Course work 2

Byoungjun Kim
KAIST
braian98@kaist.ac.kr

Junbeum Kim
KAIST
dungeon12345@kaist.ac.kr

Q1. K-means codebook

1.1. Vocabulary size and bag-of-words histogram

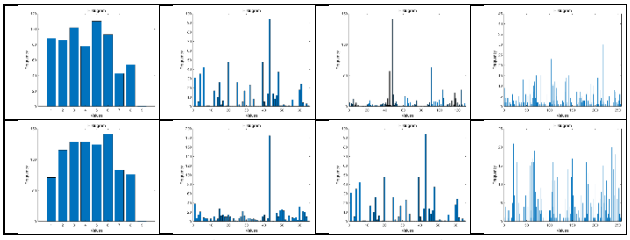


Figure 1. Histogram of train/test image. From left to right, vocabulary size is 8, 64, 128, 256.

Vocabulary size is equal to number of cluster in K-means clustering algorithm. If number is too small, it would be not enough to represents image and when it is too big, it would separate meaningless feature. In summary, histogram should show low entropy. Fig 1. shows bag-of-words histogram for training and testing data and it shows 64 has lowest entropy among above. It indicates 64 is appropriate vocabulary size compare to 4 of candidates.
Although having 100k descriptors, number of appropriate cluster is quite small. It is because k-means clustering algorithm forms same radius of cluster without considering distribution of data inside each cluster.

1.2. Vector quantization process

After creating k cluster using k-means algorithm with 100k descriptor, mean value in each cluster represents code word. For the vector quantization process, first we assign discrete value for code word, which is from zero to k-1. Then if we have n number of descriptor in image, we compare each n descriptor with code word in vocabulary and find the code word with smallest distance. Then make k-dimensional vector with i-th index value as number of i-th code word occurrence in above stage. Result k-dimensional vector is now can be used for downstream task and above process is vector quantization process.

We used vl_kmean library for making k-means codebook. Since initialization is important for k-means clustering algorithm, we used ‘plusplus’ which spreads initial mean point. Then we used ‘knnsearch’ to find input descriptor’s nearest neighborhood code word in code-book.

Q2. RF Classifier

2.1. Impact of Vocabulary Size on Accuracy

In this section, we tested the RF classifier via data of various vocabulary sizes. We have changed the cluster size from 4 to 256 and accumulated their test results in Fig 2. When the vocabulary size is small(i.e. 4) we can see that classification accuracy is relative lower than others. This is because feature of only 4 dimension does not have enough information need for classification. We can see that as vocabulary size increases the accuracy tends to increase also. But if the vocabulary size increases too much the accuracy starts to decrease again. As we make more clusters, the overall codebook size increase. But the number of codeword that are selected by majority of the descriptors doesn't increase that much there by creating a large feature with lots of useless dimensions. Therefore random forest classifier's splitting via the dimension is not very effective. To summarize, cluster number of K-means should be not too small or big. From our experiment, the choice of K=16 worked the best. Therefore we chose K=16 for the section 2.2 and 2.3.

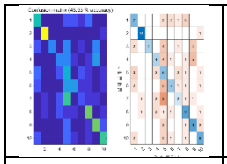
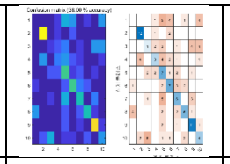
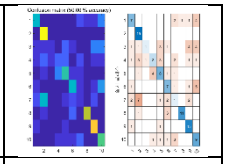
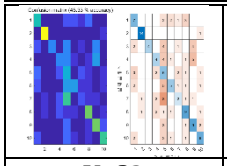
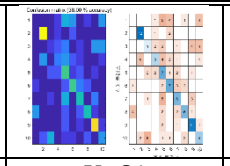
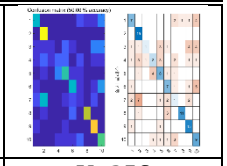
		
K=4	K=8	K=16
acc = 38.00%	acc =45.33%	acc =50.00%
		
K=32	K=64	K=256
acc = 46.00%	acc = 44.00%	acc = 43.33%

Figure 2. RF classifier accuracy test for various cluster number.

2.2 Effect of the RF classifier Parameters

In this section, we trained and tested random forest classifiers using the data created by K-means clustering codebook(K=16). We fixed the size of K-mean cluster in order to test the effect that are only from the RF classifier parameters. We have tested for varying the number of tree, tree depth, and split number. The default values for each parameters are 10, 10, 4, respectively. For this test, we utilized the codes from RF_code.zip which were provided

by TAs. The Experiment result is represented in Table 1. The example of failure sucess images and confusion matrix are in Fig 3.

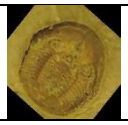
Tree Nums	Accuracy	Train Time	Test time
2	0.2733	0.0832	0.0187
4	0.4267	0.1408	0.0299
6	0.4800	0.2038	0.0403
8	0.5067	0.2663	0.0620
10	0.5267	0.3088	0.0684
Tree Depths	**Accuracy**	**Train Time**	**Test time**
2	0.3467	0.0145	0.0025
4	0.4267	0.0288	0.0048
6	0.4533	0.0790	0.0073
8	0.5667	0.1472	0.0186
10	0.5267	0.3088	0.0684
Split Nums	**Accuracy**	**Train Time**	**Test time**
1	0.4800	0.2760	0.0673
2	0.4733	0.2998	0.0635
3	0.5267	0.3088	0.0684
4	0.5333	0.3192	0.0625
5	0.5467	0.3400	0.0627

Table 1. Test results of RF classifier varying number of trees, tree depths and internal node's split number.

As the number of tree increases, the test accuracy also tends to increase. This is because as we use more trees, the overall forest classifier becomes more generalized. But as tree number increases, the training time and test time also increases linearly because we need to run more tests on those indiviual classifiers. Therefore if we increase the tree number we will be able to learn more generized classifier over computation tradeoff.

As the tree depths increases we can see that accuracy increases at the begining and starts to decreases at a certain point. When tree depth is too low, random forest will not be able to separable the data enough to make a correct classification. Resulting a large entropy at the leaf node so that it makes a false classification very often. But if the tree depth is too big, it will start to separe the data too much, making the classifier overfitting to the training data. It is straighforward that the training an test time increases as the tree gets deeper, since it has to go through more nodes.

Split number is for changing the degree of randomness parameter. It decides how many split it will check to find the best information gain split. So the randomness is inversely proportional to the split number. We can see that as split number increases the accuracy tends to increase. This is because since classifier with larger split number will checks more to find the best one, it is likely that is will find a split with better information gain. But since the split number we have tested are relatively small, the results contains a little volatility. Nevertheless, if we keep increasing the split number we expect that test accuracy will also increase until it gets too overfitted to the data. The training increases porpotionally to split number. It is because the internal nodes have to check more split to find the best one. But in the testing phase the split is fixed, therefore we can see that measured tiems are about the same.

Input Image		
predict	water_lilly	trilobite

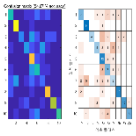
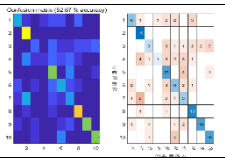
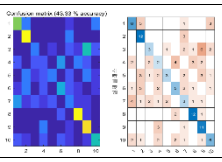


Figure 3 RF classifier Failure/Success, Confustion Matrix example

2.3 Effect of the Weak-Learners





axis-aligned	two-pixel
acc =52.67%	acc =47.33%
train= 0.3088	train= 0.4022
test=0.0684	test=0.0780

Figure 4. Comparison of test results for axis align and two pixel test .

In order to test the effect of the weak-learners, i.e. axis-aligned and two-pixel test, we had to change some parts of the corresponding codes of the given RF_code (codes for two-pixel tests are presented in Appendix A). After implementing two-pixel test, we have tested them on the default paramter setting and compared it with axis-aligned test. The comparsion is illustrated in Fig 4. We can see that train and test time for two-pixel test is slightly higher than axis-aligned. It seems pretty reasonable because in the split node phase, it has to check an additional dimension thereby doing some more calculations. The accuracy for two-pixel test were tended to be lower than axis-aligned. We think that since two-pixel test have much more combinations of dimension choices, It is likely to find a worse split compared to the axis-aligned. This is also because of the characteristics of our data. It contains too many unuseful values(almost 0 in the histogram). So if two-pixel test chooses one of these feature dimension the split will not gain much information well resulting in lower accuracy. So since two-pixel test have higher chance of meeting this problem, test accuracy tends to be lower than axis-aligned.

Q3. RF codebook

3.1. Vector Quantisation process

For implementing RF codebook's vector quantisation process, we have referenced the 'RF Codebook' Lecture. Since We assume that all of the descriptors from a identical image share the same label. So, in order to find which image a certain descriptor came from, we made a cumulative sum of the number of descriptors in the images. With this, we can specify the label of a descriptor by just looking up. Now that we have label of descriptors, we can

use it to train RF classifiers. We have utilized the given RF classifier code for this process. After training, we inferenced it with testTree_fast function and got histogram of arrived leaf node indices of each tree contained the forest. By combining theses leaf indices, we were able to get histogram size of (tree_num* leaf_count).
As seen in Fig 5, shows the lowest entropy so that it is most appropriate vocabulary size from among.

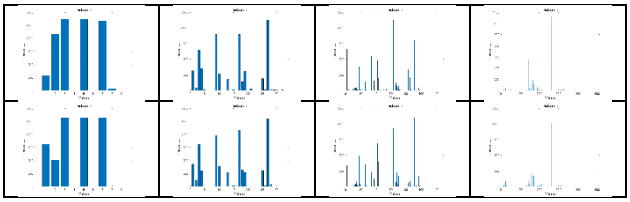


Figure 5. Histogram of train/test image. From left to right, vocabulary size is 8, 16, 32, 64.

3.2 Effect of RF codebook parameters

Tree Nums	Accuracy	Train Time	Test time
2	0.4400	4.6641	0.1363
4	0.4667	9.0302	0.2385
6	0.4733	13.283	0.3185
8	0.4467	17.217	0.3648
Tree Depths	**Accuracy**	**Train Time**	**Test time**
2	0.4667	3.2797	0.0949
4	0.4667	9.0302	0.2385
6	0.3800	15.010	0.5592
8	0.3200	26.314	1.5892
Split Nums	**Accuracy**	**Train Time**	**Test time**
1	0.4533	3.7260	0.2520
2	0.3733	6.2792	0.2203
4	0.4667	9.0302	0.2385
4	0.5333	11.353	0.2378
5	0.5467	14.256	0.2176

Table 2. Test results of codebook RF classifier varying number of trees, tree depths and internal node's split number.

Table 2 shows that performance tend to increase as number of tree in RF codebook increase. These is because in random forest, performance increase linearly by number of tree in forest. Train and test time also increase linearly by number of tree increase, but it would be fixed to some point if we use parallel execution among trees.

As tree depth increases, performance tend to increase to some point then decrease. If we increase tree depth each tree is able to represent more data so performance tend to increase. However, if we make tree too deep, it can be overfit to train data and performance decrease. Train and test time increase by tree depth increase, which is because number of calculating split function increase.

If we increase split-num randomness decrease and each split node perform better for train dataset. Since split node perform better accuracy tend to increase if we decrease randomness. However, for certain point, we expect accuracy would drop by overfitting. Train time tend to increase linearly since execution number of spilt function increased linearly. For test time, number of operation is equal so it doesn't change.

3.3 Effect of RF classifier parameters

Tree Nums	Accuracy	Train Time	Test time
2	0.3467	0.0647	0.0160
4	0.3533	0.1314	0.0307
6	0.3600	0.2163	0.0501
8	0.3733	0.2736	0.0561
10	0.4267	0.3291	0.0645
Tree Depths	**Accuracy**	**Train Time**	**Test time**
2	0.2733	0.006536	0.000943
4	0.4067	0.0346	0.0031
6	0.4400	0.0820	0.0078
8	0.4133	0.2044	0.0340
10	0.4267	0.3291	0.0645
Split Nums	**Accuracy**	**Train Time**	**Test time**
1	0.3867	0.2845	0.0425
2	0.4733	0.3041	0.0605
3	0.4267	0.3291	0.0645
4	0.4533	0.3375	0.0695
5	0.3867	0.4287	0.0719

Table 3. Test results of classifier RF classifier varying number of trees, tree depths and internal node's split number.

Table 3 shows result varying split-num in RF classifier. Result and reason is same with explanation of Table 2.

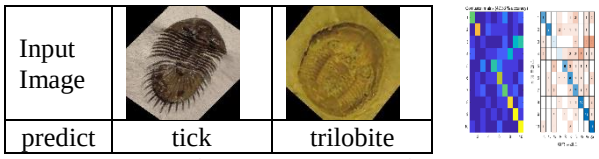


Figure 6. RF classifier Failure/Success Confustion Matrix example

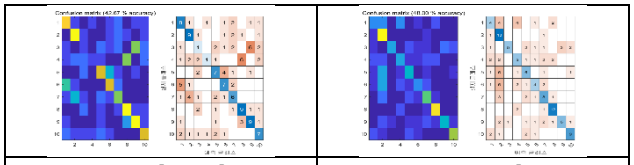


Figure 7. Comparison of test results for axis align and two pixel test.

Fig 6. shows failure case of RF classifier. As can observe in figure, although it is failure case, we can see classifier classified to similar class. In Fig 7, two-pixel showed better result than axis-aligned in RF codebook. However, these difference is negligible since RF codebook and classifier performance varies significantly by random initialization. Because of this randomness injection, RF codebook performance was lower than Q2. However, RF codebook would outperform if we tune randomness parameter or train with bigger data.

Q4. Convolutional Neural Networks

Before conducting experiments and compare result, set default as follow and only changed each manipulated variable. # of layer = 5, hidden layer channel = 64, dropout probability = 0, no skip connection, softmax loss, output layer = FC layer, learning rate = 0.001, batch size = 128, no pretrain, # epoch = 20.

4.1. Various Network Design Test

#layer	Acc
3	0.911
5	0.871
10	0.687
Filter size	Acc
3	0.894
5	0.871
10	0.832
Skip connection	Acc
None	0.871
[2]	0.913
[2, 4]	0.871
Norm Type	Acc
Batch norm	0.871
Layer norm	0.507
Instance norm	0.914
Group norm	0.623
regularization	Acc
none	0.871
L1	0.856
L2	0.905

Loss function	Acc
Softmax	0.871
Squared hinge	0.891
Compress	Acc
Fc	0.871
svd	0.880
Learning rates	Acc
0.001	0.871
0.005	0.918
0.01	0.860
Batch size	Acc
64	0.917
128	0.871
256	0.801
Initailization	Acc
From scratch	0.871
pretrain	0.907

Table 4. Test results of various CNN Designs

First, as we increase the layer it shows decrease in accuracy. Within deeper layer, since it is more complicate model it should get higher accuracy but the test result show differently. That is because for test efficiency we set number of epoch to 10 which can implies number of 10 epoch is not enough for complex model to fully learn. Also within deeper layer, without residual connection, by gradient vanishing problem, performance can be low.

As we increase filter size, accuracy tends to drop as filter size become bigger. This is because if we increase filter size, we have to learn more parameter and accuracy drop can be affected by underfitting. Also, if we increase filter size, it cannot extract local feature compare to small size filter which makes latent variable with lack of diverse representation.

For adding skip connection, since we have 5 number of layer, conducted experiment with adding skip connection at 2^{nd} layer and 2^{nd}, 4^{th} layer. Adding skip connection to 2^{nd} layer only shows best result. This is because layer is not very deep and adding to 2^{nd} and 4^{th} at same time need more time to converge since model should learn in more diverse space. However, adding skip connection to 2^{nd} layer shows better result than adding no skip connection. That is because residual connection itself can help deeper layer to learn original context.

Among different batch normalization technique, instance normalization shows best result. Choosing normalization techniques can vary from task and architecture and instance normalization is suitable for spatial information of each differs a lot and since our dataset varies a lot by class, instance normalization showed best result.

L2 regularization shows best result because L2 normalization lead to smoother and more stable convergence in gradient based optimization.

Squared hinge loss shows better result than softmax loss. Choosing loss function depends on the task, however, squared loss function is margin based loss function which penalize prediction within boundary so that different class can be more separate in feature space. Also it is robust to outliers since squaring reduce the impact of extreme value.

Next, we Compared it using fc layer and SVD. Number of parameter is same(i.e. 982479) for since both have same number of inner structure and fc layer only varying process after fc layer. Within same number of parameter, SVD shows better result which is because SVD reduce dimension so that it can be robust to noisy response. Also, SVD help model to focus on significant value which is singular value and it's corresponding vector, it help model to not overfit and leads to better performance.

Accuracy tends to increase as learning rates increase at certain point, but get lowest accuracy with learning rates 0.01. This is because if we have higher learning rates, model can converge faster and helps model to jump out the local minima and that is why 0.005 was better than 0.001. However, if we set too big value for learning rates, it can make model to jump around without converge into certain point.

Small batch size makes model to update its weight within high frequency. Which is also benefit in aspect of avoiding local minima. Therefore, 64 batch size shows best result. However, it is not always correct. For different case, it can lead to unstable learning.

We pretrained our model with CIFAR10 with 20 epoch. With pretrain model shows better result in accuracy. This is because loading weights from pretrain helps model to start from good local minima which can make higher possibility to converge to global minima.

4.2. Compare with Q1, Q2

Compared to Q1 and Q2, CNN outperformed accuracy. We think it is beaause CNN has better inductive bias for learning image features. However training time took significantly longer since it has much more parameters and they are updated each iteration.

Appendix

Appendix A. Implememtations

A.1. RandForest Two-pixel test

```
if two_pixel == 1
    for n = 1:iter
        % Two-pixel test
        dim1 = randi(D);
        dim2 = randi(D);
        while dim2 == dim1
            dim2 = randi(D);
        end
        d1_values = data(:, dim1);
        d2_values = data(:, dim2);
        diff_values = d1_values - d2_values;
        t_diff = rand * (max(diff_values) - min(diff_values)) + min(diff_values);
        idx_ = diff_values < t_diff;
        ig = getIG(data, idx_);
        if (sum(idx_) > 0 && sum(~idx_) > 0)
            [node, ig_best, idx_best] = updateIG_two_pixel(...
                                        node,ig_best,ig,t_diff,idx_,dim1,dim2,idx_best);
        end
    end
end

function [node, ig_best, idx_best] = updateIG_two_pixel(...
                                    node,ig_best,ig,t,idx,dim1,dim2,idx_best)
if ig > ig_best
    ig_best = ig;
    node.t = t;
    node.dim1 = dim1;
    node.dim2 = dim2;
    idx_best = idx;
else
    idx_best = idx_best;
end
end
```

splitNode.m

```
if two_pixel
            if ~tree(T).node(n).dim1
                leaf_idx = tree(T).node(n).leaf_idx;
                if ~isempty(tree(T).leaf(leaf_idx))
                    label(idx{n}',T) = tree(T).leaf(leaf_idx).label;
                end
                continue;
            end
            idx_left = data(idx{n},tree(T).node(n).dim1) - ...
                       data(idx{n},tree(T).node(n).dim2) < tree(T).node(n).t;
            idx{n*2} = idx{n}(idx_left');
            idx{n*2+1} = idx{n}(~idx_left');
```

testTrees_fast.m

```
if two_pixel
            while tree(T).node(idx).dim1 && tree(T).node(idx).dim2
                t = tree(T).node(idx).t;
                dim1 = tree(T).node(idx).dim1;
                dim2 = tree(T).node(idx).dim2;
                % Decision
                if data(m,dim1)-data(m,dim2) < t % Pass data to left node
                    idx = idx*2;
                else
                    idx = idx*2+1; % and to right
                end
            end
```

testTrees.m

```
% Initialise base node
    tree(T).node(1) = struct('idx',idx,'t',nan,'dim',-1,'dim1',0,'dim2',0,'prob',[]);
```

growTrees.m

A.2. K-means visual codebook

```
% K-means clustering
        numBins = 16;
        %% write your own codes here
        tic
        vocab = vl_kmeans(desc_sel, numBins,'Initialization', 'plusplus', ...
                          'algorithm', 'lloyd') ;
        disp('Encoding Images...')
```

```
% Vector Quantisation
        %% write your own codes here
        % ...

        histogram_tr = zeros(length(classList) * imgSel(1), numBins);
        cnt = 1;
        for c = 1:length(classList)
            for i = 1:imgSel(1)
                quantized_descriptors = knnsearch(vocab', single(desc_tr{c, i}'));
                histogram_tr(cnt, :) = histcounts(quantized_descriptors, 1:numBins + 1);
                cnt = cnt + 1;
            end
        end
        % ...
        toc
        test_img_idx = cell(1, length(classList));
        for c = 1:length(classList)
            test_img_idx{c}= imgIdx{c}(imgSel(1)+1:sum(imgSel));
        end
```

getdata.m

A.3. RF visual codebook

```
% Build visual vocabulary (codebook) for 'Bag-of-Words method'
        [desc_sel,y] = vl_colsubset(cat(2,desc_tr{:}), 10e4);
        desc_sel = single(desc_sel);
        %% write your own codes here
        tic

        cum_sum = cumsum(cellfun(@(x) length(x), desc_tr(:)));
        Y = arrayfun(@(index) find(index <= cum_sum, 1, 'first'), y);
        data = cat(1, desc_sel,Y);

        param.num = 4;     % number of trees
        param.depth = 4;    % trees depth
        param.splitNum = 5; % Number of trials in split function
        numT = param.num;
        depth = param.depth;
        split = param.splitNum;
        param.split = 'IG';
        trees = growTrees(data',param);
```

```
% Vector Quantisation
        %% write your own codes here
        % ...

        numBins = 2^(param.depth-1);
        histogram_tr = zeros(length(classList) * imgSel(2), numBins * param.num);
        cnt = 1;
        for c = 1:length(classList)
            for i = 1:imgSel(1)
                desc_tr_size = size(desc_tr{c,i});
                z = cnt* ones(1, desc_tr_size(2));
                train_data = cat(1,desc_tr{c,i},z);
                leaves = reshape(testTrees_fast(train_data',trees), [], 1);
                histogram_tr(cnt, :) = histcounts(leaves, 1: numBins*param.num + 1);
                cnt = cnt + 1;
            end
        end
```

getdata.m