```
[info] ROOT=/home/kim1484/assignment1_colab/assignment1
[info] DATA_DIR=/home/kim1484/assignment1_colab/assignment1/cs231n/datasets
[ready] 환경 준비 완료.
```

Softmax Classifier exercise

Complete and hand in this completed worksheet (including its outputs and any supporting code outside of the worksheet) with your assignment submission. For more details see the assignments page on the course website.

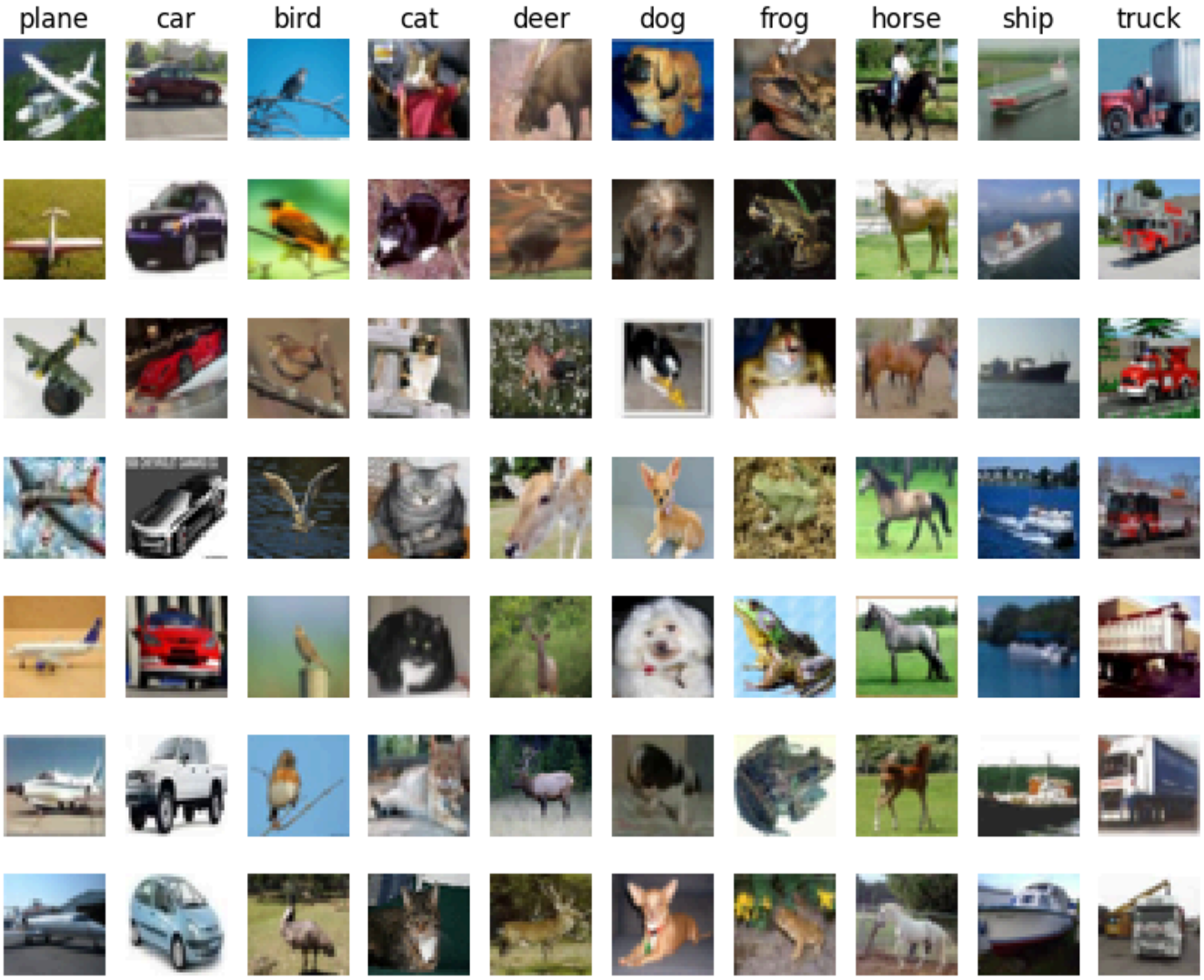
In this exercise you will:

- implement a fully-vectorized **loss function** for the Softmax classifier.
- implement the fully-vectorized expression for its **analytic gradient**
- **check your implementation** using numerical gradient
- use a validation set to **tune the learning rate and regularization** strength
- **optimize** the loss function with **SGD**
- **visualize** the final learned weights

```
[ready] Visualization & autoreload settings complete.
```

CIFAR-10 Data Loading and Preprocessing

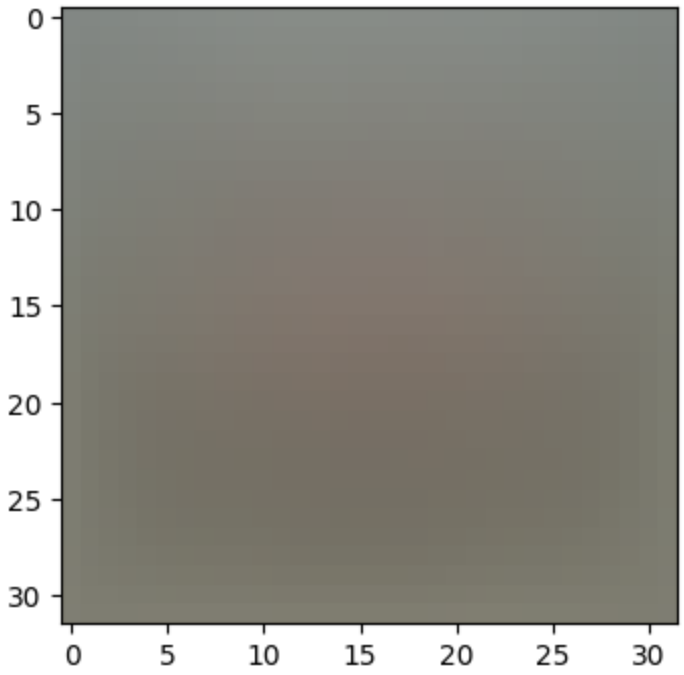
```
Training data shape:  (50000, 32, 32, 3)
Training labels shape:  (50000,)
Test data shape:  (10000, 32, 32, 3)
Test labels shape:  (10000,)
```




```
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,)

Training data shape:  (49000, 3072)
Validation data shape:  (1000, 3072)
Test data shape:  (1000, 3072)
dev data shape:  (500, 3072)

[130.64189796 135.98173469 132.47391837 130.05569388 135.34804082
 131.75402041 130.96055102 136.14328571 132.47636735 131.48467347]
```




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

Softmax Classifier

Your code for this section will all be written inside `cs231n/classifiers/softmax.py`.

As you can see, we have prefilled the function `softmax_loss_naive` which uses for loops to evaluate the softmax loss function.

```
loss: 2.388963
loss: 2.388963
sanity check: 2.302585
```

Inline Question 1

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

$Your Answer:$ *Since there are 10 classes, a random prediction would give each class a probability of 1/10, which is 0.1.*

The `grad` returned from the function above is right now all zero. Derive and implement the gradient for the softmax loss function and implement it inline inside the function `softmax_loss_naive`. You will find it helpful to interleave your new code inside the existing function.

To check that you have correctly implemented the gradient, you can numerically estimate the gradient of the loss function and compare the numeric estimate to the gradient that you computed. We have provided code that does this for you:

```
numerical: -1.700435 analytic: -1.700435, relative error: 3.644047e-09
numerical: -0.352173 analytic: -0.352173, relative error: 1.929128e-07
numerical: 2.418968 analytic: 2.418967, relative error: 3.997566e-08
numerical: -1.057496 analytic: -1.057496, relative error: 1.467064e-08
numerical: 1.863354 analytic: 1.863354, relative error: 2.638738e-08
numerical: 0.537221 analytic: 0.537221, relative error: 1.318414e-08
numerical: 4.258377 analytic: 4.258377, relative error: 1.221471e-11
numerical: -0.677913 analytic: -0.677913, relative error: 4.155236e-08
numerical: 0.412116 analytic: 0.412116, relative error: 2.328918e-07
numerical: -1.805972 analytic: -1.805972, relative error: 9.318435e-09
numerical: -1.394039 analytic: -1.394039, relative error: 3.926214e-08
numerical: -1.899599 analytic: -1.899599, relative error: 1.148263e-08
numerical: -0.196209 analytic: -0.196209, relative error: 2.470200e-07
numerical: -0.199058 analytic: -0.199058, relative error: 4.392419e-08
numerical: -1.208800 analytic: -1.208800, relative error: 1.844511e-08
numerical: -3.001003 analytic: -3.001003, relative error: 3.431242e-08
numerical: -1.380397 analytic: -1.380397, relative error: 7.616553e-09
numerical: 4.914987 analytic: 4.914987, relative error: 1.968208e-08
numerical: -2.546576 analytic: -2.546576, relative error: 5.239118e-10
numerical: 0.445246 analytic: 0.445245, relative error: 1.272662e-07
```

Inline Question 2

Although gradcheck is reliable softmax loss, it is possible that for SVM loss, once in a while, a dimension in the gradcheck will not match exactly. What could such a discrepancy be caused by? Is it a reason for concern? What is a simple example in one dimension where a svm loss gradient check could fail? How would change the margin affect of the frequency of this happening?

Note that SVM loss for a sample (x_i, y_i) is defined as:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

where j iterates over all classes except the correct class y_i and s_j denotes the classifier score for j^{th} class. Δ is a scalar margin. For more information, refer to 'Multiclass Support Vector Machine loss' on this page.

Hint: the SVM loss function is not strictly speaking differentiable.

$Your Answer$: *SVM loss have a nondifferentiable points. When* $s_j - s_{y_i} + margin = 0$ *the gradient is not uniquely defined. Because of this, numerical gradient checking can occasionally fail. During numerical differentiation using* $\frac{f(x+h)-f(x-h)}{2h}$ *the two evaluations may fall on different sides of the hinge:*

- $(x + h)$ *: analytic gradient = 1*
- $(x - h)$ *: analytic gradient = 0*

Then the numerical gradient will be $\frac{1-0}{2} = 0.5$*, which differs from the analytic gradient.*

So, a simple 1D example is $L(z) = max(0, z)$ where gradient checking will fail near (z = 0) as written above.

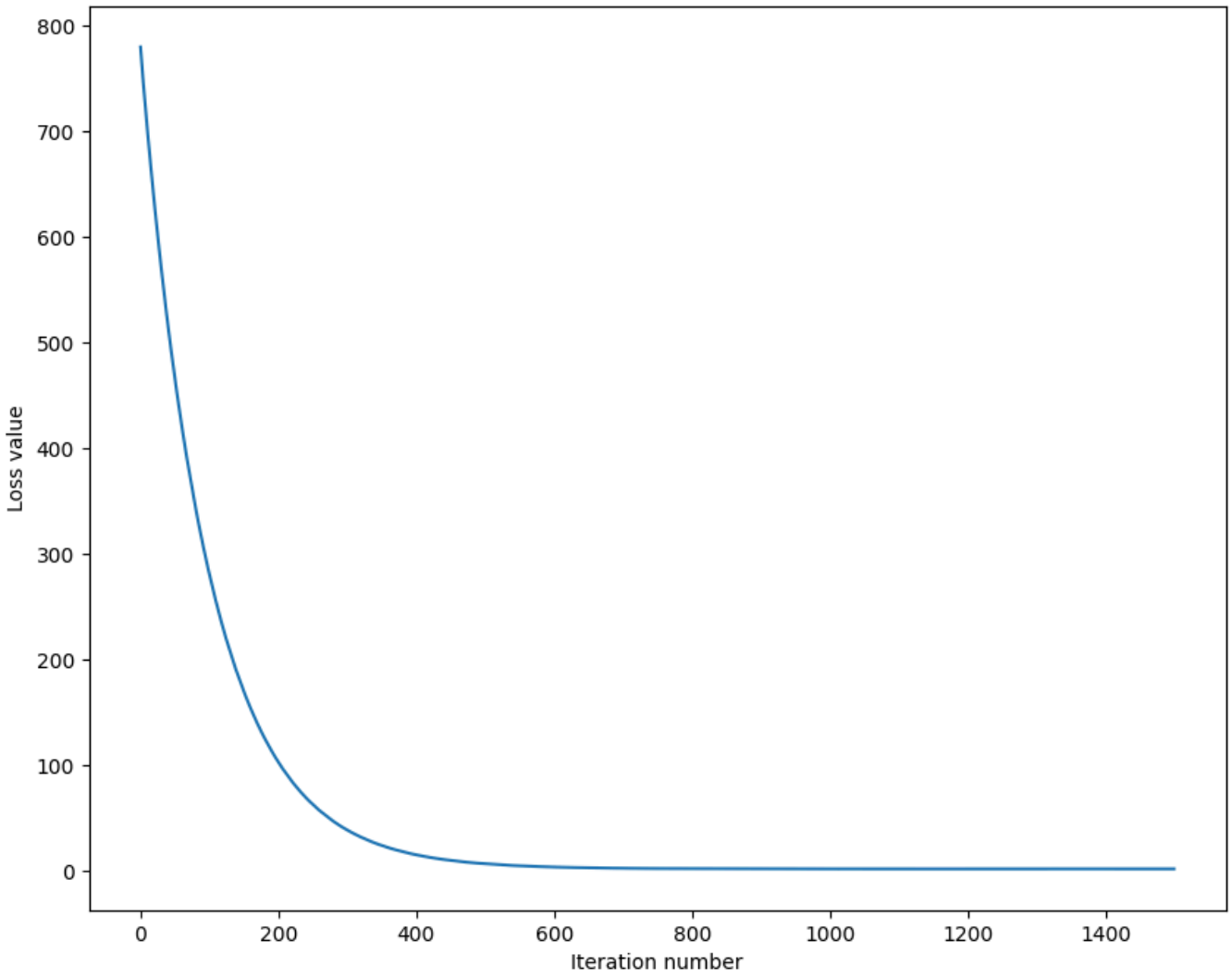
```
Naive loss: 2.388963e+00 computed in 0.042176s
Vectorized loss: 2.388963e+00 computed in 0.001615s
difference: -0.000000
```

```
Naive loss and gradient: computed in 0.027760s
Vectorized loss and gradient: computed in 0.001544s
difference: 1487.845189
```

Stochastic Gradient Descent

We now have vectorized and efficient expressions for the loss, the gradient and our gradient matches the numerical gradient. We are therefore ready to do SGD to minimize the loss. Your code for this part will be written inside `cs231n/classifiers/linear_classifier.py`.

```
iteration 0 / 1500: loss 779.342178
iteration 100 / 1500: loss 280.282180
iteration 200 / 1500: loss 103.210536
iteration 300 / 1500: loss 38.990252
iteration 400 / 1500: loss 15.643946
iteration 500 / 1500: loss 7.179823
iteration 600 / 1500: loss 4.084288
iteration 700 / 1500: loss 2.968384
iteration 800 / 1500: loss 2.581588
iteration 900 / 1500: loss 2.458608
iteration 1000 / 1500: loss 2.398108
iteration 1100 / 1500: loss 2.385488
iteration 1200 / 1500: loss 2.366672
iteration 1300 / 1500: loss 2.382821
iteration 1400 / 1500: loss 2.389853
That took 2.411880s
```

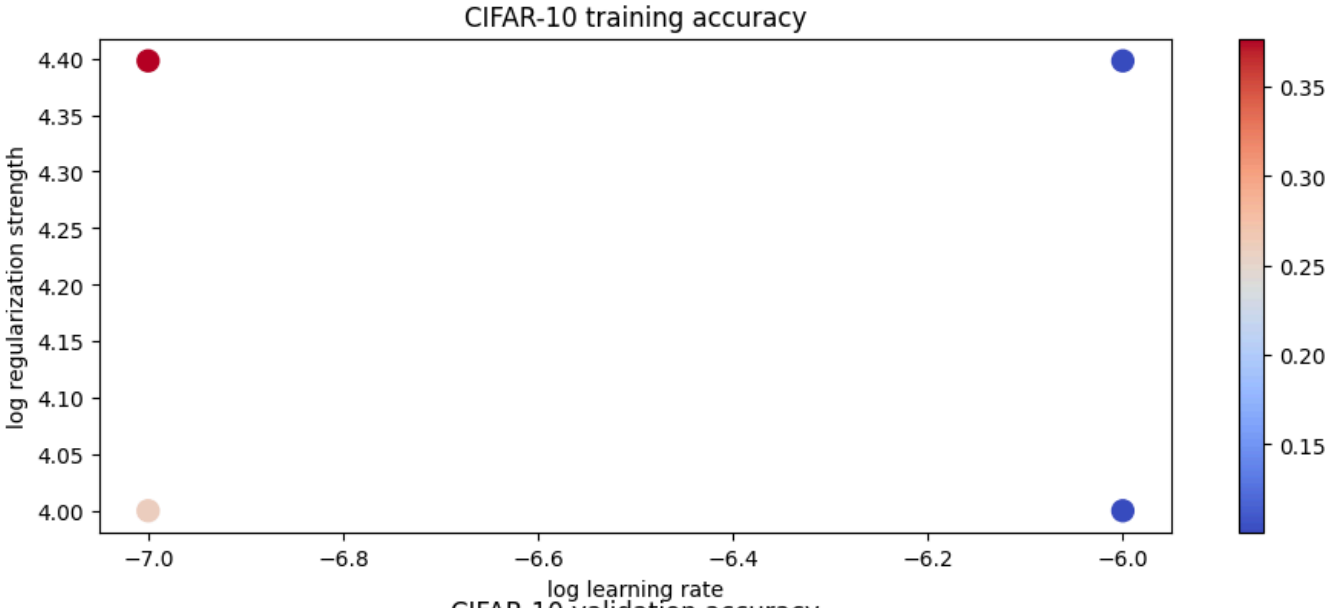


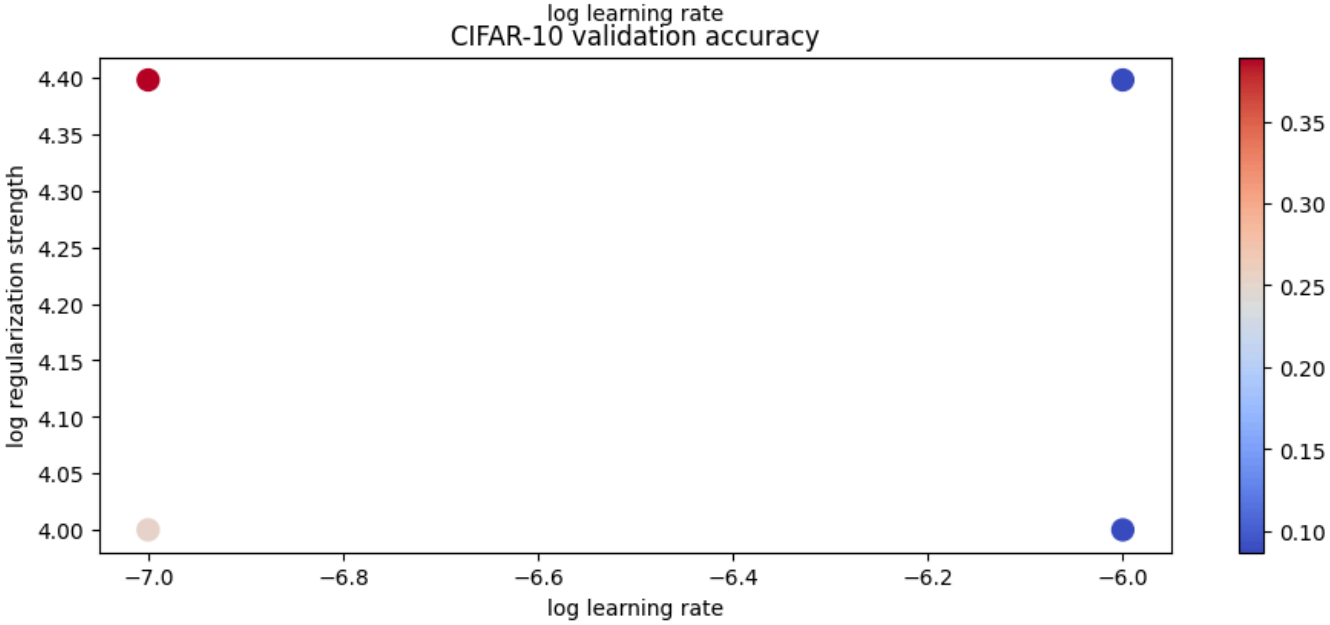

```
training accuracy: 0.380878
validation accuracy: 0.393000

softmax.npy saved.
```

```
/home/kim1484/assignment1_colab/assignment1/cs231n/classifiers/softmax.py:85: Runtime
Warning: divide by zero encountered in log
  loss -= np.sum(np.log(result))
/home/kim1484/assignment1_colab/assignment1/cs231n/classifiers/softmax.py:86: Runtime
Warning: overflow encountered in scalar multiply
  loss = loss / num_train + reg * np.sum(W * W)
/home/kim1484/anaconda3/envs/cs231n/lib/python3.8/site-packages/numpy/core/fromnumeri
c.py:86: RuntimeWarning: overflow encountered in reduce
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)
/home/kim1484/assignment1_colab/assignment1/cs231n/classifiers/softmax.py:86: Runtime
Warning: overflow encountered in multiply
  loss = loss / num_train + reg * np.sum(W * W)
/home/kim1484/assignment1_colab/assignment1/cs231n/classifiers/softmax.py:79: Runtime
Warning: invalid value encountered in subtract
  scores -= np.reshape(np.max(scores, axis=1), (-1, 1))  # (500, 10)
```

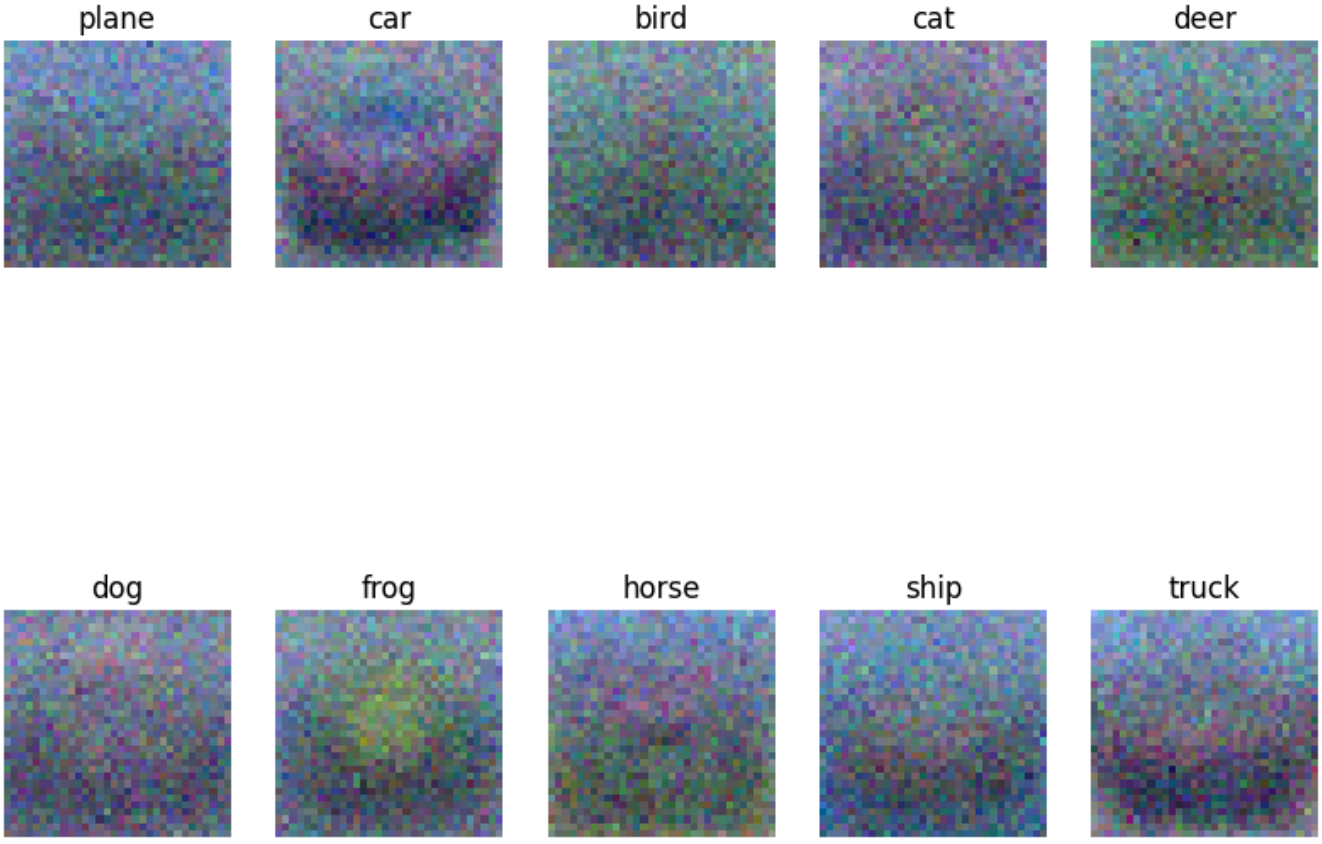
```
lr 1.000000e-07 reg 1.000000e+04 train accuracy: 0.260592 val accuracy: 0.253000
lr 1.000000e-07 reg 2.500000e+04 train accuracy: 0.376918 val accuracy: 0.389000
lr 1.000000e-06 reg 1.000000e+04 train accuracy: 0.100265 val accuracy: 0.087000
lr 1.000000e-06 reg 2.500000e+04 train accuracy: 0.100265 val accuracy: 0.087000
best validation accuracy achieved during cross-validation: 0.389000
```






```
Softmax classifier on raw pixels final test set accuracy: 0.374000

best_softmax.npy saved.
```



Inline question 3

Describe what your visualized Softmax classifier weights look like, and offer a brief explanation for why they look the way they do.

$Your Answer$: *The visualized Softmax weights appear as noisy and blurry color patterns rather than distinct shapes. This is because the softmax classifier is a linear model, it cannot learn complex spatial patterns or object structures. As a result, it only reflects the average color distribution or coarse shape tendency of each class (e.g. the car class)*

Inline Question 4 - *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 change the softmax loss, but leave the SVM loss unchanged.

$Your Answer$: Yes

$Your Explanation$: For the SVM loss, each term has the form $max(0, s_j - s_{y_i} + margin)$. If $(s_j - s_{y_i} + margin)$ is below 0 for all incorrect classes, the hinge loss for that datapoint becomes 0, so adding this datapoint does not change the total SVM loss.