MLCV Coursework 2 Report

Shinyoung Yu
20200405
School of Computing
sahaa0918@kaist.ac.kr

Jiyun Park
20210263
School of Computing
jiyun02@kaist.ac.kr

1. K-means codebook

We implemented the classification process following the visual codebook pipeline provided in the instruction. First, we extracted 100K SIFT feature descriptors from the training data and applied K-means clustering to construct a visual vocabulary.

1.1. Vocabulary size of K-means clustering

Theoretically, a smaller visual vocabulary size K increases the risk of underfitting but offers better time-wise efficiency. Conversely, a larger K poses a higher risk of overfitting and results in lower efficiency. Theoretically, the time complexity of K-means clustering training and vector quantization is proportional to the value of K, and our result Fig. 1 align with it.

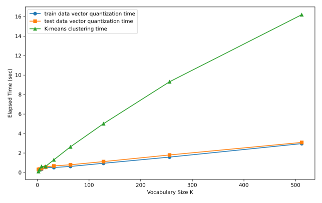


Figure 1. K-means visual vocabulary: training and quantization time according to the value of k

1.2. Vector quantisation process

To quantize images, the SIFT feature descriptors extracted from each image were assigned to the nearest vocabulary cluster, and a histogram vector was generated based on the vocabulary frequency of such assigning result. Consequently, after vector quantization, the dimension of each image was reduced to the size of visual vocabulary, denoted by K. The visualisation result of those histogram according to the various word size is visualized on Fig. 2. This example also demonstrates the issues of underfitting and overfitting with respect to the size of K. When $K = 4$, the histogram cosine similarity between 2nd and 3rd image is higher than

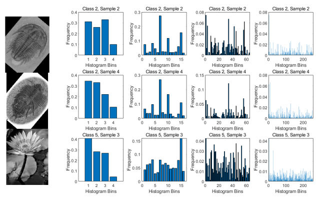


Figure 2. K-means visual vocabulary: training and quantization time according to the value of k

between 1st and 2nd image. However as the value of K increases, the similarity between 1st and 2nd image which are in same class becomes higher. Additionally, if the value of K becomes excessively large, the inner-class similarity decreases, which can lead to the potential problem of overfitting. Various sample histogram of train and test images and exact measurement of cosine similarity is in Appendix A and Appendix B

2. RF classifier

The random forest model has several key hyperparameters, and testing them all simultaneously would require excessive effort and time. Therefore, we fixed the optimal parameters in the sequence described below. Also, to ensure result stability, each test was executed 10 times. The average of 10 results displayed in a graph, and all the confusion matrix represents the best result among those. All training and testing time was measured excluding the codebook generation and vector quantization time; in other words, only the time spent on random forest classification applied on the 256-dimension quantized vectors was measured.

1. Number of Trees & Tree Depth: We tested using an axis-aligned weak learner with the split number fixed at 10.
2. Split Number (Randomness Parameter): Using the optimal number of trees&tree depth derived in step 1 and axis-aligned weak learner.
3. Type of Weak Learner: Using the optimal values for other parameters determined in steps 1 and 2, we tested different weak learners.

2.1. Number of trees & The depth of trees

Changing the number of trees and depth, we extracted the result in Fig. 3a. In the graph, the best accuracy 0.685 occurred at N(number of trees) $= 250$ and D (depth) $= 8$, explained as follows:

- Number of Trees: A single tree in random forests tends to overfit to data; therefore, we can generalize the model using ensembles of trees. As shown in Fig. 3b, the test accuracy converges near $N = 250$, which is selected as optimal parameter.
- Tree Depth: When depth is D, maximum of $2^D - 1$ nodes are generated. As shown in Fig. 3c, for a given N, test accuracy initially increases with tree depth but later decreases due to overfitting. Notably, the optimal depth D increases with N, indicating that a larger N result in less overfitting.

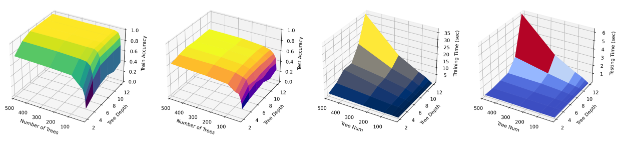
When $N =$ (number of trees), $D =$ (depth) and $S =$ (number of attempts to split per node), the theoretical training/testing time is as follows, and our training time results almost align with theoretical predictions as shown in Fig. 3b, Fig. 3c. However, since the testing time is generally more than 10 times shorter than the training time, it is sensitive to noise caused by memory loading time. Unlike our expectations, the testing time graph with respect to the value of D is not seem as linear complexity, due to the influence of noise caused by the exponentially increasing memory with tree depth.

- Training time: $O(NDS)$
- Testing time: $O(ND)$
- Space complexity (= Number of nodes): $O(N^{(D-1)})$

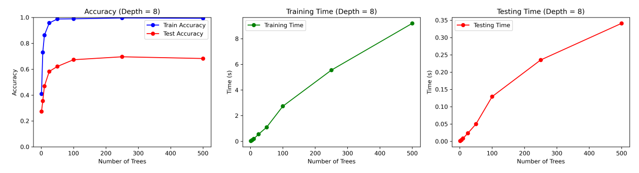
Moreover, our code do not utilize parallelization of each tree's growth. Based on theoretical insights from course materials, training each tree in parallel could reduce the time complexity from $O(N)$ to $O(1)$ with respect to the number of trees.

2.2. Randomness parameter

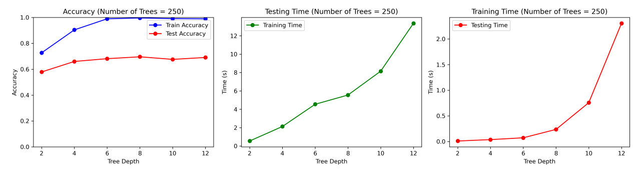
We randomly select dimensions and threshold values to create the split function. For each split function, we attempt ρ random splits and choose the one with highest information gain. There is a trade-off in the magnitude of ρ value: If the ρ value is too low, fewer splits are attempted, which reduces similarity between trees but increases the risk of less-optimal split. Conversely, as ρ increases, more various split functions are tested, leading to greater similarity among trees, which decreases the advantage of ensemble multiple trees. As shown in Fig. 4, test accuracy initially increases with higher ρ values until $\rho = 10$ and then decreases, as expected. Training and testing time is same as the expectation in Sec. 2.1.



(a) Training&Testing accuracy and time according to the number of tree and the depth of tree



(b) Training&Testing accuracy and time according to the number of tree (D = 8)



(c) Training&Testing accuracy and time according to the depth of tree (N = 250)

Figure 3. Training&Testing accuracy and time according to the number and depth of trees

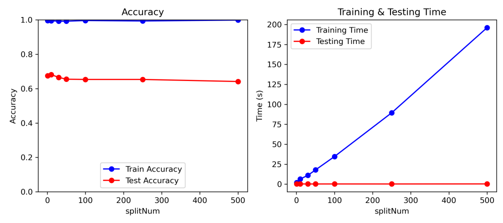


Figure 4. Training&Testing accuracy and time according to the randomness parameter

2.3. Impact of the vocabulary size on classification accuracy

We fixed the random forest classification parameters obtained from the previous experiments and varied the K-means vocabulary size. As shown in Fig. 5, the vocabulary size used in vector quantization significantly affects both test and train accuracy. As expected, when the value of K is small, because of the insufficient extraction of important features from the images, underfitting occurs, resulting in low training and testing accuracy. Up to $K = 16$, train and test accuracy increase rapidly, but beyond that point, the rate of increase for both becomes smaller. Particularly, when K increases from $K = 256$ to $K = 512$, train accuracy shows a slight improvement, but due to overfitting, test accuracy decreases afterwards. This is because vector quantization extracts unnecessary small details from the images, reducing inner-class vector similarity.

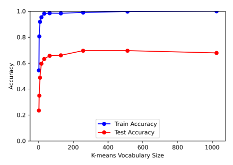


Figure 5. Training&Testing accuracy according to the K-means vocabulary size

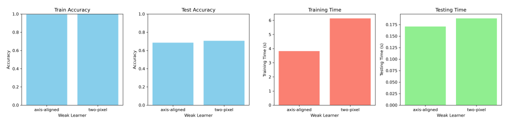


Figure 6. Axis-aligned vs Two-pixel test

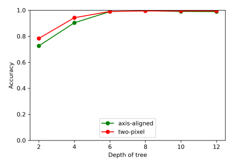


Figure 7. Axis-aligned vs Two-pixel test: Train accuracy according to depth of tree

2.4. Impact of the different types of weak-learners

Fig. 6 shows the impact of weak learner. As expected, the axis-aligned method provides sufficiently good testing accuracy with shorter training and testing times. However, the two-pixel test achieves 2.13% better accuracy then axis-aligned under the same parameters. You can see Fig. 7, indicating that the two-pixel test has higher information gain per split. Other methods, such as linear and non-linear weak learners, were also tested but omitted from the report due to impractically long training and testing times.

The main results' confusion matrix, example success/failures, example node's histogram representing information gain along splitting are written in Appendix C and Appendix D

3. RF codebook

4. Convolutional Neural Networks

CNNs are strong architecture for computer vision tasks. We are going to explore its performace on image classification.

4.1. CNN architecture

We desinged our CNN with
- 4 convolutional layers with ReLU and Maxpooling
- 4 fully connected layers with drop-out and ReLU
- Residual connections to every one layers
- Use CrossEntropy as a loss function for optimization
- The size of convolutional filter size 3

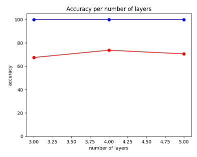
In addition, the input of our CNN is resized to 128*128 as Caltech101 dataset has image sizes 100 to 300. Also, the input image channel is 3(RGB) and it is amplified up to 256 through convolutional layers. For the output, it is reduced to 10 channels as we have 10 classed for the classification. Above is our basic model. We tried to choose simplest yet high-performance model. Detailed reasons for our choice will be introduced below.

4.2. Changing architecture

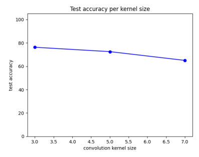
First, the number of layers. We changed the number of layers from 3 to 5. As shown in **??**, the test accuracy of CNN is highest when the number of layers is 4. It may because, when CNN has 3 layers, it is not enough to capture features from images and when it has 5 layers, it may occur overfitting due to many weights and gradient vanishing. Hence, we concluded that 4 layers are the best choice for the performance.

Next, we changed the size of kernel to 3, 5, and 7 to reveal its impact on CNN. As Fig. 8b shows, the accuracy is highest when the kernel size is 3 and it decreases as the kernel size incrases. It is because, when the kernel size is small, it is adequate to capture the details of images and when the size is big, it is good to percieve the global scene. However, as our image has 128*128 size, which is small, kernel size 3 is enough to extract useful feature of the image. other sizes are too big to recognize the features corresponding the image label.

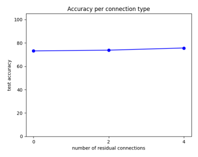
Lastly, we conduct the experiments on connection between layers. There are three cases: no connection between layers, connect every 2 layers, connect every one layers. Since we have total 4 layers, the total number of connection is 0, 2, and 4. The test accuracy are shown in Fig. 8c. The accuracy is increases as the number of connection incrases. It is because, the more connection CNN has, the more information it gets. By residual connection, CNN does not lose the information from initial layers thus it can learn more general information about images and its label.



(a) Accuracy according to layers



(b) Accuracy according to kernel size



(c) Accuracy according to connections

Figure 8. impact of changing architecture of CNNs

4.3. Batch normalization

4.4. Generalization

There are many method for Generalization of CNN. In this section, we are going to check the impact of drop-out and L2-regulization on weights(which is in Deep_Learning_intro Lecture Note pg.84) for the generalization. We test our basic model which has drop-out, the basic model without drop-out, and the basic model with L2-regulization on weights. Results are shown in Tab. 1. The model without drop-out has the lowest accuracy and the model with L2-regulization has the highest accuracy. This is because, drop-out randmly deactivate some neurons, so it helps model to avoid overfitting and be good at generalization. And L2-regulization helps model to avoid overfitting by regulize the size of weights. Hence, it is natural result that the model with both drop-out and L2-regulization has the best performance.

Table 1. Impact of generalization

	accuracy
original	75.625
without dropout	68.125
with L2-regulization term	76.250

4.5. Loss function

We use CrossEntropy, which is based on softmax function, as a loss function as usual. And compare it with SquaredHinge loss.

Using CrossEntropy for the loss function, we get 73.75 accuracy. However, when we use SquaredHinge loss, we get 80.00 accuracy. The accuracy with SquaredHinge loss is higher. This is because, as CrossEntropy is based on probability distribution and SquaredHinge loss is based on the margein between expected and real classes, SquaredHinge is better since our dataset is small. (which has only 10 classes) Because the dataset is small, it is hard to make an accurate probability distribution due to the law of large numbers. Hence, it is better to use direct borders and margins between classes. Despite this reason, we adopted CrossEntropy as a basic loss function as it is widely used.

4.6. Compressing CNN

We conduct an experiment on compressing layers of CNN using truncated SVD. Since our model has 4 layers with rank 512, 128, 32, 10, we compress first two layers with 100(original), 75, 50, 25 percent each because last two layers are too small to compress, thus it might be cause the huge loss of data when they are compressed. The experiment result are shown in Fig. 9. Test accuracy and test time are both decrased as layers are more compressed. It is because, as weight matrix of layers become compressed, there remain less computation to get the result, so, the test time decrased. However, as the matrix is more compressed, it causes more data loss than the original weight matrix.

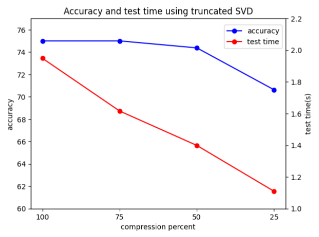


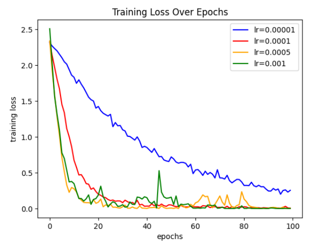
Figure 9. Accuracy and test time using truncated SVD

4.7. Changing parameters

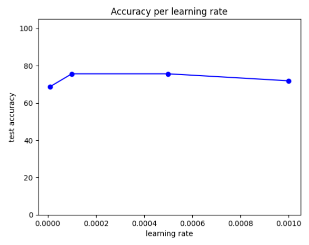
We observe the influence of main hyper-parameters by experiments. We changed 3 parameters from base model: learning rate, batch size, and the numger of epochs. Fig. 10a and Fig. 10b indicates the result of varying learning rate. We changed learning rate to 0.00001, 0.0001, 0.0005, and 0.005. As shown in pictures, when the learning rate is 0.00001, the model doesn't converge sufficiently since it is too small, thus it can not reach the local minimum of loss during the given epochs. Conversly, when the learning rate is 0.0005 and 0.001, it vibrates too much until it reaches the convergence point. More specifically, amplitude of 0.001 is larger than 0.0005. This is because, the amount of moving is too big, so, the model oscillates near the local minimum largly. Therfore, we decided that learning rate = 0.0001 is the best for our model. Also, regarding the accuracy, the accuracy is higher when the learning rate is 0.0001 and less than this when the learning rate is smaller or bigger than 0.0001. It is because, if the learning rate is too small like 0.00001, it can not converge in the training time sufficiently. And if the learning rate is too big, it might pass through optimization point.

4.8. Pre-trained model

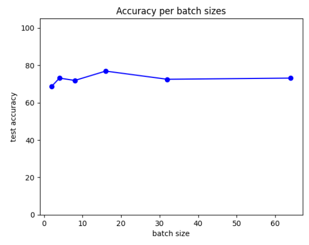
In this section, we compare training methods: fine-tuning pre-trained model and training from scratch from random initialization. Since our CNN does not have pre-trained version because we have only one dataset for both training and test, we used pre-trained ResNet model(microsoft/resnet-50, trained with ImageNet) and custom it with our Caltech101 dataset. Due to the time limit, we only train it with 10 epochs. When we use pre-trained weights and fine-tuning them, we get 0.9933 accuracy. And when we train the model from scratch, we get 0.2133 accuracy which is lower than before. Based on this result, with the limited amount of time and resources, it is much better to use pre-trained model to get high performance. It may because, since we used pre-trained model with same task(image classification), trained weight is specialized for image classification even with different dataset. Therefore, fine-tuning using a custom dataset can achieve higher accuracy with fewer training epochs than train from scratch.



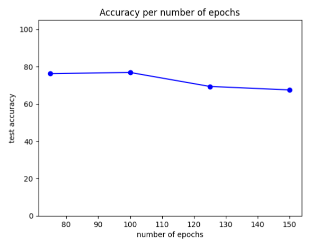
(a) Train losses according to learning rates



(b) Accuracy according to learning rates



(c) Accuracy according to batch sizes



(d) Accuracy according to epochs

Figure 10. Changing hyper-parameters of CNN

4.9. Comparison with our methods

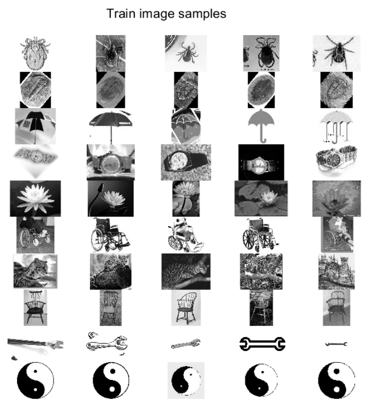
Considering that our CNN have accuracy about 75, it is much better model than RF classifier and RF codebook on classifying Caltech101 dataset. Tab. 2 This may because, CNN can learn hierarchical features through layers, so, it can capture various characteristics of images. On the other hand, RF classifier and RF codebook use fixed method for extracting features from images, so they can only learn limited features.

	Train Accuracy (%)	Test Accuracy (%)
K-means Codebook - RF Classifier	99.50	68.5
RF Codebook - RF Classifier	97.80	60.67
CNN	100.00	75.625

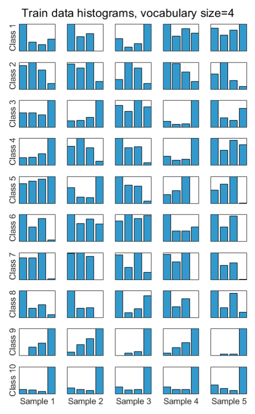
Table 2. Accuracy of Models

Appendix

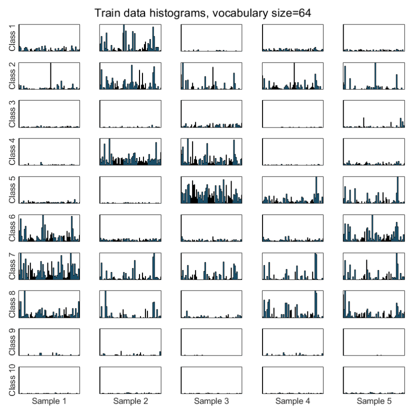
A. Q1:Visualization of quantization result of sample training / testing images



(a) Train image samples

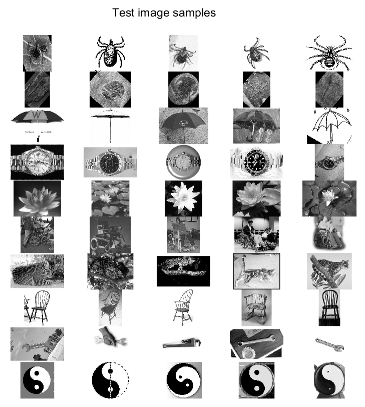


(b) Train data quantisation result, $K = 4$

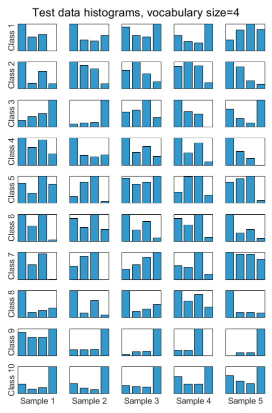


(c) Train data quantisation result, $K = 64$

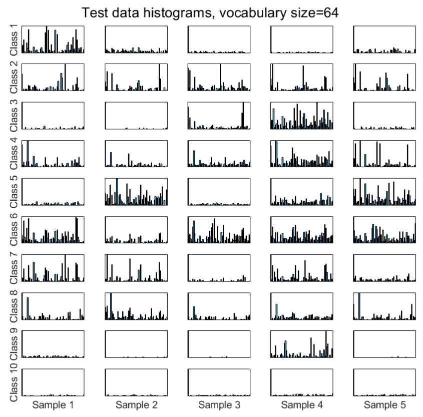
Figure 11. Visualistaion of train data quantisation result



(a) Test image samples



(b) Test data quantisation result, $K = 4$



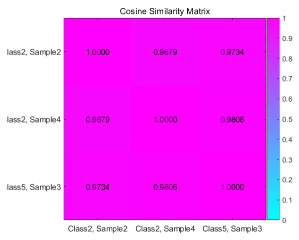
(c) Test data quantisation result, $K = 64$

Figure 12. Visualistaion of test data quantisation result

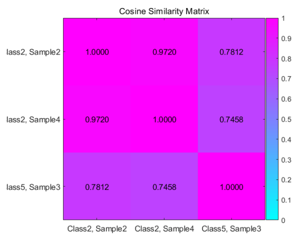
B. Q1:Cosine similarity between images in Fig. 2

C. Q2: Test result's confusion matrix and success/failure cases

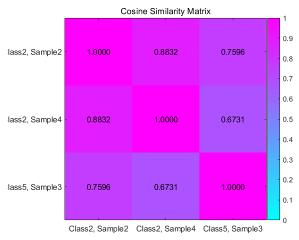
D. Q2: Visualization of random forest's information gain process



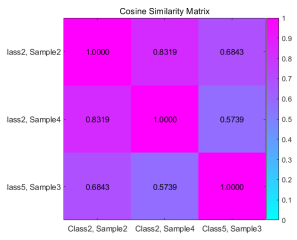
(a) $K = 4$



(b) $K = 16$



(c) $K = 64$



(d) $K = 256$

Figure 13. Visualization of cosine similarity matrix of Fig. 2 according to the different K