CVDL2019 课程作业报告

简述

CNN，也即卷积神经网络，在图像处理上的表现可以说是十分出色。在2012年，Alexnet在ImageNet LSVRC比赛上获得了冠军，自此之后几乎所有的冠军都是利用CNN进行设计模型的，层数也越来越深，但随着层数的增加，模型的泛化能力会有所下降。2015年 Kaiming.He提出了Resnet（残差神经网络），通过加入Residual Block，解决了网络随着层数增加效果下降的问题。本次作业将使用Resnet-50作为网络，使用Pytorch深度学习框架，通过对 5,3881 张，共80个类别的数据集进行训练以及建模，最后在Kaggle上的场景分类任务上达到了66%的准确率。

数据描述

是次作业的数据集分为训练集与测试集两部分，训练集中共有5,3881张图片，测试集共7,121张图片。通过80%、20%的比例将数据划分成训练集和验证集。

使用设备

系统：*Ubuntu 16.04*

显卡：*NVIDIA® Tesla® P100　(Provided by Google Cloud Platform)*

模块读入

本次作业主要用到常用的深度学习模块，如torchvision, torch, numpy...

```
import pandas as pd
import torch
from torch.utils.data.dataset import Dataset
from torchvision import transforms
import numpy as np
from PIL import Image
#import matplotlib.pyplot as plt
import torchvision
from torch.utils.data.sampler import SubsetRandomSampler
import time
import os
import copy
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
from torchvision import datasets, models
from torch.autograd import Variable
```

数据处理

数据读入

首先通过Pandas把数据读入

```
label_df = pd.read_csv('cvdl2019/scene_classes.csv',header = None)
labels_ls = list(label_df.iloc[:,2])
testdatadir = 'cvdl2019/validation_images/'
testFileName = os.listdir(testdatadir)
```

因为使用的数据集处在一个独立的文件夹内，标签是以"File Name + Label"的方式存于csv文件中，所以需要对pytorch里面的dataset类进行重载，以便于加载进dataloader。

```
class MyTestDataset(Dataset):
    def __init__(self):
        """
         Args:
            csv_path (string): csv 文件路径
            img_path (string): 图像文件所在路径
            transform: transform 操作
        """
        #Transformation
        self.to_tensor = transforms.ToTensor()
        #self.Resize = transforms.RandomResizedCrop(224)
        #self.Flip = transforms.RandomHorizontalFlip()
        #read the csv
        self.data_info = testFileName
        #get the name
        self.image_arr = np.asarray(self.data_info)
        #get the length
        self.data_length = len(self.data_info)

    def __getitem__(self, index):
        #get the file name from df
        single_image_name = self.image_arr[index]
        #get the data
        test_path = "cvdl2019/validation_images/" + single_image_name
        img_data = Image.open(test_path)
        #Transformation
        #imgtmp = self.Resize(img_data)
        #imgflip = self.Flip(imgtmp)
        #turn the image into tensor
        img_as_tensor = self.to_tensor(img_data).to(device)
        #GPU

        #get the label
```

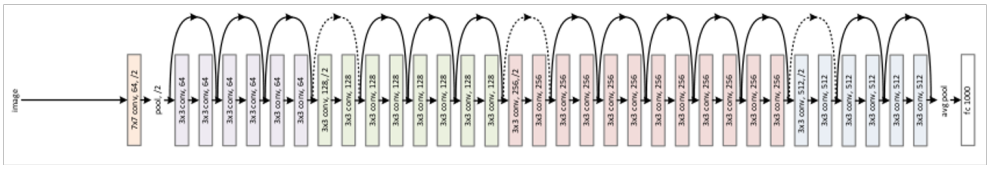
```
        #single_image_label = self.label_arr[index]

        return (img_as_tensor, single_image_name)
    def __len__(self):
        return self.data_length
```

为了提高模型的泛化能力，首先对数据进行**数据增强**。是次作业中通过对训练时每个batch的数据进行 随机剪裁、翻转、亮度增强，可以看作对数据加入了一定的噪声。

```
#Transformation
        self.to_tensor = transforms.ToTensor()
        self.Resize = transforms.RandomResizedCrop(224)
        self.Flip = transforms.RandomHorizontalFlip()
        self.jittering = transforms.ColorJitter(brightness=0.5)
```

迁移学习

本次将使用在Imagenet上训练好的Resnet-50模型作为预训练模型，在Finetune步骤把最后的全连接层输出改为80。



可以看出，Resnet50的主要结构还是基于CNN，但是加入了比较关键的Residual Block，也可以理解为Short-cut，将前几层的输出直接链接到后面。实际训练中，每层其实都做了Batch-Normalization，使得模型的分布较为集中，此外，激活函数使用的是ReLu，损失函数使用的是交叉熵损失。

为了使用预训练好的模型，我们需要写一个训练用的函数。

```
def train_model(model, criterion,optimizer,scheduler, num_epoch = 25):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epoch):
        print('Epoch {}/{}'.format(epoch, num_epoch - 1))
        print('*'*10)

        #each stage have training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
```

```
                scheduler.step()
                model.train()
            else:
                model.eval()
            running_loss = 0.0
            running_corrects = 0

            #Iterate
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.cuda()
                labels = labels.cuda()
                optimizer.zero_grad()

                #forward
                with torch.set_grad_enabled(phase == 'train'):
                    inputs = inputs.cuda()
                    outputs = model(inputs)
                    _, preds = torch.max(outputs,1)
                    loss = criterion(outputs,labels)

                    #back
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()
                #stat
                running_loss += loss.item() *inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            epoch_loss = running_loss / datasetSize[phase]
            epoch_acc = running_corrects.double() / datasetSize[phase]
            print('{} Loss: {:.4f} Acc: {:.4f}'.format(phase, epoch_loss,
epoch_acc))

            #deep copy the model
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())
        print()
    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best val Acc: {:4f}'.format(best_acc))

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model
```

再者就是Finetune模型。

```
#resnet Finetuning
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, 80)
```

定义损失函数、优化器、以及控制学习率的Scheduler（每7个epoch*0.1倍，初始为0.01）

```
criterion = nn.CrossEntropyLoss()
# Observe that all parameters are being optimized
optzer = optim.Adam(model_ft.parameters(),lr=0.01)
# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optzer, step_size=7, gamma=0.1)
```

接下来就可以开始训练了

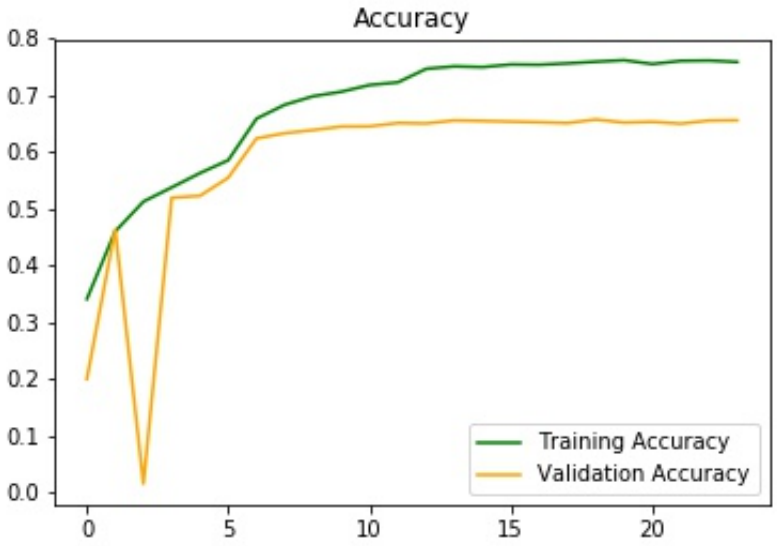
```
model_ft = model_ft.to(device)
model_ft = train_model(model_ft, criterion, optzer,exp_lr_scheduler,
                       num_epoch = 4)
```

输出结果

我们可以用训练好的模型对数据进行预测，并保存为预测文件。

```
count = 0
model_ft.eval()
test_pred = torch.LongTensor()
for i, data in enumerate(testloader):
    data = Variable(data[0], volatile=True)
    if torch.cuda.is_available():
        data = data.cuda()
    output = model_ft(data)
    pred = output.cpu().data.max(1, keepdim=True)[1]
    test_pred = torch.cat((test_pred, pred), dim=0)
tp = list(test_pred[:,0].numpy())
c = {'Id':testFileName,'Category':tp}
ans_df = pd.DataFrame(c)
ans_df.head()
ans_df.to_csv('submission_inc.csv',index=False)
```

模型评估



一共训练了25个Epoch，batch大小为102，可见模型的准确率在5个epoch之后开始收敛，达到了约65%准确率。

可改进的空间

- 数据增强可以做的更仔细，例如数据白化（归一化）、灰度处理
- 由于中途误触GCP中的AutoML，使得2000多人民币的额度瞬间用完，使得失去了很多训练时间，最后千辛万苦找到了新的一张信用卡才能重新配置环境，下次一定不能乱按按钮。
- 可使用不同的模型，如VGG、Inception_v3、InceptionResnet_v2进行训练
- Finetune步骤可以多增加几层并加以训练