机器学习识别普适物体数据集

小组编号：11
小组成员1：1854205 郑昕瑶-深度学习
小组成员2：1852329 赵孟石-深度学习
小组成员3：1853790 庄镇华-传统方法
小组成员4：1852305 包广垠-传统方法
[注]：排名不分先后

本次任务基于CIFAR-10 数据集进行分类，分别使用传统机器学习的方法和深度学习的方法进行分类，并结合实验数据分析总结。在传统机器学习方面，使用PCA、Hog、Daisy进行特征提取，采用了十种传统模型进行预测，并进行比较分析。在深度学习方面，使用了数据增强、权重衰减、拟正则化、变化学习率等方法进行优化，分别使用Simple_CNN、Deep_cnn、Vgg16、ResNet等网络架构进行训练和比较，在测试集上得到92%的结果。最后总结部分描述了神经网络调参的感悟，并探究了传统方法和深度学习方法的模型特点和不同。同时，还实现了加分项1手写实现贝叶斯分类，和加分项4构造并优化CNN。

一、数据准备与数据清洗

I.传统方法

原始数据为(32,32)的RGB图片，特征空间的维度为3072。传统的机器学习方法不能适应如此高维的特征空间。一方面，维度越高，数据在每个特征维度上的分布就越稀疏；另一方面，单独使用这3072个特征进行分类，将会忽略了数据之间的相关性，无法得到好的效果。综上所述，在使用传统的机器学习分类算法之前，我们需要对原始数据进行数据降维和特征选取。

特征选取的方法有很多，我们主要采取了三种特征选取的方法：

1.1 主成分分析PCA

主成分分析(PCA)是最重要的降维方法之一，在数据压缩消除冗余和噪音消除等领域都有广泛的应用，通过线性变换将原始数据变换为一组各维度线性无关的表示，找出数据里最主要的部分来代替原始数据。

我们读取数据后，通过reshape将训练数据和测试数据转换为二维矩阵。调用sklearn.decomposition内置的PCA（）函数进行主成分分析，选取了前628个主成分，使得主成分占比到达99%以上。具体见“附件5-PCA.ipynb”。

```
pca = PCA()
pca.fit_transform(x_train)
k = 0,current_sum = 0
total = sum(pca.explained_variance_)
while(current_sum / total < 0.99):
    current_sum += pca.explained_variance_[k]
    k += 1    #658
pca = PCA(n_components=k, whiten=True)
x_train_pca = pca.fit_transform(x_train)   #(50000, 658)
x_test_pca = pca.transform(x_test)         #(10000, 658)
```

1.2 特征检测算法HOG

方向梯度直方图(HOG)能够很好地描述局部目标区域的特征，是一种常用的特征提取方法。HOG+SVM在行人检测中有着优异的效果，通过计算和统计图像局部区域的梯度方向直方图来构成特征。在一副图像中，局部目标的表象和形状能够被梯度或边缘的方向密度分布很好地描述。HOG特征是在图像的局部方格单元上操作，所以它对图像几何的和光学的形变都能保持很好的不变性，这两种形变只会出现在更大的空间领域上。

HOG特征提取的过程：把样本图像分割为若干个像素的单元，把梯度方向平均划分为多个区间，在每个单元里面对所有像素的梯度方向在各个方向区间进行直方图统计，得到一个多维的特征向量，每相邻的单元构成一个区间，把一个区间内的特征向量联起来得到多维的特征向量，用区间对样本图像进行扫描，扫描步长为一个单元。最后将所有块的特征串联起来，就得到了人体的特征。

因为Hog特征提取的是纹理特征，颜色信息不起作用，所以现将彩色图转为灰度图；通过调用skimage.feature内置的Hog函数提取Hog特征。在使用过程中，我们提取8个方向上的HOG特征，将数据由3072维降低至324维。具体见“附件6-Hog.ipynb”。

```
def rgb2gray(im):
    gray = im[:, :, 0]*0.2989+im[:, :, 1]*0.5870+im[:, :, 2]*0.1140
    return gray
cnt=0
for data in TrainData:
    image = np.reshape(data[0].T, (32, 32, 3))
    gray = rgb2gray(image)/255.0
    fd = hog(gray, orientations, pixels_per_cell, cells_per_block)
    train_feature[cnt] = fd
    cnt = cnt + 1
```

1.3 局部特征描述子DAISY

DAISY是面向稠密特征提取的可快速计算的局部图像特征描述子，它本质思想和SIFT是一样的：分块统计梯度方向直方图。DAISY在分块策略上进行了改进，利用高斯卷积来进行梯度方向直方图的分块汇聚，这样利用高斯卷积的可快速计算性就可以快速稠密地进行特征描述子的提取。调用方法与Hog类似，我们提取图片的DAISY特征，使数据降维至200维。具体见“附件7-Daisy.ipynb”。

```
def rgb2gray(im):
    gray = im[:, :, 0]*0.2989+im[:, :, 1]*0.5870+im[:, :, 2]*0.1140
    return gray
cnt=0
for data in TrainData:
    image = np.reshape(data[0].T, (32, 32, 3))
    gray = rgb2gray(image)/255.0
    fd = daisy(gray, orientations, pixels_per_cell, cells_per_block)
    train_feature[cnt] = fd
    cnt = cnt + 1
```

II.深度学习

CIFAR10 Dataset是一个内涵六万张图片的数据集，它的通道数是三个，用来表示其彩色的画面，并且图像尺寸是 32*32，其中分成训练集五万张与测试集一万张。其内部排列方式为一个大的字典，图片数据对应 'data' 字典键，标签数据对应 'labels' 字典键，而单张图片数据排布方式为一个一维列表：[...1024 red ... ...1024 green... ...1024 blue...]。

1.1 数据集加载

1) 方法一：使用tensorflow内置数据集进行加载。

```
from keras.datasets import cifar10
(x_train,y_train),(x_test,y_test) = cifar10.load_data()
```

2) 方法二：下载数据集，使用CIFAR-10官方给出的使用方法加载数据集。

由于数据是由 5 个批次储存，把训练数据融合成一块，后面处理和调用也更方便，并且其图片大小为 32x32 的尺寸，并不至于大到没办法一次容纳。编写loadData函数如下图陈列：通过unpickle加载各批次数据，并通过append函数将5个批次数据融合，通过内部字典键'data'和 'labels' 得到数据和标签。

```
def unpickle(file):
    with open(file, 'rb') as fo:
        dict = pickle.load(fo, encoding = 'iso-8859-1')
    return dict
def loadData(train_dir, test_dir):
    x_train = np.empty(shape = [0, 3072]),y_train = []
    for i in range(5):# 加载训练集
        file_batch = train_dir + str(i + 1)
        dict_train_batch = unpickle(file_batch)
        data_train_batch = dict_train_batch['data'], labels = dict_train_batch['labels']
        x_train = np.append(x_train, data_train_batch, axis = 0), y_train = np.append(y_train,
labels)
    dict_test = unpickle(test_dir)# 加载测试集
    x_test = dict_test['data'], y_test = dict_test['labels']
    return x_train, y_train, x_test, y_test
(x_train,y_train),(x_test,y_test) =loadData('./data_batch_','./test_batch')
```

1.2 构造批次数据

tensorflow调用model.fit函数训练模型时，设置batch_size、epochs、steps_per_epoch、shuffle等参数的值，进行批次数据的构造、分割和打乱。参数说明如下，参数设置将在模型构造和训练部分具体说明：

batch_size：整数，指定进行梯度下降时每个batch包含的样本数;

epochs：整数，训练终止时的epoch值;

shuffle：布尔值，表示是否在训练过程中每个epoch前随机打乱输入样本的顺序;

steps_per_epoch: 一个epoch包含的步数（每一步是一个batch的数据送入）。

1.3 异常点清洗

如果数据集中存在反例，将会对学习过程产生不利影响。在CIFAR-10数据集中，可能存在标签错误的情况，为了验证导入的数据集是否与标签匹配，编写函数浏览数据集并验证每个样本的标签。

```
(x_train,y_train),(x_test,y_test) = cifar10.load_data()
label_dict=
{0:'airplane',1:'automobile',2:'bird',3:'cat',4:'deer',5:'dog',6:'frog',7:'horse',8:'ship',9:'tr
uck'}
def plot_images_labels_prediction(images,labels,idx,num):
    fig=plt.gcf()
    fig.set_size_inches(12,14)
    for i in range(0,num):
        ax = plt.subplot(7,7,i+1)
        ax.imshow(images[i+49*idx],cmap='binary')
        title= str(i+49*idx)+' '+label_dict[labels[i+49*idx][0]]   #显示数字对应的类别
        ax.set_title(title,fontsize=10)
        ax.set_xticks([])
        ax.set_yticks([])
    plt.show()
```

如下图所示，抽查数据检查标签是否错误，未发现明显的标签错误。

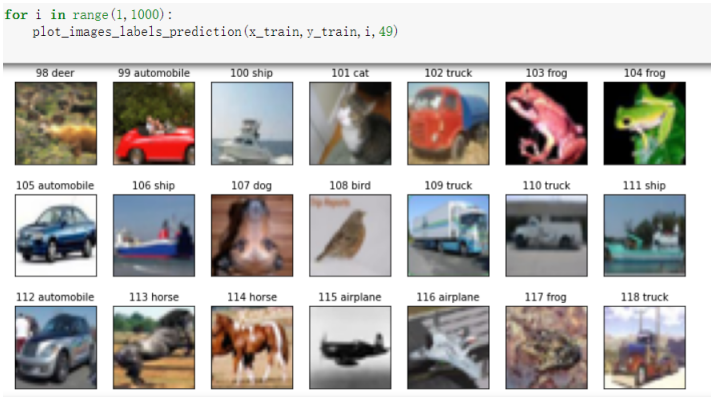

图1：检查标签是否错误

1.4 数据规范化

图像特征值规范化可以减少数值大的特征的影响，使模型更准确，并且加快学习算法的收敛速度。常用的特征放缩如下，其中第一项归一化和第三项标准化最为常见。如果数据不稳定，存在极端的最大值最小值，则归一化不适应，我们通过一个小实验排除这种可能，决定使用归一化对原始数据进行线性变换。

Rescaling (min-max normalization) $$x^{'} = \frac{x - min(x)}{max(x) - min(x)}$$

Mean normalization $$x^{'} = \frac{x - mean(x)}{max(x) - min(x)}$$

Standardization(Z-score normalization) $$x^{'} = \frac{x - mean(x)}{\sigma}$$

通过调用numpy内置的max函数和min函数得到最大最小值，进行归一化，相关代码如下。

```
xtrain_max=np.max(x_train),xtrain_min=np.min(x_train)
xtest_max=np.max(x_test),xtest_min=np.min(x_test)
x_train = (x_train.astype('float32')-xtrain_min)/(xtrain_max-xtrain_min)
x_test = (x_test.astype('float32')-xtest_min)/(xtest_max-xtest_min)
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

为了说明数据规范化在深度学习中的重要性，我们进行了一个简单的小实验。用一个简单的CNN模型进行测试，相同参数设置下对数据进行训练，test1不对数据进行处理，test2对数据进行归一化处理，test3对数据进行标准化处理，得到的训练结果在测试集上的accuracy分别为54.13%、66.88%、64.44%，由此可以看出数据规范化处理的重要性。通过下图可以看出，整体看来，归一化较标准化结果更优，因此我们选择对数据进行min-max归一化处理。具体参见“附件1-归一化对比实验”。

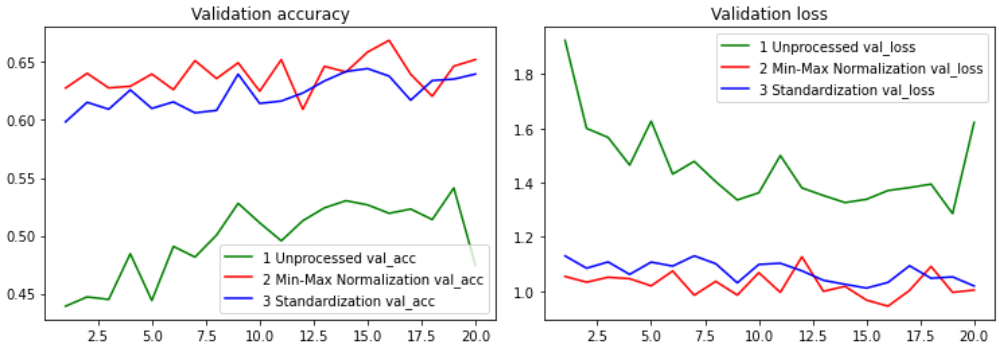

图2：归一化处理对比试验结果

1.5 数据增强

Keras提供了ImageDataGenerator（）接口用于数据增强。丰富的数据集是神经网络能够达到更高归类准确率的基本要素，同时还可减少过拟合的结果发生，它们就如同数据的噪声，为过拟合可能发生的情况提供了一道保险。不过使用数据增强产生数据会添加计算的负担，进而造成时间上的消耗，是我们应用此方法的时候一个重要的考虑要点。模型中训练时调用数据生成器如下：

```
train_datagan = ImageDataGenerator(
    rotation_range=10,    #随机旋转的度数范围；
    width_shift_range=0.1,     #浮点数：如果<1，则为总宽度的分数；
    height_shift_range=0.1,    #浮点数：如果<1，则为总高度的分数
    fill_mode='constant',      #"constant", "nearest", "reflect",  "wrap"四种填充模式
    cval=0.0,        #fill_mode = "constant"时填充边界的常值
    horizontal_flip=True)      #布尔值，水平随机翻转输入
test_datagen = ImageDataGenerator(rescale=1./255)
hist = model.fit_generator(
    train_datagan.flow(x_train, y_train, batch_size=128),
    steps_per_epoch = iterations, epochs = epochs_num,
    validation_data=(x_test,y_test),
    shuffle=True)#打乱批次数据
```

二、模型搭建

I.传统方法

在模型选择上，对于PCA、HOG、DAISY三种特征，我们每种特征都用了10个模型。分别是：伯努利朴素贝叶斯模型、高斯朴素贝叶斯模型、K最近邻模型、逻辑回归模型、线性支持向量机模型、支持向量机模型、决策树模型、随机森林模型、Adaboost模型和Xgboost模型。从分类上来说，我们选取的10种方法涵盖课程所讲述的贝叶斯方法、KNN、线性分类模型、SVM、决策树、集成学习方法。

2.1 朴素贝叶斯模型

朴素贝叶斯模型是一组非常简单快速的分类算法，通常适用于维度非常高的数据集。因为运行速度快，而且可调参数少，因此非常适合为分类问题提供快速粗糙的基本方案。

朴素贝叶斯分类器建立在贝叶斯分类方法的基础上，其数学基础是贝叶斯定理。它假设样本的先验分布，通过数据集上的观测值得到样本的后验分布。这是一种生成模型，不同类型的朴素贝叶斯分类器是由对数据的不同假设决定的，高斯朴素贝叶斯模型即假设样本的先验分布服从高斯分布，伯努利朴素贝叶斯模型即假设样本的先验分布服从伯努利分布。

```
gbn = naive_bayes.GaussianNB()#高斯朴素贝叶斯
gbn.fit(x_train_pca, y_train)
bbn = naive_bayes.BernoulliNB()#伯努利朴素贝叶斯
bbn.fit(x_train_pca, y_train)
```

2.2 KNN

KNN（K- Nearest Neighbor）法即K最邻近法，最初由 Cover和Hart于1968年提出，是一个理论上比较成熟的方法，也是最简单的机器学习算法之一。该方法的思路非常简单直观：如果一个样本在特征空间中的K个最相似（即特征空间中最邻近）的样本中的大多数属于某一个类别，则该样本也属于这个类别。

该方法在定类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。我们使用的模型中，k的取值为5，这5个参与投票的点的权重相等，在距离度量上选取的是欧氏距离。

```
knn = KNeighborsClassifier()
knn.fit(x_train_pca, y_train)
```

2.3 逻辑回归

Logistic Regression 虽然被称为回归，但其实际上是分类模型。它使用Sigmoid函数去拟合数据分布，从而进行分类任务。

```
lr = LogisticRegression()
lr.fit(x_train_pca, y_train)
```

2.4 支持向量机

支持向量机（support vector machines, SVM）是一种二分类模型，它的基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机；SVM还包括核技巧，这使它成为实质上的非线性分类器。

SVM的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题。SVM的学习算法就是求解凸二次规划的最优化算法。我们使用了均方误差，一般的支持向量机采用了最基本的RBF核函数，最大迭代次数均为1000次。

```
lsvc = svm.LinearSVC()#线性SVM
lsvc.fit(x_train_pca, y_train)
svc = svm.SVC()#非线性SVM
svc.fit(x_train_pca, y_train)
```

2.5 决策树

决策树是一类常见的机器学习算法。它是在已知各种情况发生概率的基础上，通过构成决策树来求取净现值的期望值大于等于零的概率，评价项目风险，判断其可行性的决策分析方法。

决策树中的每一个的节点相当于分支条件，在对未知样本进行分类时，需要根据这些分支条件进行选择，从而得到最终的类别。我们采用了CART决策树，即使用基尼指数来选择划分属性，对于决策树深度、最小不纯度不加以限制。

```
dt = tree.DecisionTreeClassifier()
dt.fit(x_train_pca, y_train)
```

2.6 集成学习方法

集成学习是组合这里的多个弱监督模型以期得到一个更好更全面的强监督模型，集成学习潜在的思想是即便某一个弱分类器得到了错误的预测，其他的弱分类器也可以将错误纠正回来。

随机森林：决策树的数量为10，使用CART决策树，不限制决策树深度。

Adaboost：基分类器为CART决策树，集成算法使用了SAMME.R，对样本集分类的预测概率大小来作为弱学习器权重，迭代更快。

Xgboost：基分类器为gbtree树，树的最大深度为6，学习率为0.3，为线性回归任务。

```
rf = RandomForestClassifier()#随机森林
rf.fit(x_train_pca, y_train)
adb = ensemble.AdaBoostClassifier()#Adaboost
adb.fit(x_train_pca, y_train)
xgb = xgboost.XGBClassifier()#Xgboost
xgb.fit(x_train_pca, y_train)
```

II.深度学习

2.1 Deep_CNN

在“附件1-归一化对比实验”中。我们使用简单的五层CNN进行训练，使用crossentropy作为目标函数，使用Adam梯度下降法进行参数更新，最好的情况下，可在测试集上得到69%的准确度。为了提高模型的性能，我们提高网络深度，搭建了Deep_CNN，该模型结构如下图，包括三层图3所示的卷积和一层图4所示的全连接层，完整结构参见“附件2-Deep_CNN.png”。相比于的简单的CNN，深度提高了五倍，测试集上训练准确度提高到约89%，训练结果和代码见“附件2-Deep_CNN”。

图3：卷积层结构

图4：全连接层结构

2.2 VGG16

VGG网络的主要思想是，使用多个较小卷积核（3x3）的卷积层代替一个卷积核较大的卷积层，一方面可以减少参数，另一方面相当于进行了更多的非线性映射，可以增加网络的拟合/表达能力。相比AlexNet的3x3的池化核，VGG全部采用2x2的池化核。第一层的通道数为64，后面每层都进行了翻倍，最多到512个通道，通道数的增加，使得更多的信息可以被提取出来。卷积核专注于扩大通道数、池化专注于缩小宽和高，使得模型架构上更深更宽的同时，控制了计算量的增加规模。

模型整体结构如图5，其中，Conv-64结构如图6，Conv层采用Relu激活函数，并加入BatchNormalization层进行拟正则化，相同Conv层采用Dropout层提高泛化能力，模型的完整结构见“附件3-VGG.png”。相比于Deep_CNN，VGG在测试集上的准确度提高到93%，训练结果和代码见“附件3-VGG”。

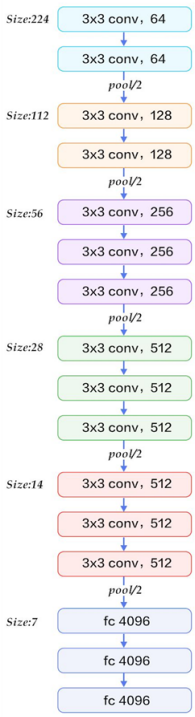

图5：VGG15结构示意

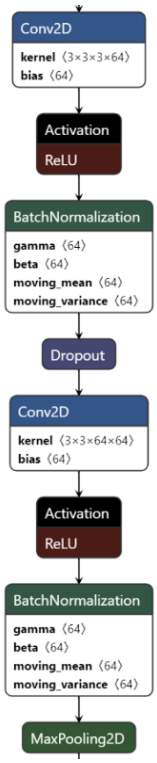

图6：Conv64结构示意

2.3 ResNet

以上两个卷积神经网络模型，都以层叠卷积层的方式提高网络深度，从而提高识别精度。但层叠过多的卷积层会出现梯度弥散(Vanishing)，backprop无法把有效地把梯度更新到前面的网络层，导致前面的层参数无法更新，因而存在“随着网络加深，准确率不下降”的问题。ResNet的skip connection就是为了解决这个问题，在后传过程中更好地把梯度传到更浅的层次中，网络的性能也就不会随着深度增加而降低了。

ResNet的基本模块如下代码所示，这个模块包含了一个卷积层，一个BN层，一个激活层：

```
def resnet_block(inputs, num_filters=16, kernel_size=3,strides=1, activation='relu'):
    x = Conv2D(num_filters, kernel_size=kernel_size, strides=strides, padding='same',
           kernel_initializer='he_normal', kernel_regularizer=l2(1e-4))(inputs)
    x = BatchNormalization()(x)
    if (activation):
        x = Activation('relu')(x)
    return x
```

我们采用20层的ResNet，模型结构如下图。第8、14层采用图7所示结构，第2~19层采用图8所示结构，第20层采用全连接层。具体结构见“附件4-ResNet.png"。

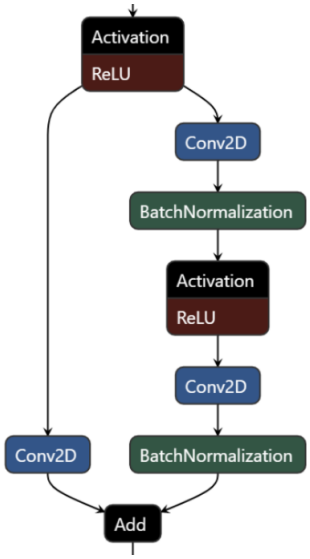

图7：ResNet结构一

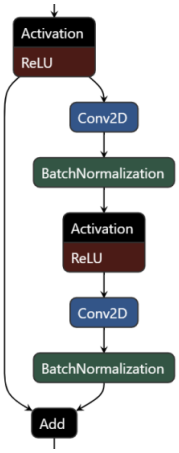

图8：ResNet结构二

三、模型优化与可视化

I.传统方法

3.1 模型优化

使用cifar10数据集进行实验，该数据集共有60000张彩色图像，这些图像是32*32，分为10个类，每类6000张图。这里面有50000张用于训练，构成了5个训练批，每一批10000张图；另外10000用于测试，单独构成一批。测试批的数据里，取自10类中的每一类，每一类随机取1000张。抽剩下的就随机排列组成了训练批。注意一个训练批中的各类图像并不一定数量相同，总的来看训练批，每一类都有5000张图。

使用最常用的特征提取方法PCA主成分分析方法，在Xgboost模型上得到的结果最最优，但只有0.5015。为了改进训练准确率，使用HOG+SVM这一在行人检测方面效果较好的机器学习方法，达到61.81%的准确率。继续改进，使用局部图像特征描述子DAISY进行特征提取，在SVM模型上能达到64.27%的准确率。

为了探索不同模型的优缺点，我们分别用三种特征提取方法，在十个传统机器学习模型上进行训练，最终得到的结果为DAISY+SVM效果最优。

3.2 可视化

3.2.1 特征可视化

下图展示用HOG方法提取特征得到的特征可视化分布。

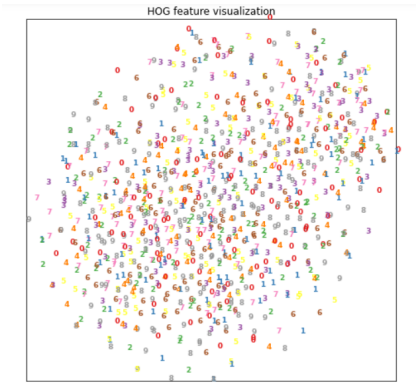

图9：HOG特征可视化

3.2.2 结果可视化

下图展示10个模型真实标签和预测标签混淆矩阵可视化结果。

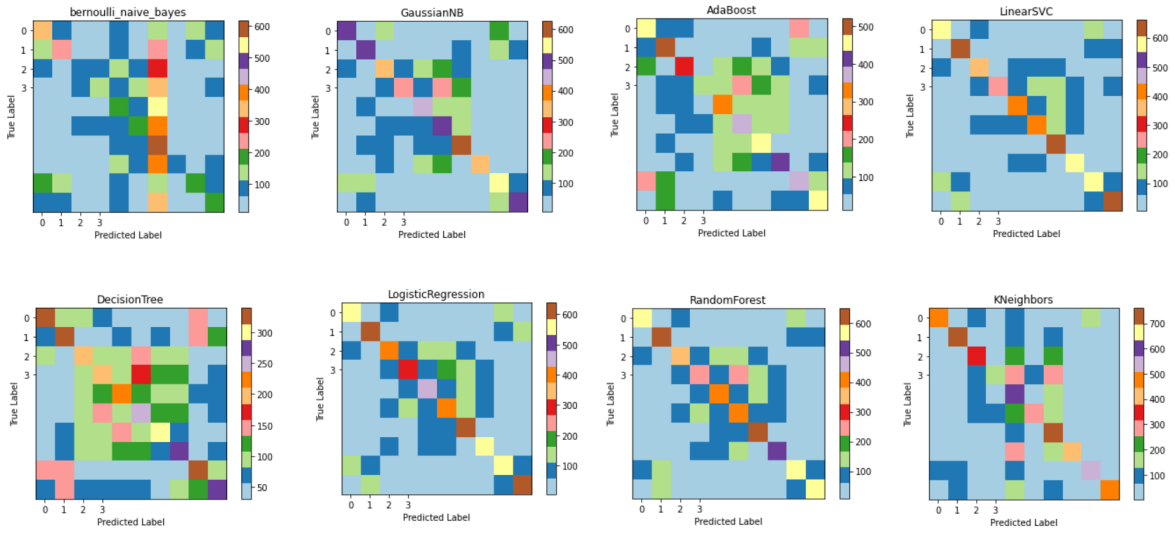

图10：传统方法预测结果混淆矩阵可视化

3.3 训练结果

对训练集和测试集都进行三种方式的特征提取，使用训练集训练模型，在测试集上进行验证，在测试集上的准确率如下图。

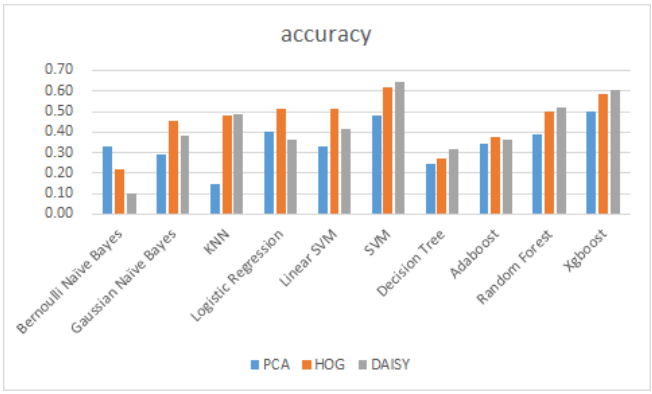

图11：三种特征提取方法在十种模型上的测试集准确率

具体数据见下表。

classification accuracy on the test set

Feature extraction method	PCA	HOG	DAISY
Bernoulli Naïve Bayes	0.3292	0.2216	0.1000
Gaussian Naïve Bayes	0.2931	0.4515	0.3839
KNN	0.1454	0.4788	0.4859
Logistic Regression	0.4030	0.5119	0.3604
Linear SVM	0.3317	0.5137	0.4117
SVM	0.4824	**0.6181**	**0.6427**
Decision Tree	0.2438	0.2696	0.3145

classification accuracy on the test set			
Adaboost	0.3426	0.3735	0.3648
Random Forest	0.3857	0.5000	0.5228
Xgboost	**0.5015**	0.5830	0.6063

II.深度学习

3.1 参数调整

深度神经网络中，训练深层神经网络十分耗时，且需要配置多个参数，这些参数的设置对网络的性能又十分重要。

- 学习率：学习率是指在优化算法中更新网络权重的幅度大小。学习率可以是恒定的、逐渐降低的、基于动量的或者是自适应的，学习率测量取决于所用的优化器算法，如SGD、Adam、Adagrad、AdaDelta或RMSProp等算法。在我们的网络中，采用了性能较好的Adam优化器和SGD优化器，使用了恒定学习率与逐渐降低的学习率两种策略，在后续Test4中，将体现学习率变化对网络性能的影响。
- 迭代次数：迭代次数是指整个训练集输入到神经网络进行训练的次数。当测试错误率和训练错误率相差较小时，可认为当前的迭代次数是合适的，否则需继续增大迭代次数，或调整网络结构。根据网络多次训练结果，我们发现训练次数过多易出现过拟合，训练次数较低，模型拟合效果不好，我们通过绘制loss变化图寻找较好的迭代次数100。
- 批次大小：在卷积神经网络的学习过程中，小批次会表现得更好，选取范围一般位于区间[16,128]内，CNN网络对批次大小的调整十分敏感，我们的网络中，因为数据量足够，批次大小统一采用128。
- 激活函数：激活函数具有非线性，理论上可以使模型拟合出任何函数。可以根据实际任务，选择其他类型的激活函数，如Sigmoid和Tanh等等。
- 隐含层的数目和单元数：增加隐含层数目以加深网络深度，会在一定程度上改善网络性能，但是当测试错误率不再下降时，就需要寻求其他的改良方法。但是增加隐含层数目也带来一个问题，即提高了训练该网络的计算成本，于是我们采用了ResNet网络解决这一问题。当网络的单元数设置过少时，可能会导致欠拟合，而单元数设置过多时，只要采取合适的正则化方式，就不会产生不良影响。
- Dropout方法：作为一种常用的正则化方式，加入Dropout层可以减弱深层神经网络的过拟合效应。该方法会按照所设定的概率参数，在每次训练中随机地不激活一定比例的神经单元，Dropout参数的默认值为0.5。在Test3中，将体现Dropout测量对网络性能的影响。

3.2 模型优化

Test1：Simple_CNN

模型的输入数据是网络的输入是一个4维tensor，尺寸为(128, 32, 32, 3)，分别表示一批图片的个数128、图片的宽的像素点个数32、高的像素点个数32和信道个数3。使用categorical_crossentropy作为损失函数，使用Adam梯度下降法进行参数更新，学习率设为固定值0.0001。训练50轮，观察到loss变化曲线、训练集准确率变化曲线和验证集准确率变化曲线如图11，测试集准确率为70.19%。

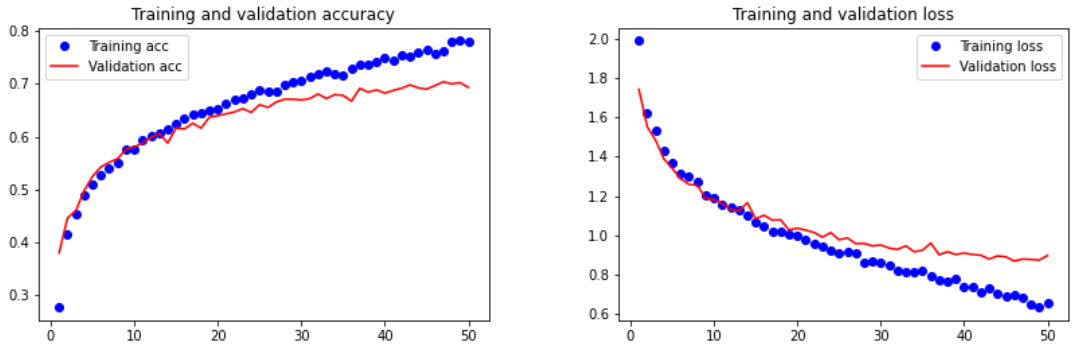

图12：未优化的训练结果

Test2：数据增强

数据增强：在训练数据上增加微小的扰动或者变化，一方面可以增加训练数据，从而提升模型的性能，另一方面可以增加噪声数据，增强模型的鲁棒性。训练结果如图12。训练集上准确率79.08%，测试集准确率为71.17%。

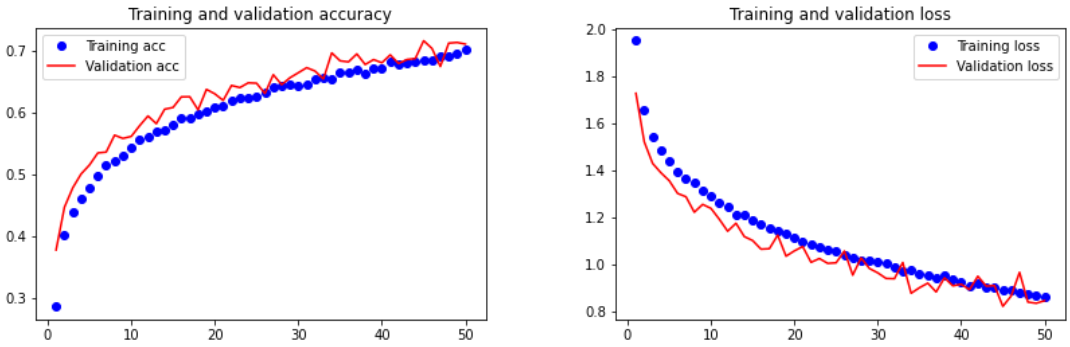

图13：数据增强训练结果

Test3：Test2+权重衰减+Dropout+拟正则化

权重衰减：对目标函数加入正则化项，限制权重参数的个数，防止过拟合。

Dropout：每次训练时，让神经元以一定的概率不被激活，这样可以防止过拟合，提高泛化能力。

批正则化：batch normalization对神经网络的每一层的输入数据都进行正则化处理，有利于让数据的分布更加均匀，是一种数据标准化方法，能够提升模型的拟合能力。

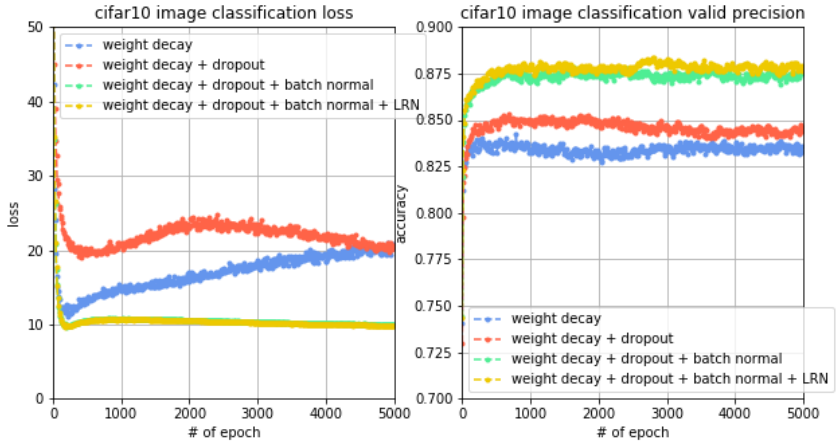

图14：优化训练对比实验

随着每一个模型提升的方法，都会使训练集误差和验证集准确率有所提升，其中，批正则化技术和dropout技术带来的提升非常明显，而如果同时使用这些模型提升技术，会使验证集的准确率从82%左右提升至88%左右，提升效果十分明显。而对于测试集，准确率也提升至85.72%。

Test4：Test3+变化学习率

在Test2的基础上对模型进行进一步的改进，使用变化学习率的方法，过程如下：首先使用较大的学习率进行训练，观察目标函数值和验证集准确率的收敛曲线。如果目标函数值下降速度和验证集准确率上升速度出现减缓时，减小学习率。循环步骤直到减小学习率也不会影响目标函数下降或验证集准确率上升为止。可以观察到，当100轮时，学习率从0.01降到0.001时，目标函数值有明显的下降，验证集准确率有明显的提升，而当200轮时，学习率从0.001降到0.0005时，目标函数值没有明显的下降，但是验证集准确率有一定的提升，而对于测试集，准确率也提升至86.24%。这说明，学习率的变化确实能够提升模型的拟合能力，从而提升准确率。

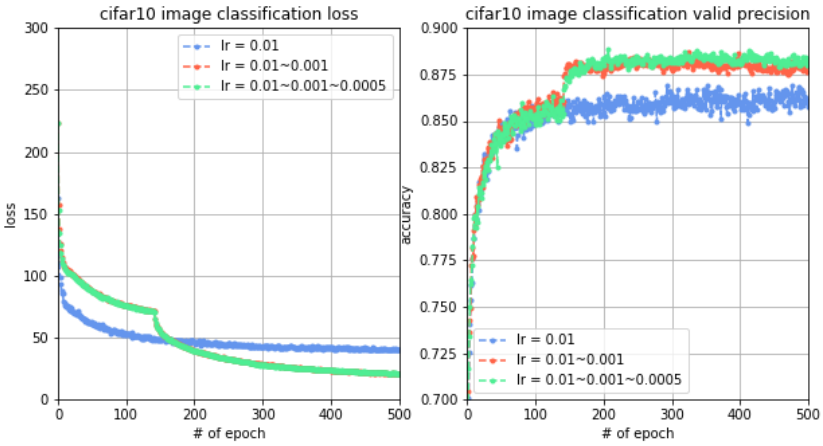

图15：变化学习率对比试验结果

Test5：提高网络深度

使用第二部分搭建的Deep_CNN和VGG来进行实验，使用以上优化方法，训练结果如下图，测试集上准确率增加到87.23%和90.37%。

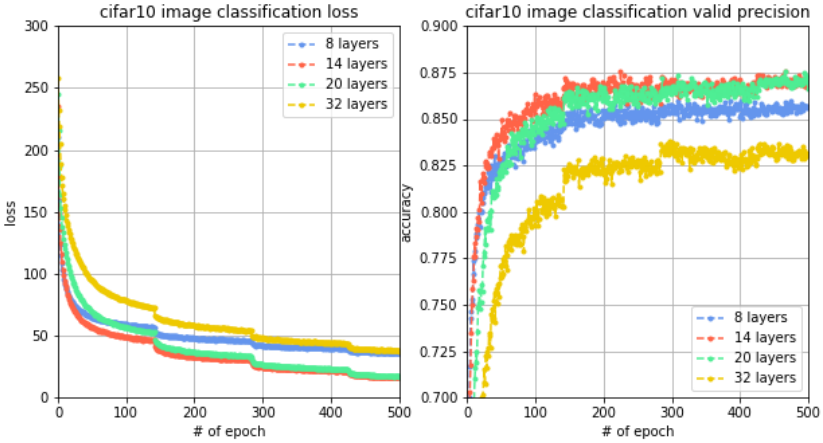

图16：网络深度对比试验结果

这两个网络都是通过增加深度来提高模型性能的，为了探究深度的提高对于网络性能的影响，我们进行了对比试验。可以观察到，网络层数从8层增加到14层，准确率有所上升，从14层增加到20层再增加到32层，准确率不升反降，这说明如果网络层数过大，由于梯度衰减的原因，导致网络性能下降，因此，需要使用其他方法对网络进行改进。

Test6：ResNet

第二部分模型搭建已经介绍过，残差网络技术，解决了由于网络层数加深而引起的梯度衰减问题。由于网络层数加深，误差反传的过程中会使梯度不断地衰减，而通过跨层的直连边，可以使误差在反传的过程中减少衰减，使得深层次的网络可以成功训练。下图可以看出，网络从20层增加到56层，训练loss在稳步降低，验证集准确率在稳步提升，并且当网络层数是56层时能够在验证集上达到91.55%的准确率。使用残差网络的技术，可以解决梯度衰减问题，发挥深层网络的特征提取能力，使模型获得很强的拟合能力和泛化能力。

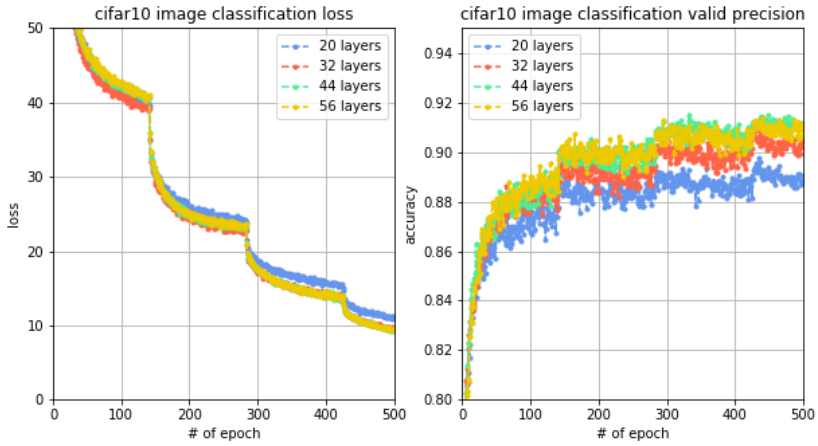

图17：ResNet网络训练结果

3.3 训练结果

采用Adam优化器或变化学习率的sgd优化器，使用categorical_crossentropy损失函数，对模型进行不断改进，在测试集上的预测准确率从最初的61.49%提升到92.37%、

改进方法	steps_per_epoch	epochs	测试集的准确率	提升
基本神经网络	390	100	61.49%	-
+数据归一化	390	100	70.19%	8.7%
+ 数据增强	390	100	71.17%	9.68%
+ 模型优化	390	100	85.72%	24.23%
+ 变化学习率	390	100	86.24%	27.75%
Deep_CNN	390	100	87.23%	25.74%
VGG_CNN	390	100	90.37%	28.88%
ResNet	390	100	91.87%	30.38%

3.4 可视化

我们可以通过可视化各个卷积层，来更好地了解卷积神经网络是如何学习输入图像的特征。

3.4.1可视化中间激活

可视化卷积神经网络的中间输出：有助于理解卷积神经网络连续的层如何对输入进行变换，也有助于初步了解卷积神经网络每个过滤器的含义。

可视化中间激活，是指对于给定输入，展示网络中各个卷积层和池化层输出的特征图（层的输出通常被称为该层的激活，即激活函数的输出）。我们希望在三个维度对特征图进行可视化：宽度、高度和深度。每个通道都对应相对独立的特征，所以将这些特征图可视化的正确方法是将每个通道的内容分别绘制成二维图像。

```
#提取前 8 层的输出
layer_outputs = [layer.output for layer in model.layers[:8]]
#创建一个模型，给定模型输入，可以返回这些输出
activation_model = Deep_CNN()
#返回8个Numpy数组组成的列表，每个层激活对应一个 Numpy 数组
activations = activation_model.predict(img_tensor)
#第一层激活的第六个通道
plt.matshow(first_layer_activation[0, :, :, 6], cmap='viridis')
```

图18：可视化中间激活

3.4.2可视化过滤器

可视化卷积神经网络的过滤器：有助于精确理解卷积神经网络中每个过滤器容易接受的视觉模式或视觉概念。想要观察卷积神经网络学到的过滤器，一种简单的方法是显示每个过滤器所响应的视觉模式。这可以通过在输入空间中进行梯度上升来实现：从空白输入图像开始，将梯度下降应用于卷积神经网络输入图像的值，其目的是让某个过滤器的响应最大化。得到的图像是选定过滤器具有最大响应的图像。

```
#为过滤器的可视化定义损失张量
model = vgg(), layer_name = 'conv1', filter_index = 0
layer_output = model.get_layer(layer_name).output
loss = K.mean(layer_output[:, :, :, filter_index])
#获取损失相对于输入的梯度
grads = K.gradients(loss, model.input)[0]
#梯度标准化技巧
grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-5)
#给定Numpy输入值，得到两个Numpy输出值，两个张量分别是损失值和梯度值
iterate = K.function([model.input], [loss, grads])
loss_value, grads_value = iterate([np.zeros((1, 32, 32, 3))])
#通过随机梯度下降让损失最大化
input_img_data = np.random.random((1, 32, 32, 3)) * 20 + 128.
step = 1. #每次梯度更新的步长
for i in range(40): #运行 40 次梯度上升
    loss_value, grads_value = iterate([input_img_data]) #计算损失值和梯度值
    input_img_data += grads_value * step#沿着让损失最大化的方向调节输入图像
#将张量转换为有效图像的实用函数
def deprocess_image(x):
     '''此处省略'''
#生成过滤器可视化的函数
def generate_pattern(layer_name, filter_index, size=150):
     '''此处省略'''
plt.imshow(generate_pattern('conv1', 0))
```

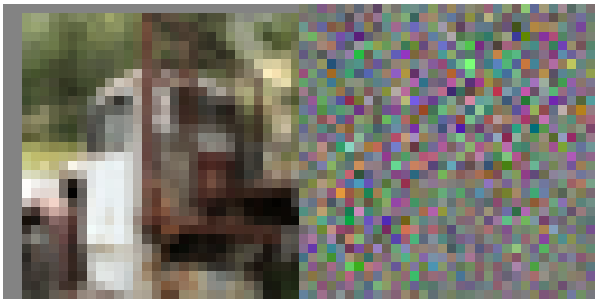

图19：conv1第0个通道具有最大响应的模式

六. 分析与总结

I.传统方法

- 在所有特征和模型的组合中，DAISY特征与SVM分类模型具有最高的分类准确率0.6427，而DAISY特征与伯努利朴素贝叶斯模型的分类准确率最低，仅有0.1。
- 从准确率的角度分析，在10种分类模型中，SVM与Xgboost两种模型的准确率较高，三种特征下的平均准确率达到了0.5811和0.5636，这在传统的机器学习方法中是比较高的。而决策树和伯努利朴素贝叶斯模型的表现较差，三种特征下的平均准确率仅为0.2760和0.2169，准确率较低表明此类连续数值的分类问题并不适合使用这两种模型。
- 从模型稳定性的角度分析，Adaboost这一集成学习方法具有较高的稳定性，对于三类不同的特征，模型的分类准确性都相差不大。反观其他分类模型，对于特征的选取都是较为敏感的，不同的特征提取方法将导致模型的分类准确性大不相同。
- 从特征提取的角度分析，HOG特征的平均分类准确率高于DAISY特征的平均分类准确率，而DAISY特征的平均分类准确率高于PCA的平均分类准确率。这一结果是可想而知的，因为HOG和DAISY都是提取了图片的结构内容上的特征，而PCA仅仅提取了像素值上的特征。基于图片内容来进行分类，分类效果显然是要好于PCA的。换一种说法，HOG和DAISY是看到了图片内容上的特征，而PCA仅仅看到了像素值。

II.深度学习

模型优化总结

- 数据归一化对原始数据进行线性变换，加快网络收敛速度。
- 数据增强技术使用翻转图像、切割图像、白化图像等方法增加数据量，增加数据量。
- 模型改进技术包括batch normalization、weight decay、dropout等防止过拟合，增加模型的泛化能力。
- 变化学习率通过在训练过程中递减学习率，使得模型能够更好的收敛，增加模型的拟合能力。
- 使用Deep_CNN和VGG网络架构加深网络层数，通过残差网络技术加深模型层数和解决梯度衰减问题，增加模型的性能。
- 这些改进方法的一步步堆叠递进，使得网络的拟合能力和泛化能力越来越强，最终在测试集上获得较高的分类准确率92.87%。

参数调整心得

- Dropout 。Dropout的放置位置以及大小非常重要，最后一层softmax 前用Dropout能够明显防止过拟合。
- 数据的shuffle 和augmentation 。shuffle设置为true会比false效果更好；因为迭代次数设置较小，augmentation 在本实验中对性能提升的效果不是很明显，且augmentation 是的计算量增加，训练时间大幅增长。
- 降学习率：随着网络训练的进行，学习率要逐渐降下来，Test4实验表明在学习率下降的一瞬间，网络有个巨大的性能提升。
- Batchsize：影响没那么大，设置为64与128差别不大。
- Relu+BN：Relu函数+batch normazation对性能提高相当明显。
- Early stop：在epoch设置较大时设置early stop回调函数可以节约训练时间。

- 优化函数：训练过程中使用了sgd、adam，相对来说adam不需要设置参数，收敛速度要快一些，但是训练结果相对于sgd+momentum要差一些。

III.传统方法与深度学习对比

- 传统机器学习方法需要从数据中学习到好的特征表达，大多数应用的特征都需要专家确定然后编码为一种数据类型，特征可以为像素值、形状、纹理、位置和方向。如果特征表达和特征提取方法选择不恰当，再好的算法也是徒劳，因此，往往需要人工投入大量时间去研究和调整特征工程，决定哪一个特征可以区分不同类型的物体；深度学习的方法建立好模型后，可以自动学习特征和任务之间的关联，找到最具有描述性和明显的特征，从数据中直接获取高等级的特征。
- 传统的机器学习表达能力有限，一般训练时间短，对硬件性能的要求低。当数据量比较小的时候，用传统机器学习方法也许更合适；深度学习表达能力强，学习时间长，对硬件性能要求高但如果训练集很小，机器可能会过拟合训练数据，不能通用化这个任务；深度学习依赖于大量带标签的数据，才能够得到深层的、数据集特定的特征表示，如果标注数据集不够大，会导致网络性能差。

七、加分项

加分项1

要求：手写实现传统分类算法朴素贝叶斯分类，并设计实现基于最小风险的贝叶斯决策算法，应用于cifar10图片分类任务。具体代码见“附件8-手写Bayes.ipynb”

特征提取

实验首先对输入数据处理，提取特征。每个样本原始数据都是32 x 32 x 3的RGB图像， 即原始特征数目为3072，因此需要进行降维处理，常见的降维方法有奇异值分解(SVD)、主成分分析(PCA)、因子分析(FA)、独立成分分析(ICA)等，本次实验采用的是主成分分析法进行降维，并经过一系列的实验，得到最终维度数目为25时效果最优。

```
pca = PCA(n_components = feature_count)
new_train_data = pca.fit_transform(x_train)
```

模型训练

本次实验采用高斯朴素贝叶斯模型。朴素贝叶斯是指类条件概率中各个特征相互独立，$x_i(1 \leq k \leq K)$为样本的第k个特征，则类条件概率分布为

$$P(x_1,\dots,x_k,\dots x_K|\omega_i) = P(x_1|\omega_i)P(x_2|\omega_i)\dots P(x_k|\omega_i)\dots P(x_K|\omega_i)$$

高斯模型即假设样本的条件概率分布服从高斯分布。设为某样本的第k个特征，则其在某一类ω_i中的出现的条件概率为：

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma}exp(-\frac{(x-\mu)^2}{2\sigma^2})$$

由上述分析，在模型训练阶段，我们只需要求得第i类样本在第k个属性上的均值和方差即可，用矩阵分别表示为E和C，其中$E_{i,j}$和$C_{i,j}$分别表示第i类样本在第k个属性上的均值和方差。

```
# 计算类内均值 类内离散度 先验概率
def calculateEC(x_train, y_train, feature_count):
    # matrixE[i][j]表示第i类样本在第j个属性上的均值
    # matrixC[i][j]表示第i类样本在第j个属性上的方差
    matrixE = np.empty((10, feature_count))
    matrixC = np.empty((10, feature_count))
```

```
    pc = []  # 先验概率
    for i in range(10):
        listC = extractClass(y_train, new_train_data, i)
        pc.append(len(listC) / len(y_train))
        for j in range(feature_count):
            list1 = []
            for k in range(len(listC)):
                list1.append(listC[k][j])
            matrixE[i, j] = np.mean(list1)
            matrixC[i, j] = np.cov(list1)
    return matrixE, matrixC, pc,
```

样本测试

最小风险贝叶斯分类器依据条件风险R对样本进行分类。其中决策α_i指把样本x归到类ω_i中，损失$\lambda_{i,j}$指把真实属于类的ω_j样本x归到类ω_j中带来的损失，条件风险$R(\alpha_i|x)$指对样本x采取决策α_i后可能的总风险:

$$R(\alpha_i|x) = E[\lambda_{i,j}] = \sum_{j=1}^{c} \lambda_{i,j} P(\omega_i|x), i = 1, 2, \ldots, c$$

$$P(\omega_i|x) = \frac{P(x|\omega_i)P(\omega_i)}{P(x)}$$

综合上述两式可得:

$$R(\alpha_i|x) = \sum_{j=1}^{c} \lambda_{i,j} \frac{P(x|\omega_j)P(\omega_j)}{P(x)} = \sum_{j=1}^{c} \lambda_{i,j} \frac{\prod_{k=1}^{K} P(x|\omega_j)P(\omega_j)}{P(x)} = \sum_{j=1}^{c} \lambda_{i,j} \frac{\prod_{k=1}^{K} \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(x-\mu)^2}{2\sigma^2})P(\omega_j)}{P(x)}$$

$$R(\alpha_i|x)P(x) = \sum_{j=1}^{c} \lambda_{i,j} \prod_{k=1}^{K} \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(x-\mu)^2}{2\sigma^2})P(\omega_j)$$

$$i = 1, 2, \ldots c$$

> **分类决策规则** $x \in \omega_k, k = argmin_i R(\alpha_i|x) = argmin_i R(\alpha_i|x)P(x)$

```
# 根据最小风险贝叶斯模型进行分类
def calculatePostrior(matrixE, matrixC, pc, test_data, feature_count):
    risk = []
    for i in range(10):
        r = 0
        for j in range(10):
            p_x_wj = 1
            for k in range(feature_count):
                p_x_wj = p_x_wj * math.exp(- (test_data[k] - matrixE[j][k]) ** 2 / (2 *
matrixC[j][k]))
/ math.sqrt(2 * math.pi * matrixC[j][k])
            r += matrixLambda[i][j] * pc[i] * p_x_wj
        risk.append(r)
    return risk.index(min(risk))
```

参数调整

> **最小风险矩阵**

类别顺序为飞机、汽车、鸟、猫、鹿、狗、蛙、马、船、卡车，$\lambda_{i,j}$指把真实属于ω_j类的样本x归到ω_i类中带来的损失:

$$\lambda_a = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

此时基于最小风险的贝叶斯决策算法和基于最小错误的贝叶斯决策算法等价，最终在测试集上的准确率为0.3716。

若增大两个相差甚远的类别之间误判的风险，即把将交通工具误判为动物的风险加大，得到如下风险矩阵：

$$\lambda_b = \begin{pmatrix} 0 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 2 \\ 2 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 \\ 2 & 1 & 0 & 2 & 2 & 2 & 2 & 2 & 1 & 1 \\ 1 & 1 & 2 & 0 & 2 & 2 & 2 & 2 & 1 & 1 \\ 1 & 1 & 2 & 2 & 0 & 2 & 2 & 2 & 1 & 1 \\ 1 & 1 & 2 & 2 & 2 & 0 & 2 & 2 & 1 & 1 \\ 1 & 1 & 2 & 2 & 2 & 2 & 0 & 2 & 1 & 1 \\ 1 & 1 & 2 & 2 & 2 & 2 & 2 & 0 & 1 & 1 \\ 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 2 \\ 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 0 \end{pmatrix}$$

此时在测试集上的准确率为0.3014，准确率不升反降。

若增大两个相差较近的类别之间误判的风险，即把将交通工具误判为另一种交通工具的风险加大，即加大困难样本的风险，得到如下风险矩阵：

$$\lambda_c = \begin{pmatrix} 0 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 1 & 1 \\ 1 & 0 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & 1 \\ 1 & 2 & 0 & 1 & 1 & 1 & 1 & 1 & 2 & 2 \\ 2 & 2 & 1 & 0 & 1 & 1 & 1 & 1 & 2 & 2 \\ 2 & 2 & 1 & 1 & 0 & 1 & 1 & 1 & 2 & 2 \\ 2 & 2 & 1 & 1 & 1 & 0 & 1 & 1 & 2 & 2 \\ 2 & 2 & 1 & 1 & 1 & 1 & 0 & 1 & 2 & 2 \\ 2 & 2 & 1 & 1 & 1 & 1 & 1 & 0 & 2 & 2 \\ 1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 0 & 1 \\ 1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 1 & 0 \end{pmatrix}$$

此时在测试集上的准确率为0.3692，准确率有所回升。

> **PCA特征个数**

特征数目	20	25	30	35	40	50	60	80	100
测试集精确度	0.3651	0.3716	0.3684	0.3668	0.3669	0.3485	0.3526	0.3438	0.3368

可知，随着特征数目的增加，精确度先增加后减少，这说明特征数目不是越多越好，特征数目越多，就越可能造成过拟合，反而在训练集的表现不好，这样验证了奥卡姆剃刀原理。

加分项4

要求：自主设计搭建合适的深度卷积神经网络，说明设计思路，也可优化改进已有的网络模型结构。

优化改进Simple_CNN，搭建myDeep_CNN，将测试集上的准确率从61.49%改进到88.02%，具体代码见“附件10-myDeep_CNN.ipynb”。采用的优化策略及相应代码如下。

权重衰减

虽然通过数据增强增大训练数据集可能会减轻过拟合，但是获取额外的训练数据往往代价高昂。L2范数正则化通过为模型损失函数添加惩罚项使学出的模型参数值较小，是应对过拟合的常用手段。

```
weight_decay = 0.0005
model.add(Conv2D(32, (3, 3), padding='same',kernel_regularizer=regularizers.l2(weight_decay)))
```

Relu+BN

Relu激活函数是大多数前馈神经网络默认使用的激活函数，相比于 sigmoid 函数和 tanh 函数计算速度快得多。Batch Normalization带来很多优点：可以使用更高的学习率、移除或使用较低的dropout、降低L2权重衰减系数、取消LRN等。

```
model.add(Activation('relu'))
model.add(BatchNormalization())
```

Dropout

Dropout可以比较有效的缓解过拟合的发生，在一定程度上达到正则化的效果。这种方式可以减少特征检测器（隐层节点）间的相互作用，检测器相互作用是指某些检测器依赖其他检测器才能发挥作用。其原理是：我们在前向传播的时候，让某个神经元的激活值以一定的概率p停止工作，这样可以使模型泛化性更强，因为它不会太依赖某些局部的特征。

```
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Dropout(0.25))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

数据规范化

比较归一化和标准化的性能，选取最好的规范化方法。

```
#标准化
def standardization(x_train,x_test):
    mean = np.mean(x_train,axis=(0,1,2,3))
    std = np.std(x_train, axis=(0, 1, 2, 3))
    x_train = (x_train-mean)/(std+1e-7)
    x_test = (x_test-mean)/(std+1e-7)
    return x_train, x_test

#归一化
def normalize(x_train,x_test):
    x_train_max=np.max(x_train),x_train_min=np.min(x_train)
    x_test_max=np.max(x_test),x_test_min=np.min(x_test)
    x_train=(x_train-x_train_min)/(x_train_max-x_train_min)
    x_test=(x_test-x_test_min)/(x_test_max-x_test_min)
    return x_train, x_test
```

动态学习率

动态学习率在训练过程中递减学习率，使得模型能够更好的收敛，增加模型的拟合能力。

```
  def lr_scheduler(epoch):
    return learning_rate * (0.5 ** (epoch // lr_drop))
  reduce_lr = keras.callbacks.LearningRateScheduler(lr_scheduler)
```

优化函数SGD+Momentum

当训练数据很大时，常用的方法是计算训练集中的小批量，这就是SGD。和学习率随着时间退火类似，Momentum 随时间变化的设置有时能略微改善最优化的效果。

```
#优化器设置
sgd = optimizers.SGD(lr=learning_rate, decay=lr_decay, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd,metrics=['accuracy'])
```

提前终止策略

如果epoch数量太多，则有可能发生过拟合，使用EarlyStopping解决epoch数量需要手动设置的问题，被视为一种能够避免网络发生过拟合的正则化方法，我们设置为监视测试集上的准确度，当20次训练结果没有提升时终止训练。

```
  earlystop=keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=20, verbose=1,
mode='auto')
```