计算机科学与技术学院神经网络与深度学习课程实验报告

实验题目：实现各种classifier		学号：201900130143
日期：9/30	班级： 智能	姓名： 吴家麒
Email：996362192@qq.com		

实验目的：

- understand the basic **Image Classification pipeline** and the data-driven approach (train/predict stages)
- understand the train/val/test **splits** and the use of validation data for **hyperparameter tuning**.
- develop proficiency in writing efficient **vectorized** code with numpy
- implement and apply a k-Nearest Neighbor (**kNN**) classifier
- implement and apply a Multiclass Support Vector Machine (**SVM**) classifier
- implement and apply a **Softmax** classifier
- implement and apply a **Three layer neural network** classifier
- understand the differences and tradeoffs between these classifiers
- get a basic understanding of performance improvements from using **higher-level representations** than raw pixels (e.g. color histograms, Histogram of Gradient (HOG) features)

实验软件和硬件环境：
Anaconda3+Jupyter NoteBook

实验原理和方法：
1. KNN classifier 实现
   加载数据集：

   ```
   Training data shape:  (50000, 32, 32, 3)
   Training labels shape:  (50000,)
   Test data shape:  (10000, 32, 32, 3)
   Test labels shape:  (10000,)
   ```

   预览部分训练数据：

compute_distances_two_loops：使用两重循环计算 distancew matrix

```
# Open sducs2019/classifiers/k_nearest_neighbor.py and implement
# compute_distances_two_loops.

# Test your implementation:
dists = classifier.compute_distances_two_loops(X_test)
print(dists.shape)
```

```
(500, 5000)
```

```
# We can visualize the distance matrix: each row is a single test example and
# its distances to training examples
plt.imshow(dists, interpolation='none')
plt.show()
```
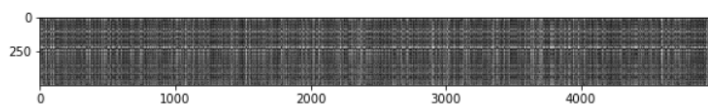

分别用 k=1，

```
Got 137 / 500 correct => accuracy: 0.274000
```

和 k=5 计算预测精度：

```
Got 139 / 500 correct => accuracy: 0.278000
```

使用部分矢量化和一重循环加速 distance matrix 的计算：

```
# Now lets speed up distance matrix computation by using partial vectorization
# with one loop.  Implement the function compute_distances_one_loop and run the
# code below:
dists_one = classifier.compute_distances_one_loop(X_test)

# To ensure that our vectorized implementation is correct, we make sure that it
# agrees with the naive implementation. There are many ways to decide whether
# two matrices are similar; one of the simplest is the Frobenius norm. In case
# you haven't seen it before, the Frobenius norm of two matrices is the square
# root of the squared sum of differences of all elements; in other words, reshape
# the matrices into vectors and compute the Euclidean distance between them.
difference = np.linalg.norm(dists - dists_one, ord='fro')
print('One loop difference was: %f' % (difference, ))
if difference < 0.001:
    print('Good! The distance matrices are the same')
else:
    print('Uh-oh! The distance matrices are different')
```

```
One loop difference was: 0.000000
Good! The distance matrices are the same
```

完全矢量化而不使用循环：

```
# Now implement the fully vectorized version inside compute_distances_no_loops
# and run the code
dists_two = classifier.compute_distances_no_loops(X_test)

# check that the distance matrix agrees with the one we computed before:
difference = np.linalg.norm(dists - dists_two, ord='fro')
print('No loop difference was: %f' % (difference, ))
if difference < 0.001:
    print('Good! The distance matrices are the same')
else:
    print('Uh-oh! The distance matrices are different')
```
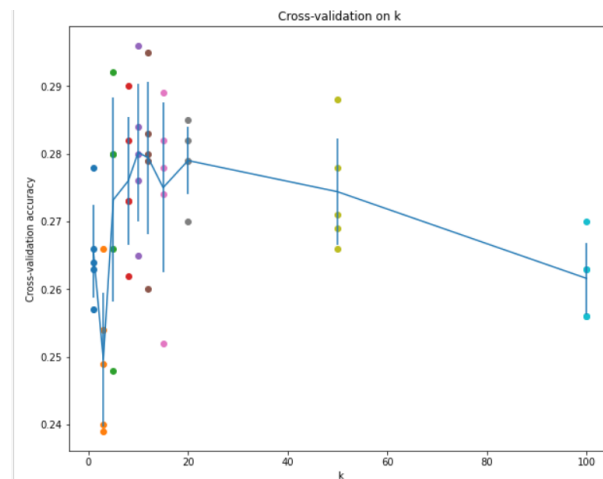
```
No loop difference was: 0.000000
Good! The distance matrices are the same
```

比较上述几种方法的效率：

```
Two loop version took 31.478619 seconds
One loop version took 66.148429 seconds
No loop version took 0.213431 seconds
```

使用交叉验证确定超参的最佳值：

2. SVM classifier 实现
   导入数据集：

```
Training data shape:  (50000, 32, 32, 3)
Training labels shape:  (50000,)
Test data shape:  (10000, 32, 32, 3)
Test labels shape:  (10000,)
```

分为训练集、验证集、训练集：

```
Train data shape:  (49000, 32, 32, 3)
Train labels shape:  (49000,)
Validation data shape:  (1000, 32, 32, 3)
Validation labels shape:  (1000,)
Test data shape:  (1000, 32, 32, 3)
Test labels shape:  (1000,)
```

再对数据进行一系列预处理后，调用 SVM classifier,通过 naïve 和 vectorized 两

种方法进行是实现，进行比较：

```
Naive loss: 9.120340e+00 computed in 0.123661s
Vectorized loss: 9.120340e+00 computed in 0.003963s
difference: 0.000000

Naive loss and gradient: computed in 0.132611s
Vectorized loss and gradient: computed in 0.002993s
difference: 0.000000
```

利用 SGD 方法最小化损失函数：

```python
# In the file linear_classifier.py, implement SGD in the function
# LinearClassifier.train() and then run it with the code below.
from sducs2019.classifiers import LinearSVM
svm = LinearSVM()
tic = time.time()
loss_hist = svm.train(X_train, y_train, learning_rate=1e-7, reg=2.5e4,
                num_iters=1500, verbose=True)
toc = time.time()
print('That took %fs' % (toc - tic))
```

```
iteration 0 / 1500: loss 396.524657
iteration 100 / 1500: loss 236.428358
iteration 200 / 1500: loss 144.157705
iteration 300 / 1500: loss 88.272558
iteration 400 / 1500: loss 54.764083
iteration 500 / 1500: loss 35.191992
iteration 600 / 1500: loss 23.452266
iteration 700 / 1500: loss 16.118776
iteration 800 / 1500: loss 11.755828
iteration 900 / 1500: loss 9.386926
iteration 1000 / 1500: loss 7.177540
iteration 1100 / 1500: loss 6.915250
iteration 1200 / 1500: loss 5.918573
iteration 1300 / 1500: loss 5.579504
iteration 1400 / 1500: loss 5.331826
That took 6.926160s
```
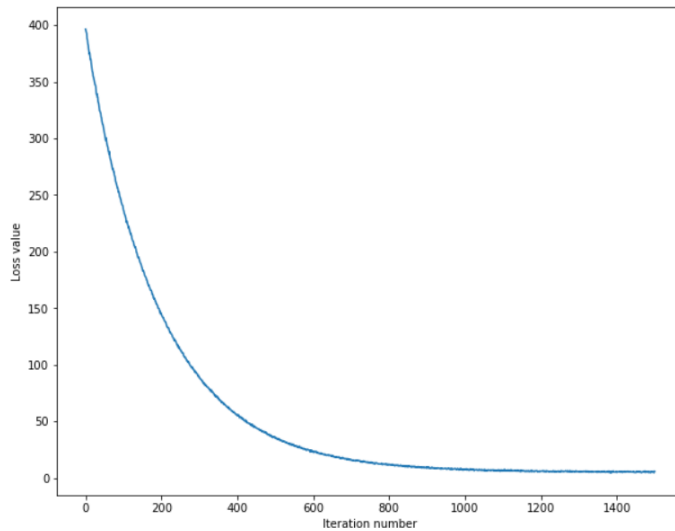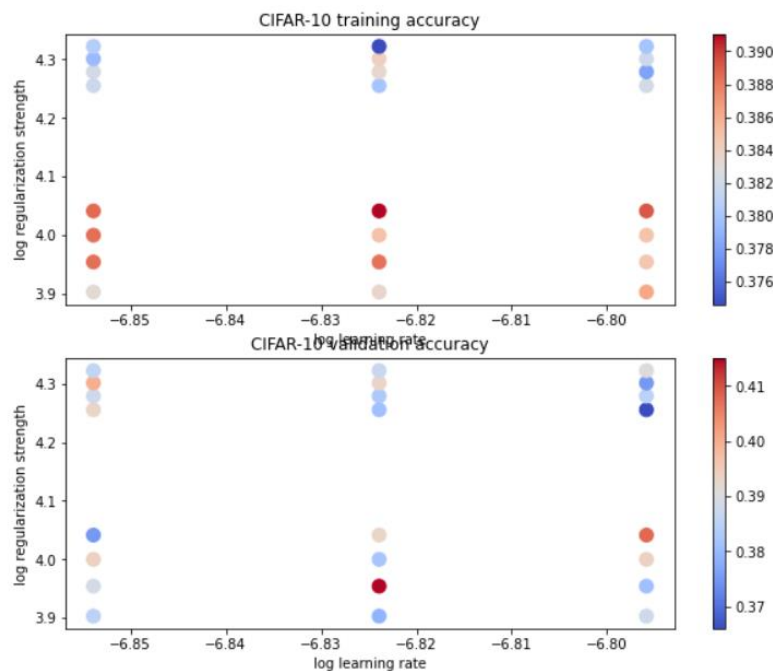

将模型用于验证集和测试集：

```
training accuracy: 0.381082
validation accuracy: 0.379000
```

交叉验证调参：

```
lr 1.600000e-07 reg 2.100000e+04 train accuracy: 0.379918 val accuracy: 0.391000
best validation accuracy achieved during cross-validation: 0.415000
```

可视化：

评估测试集最终的 accuracy：

```
:  # Evaluate the best svm on test set
   y_test_pred = best_svm.predict(X_test)
   test_accuracy = np.mean(y_test == y_test_pred)
   print('linear SVM on raw pixels final test set accuracy: %f' % test_accuracy)

   linear SVM on raw pixels final test set accuracy: 0.362000
```

3. Softmax classifier 实现
   导入数据集：

```
Train data shape:  (49000, 3073)
Train labels shape:  (49000,)
Validation data shape:  (1000, 3073)
Validation labels shape:  (1000,)
Test data shape:  (1000, 3073)
Test labels shape:  (1000,)
dev data shape:  (500, 3073)
dev labels shape:  (500,)
```

通过 naïve 和 vectorized 两种方法计算 Softmax Loss：

```
naive loss: 2.338143e+00 computed in 0.079806s
vectorized loss: 2.338143e+00 computed in 0.002995s
Loss difference: 0.000000
Gradient difference: 0.000000
```

结果相同但 Vectorized 更快。

交叉验证调整超参：

```
optimization time : 29.482837s
lr 1.000000e-07 reg 2.500000e+04 train accuracy: 0.251143 val accuracy: 0.251000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.241959 val accuracy: 0.243000
lr 5.000000e-07 reg 2.500000e+04 train accuracy: 0.314939 val accuracy: 0.315000
lr 5.000000e-07 reg 5.000000e+04 train accuracy: 0.311245 val accuracy: 0.301000
best validation accuracy achieved during cross-validation: 0.315000
```

计算测试集 accuracy：

```
# evaluate on test set
# Evaluate the best softmax on test set
y_test_pred = best_softmax.predict(X_test)
test_accuracy = np.mean(y_test == y_test_pred)
print('softmax on raw pixels final test set accuracy: %f' % (test_accuracy, ))
```

```
softmax on raw pixels final test set accuracy: 0.312000
```

4. 三层神经网络的实现

前向传播计算得分：

```
Your scores:
[[-0.03596154 -0.01613583 -0.00048556]
 [-0.09480192  0.20724618 -0.09763798]
 [-0.07015667  0.17081869 -0.07675745]
 [ 0.01473052  0.09321097 -0.0395178 ]
 [-0.05306489  0.04729807  0.01750587]]

correct scores:

Difference between your scores and correct scores:
4.807962381448306e-08
```

计算 loss：

```
Difference between your loss and correct loss:
7.829292769656604e-13
```

通过 numeric gradient 验证反向传播的准确性，得到 numeric 与 analytic 之间的差值：

```
W1 max relative error: 3.561318e-09
b1 max relative error: 7.670388e-09
W2 max relative error: 2.542856e-08
b2 max relative error: 4.590211e-10
W3 max relative error: 1.067978e-08
b3 max relative error: 8.037345e-11
```

使用 SGD 最小化损失：

Final training loss: 0.26699892976346745

Training Loss history
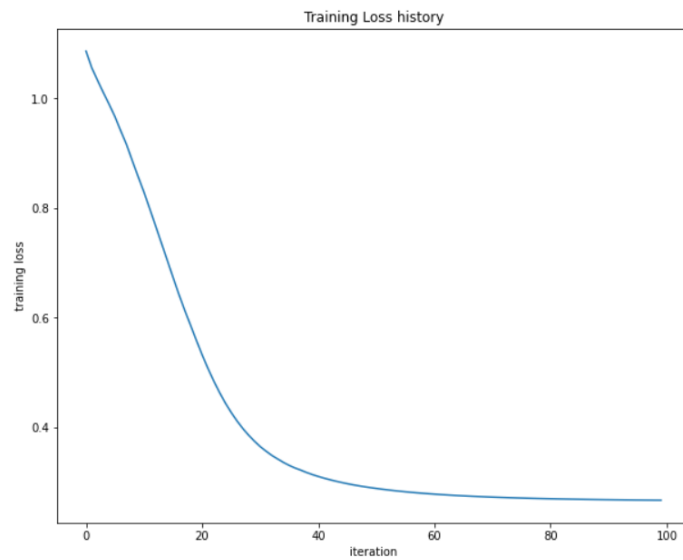
导入数据集，训练神经网络：

```
iteration 0 / 2000: loss 2.302716
iteration 100 / 2000: loss 2.302714
iteration 200 / 2000: loss 2.302714
iteration 300 / 2000: loss 2.302715
iteration 400 / 2000: loss 2.302713
iteration 500 / 2000: loss 2.302713
iteration 600 / 2000: loss 2.302707
iteration 700 / 2000: loss 2.302713
iteration 800 / 2000: loss 2.302712
iteration 900 / 2000: loss 2.302710
iteration 1000 / 2000: loss 2.302714
iteration 1100 / 2000: loss 2.302712
iteration 1200 / 2000: loss 2.302706
iteration 1300 / 2000: loss 2.302708
iteration 1400 / 2000: loss 2.302704
iteration 1500 / 2000: loss 2.302706
iteration 1600 / 2000: loss 2.302708
iteration 1700 / 2000: loss 2.302706
iteration 1800 / 2000: loss 2.302708
iteration 1900 / 2000: loss 2.302713
Validation accuracy: 0.079
```
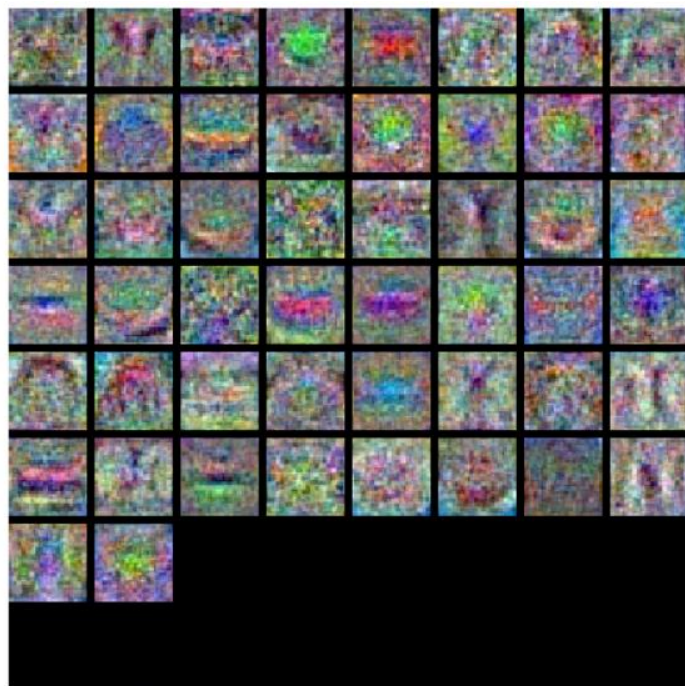
可视化损失下降过程和测试集验证集结果：
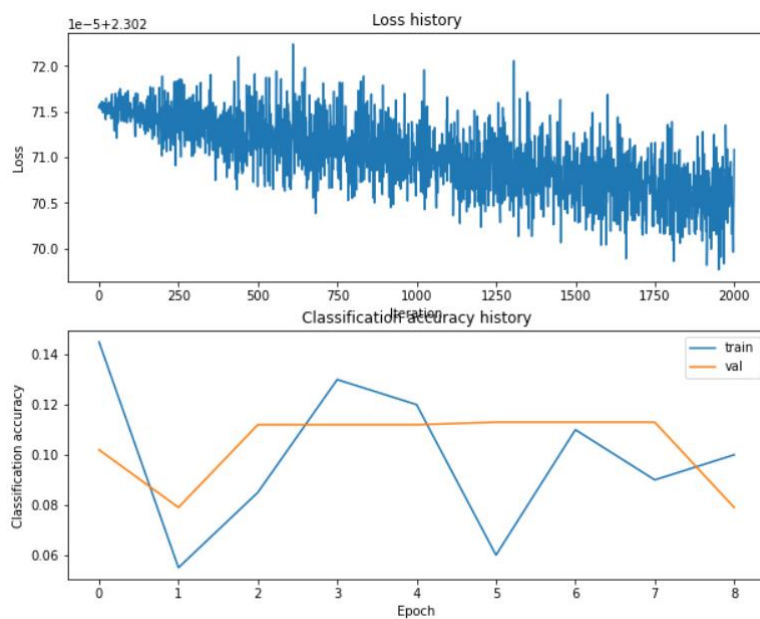
交叉验证调参，得到 accuracy 为 0.513：

```
iteration 9300 / 10000: loss 1.227002
iteration 9400 / 10000: loss 1.157315
iteration 9500 / 10000: loss 1.250118
iteration 9600 / 10000: loss 1.056515
iteration 9700 / 10000: loss 1.184403
iteration 9800 / 10000: loss 1.314326
iteration 9900 / 10000: loss 1.277896
0.513
```

可视化 best network：

Run on the test set:

```
: test_acc = (best_net.predict(X_test) == y_test).mean()
  print('Test accuracy: ', test_acc)

  Test accuracy:  0.509
```

得到测试集结果为 0.509

5. High-Level Features
   导入 CIFAR10 数据集，提取特征。
   训练基于特征的 SVM，交叉验证后最终结果如下：

```
lr 1.000000e-07 reg 5.000000e+05 train accuracy: 0.405490 val accuracy: 0.401000
lr 1.000000e-07 reg 5.000000e+06 train accuracy: 0.359245 val accuracy: 0.365000
best validation accuracy achieved during cross-validation: 0.409000
```

测试集结果：

```
# Evaluate your trained SVM on the test set
y_test_pred = best_svm.predict(X_test_feats)
test_accuracy = np.mean(y_test == y_test_pred)
print(test_accuracy)

0.422
```

部分预测结果：
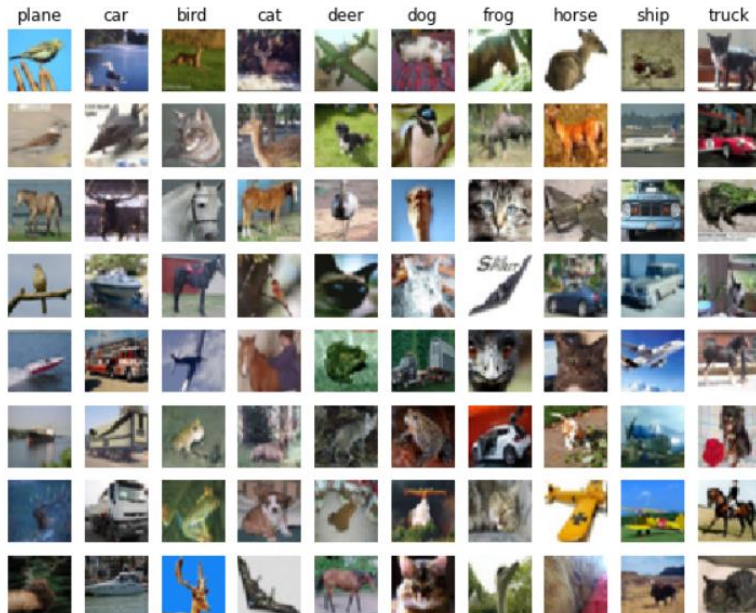
训练基于特征的神经网络，
设置学习率和正则化值：

```
learning_rates = [1e-4, 5e-3, 1e-3, 1e-2, 1]
regularization_strengths = [1e-4, 5e-4, 8e-4]
```

交叉验证结果：

```
lr: 1.000000e+00 reg: 1.000000e-04 val accuracy: 0.591000
lr: 1.000000e+00 reg: 5.000000e-04 val accuracy: 0.087000
lr: 1.000000e+00 reg: 8.000000e-04 val accuracy: 0.098000
lr: 1.000000e-01 reg: 1.000000e-04 val accuracy: 0.102000
```

测试集：

```
3]:   # Run your best neural net classifier on the test set. You should be able
      # to get more than 55% accuracy.

      test_acc = (best_net.predict(X_test_feats) == y_test).mean()
      print(test_acc)
```

0.556

结论分析与体会：
1. 通过本次实验，回顾了几种常见的分类方法实现：KNN、SVM、Softmax，学习了构建三层神经网络学习器的方法，对于反向传播算法的内涵有了更深入的理解。
2. 对于学习模型的参数调整的过程也进行重新的熟悉，使用 SGD 最小化损失函数，通过交叉验证调整超参，最终运用于测试集查看模型训练的准确性。
3. 学习了对于各种训练模型的效率优化算法，对于提高模型学习效率有很大的帮助。
4. 通过对 CIFAR 数据集模型训练结果的直观体验，感受到了模型训练的实际效果与不足之处。

就实验过程中遇到和出现的问题，你是如何解决和处理的，自拟 1－3 道问答题：
1. 使用三层神经网络进行训练时，使用原始参数的效果不佳？
   通过不断的调整学习率与正则化项进行优化，增加迭代次数，可以实现更好的训练效果。