

**研究生课程作业报告**

**作业名称： 基于ResNet50实现多目标美味蛋糕图像分类**

**学 院： 信息科学技术学院**

**学 系： 计算机科学系**

**专 业： 计算机软件与理论**

**课程名称： 高级计算机系统结构**

**学生姓名： 唐杰烽**

**学生学号： 202234061004**

**授课教师： 陈子良、官全龙**

**2023年 8 月 20 日**

1. **项目简介**

本项目是一个使用ResNet50实现图像分类的任务，但是不同于传统分类，本项目是一个多目标分类任务，传统分类任务需要判别一张图片的单一类别，例如判别是猫或是狗，但是如果我们有一种需求就是如果一张图片中既有狗又有猫，就需要将两个类别同时判为1，本项目是判断一个蛋糕中是否同时有奶油、水果、撒料，是一个多目标分类任务。

1. **数据集介绍**

本项目所采用的数据集是Kaggle上公开的数据集。下载链接：

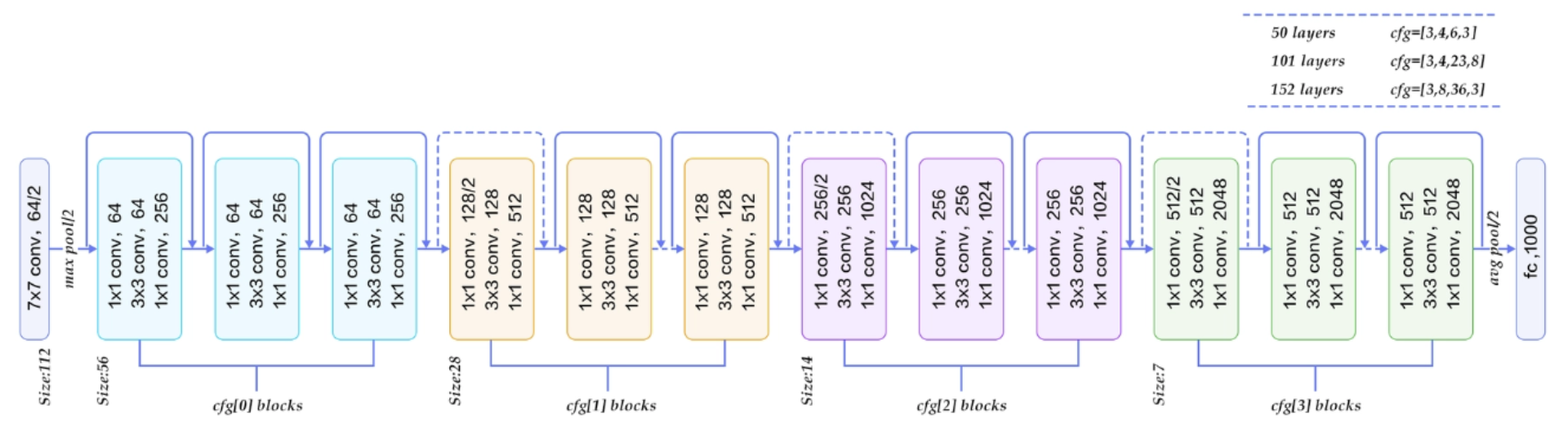
[**https://www.kaggle.com/datasets/rajkumarl/cakey-bakey**](https://www.kaggle.com/datasets/rajkumarl/cakey-bakey)

这是一个包含蛋糕图像及其多目标类的集合。其中：

Cake\_annotated.csv包含以下条目：

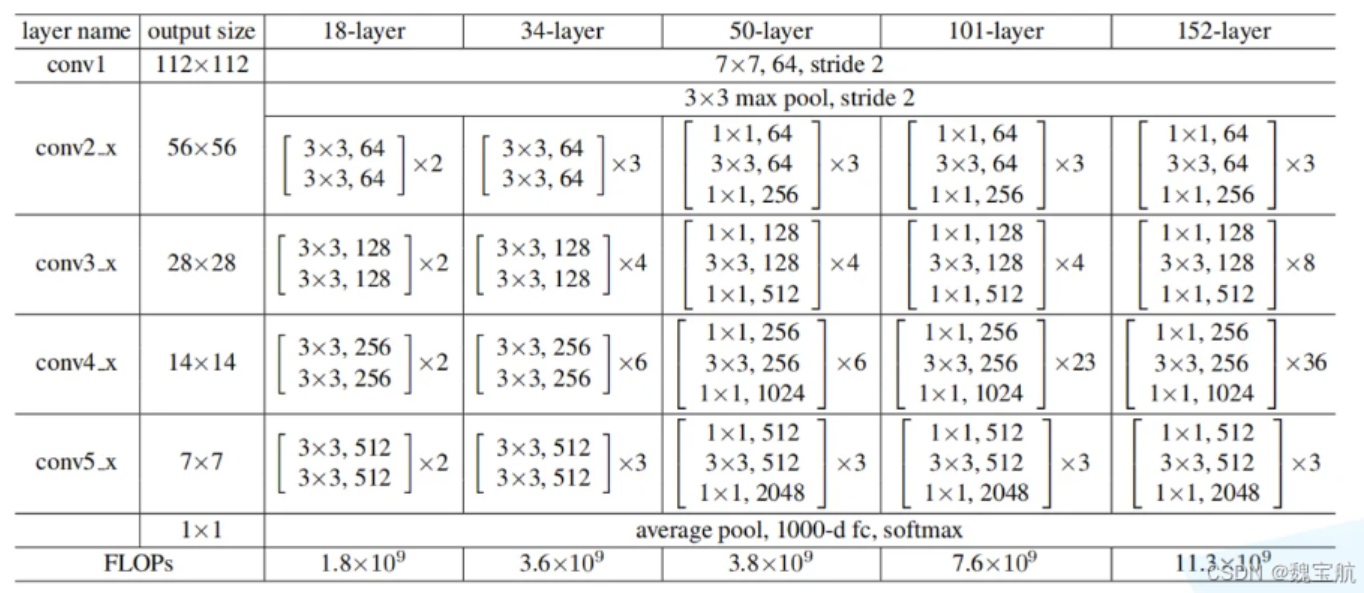
* Filename (str: uniquevalues)- JPG图像文件名
* 奶油(str:[‘yes’，‘no’]- yes表示奶油存在，no表示不存在。
* 水果(str:[‘yes’，‘no’]- yes表示水果存在，no表示不存在。
* spille\_toppings (str:[‘yes’，‘no’]- yes表示有洒上的浇头，如糖果、彩虹球，no表示没有。

1. **ResNet50模型介绍**



ResNet作者何恺明，CVPR2016最佳论文奖

ResNet解决了深度CNN模型难训练的问题，从图中可以看到2014年的VGG只有19层，而2015年的ResNet多达152层，在网络深度方面完全不是一个量级。ResNet还有架构上的trick，这才使得网络的深度发挥出作用，这个trick就是残差学习（Residual Learning）。



1. **加载ResNet50模型**

为了方便及提高模型的泛化性，我们加载Pytorch中已经训练好的ResNet模型，需要一定时间下载训练好的模型权重。

由于预训练的模型和我们的任务需要不同，因此我们将要修改最后一层的全连接层，将输出维度修改为我们任务要求中的二分类。

需要注意的是，需要同时冻结其他层的参数，防止训练过程中将其进行改动，然后进行训练微调最后一层即可。

|  |
| --- |
| # 加载ResNet50模型  model = torchvision.models.resnet50(pretrained=True)  # 加载预训练好的ResNet50模型  # 冻结模型参数  for param in model.parameters():  param.requires\_grad = False  # 修改最后一层的全连接层  model.fc = nn.Sequential(  nn.Linear(model.fc.in\_features, 3),  nn.Sigmoid())  # 将模型加载到cpu中  model = model.to('cpu')  criterion = MyLoss() # 损失函数  optimizer = torch.optim.Adam(model.parameters(), lr=0.01) # 优化器 |

1. **加载数据集**

我们所有的图片都是在一个文件下，每个图像的标签含在一个csv文件中，因此不能利用Pytorch中的ImageFolder进行加载，需要自己重写DataSet类，实现读写数据。

重写DataSet类需要重写3个方法：

\_\_init\_\_:该方法主要进行一些参数初始化工作，定义一些路径或者变量。

\_\_getitem\_\_:该方法用于加载数据，用于读取每一条数据，它有一个参数idx，就是对应的索引，索引从0开始。由于我们的图片是从001.jpg到230.jpg，因此可以利用这个索引依次读取文件夹中的所有图片，然后从标签csv中读取它对应的行拿到对应的标签，然后返回即可。

\_\_len\_\_:返回整个数据集的大小。

|  |
| --- |
| # 加载数据集，自己重写DataSet类  class dataset(Dataset):  # image\_dir为数据目录，label\_file，为标签文件  def \_\_init\_\_(self, image\_dir, label\_file, transform=None):  self.image\_dir = image\_dir # 图像文件所在路径  self.label\_file = pd.read\_csv(label\_file) # 图像对应的标签文件  self.transform = transform # 数据转换操作  # 加载每一项数据  def \_\_getitem\_\_(self, idx):  # 每个图片，其中idx为数据索引  img\_name = os.path.join(self.image\_dir, '%.3d.jpg' % (idx + 1)) # 加载每一张照片  image = Image.open(img\_name)  # 对应标签  labels = (self.label\_file[['cream', 'fruits', 'sprinkle\_toppings']] == 'yes').astype(int).values[idx, :]  if self.transform:  image = self.transform(image)  # 返回一张照片，一个标签  return image, labels  # 数据集大小  def \_\_len\_\_(self):  return (len(self.label\_file)) |

1. **形成迭代器**

|  |
| --- |
| # 数据转换  data\_transform = {  训练中的数据增强和归一化  'train': transforms.Compose([  transforms.RandomResizedCrop(224), # 随机裁剪  transforms.RandomHorizontalFlip(), # 左右翻转  transforms.ToTensor(),  transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]) # 均值方差归一化  ])  }  # 形成训练集  train\_dataset = dataset(image\_dir, label\_file, data\_transform['train'])  # 形成迭代器  train\_loader = torch.utils.data.DataLoader(train\_dataset,  batch\_size,  True)  print('using {} images for training.'.format(len(train\_dataset))) |

1. **定义损失函数**

本任务是一个多目标分类任务，因此我们的损失函数不同于传统的单一图像分类，比如一张照片上既有cream又有fruit，但是没有spille\_toppings，那么这张图片对应的标签为[1,1,0]，所以网络的最后输出层要输出一个3维向量用于分类，这3个维度中的每个维度就是一个典型的二分类问题，每列都为“yes or no”的问题，对于二分类常用的就是Binary-Cross-Entropy，因此本任务可以看成是3个二分类任务，针对每一列计算损失然后汇总即可。

|  |
| --- |
| # 自定义损失函数，需要在forward中定义过程  class MyLoss(nn.Module):  def \_\_init\_\_(self):  super(MyLoss, self).\_\_init\_\_()  # 参数为传入的预测值和真实值，返回所有样本的损失值，自己只需定义计算过程，反向传播PyTroch会自动记录，最好用PyTorch进行计算  def forward(self, pred, label):  # pred：[32, 3] label：[32, 3] 第一维度是样本数  # 由于是二分类，使用BCE损失  return F.binary\_cross\_entropy(pred, label) |

1. 训练过程

|  |
| --- |
| train epoch[1/10] loss:1.459: 100%|██████████| 8/8 [00:26<00:00, 3.29s/it]  0%| | 0/8 [00:00<?, ?it/s]【EPOCH: 】1  训练损失为11.114460706710815  cream训练精度为52.60%  fruits训练精度为51.73%  sprinkle\_toppings训练精度为55.21%  train epoch[2/10] loss:0.645: 100%|██████████| 8/8 [00:23<00:00, 2.95s/it]  0%| | 0/8 [00:00<?, ?it/s]【EPOCH: 】2  训练损失为5.296277761459351  cream训练精度为66.95%  fruits训练精度为78.26%  sprinkle\_toppings训练精度为59.56%  train epoch[3/10] loss:0.562: 100%|██████████| 8/8 [00:24<00:00, 3.07s/it]  【EPOCH: 】3  训练损失为4.1593363881111145  cream训练精度为72.60%  fruits训练精度为80.86%  sprinkle\_toppings训练精度为70.43%  train epoch[4/10] loss:0.761: 100%|██████████| 8/8 [00:25<00:00, 3.13s/it]  【EPOCH: 】4  训练损失为4.472545355558395  cream训练精度为74.34%  fruits训练精度为82.17%  sprinkle\_toppings训练精度为62.17%  train epoch[5/10] loss:0.588: 100%|██████████| 8/8 [00:27<00:00, 3.45s/it]  【EPOCH: 】5  训练损失为4.48739156126976  cream训练精度为77.39%  fruits训练精度为79.13%  sprinkle\_toppings训练精度为65.21%  train epoch[6/10] loss:0.410: 100%|██████████| 8/8 [00:24<00:00, 3.03s/it]  0%| | 0/8 [00:00<?, ?it/s]【EPOCH: 】6  训练损失为4.149786680936813  cream训练精度为74.34%  fruits训练精度为79.13%  sprinkle\_toppings训练精度为70.86%  train epoch[7/10] loss:0.466: 100%|██████████| 8/8 [00:25<00:00, 3.13s/it]  0%| | 0/8 [00:00<?, ?it/s]【EPOCH: 】7  训练损失为3.7354097962379456  cream训练精度为75.21%  fruits训练精度为84.34%  sprinkle\_toppings训练精度为75.65%  train epoch[8/10] loss:0.440: 100%|██████████| 8/8 [00:25<00:00, 3.21s/it]  【EPOCH: 】8  训练损失为3.110919088125229  cream训练精度为77.82%  fruits训练精度为92.17%  sprinkle\_toppings训练精度为81.73%  train epoch[9/10] loss:0.488: 100%|██████████| 8/8 [00:25<00:00, 3.21s/it]  【EPOCH: 】9  训练损失为3.6831241250038147  cream训练精度为83.47%  fruits训练精度为84.78%  sprinkle\_toppings训练精度为70.43%  train epoch[10/10] loss:0.231: 100%|██████████| 8/8 [00:25<00:00, 3.17s/it]  【EPOCH: 】10  训练损失为3.748447984457016  cream训练精度为81.73%  fruits训练精度为83.04%  sprinkle\_toppings训练精度为67.39%  Finished Training |

1. 图像测试

模型训练结束后，加载训练好的权重，然后上传一张图片进行测试，下图（Picture 157）的预测结果是：



【预测cream结果分类】：no

【预测fruits结果分类】：yes

【预测sprinkle\_toppings结果分类】：no

下图（Picture 144）的预测结果是：



【预测cream结果分类】：no

【预测fruits结果分类】：no

【预测sprinkle\_toppings结果分类】：yes

下图（Picture 165）的预测结果是：

A group of muffins on a wood surface

Description automatically generated

【预测cream结果分类】：no

【预测fruits结果分类】：no

【预测sprinkle\_toppings结果分类】：no

**对照传入的测试照片与程序输出的预测结果可知，预测精度较高。**

1. 完整源码

|  |
| --- |
| import torchvision  from torch import nn  import numpy as np  import pandas as pd  import os  import json  import pickle  import torch  import torch.optim as optim  import torch.nn.functional as F  from torchvision import transforms, datasets  import torchvision.models as models  from torch.utils.data import Dataset, DataLoader  from tqdm import tqdm  from PIL import Image  import matplotlib.pyplot as plt  # 加载数据集，自己重写DataSet类  class dataset(Dataset):  # image\_dir为数据目录，label\_file，为标签文件  def \_\_init\_\_(self, image\_dir, label\_file, transform=None):  self.image\_dir = image\_dir # 图像文件所在路径  self.label\_file = pd.read\_csv(label\_file) # 图像对应的标签文件  self.transform = transform # 数据转换操作  # 加载每一项数据  def \_\_getitem\_\_(self, idx):  # 每个图片，其中idx为数据索引  img\_name = os.path.join(self.image\_dir, '%.3d.jpg' % (idx + 1)) # 加载每一张照片  image = Image.open(img\_name)  # 对应标签  labels = (self.label\_file[['cream', 'fruits', 'sprinkle\_toppings']] == 'yes').astype(int).values[idx, :]  if self.transform:  image = self.transform(image)  # 返回一张照片，一个标签  return image, labels  # 数据集大小  def \_\_len\_\_(self):  return (len(self.label\_file))  image\_dir = './data/images' # 数据集路径  label\_file = './data/cake\_annotated.csv' # 标签位置  epochs = 10  lr = 0.003  batch\_size = 32  save\_path = './best\_model.pkl'  device = torch.device('cuda:0' if torch.cuda.is\_available() else 'cpu')  # 数据转换  data\_transform = {  # 训练中的数据增强和归一化  'train': transforms.Compose([  transforms.RandomResizedCrop(224), # 随机裁剪  transforms.RandomHorizontalFlip(), # 左右翻转  transforms.ToTensor(),  transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]) # 均值方差归一化  ])  }  # 形成训练集  train\_dataset = dataset(image\_dir, label\_file, data\_transform['train'])  # 形成迭代器  train\_loader = torch.utils.data.DataLoader(train\_dataset,  batch\_size,  True)  print('using {} images for training.'.format(len(train\_dataset)))  # 自定义损失函数，需要在forward中定义过程  class MyLoss(nn.Module):  def \_\_init\_\_(self):  super(MyLoss, self).\_\_init\_\_()  # 参数为传入的预测值和真实值，返回所有样本的损失值，自己只需定义计算过程，反向传播PyTroch会自动记录，最好用PyTorch进行计算  def forward(self, pred, label):  # pred：[32, 3] label：[32, 3] 第一维度是样本数  # 由于是二分类，使用BCE损失  return F.binary\_cross\_entropy(pred, label)  # 加载ResNet50模型  model = torchvision.models.resnet50(pretrained=True) # 加载预训练好的ResNet50模型  # 冻结模型参数  for param in model.parameters():  param.requires\_grad = False  # 修改最后一层的全连接层  model.fc = nn.Sequential(  nn.Linear(model.fc.in\_features, 3),  nn.Sigmoid())  # 将模型加载到cpu中  model = model.to('cpu')  criterion = MyLoss() # 损失函数  optimizer = torch.optim.Adam(model.parameters(), lr=0.01) # 优化器  # 模型训练  best\_acc = 0 # 最优精确率  best\_model = None # 最优模型参数  for epoch in range(epochs):  model.train()  running\_loss = 0 # 损失  epoch\_acc\_cream = 0 # 每个epoch的准确率  epoch\_acc\_fruits = 0 # 每个epoch的准确率  epoch\_acc\_sprinkle\_toppings = 0 # 每个epoch的准确率  epoch\_acc\_count\_cream = 0 # 每个epoch训练的样本数  epoch\_acc\_count\_fruits = 0 # 每个epoch训练的样本数  epoch\_acc\_count\_sprinkle\_toppings = 0 # 每个epoch训练的样本数  train\_count = 0 # 用于计算总的样本数，方便求准确率  train\_bar = tqdm(train\_loader)  for data in train\_bar:  images, labels = data  optimizer.zero\_grad()  output = model(images.to(device))  loss = criterion(output, labels.float().to(device))  loss.backward()  optimizer.step()  running\_loss += loss.item()  train\_bar.desc = "train epoch[{}/{}] loss:{:.3f}".format(epoch + 1, epochs,loss)  # 计算每个epoch正确的个数  epoch\_acc\_count\_cream += ((output > 0.5).int() == labels.float().to(device)).sum(axis=0)[0]  epoch\_acc\_count\_fruits += ((output > 0.5).int() == labels.float().to(device)).sum(axis=0)[1]  epoch\_acc\_count\_sprinkle\_toppings += ((output > 0.5).int() == labels.float().to(device)).sum(axis=0)[2]  train\_count += len(images)  # 每个epoch对应的准确率  epoch\_acc\_cream = epoch\_acc\_count\_cream / train\_count  epoch\_acc\_fruits = epoch\_acc\_count\_fruits / train\_count  epoch\_acc\_sprinkle\_toppings = epoch\_acc\_count\_sprinkle\_toppings / train\_count  # 打印信息  print("【EPOCH: 】%s" % str(epoch + 1))  print("训练损失为%s" % str(running\_loss))  print("cream训练精度为%s" % (str(epoch\_acc\_cream.item() \* 100)[:5]) + '%')  print("fruits训练精度为%s" % (str(epoch\_acc\_fruits.item() \* 100)[:5]) + '%')  print("sprinkle\_toppings训练精度为%s" % (str(epoch\_acc\_sprinkle\_toppings.item() \* 100)[:5]) + '%')  if epoch\_acc\_cream > best\_acc:  best\_acc = epoch\_acc\_cream  best\_model = model.state\_dict()  # 在训练结束保存最优的模型参数  if epoch == epochs - 1:  # 保存模型  torch.save(best\_model, save\_path)  print('Finished Training')  # 数据变换  data\_transform = transforms.Compose(  [transforms.Resize(256),  transforms.CenterCrop(224),  transforms.ToTensor(),  transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])])  # 图片路径  img\_path = r'./data/images/165.jpg'  # 打开图像  img = Image.open(img\_path)  # 绘制图像  plt.imshow(transforms.ToTensor()(img).permute(1,2,0))  plt.show()  # 对图像进行变换  img = data\_transform(img)  # 将图像升维，增加batch\_size维度  img = torch.unsqueeze(img, dim=0)  # 获取预测结果  model.eval()  pred = model(img) > 0.5  dic = {False: 'no', True: 'yes'}  # 预测输出  print('【预测cream结果分类】：%s' % dic[pred[0][0].item()])  print('【预测fruits结果分类】：%s' % dic[pred[0][1].item()])  print('【预测sprinkle\_toppings结果分类】：%s' % dic[pred[0][2].item()]) |

**项目报告至此结束。**