烟台大学课程设计

课程名称： 机器学习

课程设计题目： 基于Resnet50的图像分类方法

指导老师姓名： 刘志中

设计时间： 2021.12.26

课程设计成绩评定

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **评定内容** | **评定指标** | | **分值** | **成 绩** |
| **工作态度** | **态度认真；严谨务实； 不旷课；不迟到；不早退** | | 15 |  |
| **论文格式** | **语言通畅；格式规范；结构合理；内容完整** | | 25 |  |
| **论文质量** | **内容全面；表述清晰；模型正确；严谨合理；技术用语准确；符号统一；书写工整规范；图表完备、整洁、正确；** | | 30 |  |
| **工作创新** | **论文中有一定的创新意识、有自己的贡献；**  **对前人工作有改进或突破，或有独特见解；** | | 10 |  |
| **量与难度** | **工作量饱满，工作难度大** | | 20 |  |
| **总成绩：** | | | | |
| **指导老师签字：** | | **论文完成日期：** | | |
| **评定日期：** | | |

目录

[摘要 1](#_Toc14633)

[一、 课程论文目的 1](#_Toc30432)

[二、 总体思路 1](#_Toc19784)

[三、 实现过程 1](#_Toc24024)

[3.1数据预处理 1](#_Toc27013)

[3.2构建网络 3](#_Toc25687)

[3.3网络训练与测试 3](#_Toc22659)

[四、 总结 6](#_Toc21383)

[参考文献 7](#_Toc2962)

[附录 7](#_Toc6426)

摘要

本文的主要工作是使用深度学习模型对酒店图像进行分类，能够根据酒店图片给出对应的酒店类别。由于数据量非常小，因此我们使用了微调，在预训练的Resnet50模型上进行训练，并取得了很好的的准确率。

1. 课程论文目的

通过本次课程设计，了解学习掌握深度学习完整的流程，包括数据预处理，数据转换，网络构建，网络训练和测试。其中数据预处理是本次课程设计的主要内容，实际日常生产生活中并没有已经处理好的tensor格式的数据集，往往需要自己对数据进行一些提取和预处理，然后转换成pytorch可以接受到tensor张量。

1. 总体思路

首先需要把文档中的图片提取出来，然后将对每一类图片建立一个文件夹，每个文件夹的名字是该酒店的名称。其次使用torchvisiom.datasets中的ImageFolder类，就可以对本地图像文件进行加载，使用torchvision的transforms将图片进行裁剪，然后转换成tensor，然后进行标准化，使用torch.utils.data中的DataLoader生成数据加载器。

第二步是使用torchvision.models中的resnet50预训练模型对模型进行微调，并且训练和测试的损失和精度绘制出来。

1. 实现过程

**3.1数据预处理**

图片提取过程：

将word另存为html文件即可到处文档中的所有图片，并将所有图片复制到img文件夹中；

使用如下正则表达式提取全部酒店名称：

酒店名称[:：\s]\*[\u4E00-\u9FA5A-Za-z0-9\_]+[•\s]\*[\u4E00-\u9FA5A-Za-z0-9\_]\*[（）()]?[\u4E00-\u9FA5A-Za-z0-9\_]+[（）()]?[\n]+

将所有名称保存到name.txt文件

利用python os、shutil库为每个酒店建立文件夹并将图片放入：

这里需要注意一个细节，图片到处的时候有两份，一个jpg一个png，这里我们选择使用文件小一些的jpg，他们的编号是奇数。

with open("name.txt", encoding='utf-8') as f:

res = []

for i in f:

if i == '\n':

continue

res.append(i.strip('\n'))

print(len(res))

imgpath = './img'

# 取出所有文件名字

files = os.listdir(imgpath)

targetNames = []

for i in range(len(files)):

#只复制jpg

if i % 2 != 0:

targetNames.append(imgpath + '/' + files[i])

os.mkdir('./data')

for i in range(len(res)):

folderName = './res/' + res[i]

print(folderName)

os.mkdir(folderName)

copy(targetNames[i], folderName)

使用ImageFolder加载数据并用DataLoader加载:

# 使用三个RGB通道的均值和标准偏差，以标准化每个通道

normalize = transforms.Normalize([0.485, 0.456, 0.406],

[0.229, 0.224, 0.225])

train\_augs = transforms.Compose([

transforms.RandomResizedCrop(224),

transforms.RandomHorizontalFlip(),

transforms.ToTensor(), normalize])

val\_augs = transforms.Compose([

transforms.Resize(256),

transforms.CenterCrop(224),

transforms.ToTensor(), normalize])

train\_iter = DataLoader(ImageFolder(data\_dir,

transform=train\_augs),

batch\_size=batch\_size, shuffle=True)

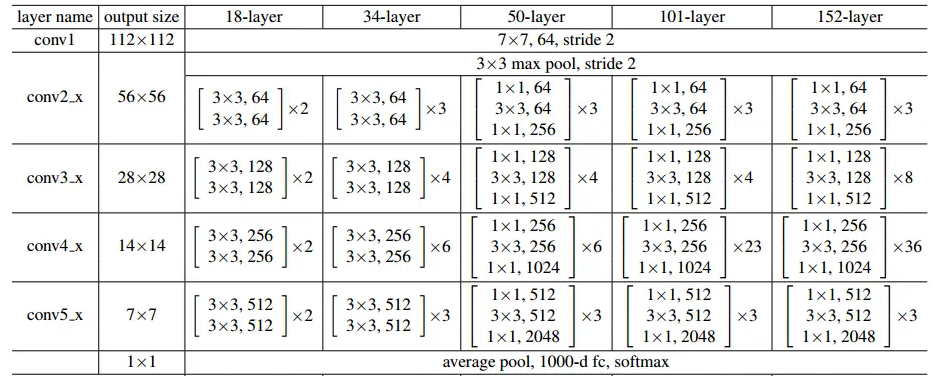
val\_iter = DataLoader(ImageFolder(data\_dir,

transform=val\_augs),

batch\_size=batch\_size)

**3.2构建网络**

resnet参数信息：



使用resnet50预训练模型并根据酒店类别数量重置最后的MLP分类层。

net = models.resnet50(pretrained=True)

# 冻结前面的特征提取层

for param in net.parameters():

param.requires\_grad = False

# 重置最后的MLP分类层

net.fc = nn.Linear(net.fc.in\_features, 61)

# 初始化

nn.init.xavier\_uniform\_(net.fc.weight)

**3.3网络训练与测试**

由于本次课程设计数据量非常小，所以我们冻结了resnet50前面的特征提取层参数，只训练最后的MLP分类层，并且使用了nn.DataParallel使用GPU训练，这个类可以自动将模型和数据放到GPU上；使用Visdom库对训练和验证的结果进行可视化。

主要训练和验证代码:

# 每一个训练epoch

def train\_epoch(model, train\_loader, optimizer, loss\_fn, epoch):

size = len(train\_loader.dataset)

num\_batches = len(train\_loader)

model.train() # 将模型设置为训练模式

train\_loss, train\_correct = 0, 0

for batch\_idx, (X, y) in enumerate(train\_loader):

pred = model(X)

y = y.to(pred.device)

loss = loss\_fn(pred, y)

#print(X.shape,y.shape, pred.shape)

# 梯度清零， 反向传播，更新网络参数

optimizer.zero\_grad()

loss.backward()

optimizer.step()

# 记录损失与正确率

train\_loss += loss.item()

train\_correct += (pred.argmax(1) == y).type(torch.float).sum().item()

# 每 2批输出一次

if batch\_idx % 2 == 0:

print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(

epoch, batch\_idx \* len(X), size,

100. \* batch\_idx / num\_batches, loss.item()))

return train\_loss / num\_batches, train\_correct / size

# 验证epoch

def val\_epoch(model, val\_loader, loss\_fn):

size = len(val\_loader.dataset)

num\_batches = len(val\_loader)

model.eval() # 设为评估模式

test\_loss, test\_correct = 0, 0

# 不记录梯度，节省内存

with torch.no\_grad():

for X, y in val\_loader:

pred = model(X)

y = y.to(pred.device)

loss = loss\_fn(pred, y)

test\_loss += loss.item()

test\_correct += (pred.argmax(1) ==

y).type(torch.float).sum().item()

test\_loss /= num\_batches

print('\nVal set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(

test\_loss, test\_correct, size, 100. \* test\_correct / size))

return test\_loss, test\_correct / size

# 训练

def trianModel(net, trainer, train\_iter, val\_iter, num\_epoch, loss\_fn):

net = nn.DataParallel(net)

# 记录损失和正确率

train\_loss, train\_accuracy = [], []

val\_loss, val\_accuracy = [], []

for epoch in range(1, num\_epoch + 1):

print(f"Epoch {epoch}\n-------------------------------")

a, b = train\_epoch(net, train\_iter, trainer, loss\_fn, epoch)

train\_loss.append(a)

train\_accuracy.append(b)

c, d = val\_epoch(net, val\_iter, loss\_fn)

val\_loss.append(c)

val\_accuracy.append(d)

x = np.asarray(range(len(train\_loss)))

x = np.column\_stack((x, x))

y1 = np.column\_stack((train\_loss, train\_accuracy))

viz.line(y1, x, win='train', opts=dict(title='Train\_loss&Train\_acc',

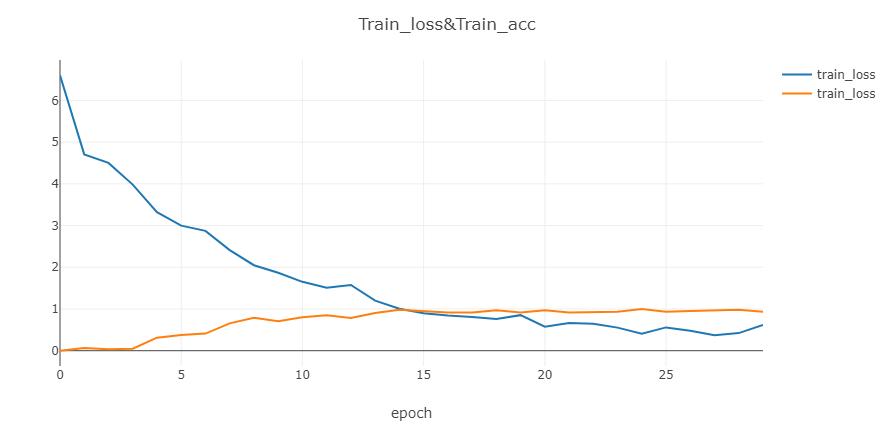
xlabel='epoch', legend=['train\_loss','train\_loss']))

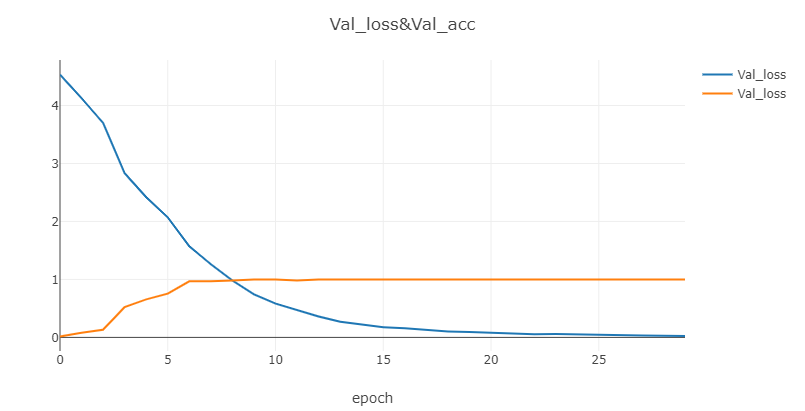
y2 = np.column\_stack((val\_loss, val\_accuracy))

viz.line(y2, x, win='val', opts=dict(title='Val\_loss&Val\_acc',

xlabel='epoch', legend=['Val\_loss','Val\_loss']))

训练和测试结果:





1. 总结

通过本次课程设计，对一个完整的深度学习模型应用日常生活中的基本流程有了一个初步的了解，能够运用自己所学的知道对日常生产生活中的一些问题进行解决，对数据预处理的基本方法有了进一步的了解与认识，在进行数据处理的过程也学习了一些基本的正则表达式语法，可以利用正则表达式的强大功能提取自己需要的文本信息。

虽然在本次课程设计中对一个实际深度学习模型基本流程有了一些基本认识，但是实际工业界模型落地考虑的因素远远不止这些，首先我们缺失数据采集与获取步骤，可以使用爬虫等技术去采集自己想要的数据或者一些大型企业有自己的大数据库，但是在进行训练前需要对数据进行脱敏操作，就是去除数据中涉及用户隐私的内容，因此数据的获取与采集不仅仅涉及到技术问题还涉及到用户隐私问题；

其次我们的数据预处理其实过于简单，实际大型模型需要利用各种大数据技术对数据进行一个有效的处理包括脱敏操作，而且在海量数据面前，还需要对数据进行一个初步的选择，海量数据不可能都加载到内存中，这里还涉及到如何分批加载磁盘中的数据，提升I/O效率，尽量避免训练时要等待数据集从磁盘上加载的过程。

另一个是说训练过程，实际应用的模型通常非常大，因此不可避免的要使用分布式训练技术，由于本次课设模型不大，因此我们只是简单使用了单卡GPU加速，并没有使用分布式训练，也没有分布式训练所需要的硬件资源。

另一个是模型部署，我们没有这个环节，模型可能要运行在移动端或网页端或PC端，都需要针对具体设备进行优化，使用合理的部署，针对移动设备计算资源小，还要进行模型蒸馏操作，减少模型参数，加快模型推理过程。

除了关于模型落地基本流程的一些了解，还对resnet模型有了一些了解，resnet模型的重要思想是残差连接，这个思想直接加速了模型的训练速度，而且使得模型的深度可以继续加深，残差连接的思想非常简单，就是把前面的输出直接连接到后面，这样可以使得学习路径多了一个快速通道，可以跳过中间的一些层数，这样可以在模型较深的时候，减免梯度消失的现象，加快了模型的训练过程，同时也可以保证即便在模型很深的情况下，也能保证只少存在一个浅层网络模型，在浅层模型的保证下，会尝试去学习深层的权重，保证了模型的训练效果。

参考文献

李沐，《动手学深度学习(pytorch版)》

附录

import torch

from torch import nn

from torch.utils.data import DataLoader

import os

from torchvision import transforms, models

from torchvision.datasets import ImageFolder

import matplotlib.pyplot as plt

from visdom import Visdom

import numpy as np

from shutil import copy

viz = Visdom()

with open("name.txt", encoding='utf-8') as f:

res = []

for i in f:

if i == '\n':

continue

res.append(i.strip('\n'))

print(len(res))

imgpath = './img'

# 取出所有文件名字

files = os.listdir(imgpath)

targetNames = []

for i in range(len(files)):

#只复制jpg

if i % 2 != 0:

targetNames.append(imgpath + '/' + files[i])

os.mkdir('./data')

for i in range(len(res)):

folderName = './res/' + res[i]

print(folderName)

os.mkdir(folderName)

copy(targetNames[i], folderName)

# 使用三个RGB通道的均值和标准偏差，以标准化每个通道

normalize = transforms.Normalize([0.485, 0.456, 0.406],

[0.229, 0.224, 0.225])

train\_augs = transforms.Compose([

transforms.RandomResizedCrop(224),

transforms.RandomHorizontalFlip(),

transforms.ToTensor(), normalize])

val\_augs = transforms.Compose([

transforms.Resize(256),

transforms.CenterCrop(224),

transforms.ToTensor(), normalize])

# 每一个训练epoch

def train\_epoch(model, train\_loader, optimizer, loss\_fn, epoch):

size = len(train\_loader.dataset)

num\_batches = len(train\_loader)

model.train() # 将模型设置为训练模式

train\_loss, train\_correct = 0, 0

for batch\_idx, (X, y) in enumerate(train\_loader):

pred = model(X)

y = y.to(pred.device)

loss = loss\_fn(pred, y)

#print(X.shape,y.shape, pred.shape)

# 梯度清零， 反向传播，更新网络参数

optimizer.zero\_grad()

loss.backward()

optimizer.step()

# 记录损失与正确率

train\_loss += loss.item()

train\_correct += (pred.argmax(1) == y).type(torch.float).sum().item()

# 每 2批输出一次

if batch\_idx % 2 == 0:

print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(

epoch, batch\_idx \* len(X), size,

100. \* batch\_idx / num\_batches, loss.item()))

return train\_loss / num\_batches, train\_correct / size

# 验证epoch

def val\_epoch(model, val\_loader, loss\_fn):

size = len(val\_loader.dataset)

num\_batches = len(val\_loader)

model.eval() # 设为评估模式

test\_loss, test\_correct = 0, 0

# 不记录梯度，节省内存

with torch.no\_grad():

for X, y in val\_loader:

pred = model(X)

y = y.to(pred.device)

loss = loss\_fn(pred, y)

test\_loss += loss.item()

test\_correct += (pred.argmax(1) ==

y).type(torch.float).sum().item()

test\_loss /= num\_batches

print('\nVal set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(

test\_loss, test\_correct, size, 100. \* test\_correct / size))

return test\_loss, test\_correct / size

# 训练

def trianModel(net, trainer, train\_iter, val\_iter, num\_epoch, loss\_fn):

net = nn.DataParallel(net)

# 记录损失和正确率

train\_loss, train\_accuracy = [], []

val\_loss, val\_accuracy = [], []

for epoch in range(1, num\_epoch + 1):

print(f"Epoch {epoch}\n-------------------------------")

a, b = train\_epoch(net, train\_iter, trainer, loss\_fn, epoch)

train\_loss.append(a)

train\_accuracy.append(b)

c, d = val\_epoch(net, val\_iter, loss\_fn)

val\_loss.append(c)

val\_accuracy.append(d)

x = np.asarray(range(len(train\_loss)))

x = np.column\_stack((x, x))

y1 = np.column\_stack((train\_loss, train\_accuracy))

viz.line(y1, x, win='train', opts=dict(title='Train\_loss&Train\_acc',

xlabel='epoch', legend=['train\_loss','train\_loss']))

y2 = np.column\_stack((val\_loss, val\_accuracy))

viz.line(y2, x, win='val', opts=dict(title='Val\_loss&Val\_acc',

xlabel='epoch', legend=['Val\_loss','Val\_loss']))

# 如果 `param\_group=True`，输出层中的模型参数将使用十倍的学习率

def train\_fine\_tuning(net, learning\_rate, batch\_size=16, num\_epoch=10, data\_dir='./data',

param\_group=False):

train\_iter = DataLoader(ImageFolder(data\_dir,

transform=train\_augs),

batch\_size=batch\_size, shuffle=True)

val\_iter = DataLoader(ImageFolder(data\_dir,

transform=val\_augs),

batch\_size=batch\_size)

loss = nn.CrossEntropyLoss()

if param\_group:

params\_1x = [

param for name, param in net.named\_parameters()

if name not in ["fc.weight", "fc.bias"]]

trainer = torch.optim.Adam([{

'params': params\_1x}, {

'params': net.fc.parameters(),

'lr': learning\_rate \* 10}], lr=learning\_rate,

weight\_decay=0.001)

else:

trainer = torch.optim.Adam(net.parameters(), lr=learning\_rate)

trianModel(net, trainer, train\_iter, val\_iter, num\_epoch, loss)

net = models.resnet50(pretrained=True)

# 冻结前面的特征提取层

for param in net.parameters():

param.requires\_grad = False

# 重置最后的MLP分类层

net.fc = nn.Linear(net.fc.in\_features, 61)

# 初始化

nn.init.xavier\_uniform\_(net.fc.weight)

train\_fine\_tuning(net, 0.002, num\_epoch=30)