Image features 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 (http://vision.stanford.edu/teaching/cs231n/assignments.html)](http://vision.stanford.edu/teaching/cs231n/assignments.html) on the course website.

We have seen that we can achieve reasonable performance on an image classification task by training a linear classifier on the pixels of the input image. In this exercise we will show that we can improve our classification performance by training linear classifiers not on raw pixels but on features that are computed from the raw pixels.

All of your work for this exercise will be done in this notebook.

In [ ]:

```
import random
import numpy as np
from cs231n.data_utils import load_CIFAR10
import matplotlib.pyplot as plt


%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# for auto-reloading extenrnal modules
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipytho
n
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

Load data

Similar to previous exercises, we will load CIFAR-10 data from disk.

In [ ]:

```
from cs231n.features import color_histogram_hsv, hog_feature

def get_CIFAR10_data(num_training=49000, num_validation=1000, num_test=1000):
    # Load the raw CIFAR-10 data
    cifar10_dir = 'cs231n/datasets/cifar-10-batches-py'

    # Cleaning up variables to prevent loading data multiple times (which may ca
use memory issue)
    try:
       del X_train, y_train
       del X_test, y_test
       print('Clear previously loaded data.')
    except:
       pass

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

    # Subsample the data
    mask = list(range(num_training, num_training + num_validation))
    X_val = X_train[mask]
    y_val = y_train[mask]
    mask = list(range(num_training))
    X_train = X_train[mask]
    y_train = y_train[mask]
    mask = list(range(num_test))
    X_test = X_test[mask]
    y_test = y_test[mask]

    return X_train, y_train, X_val, y_val, X_test, y_test

X_train, y_train, X_val, y_val, X_test, y_test = get_CIFAR10_data()
```

Extract Features

For each image we will compute a Histogram of Oriented Gradients (HOG) as well as a color histogram using the hue channel in HSV color space. We form our final feature vector for each image by concatenating the HOG and color histogram feature vectors.

Roughly speaking, HOG should capture the texture of the image while ignoring color information, and the color histogram represents the color of the input image while ignoring texture. As a result, we expect that using both together ought to work better than using either alone. Verifying this assumption would be a good thing to try for your own interest.

The `hog_feature` and `color_histogram_hsv` functions both operate on a single image and return a feature vector for that image. The extract_features function takes a set of images and a list of feature functions and evaluates each feature function on each image, storing the results in a matrix where each column is the concatenation of all feature vectors for a single image.

In [ ]:

```
from cs231n.features import *

num_color_bins = 10 # Number of bins in the color histogram
feature_fns = [hog_feature, lambda img: color_histogram_hsv(img, nbin=num_color_
bins)]
X_train_feats = extract_features(X_train, feature_fns, verbose=True)
X_val_feats = extract_features(X_val, feature_fns)
X_test_feats = extract_features(X_test, feature_fns)

# Preprocessing: Subtract the mean feature
mean_feat = np.mean(X_train_feats, axis=0, keepdims=True)
X_train_feats -= mean_feat
X_val_feats -= mean_feat
X_test_feats -= mean_feat

# Preprocessing: Divide by standard deviation. This ensures that each feature
# has roughly the same scale.
std_feat = np.std(X_train_feats, axis=0, keepdims=True)
X_train_feats /= std_feat
X_val_feats /= std_feat
X_test_feats /= std_feat

# Preprocessing: Add a bias dimension
X_train_feats = np.hstack([X_train_feats, np.ones((X_train_feats.shape[0], 1))])
X_val_feats = np.hstack([X_val_feats, np.ones((X_val_feats.shape[0], 1))])
X_test_feats = np.hstack([X_test_feats, np.ones((X_test_feats.shape[0], 1))])
```

```
Done extracting features for 1000 / 49000 images
Done extracting features for 2000 / 49000 images
Done extracting features for 3000 / 49000 images
Done extracting features for 4000 / 49000 images
Done extracting features for 5000 / 49000 images
Done extracting features for 6000 / 49000 images
Done extracting features for 7000 / 49000 images
Done extracting features for 8000 / 49000 images
Done extracting features for 9000 / 49000 images
Done extracting features for 10000 / 49000 images
Done extracting features for 11000 / 49000 images
Done extracting features for 12000 / 49000 images
Done extracting features for 13000 / 49000 images
Done extracting features for 14000 / 49000 images
Done extracting features for 15000 / 49000 images
Done extracting features for 16000 / 49000 images
Done extracting features for 17000 / 49000 images
Done extracting features for 18000 / 49000 images
Done extracting features for 19000 / 49000 images
Done extracting features for 20000 / 49000 images
Done extracting features for 21000 / 49000 images
Done extracting features for 22000 / 49000 images
Done extracting features for 23000 / 49000 images
Done extracting features for 24000 / 49000 images
Done extracting features for 25000 / 49000 images
Done extracting features for 26000 / 49000 images
Done extracting features for 27000 / 49000 images
Done extracting features for 28000 / 49000 images
Done extracting features for 29000 / 49000 images
Done extracting features for 30000 / 49000 images
Done extracting features for 31000 / 49000 images
Done extracting features for 32000 / 49000 images
Done extracting features for 33000 / 49000 images
Done extracting features for 34000 / 49000 images
Done extracting features for 35000 / 49000 images
Done extracting features for 36000 / 49000 images
Done extracting features for 37000 / 49000 images
Done extracting features for 38000 / 49000 images
Done extracting features for 39000 / 49000 images
Done extracting features for 40000 / 49000 images
Done extracting features for 41000 / 49000 images
Done extracting features for 42000 / 49000 images
Done extracting features for 43000 / 49000 images
Done extracting features for 44000 / 49000 images
Done extracting features for 45000 / 49000 images
Done extracting features for 46000 / 49000 images
Done extracting features for 47000 / 49000 images
Done extracting features for 48000 / 49000 images
Done extracting features for 49000 / 49000 images
```

Train SVM on features

Using the multiclass SVM code developed earlier in the assignment, train SVMs on top of the features extracted above; this should achieve better results than training SVMs directly on top of raw pixels.

In [ ]:

```
# Use the validation set to tune the learning rate and regularization strength

from cs231n.classifiers.linear_classifier import LinearSVM

learning_rates = [1e-9, 1e-8, 1e-7]
regularization_strengths = [5e4, 5e5, 5e6]

results = {}
best_val = -1
best_svm = None

################################################################################
# TODO:                                                                        #
# Use the validation set to set the learning rate and regularization strength. #
# This should be identical to the validation that you did for the SVM; save    #
# the best trained classifer in best_svm. You might also want to play          #
# with different numbers of bins in the color histogram. If you are careful    #
# you should be able to get accuracy of near 0.44 on the validation set.       #
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

for reg in regularization_strengths:
    for lr in learning_rates:
        svm = LinearSVM()
        # Train the svm
        loss_hist = svm.train(X_train_feats, y_train, lr, reg, num_iters=5000)

        # Predict on the validation set
        train_acc = (svm.predict(X_train_feats) == y_train).mean()
        val_acc = (svm.predict(X_val_feats) == y_val).mean()

        if (val_acc > best_val):
            best_svm = svm
            best_val = val_acc

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

# 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
)
```

```
best validation accuracy achieved during cross-validation: 0.419000
```

In [ ]:

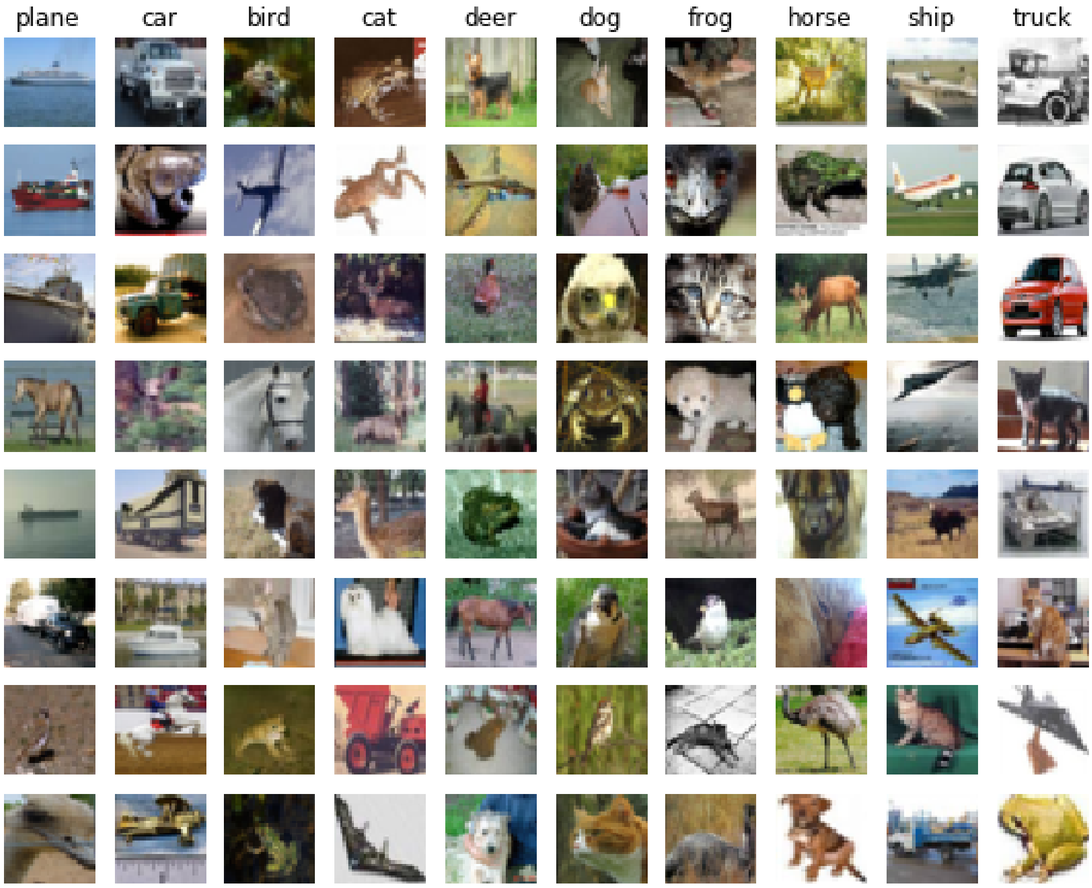
```
# Evaluate your trained SVM on the test set: you should be able to get at least
 0.40
y_test_pred = best_svm.predict(X_test_feats)
test_accuracy = np.mean(y_test == y_test_pred)
print(test_accuracy)
```

```
0.419
```

In [ ]:

```
# An important way to gain intuition about how an algorithm works is to
# visualize the mistakes that it makes. In this visualization, we show examples
# of images that are misclassified by our current system. The first column
# shows images that our system labeled as "plane" but whose true label is
# something other than "plane".

examples_per_class = 8
classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship'
, 'truck']
for cls, cls_name in enumerate(classes):
    idxs = np.where((y_test != cls) & (y_test_pred == cls))[0]
    idxs = np.random.choice(idxs, examples_per_class, replace=False)
    for i, idx in enumerate(idxs):
        plt.subplot(examples_per_class, len(classes), i * len(classes) + cls + 1
)
        plt.imshow(X_test[idx].astype('uint8'))
        plt.axis('off')
        if i == 0:
            plt.title(cls_name)
plt.show()
```



Inline question 1:

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

$\color{blue}{\textit{Your Answer :}}$

Take first column for example, the pictures are mostly realtively bluer, which suits the environment of the plane. There is a horse which is misclassfied into the plane class, and the background of the horse image is blue, which may be the reasom. Since the weights of svm is the 'average' of the training dataset, svm would always misclassified the picture with similar shape or environment.

Neural Network on image features

Earlier in this assigment we saw that training a two-layer neural network on raw pixels achieved better classification performance than linear classifiers on raw pixels. In this notebook we have seen that linear classifiers on image features outperform linear classifiers on raw pixels.

For completeness, we should also try training a neural network on image features. This approach should outperform all previous approaches: you should easily be able to achieve over 55% classification accuracy on the test set; our best model achieves about 60% classification accuracy.

In [ ]:

```
# Preprocessing: Remove the bias dimension
# Make sure to run this cell only ONCE
print(X_train_feats.shape)
X_train_feats = X_train_feats[:, :-1]
X_val_feats = X_val_feats[:, :-1]
X_test_feats = X_test_feats[:, :-1]

print(X_train_feats.shape)
```

In [ ]:

```
from cs231n.classifiers.neural_net import TwoLayerNet

input_dim = X_train_feats.shape[1]
hidden_dim = 500
num_classes = 10

net = TwoLayerNet(input_dim, hidden_dim, num_classes)
best_net = None
best_accuracy = 0.0

################################################################################
# TODO: Train a two-layer neural network on image features. You may want to    #
# cross-validate various parameters as in previous sections. Store your best   #
# model in the best_net variable.                                              #
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

regs = [0.001, 0.01, 0.5]
lrs = [0.09, 0.1, 0.2, 0.3, 0.5]
learning_rate_decays = [0.99, 0.95, 0.75, 0.5]
batch_sizes = [100, 200, 400, 600]

for reg in regs:
    for lr in lrs:
        for lr_decay in learning_rate_decays:
            for batch_size in batch_sizes:
                net = TwoLayerNet(input_dim, hidden_dim, num_classes)

                # Train the network
                stats = net.train(X_train_feats, y_train, X_val_feats, y_val,
                            num_iters=1000, batch_size=batch_size,
                            learning_rate=lr, learning_rate_decay=lr_decay,
                            reg=reg, verbose=False)

                # Predict on the validation set
                train_acc = (net.predict(X_train_feats) == y_train).mean()
                val_acc = (net.predict(X_val_feats) == y_val).mean()

                print("batch_size:", batch_size, "reg:", reg, "lr:", lr, "lr_dec
ay:", lr_decay, "val_acc:", val_acc, "train_acc:", train_acc)
                if (val_acc > best_accuracy):
                    best_net = net
                    best_accuracy = val_acc

print("best accuracy:", best_accuracy)

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

```
batch_size: 100 reg: 0.001 lr: 0.09 lr_decay: 0.99 val_acc: 0.525 tr
ain_acc: 0.5195102040816326
batch_size: 200 reg: 0.001 lr: 0.09 lr_decay: 0.99 val_acc: 0.519 tr
ain_acc: 0.5238367346938776
batch_size: 400 reg: 0.001 lr: 0.09 lr_decay: 0.99 val_acc: 0.512 tr
ain_acc: 0.5245102040816326
batch_size: 600 reg: 0.001 lr: 0.09 lr_decay: 0.99 val_acc: 0.512 tr
ain_acc: 0.5238979591836734
batch_size: 100 reg: 0.001 lr: 0.09 lr_decay: 0.95 val_acc: 0.507 tr
ain_acc: 0.5156938775510204
batch_size: 200 reg: 0.001 lr: 0.09 lr_decay: 0.95 val_acc: 0.525 tr
ain_acc: 0.5164081632653061
batch_size: 400 reg: 0.001 lr: 0.09 lr_decay: 0.95 val_acc: 0.509 tr
ain_acc: 0.5167551020408163
batch_size: 600 reg: 0.001 lr: 0.09 lr_decay: 0.95 val_acc: 0.512 tr
ain_acc: 0.5221020408163265
batch_size: 100 reg: 0.001 lr: 0.09 lr_decay: 0.75 val_acc: 0.491 tr
ain_acc: 0.48828571428571427
batch_size: 200 reg: 0.001 lr: 0.09 lr_decay: 0.75 val_acc: 0.447 tr
ain_acc: 0.4588979591836735
batch_size: 400 reg: 0.001 lr: 0.09 lr_decay: 0.75 val_acc: 0.448 tr
ain_acc: 0.4599591836734694
batch_size: 600 reg: 0.001 lr: 0.09 lr_decay: 0.75 val_acc: 0.492 tr
ain_acc: 0.502265306122449
batch_size: 100 reg: 0.001 lr: 0.09 lr_decay: 0.5 val_acc: 0.392 tra
in_acc: 0.39155102040816325
batch_size: 200 reg: 0.001 lr: 0.09 lr_decay: 0.5 val_acc: 0.259 tra
in_acc: 0.2433061224489796
batch_size: 400 reg: 0.001 lr: 0.09 lr_decay: 0.5 val_acc: 0.267 tra
in_acc: 0.25479591836734694
batch_size: 600 reg: 0.001 lr: 0.09 lr_decay: 0.5 val_acc: 0.446 tra
in_acc: 0.45991836734693875
batch_size: 100 reg: 0.001 lr: 0.1 lr_decay: 0.99 val_acc: 0.504 tra
in_acc: 0.519734693877551
batch_size: 200 reg: 0.001 lr: 0.1 lr_decay: 0.99 val_acc: 0.517 tra
in_acc: 0.525734693877551
batch_size: 400 reg: 0.001 lr: 0.1 lr_decay: 0.99 val_acc: 0.507 tra
in_acc: 0.5293877551020408
batch_size: 600 reg: 0.001 lr: 0.1 lr_decay: 0.99 val_acc: 0.521 tra
in_acc: 0.5299387755102041
batch_size: 100 reg: 0.001 lr: 0.1 lr_decay: 0.95 val_acc: 0.504 tra
in_acc: 0.5166326530612245
batch_size: 200 reg: 0.001 lr: 0.1 lr_decay: 0.95 val_acc: 0.515 tra
in_acc: 0.5240408163265307
batch_size: 400 reg: 0.001 lr: 0.1 lr_decay: 0.95 val_acc: 0.52 trai
n_acc: 0.5240816326530612
batch_size: 600 reg: 0.001 lr: 0.1 lr_decay: 0.95 val_acc: 0.519 tra
in_acc: 0.5289387755102041
batch_size: 100 reg: 0.001 lr: 0.1 lr_decay: 0.75 val_acc: 0.497 tra
in_acc: 0.49883673469387757
batch_size: 200 reg: 0.001 lr: 0.1 lr_decay: 0.75 val_acc: 0.477 tra
in_acc: 0.4741020408163265
batch_size: 400 reg: 0.001 lr: 0.1 lr_decay: 0.75 val_acc: 0.467 tra
in_acc: 0.4744897959183674
batch_size: 600 reg: 0.001 lr: 0.1 lr_decay: 0.75 val_acc: 0.511 tra
in_acc: 0.5134285714285715
batch_size: 100 reg: 0.001 lr: 0.1 lr_decay: 0.5 val_acc: 0.404 trai
n_acc: 0.41475510204081634
batch_size: 200 reg: 0.001 lr: 0.1 lr_decay: 0.5 val_acc: 0.276 trai
n_acc: 0.2664489795918367
batch_size: 400 reg: 0.001 lr: 0.1 lr_decay: 0.5 val_acc: 0.287 trai
```

```
n_acc: 0.2693877551020408
batch_size: 600 reg: 0.001 lr: 0.1 lr_decay: 0.5 val_acc: 0.466 trai
n_acc: 0.4717959183673469
batch_size: 100 reg: 0.001 lr: 0.2 lr_decay: 0.99 val_acc: 0.529 tra
in_acc: 0.5546326530612244
batch_size: 200 reg: 0.001 lr: 0.2 lr_decay: 0.99 val_acc: 0.53 trai
n_acc: 0.5605918367346939
batch_size: 400 reg: 0.001 lr: 0.2 lr_decay: 0.99 val_acc: 0.544 tra
in_acc: 0.5627551020408164
batch_size: 600 reg: 0.001 lr: 0.2 lr_decay: 0.99 val_acc: 0.555 tra
in_acc: 0.5688367346938775
batch_size: 100 reg: 0.001 lr: 0.2 lr_decay: 0.95 val_acc: 0.532 tra
in_acc: 0.5545510204081633
batch_size: 200 reg: 0.001 lr: 0.2 lr_decay: 0.95 val_acc: 0.52 trai
n_acc: 0.5549795918367347
batch_size: 400 reg: 0.001 lr: 0.2 lr_decay: 0.95 val_acc: 0.537 tra
in_acc: 0.5592040816326531
batch_size: 600 reg: 0.001 lr: 0.2 lr_decay: 0.95 val_acc: 0.553 tra
in_acc: 0.5668163265306122
batch_size: 100 reg: 0.001 lr: 0.2 lr_decay: 0.75 val_acc: 0.516 tra
in_acc: 0.5316938775510204
batch_size: 200 reg: 0.001 lr: 0.2 lr_decay: 0.75 val_acc: 0.519 tra
in_acc: 0.5289387755102041
batch_size: 400 reg: 0.001 lr: 0.2 lr_decay: 0.75 val_acc: 0.512 tra
in_acc: 0.5314285714285715
batch_size: 600 reg: 0.001 lr: 0.2 lr_decay: 0.75 val_acc: 0.536 tra
in_acc: 0.549
batch_size: 100 reg: 0.001 lr: 0.2 lr_decay: 0.5 val_acc: 0.499 trai
n_acc: 0.5114897959183673
batch_size: 200 reg: 0.001 lr: 0.2 lr_decay: 0.5 val_acc: 0.451 trai
n_acc: 0.4625714285714286
batch_size: 400 reg: 0.001 lr: 0.2 lr_decay: 0.5 val_acc: 0.451 trai
n_acc: 0.4632857142857143
batch_size: 600 reg: 0.001 lr: 0.2 lr_decay: 0.5 val_acc: 0.508 trai
n_acc: 0.530795918367347
batch_size: 100 reg: 0.001 lr: 0.3 lr_decay: 0.99 val_acc: 0.533 tra
in_acc: 0.5755714285714286
batch_size: 200 reg: 0.001 lr: 0.3 lr_decay: 0.99 val_acc: 0.55 trai
n_acc: 0.5879795918367346
batch_size: 400 reg: 0.001 lr: 0.3 lr_decay: 0.99 val_acc: 0.59 trai
n_acc: 0.5990816326530612
batch_size: 600 reg: 0.001 lr: 0.3 lr_decay: 0.99 val_acc: 0.576 tra
in_acc: 0.6054693877551021
batch_size: 100 reg: 0.001 lr: 0.3 lr_decay: 0.95 val_acc: 0.544 tra
in_acc: 0.5675102040816327
batch_size: 200 reg: 0.001 lr: 0.3 lr_decay: 0.95 val_acc: 0.544 tra
in_acc: 0.5807551020408164
batch_size: 400 reg: 0.001 lr: 0.3 lr_decay: 0.95 val_acc: 0.558 tra
in_acc: 0.5902448979591837
batch_size: 600 reg: 0.001 lr: 0.3 lr_decay: 0.95 val_acc: 0.576 tra
in_acc: 0.599
batch_size: 100 reg: 0.001 lr: 0.3 lr_decay: 0.75 val_acc: 0.519 tra
in_acc: 0.5542448979591836
batch_size: 200 reg: 0.001 lr: 0.3 lr_decay: 0.75 val_acc: 0.532 tra
in_acc: 0.5475714285714286
batch_size: 400 reg: 0.001 lr: 0.3 lr_decay: 0.75 val_acc: 0.53 trai
n_acc: 0.5513877551020409
batch_size: 600 reg: 0.001 lr: 0.3 lr_decay: 0.75 val_acc: 0.554 tra
in_acc: 0.5770612244897959
batch_size: 100 reg: 0.001 lr: 0.3 lr_decay: 0.5 val_acc: 0.525 trai
n_acc: 0.5272653061224489
```

```
batch_size: 200 reg: 0.001 lr: 0.3 lr_decay: 0.5 val_acc: 0.502 trai
n_acc: 0.5075510204081632
batch_size: 400 reg: 0.001 lr: 0.3 lr_decay: 0.5 val_acc: 0.492 trai
n_acc: 0.5054081632653061
batch_size: 600 reg: 0.001 lr: 0.3 lr_decay: 0.5 val_acc: 0.521 trai
n_acc: 0.5471020408163265
batch_size: 100 reg: 0.001 lr: 0.5 lr_decay: 0.99 val_acc: 0.555 tra
in_acc: 0.5933877551020408
batch_size: 200 reg: 0.001 lr: 0.5 lr_decay: 0.99 val_acc: 0.569 tra
in_acc: 0.6205102040816326
batch_size: 400 reg: 0.001 lr: 0.5 lr_decay: 0.99 val_acc: 0.592 tra
in_acc: 0.6394489795918368
batch_size: 600 reg: 0.001 lr: 0.5 lr_decay: 0.99 val_acc: 0.594 tra
in_acc: 0.6524897959183673
batch_size: 100 reg: 0.001 lr: 0.5 lr_decay: 0.95 val_acc: 0.553 tra
in_acc: 0.5903469387755103
batch_size: 200 reg: 0.001 lr: 0.5 lr_decay: 0.95 val_acc: 0.562 tra
in_acc: 0.6181224489795918
batch_size: 400 reg: 0.001 lr: 0.5 lr_decay: 0.95 val_acc: 0.568 tra
in_acc: 0.632265306122449
batch_size: 600 reg: 0.001 lr: 0.5 lr_decay: 0.95 val_acc: 0.591 tra
in_acc: 0.647061224489796
batch_size: 100 reg: 0.001 lr: 0.5 lr_decay: 0.75 val_acc: 0.561 tra
in_acc: 0.5851632653061225
batch_size: 200 reg: 0.001 lr: 0.5 lr_decay: 0.75 val_acc: 0.571 tra
in_acc: 0.5844897959183674
batch_size: 400 reg: 0.001 lr: 0.5 lr_decay: 0.75 val_acc: 0.567 tra
in_acc: 0.5901020408163266
batch_size: 600 reg: 0.001 lr: 0.5 lr_decay: 0.75 val_acc: 0.573 tra
in_acc: 0.6231836734693877
batch_size: 100 reg: 0.001 lr: 0.5 lr_decay: 0.5 val_acc: 0.524 trai
n_acc: 0.5554489795918367
batch_size: 200 reg: 0.001 lr: 0.5 lr_decay: 0.5 val_acc: 0.522 trai
n_acc: 0.5371020408163265
batch_size: 400 reg: 0.001 lr: 0.5 lr_decay: 0.5 val_acc: 0.513 trai
n_acc: 0.5368979591836734
batch_size: 600 reg: 0.001 lr: 0.5 lr_decay: 0.5 val_acc: 0.56 train
_acc: 0.5891836734693877
batch_size: 100 reg: 0.01 lr: 0.09 lr_decay: 0.99 val_acc: 0.501 tra
in_acc: 0.5057551020408163
batch_size: 200 reg: 0.01 lr: 0.09 lr_decay: 0.99 val_acc: 0.49 trai
n_acc: 0.5018571428571429
batch_size: 400 reg: 0.01 lr: 0.09 lr_decay: 0.99 val_acc: 0.496 tra
in_acc: 0.5029591836734694
batch_size: 600 reg: 0.01 lr: 0.09 lr_decay: 0.99 val_acc: 0.502 tra
in_acc: 0.505061224489796
batch_size: 100 reg: 0.01 lr: 0.09 lr_decay: 0.95 val_acc: 0.491 tra
in_acc: 0.4903469387755102
batch_size: 200 reg: 0.01 lr: 0.09 lr_decay: 0.95 val_acc: 0.482 tra
in_acc: 0.49489795918367346
batch_size: 400 reg: 0.01 lr: 0.09 lr_decay: 0.95 val_acc: 0.481 tra
in_acc: 0.49742857142857144
batch_size: 600 reg: 0.01 lr: 0.09 lr_decay: 0.95 val_acc: 0.492 tra
in_acc: 0.504265306122449
batch_size: 100 reg: 0.01 lr: 0.09 lr_decay: 0.75 val_acc: 0.462 tra
in_acc: 0.4767959183673469
batch_size: 200 reg: 0.01 lr: 0.09 lr_decay: 0.75 val_acc: 0.426 tra
in_acc: 0.438469387755102
batch_size: 400 reg: 0.01 lr: 0.09 lr_decay: 0.75 val_acc: 0.423 tra
in_acc: 0.4403877551020408
batch_size: 600 reg: 0.01 lr: 0.09 lr_decay: 0.75 val_acc: 0.482 tra
```

```
in_acc: 0.48412244897959184
batch_size: 100 reg: 0.01 lr: 0.09 lr_decay: 0.5 val_acc: 0.367 trai
n_acc: 0.36253061224489797
batch_size: 200 reg: 0.01 lr: 0.09 lr_decay: 0.5 val_acc: 0.246 trai
n_acc: 0.23087755102040816
batch_size: 400 reg: 0.01 lr: 0.09 lr_decay: 0.5 val_acc: 0.243 trai
n_acc: 0.2296326530612245
batch_size: 600 reg: 0.01 lr: 0.09 lr_decay: 0.5 val_acc: 0.421 trai
n_acc: 0.4376938775510204
batch_size: 100 reg: 0.01 lr: 0.1 lr_decay: 0.99 val_acc: 0.517 trai
n_acc: 0.505734693877551
batch_size: 200 reg: 0.01 lr: 0.1 lr_decay: 0.99 val_acc: 0.504 trai
n_acc: 0.5118979591836734
batch_size: 400 reg: 0.01 lr: 0.1 lr_decay: 0.99 val_acc: 0.497 trai
n_acc: 0.5108163265306123
batch_size: 600 reg: 0.01 lr: 0.1 lr_decay: 0.99 val_acc: 0.5 train_
acc: 0.5133877551020408
batch_size: 100 reg: 0.01 lr: 0.1 lr_decay: 0.95 val_acc: 0.507 trai
n_acc: 0.502061224489796
batch_size: 200 reg: 0.01 lr: 0.1 lr_decay: 0.95 val_acc: 0.494 trai
n_acc: 0.5013265306122449
batch_size: 400 reg: 0.01 lr: 0.1 lr_decay: 0.95 val_acc: 0.502 trai
n_acc: 0.5061224489795918
batch_size: 600 reg: 0.01 lr: 0.1 lr_decay: 0.95 val_acc: 0.51 train
_acc: 0.5123877551020408
batch_size: 100 reg: 0.01 lr: 0.1 lr_decay: 0.75 val_acc: 0.473 trai
n_acc: 0.4850408163265306
batch_size: 200 reg: 0.01 lr: 0.1 lr_decay: 0.75 val_acc: 0.439 trai
n_acc: 0.45671428571428574
batch_size: 400 reg: 0.01 lr: 0.1 lr_decay: 0.75 val_acc: 0.442 trai
n_acc: 0.4571224489795918
batch_size: 600 reg: 0.01 lr: 0.1 lr_decay: 0.75 val_acc: 0.484 trai
n_acc: 0.49448979591836734
batch_size: 100 reg: 0.01 lr: 0.1 lr_decay: 0.5 val_acc: 0.388 train
_acc: 0.39691836734693875
batch_size: 200 reg: 0.01 lr: 0.1 lr_decay: 0.5 val_acc: 0.276 train
_acc: 0.2589183673469388
batch_size: 400 reg: 0.01 lr: 0.1 lr_decay: 0.5 val_acc: 0.277 train
_acc: 0.26285714285714284
batch_size: 600 reg: 0.01 lr: 0.1 lr_decay: 0.5 val_acc: 0.445 train
_acc: 0.4545102040816327
batch_size: 100 reg: 0.01 lr: 0.2 lr_decay: 0.99 val_acc: 0.513 trai
n_acc: 0.5083877551020408
batch_size: 200 reg: 0.01 lr: 0.2 lr_decay: 0.99 val_acc: 0.516 trai
n_acc: 0.5149591836734694
batch_size: 400 reg: 0.01 lr: 0.2 lr_decay: 0.99 val_acc: 0.5 train_
acc: 0.5189795918367347
batch_size: 600 reg: 0.01 lr: 0.2 lr_decay: 0.99 val_acc: 0.515 trai
n_acc: 0.5216326530612245
batch_size: 100 reg: 0.01 lr: 0.2 lr_decay: 0.95 val_acc: 0.504 trai
n_acc: 0.5108979591836734
batch_size: 200 reg: 0.01 lr: 0.2 lr_decay: 0.95 val_acc: 0.496 trai
n_acc: 0.5175102040816326
batch_size: 400 reg: 0.01 lr: 0.2 lr_decay: 0.95 val_acc: 0.512 trai
n_acc: 0.5235510204081633
batch_size: 600 reg: 0.01 lr: 0.2 lr_decay: 0.95 val_acc: 0.514 trai
n_acc: 0.5246938775510204
batch_size: 100 reg: 0.01 lr: 0.2 lr_decay: 0.75 val_acc: 0.52 train
_acc: 0.5177755102040816
batch_size: 200 reg: 0.01 lr: 0.2 lr_decay: 0.75 val_acc: 0.502 trai
n_acc: 0.515061224489796
```

```
batch_size: 400 reg: 0.01 lr: 0.2 lr_decay: 0.75 val_acc: 0.51 train
_acc: 0.5159387755102041
batch_size: 600 reg: 0.01 lr: 0.2 lr_decay: 0.75 val_acc: 0.504 trai
n_acc: 0.5228979591836734
batch_size: 100 reg: 0.01 lr: 0.2 lr_decay: 0.5 val_acc: 0.487 train
_acc: 0.4919387755102041
batch_size: 200 reg: 0.01 lr: 0.2 lr_decay: 0.5 val_acc: 0.427 train
_acc: 0.4417142857142857
batch_size: 400 reg: 0.01 lr: 0.2 lr_decay: 0.5 val_acc: 0.429 train
_acc: 0.43942857142857145
batch_size: 600 reg: 0.01 lr: 0.2 lr_decay: 0.5 val_acc: 0.5 train_a
cc: 0.5111632653061224
batch_size: 100 reg: 0.01 lr: 0.3 lr_decay: 0.99 val_acc: 0.499 trai
n_acc: 0.5078775510204082
batch_size: 200 reg: 0.01 lr: 0.3 lr_decay: 0.99 val_acc: 0.51 train
_acc: 0.5130408163265306
batch_size: 400 reg: 0.01 lr: 0.3 lr_decay: 0.99 val_acc: 0.507 trai
n_acc: 0.5219183673469387
batch_size: 600 reg: 0.01 lr: 0.3 lr_decay: 0.99 val_acc: 0.521 trai
n_acc: 0.5266530612244898
batch_size: 100 reg: 0.01 lr: 0.3 lr_decay: 0.95 val_acc: 0.503 trai
n_acc: 0.5020816326530613
batch_size: 200 reg: 0.01 lr: 0.3 lr_decay: 0.95 val_acc: 0.514 trai
n_acc: 0.5186122448979592
batch_size: 400 reg: 0.01 lr: 0.3 lr_decay: 0.95 val_acc: 0.508 trai
n_acc: 0.5218979591836734
batch_size: 600 reg: 0.01 lr: 0.3 lr_decay: 0.95 val_acc: 0.521 trai
n_acc: 0.5245510204081633
batch_size: 100 reg: 0.01 lr: 0.3 lr_decay: 0.75 val_acc: 0.501 trai
n_acc: 0.5119387755102041
batch_size: 200 reg: 0.01 lr: 0.3 lr_decay: 0.75 val_acc: 0.506 trai
n_acc: 0.5219795918367347
batch_size: 400 reg: 0.01 lr: 0.3 lr_decay: 0.75 val_acc: 0.508 trai
n_acc: 0.5234285714285715
batch_size: 600 reg: 0.01 lr: 0.3 lr_decay: 0.75 val_acc: 0.512 trai
n_acc: 0.521673469387755
batch_size: 100 reg: 0.01 lr: 0.3 lr_decay: 0.5 val_acc: 0.498 train
_acc: 0.5145306122448979
batch_size: 200 reg: 0.01 lr: 0.3 lr_decay: 0.5 val_acc: 0.483 train
_acc: 0.4913265306122449
batch_size: 400 reg: 0.01 lr: 0.3 lr_decay: 0.5 val_acc: 0.494 train
_acc: 0.4887755102040816
batch_size: 600 reg: 0.01 lr: 0.3 lr_decay: 0.5 val_acc: 0.506 train
_acc: 0.5208775510204081
batch_size: 100 reg: 0.01 lr: 0.5 lr_decay: 0.99 val_acc: 0.484 trai
n_acc: 0.4850816326530612
batch_size: 200 reg: 0.01 lr: 0.5 lr_decay: 0.99 val_acc: 0.5 train_
acc: 0.5131224489795918
batch_size: 400 reg: 0.01 lr: 0.5 lr_decay: 0.99 val_acc: 0.518 trai
n_acc: 0.5212448979591837
batch_size: 600 reg: 0.01 lr: 0.5 lr_decay: 0.99 val_acc: 0.509 trai
n_acc: 0.528795918367347
batch_size: 100 reg: 0.01 lr: 0.5 lr_decay: 0.95 val_acc: 0.467 trai
n_acc: 0.4887959183673469
batch_size: 200 reg: 0.01 lr: 0.5 lr_decay: 0.95 val_acc: 0.49 train
_acc: 0.5015510204081632
batch_size: 400 reg: 0.01 lr: 0.5 lr_decay: 0.95 val_acc: 0.504 trai
n_acc: 0.5165102040816326
batch_size: 600 reg: 0.01 lr: 0.5 lr_decay: 0.95 val_acc: 0.518 trai
n_acc: 0.5275510204081633
batch_size: 100 reg: 0.01 lr: 0.5 lr_decay: 0.75 val_acc: 0.495 trai
```

```
n_acc: 0.5086122448979592
batch_size: 200 reg: 0.01 lr: 0.5 lr_decay: 0.75 val_acc: 0.494 trai
n_acc: 0.5245714285714286
batch_size: 400 reg: 0.01 lr: 0.5 lr_decay: 0.75 val_acc: 0.522 trai
n_acc: 0.5271632653061225
batch_size: 600 reg: 0.01 lr: 0.5 lr_decay: 0.75 val_acc: 0.517 trai
n_acc: 0.5281020408163265
batch_size: 100 reg: 0.01 lr: 0.5 lr_decay: 0.5 val_acc: 0.5 train_a
cc: 0.5186326530612245
batch_size: 200 reg: 0.01 lr: 0.5 lr_decay: 0.5 val_acc: 0.516 train
_acc: 0.5191632653061224
batch_size: 400 reg: 0.01 lr: 0.5 lr_decay: 0.5 val_acc: 0.513 train
_acc: 0.5204081632653061
batch_size: 600 reg: 0.01 lr: 0.5 lr_decay: 0.5 val_acc: 0.537 train
_acc: 0.5238367346938776
batch_size: 100 reg: 0.5 lr: 0.09 lr_decay: 0.99 val_acc: 0.087 trai
n_acc: 0.10026530612244898
batch_size: 200 reg: 0.5 lr: 0.09 lr_decay: 0.99 val_acc: 0.087 trai
n_acc: 0.10026530612244898
batch_size: 400 reg: 0.5 lr: 0.09 lr_decay: 0.99 val_acc: 0.119 trai
n_acc: 0.09961224489795918
batch_size: 600 reg: 0.5 lr: 0.09 lr_decay: 0.99 val_acc: 0.079 trai
n_acc: 0.10042857142857142
batch_size: 100 reg: 0.5 lr: 0.09 lr_decay: 0.95 val_acc: 0.113 trai
n_acc: 0.09973469387755102
batch_size: 200 reg: 0.5 lr: 0.09 lr_decay: 0.95 val_acc: 0.105 trai
n_acc: 0.09989795918367347
batch_size: 400 reg: 0.5 lr: 0.09 lr_decay: 0.95 val_acc: 0.119 trai
n_acc: 0.09961224489795918
batch_size: 600 reg: 0.5 lr: 0.09 lr_decay: 0.95 val_acc: 0.078 trai
n_acc: 0.10044897959183674
batch_size: 100 reg: 0.5 lr: 0.09 lr_decay: 0.75 val_acc: 0.079 trai
n_acc: 0.10042857142857142
batch_size: 200 reg: 0.5 lr: 0.09 lr_decay: 0.75 val_acc: 0.078 trai
n_acc: 0.10044897959183674
batch_size: 400 reg: 0.5 lr: 0.09 lr_decay: 0.75 val_acc: 0.098 trai
n_acc: 0.10004081632653061
batch_size: 600 reg: 0.5 lr: 0.09 lr_decay: 0.75 val_acc: 0.098 trai
n_acc: 0.10004081632653061
batch_size: 100 reg: 0.5 lr: 0.09 lr_decay: 0.5 val_acc: 0.102 train
_acc: 0.09995918367346938
batch_size: 200 reg: 0.5 lr: 0.09 lr_decay: 0.5 val_acc: 0.078 train
_acc: 0.10044897959183674
batch_size: 400 reg: 0.5 lr: 0.09 lr_decay: 0.5 val_acc: 0.078 train
_acc: 0.10044897959183674
batch_size: 600 reg: 0.5 lr: 0.09 lr_decay: 0.5 val_acc: 0.087 train
_acc: 0.10026530612244898
batch_size: 100 reg: 0.5 lr: 0.1 lr_decay: 0.99 val_acc: 0.087 train
_acc: 0.10026530612244898
batch_size: 200 reg: 0.5 lr: 0.1 lr_decay: 0.99 val_acc: 0.102 train
_acc: 0.09995918367346938
batch_size: 400 reg: 0.5 lr: 0.1 lr_decay: 0.99 val_acc: 0.078 train
_acc: 0.10044897959183674
batch_size: 600 reg: 0.5 lr: 0.1 lr_decay: 0.99 val_acc: 0.078 train
_acc: 0.10044897959183674
batch_size: 100 reg: 0.5 lr: 0.1 lr_decay: 0.95 val_acc: 0.113 train
_acc: 0.09973469387755102
batch_size: 200 reg: 0.5 lr: 0.1 lr_decay: 0.95 val_acc: 0.102 train
_acc: 0.09995918367346938
batch_size: 400 reg: 0.5 lr: 0.1 lr_decay: 0.95 val_acc: 0.119 train
_acc: 0.09961224489795918
```

```
batch_size: 600 reg: 0.5 lr: 0.1 lr_decay: 0.95 val_acc: 0.079 train
_acc: 0.10042857142857142
batch_size: 100 reg: 0.5 lr: 0.1 lr_decay: 0.75 val_acc: 0.112 train
_acc: 0.09975510204081632
batch_size: 200 reg: 0.5 lr: 0.1 lr_decay: 0.75 val_acc: 0.078 train
_acc: 0.10044897959183674
batch_size: 400 reg: 0.5 lr: 0.1 lr_decay: 0.75 val_acc: 0.078 train
_acc: 0.10044897959183674
batch_size: 600 reg: 0.5 lr: 0.1 lr_decay: 0.75 val_acc: 0.098 train
_acc: 0.10004081632653061
batch_size: 100 reg: 0.5 lr: 0.1 lr_decay: 0.5 val_acc: 0.078 train_
acc: 0.10044897959183674
batch_size: 200 reg: 0.5 lr: 0.1 lr_decay: 0.5 val_acc: 0.078 train_
acc: 0.10044897959183674
batch_size: 400 reg: 0.5 lr: 0.1 lr_decay: 0.5 val_acc: 0.078 train_
acc: 0.10044897959183674
batch_size: 600 reg: 0.5 lr: 0.1 lr_decay: 0.5 val_acc: 0.087 train_
acc: 0.10026530612244898
batch_size: 100 reg: 0.5 lr: 0.2 lr_decay: 0.99 val_acc: 0.113 train
_acc: 0.09973469387755102
batch_size: 200 reg: 0.5 lr: 0.2 lr_decay: 0.99 val_acc: 0.107 train
_acc: 0.09985714285714285
batch_size: 400 reg: 0.5 lr: 0.2 lr_decay: 0.99 val_acc: 0.105 train
_acc: 0.09989795918367347
batch_size: 600 reg: 0.5 lr: 0.2 lr_decay: 0.99 val_acc: 0.098 train
_acc: 0.10004081632653061
batch_size: 100 reg: 0.5 lr: 0.2 lr_decay: 0.95 val_acc: 0.112 train
_acc: 0.09975510204081632
batch_size: 200 reg: 0.5 lr: 0.2 lr_decay: 0.95 val_acc: 0.087 train
_acc: 0.10026530612244898
batch_size: 400 reg: 0.5 lr: 0.2 lr_decay: 0.95 val_acc: 0.078 train
_acc: 0.10044897959183674
batch_size: 600 reg: 0.5 lr: 0.2 lr_decay: 0.95 val_acc: 0.078 train
_acc: 0.10044897959183674
batch_size: 100 reg: 0.5 lr: 0.2 lr_decay: 0.75 val_acc: 0.119 train
_acc: 0.09961224489795918
batch_size: 200 reg: 0.5 lr: 0.2 lr_decay: 0.75 val_acc: 0.079 train
_acc: 0.10042857142857142
batch_size: 400 reg: 0.5 lr: 0.2 lr_decay: 0.75 val_acc: 0.087 train
_acc: 0.10026530612244898
batch_size: 600 reg: 0.5 lr: 0.2 lr_decay: 0.75 val_acc: 0.119 train
_acc: 0.09961224489795918
batch_size: 100 reg: 0.5 lr: 0.2 lr_decay: 0.5 val_acc: 0.098 train_
acc: 0.10004081632653061
batch_size: 200 reg: 0.5 lr: 0.2 lr_decay: 0.5 val_acc: 0.087 train_
acc: 0.10026530612244898
batch_size: 400 reg: 0.5 lr: 0.2 lr_decay: 0.5 val_acc: 0.107 train_
acc: 0.09985714285714285
batch_size: 600 reg: 0.5 lr: 0.2 lr_decay: 0.5 val_acc: 0.102 train_
acc: 0.09995918367346938
batch_size: 100 reg: 0.5 lr: 0.3 lr_decay: 0.99 val_acc: 0.098 train
_acc: 0.10004081632653061
batch_size: 200 reg: 0.5 lr: 0.3 lr_decay: 0.99 val_acc: 0.119 train
_acc: 0.09961224489795918
batch_size: 400 reg: 0.5 lr: 0.3 lr_decay: 0.99 val_acc: 0.112 train
_acc: 0.09975510204081632
batch_size: 600 reg: 0.5 lr: 0.3 lr_decay: 0.99 val_acc: 0.107 train
_acc: 0.09985714285714285
batch_size: 100 reg: 0.5 lr: 0.3 lr_decay: 0.95 val_acc: 0.079 train
_acc: 0.10042857142857142
batch_size: 200 reg: 0.5 lr: 0.3 lr_decay: 0.95 val_acc: 0.102 train
```

```
_acc: 0.09995918367346938
batch_size: 400 reg: 0.5 lr: 0.3 lr_decay: 0.95 val_acc: 0.079 train
_acc: 0.10042857142857142
batch_size: 600 reg: 0.5 lr: 0.3 lr_decay: 0.95 val_acc: 0.078 train
_acc: 0.10044897959183674
batch_size: 100 reg: 0.5 lr: 0.3 lr_decay: 0.75 val_acc: 0.078 train
_acc: 0.10044897959183674
batch_size: 200 reg: 0.5 lr: 0.3 lr_decay: 0.75 val_acc: 0.098 train
_acc: 0.10004081632653061
batch_size: 400 reg: 0.5 lr: 0.3 lr_decay: 0.75 val_acc: 0.079 train
_acc: 0.10042857142857142
batch_size: 600 reg: 0.5 lr: 0.3 lr_decay: 0.75 val_acc: 0.087 train
_acc: 0.10026530612244898
batch_size: 100 reg: 0.5 lr: 0.3 lr_decay: 0.5 val_acc: 0.078 train_
acc: 0.10044897959183674
batch_size: 200 reg: 0.5 lr: 0.3 lr_decay: 0.5 val_acc: 0.087 train_
acc: 0.10026530612244898
batch_size: 400 reg: 0.5 lr: 0.3 lr_decay: 0.5 val_acc: 0.079 train_
acc: 0.10042857142857142
batch_size: 600 reg: 0.5 lr: 0.3 lr_decay: 0.5 val_acc: 0.079 train_
acc: 0.10042857142857142
batch_size: 100 reg: 0.5 lr: 0.5 lr_decay: 0.99 val_acc: 0.078 train
_acc: 0.10044897959183674
batch_size: 200 reg: 0.5 lr: 0.5 lr_decay: 0.99 val_acc: 0.113 train
_acc: 0.09973469387755102
batch_size: 400 reg: 0.5 lr: 0.5 lr_decay: 0.99 val_acc: 0.105 train
_acc: 0.09989795918367347
batch_size: 600 reg: 0.5 lr: 0.5 lr_decay: 0.99 val_acc: 0.087 train
_acc: 0.10026530612244898
batch_size: 100 reg: 0.5 lr: 0.5 lr_decay: 0.95 val_acc: 0.113 train
_acc: 0.09973469387755102
batch_size: 200 reg: 0.5 lr: 0.5 lr_decay: 0.95 val_acc: 0.105 train
_acc: 0.09989795918367347
batch_size: 400 reg: 0.5 lr: 0.5 lr_decay: 0.95 val_acc: 0.107 train
_acc: 0.09985714285714285
batch_size: 600 reg: 0.5 lr: 0.5 lr_decay: 0.95 val_acc: 0.078 train
_acc: 0.10044897959183674
batch_size: 100 reg: 0.5 lr: 0.5 lr_decay: 0.75 val_acc: 0.078 train
_acc: 0.10044897959183674
batch_size: 200 reg: 0.5 lr: 0.5 lr_decay: 0.75 val_acc: 0.113 train
_acc: 0.09973469387755102
batch_size: 400 reg: 0.5 lr: 0.5 lr_decay: 0.75 val_acc: 0.107 train
_acc: 0.09985714285714285
batch_size: 600 reg: 0.5 lr: 0.5 lr_decay: 0.75 val_acc: 0.119 train
_acc: 0.09961224489795918
batch_size: 100 reg: 0.5 lr: 0.5 lr_decay: 0.5 val_acc: 0.087 train_
acc: 0.10026530612244898
batch_size: 200 reg: 0.5 lr: 0.5 lr_decay: 0.5 val_acc: 0.113 train_
acc: 0.09973469387755102
batch_size: 400 reg: 0.5 lr: 0.5 lr_decay: 0.5 val_acc: 0.078 train_
acc: 0.10044897959183674
batch_size: 600 reg: 0.5 lr: 0.5 lr_decay: 0.5 val_acc: 0.102 train_
acc: 0.09995918367346938
best accuracy: 0.594
```

In [ ]:

```
# 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)
```

0.587