Dog Recognition System:

The Effects of Different Classifier on Accuracy of Dog Classification

Xuanyu Liu, Jingxu Wang and Zhe Han

Stevens Institute of Technology

Abstract

The study of this project is to develop a simple image recognition system. In this system, we are asked to input a text file, where each line gives the full path to a dog image file (called train dataset), and a “1” or a “0” designating whether the image belongs to a dog category that the system is being asked to learn. Then the system reads another text file, with full paths to images, but no information regarding category belonging. This set of images is treated as a test dataset. Our system should learn from the first set of images, then make predictions on the second set, and compute an accuracy of the machine predictions. Our solution is based on MATLAB, which has toolbox for machine learning and feature detection. First we installed vlfeat on MATLAB and run histogram of oriented gradients (HoG) method to extract the features of every images. Then our team trained four different classification models to classify these features. The classifications are Support Vector Machine (SVM), Naive Bayes (NB), K-Nearest Neighbors (KNN), and Decision Trees (DT). Using these four trained classification models, we predicted the category of test dataset. Finally we computed the accuracy of predictions and found SVM has a better performance on classification.

Dog Recognition System:

The Effects of Different Classifier on Accuracy of Dog Classification

1. Introduction

Object recognition is a basic application domain of image processing and computer vision. For many decades it has been – and still is – an area of extensive research. The common proceeding of most of the schemes is that, given some knowledge about the appearance of certain objects, one or more images are examined in order to evaluate which objects are present and where. Apart from that, however, each application has specific requirements and constraints. This fact has led to a rich diversity of algorithms. For our system, we need to find an optimal feature detection and extraction, then try to run different classifications to figure out which one has a better performance on our research problem.

2. Dataset

We chose Oxford-IIIT-Pet dataset and Caltech 256 as our dataset. First we manually extracted 2000 images of dog from Oxford-IIIT-Pet dataset, and 2000 images of notdog from Caltech 256. This dataset is our basic dataset. Then we built an algorithm to pick 1500 dog images and 1500 notdog images randomly as our train dataset. The rest is our test dataset.

3. Feature Detection and Extraction

3.1 Histogram of Oriented Gradients (HoG)

The HoG is a feature descriptor used in image processing for the purpose of object detection. Navneet Dalal and Bill Triggs described HoG descriptors at their paper “Histograms of Oriented Gradients for Human Detection”. The technique counts occurrences of gradient orientation in localized portions of an image. The merit of Hog descriptors is the better performance, the two scientist said, “The proposed descriptors are reminiscent of edge orientation histograms, SIFT descriptors and shape contexts, but they are computed on a dense grid of uniformly spaced cells and they use overlapping local contrast normalizations for improved performance. ” This is why we chose this algorithm to extract features.

We chose VLFeat library for extracting hog features. It is a open source library implements popular computer vision algorithms specializing in image understanding and local features extraction and matching. HoG exists in many variants. VLFeat supports two: the UoCTTI variant and the original Dalal-Triggs variant (with 2×2 square HoG blocks for normalization). The main difference is that the UoCTTI variant computes both directed and undirected gradients as well as a four dimensional texture-energy feature, but projects the result down to 31 dimensions. Dalal-Triggs works instead with undirected gradients only and does not do any compression, for a total of 36 dimension.

The function is used to get HoG features of a image:

hog = vl_hog(image, cellSize, 'verbose', 'variant', 'dalaltriggs').

4. Classification

Classification is a type of supervised machine learning in which an algorithm "learns" to classify new observations from examples of labeled data. In this system, our team explores four main classification of object detection and recognition.

4.1 Support Vector Machine (SVM)

Our team first trains a support vector machine, and then cross validate the classifier. Use the trained machine to classify (predict) new data. In addition, to obtain satisfactory predictive accuracy, we tune the parameters of the kernel functions (kernel scale and box constraint).

Training an SVM Classifier

The function *fitcsvm* is used to train, and optionally cross validate, an SVM classifier. The syntax is:

SVMModel = fitcsvm(X,Y,'KernelFunction','rbf','KernelScale','auto','BoxConstraint',InF);

The inputs are:

X — Matrix of train data, where each row is one observation.

Y — Array of train labels with each row corresponding to the value of the corresponding row in X.

KernelFunction — The value 'rbf' is the default for one-class learning, and uses a Gaussian radial basis function. Compared to ‘linear’, ‘rbf’ has a better performance in our system.

Standardize — Flag indicating whether the software should standardize the predictors before training the classifier.

The resulting, trained model (SVMModel) contains the optimized parameters from the SVM algorithm. Then, we use this model to classify new datas.

Classifying New Data with an SVM Classifier

Classify test data using *predict* function. The syntax for classifying new data using a trained SVM classifier (SVMModel) is:

[predicted_label] = predict(SVMModel, test_data);

The resulting vector, label, represents the classification of each row in test_data.

After we get the predictions, we use *getAccuracy* function to gain the accuracy of this classification. The syntax of computing accuracy is:

[accuracy] = getAccuracy(predicted_lable, real_lable);

real_lable — The actual classification of each row in test_data.

Tuning an SVM Classifier

To retrain the SVM classifier by adjusting the 'KernelScale' and 'BoxConstraint' name-value pair arguments. ‘BoxConstraint’ is a strategy to try a geometric sequence of the box constraint parameter. For example, take 11 values, from 1e-5 to 1e5 by a factor of 10. ‘KernelScale’ is a strategy to try a geometric sequence of the RBF sigma parameter scaled at the original kernel scale. Suppose the ‘KernelScale’ we got from ‘auto’ is *m*. The new kernel scales factors is the 1e-5 to 1e5 times of the original.

4.2 K-Nearest Neighbors (KNN)

KNN classification is one of the simplest of classification algorithms available for supervised learning. The idea is to search for closest match of the test data in feature space.

The syntax for classifying test data by using function *getKnn*:

[predicted_lable] = getKnn(train_data, train_lable, test_data, K);

K — the number of neighbors.

To retrain the KNN classifier by adjusting the value of K.

4.3 Naive Bayes (NB)

The Naive Bayes classifier is a classification whose algorithm is based on Bayes rule, that assumes the attributes $X_1 ... X_n$ are all conditionally independent of one another, given Y. The value of this assumption is that it dramatically simplifies the representation of P(X|Y), and the problem of estimating it from the training data.

Naive Bayes models assume that observations have some multivariate distribution given class membership, but the predictor or features composing the observation are independent. This framework can accommodate a complete feature set such that an observation is a set of multinomial counts.

To train a Naive Bayes model, we use *fitcnb* function. After training, predict labels or estimate posterior probabilities by passing the model and predictor data to the *predict* function.

The syntaxes for training a naive bayes model and predicting the classification of test data are:

```
NBModel = fitcnb(train_data',train_lable,'Crossval','on');
NB_predicted_lable = predict(NBModel,test_data');
```

4.4 Decision Trees (DT)

Decision trees, or Classification trees and regression trees, predict responses to data. To predict a response, follow the decisions in the tree from the root (beginning) node down to a leaf node. The leaf node contains the response. Classification trees give responses that are nominal, such as 'true' or 'false'. Regression trees give numeric responses.

We use *fitctree* method to create decision trees. After creating a tree, we can easily predict responses for new data. The syntaxes for training a decision tree and predicting the classification of test data are:

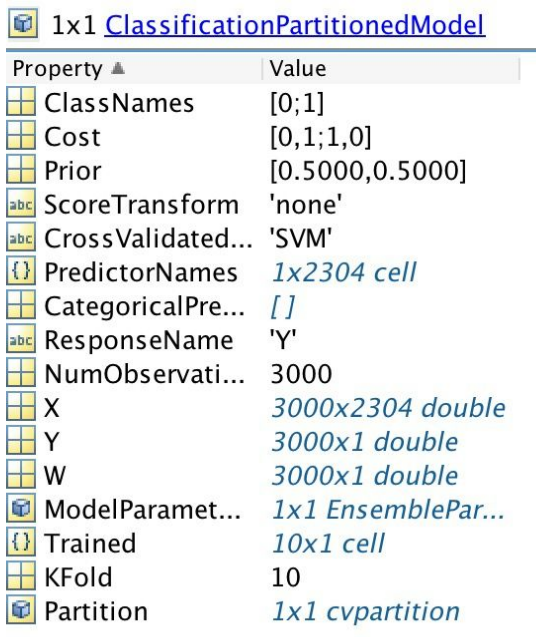
```
DecisionTrees = fitctree(train_data',train_lable,'Crossval','on');
DT_predicted_lable = predict(DecisionTrees, test_data');
```

5. Result and Discussion

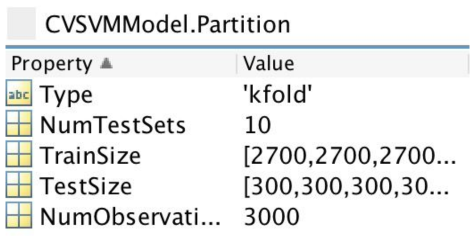
5.1 Cross Validation Model

Cross validation is a model evaluation method that is better than residuals. The problem with residual evaluations is that they do not give an indication of how well the learner will do when it is asked to make new predictions for data it has not already seen. One way to overcome this problem is to not use the entire data set when training a learner. Some of the data is removed before training begins. Then when training is done, the data that was removed can be used to test the performance of the learned model on “new” data. This is the basic idea for a whole class of model evaluation methods called *cross validation*. In our system, we tried to get the cross validation models for all classifications, but we still met some troubles to use these models to improve performance. We will figure it out in future research.

SVMModel

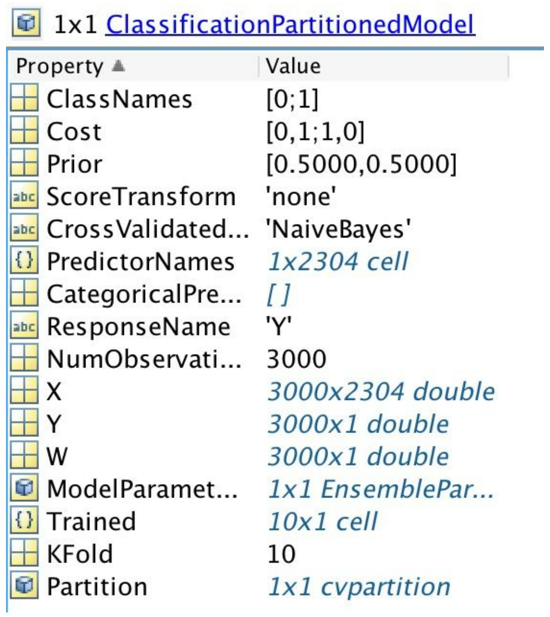

1x1 ClassificationPartitionedModel

Property	Value
ClassNames	[0;1]
Cost	[0,1;1,0]
Prior	[0.5000,0.5000]
ScoreTransform	'none'
CrossValidated...	'SVM'
PredictorNames	1x2304 cell
CategoricalPre...	[ ]
ResponseName	'Y'
NumObservati...	3000
X	3000x2304 double
Y	3000x1 double
W	3000x1 double
ModelParamet...	1x1 EnsemblePar...
Trained	10x1 cell
KFold	10
Partition	1x1 cvpartition


CVSVMModel.Partition

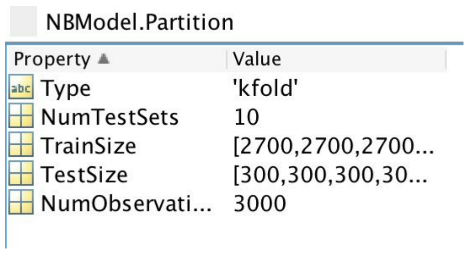
Property	Value
Type	'kfold'
NumTestSets	10
TrainSize	[2700,2700,2700...
TestSize	[300,300,300,30...
NumObservati...	3000

NBModel



1x1 ClassificationPartitionedModel

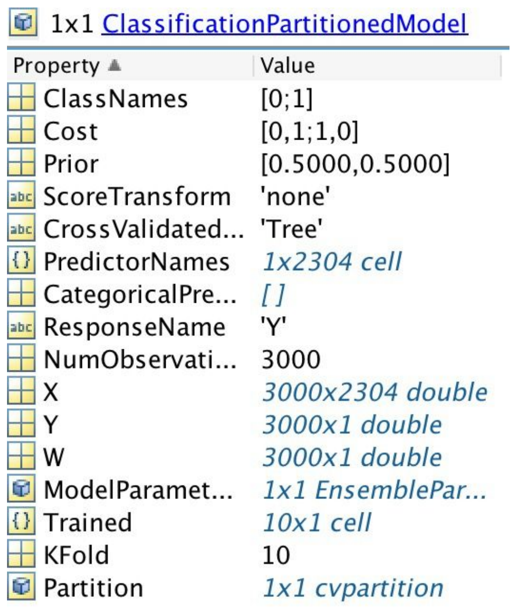
Property	Value
ClassNames	[0;1]
Cost	[0,1;1,0]
Prior	[0.5000,0.5000]
ScoreTransform	'none'
CrossValidated...	'NaiveBayes'
PredictorNames	*1x2304 cell*
CategoricalPre...	*[ ]*
ResponseName	'Y'
NumObservati...	3000
X	*3000x2304 double*
Y	*3000x1 double*
W	*3000x1 double*
ModelParamet...	*1x1 EnsemblePar...*
Trained	*10x1 cell*
KFold	10
Partition	*1x1 cvpartition*



NBModel.Partition

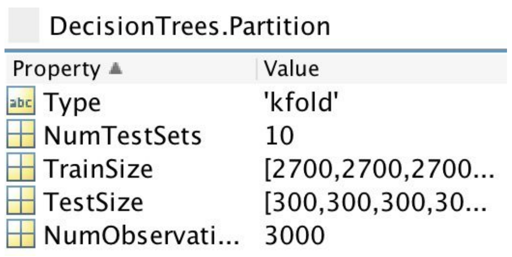
Property	Value
Type	'kfold'
NumTestSets	10
TrainSize	[2700,2700,2700...
TestSize	[300,300,300,30...
NumObservati...	3000

DTModel



1x1 ClassificationPartitionedModel

Property	Value
ClassNames	[0;1]
Cost	[0,1;1,0]
Prior	[0.5000,0.5000]
ScoreTransform	'none'
CrossValidated...	'Tree'
PredictorNames	*1x2304 cell*
CategoricalPre...	*[ ]*
ResponseName	'Y'
NumObservati...	3000
X	*3000x2304 double*
Y	*3000x1 double*
W	*3000x1 double*
ModelParamet...	*1x1 EnsemblePar...*
Trained	*10x1 cell*
KFold	10
Partition	*1x1 cvpartition*



DecisionTrees.Partition

Property	Value
Type	'kfold'
NumTestSets	10
TrainSize	[2700,2700,2700...
TestSize	[300,300,300,30...
NumObservati...	3000

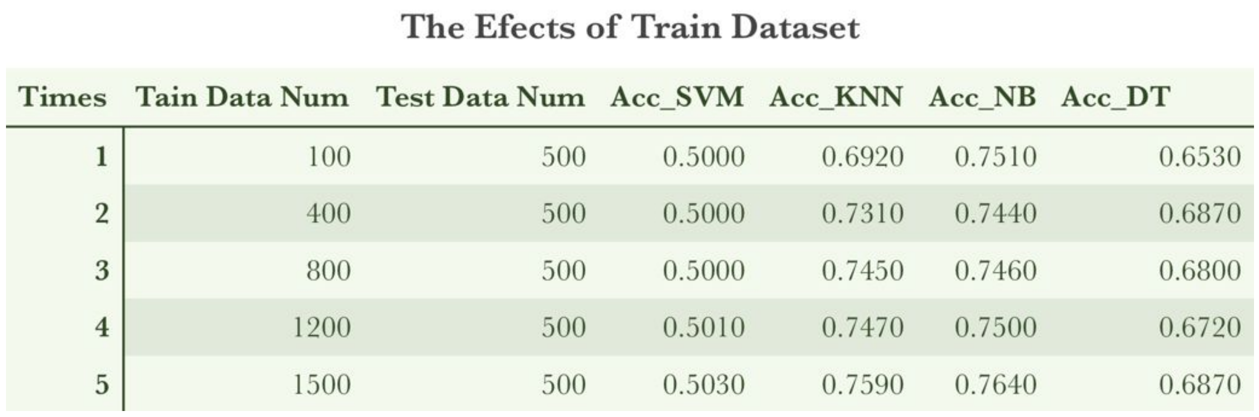
Briefly, we run the default cross validation model for svm, nb and dt classification.

Compared to get normal classification model, cross validation model is really time-consuming.

According to related documents, the cross validation model can improve performance significantly.

5.2 Choose Train Dataset

Train Dataset is very important in computer vision and machine learning research. In this part, we adjusted the scale of train dataset (still randomly pick images) to explore the relationship between classification and accuracy. The basis fixed parameters: resize images to 64*64 and set cell size as 16 to extract HoG features, set the parameters of svm to BoxConstraint=1, KernelScale=1, KernelFunction to be rbf, set the neighbors' numbers of knn as 4.

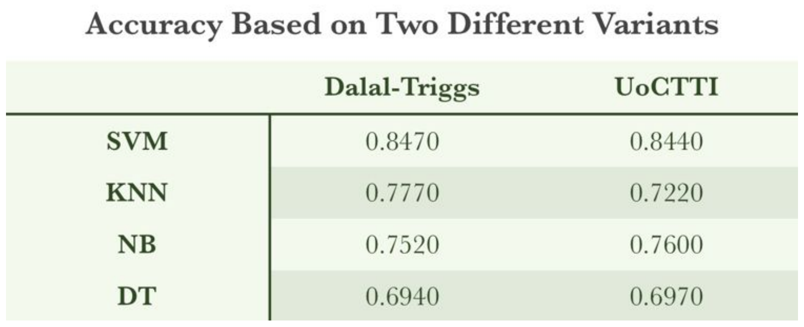

The Efects of Train Dataset

Times	Tain Data Num	Test Data Num	Acc_SVM	Acc_KNN	Acc_NB	Acc_DT
1	100	500	0.5000	0.6920	0.7510	0.6530
2	400	500	0.5000	0.7310	0.7440	0.6870
3	800	500	0.5000	0.7450	0.7460	0.6800
4	1200	500	0.5010	0.7470	0.7500	0.6720
5	1500	500	0.5030	0.7590	0.7640	0.6870

First the numbers of train and test dataset mean that we choose that number for one class. For example, the train data number is 1500, so we use a 1500-dog-image dataset and 1500-notdog-image dataset to be our train dataset. From this table, we found the change of train dataset has less impact on svm and decision tree classifications based on our settings, but the increasing of train images does improve the performance of kNN and Naive Bayes classification. As the result, we chose 5 level train dataset as our actual dataset on later research.

5.3 HoG Features and Accuracy

As we have discussed in Section 3 (Feature Detection and Extraction), VLFeat supports two varian, the UoCTTI variant and the original Dalal-Triggs variant. Our team extracted HoG by resizing the image to 256 * 256 and setting cellSize as 16, then we run our system to get the corresponding accuracy of the classifications by only changing the variant methods.



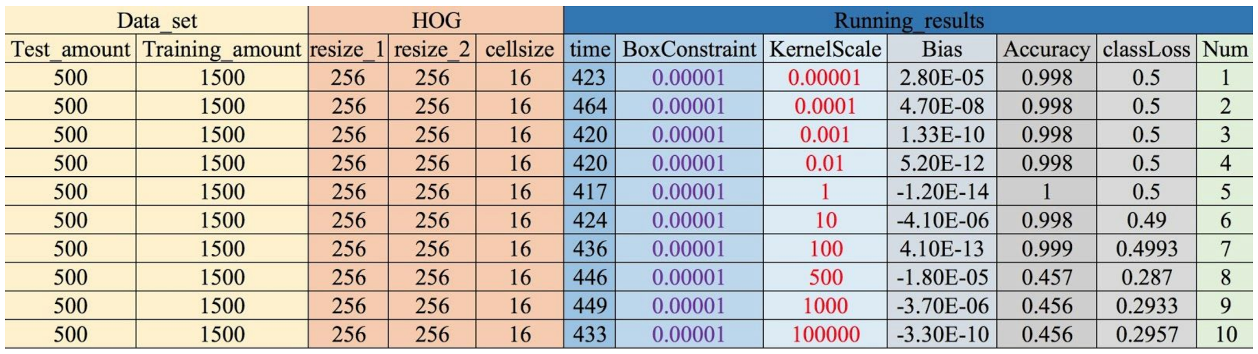
Accuracy Based on Two Different Variants

	Dalal-Triggs	UoCTTI
SVM	0.8470	0.8440
KNN	0.7770	0.7220
NB	0.7520	0.7600
DT	0.6940	0.6970

In this table, we found Dalal-Triggs showed a better performance on the aspect of accuracy. So we chose Dalal-Triggs variant for later studying.

5.4 SVM

We have discussed the influence of data set, now we will discuss the parameters that can improve the performance of svm.



Data_set		HOG			Running_results						
Test_amount	Training_amount	resize_1	resize_2	cellsize	time	BoxConstraint	KernelScale	Bias	Accuracy	classLoss	Num
500	1500	256	256	16	423	0.00001	0.00001	2.80E-05	0.998	0.5	1
500	1500	256	256	16	464	0.00001	0.0001	4.70E-08	0.998	0.5	2
500	1500	256	256	16	420	0.00001	0.001	1.33E-10	0.998	0.5	3
500	1500	256	256	16	420	0.00001	0.01	5.20E-12	0.998	0.5	4
500	1500	256	256	16	417	0.00001	1	-1.20E-14	1	0.5	5
500	1500	256	256	16	424	0.00001	10	-4.10E-06	0.998	0.49	6
500	1500	256	256	16	436	0.00001	100	4.10E-13	0.999	0.4993	7
500	1500	256	256	16	446	0.00001	500	-1.80E-05	0.457	0.287	8
500	1500	256	256	16	449	0.00001	1000	-3.70E-06	0.456	0.2933	9
500	1500	256	256	16	433	0.00001	100000	-3.30E-10	0.456	0.2957	10

KernelScale

As we can see from above table, increasing the scale of kernel first can slightly improve the accuracy, but after a point (KernelScale = 1), the accuracy is rapidly decreasing. Thus, the scale of kernel should be smaller than 5*e2.

BoxConstraint

Data_set		HOG			Running_results						
Test_amount	Training_amount	resize_1	resize_2	cellsize	time	BoxConstraint	KernelScale	Bias	Accuracy	classLoss	Num
500	1500	256	256	16	435	0.0000001	1	-5.50E-17	1	0.4993	11
500	1500	256	256	16	417	0.00001	1	-1.20E-14	1	0.5	12
500	1500	256	256	16	428	0.0001	1	-1.20E-13	1	0.5	13
500	1500	256	256	16	429	0.01	1	-1.20E-11	1	0.5	14
500	1500	256	256	16	430	1	1	-1.20E-09	1	0.5	15
500	1500	256	256	16	426	1000	1	4.10E-03	0.999	0.4997	16
500	1500	256	256	16	436	100000	1	4.10E-13	0.999	0.4997	17
500	1500	256	256	16	439	10000000	1	4.10E-13	0.999	0.4997	18

Data_set		HOG			Running_results						
Test_amount	Training_amount	resize_1	resize_2	cellsize	time	BoxConstraint	KernelScale	Bias	Accuracy	classLoss	Num
500	1500	256	256	16	420	0.00001	0.01	5.20E-12	0.998	0.5	19
500	1500	256	256	16	422	0.0001	0.01	5.20E-11	0.998	0.5	20
500	1500	256	256	16	423	0.001	0.01	5.20E-10	0.998	0.5	21

Both of tables show that BoxConstraint has few influence on the result and time using.

Data_set		HOG			Running_results						
Test_amount	Training_amount	resize_1	resize_2	cellsize	time	BoxConstraint	KernelScale	Bias	Accuracy	classLoss	Num
500	1500	256	256	64	157	Inf	108	-4.05	0.671	0.206	22
500	1500	256	256	32	103	Inf	108	-0.84	0.736	0.1713	23
500	1500	256	256	16	296	Inf	108	-0.531	0.836	0.161	24

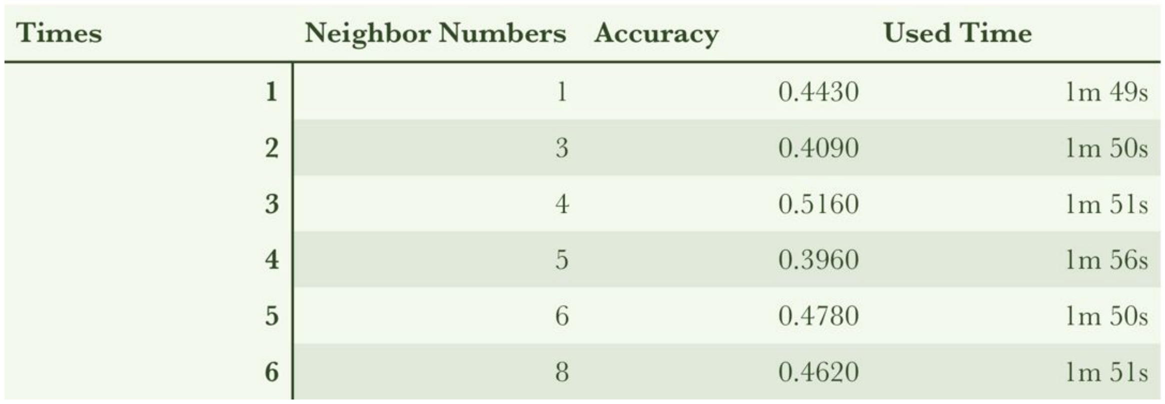
Data_set		HOG			Running_results						
Test_amount	Training_amount	resize_1	resize_2	cellsize	time	BoxConstraint	KernelScale	Bias	Accuracy	classLoss	Num
500	1500	256	256	16	430	1	1	-1.20E-09	1	0.5	31
500	1500	256	256	32	109	1	1	-7.30E-04	0.998	0.4993	32
500	1500	256	256	64	71	1	1	-1.20E-03	1	0.5	33
500	1500	256	256	128	60	1	1	-1.40E-03	0.998	0.497	34
500	1500	256	256	256	59	1	1	-3.43E-01	0.637	0.306	35

Above two table compares the different cell size whether can impact the final data or not. The answer is yes. If the kernel scale is 1*e2, the large ratio of image size and cell size do train a better model, but also use more time. But when both of kernel scale and box constraint equal to 1, the influence of cell size is not very important, but do reduce the time using if ratio is small.

5.5 KNN

In this part, we adjusted the neighbors' number to explore the relationship between KNN classification and accuracy. The basis fixed parameters: resize images to 64*256 and set cell size as 64 to extract HoG features. The reason of HoG setting is to find the optimal neighbor number by using less time. So the accuracy of classifying may not be high.

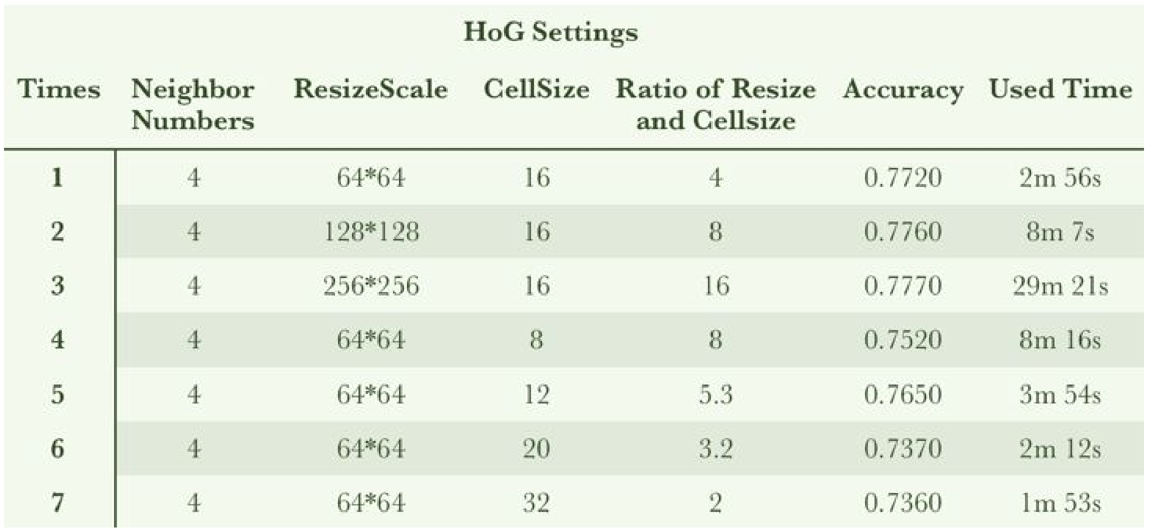
The Effects of Neighbor Number



Times	Neighbor Numbers	Accuracy	Used Time
1	1	0.4430	1m 49s
2	3	0.4090	1m 50s
3	4	0.5160	1m 51s
4	5	0.3960	1m 56s
5	6	0.4780	1m 50s
6	8	0.4620	1m 51s

In this table, it's obviously that having 4 neighbors can get higher accuracy. So the next step is to find whether the scale of resize images and cell size have influence on accuracy or not.

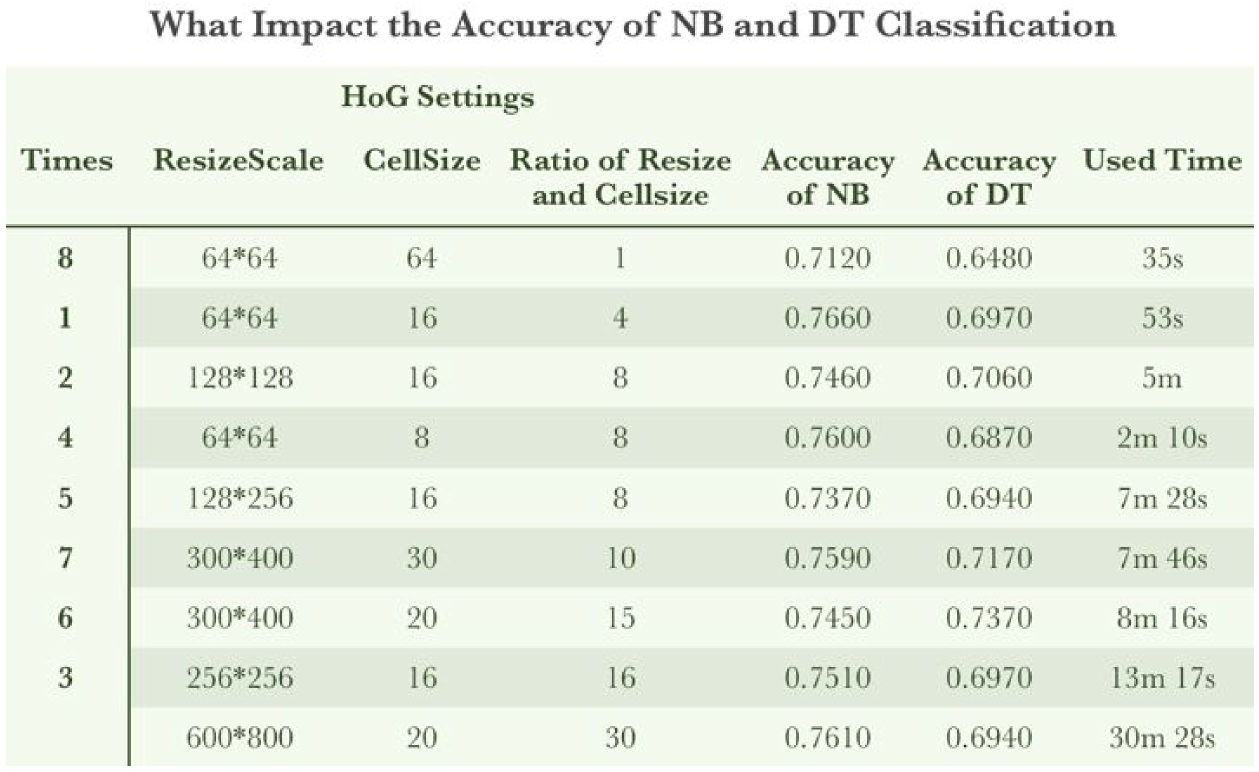
What Impact the Accuracy of KNN Classification



		HoG Settings				
Times	Neighbor Numbers	ResizeScale	CellSize	Ratio of Resize and Cellsize	Accuracy	Used Time
1	4	64*64	16	4	0.7720	2m 56s
2	4	128*128	16	8	0.7760	8m 7s
3	4	256*256	16	16	0.7770	29m 21s
4	4	64*64	8	8	0.7520	8m 16s
5	4	64*64	12	5.3	0.7650	3m 54s
6	4	64*64	20	3.2	0.7370	2m 12s
7	4	64*64	32	2	0.7360	1m 53s

This table clearly shows the ratio of resize and cellsize has directly impact on time using. The larger ratio, KNN need longer time to acquire predictions. But comparing the first three data, our time concluded that logically decreasing the ratio can keep the accuracy and reduce the time using. As the result, we choose the first data pair as our parameters for KNN classification.

5.6 NB and DT



What Impact the Accuracy of NB and DT Classification

	HoG Settings					
Times	ResizeScale	CellSize	Ratio of Resize and Cellsize	Accuracy of NB	Accuracy of DT	Used Time
8	64*64	64	1	0.7120	0.6480	35s
1	64*64	16	4	0.7660	0.6970	53s
2	128*128	16	8	0.7460	0.7060	5m
4	64*64	8	8	0.7600	0.6870	2m 10s
5	128*256	16	8	0.7370	0.6940	7m 28s
7	300*400	30	10	0.7590	0.7170	7m 46s
6	300*400	20	15	0.7450	0.7370	8m 16s
3	256*256	16	16	0.7510	0.6970	13m 17s
	600*800	20	30	0.7610	0.6940	30m 28s

By using default parameters, naive bayes classification always is more accurate than decision tree. In future research, we plan to explore how to tune the parameters of these classifications and train a better model.

6. Conclusion

In our team experiment, we use HoG and SVM for training data. The parameters of HoG are resizing image 256*256 and cell size 64, both the box constraint and kernel scale of SVM are 1. The training is based on 1500-image dataset. In this way, we only used 71 seconds to train and test, and got almost 100% accuracy.

References

Navneet Dalal, Bill Triggs "Histograms of Oriented Gradients for Human Detection." Retrieved Dec. 22, 2015 from http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf

Jeff Schneider "Cross Validation." Retrieved Dec. 22, 2015 from CMU https://www.cs.cmu.edu/~schneide/tut5/node42.html

MathWorks. Inc. "Classification." Retrieved Dec. 18, 2015 from MathWorks Documents http://www.mathworks.com/help/stats/classification.html

MathWorks. Inc. "Feature Detection and Extraction." Retrieved Dec. 18, 2015 from MathWorks Documents http://www.mathworks.com/help/vision/feature-detection-and-extraction.html

The authors of VLFeat. "HOG features" Retrieved Dec. 10, 2015 from VLFeat Tutorials http://www.vlfeat.org/overview/hog.html

Vondrick, C. (Ed.). "HOGgles: Visualizing Object Detection Features." MIT. Retrieved Dec. 10, 2015 from http://web.mit.edu/vondrick/ihog/#code

Scikit-learn developers. "RBF SVM parameters." Retrieved Dec. 10, 2015 from Scikit Learn http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html