:,:] = read_image(os.path.join(os.path.join(img_path,'rgb'), "{}.png".format(i)))
        f = open(os.path.join(img_path,'field_id.pkl'),'rb')
        sample_id = pkl.load(f)
        f.close()
        if self.transform is not None:
            for i in range(3):
                rgbd_img[i,:3,:,:] = self.transform(rgbd_img[i,:3,:,:])
                rgbd_img[i,3,:,:] = self.d_transform(rgbd_img[i,3,:,:])
        if self.load_y:
            return rgbd_img, coordinates.float(), sample_id
        else:
            return rgbd_img.float(), sample_id
```

Then, I have implemented data preprocessing. I calculate the corresponding mean and standard deviation in order to normalize tensor images. Furthermore, in order to get more useful information for the dataset and increase the performance of my neural network, I implement automatic data augmentation.

```
def find_mean_and_std():
    transform=transforms.Compose([
            transforms.ToPILImage(),
            transforms.ToTensor(),
        ])
    all_dataset = RoboticDataset("./lazydata/train",transform=transform,d_transform=transform,val=False,train_size=1)
    all_dataloader = torch.utils.data.DataLoader(all_dataset,batch_size=batch_size,num_workers=16)
    mean_,std_ = torch.zeros(4),torch.zeros(4)
    for batch_idx, (inputs, targets, sample_id) in enumerate(all_dataloader):
        inputs_rgb_imgs = inputs[:,:,:3,:,:].view(-1,3,224*224)
        inputs_d = inputs[:,:,3,:,:].view(-1,224*224)
        mean_[:3] += inputs_rgb_imgs.mean(axis=2).mean(axis=0)
        std_[:3] += inputs_rgb_imgs.std(axis=2).mean(axis=0)
        mean_[3] += inputs_d.mean(axis=1).mean(axis=0)
        std_[3] += inputs_d.std(axis=1).mean(axis=0)
    return mean_ / len(all_dataloader), std_ / len(all_dataloader)
find_mean_and_std()
```

```
(tensor([0.5570, 0.5416, 0.5187, 0.3739]),
 tensor([0.2366, 0.2176, 0.2187, 0.3120]))
```

```
DATA_MEAN = [0.5570, 0.5416, 0.5187, 0.3739]
DATA_STD = [0.2366, 0.2176, 0.2187, 0.3120]

normalize = transforms.Normalize(mean=DATA_MEAN[:3], std=DATA_STD[:3])
train_transform=transforms.Compose([
            transforms.ToPILImage(),
            transforms.AutoAugment(),
            transforms.ToTensor(),
            normalize,
        ])
test_transform=transforms.Compose([
            transforms.ToPILImage(),
            transforms.ToTensor(),
            normalize,
        ])
depth_transform=transforms.Compose([
            transforms.ToPILImage(),
            transforms.ToTensor(),
            transforms.Normalize(mean=DATA_MEAN[3], std=DATA_STD[3]),
        ])

train_dataset = RoboticDataset("./lazydata/train",transform=train_transform,d_transform=depth_transform,val=False)
val_dataset = RoboticDataset("./lazydata/train",transform=test_transform,d_transform=depth_transform,val=True)
test_dataset = RoboticDataset("./lazydata/test",transform=test_transform,d_transform=depth_transform,load_y=False,val=F
```

The next step is to build the model. I notice that the datasets are RGB-D images and the data sets are really large. The traditional convolutional neural network might have the problem of vanishing gradient when there are more layers in the network. In order to solve this problem, I choose to use the ResNet50 network since using identity mapping could solve the problem of vanishing gradient. ResNet50 is a 50 layer convolutional neural network, including 48 convolutional layers, one MaxPool layer, and one average pool layer, and it improves the efficiency of deep neural networks with more neural layers while minimizing the percentage of errors. While building the model, because the dataset contains 4 channels, I change the input of the first convolutional layer of ResNet50 from 3 to 4.

```
import timm
net = timm.create_model("resnet50", pretrained=False)
net.fc = nn.Linear(net.fc.in_features, 12)
net.conv1 = nn.Conv2d(4, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
net = net.cuda()
```

In order to avoid overfitting, I re-divided the given training set to get the validation set. I used Adam optimizer, and searched for epoch and learning rate.

```
n_epochs = 100
lr = 1e-3
optimizer = optim.Adam(net.parameters(), lr=lr)
# use cosine scheduling
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, n_epochs)
criterion = nn.MSELoss()
```

```
##### Training
def train(epoch):
    net.train()
    train_loss = 0
    for batch_idx, (inputs, targets, sample_id) in enumerate(trainloader):
        inputs, targets = inputs.cuda(), targets.cuda()
        #inputs_rgb_imgs = inputs[:,:,:3,:,:].view(-1,3,224,224)
        #inputs_d = inputs[:,:,3,:,:].view(-1,1,224,224)
        #outputs = net(inputs_rgb_imgs)
        #outputs_d = net_d(inputs_d.expand(inputs_rgb_imgs.shape[0],3,224,224))
        #predy = ((outputs + outputs_d)/2).view(-1,3,12).mean(axis=1)
        predy = net(inputs.view(-1,4,224,224)).view(-1,3,12).mean(axis=1)
        loss = criterion(predy, targets)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        train_loss += loss.item()
        if batch_idx % 50 == 0:
            print('[{}/{}]: Train Loss: {:.5f}\r'.format(batch_idx,len(trainloader),loss.item()))
    return train_loss/(batch_idx+1)
```

```
##### Validation
def test(epoch):
    global best_acc
    net.eval()
    test_loss = 0
    with torch.no_grad():
        for batch_idx, (inputs, targets, sample_id) in enumerate(valloader):
            inputs, targets = inputs.cuda(), targets.cuda()
            #inputs_rgb_imgs = inputs[:,:,:3,:,:].view(-1,3,224,224)
            #inputs_d = inputs[:,:,3,:,:].view(-1,1,224,224)
            #outputs = net(inputs_rgb_imgs)
            #outputs_d = net_d(inputs_d.expand(inputs_rgb_imgs.shape[0],3,224,224))
            #predy = ((outputs + outputs_d)/2).view(-1,3,12).mean(axis=1)
            predy = net(inputs.view(-1,4,224,224)).view(-1,3,12).mean(axis=1)
            loss = criterion(predy, targets)

            test_loss += loss.item()

            if batch_idx % 50 == 0:
                print('[{}/{}]: Val Loss: {:.5f}'.format(batch_idx,len(valloader),loss.item()))

    return test_loss
```

Finally, I trained on the full dataset (including the split validation set) with appropriate learning rate and epoch parameters and used the final model to make predictions on the test set.

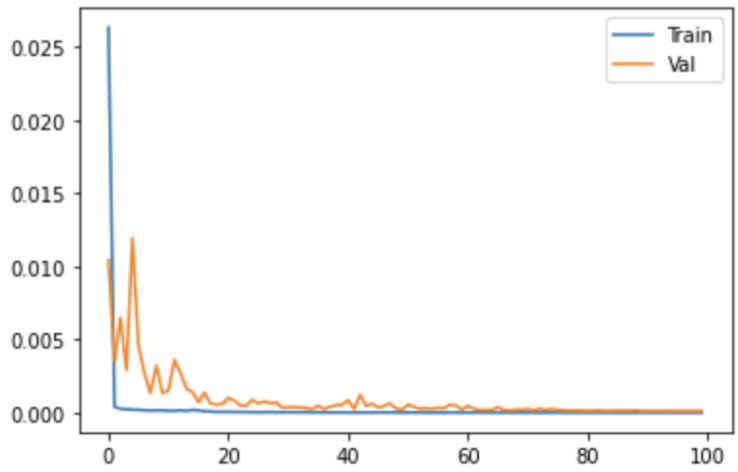
```
##### Training with all data
all_dataset = RoboticDataset("./lazydata/train",transform=train_transform,d_transform=depth_transform,val=False,train_s
all_dataloader = torch.utils.data.DataLoader(all_dataset,batch_size=batch_size,num_workers=16)

net = timm.create_model("resnet50", pretrained=False)
net.fc = nn.Linear(net.fc.in_features, 12)
net.conv1 = nn.Conv2d(4, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
net = net.cuda()

n_epochs = 100
lr = 1e-3
optimizer = optim.Adam(net.parameters(), lr=lr)
# use cosine scheduling
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, n_epochs)
criterion = nn.MSELoss()

def train(epoch):
    net.train()
    train_loss = 0
    for batch_idx, (inputs, targets, sample_id) in enumerate(all_dataloader):
        inputs, targets = inputs.cuda(), targets.cuda()
        predy = net(inputs.view(-1,4,224,224)).view(-1,3,12).mean(axis=1)
        loss = criterion(predy, targets)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        train_loss += loss.item()
        if batch_idx % 50 == 0:
            print('[{}/{}]: Train Loss: {:.5f}\r'.format(batch_idx,len(all_dataloader),loss.item()))
    return train_loss/(batch_idx+1)

for epoch in range(n_epochs):
    #start = time.time()
    print("Epoch",epoch)
    train_loss = train(epoch)
```

Experimental results

After parameter search, the parameters I got are epochs = 100, learning rate = 0.001, and batch_size = 64.

The results of the training set and validation set are as follows:

```
Epoch 99
[0/43]: Train Loss: 0.00000
[0/11]: Val Loss: 0.00001
```



The training results of all data sets (including the validation set) are as follows:

```
Epoch 99
[0/54]: Train Loss: 0.00002
[50/54]: Train Loss: 0.00002
```

After submitting the submission.csv file in the kaggle, I got a score of 0.00708 in the public leaderboard

Discussion

Before using the above method, I first tried to use Resnet101 since I thought more layers would have a better result. However, I found out the time to train is slow. Moreover, after doing some research, I learned that Resnet101 might overly care about the details. Resnet101 is more likely to overfit compared with Resnet50. After weighing the advantages and disadvantages, I decided that ResNet50 might be the most suitable neural network for this project for processing speed and accuracy. Besides, I have tried feeding the depth information separately into another network, and the performance was not satisfactory. Larger values in the depth information do not contain enough valid information since this may be information far away. Treating depth as a separate channel seems like a good attempt. I also tried to use the pre-trained model, and I found that the pre-trained model sometimes does not converge, and the pre-trained model may not match the task of predicting coordinates.

Future work

Limited by time and computing power, I did not try more types of models such as Vision transformer, etc. These models may perform better on tasks. Also, I didn't try more on the pre-trained model in this project, I will try to apply the pre-trained model on RGB data to the RGB-D task. My encoding of depth may not be perfect, just have depth as another channel besides RGB. In the future, I will also try more ways to handle gradients, such as using the autoencoder.

Reference

Li, Y. (2020, April 24). Automating Data Augmentation: Practice, Theory and New Direction. Stanford AI Lab Blog. http://ai.stanford.edu/blog/data-augmentation/

He, K., Zhang, X., Ren, S., Sun, J. Deep residual learning for image recognition.https://arxiv.org/abs/1512.03385

Prakash, P. (2021, September 7). Torch Dataset and Dataloader – Early Loading of Data Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/09/torch-dataset-and-dataloader-early-loading-of-data/