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：组合一个简单的图像分类管道		学号：201900150221
日期：9.29	班级： 19 智能	姓名： 张进华
Email：zjh15117117428@163.com		

实验目的：

在本作业中，练习组合一个简单的图像分类管道，基于 k-最近邻或 SVM/Softmax 和 Three-Layer Neural Network 分类器，实现图像分类并比较不同算法下的准确率

实验软件和硬件环境：

Visual studio Code python 3.9.7

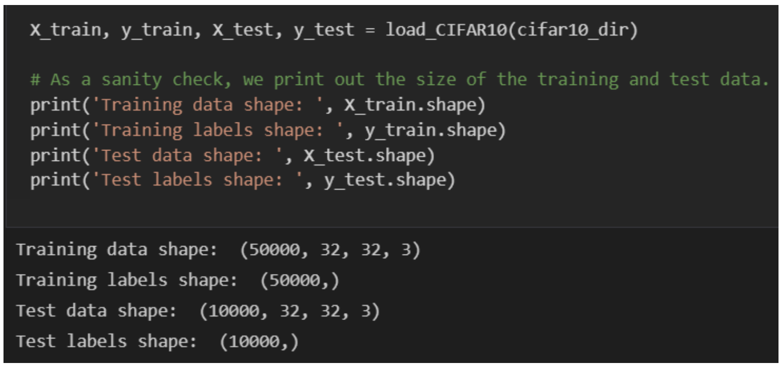
Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz   2.59 GHz

实验原理和方法

步骤一 : k-Nearest Neighbor classifier

1. 数据加载

加载 CIFAR-10 数据，并打印训练集和测试集的图片、标签尺寸

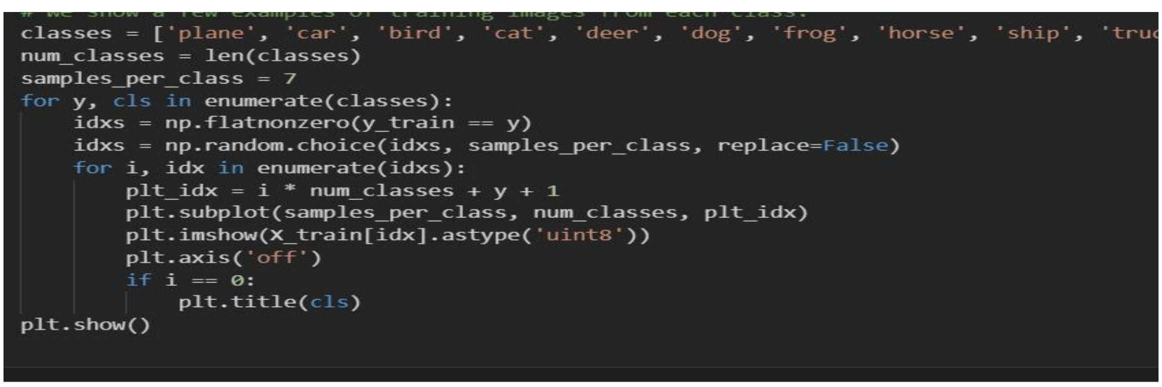


```
X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

# As a sanity check, we print out the size of the training and test data.
print('Training data shape: ', X_train.shape)
print('Training labels shape: ', y_train.shape)
print('Test data shape: ', X_test.shape)
print('Test labels shape: ', y_test.shape)
```

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

2. 可视化部分数据

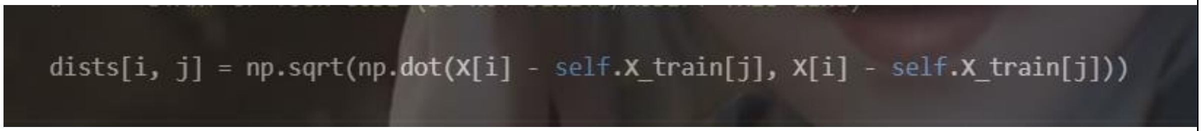
对于训练集中的 10 类数据，每类随机取出 7 张并可视化



```
classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truc
num_classes = len(classes)
samples_per_class = 7
for y, cls in enumerate(classes):
    idxs = np.flatnonzero(y_train == y)
    idxs = np.random.choice(idxs, samples_per_class, replace=False)
    for i, idx in enumerate(idxs):
        plt_idx = i * num_classes + y + 1
        plt.subplot(samples_per_class, num_classes, plt_idx)
        plt.imshow(X_train[idx].astype('uint8'))
        plt.axis('off')
        if i == 0:
            plt.title(cls)
plt.show()
```

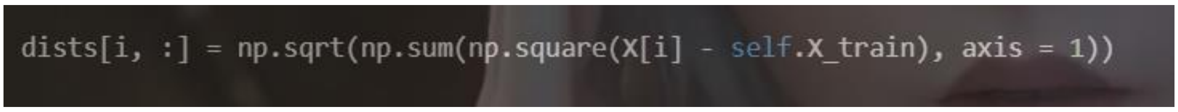
3. 编辑 k_nearest_neighbor.py 文件

先打开分类器文件夹里的 k_nearest_neighbor.py 文件： 首先填写函数 compute_distances_two_loops，计算数据集之间的样本距离，公式就是差的平方再开根号



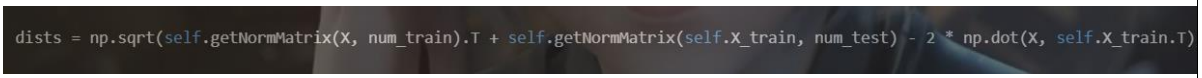
```
dists[i, j] = np.sqrt(np.dot(X[i] - self.X_train[j], X[i] - self.X_train[j]))
```

填写函数 compute_distances_one_loop，用于计算输入和数据集中的其他已知标签的距 离，也是上面的公式，只是系数稍微有一些改变



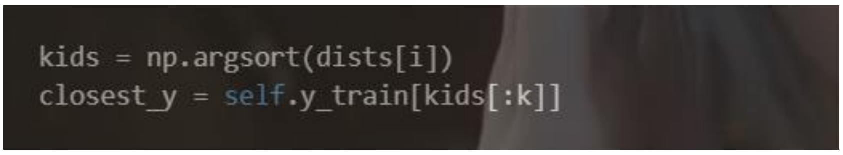
```
dists[i, :] = np.sqrt(np.sum(np.square(X[i] - self.X_train), axis = 1))
```

再然后填写 compute_distances_no_loop 函数，用于计算输入 x 和数据集的距离，并且不使用循环。直接用矩阵存放结果，利用 numpy 的函数进行计算



```
dists = np.sqrt(self.getNormMatrix(X, num_train).T + self.getNormMatrix(self.X_train, num_test) - 2 * np.dot(X, self.X_train.T)
```

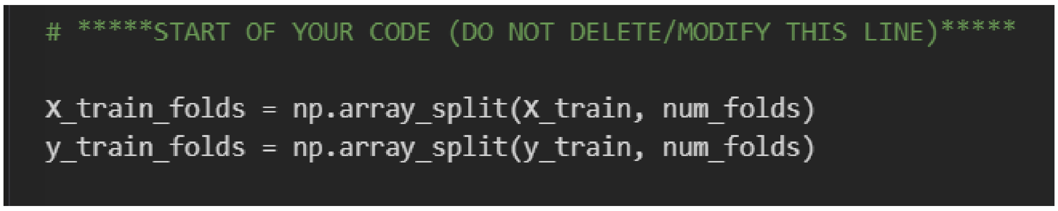
填写 predict_labels 这个函数，先拿出距离最近的 k 个



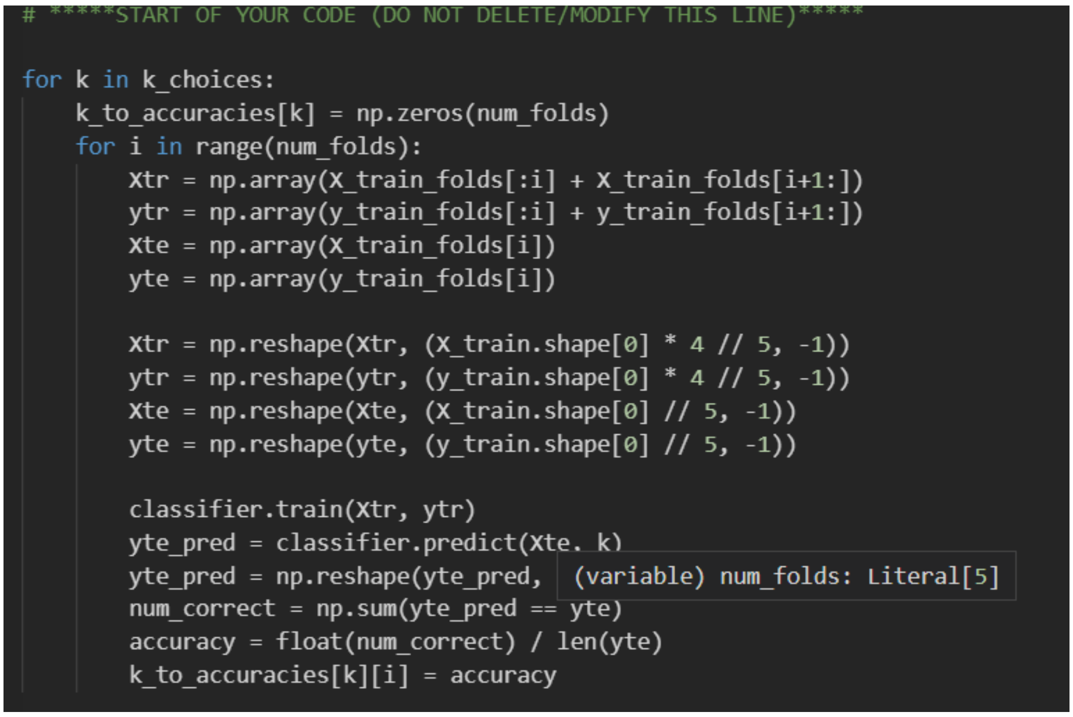
```
kids = np.argsort(dists[i])
closest_y = self.y_train[kids[:k]]
```

4. 交叉验证

建立 k—交叉验证算法，将数据分成好几组，可以单独训练。并使用不同的 k 值来计算哪个 k 的预测结果最好



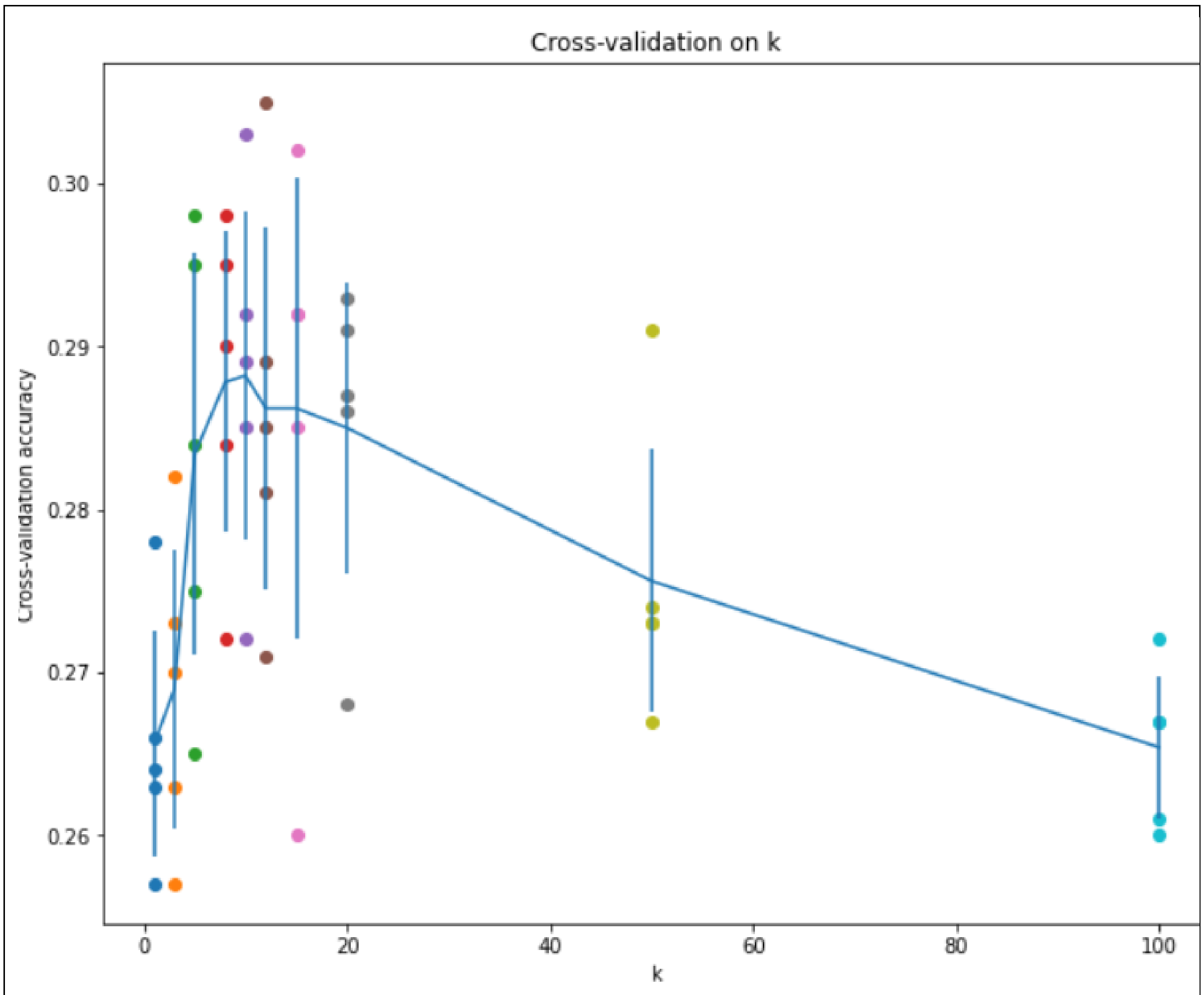
```
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

X_train_folds = np.array_split(X_train, num_folds)
y_train_folds = np.array_split(y_train, num_folds)
```



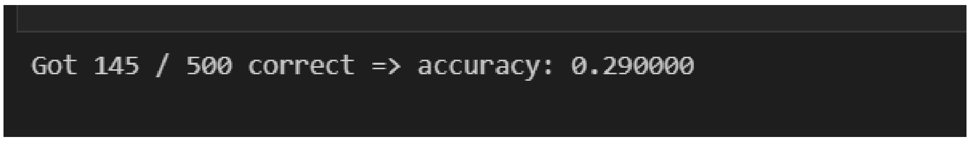
```
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

for k in k_choices:
    k_to_accuracies[k] = np.zeros(num_folds)
    for i in range(num_folds):
        Xtr = np.array(X_train_folds[:i] + X_train_folds[i+1:])
        ytr = np.array(y_train_folds[:i] + y_train_folds[i+1:])
        Xte = np.array(X_train_folds[i])
        yte = np.array(y_train_folds[i])

        Xtr = np.reshape(Xtr, (X_train.shape[0] * 4 // 5, -1))
        ytr = np.reshape(ytr, (y_train.shape[0] * 4 // 5, -1))
        Xte = np.reshape(Xte, (X_train.shape[0] // 5, -1))
        yte = np.reshape(yte, (y_train.shape[0] // 5, -1))

        classifier.train(Xtr, ytr)
        yte_pred = classifier.predict(Xte, k)
        yte_pred = np.reshape(yte_pred,  (variable) num_folds: Literal[5]
        num_correct = np.sum(yte_pred == yte)
        accuracy = float(num_correct) / len(yte)
        k_to_accuracies[k][i] = accuracy
```

5. 选择最佳的 k，训练分类器

通过可视化每一个 K 的误差，选择最佳的 K 重新训练分类器



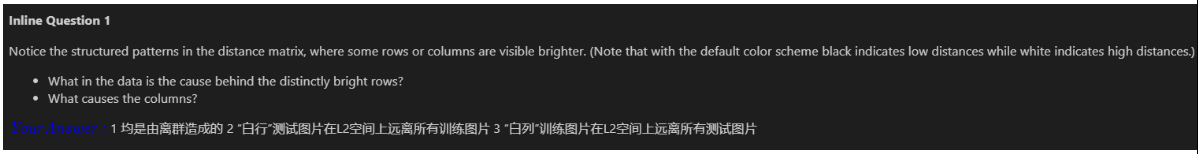
选择最佳 k = 6,训练后得出准确率为 0.29



```
Got 145 / 500 correct => accuracy: 0.290000
```

6 .问题回答

问题 1：



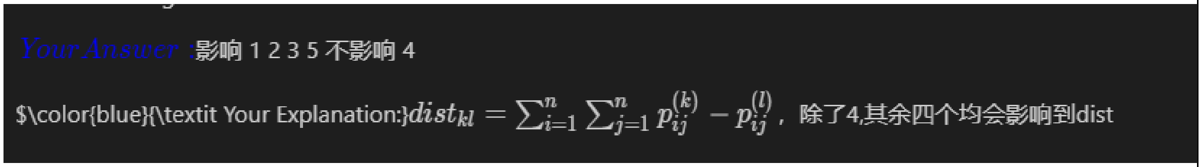
Inline Question 1

Notice the structured patterns in the distance matrix, where some rows or columns are visible brighter. (Note that with the default color scheme black indicates low distances while white indicates high distances.)

- What in the data is the cause behind the distinctly bright rows?
- What causes the columns?

Your Answer : 1 均是由离群造成的 2 "白行"测试图片在L2空间上远离所有训练图片 3 "白列"训练图片在L2空间上远离所有测试图片

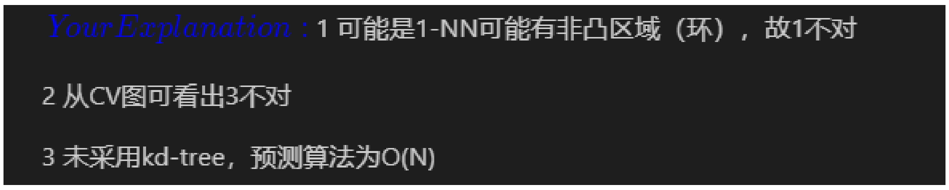
问题 2：



*Your Answer :*影响 1 2 3 5 不影响 4

$\color{blue}{\textit Your Explanation:}$$dist_{kl} = \sum_{i=1}^{n} \sum_{j=1}^{n} p_{ij}^{(k)} - p_{ij}^{(l)}$，除了4,其余四个均会影响到dist

问题 3：


$YourExplanation:$ 1 可能是1-NN可能有非凸区域（环），故1不对

2 从CV图可看出3不对

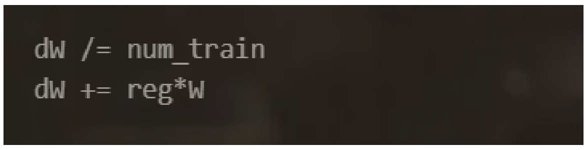
3 未采用kd-tree，预测算法为O(N)

步骤 2 Training a Support Vector Machine

1. Linear_svm.py 文件的填写

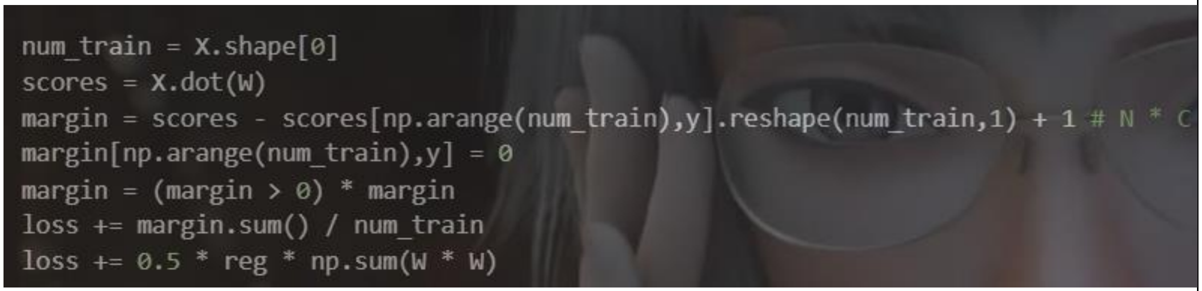
首先是 svm_loss_naive ,其作用是用 naive 的方式计算当前模型的 loss 计算在 w*x 操作下和目标的差别距离，如果有一定的差别，就加上相应的 loss，填充 dw 进行参数更新。
损失函数定义如下:

$$L_i = \sum_{j \neq y_i} \left[\max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \right]$$



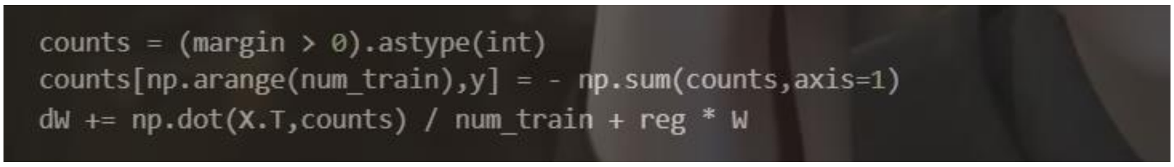
```
dW /= num_train
dW += reg*W
```

填写 svm_loss_vectorized,用向量的方式完成，计算其 loss，填写计算距离，并更新 loss



```
num_train = X.shape[0]
scores = X.dot(W)
margin = scores - scores[np.arange(num_train),y].reshape(num_train,1) + 1 # N * C
margin[np.arange(num_train),y] = 0
margin = (margin > 0) * margin
loss += margin.sum() / num_train
loss += 0.5 * reg * np.sum(W * W)
```

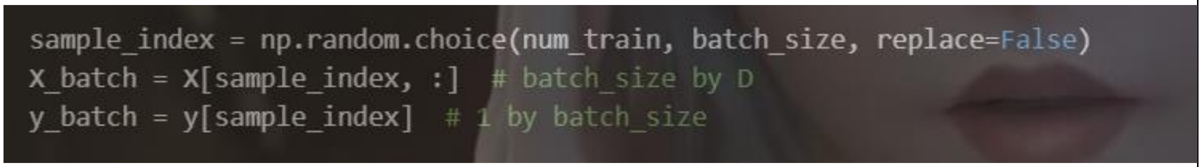
及根据计算的 margin 更新 dw 的值



```
counts = (margin > 0).astype(int)
counts[np.arange(num_train),y] = - np.sum(counts,axis=1)
dW += np.dot(X.T,counts) / num_train + reg * W
```

2. 填写 linear_classifier.py 文件

填写 train 函数，先是划分专门的训练集和测试集



```
sample_index = np.random.choice(num_train, batch_size, replace=False)
X_batch = X[sample_index, :]  # batch_size by D
y_batch = y[sample_index]  # 1 by batch_size
```

SGD 随机梯度下降

```
# evaluate loss and gradient
loss, grad = self.loss(X_batch, y_batch, reg)
loss_history.append(loss)
```

再计算真正要变化的 w，用学习率乘以函数返回的 dw

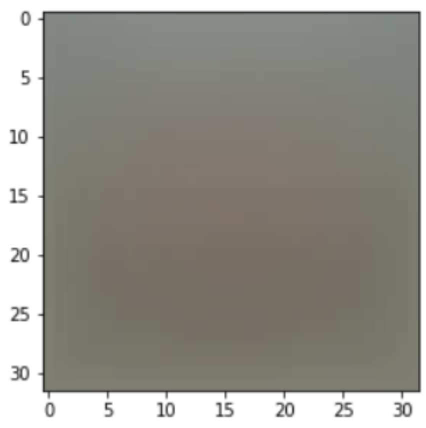
```
self.W -= learning_rate * grad
```

3. 数据预处理

将数据划分为 train 训练集、val 验证集、test 测试集以及 dev 试算集，这个试算集就是一个小样本来测试程序是否能够正常运行的。这里在选取的测试集的时候我们是不能够让 val 测试集和 train 训练集有交集的，这样才能达到随机的效果。

```
# Split the data into train, val, and test sets. In addition we will
# create a small development set as a subset of the training data;
# we can use this for development so our code runs faster.
num_training = 49000
num_validation = 1000
num_test = 1000
num_dev = 500

# Our validation set will be num_validation points from the original
# training set.
mask = range(num_training, num_training + num_validation)
X_val = X_train[mask]
y_val = y_train[mask]
```

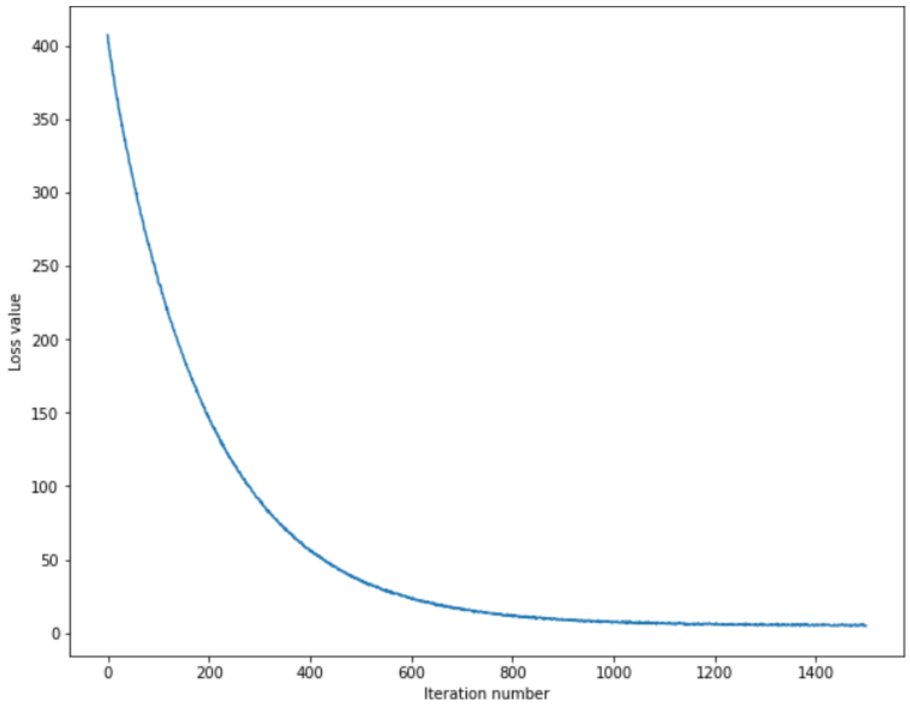
展示一下训练样本的像素点均值绘制出来的图像



```
(49000, 3073) (1000, 3073) (1000, 3073) (500, 3073)
```

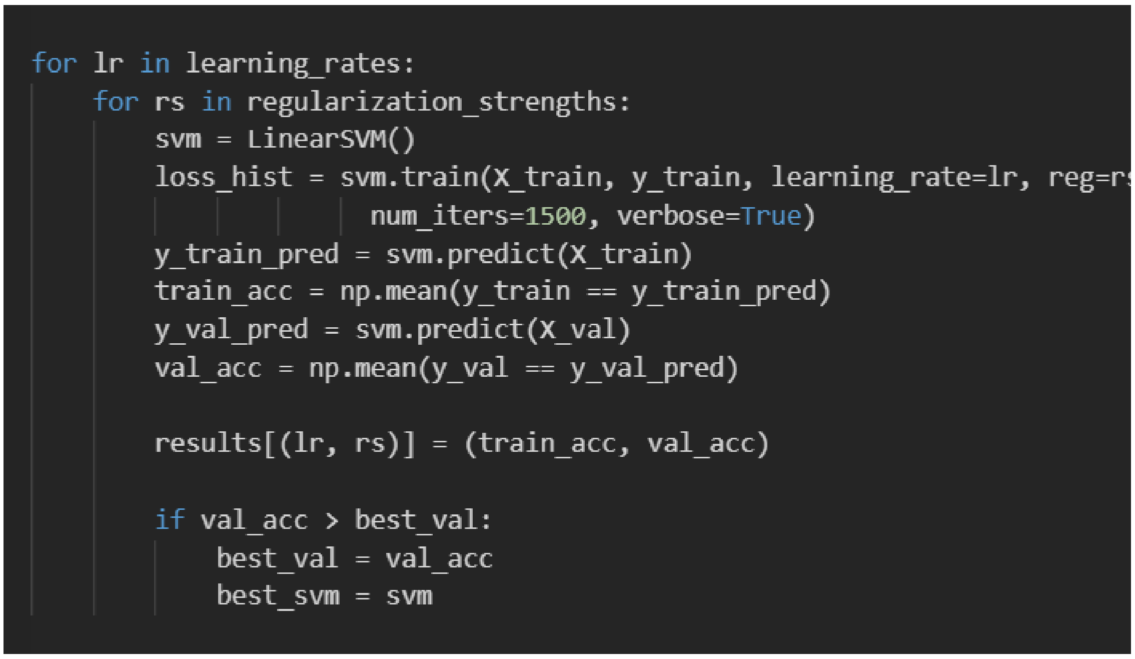
4. 完成并运行 SVM

在训练的过程中，loss 逐渐减小，模型也更加贴近数据集。training accuracy: 0.38 validation accuracy: 0.37 换上不同的学习率和不同的长度，计算最终的预测正确率，从而选择最好的参数。



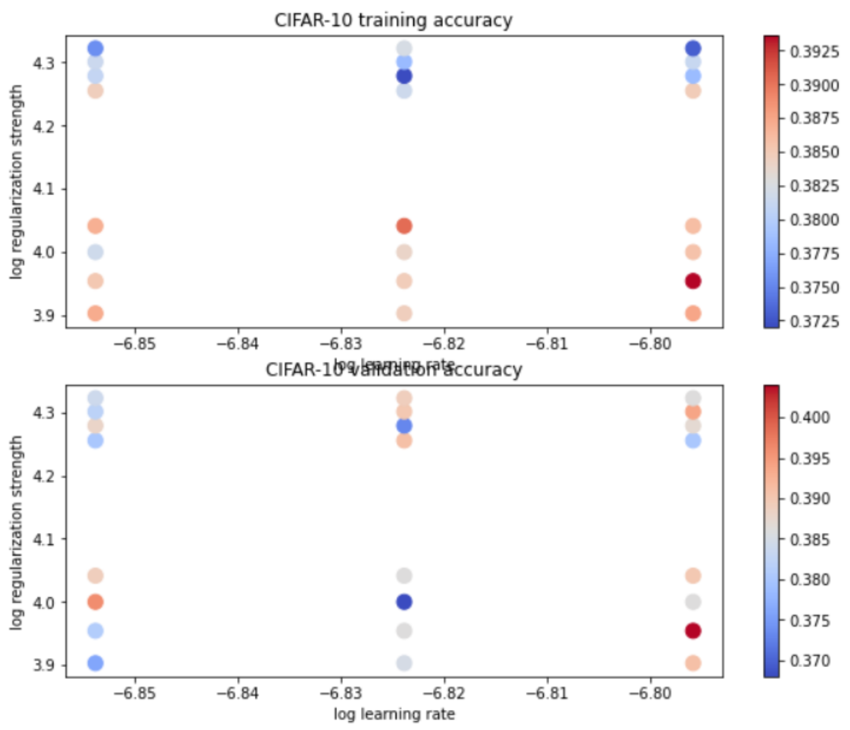
5.交叉验证

循环不同的参数进行预测，然后不断更新保留最准确的参数

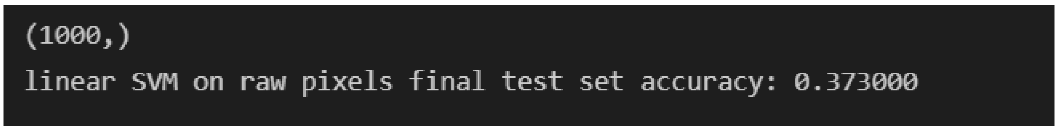


```
for lr in learning_rates:
    for rs in regularization_strengths:
        svm = LinearSVM()
        loss_hist = svm.train(X_train, y_train, learning_rate=lr, reg=r
                              num_iters=1500, verbose=True)
        y_train_pred = svm.predict(X_train)
        train_acc = np.mean(y_train == y_train_pred)
        y_val_pred = svm.predict(X_val)
        val_acc = np.mean(y_val == y_val_pred)

        results[(lr, rs)] = (train_acc, val_acc)

        if val_acc > best_val:
            best_val = val_acc
            best_svm = svm
```

展示出了不同参数下的预测精度的波动



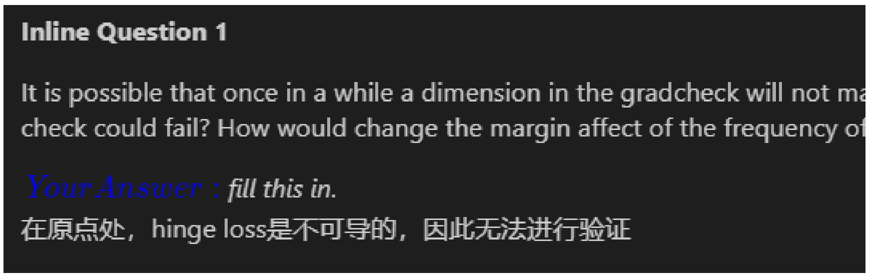
最终得到最好的预测率是 0.40400 最后直接用最好的 svm 进行最终预测，并达到了 0.373 的正确率



```
(1000,)
linear SVM on raw pixels final test set accuracy: 0.373000
```

6.问题回答

问题 1：

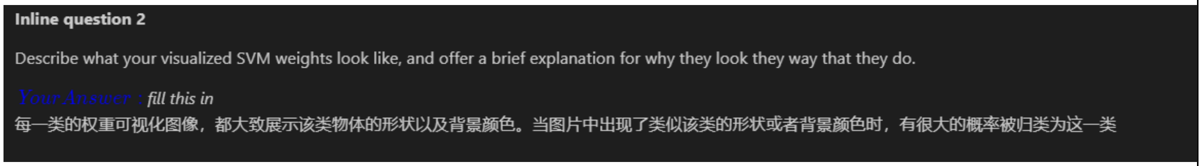


Inline Question 1

It is possible that once in a while a dimension in the gradcheck will not ma
check could fail? How would change the margin affect of the frequency of

Your Answer : *fill this in.*
在原点处，hinge loss是不可导的，因此无法进行验证

问题 2：



Inline question 2

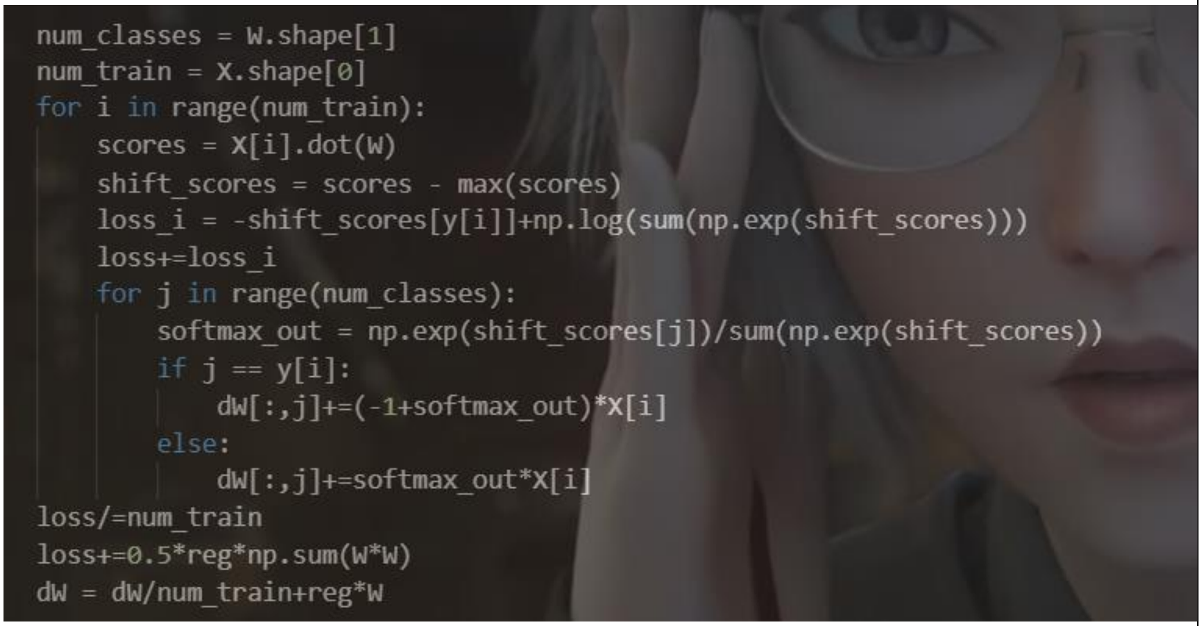
Describe what your visualized SVM weights look like, and offer a brief explanation for why they look they way that they do.

Your Answer : *fill this in*
每一类的权重可视化图像，都大致展示该类物体的形状以及背景颜色。当图片中出现了类似该类的形状或者背景颜色时，有很大的概率被归类为这一类

步骤 3 : Implement a Softmax classifier

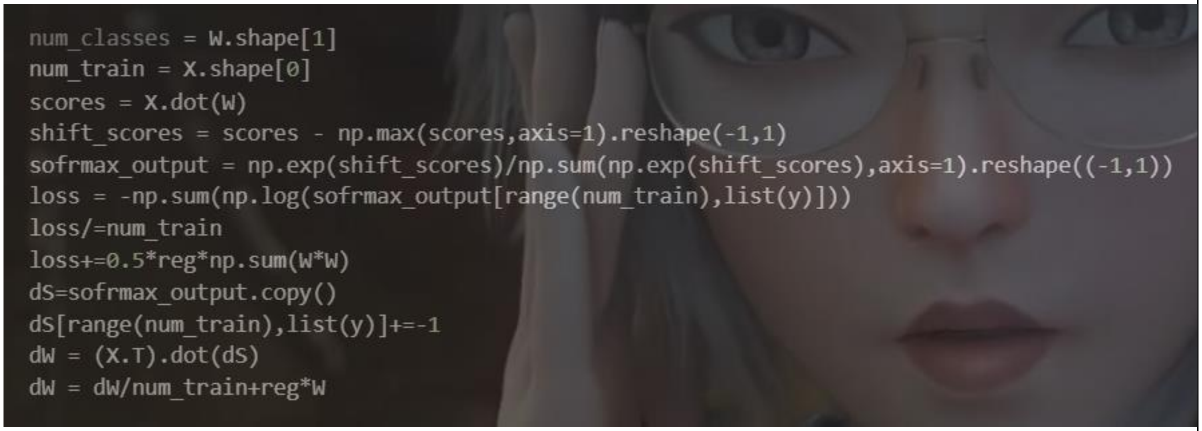
1. 修改 softmax.py 文件

修改 softmax.py 函数，用 navie 方式计算 loss。即用循环计算



```
num_classes = W.shape[1]
num_train = X.shape[0]
for i in range(num_train):
    scores = X[i].dot(W)
    shift_scores = scores - max(scores)
    loss_i = -shift_scores[y[i]]+np.log(sum(np.exp(shift_scores)))
    loss+=loss_i
    for j in range(num_classes):
        softmax_out = np.exp(shift_scores[j])/sum(np.exp(shift_scores))
        if j == y[i]:
            dW[:,j]+=(-1+softmax_out)*X[i]
        else:
            dW[:,j]+=softmax_out*X[i]
loss/=num_train
loss+=0.5*reg*np.sum(W*W)
dW = dW/num_train+reg*W
```

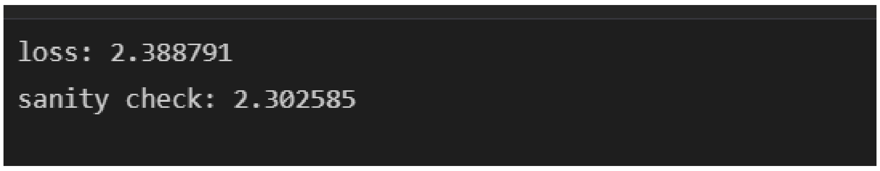
接下来实现向量化 的 loss 计算



```
num_classes = W.shape[1]
num_train = X.shape[0]
scores = X.dot(W)
shift_scores = scores - np.max(scores,axis=1).reshape(-1,1)
sofrmax_output = np.exp(shift_scores)/np.sum(np.exp(shift_scores),axis=1).reshape((-1,1))
loss = -np.sum(np.log(sofrmax_output[range(num_train),list(y)]))
loss/=num_train
loss+=0.5*reg*np.sum(W*W)
dS=sofrmax_output.copy()
dS[range(num_train),list(y)]+=-1
dW = (X.T).dot(dS)
dW = dW/num_train+reg*W
```

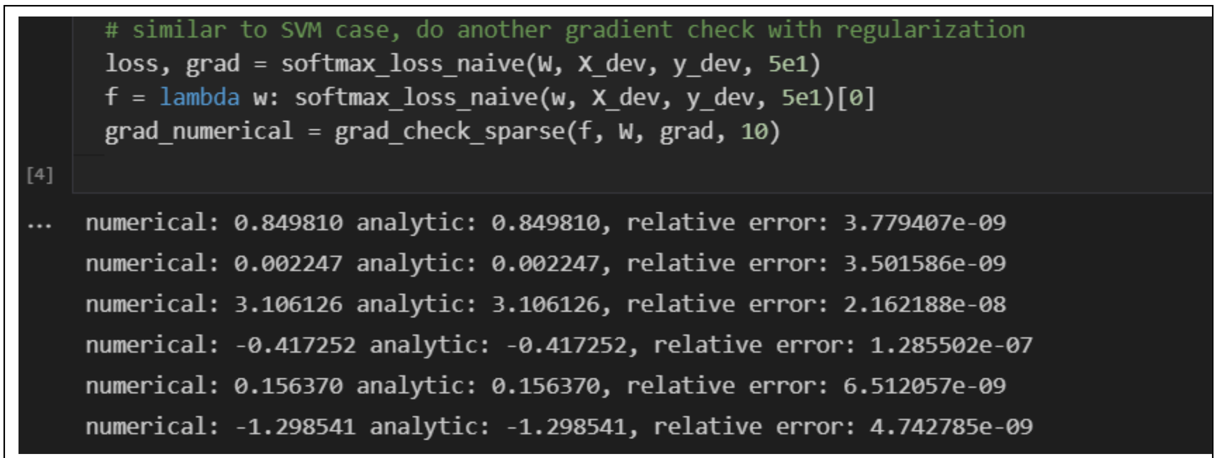
2.运行 softmax.ipynb 文件

运行作业里的代码进行运算，展示循环计算得到的结果



```
loss: 2.388791
sanity check: 2.302585
```

检验计算结果是否正确，通过和导数定义求解进行对比，如果无明显差异则认为结果一致

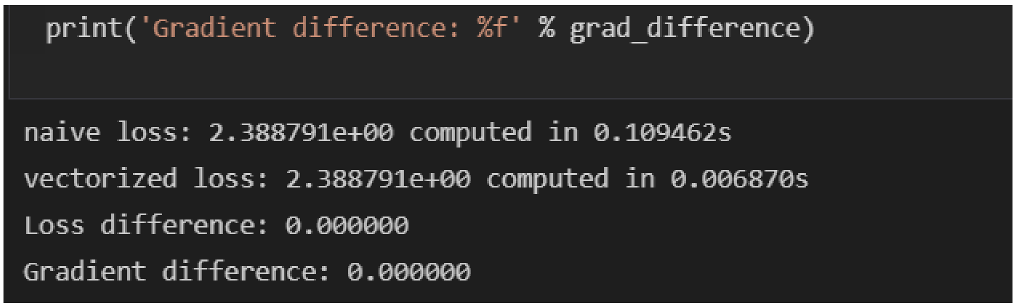


```
# similar to SVM case, do another gradient check with regularization
loss, grad = softmax_loss_naive(W, X_dev, y_dev, 5e1)
f = lambda w: softmax_loss_naive(w, X_dev, y_dev, 5e1)[0]
grad_numerical = grad_check_sparse(f, W, grad, 10)
```

[4]

```
...  numerical: 0.849810 analytic: 0.849810, relative error: 3.779407e-09
     numerical: 0.002247 analytic: 0.002247, relative error: 3.501586e-09
     numerical: 3.106126 analytic: 3.106126, relative error: 2.162188e-08
     numerical: -0.417252 analytic: -0.417252, relative error: 1.285502e-07
     numerical: 0.156370 analytic: 0.156370, relative error: 6.512057e-09
     numerical: -1.298541 analytic: -1.298541, relative error: 4.742785e-09
```

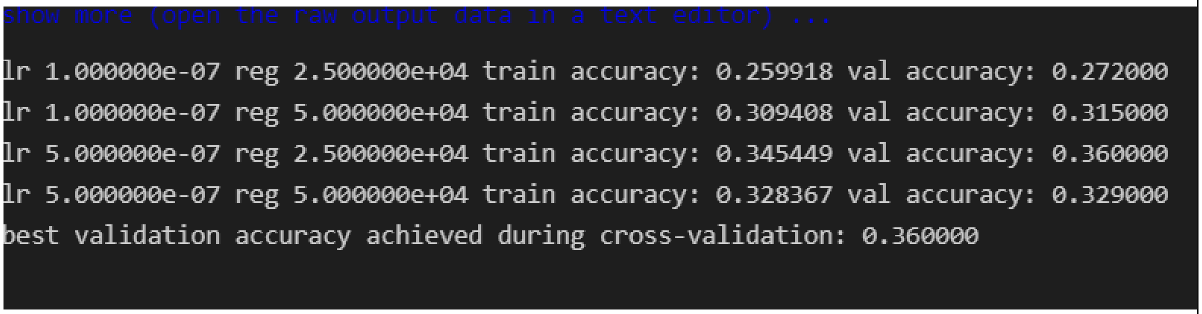
从结果来看整体的差异非常小，所以我们认为梯度的计算是正确的。然后再比较向量计算与循环计算是否有差异



```
print('Gradient difference: %f' % grad_difference)
```

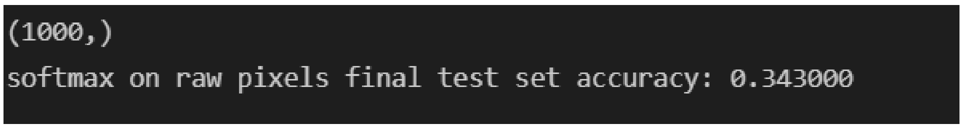
```
naive loss: 2.388791e+00 computed in 0.109462s
vectorized loss: 2.388791e+00 computed in 0.006870s
Loss difference: 0.000000
Gradient difference: 0.000000
```

可以看到向量计算与传统的循环计算在精度上没有差异，但在运行时间上向量计算的效率明显的提高。接下来通过对不同的学习率与正则系数进行交叉验证，得到最优的参数



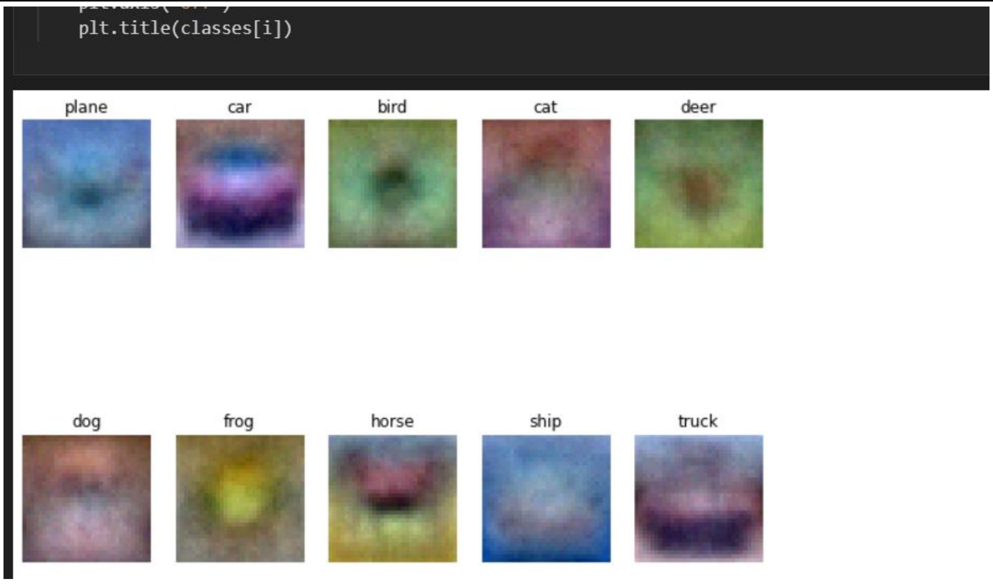
```
show more (open the raw output data in a text editor) ...

lr 1.000000e-07 reg 2.500000e+04 train accuracy: 0.259918 val accuracy: 0.272000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.309408 val accuracy: 0.315000
lr 5.000000e-07 reg 2.500000e+04 train accuracy: 0.345449 val accuracy: 0.360000
lr 5.000000e-07 reg 5.000000e+04 train accuracy: 0.328367 val accuracy: 0.329000
best validation accuracy achieved during cross-validation: 0.360000
```

可以看到最好参数的预测准确度是 0.360，那么我们将最好的参数代入 test 集看一下计算效果，得到 0.343 的正确率

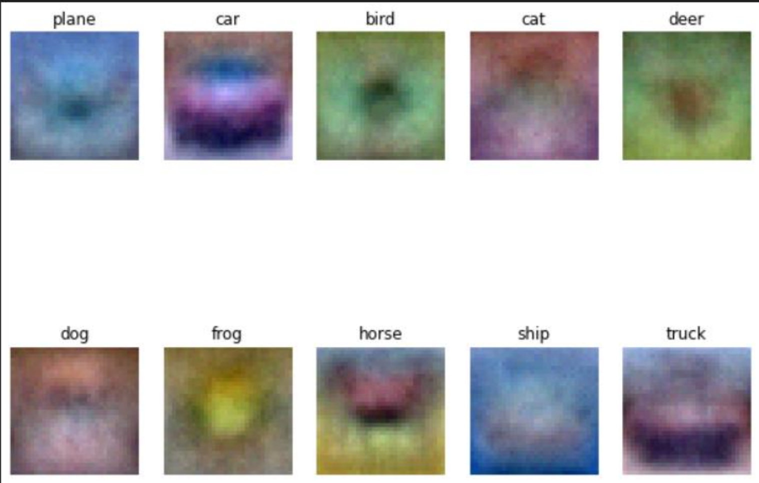


```
(1000,)
softmax on raw pixels final test set accuracy: 0.343000
```

并输出每一类的模板

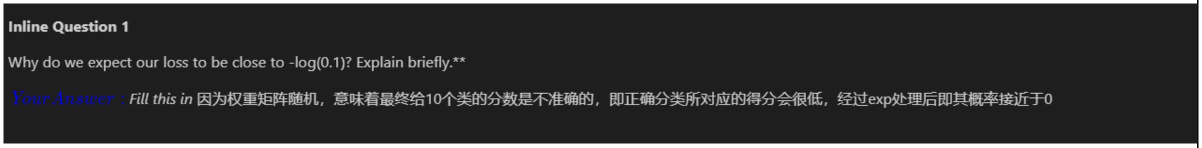


```
    plt.title(classes[i])
```



3.问题回答

问题 1：

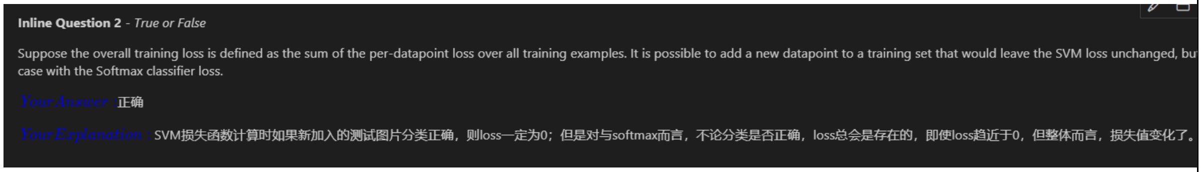


Inline Question 1

Why do we expect our loss to be close to -log(0.1)? Explain briefly.**

Your Answer : *Fill this in* 因为权重矩阵随机，意味着最终给10个类的分数是不准确的，即正确分类所对应的得分会很低，经过exp处理后即其概率接近于0

问题 2：



Inline Question 2 - *True or False*

Suppose the overall training loss is defined as the sum of the per-datapoint loss over all training examples. It is possible to add a new datapoint to a training set that would leave the SVM loss unchanged, bu
case with the Softmax classifier loss.

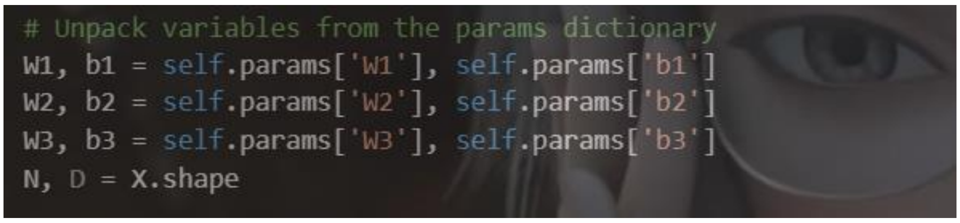
Your Answer :正确

Your Explanation :SVM损失函数计算时如果新加入的测试图片分类正确，则loss一定为0；但是对与softmax而言，不论分类是否正确，loss总会是存在的，即使loss趋近于0，但整体而言，损失值变化了。

步骤四 Three-Layer Neural Network

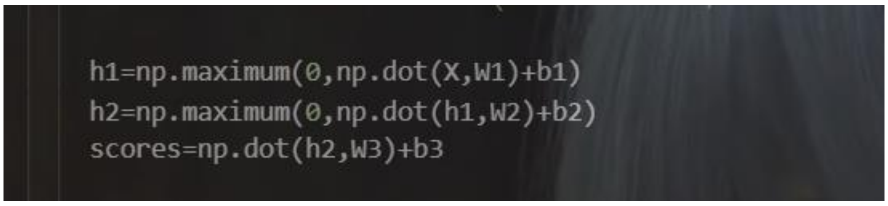
1．编写得分、损失、梯度函数

建立不同的神经层，每个神经层上有多个神经元，利用参数，激活函数进行预测。首先是 loss 函数中的 scores 的计算，在 neural_net.py 并对其中的任务进行编辑，定义每一层的权重和偏置



```
# Unpack variables from the params dictionary
W1, b1 = self.params['W1'], self.params['b1']
W2, b2 = self.params['W2'], self.params['b2']
W3, b3 = self.params['W3'], self.params['b3']
N, D = X.shape
```

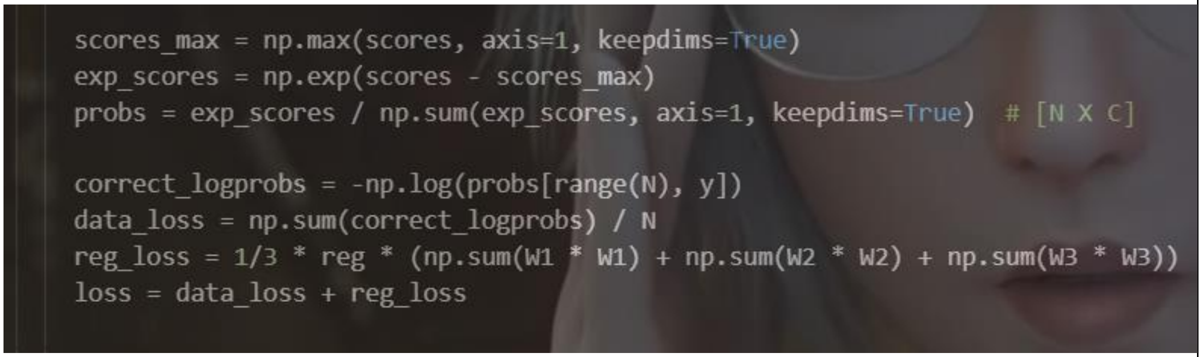
对应输出



```
h1=np.maximum(0,np.dot(X,W1)+b1)
h2=np.maximum(0,np.dot(h1,W2)+b2)
scores=np.dot(h2,W3)+b3
```

2. 实现 loss 向前传播的过程

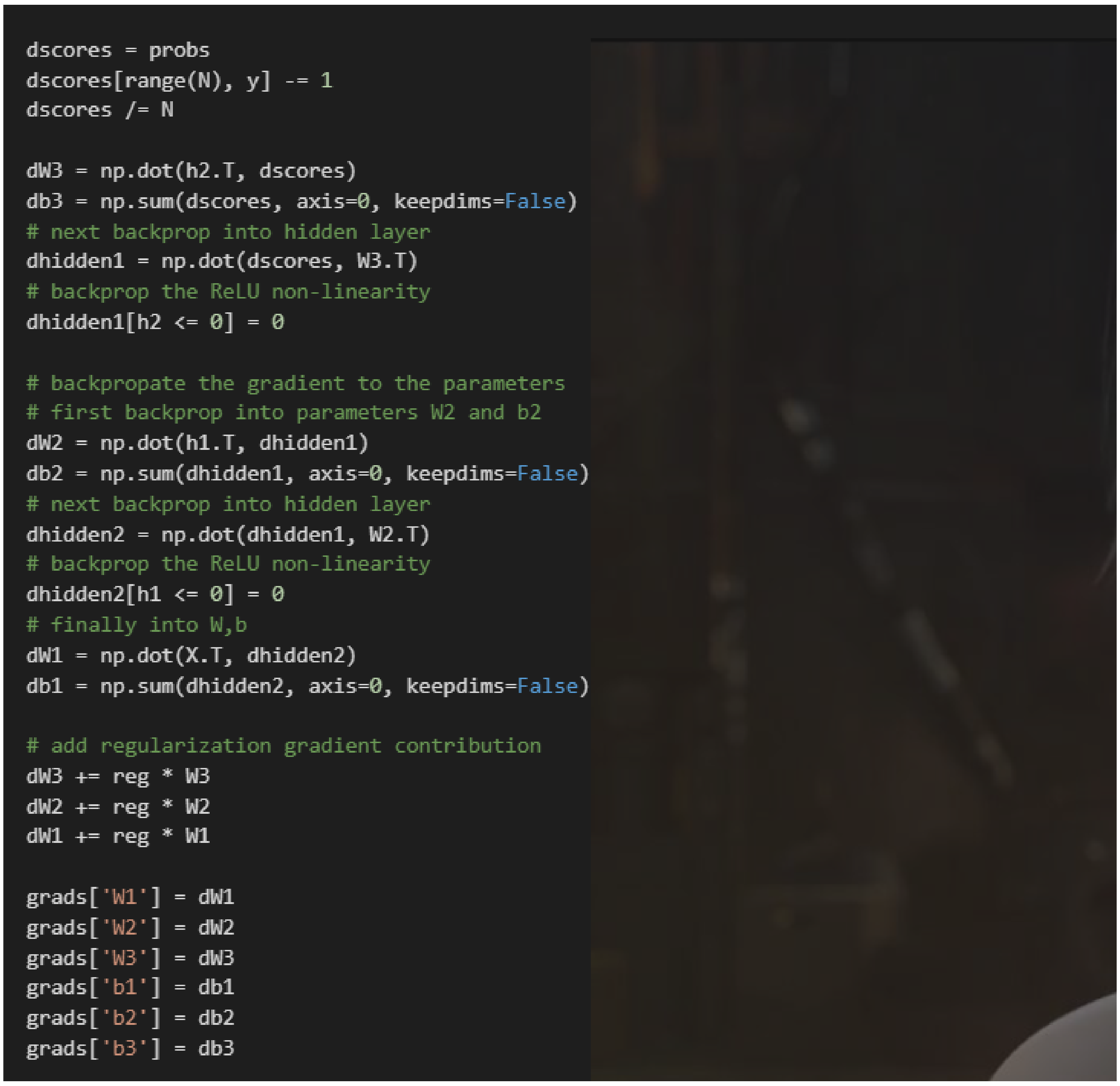
损失函数是通过计算得分函数的变形得到



```
scores_max = np.max(scores, axis=1, keepdims=True)
exp_scores = np.exp(scores - scores_max)
probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)  # [N X C]

correct_logprobs = -np.log(probs[range(N), y])
data_loss = np.sum(correct_logprobs) / N
reg_loss = 1/3 * reg * (np.sum(W1 * W1) + np.sum(W2 * W2) + np.sum(W3 * W3))
loss = data_loss + reg_loss
```

3. Three-Layer Neural Network 编写

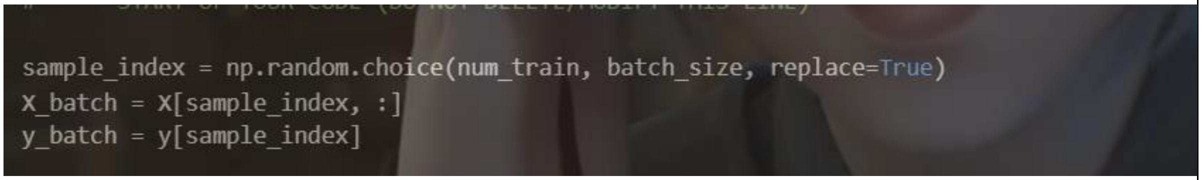
计算完得分函数、损失函数后就是需要计算梯度来对参数进行更新



```
dscores = probs
dscores[range(N), y] -= 1
dscores /= N

dW3 = np.dot(h2.T, dscores)
db3 = np.sum(dscores, axis=0, keepdims=False)
# next backprop into hidden layer
dhidden1 = np.dot(dscores, W3.T)
# backprop the ReLU non-linearity
dhidden1[h2 <= 0] = 0

# backpropate the gradient to the parameters
# first backprop into parameters W2 and b2
dW2 = np.dot(h1.T, dhidden1)
db2 = np.sum(dhidden1, axis=0, keepdims=False)
# next backprop into hidden layer
dhidden2 = np.dot(dhidden1, W2.T)
# backprop the ReLU non-linearity
dhidden2[h1 <= 0] = 0
# finally into W,b
dW1 = np.dot(X.T, dhidden2)
db1 = np.sum(dhidden2, axis=0, keepdims=False)

# add regularization gradient contribution
dW3 += reg * W3
dW2 += reg * W2
dW1 += reg * W1

grads['W1'] = dW1
grads['W2'] = dW2
grads['W3'] = dW3
grads['b1'] = db1
grads['b2'] = db2
grads['b3'] = db3
```

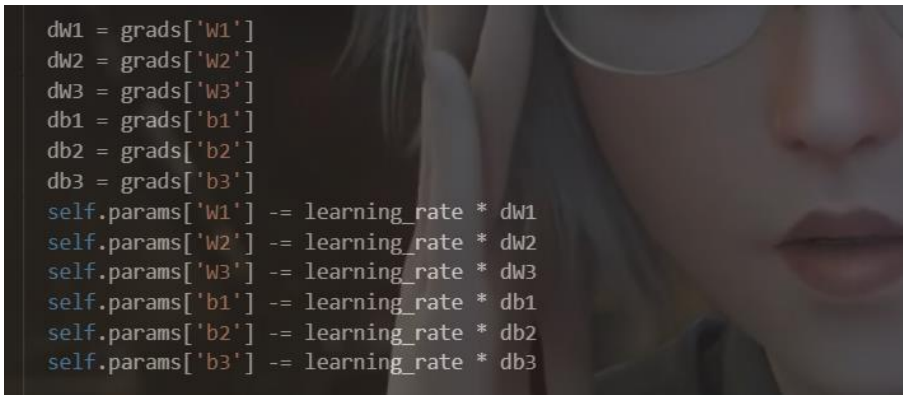
4．训练和预测函数的编写

在计算完梯度后需要去更新权重训练模型，需要编写训练函数，但这里的只需要完成随机梯度下降的小样本选择以及参数的更新过程的代码



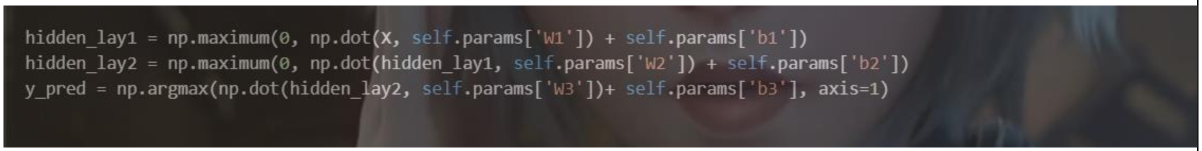
```
sample_index = np.random.choice(num_train, batch_size, replace=True)
X_batch = X[sample_index, :]
y_batch = y[sample_index]
```

更新参数



```
dW1 = grads['W1']
dW2 = grads['W2']
dW3 = grads['W3']
db1 = grads['b1']
db2 = grads['b2']
db3 = grads['b3']
self.params['W1'] -= learning_rate * dW1
self.params['W2'] -= learning_rate * dW2
self.params['W3'] -= learning_rate * dW3
self.params['b1'] -= learning_rate * db1
self.params['b2'] -= learning_rate * db2
self.params['b3'] -= learning_rate * db3
```

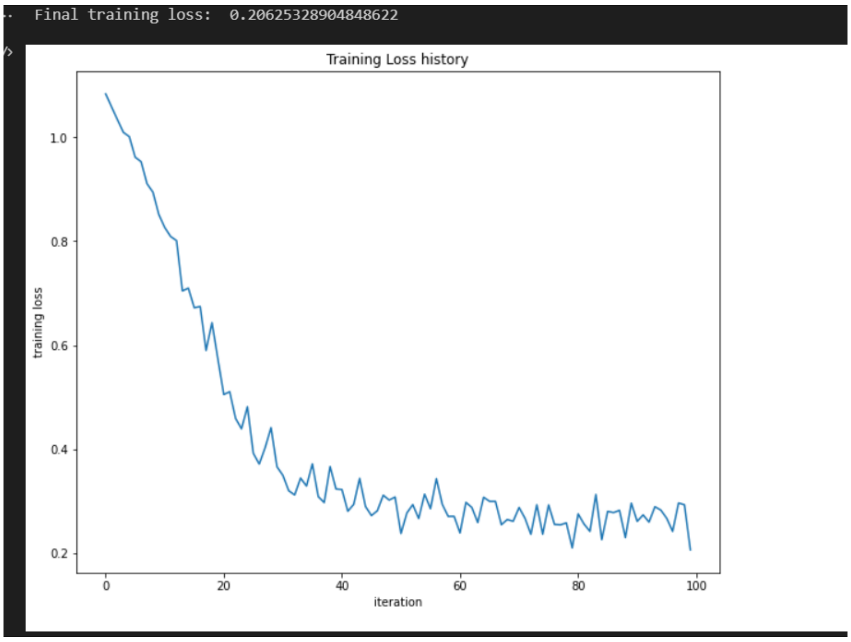
预测的函数填写



```
hidden_lay1 = np.maximum(0, np.dot(X, self.params['W1']) + self.params['b1'])
hidden_lay2 = np.maximum(0, np.dot(hidden_lay1, self.params['W2']) + self.params['b2'])
y_pred = np.argmax(np.dot(hidden_lay2, self.params['W3'])+ self.params['b3'], axis=1)
```

5.测试数据损失可视化

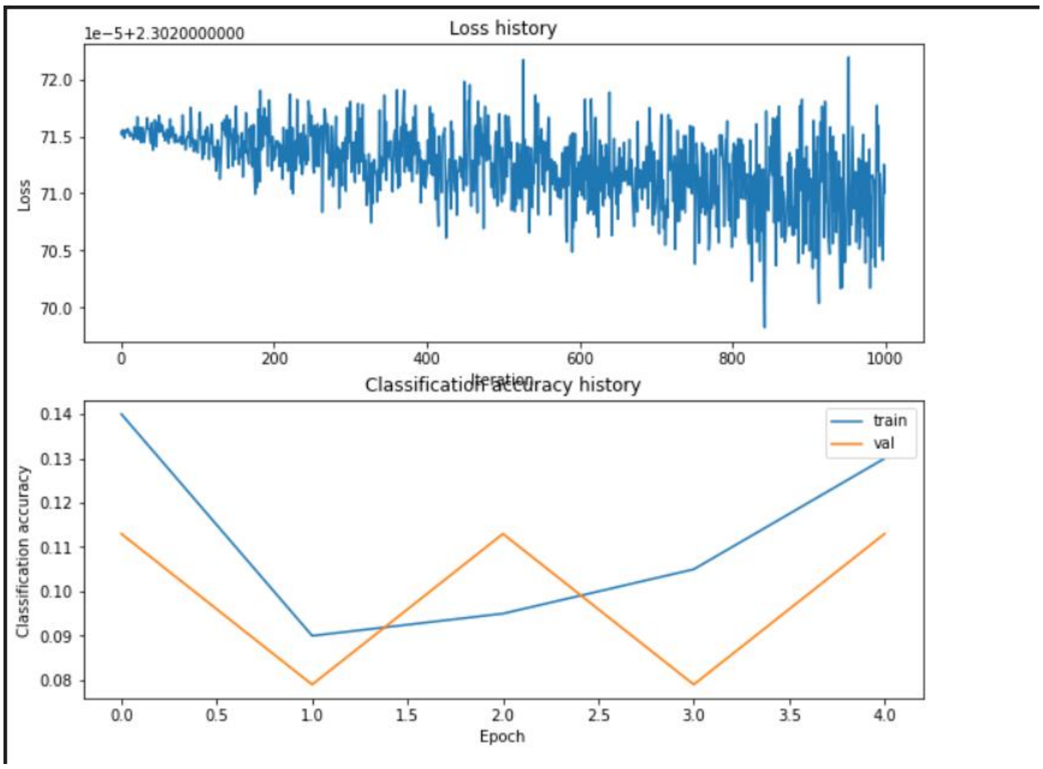
开始训练模型更新参数，观察一下其损失函数的变化，图上可以看到随着迭代次数的增加，损失的值飞速下降，在 15 次跌倒后稳定在 0 的上面，所以整个训练过程展现出来的结果是可以接受的



```
Final training loss:  0.20625328904848622
```

6. 对训练集调试

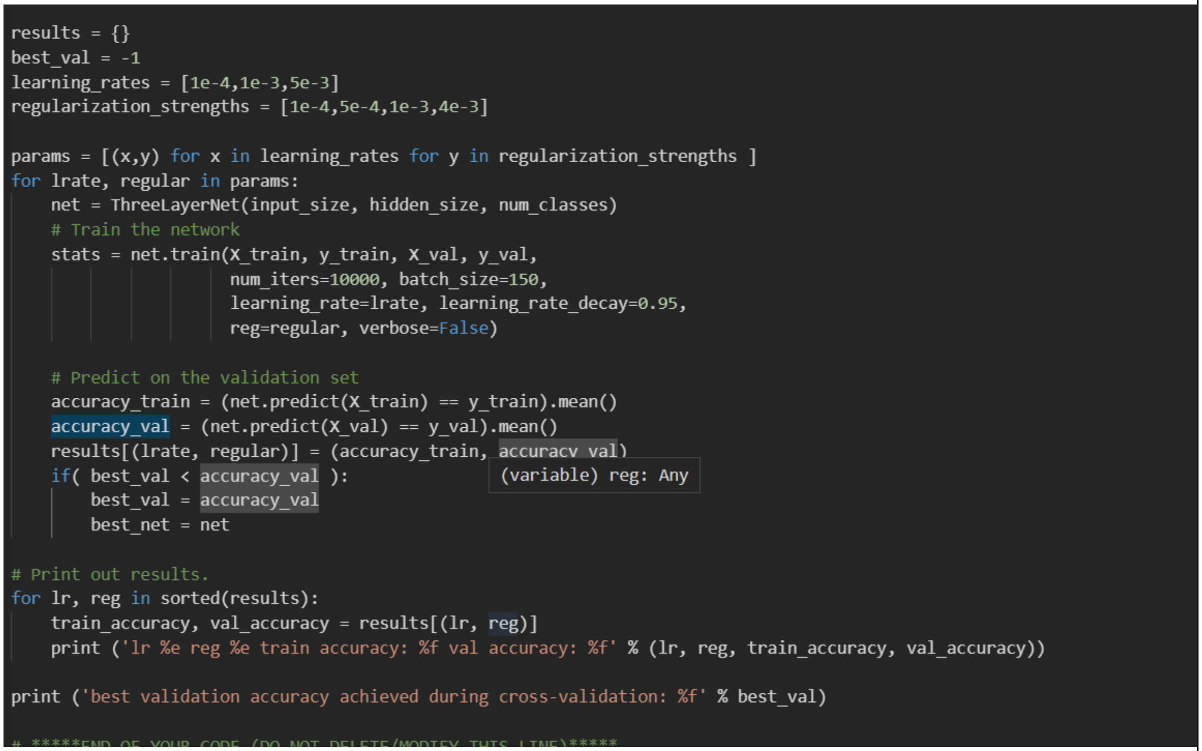
使用绘制 loss 、准确率曲线去观察整个的一个更新过程，或者绘制 W1 直观的从图像中看一下权重训练的样子，来判断一下模型的各方面参数的问题



从图上看到损失函数到第 200 次迭代之前都还没有开始明显的变化，与之前看到的损失函数形式不一样，这个导致的原因有可能是学习率过小导致迭代速度过慢。再看准确度函数就在 0.29 周围就开始平缓，可以适当增加隐藏层的神经元个数也就是维度来充分利用信息

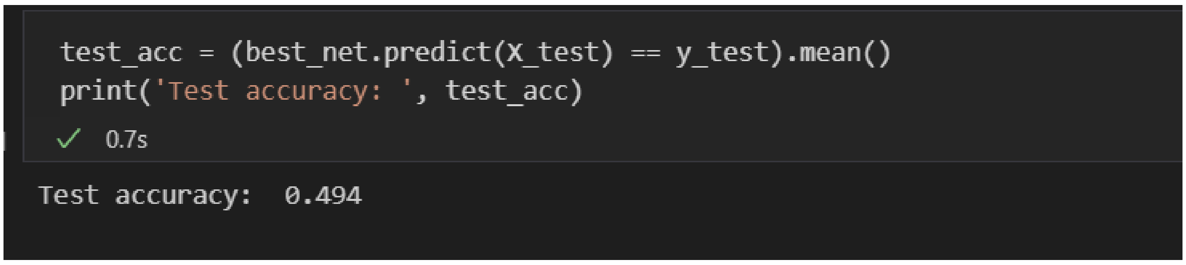
7. 交叉验证参数的调整

对不同的参数进行训练并进行验证，选择出验证效果最好的参数



```
results = {}
best_val = -1
learning_rates = [1e-4,1e-3,5e-3]
regularization_strengths = [1e-4,5e-4,1e-3,4e-3]

params = [(x,y) for x in learning_rates for y in regularization_strengths ]
for lrate, regular in params:
    net = ThreeLayerNet(input_size, hidden_size, num_classes)
    # Train the network
    stats = net.train(X_train, y_train, X_val, y_val,
                      num_iters=10000, batch_size=150,
                      learning_rate=lrate, learning_rate_decay=0.95,
                      reg=regular, verbose=False)

    # Predict on the validation set
    accuracy_train = (net.predict(X_train) == y_train).mean()
    accuracy_val = (net.predict(X_val) == y_val).mean()
    results[(lrate, regular)] = (accuracy_train, accuracy_val)
    if( best_val < accuracy_val ):
        best_val = accuracy_val
        best_net = net

# Print out results.
for lr, reg in sorted(results):
    train_accuracy, val_accuracy = results[(lr, reg)]
    print ('lr %e reg %e train accuracy: %f val accuracy: %f' % (lr, reg, train_accuracy, val_accuracy))

print ('best validation accuracy achieved during cross-validation: %f' % best_val)

# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

运行一下测试集检验



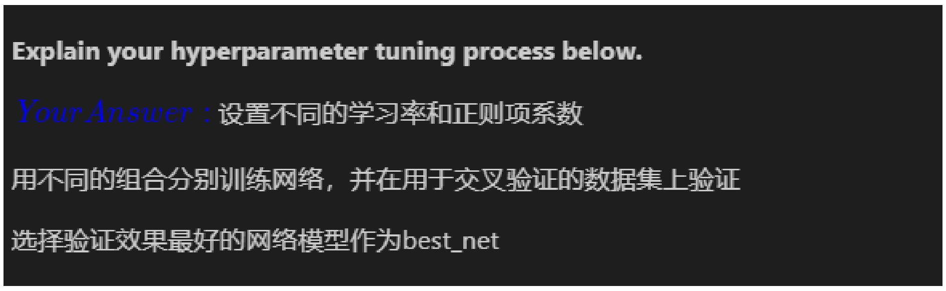
```
test_acc = (best_net.predict(X_test) == y_test).mean()
print('Test accuracy: ', test_acc)
```
✓ 0.7s

```
Test accuracy:  0.494
```

精度达到了 0.494

8.问题回答

问题 1：



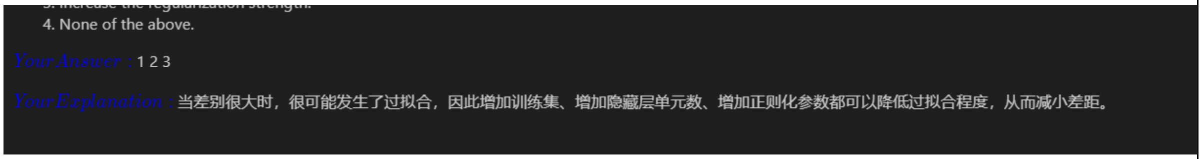
Explain your hyperparameter tuning process below.

Your Answer : 设置不同的学习率和正则项系数

用不同的组合分别训练网络，并在用于交叉验证的数据集上验证

选择验证效果最好的网络模型作为best_net

问题 2：



3. Increase the regularization strength.
4. None of the above.

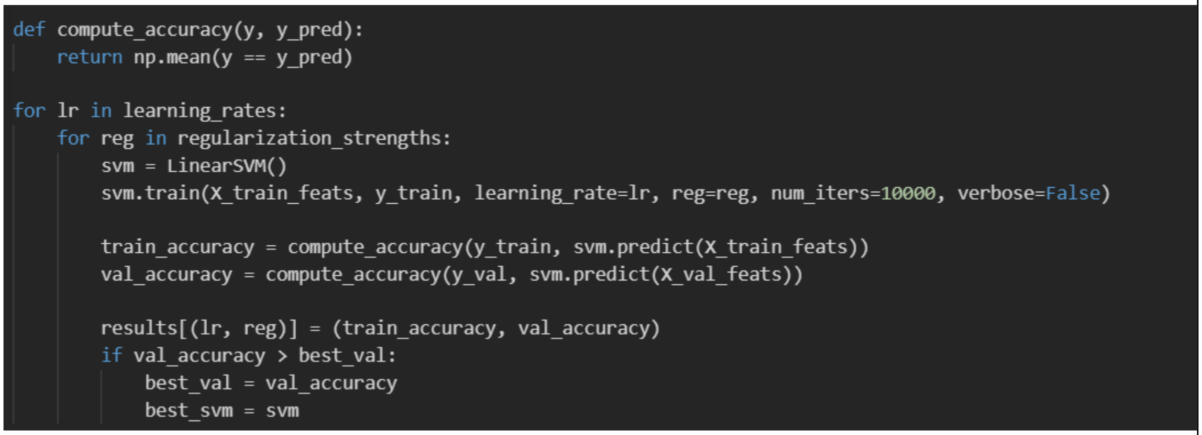
Your Answer : 1 2 3

Your Explanation : 当差别很大时，很可能发生了过拟合，因此增加训练集、增加隐藏层单元数、增加正则化参数都可以降低过拟合程度，从而减小差距。

步骤 5 : Image Features

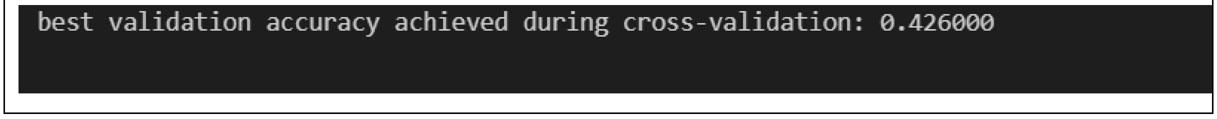
1.在特征上训练 SVM

使用验证集调整学习率和正则化强度



```
def compute_accuracy(y, y_pred):
    return np.mean(y == y_pred)

for lr in learning_rates:
    for reg in regularization_strengths:
        svm = LinearSVM()
        svm.train(X_train_feats, y_train, learning_rate=lr, reg=reg, num_iters=10000, verbose=False)

        train_accuracy = compute_accuracy(y_train, svm.predict(X_train_feats))
        val_accuracy = compute_accuracy(y_val, svm.predict(X_val_feats))

        results[(lr, reg)] = (train_accuracy, val_accuracy)
        if val_accuracy > best_val:
            best_val = val_accuracy
            best_svm = svm
```

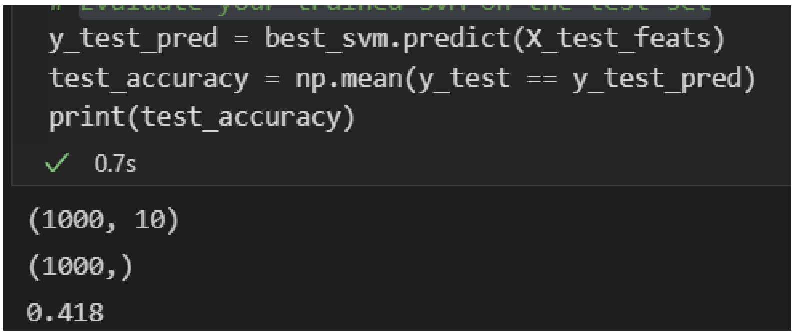
交叉验证期间实现的最佳验证准确度



```
best validation accuracy achieved during cross-validation: 0.426000
```

2.测试 SVM

在测试集上评估经过训练的 SVM,得出准确率为 0.418



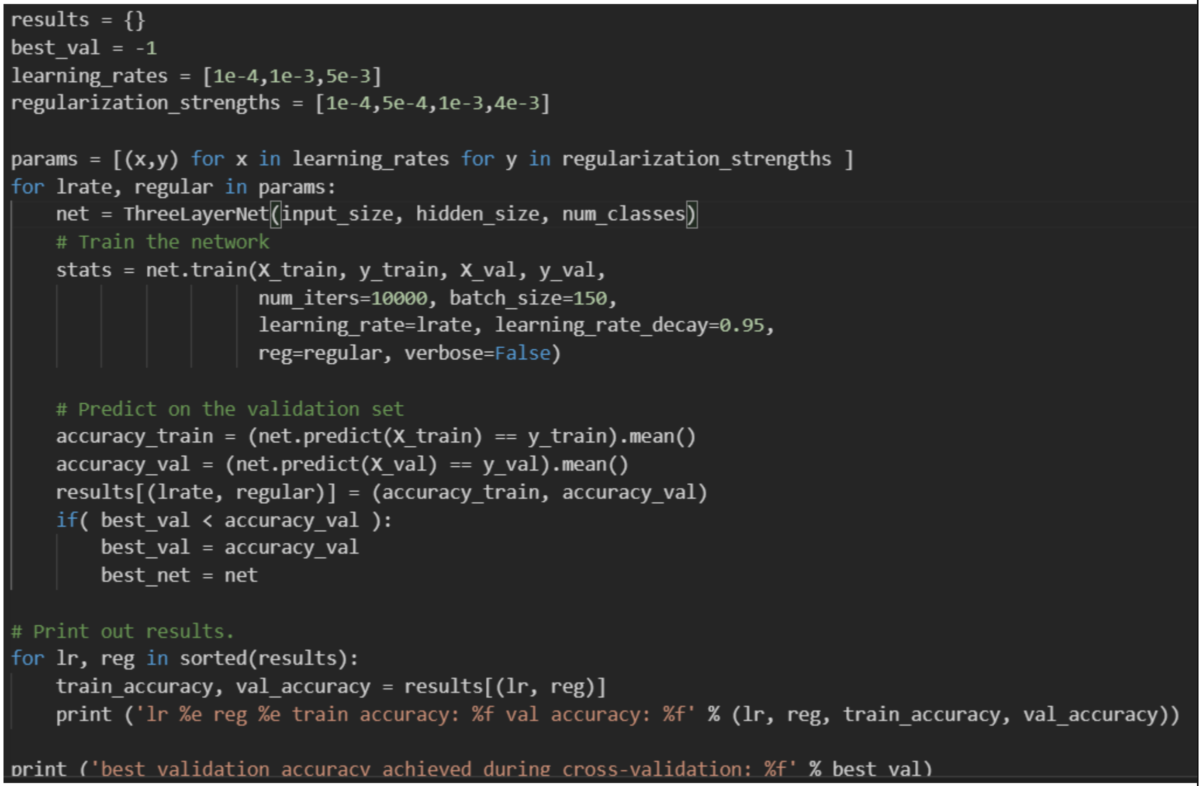
```
y_test_pred = best_svm.predict(X_test_feats)
test_accuracy = np.mean(y_test == y_test_pred)
print(test_accuracy)
```

✓ 0.7s

```
(1000, 10)
(1000,)
0.418
```

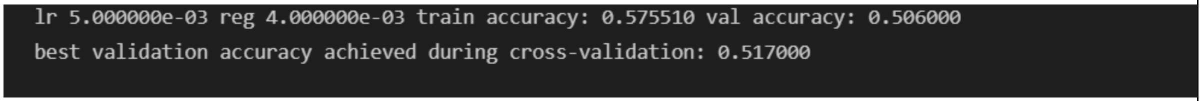
3. 训练 Three-Layer Neural Network

训练一个关于图像特征的三层神经网络，交叉验证各种参数，存储模型在 best_net 变量中



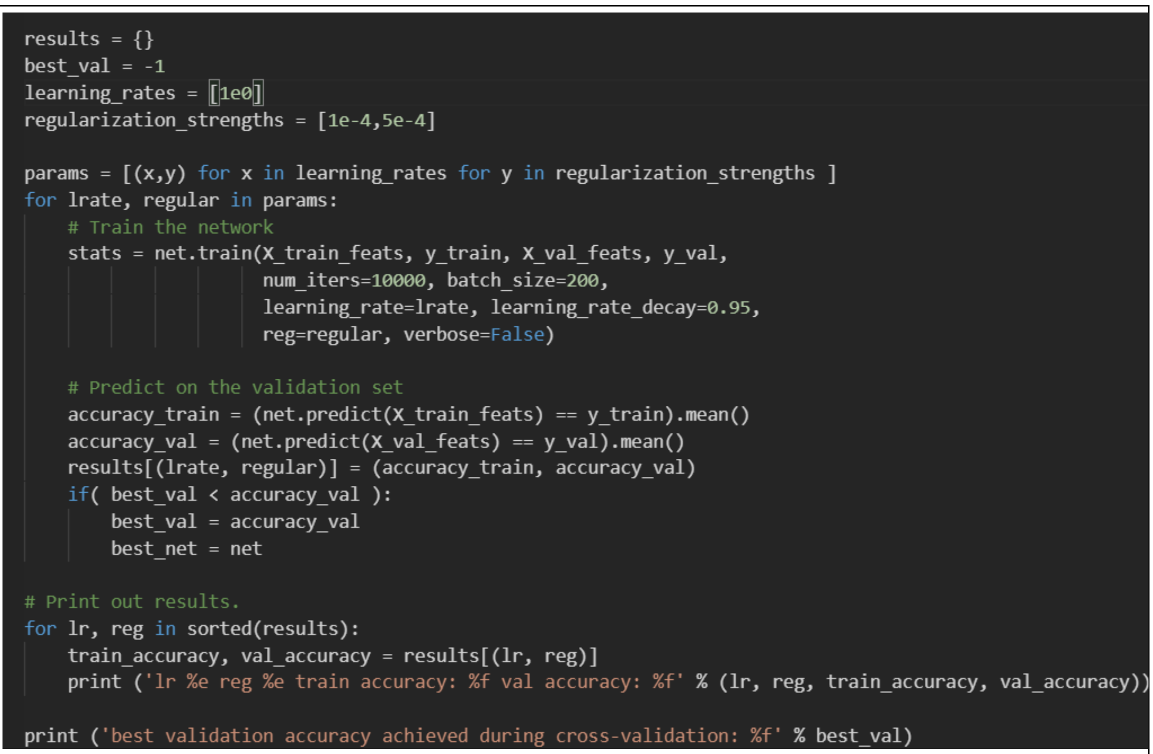
```
results = {}
best_val = -1
learning_rates = [1e-4,1e-3,5e-3]
regularization_strengths = [1e-4,5e-4,1e-3,4e-3]

params = [(x,y) for x in learning_rates for y in regularization_strengths ]
for lrate, regular in params:
    net = ThreeLayerNet(input_size, hidden_size, num_classes)
    # Train the network
    stats = net.train(X_train, y_train, X_val, y_val,
                      num_iters=10000, batch_size=150,
                      learning_rate=lrate, learning_rate_decay=0.95,
                      reg=regular, verbose=False)

    # Predict on the validation set
    accuracy_train = (net.predict(X_train) == y_train).mean()
    accuracy_val = (net.predict(X_val) == y_val).mean()
    results[(lrate, regular)] = (accuracy_train, accuracy_val)
    if( best_val < accuracy_val ):
        best_val = accuracy_val
        best_net = net

# Print out results.
for lr, reg in sorted(results):
    train_accuracy, val_accuracy = results[(lr, reg)]
    print ('lr %e reg %e train accuracy: %f val accuracy: %f' % (lr, reg, train_accuracy, val_accuracy))

print ('best validation accuracy achieved during cross-validation: %f' % best_val)
```

交叉验证期间实现的最佳验证准确度



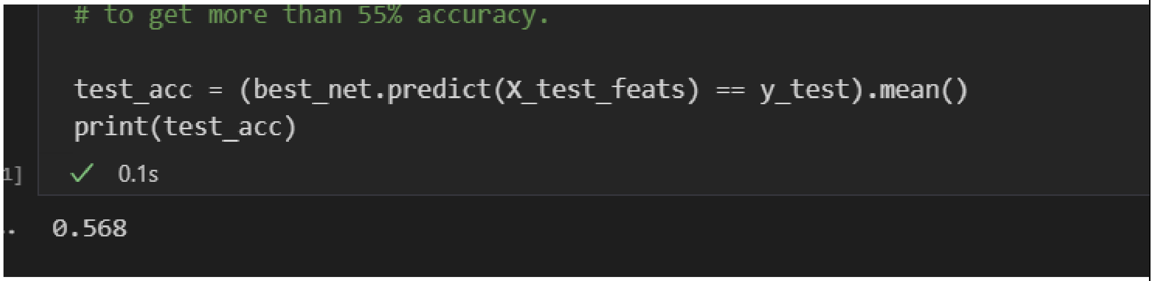
```
lr 5.000000e-03 reg 4.000000e-03 train accuracy: 0.575510 val accuracy: 0.506000
best validation accuracy achieved during cross-validation: 0.517000
```

4. 测试 Three-Layer Neural Network



```python
results = {}
best_val = -1
learning_rates = [1e0]
regularization_strengths = [1e-4,5e-4]

params = [(x,y) for x in learning_rates for y in regularization_strengths ]
for lrate, regular in params:
    # Train the network
    stats = net.train(X_train_feats, y_train, X_val_feats, y_val,
                      num_iters=10000, batch_size=200,
                      learning_rate=lrate, learning_rate_decay=0.95,
                      reg=regular, verbose=False)

    # Predict on the validation set
    accuracy_train = (net.predict(X_train_feats) == y_train).mean()
    accuracy_val = (net.predict(X_val_feats) == y_val).mean()
    results[(lrate, regular)] = (accuracy_train, accuracy_val)
    if( best_val < accuracy_val ):
        best_val = accuracy_val
        best_net = net

# Print out results.
for lr, reg in sorted(results):
    train_accuracy, val_accuracy = results[(lr, reg)]
    print ('lr %e reg %e train accuracy: %f val accuracy: %f' % (lr, reg, train_accuracy, val_accuracy))

print ('best validation accuracy achieved during cross-validation: %f' % best_val)
```

在测试集上评估经过训练的 ,得出准确率为 0.568



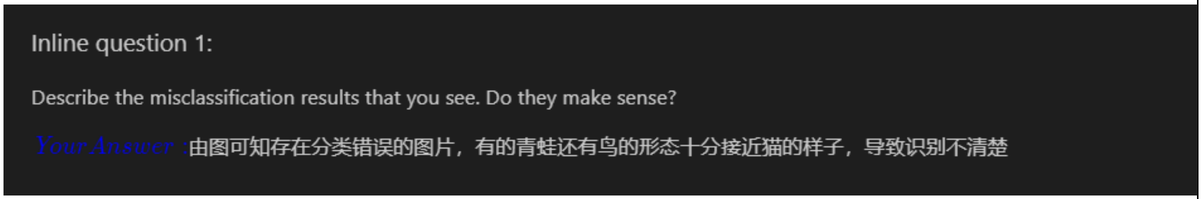
```python
# to get more than 55% accuracy.

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

```
✓ 0.1s
```

```
0.568
```

5.问题回答

问题 1:



Inline question 1:

Describe the misclassification results that you see. Do they make sense?

Your Answer :由图可知存在分类错误的图片，有的青蛙还有鸟的形态十分接近猫的样子，导致识别不清楚

结论分析与体会：

1. 了解了训练/验证/测试分割以及超参数验证数据的使用

2.了解了基本的图像分类管道和数据驱动的方法（训练/预测阶段）

3.熟练使用 numpy 编写高效的矢量代码

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.不清楚如何设计三层神经网络如何编写？

网上的教程大多数是关于两层神经网络的算法,所以需要自己根据其推导如何设计实现三层网络，但最终模型实现的准确率特别低，后期打算再修改一下调参

2.在三层神经网络中，由于参数系数问题，出现了矩阵维数不能够对等的情况，仔细推导参 数，再对系数进行修改，就成功了