features

June 7, 2020

```
In [ ]: from google.colab import drive

        drive.mount('/content/drive', force_remount=True)

        # enter the foldername in your Drive where you have saved the unzipped
        # 'cs231n' folder containing the '.py', 'classifiers' and 'datasets'
        # folders.
        # e.g. 'cs231n/assignments/assignment1/cs231n/'
        FOLDERNAME = None

        assert FOLDERNAME is not None, "[!] Enter the foldername."

        %cd drive/My\ Drive
        %cp -r $FOLDERNAME ../../
        %cd ../../
        %cd cs231n/datasets/
        !bash get_datasets.sh
        %cd ../../
```

1 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 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 [1]: 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-ipython
%load_ext autoreload
%autoreload 2
```

1.1 Load data

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

```
In [2]: 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 cause me
            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()
```

1.2 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 [3]: 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
```

1.3 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 [6]: # Use the validation set to tune the learning rate and regularization strength

        from cs231n.classifiers.linear_classifier import LinearSVM
```

```python
# learning_rates = [1e-9, 1e-8, 1e-7]
learning_rates = [1e-3, 5e-3, 1e-4, 5e-4]
regularization_strengths = [1e-2, 1e-1, 0.2, 0.4, 0.6, 0.8, 1.0]
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)*****

combos = [(lr, r) for lr in learning_rates for r in regularization_strengths]
for lr, r in combos:
    svm = LinearSVM()
    svm.train(X_train_feats, y_train, learning_rate=lr, reg=r, num_iters=2000,
            batch_size=200, verbose=False)
    y_train_pred = svm.predict(X_train_feats)
    train_acc = np.mean( y_train_pred == y_train )
    y_val_pred = svm.predict(X_val_feats)
    val_acc = np.mean( y_val_pred == y_val )
    results[lr,r] = (train_acc,val_acc)
    if val_acc > best_val:
        best_val = val_acc
        best_svm = svm

# *****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)
```

```
lr 1.000000e-04 reg 1.000000e-02 train accuracy: 0.459510 val accuracy: 0.446000
lr 1.000000e-04 reg 1.000000e-01 train accuracy: 0.459755 val accuracy: 0.448000
lr 1.000000e-04 reg 2.000000e-01 train accuracy: 0.458367 val accuracy: 0.445000
lr 1.000000e-04 reg 4.000000e-01 train accuracy: 0.458265 val accuracy: 0.453000
lr 1.000000e-04 reg 6.000000e-01 train accuracy: 0.458163 val accuracy: 0.445000
lr 1.000000e-04 reg 8.000000e-01 train accuracy: 0.458327 val accuracy: 0.453000
lr 1.000000e-04 reg 1.000000e+00 train accuracy: 0.456510 val accuracy: 0.443000
lr 5.000000e-04 reg 1.000000e-02 train accuracy: 0.495449 val accuracy: 0.483000
lr 5.000000e-04 reg 1.000000e-01 train accuracy: 0.494184 val accuracy: 0.481000
lr 5.000000e-04 reg 2.000000e-01 train accuracy: 0.493694 val accuracy: 0.485000
lr 5.000000e-04 reg 4.000000e-01 train accuracy: 0.489367 val accuracy: 0.481000
lr 5.000000e-04 reg 6.000000e-01 train accuracy: 0.487102 val accuracy: 0.476000
lr 5.000000e-04 reg 8.000000e-01 train accuracy: 0.484592 val accuracy: 0.476000
lr 5.000000e-04 reg 1.000000e+00 train accuracy: 0.481857 val accuracy: 0.473000
lr 1.000000e-03 reg 1.000000e-02 train accuracy: 0.504163 val accuracy: 0.492000
lr 1.000000e-03 reg 1.000000e-01 train accuracy: 0.502918 val accuracy: 0.493000
lr 1.000000e-03 reg 2.000000e-01 train accuracy: 0.501776 val accuracy: 0.491000
lr 1.000000e-03 reg 4.000000e-01 train accuracy: 0.495327 val accuracy: 0.483000
lr 1.000000e-03 reg 6.000000e-01 train accuracy: 0.491061 val accuracy: 0.483000
lr 1.000000e-03 reg 8.000000e-01 train accuracy: 0.487878 val accuracy: 0.480000
lr 1.000000e-03 reg 1.000000e+00 train accuracy: 0.482898 val accuracy: 0.474000
lr 5.000000e-03 reg 1.000000e-02 train accuracy: 0.511918 val accuracy: 0.504000
lr 5.000000e-03 reg 1.000000e-01 train accuracy: 0.506122 val accuracy: 0.491000
lr 5.000000e-03 reg 2.000000e-01 train accuracy: 0.498939 val accuracy: 0.487000
lr 5.000000e-03 reg 4.000000e-01 train accuracy: 0.494429 val accuracy: 0.473000
lr 5.000000e-03 reg 6.000000e-01 train accuracy: 0.487776 val accuracy: 0.485000
lr 5.000000e-03 reg 8.000000e-01 train accuracy: 0.483143 val accuracy: 0.479000
lr 5.000000e-03 reg 1.000000e+00 train accuracy: 0.482184 val accuracy: 0.469000
best validation accuracy achieved during cross-validation: 0.504000
```

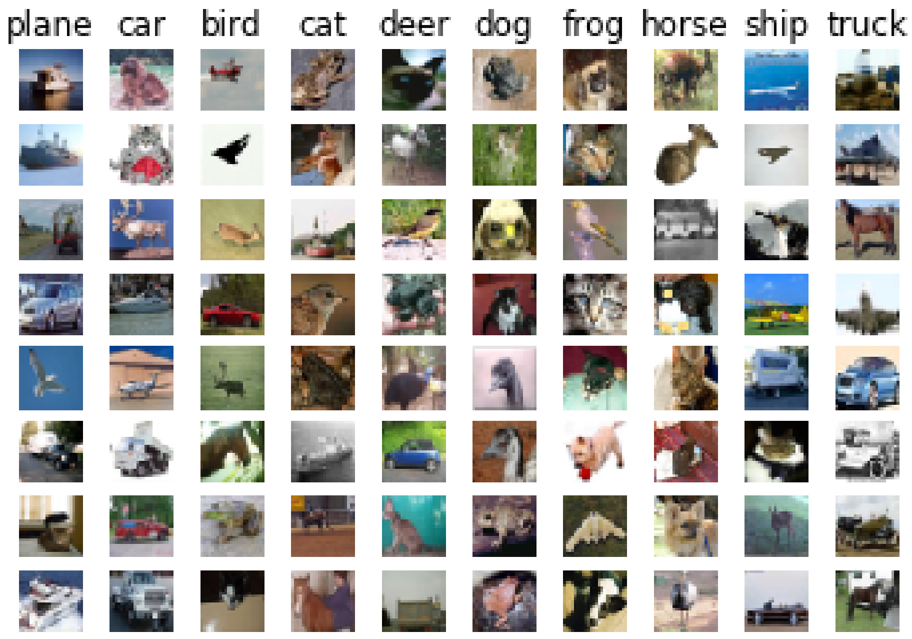
```
In [7]: # Evaluate your trained SVM on the test set: you should be able to get at least 0.40
        y_test_pred = best_svm.predict(X_test_feats)
        test_accuracy = np.mean(y_test == y_test_pred)
        print(test_accuracy)
```

```
0.49
```

```
In [8]: # 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', 'truc
        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()
```



1.3.1 Inline question 1:

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

Your Answer : Misclassification makes sense for dogs and cats here since they are very similar (4 legs small creatures) and hence the model can get confused between them. Similar is the case with car and truck. For planes, they're matched with ships because of the same color (blue) pixels. The model seems to be very confused between images of animals

In conclusion, the combination of HOG and color histogram feature vectors are not simply enough to discriminate correctly all the classes. The HOG descriptor is very useful because it takes into account the edges, but we have to take into account different type of invariances that HOG might not consider, e.g., invariance to translation. Besides that, HOG has different parameters that we can cross-validate for a better performance. Color histogram features can help in some cases (similar colors) but are not very helpful for others. Though, we can still consider this features but maybe with a lower importance when combining it with other features like HOG.

1.4 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 [9]: # 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)
```

```
(49000, 155)
(49000, 154)
```

```
In [12]: from cs231n.classifiers.neural_net import TwoLayerNet

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


         best_net = None

         ################################################################################
         # 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)*****

         # utility function to get many possible hyperparameter combinations using uniform ran
         def hparam_generator(lr_min, lr_max, hsl_min, hsl_max, r_min, r_max):
             lr = 10**np.random.uniform(lr_min,lr_max)
             hsl = np.random.randint(hsl_min, hsl_max)
             r = 10**np.random.uniform(r_min,r_max)
             return lr, hsl, r
         for i in range(30):
             lr, hsl, r = hparam_generator(-1, 0, 100, 500, -8, -12)
             net = TwoLayerNet(input_dim, hsl, num_classes)
             stats = net.train(X_train_feats, y_train, X_val_feats, y_val,
                         num_iters=3000, batch_size=200,
                         learning_rate=lr, learning_rate_decay=0.9,
                         reg=r, verbose=False)
             train_acc = (net.predict(X_train_feats) == y_train).mean()
```

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

            # Save best values
            if val_acc > best_val:
                best_val = val_acc
                best_net = net

            # Print results
            print('lr %e reg %e hid %d  train accuracy: %f val accuracy: %f' % (
                        lr, r, hsl, train_acc, val_acc))
        print('best validation accuracy achieved: %f' % best_val)

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

```
lr 2.016305e-01 reg 3.786713e-11 hid 251  train accuracy: 0.631327 val accuracy: 0.575000
lr 1.627452e-01 reg 3.556322e-11 hid 373  train accuracy: 0.610388 val accuracy: 0.571000
lr 6.617213e-01 reg 2.964423e-10 hid 361  train accuracy: 0.782551 val accuracy: 0.586000
lr 9.944831e-01 reg 2.383131e-11 hid 283  train accuracy: 0.802673 val accuracy: 0.575000
lr 6.617419e-01 reg 3.000281e-12 hid 275  train accuracy: 0.769592 val accuracy: 0.578000
lr 2.013837e-01 reg 1.943637e-12 hid 248  train accuracy: 0.630755 val accuracy: 0.595000
lr 1.342641e-01 reg 1.507899e-09 hid 484  train accuracy: 0.589796 val accuracy: 0.561000
lr 7.394359e-01 reg 2.914478e-11 hid 498  train accuracy: 0.820429 val accuracy: 0.581000
lr 1.636094e-01 reg 8.032615e-09 hid 348  train accuracy: 0.610408 val accuracy: 0.579000
lr 8.906537e-01 reg 8.055330e-09 hid 164  train accuracy: 0.736061 val accuracy: 0.574000
lr 3.169380e-01 reg 2.016409e-10 hid 335  train accuracy: 0.694041 val accuracy: 0.595000
lr 2.553810e-01 reg 2.811382e-09 hid 191  train accuracy: 0.652510 val accuracy: 0.594000
lr 1.291341e-01 reg 4.175871e-11 hid 306  train accuracy: 0.582408 val accuracy: 0.560000
lr 8.708055e-01 reg 7.961073e-09 hid 108  train accuracy: 0.697939 val accuracy: 0.564000
lr 6.841947e-01 reg 3.413648e-10 hid 460  train accuracy: 0.805490 val accuracy: 0.586000
lr 8.931708e-01 reg 4.946611e-12 hid 191  train accuracy: 0.759347 val accuracy: 0.559000
lr 7.935238e-01 reg 6.263380e-09 hid 185  train accuracy: 0.746694 val accuracy: 0.590000
lr 3.196862e-01 reg 4.356551e-11 hid 171  train accuracy: 0.670551 val accuracy: 0.587000
lr 3.904651e-01 reg 2.563719e-12 hid 401  train accuracy: 0.725755 val accuracy: 0.603000
lr 2.752171e-01 reg 6.236971e-09 hid 343  train accuracy: 0.673959 val accuracy: 0.587000
lr 6.905615e-01 reg 1.900107e-12 hid 412  train accuracy: 0.800449 val accuracy: 0.572000
lr 2.135757e-01 reg 8.774284e-10 hid 157  train accuracy: 0.629490 val accuracy: 0.589000
lr 3.077463e-01 reg 1.944752e-09 hid 482  train accuracy: 0.697429 val accuracy: 0.583000
lr 1.536493e-01 reg 2.925708e-11 hid 109  train accuracy: 0.587694 val accuracy: 0.548000
lr 5.821829e-01 reg 3.807344e-10 hid 214  train accuracy: 0.739694 val accuracy: 0.579000
lr 7.448459e-01 reg 1.313516e-11 hid 267  train accuracy: 0.769531 val accuracy: 0.590000
lr 1.153041e-01 reg 8.605468e-11 hid 229  train accuracy: 0.572204 val accuracy: 0.552000
lr 1.710082e-01 reg 2.132935e-10 hid 238  train accuracy: 0.609061 val accuracy: 0.568000
lr 8.259165e-01 reg 9.920046e-11 hid 328  train accuracy: 0.800878 val accuracy: 0.599000
lr 3.718924e-01 reg 3.421542e-09 hid 386  train accuracy: 0.715286 val accuracy: 0.599000
best validation accuracy achieved: 0.603000
```

```
In [14]: # 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)
```

```
0.592
```

---

2 IMPORTANT

This is the end of this question. Please do the following:

1. Click `File -> Save` to make sure the latest checkpoint of this notebook is saved to your Drive.
2. Execute the cell below to download the modified `.py` files back to your drive.