

南開大學
Nankai University



《课程实验-深度学习》
人工智能导论第二次作业

题 目：	课程实验-深度学习
上课时间：	周三下午
授课教师：	张玉志
姓 名：	张怡桢
学 号：	2013747
年 级：	2020级本科生
日 期：	2022/12/30

课程实验-深度学习

张怡桢，2013747

南开大学软件学院

摘要：我们知道，目前，深度学习十分热门，深度学习在搜索技术，数据挖掘，机器学习，机器翻译，自然语言处理，多媒体学习，语音，推荐和个性化技术，以及其他相关领域都取得了很多成果。深度学习使机器模仿视听和思考等人类的活动，解决了很多复杂的模式识别难题，使得人工智能相关技术取得了很大进步。从广义上来说，NN（或是更美的DNN）可以认为包含了CNN、RNN这些具体的变种形式。本次实验，我使用tensorflow实现深度学习的三种神经网络：FCNN，CNN，RNN来实现CIFAR-10数据集的分类，同时使用机器学习的支持向量机SVM作为对比算法，对预测结果进行分析。

关键词：CNN FCNN RNN SVM 深度学习 机器学习 图像分类

1 实验目的说明

2 数据集详情——CIFAR-10数据集

2.1 数据集内容

2.2 数据集结构

2.3 数据集使用

3 深度神经网络

3.1 全连接网络(fully connected neural network, FCNN)

3.1.1 说明

3.1.2 神经网络

3.1.3 完整实现

3.1.4 输出结果

3.2 卷积神经网络 (Convolutional Neural Network, CNN)

3.2.1 说明

3.2.2 神经网络

3.2.3 完整实现

3.2.4 输出结果

3.3 循环神经网络 (Recurrent neural network, RNN)

3.3.1 说明

3.3.2 神经网络

3.3.3 完整实现

3.3.4 输出结果

4 机器学习分类算法——支持向量机SVM

4.1 说明

4.2 实现代码

4.3 输出结果

4.3.1 输出可视化

4.3.2 参数输出分析

5 分类结果对比分析

5.1 随机图片预测

5.2 损失与ACC

6 附件

1 实验目的说明

分别使用全连接网络，卷积神经网络，循环神经网络去预测数据（图像分类）CIFAR-10，使用几个机器学习算法（自选）作为比较，对预测结果进行分析。（深度学习框架推荐使用 tensorflow、keras 或 pytorch，编程语言为 python）

在本次实验中，我的深度学习框架使用的是 tensorflow与keras，对比的机器学习算法选择SVM支持向量机。

2 数据集详情——CIFAR-10数据集





































































































2.1 数据集内容

CIFAR10数据集是由10个类组成的60000个样本，每个样本都是一张32*32像素的RGB图像（彩色图像），每个RGB图像又必定分为3个通道（R通道、G通道、B通道）。这60000个样本被分成了50000个训练样本和10000个测试样本。

CIFAR10数据集是用来监督学习训练的，那么每个样本就一定都配备了一个标签值（用来区分这个样本是什么），不同类别的物体用不同的标签值，CIFAR10中有10类物体，标签值分别按照0~9来区分,他们分别是飞机（ airplane ）、汽车（ automobile ）、鸟（ bird ）、猫（ cat ）、鹿（ deer ）、狗（ dog ）、青蛙（ frog ）、马（ horse ）、船（ ship ）和卡车（ truck ）。

数据集分为五个训练批次和一个测试批次，每个批次都有10000张图像。测试批次正好包含每个类的1000张随机选择的图像。训练批次以随机顺序包含剩余图像，一些训练批次可能包含来自一个类的图像比另一个类的图像更多，但是在训练批次之间，全部的训练批次正好包含每个类的5000张图像。

以下是数据集中的类，以及每个数据集中的10个随机图像：

飞机										
汽车										
鸟										
猫										
鹿										
狗										
蛙类										
马										
船										
卡车										

这些类之间完全相互排斥。汽车和卡车之间没有重叠。“汽车”包括轿车、SUV之类的东西。“卡车”只包括大卡车。两者都不包括皮卡。

2.2 数据集结构

CIFAR10数据集结构组成可分为这四个部分：

- train_x:(50000, 32, 32, 3)——训练样本
- train_y:(50000, 1)——训练样本标签
- test_x:(10000, 32, 32, 3)——测试样本
- test_y:(10000, 1)——测试样本标签

2.3 数据集使用

在Keras中已经内置了多种公共数据集，其中就包含CIFAR-10数据集，所以可以直接调`tf.keras.datasets.cifar10`，直接下载数据集。

```
# 加载数据
cifar10 = tf.keras.datasets.cifar10
(train_x, train_y), (test_x, test_y) = cifar10.load_data()
```

```
-----
2023-01-01 18:47:26
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
42786816/170498071 [=====>.....] - ETA: 22s|
```

3 深度神经网络

3.1 全连接网络(fully connected neural network, FCNN)

3.1.1 说明

顾名思义，全连接神经网络中，对n-1层和n层而言，n-1层的任意一个节点，都和第n层所有节点有连接。即第n层的每个节点在进行计算的时候，激活函数的输入是n-1层所有节点的加权，这个激活函数是非线性的。**它的缺点就是权重太多了，计算量很大。**
它可作用于大多数场景。

3.1.2 神经网络

下面是一个使用 TensorFlow 实现FCNN算法的 CIFAR10 图像分类的模型代码：

```

model = keras.Sequential([
    keras.layers.Flatten(),
    keras.layers.Dense(512, activation="relu", input_shape=(x_train.shape[1],)),
    keras.layers.Dense(256, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])

```

这段代码创建了一个 Keras 模型，该模型是一个序列模型，包含了多个网络层。

首先，模型的第一层是一个 Flatten 层，它将输入的多维数组展平为一维数组，以便于后面的全连接层使用。

接下来，模型包含了两个 Dense 层。Dense 层就是全连接层，每个节点都与上一层的所有节点连接。第一个 Dense 层有 512 个节点，使用 relu 作为激活函数。输入层的形状为 (x_train.shape[1,])，意思是该层只有一个特征（即输入数据的列数）。第二个 Dense 层有 256 个节点，也使用 relu 作为激活函数。

最后，模型的最后一层是一个 Dense 层，有 10 个节点，使用 softmax 作为激活函数。softmax 函数可以将输出转化为概率值，使得输出的各个值的总和为 1。这在分类问题中很常用，因为概率值可以表示模型的置信度。

3.1.3 完整实现

```

import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import time
import numpy as np

# 初始化
plt.rcParams['font.sans-serif'] = ['SimHei']

# Load the CIFAR10 dataset
(train_x, train_y), (test_x, test_y) = keras.datasets.cifar10.load_data()

# Preprocess the data
x_train, x_test = tf.cast(train_x / 255.0, tf.float32), tf.cast(test_x / 255.0, tf.float32)
# 归一化
y_train, y_test = tf.cast(train_y, tf.int16), tf.cast(test_y, tf.int16)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)

# Build the model
model = keras.Sequential([

```

```

keras.layers.Flatten(),
keras.layers.Dense(512, activation="relu", input_shape=(x_train.shape[1],)),
keras.layers.Dense(256, activation="relu"),
keras.layers.Dense(10, activation="softmax")
])

# Compile the model
model.compile(optimizer="adam",
              loss="sparse_categorical_crossentropy",
              metrics=['sparse_categorical_accuracy'])

# 训练模型
# 批量训练大小为64，迭代5次，测试集比例0.2（48000条训练集数据，12000条测试集数据）
print('-----')
nowtime = time.strftime('%Y-%m-%d %H:%M:%S')
print('训练前时刻: ' + str(nowtime))

history = model.fit(x_train, y_train, batch_size=64, epochs=10, validation_split=0.2)

print('-----')
nowtime = time.strftime('%Y-%m-%d %H:%M:%S')
print('训练后时刻: ' + str(nowtime))

# 评估模型
model.evaluate(x_test, y_test, verbose=2) # 每次迭代输出一条记录，来评价该模型是否有比较好的泛化能力

# 保存整个模型
model.save('CIFAR10_FCNN_weights.h5')

# 结果可视化
print(history.history)
loss = history.history['loss'] # 训练集损失
val_loss = history.history['val_loss'] # 测试集损失
acc = history.history['sparse_categorical_accuracy'] # 训练集准确率
val_acc = history.history['val_sparse_categorical_accuracy'] # 测试集准确率

plt.figure(figsize=(10, 3))

plt.subplot(121)
plt.plot(loss, color='b', label='train')
plt.plot(val_loss, color='r', label='test')
plt.ylabel('loss')
plt.legend()

```

```

plt.subplot(122)
plt.plot(acc, color='b', label='train')
plt.plot(val_acc, color='r', label='test')
plt.ylabel('Accuracy')
plt.legend()

plt.show()

# 使用模型
plt.figure()
for i in range(10):
    num = np.random.randint(1, 10000)

    plt.subplot(2, 5, i + 1)
    plt.axis('off')
    plt.imshow(test_x[num], cmap='gray')
    demo = tf.reshape(x_test[num], (1, 32, 32, 3))
    y_pred = np.argmax(model.predict(demo))
    plt.title('标签值: ' + str(test_y[num]) + '\n预测值: ' + str(y_pred))

plt.show()

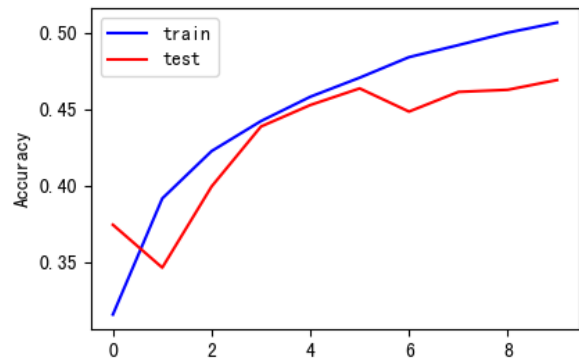
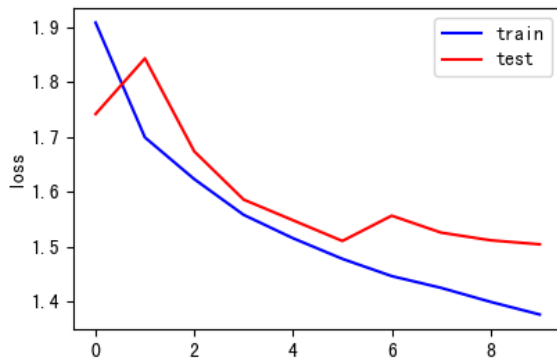
```

这段代码会自动下载 CIFAR10 数据集并进行训练和测试。训练时会使用 Adam 优化器，损失函数使用稀疏分类交叉熵（sparse categorical crossentropy），并在每个 epoch 结束时输出训练损失和精度。在训练结束后，模型会在测试集上进行评估，并输出测试损失和精度。

3.1.4 输出结果


```
pythonProject3  fcnn.py
pythonProject3  18 print(y_test.shape)  5 import numpy as np
pythonProject3  19  6
pythonProject3  20  7 # 初始化

Python Console  fcnn x
Epoch 4/10
625/625 [=====] - 5s 8ms/step - loss: 1.5581 - sparse_categorical_accuracy: 0.4425 - val_loss: 1.5858 -
val_sparse_categorical_accuracy: 0.4389
Epoch 5/10
625/625 [=====] - 5s 8ms/step - loss: 1.5156 - sparse_categorical_accuracy: 0.4584 - val_loss: 1.5482 -
val_sparse_categorical_accuracy: 0.4530
Epoch 6/10
625/625 [=====] - 5s 8ms/step - loss: 1.4781 - sparse_categorical_accuracy: 0.4709 - val_loss: 1.5106 -
val_sparse_categorical_accuracy: 0.4639
Epoch 7/10
625/625 [=====] - 5s 8ms/step - loss: 1.4467 - sparse_categorical_accuracy: 0.4844 - val_loss: 1.5566 -
val_sparse_categorical_accuracy: 0.4487
Epoch 8/10
625/625 [=====] - 5s 8ms/step - loss: 1.4253 - sparse_categorical_accuracy: 0.4923 - val_loss: 1.5258 -
val_sparse_categorical_accuracy: 0.4616
Epoch 9/10
625/625 [=====] - 5s 8ms/step - loss: 1.3999 - sparse_categorical_accuracy: 0.5005 - val_loss: 1.5118 -
val_sparse_categorical_accuracy: 0.4630
Epoch 10/10
625/625 [=====] - 5s 8ms/step - loss: 1.3768 - sparse_categorical_accuracy: 0.5070 - val_loss: 1.5045 -
val_sparse_categorical_accuracy: 0.4694
-----
训练后时刻 : 2023-01-02 21:42:09
```





3.2 卷积神经网络 (Convolutional Neural Network, CNN)

3.2.1 说明

卷积神经网络是一种具有局部连接，权重共享等特性的深层前馈神经网络。一般是由卷积层，汇聚层，全连接层交叉堆叠而成，使用反向传播算法进行训练。其有三个结构上的特征：局部连接，权重共享以及汇聚。这些特征使得卷积神经网络具有一定程度上的平移，缩放和旋转不变性。较前馈神经网络而言，其参数更少。

卷积神经网络的输入为二维的像素整阵列，输出为这个图片的属性，当网络训练学习后，所输入的图片或许经过稍微的变换，但卷积神经网络还是可以通过识别图片局部的特征而将整个图片识别出来。

- 数据输入层：该层要做的处理主要是对原始图像数据进行预处理，包括去均值（把输入数据各个维度都中心化为0，其目的就是把样本的中心拉回到坐标系原点上），归一化（幅度归一化到同样的范围）；
- 卷积计算层：相当于滤镜，将图片进行分块，对每一块进行特征处理，从而提取特征，这是最重要的一层。具体操作还未仔细学习。
- 池化层：池化层夹在连续的卷积层中间，用于压缩数据和参数的量，减小过拟合。通过对提取的高维特征进行降维，对于输入为图像的情况，这里最主要的作用应该就是压缩。
- 全连接层：对空间排列的特征化成一维的向量。

主要应用：计算机视觉，图像和视频分析的各种任务上，比如图像分类，人脸识别，物体识别，图像分割等，其准确率也远远超过了其他的人工神经网络。近年来，卷积神经网络也应用到自然语言处理和推荐系统等领域。

3.2.2 神经网络

主要使用函数：tf.keras.layers.Conv2D()

```
tf.keras.layers.Conv2D(  
    filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,  
    dilation_rate=(1, 1), activation=None, use_bias=True,  
    kernel_initializer='glorot_uniform', bias_initializer='zeros',  
    kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,  
    kernel_constraint=None, bias_constraint=None, **kwargs  
)
```

神经网络核心实现

建立模型

```
model = tf.keras.Sequential()
```

特征提取阶段

第一层两个卷积层16个3*3卷积核，一个池化层：最大池化法2*2卷积核，激活函数：ReLU

卷积层，16个卷积核，大小（3，3），保持原图像大小，relu激活函数，输入形状（28，28，1）

```
model.add(tf.keras.layers.Conv2D(16, kernel_size=(3, 3), padding='same',  
    activation=tf.nn.relu, data_format='channels_last', input_shape=x_train.shape[1:]))  
model.add(tf.keras.layers.Conv2D(16, kernel_size=(3, 3), padding='same',  
    activation=tf.nn.relu))
```

池化层，最大值池化，卷积核（2，2）

```
model.add(tf.keras.layers.MaxPool2D(pool_size=(2, 2)))
```

第二层两个卷积层32个3*3卷积核，一个池化层：最大池化法2*2卷积核，激活函数：ReLU

```
model.add(tf.keras.layers.Conv2D(32, kernel_size=(3, 3), padding='same',  
    activation=tf.nn.relu))  
model.add(tf.keras.layers.Conv2D(32, kernel_size=(3, 3), padding='same',  
    activation=tf.nn.relu))
```

```
model.add(tf.keras.layers.MaxPool2D(pool_size=(2, 2)))
```

分类识别阶段

第三层

```
model.add(tf.keras.layers.Flatten()) # 改变输入形状
```

第四层

隐含层激活函数：ReLU函数，全连接网络层，128个神经元

```
model.add(tf.keras.layers.Dense(128, activation='relu'))
```

输出层激活函数：softmax函数（实现多分类），10个节点

```
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

这段代码使用了 TensorFlow 的 `tf.keras` API 中的 `Sequential` 模型来定义一个模型。该模型由几个卷积层，最大池化层和密集层组成。

第一和第二组层每组都包括两个卷积层和一个最大池化层。卷积层分别有 16 和 32 个滤波器，使用内核大小为 (3, 3)。最大池化层使用池化大小为 (2, 2)。所有这些层使用的激活函数为 ReLU。

最终的层块包括一个 `Flatten` 层，它将前一层的输出展平为一维张量，以及两个密集层。第一个密集层有 128 个单元并使用 ReLU 激活函数，而第二个密集层有 10 个单元并使用 softmax 激活函数进行多类分类。

3.2.3 完整实现

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import time

print('-----')
nowtime = time.strftime('%Y-%m-%d %H:%M:%S')
print(nowtime)

# 初始化
plt.rcParams['font.sans-serif'] = ['SimHei']

# 加载数据
# train_x:(50000, 32, 32, 3), train_y:(50000, 1), test_x:(10000, 32, 32, 3), test_y:(10000, 1)
# 60000条训练数据和10000条测试数据, 32x32像素的RGB图像
cifar10 = tf.keras.datasets.cifar10
(train_x, train_y), (test_x, test_y) = cifar10.load_data()
print('\n train_x:%s, train_y:%s, test_x:%s, test_y:%s' % (train_x.shape, train_y.shape,
test_x.shape, test_y.shape))

# 数据预处理
x_train, x_test = tf.cast(train_x / 255.0, tf.float32), tf.cast(test_x / 255.0, tf.float32)
# 归一化
y_train, y_test = tf.cast(train_y, tf.int16), tf.cast(test_y, tf.int16)

# 建立模型
model = tf.keras.Sequential()
# 特征提取阶段

# 第一层两个卷积层16个3*3卷积核, 一个池化层: 最大池化法2*2卷积核, 激活函数: ReLU
# 卷积层, 16个卷积核, 大小 (3, 3), 保持原图像大小, relu激活函数, 输入形状 (28, 28, 1)
model.add(tf.keras.layers.Conv2D(16, kernel_size=(3, 3), padding='same',
activation=tf.nn.relu, data_format='channels_last', input_shape=x_train.shape[1:])))
```

```

model.add(tf.keras.layers.Conv2D(16, kernel_size=(3, 3), padding='same',
activation=tf.nn.relu))
# 池化层, 最大值池化, 卷积核 (2, 2)
model.add(tf.keras.layers.MaxPool2D(pool_size=(2, 2)))

# 第二层两个卷积层32个3*3卷积核, 一个池化层: 最大池化法2*2卷积核, 激活函数: ReLU
model.add(tf.keras.layers.Conv2D(32, kernel_size=(3, 3), padding='same',
activation=tf.nn.relu))
model.add(tf.keras.layers.Conv2D(32, kernel_size=(3, 3), padding='same',
activation=tf.nn.relu))
model.add(tf.keras.layers.MaxPool2D(pool_size=(2, 2)))

# 分类识别阶段

# 第三层
model.add(tf.keras.layers.Flatten()) # 改变输入形状
# 第四层
# 隐含层激活函数: ReLU函数, 全连接网络层, 128个神经元
model.add(tf.keras.layers.Dense(128, activation='relu'))
# 输出层激活函数: softmax函数 (实现多分类), 10个节点
model.add(tf.keras.layers.Dense(10, activation='softmax'))
print(model.summary()) # 查看网络结构和参数信息

# 配置模型训练方法
# adam算法参数采用keras默认的公开参数, 损失函数采用稀疏交叉熵损失函数, 准确率采用稀疏分类准确率
函数
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=
['sparse_categorical_accuracy'])

# 训练模型
# 批量训练大小为64, 迭代5次, 测试集比例0.2 (48000条训练集数据, 12000条测试集数据)
print('-----')
nowtime = time.strftime('%Y-%m-%d %H:%M:%S')
print('训练前时刻: ' + str(nowtime))

history = model.fit(x_train, y_train, batch_size=64, epochs=5, validation_split=0.2)

print('-----')
nowtime = time.strftime('%Y-%m-%d %H:%M:%S')
print('训练后时刻: ' + str(nowtime))

# 评估模型
model.evaluate(x_test, y_test, verbose=2) # 每次迭代输出一条记录, 来评价该模型是否有比较好的
泛化能力

```

保存整个模型

```
model.save('CIFAR10_CNN_weights.h5')
```

结果可视化

```
print(history.history)
```

```
loss = history.history['loss'] # 训练集损失
```

```
val_loss = history.history['val_loss'] # 测试集损失
```

```
acc = history.history['sparse_categorical_accuracy'] # 训练集准确率
```

```
val_acc = history.history['val_sparse_categorical_accuracy'] # 测试集准确率
```

```
plt.figure(figsize=(10, 3))
```

```
plt.subplot(121)
```

```
plt.plot(loss, color='b', label='train')
```

```
plt.plot(val_loss, color='r', label='test')
```

```
plt.ylabel('loss')
```

```
plt.legend()
```

```
plt.subplot(122)
```

```
plt.plot(acc, color='b', label='train')
```

```
plt.plot(val_acc, color='r', label='test')
```

```
plt.ylabel('Accuracy')
```

```
plt.legend()
```

使用模型

```
plt.figure()
```

```
for i in range(10):
```

```
    num = np.random.randint(1, 10000)
```

```
    plt.subplot(2, 5, i + 1)
```

```
    plt.axis('off')
```

```
    plt.imshow(test_x[num], cmap='gray')
```

```
    demo = tf.reshape(x_test[num], (1, 32, 32, 3))
```

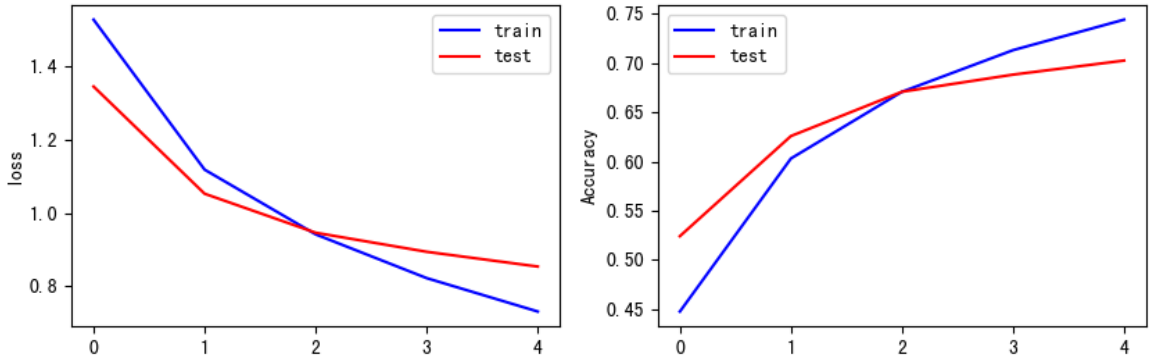
```
    y_pred = np.argmax(model.predict(demo))
```

```
    plt.title('标签值: ' + str(test_y[num]) + '\n预测值: ' + str(y_pred))
```

```
plt.show()
```

3.2.4 输出结果

```
pythonProject3  cnn.py
pythonProject3
  CIFAR10_CNN
  CIFAR10_FCNN
  cnn.py
Run:  cnn
2023-01-02 21:46:30.151280: W tensorflow/core/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU frequency: 0 Hz
Epoch 1/5
2023-01-02 21:46:30.508685: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
623/625 [=====] - ETA: 0s - loss: 1.5303 - sparse_categorical_accuracy: 0.44722023-01-02 21:46:39.268469: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
625/625 [=====] - 10s 16ms/step - loss: 1.5293 - sparse_categorical_accuracy: 0.4476 - val_loss: 1.3463 - val_sparse_categorical_accuracy: 0.5242
Epoch 2/5
625/625 [=====] - 9s 15ms/step - loss: 1.1189 - sparse_categorical_accuracy: 0.6031 - val_loss: 1.0531 - val_sparse_categorical_accuracy: 0.6259
Epoch 3/5
625/625 [=====] - 11s 18ms/step - loss: 0.9420 - sparse_categorical_accuracy: 0.6709 - val_loss: 0.9464 - val_sparse_categorical_accuracy: 0.6709
Epoch 4/5
625/625 [=====] - 9s 15ms/step - loss: 0.8222 - sparse_categorical_accuracy: 0.7132 - val_loss: 0.8943 - val_sparse_categorical_accuracy: 0.6883
Epoch 5/5
625/625 [=====] - 10s 16ms/step - loss: 0.7309 - sparse_categorical_accuracy: 0.7442 - val_loss: 0.8539 - val_sparse_categorical_accuracy: 0.7026
-----
训练后时刻: 2023-01-02 21:47:20
313/313 - 2s - loss: 0.8559 - sparse_categorical_accuracy: 0.7074 - 2s/epoch - 6ms/step
{'loss': [1.5292590856552124, 1.1189241409301758, 0.9419516324996948, 0.822246611183167, 0.7308646440505981], 'sparse_categorical_accuracy': [0.4476250112056732, 0.6031000018119812, 0.6708750128746033, 0.7131500244140625, 0.7441750168800354], 'val_loss': [1.346291422843933, 1.0530930757522583, 0.9464403986930847, 0.8942514061927795, 0.8538949489593506], 'val_sparse_categorical_accuracy': [0.5242000222206116, 0.6259000301361084, 0.6709000468254089, 0.6883000135421753, 0.7026000618934631]}
```





3.3 循环神经网络 (Recurrent neural network, RNN)

3.3.1 说明

循环神经网络是一类具有短期记忆能力的神经网络，在循环神经网络中，神经元不仅可以接受其他神经元的信息，还可以接受自身的信息，形成一个环路结构。在很多现实任务中，网络的输出不仅和当前的输入有关，也和过去一段时间的输出相关。

3.3.2 神经网络

```
model = keras.Sequential([
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, input_shape=(1024, 3),
return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
```


这是一个使用了双向 LSTM 层的循环神经网络模型，用于分类 CIFAR10 数据集。

模型的第一层是一个双向 LSTM 层，它具有 32 个神经元，并返回序列。模型的第二层是一个双向 LSTM 层，它也具有 32 个神经元。第三层是一个 Dropout 层，它可以防止过拟合。第四层是一个 Flatten 层，它把输入展平。第五层是一个 Dense 层，它具有 128 个神经元和 ReLU 激活函数。第六层是一个 Dropout 层，它也可以防止过拟合。第七层是一个 Dense 层，它具有 64 个神经元和 ReLU 激活函数。最后一层是一个 Dense 层，它具有 10 个神经元和 softmax 激活函数，用于分类。

3.3.3 完整实现

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import time
import numpy as np

plt.rcParams['font.sans-serif'] = ['SimHei']

# Load the CIFAR10 dataset
(train_x, train_y), (test_x, test_y) = keras.datasets.cifar10.load_data()

x_train = train_x.reshape(50000,1024,3)/255.0
x_test = test_x.reshape(10000,1024,3)/255.0
y_train, y_test = tf.cast(train_y, tf.int16), tf.cast(test_y, tf.int16)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)

# Build the model
model = keras.Sequential([
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, input_shape=(1024, 3),
return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
```

```

])

# Compile the model
model.compile(optimizer="adam",
              loss="sparse_categorical_crossentropy",
              metrics=['sparse_categorical_accuracy'])

# 训练模型
# 批量训练大小为32，迭代5次，测试集比例0.2（48000条训练集数据，12000条测试集数据）
print('-----')
nowtime = time.strftime('%Y-%m-%d %H:%M:%S')
print('训练前时刻: ' + str(nowtime))

history = model.fit(x_train, y_train, batch_size=32, epochs=10, validation_split=0.2)

print('-----')
nowtime = time.strftime('%Y-%m-%d %H:%M:%S')
print('训练后时刻: ' + str(nowtime))

# 评估模型
model.evaluate(x_test, y_test, verbose=2) # 每次迭代输出一条记录，来评价该模型是否有比较好的泛化能力

# 保存整个模型
model.save('CIFAR10_RNN_weights.h5')

# 结果可视化
print(history.history)
loss = history.history['loss'] # 训练集损失
val_loss = history.history['val_loss'] # 测试集损失
acc = history.history['sparse_categorical_accuracy'] # 训练集准确率
val_acc = history.history['val_sparse_categorical_accuracy'] # 测试集准确率

plt.figure(figsize=(10, 3))

plt.subplot(121)
plt.plot(loss, color='b', label='train')
plt.plot(val_loss, color='r', label='test')
plt.ylabel('loss')
plt.legend()

plt.subplot(122)
plt.plot(acc, color='b', label='train')
plt.plot(val_acc, color='r', label='test')

```

```

plt.ylabel('Accuracy')
plt.legend()

plt.show()

# 使用模型
plt.figure()
for i in range(10):
    num = np.random.randint(1, 10000)

    plt.subplot(2, 5, i + 1)
    plt.axis('off')
    plt.imshow(test_x[num], cmap='gray')
    demo = tf.reshape(x_test[num], (1, 1024, 3))
    y_pred = np.argmax(model.predict(demo))
    plt.title('label:' + str(test_y[num]) + '\npred:' + str(y_pred))

plt.show()

```

这是一个使用TensorFlow实现的RNN模型，用于对CIFAR10数据集的图像进行分类。

完成核心的模型定义后使用Adam优化器，稀疏分类交叉熵损失函数和稀疏分类准确率指标来编译模型。然后使用fit方法在训练数据上训练模型，并指定批量大小为32，迭代10次，将20 %的数据用于验证。

最后，使用evaluate方法在测试数据上评估模型的表现。模型的训练和评估过程的损失和准确率信息将被绘制到图中。最后，使用save方法保存模型。

3.3.4 输出结果

(50000, 1)
(10000, 1)

训练前时刻: 2023-01-02 18:39:32

Epoch 1/10

1250/1250 [=====] - 150s 116ms/step - loss: 2.0564 - spars
e_categorical_accuracy: 0.2356 - val_loss: 1.9899 - val_sparse_categorical_accuracy

: 0.2640

Epoch 2/10

1250/1250 [=====] - 141s 113ms/step - loss: 1.9824 - spars
e_categorical_accuracy: 0.2708 - val_loss: 1.9634 - val_sparse_categorical_accuracy

: 0.2869

Epoch 3/10

1250/1250 [=====] - 142s 113ms/step - loss: 1.9615 - spars
e_categorical_accuracy: 0.2869 - val_loss: 1.9267 - val_sparse_categorical_accuracy

: 0.2969

Epoch 4/10

1250/1250 [=====] - 141s 113ms/step - loss: 1.9479 - spars
e_categorical_accuracy: 0.2909 - val_loss: 1.9232 - val_sparse_categorical_accuracy

: 0.2962

Epoch 5/10

1250/1250 [=====] - 142s 113ms/step - loss: 1.9278 - spars
e_categorical_accuracy: 0.2984 - val_loss: 1.8924 - val_sparse_categorical_accuracy

: 0.3164

Epoch 6/10

1250/1250 [=====] - 140s 112ms/step - loss: 1.9430 - spars
e_categorical_accuracy: 0.2918 - val_loss: 1.9278 - val_sparse_categorical_accuracy

: 0.3025

Epoch 7/10

1250/1250 [=====] - 142s 113ms/step - loss: 1.9184 - spars
e_categorical_accuracy: 0.3027 - val_loss: 1.8880 - val_sparse_categorical_accuracy

: 0.3203

Epoch 8/10

1250/1250 [=====] - 141s 113ms/step - loss: 1.8753 - spars
e_categorical_accuracy: 0.3219 - val_loss: 1.8400 - val_sparse_categorical_accuracy

: 0.3352

Epoch 9/10

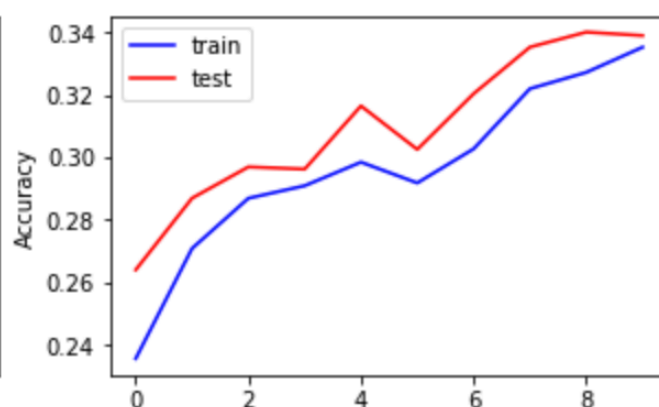
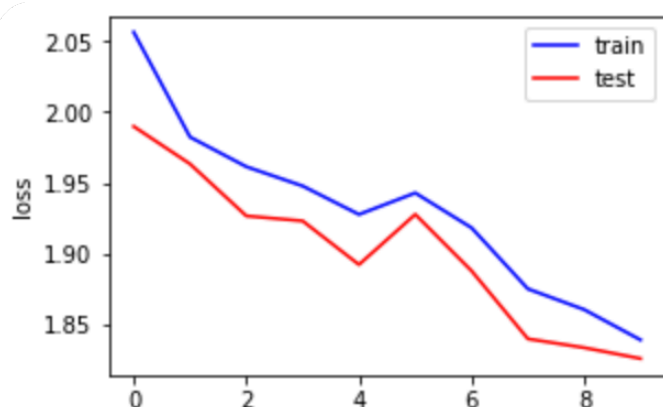
1250/1250 [=====] - 141s 113ms/step - loss: 1.8607 - spars
e_categorical_accuracy: 0.3271 - val_loss: 1.8338 - val_sparse_categorical_accuracy

: 0.3400

Epoch 10/10

1250/1250 [=====] - 141s 113ms/step - loss: 1.8393 - spars
e_categorical_accuracy: 0.3352 - val_loss: 1.8261 - val_sparse_categorical_accuracy

: 0.3389



label:[6]
pred:6



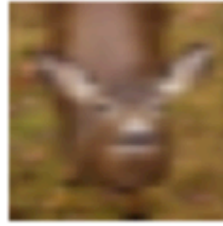
label:[7]
pred:7



label:[9]
pred:9



label:[4]
pred:6



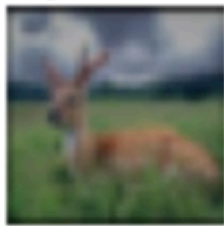
label:[5]
pred:7



label:[6]
pred:6



label:[4]
pred:4



label:[2]
pred:8



label:[3]
pred:1



label:[0]
pred:3



4 机器学习分类算法——支持向量机SVM

4.1 说明

支持向量机（SVM）是一种广泛使用的监督学习算法，可用于分类和回归。它的工作原理是找到一条决策边界，将不同类别的数据尽可能分开。

使用 SVM 分类 CIFAR-10 数据集的一般步骤：

1. 准备数据：将 CIFAR-10 数据集加载到 Python 中，并将其转换为适合模型使用的格式。可能需要对图像进行预处理，例如缩放或归一化。
2. 选择 SVM 模型：选择要使用的 SVM 模型（例如线性 SVM 或非线性 SVM）和决策函数。
3. 训练模型：使用训练数据训练 SVM 模型。
4. 评估模型：使用测试数据评估 SVM 模型。

4.2 实现代码

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
import pickle

# 加载数据
(train_x, train_y), (test_x, test_y) = keras.datasets.cifar10.load_data()

# Flatten images to 2D array
x_train = np.reshape(train_x, (len(train_x), -1))
x_test = np.reshape(test_x, (len(test_x), -1))
# Convert labels to numbers
y_train = np.ravel(train_y)
y_test = np.ravel(test_y)
print(x_train.shape)
print(y_train.shape)

# 训练 SVM 模型
clf = LinearSVC(verbose=True, loss='hinge', max_iter=50)
clf.fit(x_train, y_train)

# Save model
with open('SVMmodel.pkl', 'wb') as f:
    pickle.dump(clf, f)

# Load model
# with open('SVMmodel.pkl', 'rb') as f:
#     clf = pickle.load(f)

# Predict x
# y_pred = clf.predict(x)
# 评估模型
accuracy = clf.score(x_test, y_test)
print("Accuracy: {:.2f}".format(accuracy))

# Compute predictions
predictions = clf.decision_function(x_test)

# Compute loss
loss = np.maximum(0, 1 - test_y * predictions)
```

```

# Print mean loss
print("Loss: {:.2f}".format(np.mean(loss)))

# 使用模型
plt.figure()
for i in range(10):
    num = np.random.randint(1, 10000)

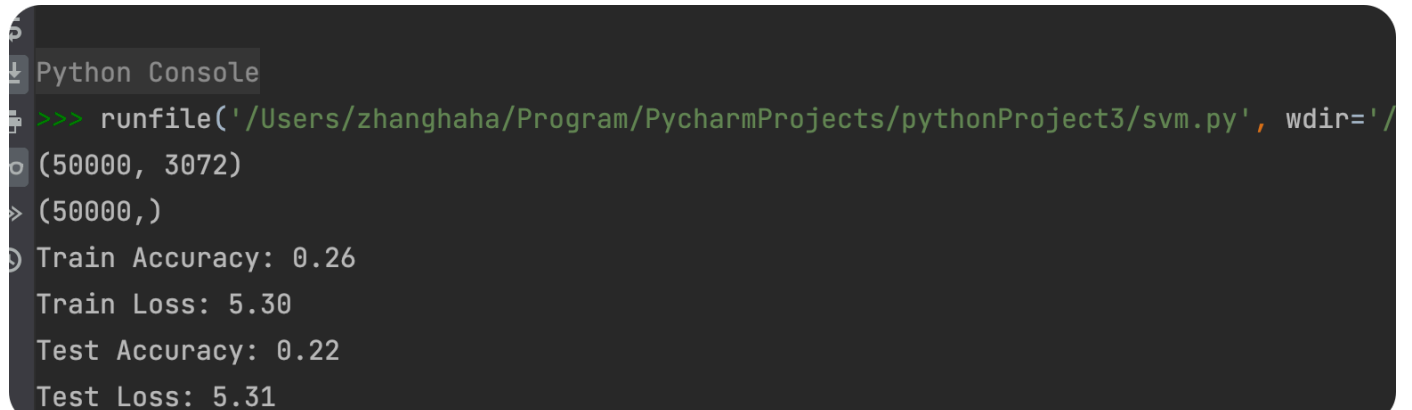
    plt.subplot(2, 5, i + 1)
    plt.axis('off')
    plt.imshow(test_x[num], cmap='gray')
    demo = np.reshape(x_test[num], (1, 3072))
    y_pred = clf.predict(demo)
    plt.title('label:' + str(test_y[num]) + '\npred:' + str(y_pred))

plt.show()

```

4.3 输出结果

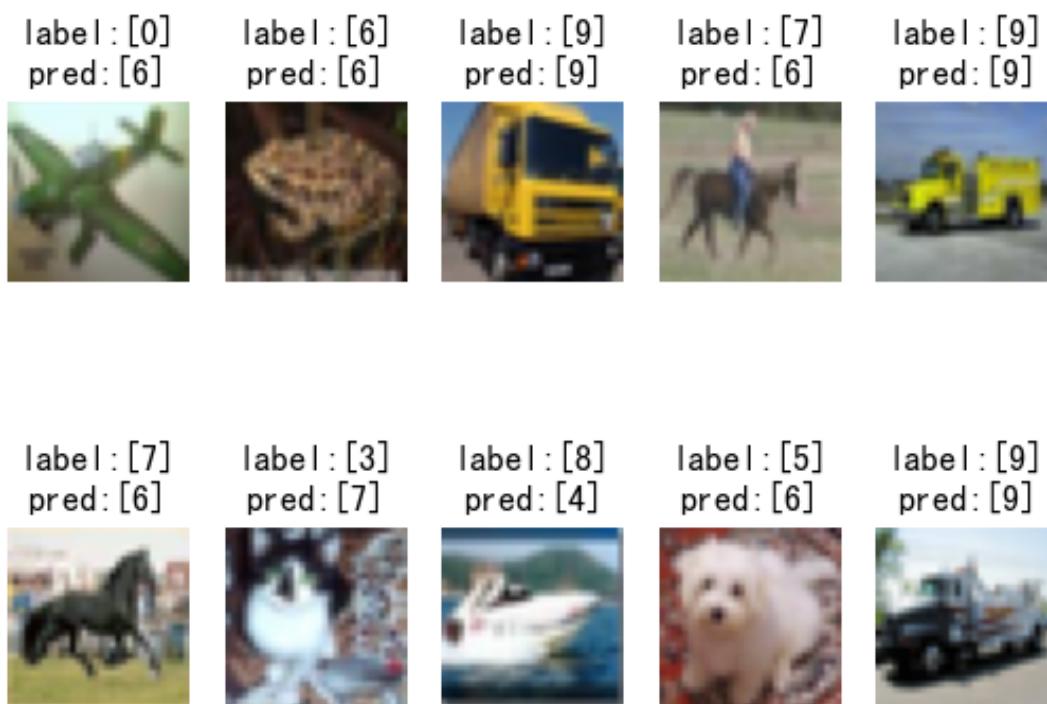
4.3.1 输出可视化



```

Python Console
>>> runfile('/Users/zhanghaha/Program/PycharmProjects/pythonProject3/svm.py', wdir='/
(50000, 3072)
> (50000,)
Train Accuracy: 0.26
Train Loss: 5.30
Test Accuracy: 0.22
Test Loss: 5.31

```



4.3.2 参数输出分析

```
svm × svm (9) × svm (10) ×
.....
optimization finished, #iter = 50

WARNING: reaching max number of iterations
Using -s 2 may be faster (also see FAQ)

Objective value = -0.014645
nSV = 33279
.....
optimization finished, #iter = 50

WARNING: reaching max number of iterations
Using -s 2 may be faster (also see FAQ)

Objective value = -0.014279
nSV = 32508
```

训练过程中的参数输出分析:


```
optimization finished, #iter = 50
Objective value = -0.014279
nSV = 32508
```

在使用sklearn的SVM进行分类时，输出的"optimization finished, #iter = 50"表示SVM分类器在训练过程中迭代了50次。

"Objective value"表示训练过程中的目标值，通常是损失函数的值。通常情况下，SVM分类器会尽量最小化损失函数值，因此当损失函数值越小时，分类器的性能越好。

"nSV"表示训练过程中所使用的支持向量的数量。支持向量是SVM分类器中的一类特殊数据点，它们对SVM分类器的决策边界有很大的影响。

5 分类结果对比分析

5.1 随机图片预测

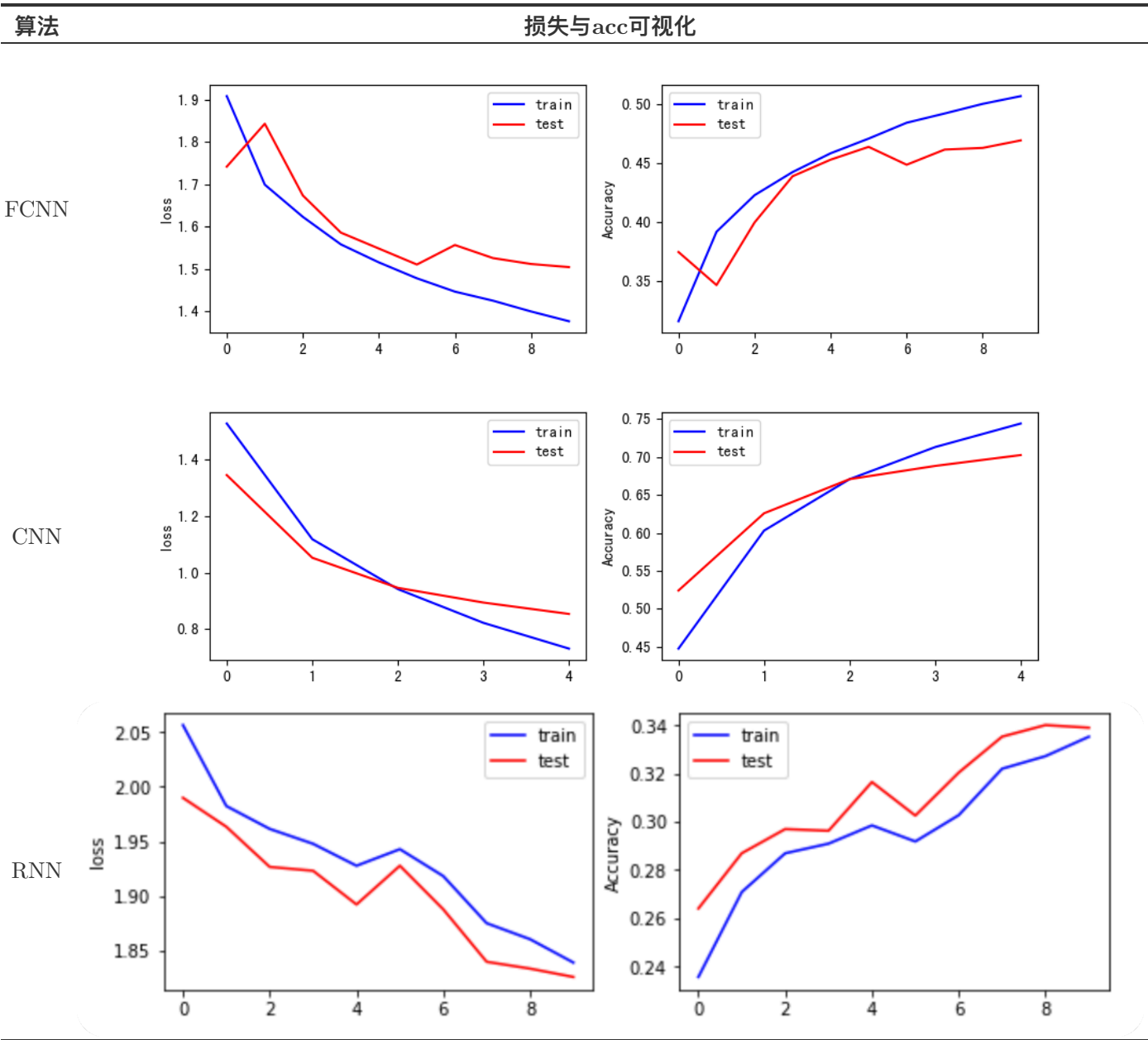
神经 算法	随即预测	算 法	随机预测	
	<div>标签值: [3] 预测值: 2</div> <div>标签值: [4] 预测值: 2</div> <div>标签值: [8] 预测值: 8</div> <div>标签值: [6] 预测值: 6</div> <div>标签值: [6] 预测值: 0</div> <div></div> <div></div> <div></div> <div></div> <div></div>		<div>标签值: [8] 预测值: 8</div> <div>标签值: [0] 预测值: 0</div> <div>标签值: [6] 预测值: 6</div> <div>标签值: [1] 预测值: 1</div> <div>标签值: [1] 预测值: 9</div> <div></div> <div></div> <div></div> <div></div> <div></div>	
FCNN	<div>标签值: [0] 预测值: 0</div> <div>标签值: [6] 预测值: 6</div> <div>标签值: [3] 预测值: 5</div> <div>标签值: [7] 预测值: 7</div> <div>标签值: [4] 预测值: 4</div> <div></div> <div></div> <div></div> <div></div> <div></div>		CNN	<div>标签值: [3] 预测值: 6</div> <div>标签值: [7] 预测值: 7</div> <div>标签值: [8] 预测值: 8</div> <div>标签值: [3] 预测值: 3</div> <div>标签值: [0] 预测值: 0</div> <div></div> <div></div> <div></div> <div></div> <div></div>
	<div>label:[6] pred:6</div> <div>label:[7] pred:7</div> <div>label:[9] pred:9</div> <div>label:[4] pred:6</div> <div>label:[5] pred:7</div> <div></div> <div></div> <div></div> <div></div> <div></div>			<div>label:[0] pred:[6]</div> <div>label:[6] pred:[6]</div> <div>label:[9] pred:[9]</div> <div>label:[7] pred:[6]</div> <div>label:[9] pred:[9]</div> <div></div> <div></div> <div></div> <div></div> <div></div>
RNN	<div>label:[6] pred:6</div> <div>label:[4] pred:4</div> <div>label:[2] pred:8</div> <div>label:[3] pred:1</div> <div>label:[0] pred:3</div> <div></div> <div></div> <div></div> <div></div> <div></div>		SVM	<div>label:[7] pred:[6]</div> <div>label:[3] pred:[7]</div> <div>label:[8] pred:[4]</div> <div>label:[5] pred:[6]</div> <div>label:[9] pred:[9]</div> <div></div> <div></div> <div></div> <div></div> <div></div>

对训练得到的模型，拿出随机10张图片进行预测：

- FCNN：有6张图片预测值与原标签对应
- CNN：有8张图片预测值与原标签对应
- RNN：有5张图片预测值与原标签对应
- SVM：有4张图片预测值与原标签对应

对于模型的图片抽查检测可知：CNN>FCNN>RNN>SVM

5.2 损失与ACC



对于SVM算法，我使用使用`decision_function()`函数计算模型对测试数据的预测值，然后使用hinge loss损失函数的公式计算损失值。

得到的值:

```
Python Console
>>> runfile('/Users/zhanghaha/Program/PycharmProjects/pythonProject3/svm.py', wdir='/
(50000, 3072)
> (50000,)
Train Accuracy: 0.26
Train Loss: 5.30
Test Accuracy: 0.22
Test Loss: 5.31
```

对上述结果分析得到，训练部分的损失函数与精确率，以及预测部分的损失函数与精确度详情如下：

	train_loss	train_acc	test_loss	test_acc
fcnn	1.3768	0.507	1.5045	0.4694
cnn	0.7309	0.7442	0.8539	0.7026
rnn	1.8393	0.3352	1.8261	0.3389
svm	5.3	0.26	5.31	0.22

可以得知，在我的神经模型设计下，出现的模型适配度：CNN>FCNN>RNN>SVM。

其中深度学习的CNN的效果最好，损失函数值最小，精确度最高。

而机器学习的SVM的损失函数值与精确度最低。

6 附件

附件包含四种网络训练出来的模型，以及源码:

算法	模型名称	源码名称
FCNN	CIFAR10_FCNN_weights.h5	fcnn.py
CNN	CIFAR10_CNN_weights.h5	cnn.py
RNN	CIFAR10_RNN_weights.h5	rnn.py
SVM	SVMmodel.pkl	svm.py