Homework 5 Writeup

Instructions

- Describe any interesting decisions you made to write your algorithm.
- Show and discuss the results of your algorithm.
- Feel free to include code snippets, images, and equations.
- Use as many pages as you need, but err on the short side If you feel you only need to write a short amount to meet the brief, th
- **Please make this document anonymous.**

Algorithm Implementation

Algorithms of this project is implemented in five parts.

0.1 Tiny Image Features

Tiny image creation is simply resizing and normalization of images of the provided paths. Full code is shown below. In line 4, the list of image features is initialized with length same as the number of image paths. Then all the paths are iterated. In line 6, image is read. In line 7, image is reshaped in to a *1*(16*16)* matrix. It is normalized in lines 8 and 9. Then it is added to the array in line 10.

```
1  function image_feats = get_tiny_images(image_paths)
2
3  imsize = 16;
4  image_feats = zeros(size(image_paths,1), imsize*imsize);
5  for i = 1:size(image_paths,1)
6          img = imread(image_paths{i});
7          img = double(reshape(imresize(img, [imsize,
              imsize]), 1, []));
8          img = img - mean(img);
9          img = img./norm(img);
10         image_feats(i,:) = img;
11 end
12
13 end
```

0.2 Nearest Neighbor Classification

The k-NN classification works by letting k nearest neighbors ”vote”. Test images are classified with respect to train images by calculating pairwise distances. This is done in line 4. Then the distances are sorted in ascending order to find the closest points in line 5. The for loop is iterated through all test images. In liness 11 and 12, labels of k closest images are found. In lines 13 to 15, the label with most votes is found using a histogram added to the matrix.

```
1  function predicted_categories = nearest_neighbor_classify(train_image_feats, train_labels, test_image_feats, k)
2
3  N = size(test_image_feats,1);
4  pd = pdist2(train_image_feats, test_image_feats);
5  [~, sortindex] = sort(pd,1,'ascend');
6
7  labels = categorical(unique(train_labels));
8  predicted_categories = cell(N,1);
9
10 for i=1:N
11         top_sort = sortindex(1:k,i);
12         top_labels = categorical(train_labels(top_sort));
13         [label_votes,elected] = histcounts(top_labels, labels);
14         [~,mx_idx] = max(label_votes, [], 2);
15         predicted_categories(i,:) = elected(mx_idx);
16 end
17
18 end
```

0.3 Building Vocabulary

Image vocabulary is built from points of certain cell size in all the images. The grid of points are created in lines 6 and 7. For each image, HOG features are found and added to the features in lines 18 and 19. Such matrix of features is clustered in line 22 to create vocabulary. Each cluster are the words.

```
1  function vocab = build_vocabulary(image_paths, vocab_size, cs, itv)
2
3  N = size(image_paths, 1);
4
5  cell_size = [cs cs];
6  [pv,ph] = meshgrid(cell_size(1)+1:itv:256, cell_size(2)+1:itv:256);
```

```
points = reshape(cat(3, pv,ph),size(pv,1)*size(pv,2),2);

feats = [];

for i = 1:N
        if (mod(i,100)==0)
                disp(i);
        end
        img = im2single(imread(image_paths{i}));
        img = imresize(img, [256 256]);

        [feature_vector,~] = extractHOGFeatures(img, points, 'CellSize',cell_size);
        feats = cat(1,feats,feature_vector);
end
size(feats)
[~,vocab] = kmeans(feats,vocab_size,'MaxIter',20000);
size(vocab)
end
```

0.4 Bag of Words

In this section, points are generated the same way as done in the previous section. Full code shown below. For each image, HOG feature is extracted in line 20. It is then searched using k-NN search from the vocabulary in line 22. The histogram of clusters is calculated and added to the image features array in line 23.

```
 1  function image_feats = get_bags_of_words(image_paths, cs, itv)
 2
 3  load('vocab.mat')
 4  vocab_size = size(vocab, 1);
 5  image_count = size(image_paths, 1);
 6  image_feats = zeros(image_count, vocab_size);
 7
 8  forest = KDTreeSearcher(vocab);
 9  cell_size = [cs cs];
10  [pv,ph] = meshgrid(cell_size(1)+1:itv:256, cell_size(2)+1:itv:256);
11  points = reshape(cat(3, pv,ph),size(pv,1)*size(pv,2),2);
12
13  for i=1:image_count
14          if (mod(i,500)==0)
15                  disp(i);
16          end
```

```
        img = im2single(imread(image_paths{i}));
        img = imresize(img, [256 256]);

        [features, ~] = extractHOGFeatures(img,points, 'CellSize', cell_size);

        [idx , ~] = knnsearch(forest , double(features), 'K', 1);
        image_feats(i,:) = histc(idx, 1:1:vocab_size)';
        end

end
```

0.5 SVM Classification

In this section, a multi-class SVM classification is implemented using multiple binary SVM classifiers. The unique categories are found in line 2. Then for each category, label was found line lines 8 and 9. Where the labels match are assigned 1 and -1 for the rest. The model is fit using the labels in line 11. The images are classified with the model in line 12. The respective scores are stored in a score matrix. After the binary classifications are done, the maximum scores are found and the corresponding category is returned in lines 16 and 18.

```
1  function predicted_categories = svm_classify(train_image_feats, train_labels, test_image_feats)
2  categories = unique(train_labels);
3  num_categories = length(categories);
4  scores = zeros(size(test_image_feats,1),num_categories);
5
6  for i=1:num_categories
7          % two-class SVM for each category
8          labels = double(strcmp(categories(i), train_labels));
9          labels(labels==0) = -1;
10
11         M = fitcsvm(train_image_feats, labels);
12         [~,score] = predict(M,test_image_feats);
13         scores(:,i) = score(:,2);
14 end
15
16 [~, max_idx] = max(scores,[],2);
17 size(max_idx)
18 predicted_categories = categories(max_idx');
19 end
```

Results

Tiny image with nearest neighbor classifier achieved an accuracy of 0.214. Bag of words with nearest neighbor classifier had 0.439. Bag of words with SVM had 0.625. Figure 1 shows the confusion matrices of each cases.

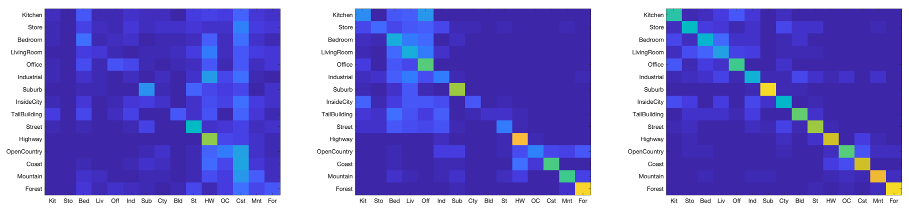

Figure 1: *Left:* Tiny image with nearest neighbor classifier *Middle:* Bag of words with nearest neighbor classifier. *Right:* Bag of words with SVM classifier

To maximize the accuracy, different parameters were adjusted. In the bag of words method, the cell size and interval of points to extract features from were adjusted. Different values are tested for accuracy and the result shows in Figure 2. The optimal value of cell size of 20 and interval of 16 was found.

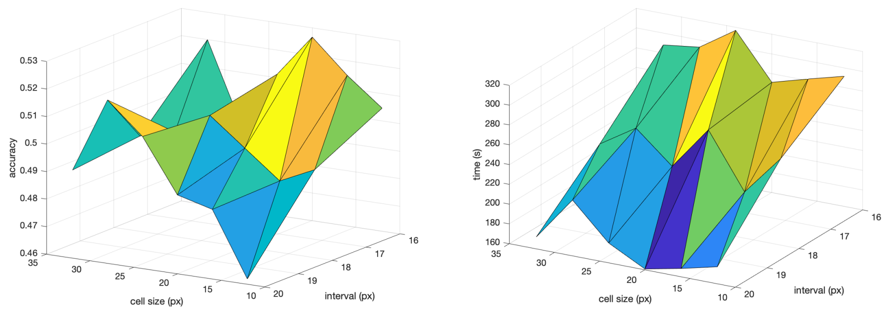

Figure 2: *Left:* Size vs Accuracy *Right:* Size vs Time

The vocabulary size is analyzed in Figure 3. The optimal vocabulary size was 1200.

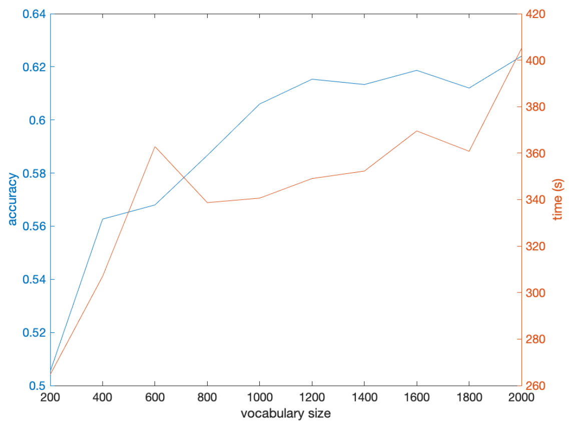

Figure 3: *Left:* Size vs Accuracy and Time

The *k* number is analyzed in Figure 4. The optimal *k* number was 6.

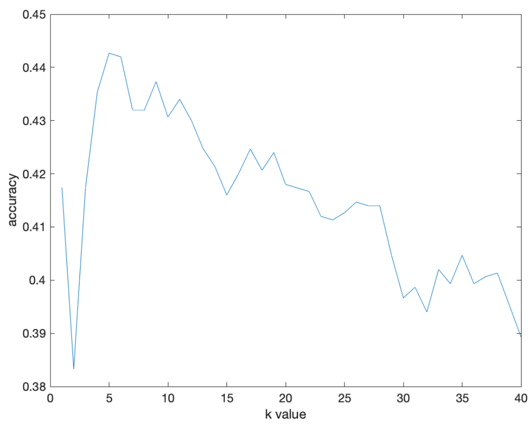

Figure 4: *Left:* k value vs Accuracy