华北水利水电大学

North China University of Water Resources and Electric Power

《机器学习与模式识别》
课程设计

学　院　信息工程学院

专　业　人工智能

姓　名　高树林

学　号　**202018526**

指导教师　杨阳蕊、谢昊洋

完成时间　**2022 年 11 月 30 日**

实验报告

一、 实验目的

1. 理解并掌握回归和分类问题的原理
2. 理解并掌握传统机器学习模型的算法原理

二、 实验要求

1. 至少用三种传统机器学习的方法训练模型。
2. 对整个报告设计流程图或思维导图，其能够反映本报告整体的思想内容。
3. 对于三个实验，均需要对模型进行简单的端到端部署，构建一个完整的项目，并利用装饰器和类对代码进行封装演示。
4. 对于分类实验，需要在测试集上进行模型评估和可视化分析。实现通过命令行传递参数。
5. 结合神经网络实验，比较其与传统机器学习模型的优缺点。**（选做）**
6. 提交报告时，文件打包成一个文件夹，不含数据集。保证运行代码的可运行性，必须保证程序的可移植性。
7. 字体优雅、代码需要格式化。报告命名规则为 2088166-乔峰-机器学习与模式识别报告，文件夹命名 2088166-乔峰-机器学习与模式识别。
8. 本报告提交时分电子版和纸质版提交，电子版提交时间截止到 deadline ，纸质版下个学期开学第一周交。

三、 实验环境

该程序开发的环境：Windows10 专业版（Professional）32 位操作系统，内存频率: 1333MHz，内存容量: 8GB 16GB，显存容量: 2GB，Pycharm 2022.2，开发语言为 python 语言，解释器为 python3.8.13。开发过程中使用的工具包机器版本号分别为：joblib==1.1.1 、matplotlib==3.3.4 、numpy==1.19.5 、opencv_python==4.6.0.66、pandas==1.1.5、scikit_learn==1.1.3、torch==1.10.2、torchvision==0.11.3、tqdm==4.64.0。

该程序运行的环境：CPU 主频: 3.0GHz，内存容量: 8GB，显存容量: 6GB，硬盘容量: 128GB，运行的平台为 win10 及其以上，Linux 等。

四、 思维导图

本报告的核心流程图如下图 1 所示。

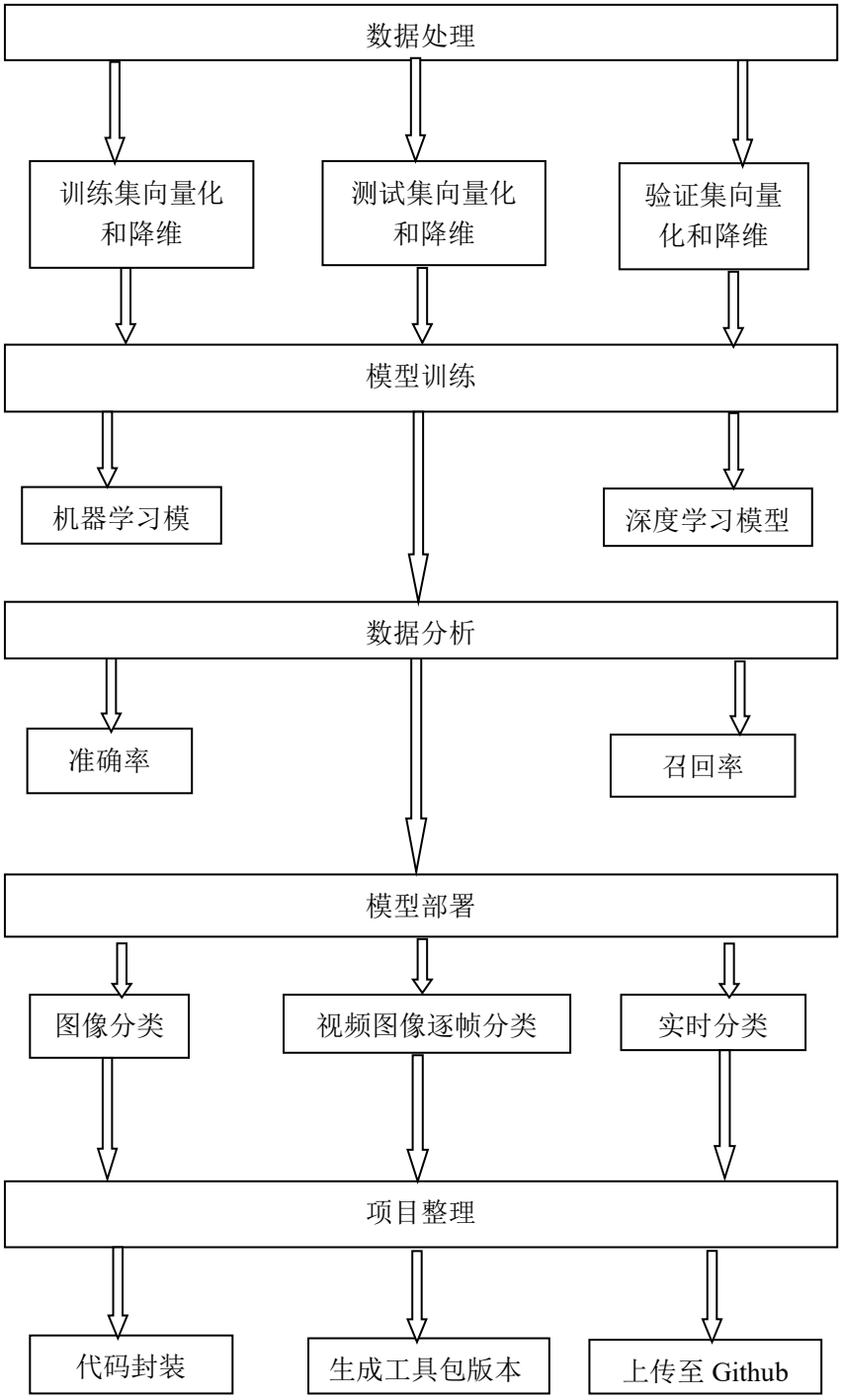

图 1 本报告核心流程图

如上述流程图所示，本次实验分为五大步，分别是：数据处理、模型训练、数据分析、模型部署以及代码优化和封装。

由于后期训练方式的不同，因此数据处理按照传统模型和深度学习也有所不同。利用传统模型训练时，处理数据的思想是：读取每一张图片的值，并对其进行裁剪，获

得一个长宽像素固定大小的图像，并将这张图象拉伸至一维空间，每一个像素值代表了其中的一个属性。利用深度学习模型训练时，我的做法是先进行预处理之后定义数据加载器。预处理的过程是将图像裁剪成一个长款固定的图像，这于之前提到的传统数据处理时的裁剪是一致的。与此同时将原本的 numpy 数据转化为 tensor 类型，因为在做深度学习的时候数据量太大，只能借助 GPU 来完成图像单元的计算，但是 GPU 不能够识别 numpy 类型的数据，因此需要将其转化为 tensor 类型。此外要对图像进行数据增强和归一化化。数据增强的原理时使图像偏转某个角度或增加噪声，或按某种策略改变像素值，依次增加模型的鲁棒性；归一化化是利用 imagenet 所有图像的 RGB 通道的均值和方差来做正则化（由于图像交叉大，因此假定他们都服从同一种分布，事实上也正是如此），以减小运算量。之后设置数据加载器。数据加载器的作用是将若干张（一般为 bich_size 大小）图像打乱后集成在一个数据表里，相当于这一个数据表就代表了 bich_size 个图像的特征。至此，深度学习的数据处理部分结束。

数据处理完之后就要进行模型训练，在训练模型的时候采用两种方法：传统机器学习和深度学习。传统学习本实验主要采取了 SVM 算法、KNN 算法、Decision Tree 算法、Bayes 算法。由于需要使用到验证集调参，因此需要设置每个模型的超参数，并在训练的过程中对其进行不断修改，直到拟合最好的参数出现。拟合最好的参数出现时，需要选取该超参数，并以该超参数重新再训练集+验证集上训练，并在验验证集上测试模型的效果。这里选取的指标为准确率和召回率。至此，传统机器学习训练模型告一段落。利用深度学习训练模型时，我使用了 torchvision 里面集成的 12 个分类网络之一的 resnet18 网络，并对其进行微调。微调的具体做法是：冻结前面所有层，只改变最后的全连接层、将其映射到 2 维空间（因为本实验是二分类实验），因此最后的维度是 2。这么做的依据是：上述所说的网络绝大多数是在百万甚至是数以亿计的图像上拟合的网络，这些网络在经历了海量数据后早已学会了如何提取特征，提取哪个部分的特征，因此当新的图像输入进去之后，按照原来提取特征的方式一般不会出现很大的误差，因此工作的重心就是将这么多的向量映射到分属两个类别的概率，哪个概率大，那个就更容易被预测为所识别的类别。在训练过程中，为了更加精细化训练，提高模型的质量，本实验特地设置了每一定个 epoch 就将学习率减半，从而减慢模型的拟合速度，防止过拟合现象的发生。至此，深度学习的模型训练部分结束。

模型训练完之后就要进行模型在测试集上的评估和分析。为了更好的对比利用两大类型的算法训练出的模型，本实验采用对比实验的原则，分别再相同打乱程度下的测

试集上测试，并将其准确度和召回率等各大指标利用图表的格式表现出来。

模型部署模块设计一种部署方式和三种传入数据类型。所谓一种部署方式是在本机上进行端侧部署，先假设有数据传入，再考虑该数据要被处理成什么样的类型才能够被模型所接受。由此分析，从而将模型进行预封装。设计三种传输数据类型分别为：视频流类型（可以对一段视频进行检测）、文件夹下的数据集类型、和调取本地摄像头获取的实时视频流类型。针对这三种数据均实现结果的精确预测，并将其串成一段视频存储在本地。在检测的过程中能实时显示当前检测对象的检测结果。至此，模型部署部分告一段落。

代码优化部分主要是对整个实现过程代码的复用率、代码的可读性、程序代码的健壮性、代码的可移植性等性能进行的一个综合考量，包含对程序传入参数的异常处理，即当需要的数据是字符类型，但是传入的参数是整型数据，这个时候程序报错该如何处理？代码不可避免地遇到了错误，但是不能使程序停止，此时又将如何处理？这个就是本块考虑的重要内容。为了保证程序的可移植性，最后要检查本程序所使用的各个工具包以及其对应的版本号，并生成其对应的 requirements.txt 文件，一共程序移植时能够快速恢复环境。

五、 核心代码

1. 利用传统机器学习方法训练模型的数据处理核心代码。

```
for i in X_train_path:
    image = cv2.imdecode(np.fromfile(i, dtype=np.uint8), cv2.IMREAD_COLOR)
    # 图像像素大小一致
    img = cv2.resize(image, (224, 224), interpolation=cv2.INTER_CUBIC)
    # 计算图像直方图并存储至 X 数组
    hist = cv2.calcHist([img], [0, 1], None, [256, 256], [0.0, 255.0, 0.0, 255.0])
    XX_train.append(((hist / 255).flatten()))
Y_train = numpy.array(Y_train)
```

代码中，X_train_path 是训练集数据的路径集合，对其进行遍历就能得到每一张图片的路径 。对于第 i 张图片，当读取它时，先用 imdecode 将其读入（编码，也可以用 imread，但是 imdecode 支持路径含中文的图像读取）。之后将图像裁剪成 224*224 大小的数据图，并计算裁剪后图像的直方图，对该直方图做一维拉伸处理，就得到维度为 224*224*3 的向量，将其存储在 XX_train 列表里面。当 XX_train 处理完后，将其对应的标签也全部数组化，这是因为在传统机器学习的 fit()函数中，只接受 numpy 类型的

数据。

2.利用深度学习方法训练模型的数据处理核心代码。

```
train_transform = transforms.Compose([transforms.RandomResizedCrop(224),  # 随即裁剪成 224*224 大小，因此要求图像超过 224*224
                                      transforms.RandomHorizontalFlip(),  # 数据增强，提高模型的鲁棒性
                                      transforms.ToTensor(),
                                      transforms.Normalize([0.485, 0.456, 0.406],  # imagenet 上(R,G,B)的均值和方差，
                                                           [0.229, 0.224, 0.225])  # 因此用这一组数据正则化
                                      ])
train_path = os.path.join(os.getcwd(), train)
train_dataset = datasets.ImageFolder(train_path, train_transform)
train_loader = DataLoader(train_dataset,
                          batch_size=BATCH_SIZE,  # 每次计算 32 个样本
                          shuffle=True,  # 在每个 epochs 计算式打乱数据
                          num_workers=4  # 进程数，相当于在同一时刻计算 4*32=128 张图片
                          )
```

代码中，为了和传统机器学习的模型效果一致，这里也将图像裁剪成 224*224 大小，之后利用 RandomHorizontalFlip()函数对图像进行图像增强的操作，这样做的好处是能够增加模型的容错度，即便在输入图像有歪曲、模糊等情况下依旧可以进行正常且正确的分类。之后将数据转化为 tensor 类型，前面提到过，转化为 tensor 类型的目的是使其支持 GPU 的并行计算。最后将数据归一化处理，目的是避免计算出很庞大的数据，一定程度上简化了运算。用来归一化的数据矩阵是在 IMAGENET 大数据库中（R，G，B）三个颜色通道的均值和方差。因为本实验数据时食物和常见的非食物，这些数据和 IMAGENET 有着很大的重合性，甚至是这些图像就包含在 IMAGENET 里面，因此选择这一组值做归一化绝对可靠。之后就是处理好的数据加载到数据加载器里面。数据加载其的作用是一次能加载若干张图片，代码中的 bitch_size 定义为 32，说明这个数据加载其一次运算能过够携带 32 张图片进入到 GPU 单元里进行运算，num_workers 定义的是进程数，就是在同一时间有 4 个携带 32 张图片信息的特征图进入到 GPU 的计算单元，这表明在同一时刻，利用 GPU 能够计算 32*4=128 张照片，而利用传统机器学习算法，在同一时刻只能处理一张图片，利用 GPU 计算的效率可见一斑。

3.利用传统机器学习方法训练模型的核心代码。

```
C = [0.2, 0.4, 0.5, 0.6, 0.7, 0.8]  # SVM 的惩罚系数
for epoch in range(1, epochs + 1):
    clf = SVC(C=C[epochs // 100]).fit(XX_train, Y_train)
    if epoch % 100 == 0:
        acc = round(clf.score(V_data, V_label), 2)
        print("惩罚系数 C=%.1f 在验证集上准确率:" % (epochs // 100), acc)
        SVM_ACC[str(epochs // 100)] = acc
```

在代码中，在每一个 epochs 里面都会迭代一次模型，当模型训练到 100 轮的时候，需要测试一下其在验证集上的的得分（即其在验证集上的准确率），并将其记录下来。事实上，在每次迭代到 100 轮的时候，惩罚因子 C 的值也会改变，原因是：在 epochs 不到 100 的时候，其惩罚因子 C 的取值时索引为 epochs//100，索引值为 0，惩罚因子大小为 0.2，当其超过 100 时，其索引值增加 1，依次增加，每个惩罚因子作为超参数都会得到在这段时间内最好的预测效果并将其记录在 SVM_ACC 字典里以便于后续的数据分析，以便于找到在这个范围内的最好的惩罚因子。其他的算法也类似，就不再一一赘述。

4. 利用深度学习方法训练模型的核心代码。

```
for epoch in range(epochs):
    lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.5)
    for images, labels in tqdm(train_loader):
        #images, labels = next(iter(train_loader))  # iter()作用将 train_loader 作用成一个迭代器，我的理解就是可以根据前面的找到后面的，是有序的。
                                                    # 以张量为单位,一个 image 就是一个 32 张*3 通道*224*224 大小的数据
        images = images.to(device)  # 把图像数据传到 GPU 或者 CPU 里面
        labels = labels.to(device)  # 把标签数据传到 GPU 或者 CPU 里面
        outputs = model(images)     # 以 image 为单位进入网络进行前向预测得到这 32 个图片在 2 各类别上的概率（张量）是一个 32*2 的张量，32 列表示 32 个图像，2 表示分属两类的概率
        loss = criterion(outputs, labels)  # 返回每个样本的平均交叉熵损失函数
         """反向传播三部曲"""
         optimizer.zero_grad()  # 清除梯度至 0
         loss.backward()  # 反向传播求梯度，微调使模型损失函数最小化
         optimizer.step()  # 优化更新迭代
         _, preds = torch.max(outputs, 1)  # _是每一列具体的值，但是 I don't care, 我要的值是最大值在 outputs 里每一列
         最大值的下标 preds，它反映了这一组图像每一张被预测成什么
    lr_scheduler.step()
```

代码中，首先规定了学习率，在每一轮迭代中，学习率会减半，这是因为该深度学习的网络架构已经很好了，对数据的拟合能力也已经很好了，设置每一轮迭代步长减

小至原来的一半，能够有效的防止模型过拟合。之后对数据加载器中的数据进行遍历加载，之后把特征图和标签都送进 CPU 或者 GPU 中参与运算和对模型的拟合，outputs 代表的是模型的输出，这是通过一次迭代产生的数据，与标签是有差距的，因此要计算他的损失函数，这里选取得损失函数是交叉熵函数。之后将梯度清零，进行反向传播求梯度、优化器迭代。对于本次取得的 outputs 是一个 32*2 得向量，32 表示一个特征图上的图片书，2 表示的是分属两类得特征，利用 torch.max 函数得到当前属于两类中最大概率值的下标，即预测的类别。短横线表示的是具体的概率值，但是在本实验中用不到，因此用短横线忽略掉该值。

5.利用传统机器学习模型和深度学习测试模型的代码。

```
def model_test(model_path):
    clf = joblib.load(model_path)
    correct = 0
    acc = []
    for i in range(len(XX_test)):
        result = clf.predict([XX_test[i]])[0]
        if result == Y_test[i]:
            correct += 1
        acc.append(correct/(i+1))
         # print(correct/(i+1))
    X = [i+1 for i in range(len(XX_test))]
    plt.plot(X,acc,c='red')
    # for i in range(len(XX_test)):
    #     plt.scatter(X[i],acc[i],c='red')
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.show()
```

代码中，函数参数传入的是模型的地址，首先加载模型，之后利用模型对测试集的书籍进行预测。遍历每一次预测，如果本次预测的标签与其原本的标签一致，记预测正确的变量 correct 自增 1，并将累积预测的正确率计入到 acc 列表里面，供后续画图时用。在画图阶段，做出在每一次预测时的正确率和累计测试集数量的折线图，改图能够反映模型在测试集上的预测情况。

6．利用传统机器学习模型和深度学习测试模型的代码。

```
def process_frame(img):
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # BGR 转 RGB
    img_pil = Image.fromarray(img_rgb)  # array 转 PIL
    input_img = test_transform(img_pil).unsqueeze(0).to(device)  # 预处理
    pred_logits = model(input_img)  # 执行前向预测，得到所有类别的 logit 预测分数
```

```
    _, pred_ids = torch.max(pred_logits,1)
    img = np.array(img_pil)  # PIL 转 array
    img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)  # RGB 转 BGR
    # 图片，添加的文字，左上角坐标，字体，字体大小，颜色，线宽，线型
     cv2.putText(img, idx_to_labels[int(pred_ids)], (20, 40), cv2.FONT_HERSHEY_SIMPLEX,0.7, (0, 0, 255), 2,cv2.LINE_AA)
    return img
```

代码中，参数 img 传入的是每个测试集读取 BGR 格式的数据，进入函数主体中需要将其过渡到 RGB 格式进而转化为 PIL 格式的数据，这是因为定义的 test_transform 只能读取 PIL 类型的数据。在这里，每次读取一张照片处理后被 model 预测，model 在外部函数中已经被加载成功，经过前向预测返回值是置信度，即当前图像分属于每个类别的概率，对峙进度进行解析，求当前置信度中最大的下标，通过索引就能够找到该图像所对应的下标。由于 opencv 只能对 BGR 格式的数据进行处理，因此要将预测的信息写在图上需要将原来 PIL 格式的数据转化为 BGR 格式，因此需要对最开始的操作反向操作，之后向图片上写下标签内容。至此，利用深度学习模型在测试集上验证的过程结束。

7. 模型在本机上进行部署的代码。

```
import cv2
cap = cv2.VideoCapture(1)# 获取摄像头，传入 0 表示获取系统默认摄像头
cap.open(1)# 打开 cap
while cap.isOpened():# 无限循环，直到 break 被触发
    success, frame = cap.read()# 获取画面
    if not success:
        print('Error')
        break
    frame = process_frame(frame) ## !!!处理帧函数
     cv2.imshow('my_window', frame)# 展示处理后的三通道图像
     if cv2.waitKey(1) in [ord('q'), 27]:  # 按键盘上的 q 或 esc 退出（在英文输入法下）
         break
cap.release()# 关闭摄像头
cv2.destroyAllWindows()# 关闭图像窗口
```

以上代码是对模型在本机上进行部署的描述。在代码中，首先获取摄像头，一般传入 0 为本机摄像头，当只有一个摄像头时只能传入 0，本实验利用的外接摄像头因此调用该摄像头的编号为 1。之后打开摄像头，里面空值为系统默认摄像头。用循环来控制摄像头一直能获取图像，若成功获取返回 True 和获取的图像数据，否则返回 false 和空并输出错误后结束循环。当获取图像成功后调用 process_frame 函数对图像进行预测和

标记该图像的操作。事实上在调用模型时，由于两种数据处理方式不同，因此需要对其分别进行处理。本实验的思路是：将利用深度学习训练出的模型扩展命为.pth 文件，将利用机器学习训练出的模型扩展为.pt 文件，这样在调用模型加载的时候只需要判断模型文件的扩展名，就能判断出改以什么样的方式处理数据，进而实现程序的健壮性。

8.本实验采取了两种方式进行训练和预测。因此可按训练和预测利用的学习方式，以两种方法封装成类，其中在每一种方法中处理数据、训练数据、预测数据的函数将其内置为类方法。

六、 数据分析

数据 1：通过传统机器学习中伯努利贝叶斯算法（BernoulliNB）实验得到模型的准确率和召回率如下图 2 所示。

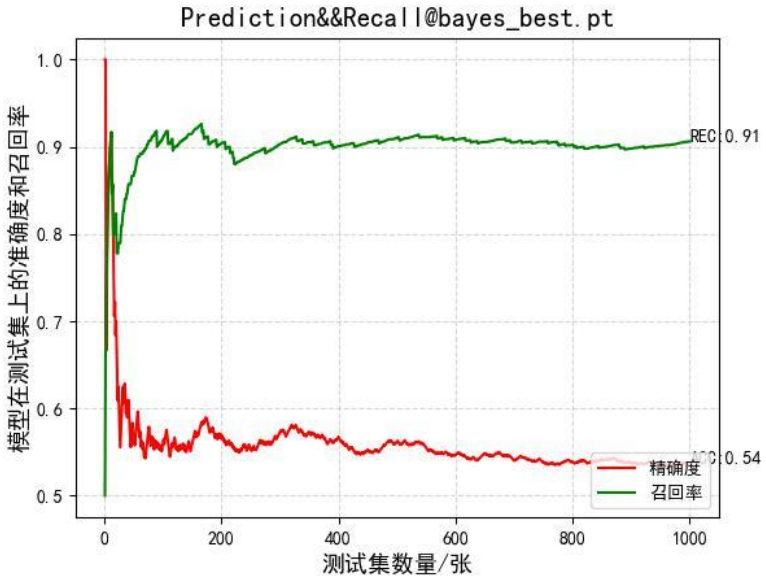

图 2 伯努利贝叶斯模型的准确率和召回率

数据 2：通过传统机器学习中决策树算法（Decision Tree）实验得到模型的准确率和召回率如下图 3 所示。

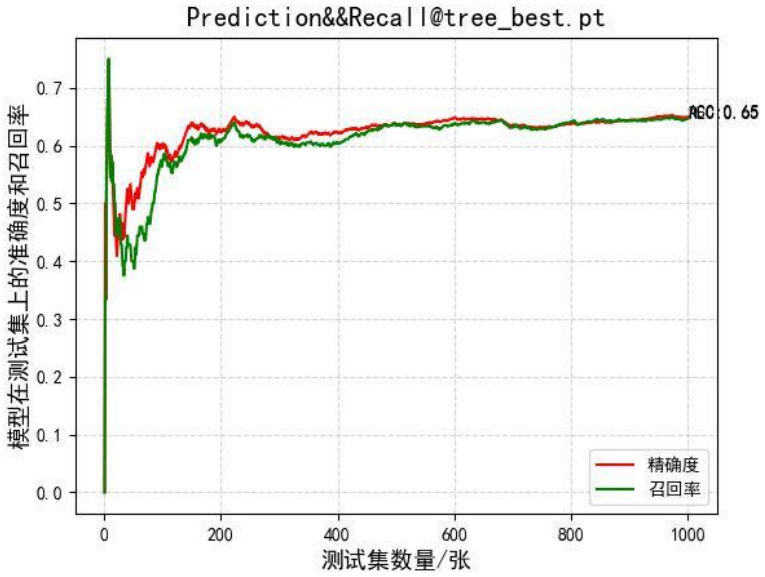

图 3 决策树模型的准确率和召回率

数据 3：通过传统机器学习中最近邻算法（KNN）实验得到模型的准确率和召回率如下图 4 所示。

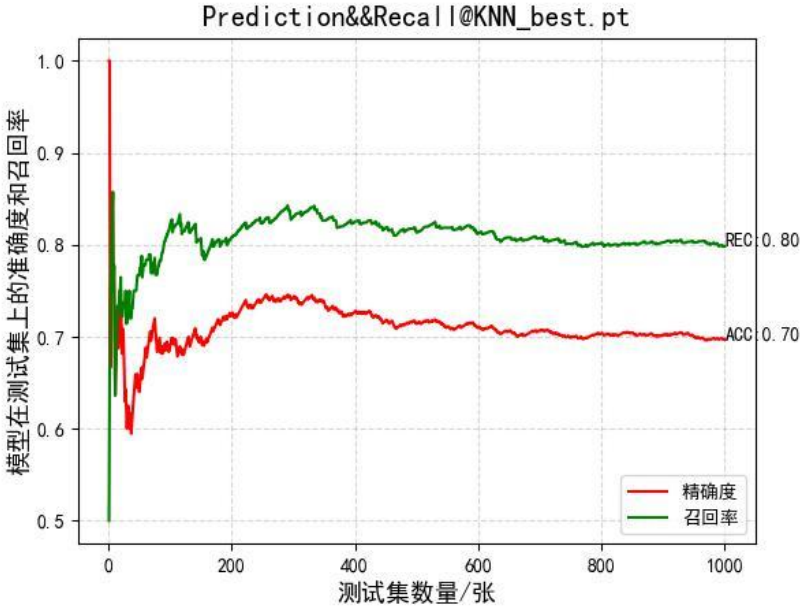

图 4 最近邻模型的准确率和召回率

数据 4：通过传统机器学习中支持向量机算法（SVM）实验得到模型的准确率和召回率如下图 5 所示。

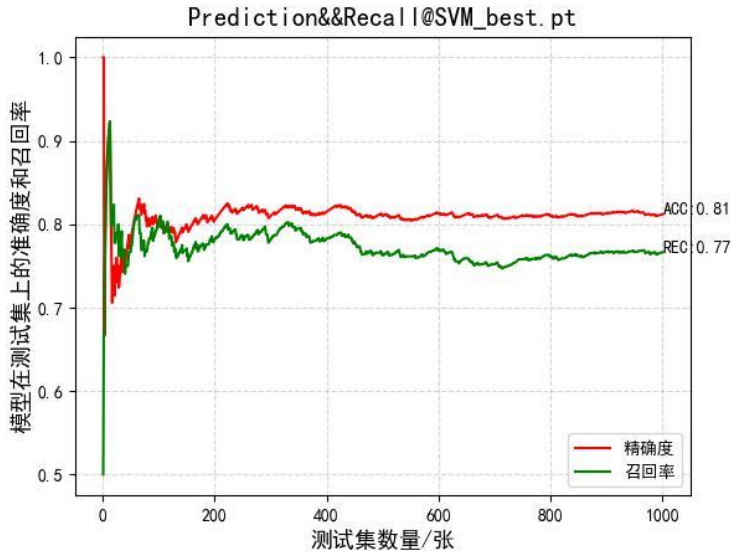

图 5 支持向量机模型的准确率和召回率

数据 5：通过深度学习里的迁移学习算法（Transformer）实验得到模型的准确率和召回率如下图 6 所示。

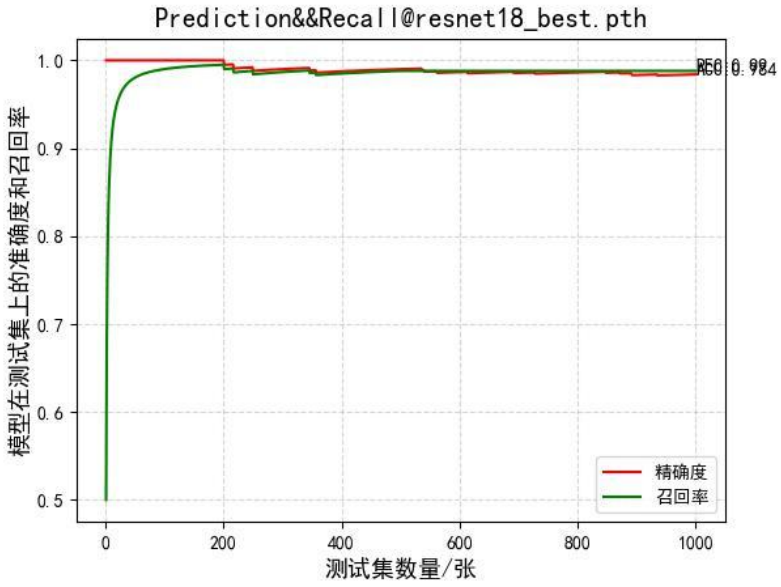

图 6 深度学习微调模型的准确率和召回率

数据 6：对上述五种方法进行纵向比较，分别对比他们的精确度和召回率，列出如表 1 所示的图标结构。

表 1 本实验 5 中模型准确率和召回率比较

模型范围	模型名称	准确度	召回率
机器学习	Bayes 模型	0.54	0.91
	KNN 模型(K=9)	0.70	0.80
	Decision Tree 模型	0.65	0.65
	SVM 模型（C=0.9）	0.81	0.77
深度学习	DNN 模型	0.98	0.99

从表中可以直观地看出深度学习模型的准确度和召回率要高于传统机器学习模型的。这是因为深度神经网络是靠卷积和池化来提取所有的特征，是将图像所有的像素点进行卷积运算，之后进行池化操作，找到代表在该图像上特定卷积核大小范围内像素的代表值。通俗来讲，就是用一个点来代表周围一部分点。如此循环往复下去，机器最终学到的特征是该图像中最具有代表性的特征，而利用传统机器学习对图像进行处理，最核心的也是降维，但是它的降维只取决于人为在直观上能够看到的具有代表性的图像范围，或者通过一定的降维策略来时限维度下降。因此在维度上，机器学习的有局限性，这就导致了，在某些情况下，机器学习的学习效果略低于深度学习模型。

其次，在机器学习模型中，本实验设置了一组对比实验，探究调参对模型准确度和召回率的影响。实验组维 KNN 组和 SVM 组，对照组是 Bayes 组和 Decision Tree 组。下图是实验组调参的过程和数据。

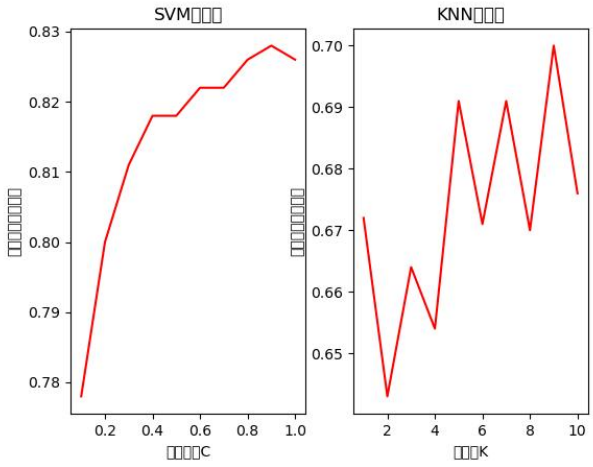

图 7 SVM 和 KNN 调参

由于训练不易（其中 SVM 模型训练在服务器上运行了 18 个小时，500 个 epochs），因此上图中文显示出现乱码，但是不影响观察结果。从做图可以看出，在验证集上的精确度在一定范围内是随着惩罚因子 C 的增大而增大，当惩罚因子为 0.9 时，SVM 模型的精确度达到最大。因此取惩罚因子 C 为 0.9，重新训练，最后训练的模型效果如上图 5 所示。同理，KNN 选取 K=9 时的精确度最大，因此利用 K=9 重新训练模型，最终得到的模型效果如上图 3 所示。通过比较，不难看出，经过调参，SVM(不调参时 C=1)的精确度较不调参的精确度提高了 1.2%，KNN（不调参时 K=5）的精确度较不调参的精确度提高了 1.4%。因此通过调参能在一定程度上提高模型的精确度。反观没有经过调参的模型，到达自身的优化瓶颈时，即便训练再多轮数也不会有明显的精确度提升。

七、 实验总结

在本次实验中，在两个领域之间来回切换的思路的经历真的终身难忘。首先我利用迁移学习理论对 Microsoft 团队的 Resnet 残差网络进行微调，得到了非常不错的效果，准确率达到 98.5 以上。但是当我在利用传统机器学习拟合模型的过程中却遇到了前所未有的波折。

一开始，我就直接把图片进行读取并向量化，最后制成数据集供模型训练，对裁剪的问题早已忘得一干二净，结果程序报错，报错的大意是两次读取的向量不是等长的，这反映在真实数据上的就是数据的属性值不一样。这是因为训练集的图像并不是同一个大小的，因此在进行向量化的时候需要裁剪固定大小的数据才能保证最后数据的维度是相同的。

其次，按照上文所述，当我把维度统一后，新的问题接踵而至。在训练后，我得到的 SVM 模型足足有 1.9G，这个模型在测试的时候光加载就花费了 15 分钟，于是我想到要做模型压缩。首先我回调制作数据集的函数，检测数据集的向量格式，返回的是 float32 类型的向量。Float32 格式的向量，一个数字就需要 4 个字节，若按照原先的向量化来算，一张图像裁剪并经过向量化后需要占用 256*256*4=262144bi，约 32M，3000 张照片所需要占用的大小不可小觑。因此我选择了利用更小的向量表示格式，unit8 格式，每个数仅用 8 个 bit 就能够表示，大大见笑了模型的体积。然而，在将数据改为 unit8 格式的数据后，并没有导致模型减小很多，甚至减小到原来的一半都不到。因此我决定从和新竹问题。

在这 256*256 大小的属性集上，已然形成了维度灾难，而且并不是所有的都对模型的正向拟合有帮助，甚至会抑制模型的拟合，因此对高维数据进行降维是对模型进行

压缩的必由之路。利用 PCA 算法，对原来 65536 维的数据进行降维至 1000 维时，模型的大小竟然被压缩到 19MB,这是一个质的飞跃，因此通过本次实验，我得出结论，模型的大小和训练数据的维度密切相关。

当在我暗暗窃喜自己终于把模型压缩到一个理想的大小时，另外一个问题出现了——模型的精确度不高。目前在实验环境下，在 1000 张测试集下，预测的正确率只有 53%左右，在正常情况下这是不应该出现的情况。我想到了可能是超参数设置的不合理，于是便走上了本实验的调参之路，也明白了验证集存在的意义。

当结束调参之路后，模型训练和预测的任务也算圆满结束。我又开始了模型部署之路。模型部署就部署在本机端。在本机端可以实现对路径下图像的食物非食物的图像识别、对传入视频的逐帧进行食物非食物的图像识别以及在本机下的实时检测与识别。其核心思想就是将图像传入，将该图像经过裁剪、拉伸等处理成为模型能够识别的数据类型，将其传入到模型中，经过处理后返回类别的值（0 或 1），最后根据类别的列表或者字典索引直接找到该值所对应的类别，将其标记在图片上，多张图片连续串起来就形成了检测的视频。其中利用本实验训练出的 Resnet_best.pth 模型部署，实现路径下的图像识别如下图 7 所示（转为 HTML 格式可见）。

图 7 识别效果

最后通过本次实验，我学会了代码优化并用命令行传递参数的实现。主要原理是利用 argparse.ArgumentParser()函数，其可以加任意多的参数。这些参数可以被**var()同

时传递，底层逻辑我还是没有弄明白，但是不耽误使用。其次，这个传参存在一定的问题，就是有时候可能会返回列表格式的数据。这个时候就要小心一点，可以首先判断传递过来的参数是什么类型的，如果直接是字符串类型，那就直接放过，任其进入函数。如果是非字符串格式（一般为列表）则取列表的第一个元素即可。一切正常后就能使用类进行封装。由于本实验用到装饰器的地方并不多，因此选用类来封装代码。封装后的代码见附页 1。在上述结束后，利用 git 命令，将代码开源到 Github 上面。

八、 实验难题

1.解决摄像头读取图像的问题

摄像头读取的数据是 np.arrar()类型的，但是图像处理是需要将该向量拉成一维的向量，最开始的时候我试过将其转换为 unit8 格式的数组，但是一直在报错，而且最后挑食的时候也没能将该数据拉成一维向量。后来我在官方文档中查询

cv2.imdecode()

查询到该函数返回的就是一个三维的 numpy.ndarry 类型的数组，这个和之前从摄像头中获取的数据类型一样。也就是说，从摄像头中获取的 frame 数据不用再进行其他处理，只需要裁剪并拉伸成模型能够识别的数据大小就能够进行进行预测了。最终的运行效果如下图所示。

2.保存视频格式问题。

在保存部署后的模型时候，我利用 video.write()方法将每一张照片保存下来，并串成一段完整的视频，当最后保存视频打开却发现该项目的编码格式不受支持。保存视频格式问题。在保存部署后的模型时候，我利用 video.write()方法将每一张照片保存下来，并串成一段完整的视频，当最后保存视频打开却发现该项目的编码格式不受支持。

图 8 视频编码问题

于是我就想起来，可能是在将图片串成视频的过程中编解码的方式有问题，果不其然，我定位到了下述地方。在代码的第一行。阅读代码可知，要保存.mp4 格式的视频文件，其编码方式不能为 MJPG，而是用 mp4v 的编码方式编码，这样保存的视频才能保存成功。

```
video = cv2.VideoWriter('test.mp4',cv2.VideoWriter_fourcc(*'MJPG')
for i in  X_train_path:
    #读取图片
    img = cv2.imread(i)
```

图 9 视频编码问题解决

3.利用命令行传参问题

利用命令行传参问题时，重写之后数据变成了列表格式，无法被识别。这是因为利用了 pathlib 库函数处理的路径和字符串拼接后，会导致输入的字符串变成列表的形式。知道这一点我就明白了，解决问题的关键就在于判断后面的输入后得到的类型是否是列表格式，如果是就需要将其转化为字符串的形式，也就是取列表的第一个元素即可。我按找这样的思想，最终解决了问题。