《**数据挖掘技术**》实验考核表

实验名称： 实验四：数据挖掘算法（4）- 综合实验

2020级 专业： 计算机科学与技术

实验时间： 2023年5月18日 3-4节中午 实验地点： 7-212

表一

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 学号 | 姓名 | 班级 | 实验摘要与总结（学生填写） | 成绩 |
| **2020** |  | 计算机8班 | 在本实验中，我尝试了利用SVM、MLP、卷积神经网络三种模型对CIFAR-10数据集进行图像分类，更深刻地感受了机器学习中模型选择和参数调试的思想。 |  |

表二

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 考核标准 | 实验过程  30分 | 实验报告  30分 | 程序运行  20分 | 交流合作  10分 | 考勤纪律  10分 |
| 各项成绩 |  |  |  |  |  |
| 考核内容 | 评价实验课堂中的综合表现，包括实验态度、实验过程等内容。 | ○完整  ○较完整  ○一般  ○内容极少 | ○运行正常  ○功能不全  ○有小错  ○无法运行 | ○良好  ○一般  ○较差 | ○良好  ○一般  ○较差 |

表三

|  |
| --- |
| 其它批改意见 |
|  |

注：

* 表1除“成绩”栏外由学生填写，表2、表3由指导教师填写。
* 本页附在实验报告正文前装订，纸质版报告可单双面打印，电子报告Word排版。
* **实验报告如发现雷同，一律以0分计。**

**一、实验目的**

1. 理解和对比各类数据挖掘技术的思想，可以使用所掌握的各类数据挖掘算法对要解决的问题进行综合分析；能够提出特定任务的解决方法；
2. 深入理解不同数据挖掘技术和工具应用条件；
3. 能够对数据挖掘结果进行综合分析和对比。
   1. **实验内容**
4. 数据集来源：

Cifar-10 / Cifar-100 数据二选一： <https://www.cs.toronto.edu/~kriz/cifar.html>

1. 任选三种不同类型的数据挖掘技术，方法不限；对数据集进行分类，可以进行数据预处理；
2. 使用两种效果评价方法，通过图表或其它方式比较不同算法测试结果，提出较优的算法；
3. 对比 <https://benchmarks.ai> 显示的 Accuracy，计算与 Top2 的差距。
4. **（选做）**比较算法执行的时间；分析算法的计算复杂度。

**三、实验要求**

实验报告中体现全部基本实验内容.，至少包括：

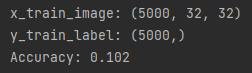
1. 实现基本的实验内容；
2. 有数据处理结果的分析，有结论；

**四、实验记录**

【支持向量机】

（代码参见[附录1.1](#支持向量机)）

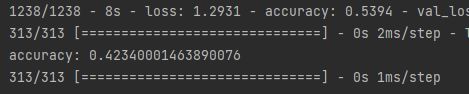
这里使用了一个RBF核的支持向量机，经过在少量数据集上的最优参数搜索后，我设定其参数为C = 1，gamm = 0.01。训练集为5000，测试集为1000。其最好的训练结果如下所示，准确率仅有0.102，几乎可以看作是随机猜测：

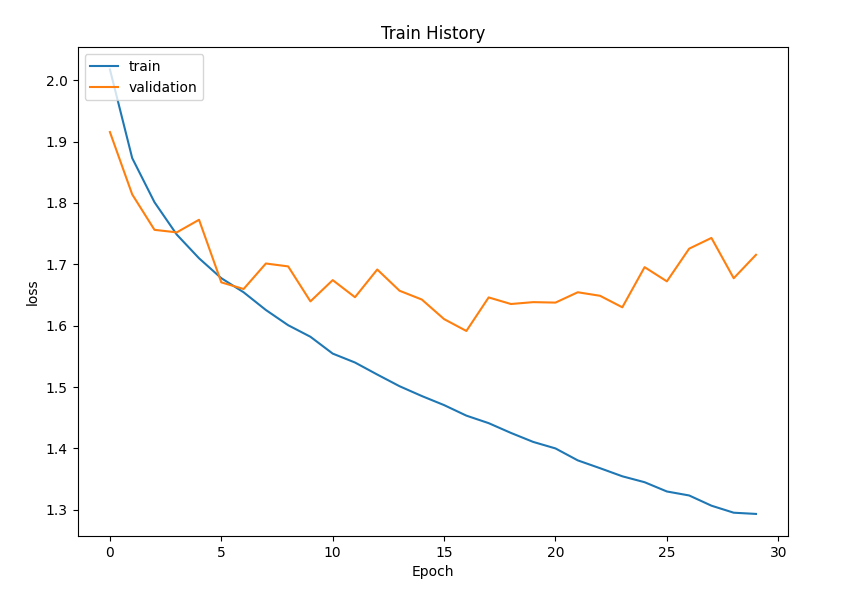
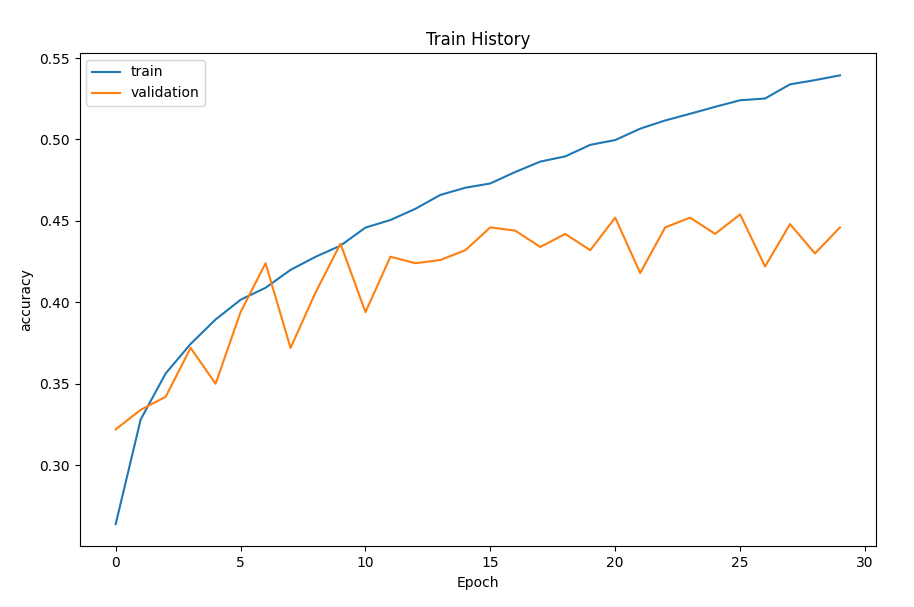


【MLP多层感知机】

（代码参见[附录1.2](#多层感知机)）

这里使用了一个简易的MLP，其核心为一个具有1024神经元的输入层，以及三个分别具有512、256、128神经元的隐藏层。训练集使用了全部5万个，测试集1万。其最好的训练结果如下所示，训练轮数30轮，准确率0.423，显著优于SVM。





【卷积神经网络】

（代码参见[附录1.3](#卷积神经网络)）

这里使用了一个简易的卷积神经网络，其核心为三层卷积层。训练使用了全部5万个训练集和1万个测试集。其最好的训练结果如下所示，训练轮数20轮，测试集准确率0.685，相较于MLP再次有了较大提升：

E-pocn 20/20 
400/400 [ 
313/313 [ 
: 0.6858000159263611 
accuracy 
--] - 129s 323ms/step - 
- 9s 28ms/step - loss 

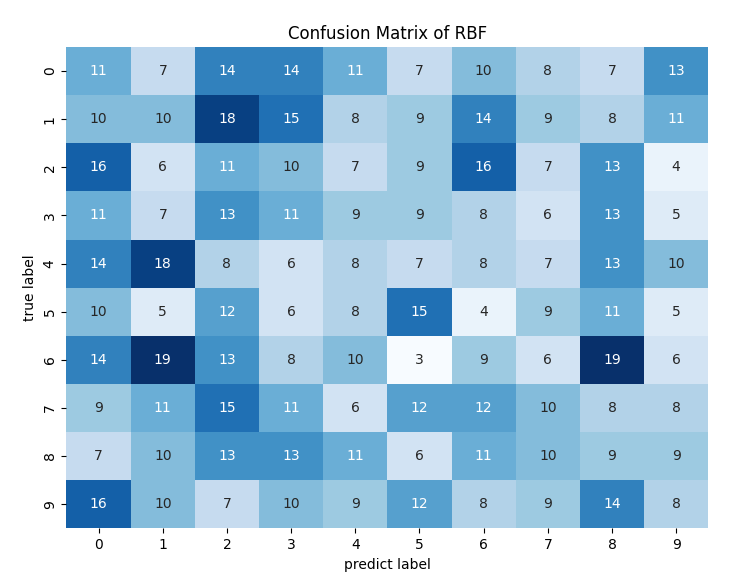
0.8 
0.6 
0.4 
train 
validation 
0.0 
2.5 
5.0 
Train History 
10.0 
Epoch 
12.5 
15.0 
17.5 1.8 
1.6 
1.4 
1.2 
1.0 
0.8 
0.6 
train 
validation 
0.0 
2.5 
5.0 
Train History 
10.0 
Epoch 
12.5 
15.0 
17.5 

我采用了准确率和混淆矩阵两种方式来对比三种模型的预测能力。

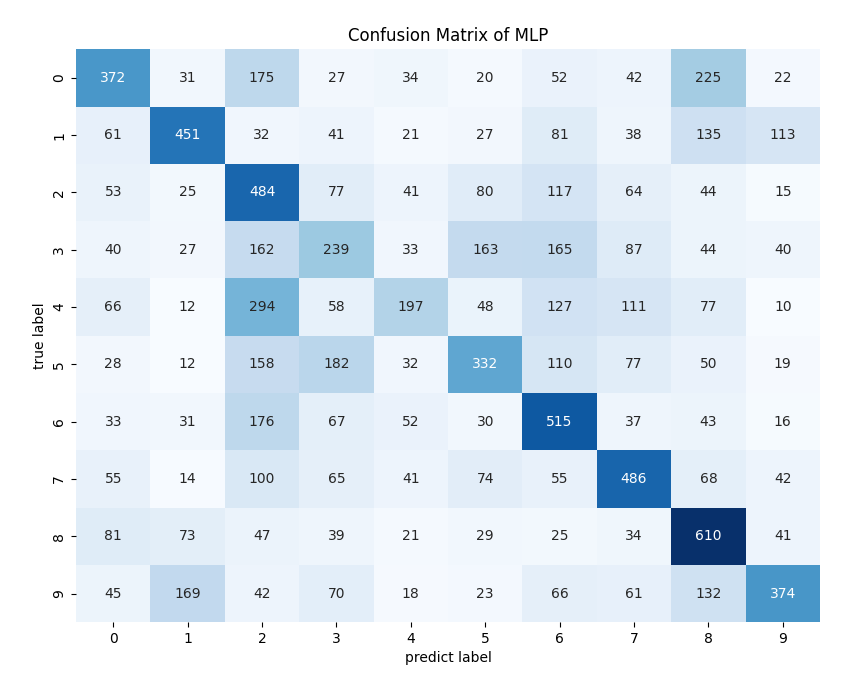
通过对比准确率，最优的模型显然是卷积神经网络，其准确率在三种测试中达到了最高的0.68。并且从原理上看，卷积运算也是最适合图像识别的算法。而最差的模型为SVM，其准确率0.102几乎等效于随机猜测。

以下是三种模型的混淆矩阵：

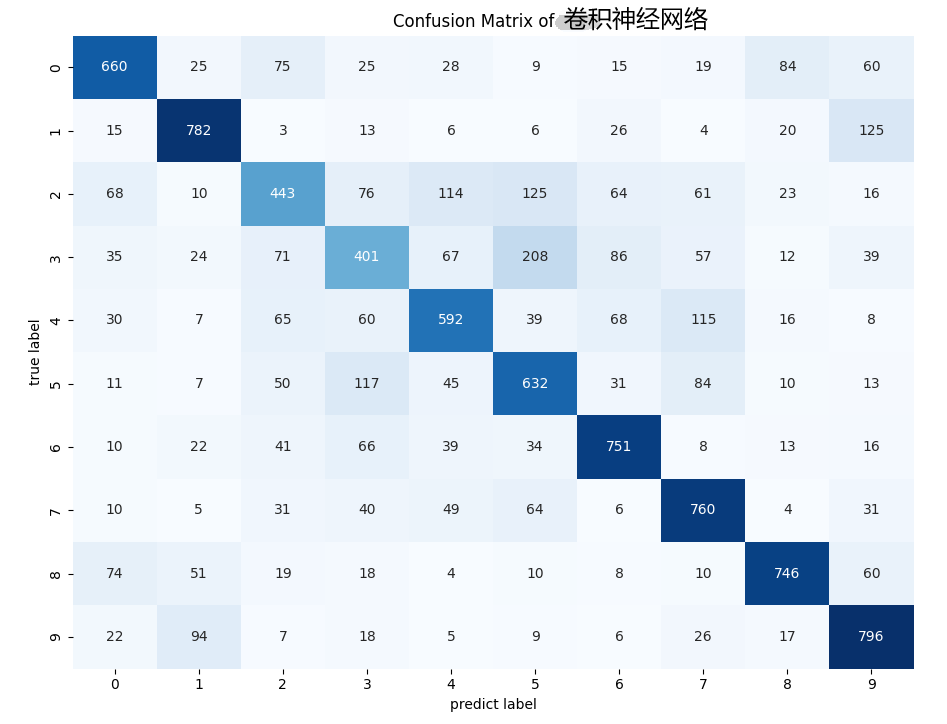
SVM的混淆矩阵非常糟糕，其预测结果并没有收敛于任何区域。



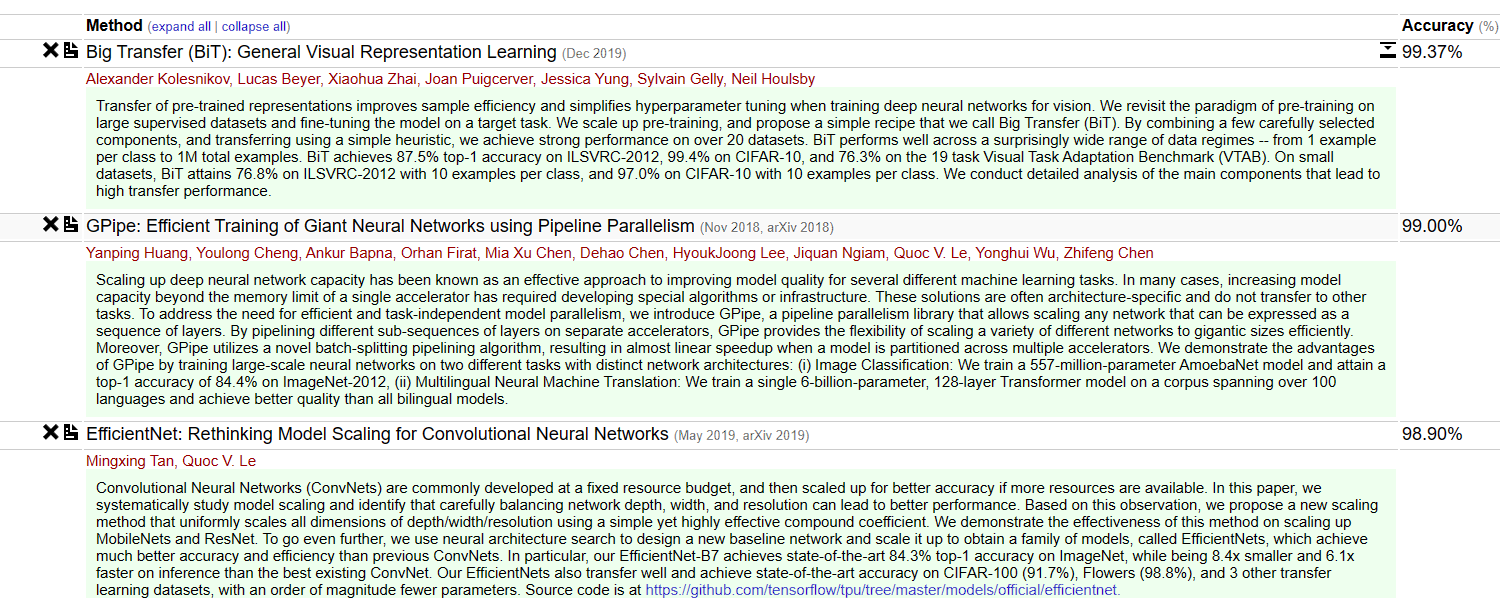
从MLP的混淆矩阵中可以看出，其对船只（label=8）的识别比较擅长，而对标签4也即鹿的识别是最差的，它大量的把鹿（4）识别为了鸟（2），而对于猫（3）和狗（5）也具有严重的混淆，甚至在识别猫的图片时，判断为狗的有163张，比正确判断为猫的162张还要多。



从卷积神经网络的混淆矩阵可以看出，其对各种数据的识别都比较优秀。尤其是对卡车（9）的识别最好，而对猫（3）的识别最差，其将208张猫（3）错误地识别为了狗（5），而其在识别狗（5）时，误判为猫（3）的数量也是最多的，达到了117张。



在给定网站中，前几名的准确率均达到了0.98-0.99。而我在实验中所达到的准确率最高也只有0.68，可见差距十分巨大。



1. (选做)

**五、实验总结**

本实验的主要任务可以概括为“用三种方法分类CIFAR-10数据集”。由于CIFAR-10数据集的整个规模比较大，因此我在实验中根据情况，有时训练全部五万的训练集，有时选择其中一个规模为一万的子集。

我利用CIFAR-10数据集官方文档提供的unpickle方法，提取出我所需要的数据集，同时由于CIFAR-10的数据集均为彩色图像，因此我将其全部转换为灰度图像以降低训练耗时。



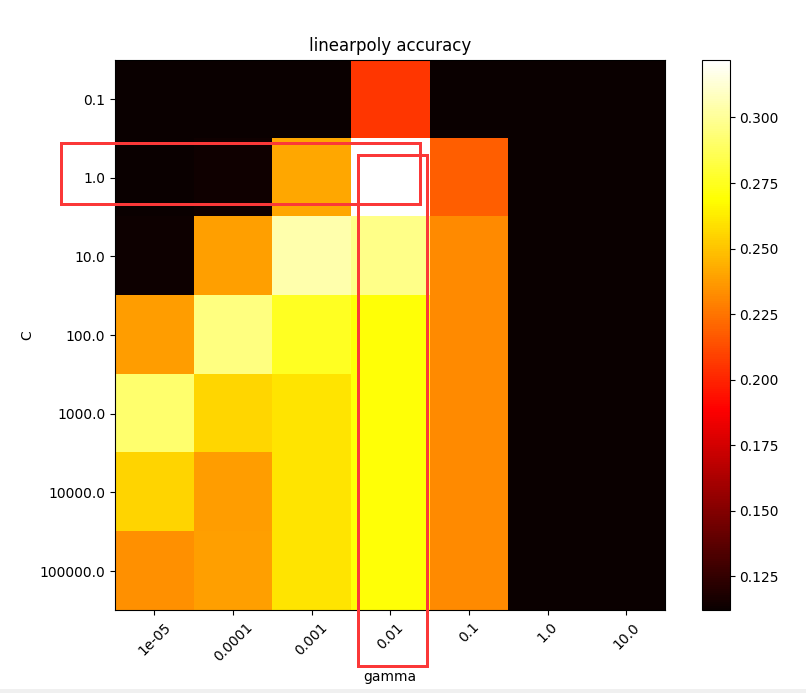
在整个实验中，我选用了支持向量机、MLP、卷积神经网络三种模型进行训练。

【SVM支持向量机】

由于支持向量机本身没有办法处理太高维的数据，因此我将数据降至2维。在这里我选用了RBF核。同时由于训练集过大，训练耗时极长，因此我选用了前5000个训练集和前1000个测试集进行训练，在不经过任何最优参数搜索的情况下，准确率只有极为糟糕的0.096：



在1000个样本上进行最优参数搜索之后，我得到了最优参数C = 1；gamma = 0.01。下为最优参数搜索的热力图，最优参数的交叉验证得分达到了约0.32。



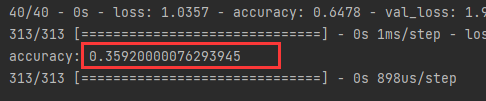
然而使用最优参数创建模型并再次进行训练后，也只得到了0.102的准确率。可见在低数据量上搜索得到的参数并不一定适用于高数据量，而其糟糕的准确率也表明：对于本数据集来说，SVM的结构过于简单了。

【MLP多层感知机】

在本学期数字图像课程的实验中，我们使用了一个简易的多层感知器来进行英文字母的识别。由于有了一定的使用经验，因此我对其进行调试后用于本次实验的测试。

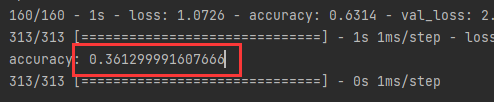
本MLP的结构较为简单，仅有一个神经元数量为256的隐藏层。

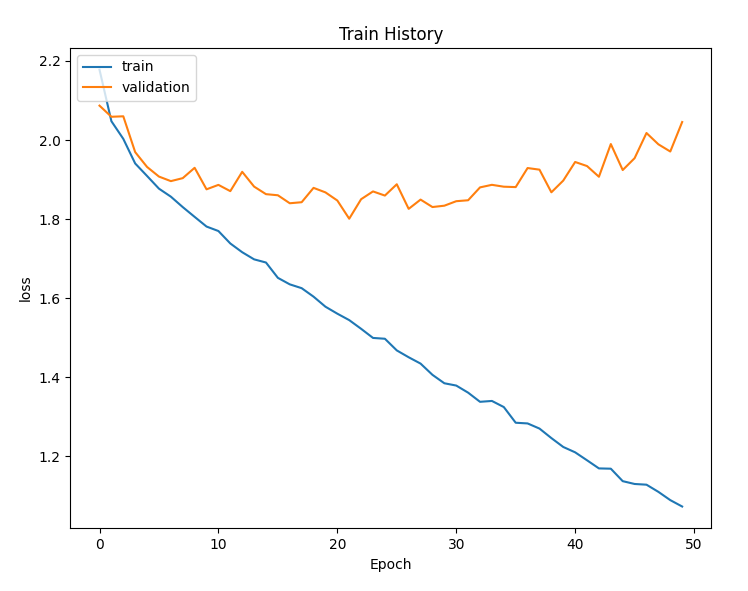
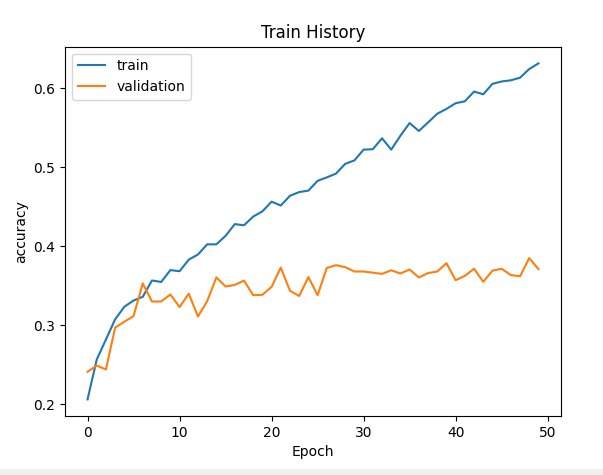
经过80轮的训练后，可以看到识别率为0.359，并不是很理想。



原本MLP识别的英文字母是一种形状较为简单的图像，而对于CIFAR-10这种复杂图案的识别出现较差的表现也在预料之中。因此我尝试通过增加MLP的复杂度的方式提升准确率：

以下是将MLP的神经元数量从256提升至512，将模型每次训练时使用的数据量batch\_size由200调整为50以提升泛化能力之后的训练结果，可以看到，准确率仅有微乎其微的提升：





从上图中可以看到，虽然随着训练轮数的提升，模型在训练集上的表现一路上涨，但是在验证集上的表现却踌躇不前。并且根据误差率图像可以看到，在训练第25轮时就出现了过拟合。

在这之后我又尝试了增加隐藏层、增大神经元数量等。但是除了训练的耗时不断增加外，仍然没有明显效果。并且在训练的过程中快速地出现了过拟合现象。

最后我尝试将全部的5万个训练集用作模型训练，得到了实验记录中的训练结果。虽然经过查阅资料发现如果对数据进行合适的预处理能够达到较高的准确率，但是关于各种预处理的方式确实感到有些超出了我目前的能力范围，暂且就此作罢。

【卷积神经网络】

由于MLP对于特征的提取是基于单个像素点的，因此更适合于对表格类数据进行识别。而卷积神经网络的卷积核是一个n\*n的模板，因此其更适合对于图像这种具有纵深区域性特征的数据的识别。

通过查阅资料以及结合对MLP的使用经验，我搭建了一个简易的卷积神经网络，其拥有3\*3的模板，一个32核的卷积层和一个64核的卷积层。其训练相较于先前的MLP而言极为耗时，但是取得了较好的训练成果：

可以看到，模型仅仅经过10轮的训练就达到了0.643的测试集准确率，并且通过损失函数图像可以看出，模型尚未过拟合，其准确率仍有提升空间。

313/313 [ 
----] - 114 
313/313 [ 
- 2s 7ms/ 
: 0.6431000232696533 
accuracy 

Train History 
0.65 
train 
validation 
0.60 
0.55 
0.50 
0.45 
0.40 
0.35 
Epoch Train History 
train 
validation 
1.8 
1.6 
1.4 
1.2 
1.0 
Epoch 

最终我为模型增加了一个128核卷积层（参见附录1.3），再将训练轮数提升至20轮后，准确率达到了迄今最高的0.685，同时并未出现过拟合情况，也即为实验记录中的效果。根据准确率图，虽然仍然有微小的提升空间，但是在训练结果具有一定不确定性的情况下也可忽略不计。

在本实验中，通过尝试三种模型，我更深刻地体会到了机器学习中模型选择和参数调试的重要性。我最大的感受就是，更复杂的模型确实能够承载更复杂的任务。而参数的调整更是一个极为繁琐、耗时、艰深的工程。

训练模型是极为耗时的，而在这种基础上寻找最优的参数更加耗时，更遑论要寻找多个参数的最优组合，在真正的机器学习前沿领域更是需要惊人的资金、时间、算力支持。

整个过程当中，我为了寻找较优的参数或模型结构，不得不总是降低数据量。然而在低数据量上获得的最优解也并不总是在高数据量上同样有效。但即便如此其所耗费的训练时间也是惊人的。这就意味着在机器学习的学习过程中，学习前人的调参经验，尝试前人得出的优秀参数，使用前人已经训练好的优秀模型能极大的节约学习成本。

而随着时代的进步，模型越来越复杂，参数需求越来越大。这可能意味着个人或者缺乏资源支持的小团队单打独斗所能取得的成果愈发有限。而观察当前正在各种大型科技企业在AI领域的巨大投入就能推断出，资金充沛的工业界恐怕会成为推动AI进步的前沿阵地。

**六、代码附录**

**【1.1支持向量机】**

from matplotlib import pyplot as plt

from sklearn import svm

from sklearn.metrics import accuracy\_score, confusion\_matrix

import numpy as np

import pickle

import seaborn as sns

from sklearn.model\_selection import GridSearchCV

from sklearn.preprocessing import StandardScaler, MinMaxScaler

def unpickle(file):

with open(file, 'rb') as fo:

dict = pickle.load(fo, encoding='bytes')

return dict

def load\_data(file):

dict\_data = unpickle(file)

x = np.array(dict\_data[b'data']).reshape(-1, 3, 32, 32).transpose(0, 2, 3, 1)

x = np.mean(x, axis=-1)

y = np.array(dict\_data[b'labels'])

return x, y

def load\_all\_data(batches):

x\_all = []

y\_all = []

for batch in batches:

x, y = load\_data(batch)

x\_all.append(x)

y\_all.append(y)

return np.concatenate(x\_all), np.concatenate(y\_all)

batches = ['data\_batch\_1', 'data\_batch\_2', 'data\_batch\_3', 'data\_batch\_4', 'data\_batch\_5']

x\_train\_image, y\_train\_label = load\_all\_data(batches)

x\_test\_image, y\_test\_label = load\_data('test\_batch')

x\_train\_image = x\_train\_image[:5000]

y\_train\_label = y\_train\_label[:5000]

x\_test\_image = x\_test\_image[:1000]

y\_test\_label = y\_train\_label[:1000]

# 输出训练数据和测试数据的数量

print('train data=', len(x\_train\_image))

print('test data=', len(x\_test\_image))

# 输出训练集的维度大小和测试集对应标签的维度大小

# 训练集维度:（数据集数量，图片行像素大小，图片列像素大小）

# 测试集标签维度:（测试集标签数量，）

print('x\_train\_image:', x\_train\_image.shape)

print('y\_train\_label:', y\_train\_label.shape)

# 降维

x\_train\_image = x\_train\_image.reshape((len(x\_train\_image), -1))

x\_test\_image = x\_test\_image.reshape((len(x\_test\_image), -1))

# 归一化

scaler = MinMaxScaler()

x\_train\_image = scaler.fit\_transform(x\_train\_image)

x\_test\_image = scaler.transform(x\_test\_image)

# # 最优参数搜索

# param\_grid\_RBF = {'C': np.logspace(-1, 5, 7),

# 'gamma': np.logspace(-5, 1, 7)}

# grid\_RBF = GridSearchCV(svm.SVC(kernel='rbf'), param\_grid\_RBF, cv=5)

# grid\_RBF.fit(x\_train\_image, y\_train\_label)

# scores = grid\_RBF.cv\_results\_['mean\_test\_score'].reshape(len(param\_grid\_RBF['C']), len(param\_grid\_RBF['gamma']))

# print('RBF核的最佳参数为：', grid\_RBF.best\_params\_['C'], grid\_RBF.best\_params\_['gamma'])

# print('RBF核的得分矩阵：', scores)

# # 绘制热力图

# def heatmap(modelname, scores, param\_grid):

# # 使用 matplotlib 绘制准确率热力图

# plt.figure(figsize=(8, 6))

# plt.imshow(scores, interpolation='nearest', cmap=plt.cm.hot)

# plt.xlabel('gamma')

# plt.ylabel('C')

# plt.colorbar()

# plt.xticks(np.arange(len(param\_grid['gamma'])), param\_grid['gamma'], rotation=45)

# plt.yticks(np.arange(len(param\_grid['C'])), param\_grid['C'])

# plt.title(modelname + 'poly accuracy')

# plt.show()

# print('ok')

#

# heatmap('linear', scores, param\_grid\_RBF)

# 创建和训练模型

clf = svm.SVC(kernel='rbf', gamma=0.01, C=1)

clf.fit(x\_train\_image, y\_train\_label)

# 观察准确率

y\_pred = clf.predict(x\_test\_image)

print('Accuracy:', accuracy\_score(y\_test\_label, y\_pred))

#

# 绘制混淆矩阵

def con\_matrix(name, model):

y\_pred = model.predict(x\_test\_image)

cm = confusion\_matrix(y\_test\_label, y\_pred)

plt.figure(figsize=(7, 5))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.xlabel('predict label')

plt.ylabel('true label')

plt.title('Confusion Matrix of ' + name)

plt.show()

con\_matrix('RBF', clf)

**【1.2MLP多层感知机】**

# 1.数据预处理

import numpy as np

import matplotlib.pyplot as plt

from keras.utils import np\_utils

from keras.models import Sequential

from keras.layers import Dense

# 设置seed可以产生的随机数据

np.random.seed(10)

def unpickle(file):

import pickle

with open(file, 'rb') as fo:

dict = pickle.load(fo, encoding='bytes')

return dict

def load\_data(file):

# 解析数据集文件

dict\_data = unpickle(file)

# 图片数据，将其转换成32x32x3的形式

x = np.array(dict\_data[b'data']).reshape(-1, 3, 32, 32).transpose(0, 2, 3, 1)

x = np.mean(x, axis=-1)

y = np.array(dict\_data[b'labels'])

return x, y

def load\_all\_data(batches):

x\_all = []

y\_all = []

for batch in batches:

x, y = load\_data(batch)

x\_all.append(x)

y\_all.append(y)

return np.concatenate(x\_all), np.concatenate(y\_all)

batches = ['data\_batch\_1', 'data\_batch\_2', 'data\_batch\_3', 'data\_batch\_4', 'data\_batch\_5']

x\_train\_image, y\_train\_label = load\_all\_data(batches)

# 获取训练集和测试集

# x\_train\_image, y\_train\_label = load\_data('data\_batch\_1')

x\_test\_image, y\_test\_label = load\_data('test\_batch')

# 以reshape转化成784个float（测试图片大小为28\*28=784）

x\_Train = x\_train\_image.reshape(len(x\_train\_image), 1024).astype('float32')

x\_Test = x\_test\_image.reshape(len(x\_test\_image), 1024).astype('float32')

# 将features标准化

x\_Train\_normalize = x\_Train / 255

x\_Test\_normalize = x\_Test / 255

# 将训练数据和测试数据的label进行one-hot encoding转化

y\_Train\_OneHot = np\_utils.to\_categorical(y\_train\_label)

y\_Test\_OneHot = np\_utils.to\_categorical(y\_test\_label)

# 2.建立模型、建立输入层、隐藏层

model = Sequential()

model.add(Dense(units=1024, input\_dim=1024, kernel\_initializer='he\_normal', activation='relu'))

model.add(Dense(units=512, kernel\_initializer='he\_normal', activation='relu'))

model.add(Dense(units=256, kernel\_initializer='he\_normal', activation='relu'))

model.add(Dense(units=128, kernel\_initializer='he\_normal', activation='relu'))

model.add(Dense(units=10, kernel\_initializer='normal', activation='softmax'))

# 对训练模型进行设置，损失函数、优化器、权值

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

# 设置训练与验证数据比例，80%训练，20%测试，执行10个训练周期，每一个周期200个数据，显示训练过程2次

train\_history = model.fit(x=x\_Train\_normalize, y=y\_Train\_OneHot, validation\_split=0.01, epochs=20,

batch\_size=50, verbose=2)

# 显示训练过程

def show\_train\_history(train\_history, train, validation):

plt.plot(train\_history.history[train])

plt.plot(train\_history.history[validation])

plt.title('Train History')

plt.ylabel(train)

plt.xlabel('Epoch')

plt.legend(['train', 'validation'], loc='upper left') # 显示左上角标签

plt.show()

# 画出准确率评估结果

show\_train\_history(train\_history, 'accuracy', 'val\_accuracy')

# 画出误差执行结果

show\_train\_history(train\_history, 'loss', 'val\_loss')

# 以测试数据评估模型准确率

# 创建变量存储评估后的准确率数据,（特征值，真实值）

scores = model.evaluate(x\_Test\_normalize, y\_Test\_OneHot)

# 输出预测准确率

print('accuracy:', scores[1])

**【1.3卷积神经网络】**

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten

from keras.layers import Conv2D, MaxPooling2D

from keras.utils import np\_utils

import numpy as np

import matplotlib.pyplot as plt

def unpickle(file):

import pickle

with open(file, 'rb') as fo:

dict = pickle.load(fo, encoding='bytes')

return dict

# 读取一份1w规模的数据集

def load\_data(file):

# 解析数据集文件

dict\_data = unpickle(file)

# 图片数据，将其转换成32x32x3的形式

x = np.array(dict\_data[b'data']).reshape(-1, 3, 32, 32).transpose(0, 2, 3, 1)

x = np.mean(x, axis=-1)

y = np.array(dict\_data[b'labels'])

return x, y

# 读取所有5W的数据集！

def load\_all\_data(batches):

x\_all = []

y\_all = []

for batch in batches:

x, y = load\_data(batch)

x\_all.append(x)

y\_all.append(y)

return np.concatenate(x\_all), np.concatenate(y\_all)

batches = ['data\_batch\_1', 'data\_batch\_2', 'data\_batch\_3', 'data\_batch\_4', 'data\_batch\_5']

x\_train\_image, y\_train\_label = load\_all\_data(batches)

# 获取训练集和测试集

# x\_train\_image, y\_train\_label = load\_data('data\_batch\_1')

# 获取所有5W个训练集和1K个测试集！

x\_test\_image, y\_test\_label = load\_data('test\_batch')

# 输出训练数据和测试数据的数量

print('train data=', len(x\_train\_image))

print('test data=', len(x\_test\_image))

# 输出训练集的维度大小和测试集对应标签的维度大小

# 训练集维度:（数据集数量，图片行像素大小，图片列像素大小）

# 测试集标签维度:（测试集标签数量，）

print('x\_train\_image:', x\_train\_image.shape)

print('y\_train\_label:', y\_train\_label.shape)

# 将features标准化

x\_Train\_normalize = x\_train\_image.reshape(len(x\_train\_image), 32, 32, 1).astype('float32') / 255

x\_Test\_normalize = x\_test\_image.reshape(len(x\_test\_image), 32, 32, 1).astype('float32') / 255

# 输入图片的维度，由于进行了灰度化，所以这里最后是1而非3

input\_shape = (32, 32, 1)

# 类别数

num\_classes = 10

# 将类别向量转换为二进制类别矩阵，这里是10种类别，以及训练集标签，训练数据和测试数据的label进行one-hot encoding转化

y\_Train\_OneHot = np\_utils.to\_categorical(y\_train\_label, num\_classes)

y\_Test\_OneHot = np\_utils.to\_categorical(y\_test\_label, num\_classes)

# 模型创建，并添加第一个卷积层，其中的模板为3x3，有32个卷积核；添加第二个卷积层，有64个卷积核

# 添加最大池化层、添加Dropout防止过拟合、将二维数据展平为一维数据

# 添加全连接层、添加Dropout防止过拟合、 添加输出层

model = Sequential()

model.add(Conv2D(32, (3, 3),

activation='relu',

input\_shape=input\_shape))

model.add(Conv2D(64, (3, 3), activation='relu'))

# model.add(Conv2D(128, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(num\_classes, activation='softmax'))

# 编译并训练模型

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

train\_history = model.fit(x\_Train\_normalize, y\_Train\_OneHot, validation\_split=0.2, batch\_size=100, epochs=30, verbose=1)

# 显示训练过程

def show\_train\_history(train\_history, train, validation):

plt.plot(train\_history.history[train])

plt.plot(train\_history.history[validation])

plt.title('Train History')

plt.ylabel(train)

plt.xlabel('Epoch')

plt.legend(['train', 'validation'], loc='upper left') # 显示左上角标签

plt.show()

# 画出准确率、误差的评估结果

show\_train\_history(train\_history, 'accuracy', 'val\_accuracy')

show\_train\_history(train\_history, 'loss', 'val\_loss')

# 创建变量存储评估后的准确率数据,（特征值，真实值）

scores = model.evaluate(x\_Test\_normalize, y\_Test\_OneHot)

# 输出预测准确率

print('accuracy:', scores[1])