Cats and Dogs Recognition

Chongyuan Xiang, Min Zhang, and Weixin Chen

December 5, 2013

Abstract

In this paper, we focus on the problem of classifying images of cats and dogs, which is also known as one Asirra. We use three schemes to tackle the problem. First, we use information of the whole image; second, we use information of the animal body; third, we only use features from the head. The asymptotic behavior of the classification improves when we restrict the regions to the more distinguishable ones. We obtain ??, ??, and ?? under our best choice of features and voting scheme. We anticipate that the performance of the second and third scheme will improve moderately if including more training samples.

1 Introduction

The Asirra CAPTCHA [7], proposed at ACM CCS 2007, relies on the problem of distinguishing images of cats and dogs (Asirra short for Animal Species Image Recognition for Restricting Access"). An Asirra challenge consists of 12 images, each of which is of either a cat or a dog. To solve the CAPTCHA, the user must select all the cat images, and none of the dog images. This is a task that humans are very good at. According to [7], Asirra can be solved by humans 99.6% of the time in under 30 seconds. The usability of Asirra is a significant advantage compared to text recognition based CAPTCHAs. The preference towards Asirra is due to the presumed difficulty of classifying images of cats and dogs solely by machine (evidence given by [7]). A classifier based on color features, described in [7], is only 56.9% accurate. Golle [1] suggests classification accuracy of better than 60% will be difficult without a significant advance in the state of the art". However, with a 60% accurate classifier, the probability of solving a 12-image Asirra challenge is roughly 0.2%.

2 Related Work

[1] proposes a classifier which is 82.7% accurate in telling part images of cats and dogs, which yields 10.3% odds in passing the 12-image Asirra task. Their choice of

features are elegant yet straightforward, namely color and texture. The colors are interpreted in the hue-saturation-value (hsv) model, which is close to how people perceive colors. The other choice are red-green-blue (rgb), hue-saturation-light (hsl), and hue-saturation-intensity (hsi) models. We expect the choice of different models will generally affect the classifying results to some degree. Their color feature records the existence of a pixel with in some particular color region (h, s, v) in a cell. We adopt the same approach and try some variant models. Their texture feature, using the structural approach (will be explained in the texture feature section), mainly describe the variance of the colors of the image in the rob space. They achieve an accuracy of 82.7% by combining the SVM results of the two kinds of features with 8000 training examples on 2000 testing samples. However, we believe using other perspectives, in our case gradient features, of the image instead of rgb-cast colors will improve the classification. Our first texture features follow statistical approach with similar underlying philosophy as []'s. We use the naive statistics ???????? offset energy (difference in means of the regions), entropy (difference in variances of the regions) blabla. The result is slightly worse than []. Considering the naivete and crudeness of the texture model, we anticipate a considerable ascendancy over []'s by refining our texture model. Improvement can be made by using more complicated features characterizing the texture, say the well known Histogram of Oriented Gradient (HOG), and BLP (binary local patterns). We do some test cases with training sample size ?? and testing sample size ??. The result is significantly better than [] with small sample sizes (xiangcy give example!!!!). There are three options. The first is to extract HOG, BLP of the whole image to run boosting, select a subset of most distinguishing features, and apply SVM. The second and third options ??? body and face??. The third option does a similar task as face detection. However, as commented in Zhang, Sun, and Tang [], applying the existing face detection approaches to detect the cat head is not feasible not only due to apparent greater appearance variations of cat faces compared to those of human faces, but also due to their more complicated textures. Same for dog faces. So it requires a different detection strategy. We tried the approaches in their paper. They used Haar of Histogram of Gradients as feature vector to train a detector for cat heads. The positive face examples are manually labeled out eyes and ears and normalized by aligning ears or eyes. However, normalizing the dog faces is much harder than cats because of larger variety of dog faces. Moreover, we are dealing with database with more diversity in face positions and directions. In particular, we took an approach mimicking the cascade ideas of Viola Jones [**?**]. Increasing layers has an considerable effect on reducing test errors – in [**?**], Viola and Jones trained a 37 layer decision tree to achieve over 85 percent performance, using thousands of positive examples and millions of negative examples. Due to limitation of time and machines, we use a decision tree of roughly 10 layers. Once we crop out heads from the original images, we use boost algorithm to train a classifier to classify cat heads and dog heads. This classifier will be much more accurate than classifiers directly trained from the whole

image. In this part, we use HOOGs as our feature vectors. By training 200 cat heads and 200 dog heads, we get a classifier with performance higher than 90%.

A trade off we made in our project is using Haar instead of HOG to train the cascaded decision tree. We tried on a small sample of images and found HOG is much more effective than Haar as feature vector. However, HOG is also much more intensive compared to Haar. Considering that there are tens of thousands of windows to be tested, the computation time of HOG will be unaffordable for this project, which makes us choose Haar as our feature vector. Computing the feature vectors require time?? The most interesting advantage of the Adaboost-based methods is its high accuracy in dynamical environments, achieved with high processing speed. Another interesting characteristic of the developed Adaboost-based methods is their relatively high training speed.

The usage of larger face sizes (48x48) slightly improves the performance of some methods (for example Adaboost-Rect features and Adaboost-Wav), however the training time increases exponentially (from hours to days). Shape feature, which is used in [**?**] to detect cat head, is also a first guess. It could have two usages: first, to crop out heads; second, to distinguish the dogs and cats. Although plausible at the first glance, the second usage was rejected once we realize there are cat-like dogs and dog-like cats, especially look alike in terms of shape. The first usage is not adopted in this project because it will take too long to train the classifier. As suggested by [**?**], cat ears

3 Color Feature

We first normalize all our pictures to be 250px×250px. This operation will change the ratio of width and height. However, this won't affect results since we are only interested in the color feature of the picture but not the size or shape of the animals. We divide the picture into $n \times n$ square cells with equal size.

Then we partition the color spaces. We use two models of color space. The first is RGB color space. A color in the RGB color space is described by indicating how much of the red, green, and blue is included. The color is represented as an RGB triplet (r, g, b). We normalize those three values to be between 0 and 1. The second color space is HSV color space. A color is represented as a HSV triplet (h, s, v). The three values are hue, saturation and value. Also, we normalize them to be between 0 and 1.

Both of the color spaces are three dimensional. In order to describe our algorithm, we name the three coordinates to be x, y, z. We then divide the x coordinate into N_x equal intervals, divide the y coordinate into N_y equal intervals, and divide the z coordinate into N_z equal intervals. As a result, we partition the color space into $N_x \times N_y \times N_z$ cubes.

We are now considering two features. The first feature $F_1(i, j)$ is an integer value. We go through all of the pixels inside the i-th cell of the picture, and then count

the number of pixels whose colors are in the j-th cube of the color space. ($1 \leq i \leq n^2, 1 \leq j \leq N_x \times N_y \times N_z$), and the count is $F_1(i,j)$. The second feature $F_2(i,j)$ is a boolean value. We still go through all of the pixels inside the i-th cell of the picture. If among those pixels, there exists at least one pixel with color in the j-th cube of the color space, $F_2(i,j)$ is 1(true); otherwise, $F_2(i,j)$ is 0(false).

We finally choose to go with $F_2(i,j)$ because in the training pictures, the portion

Table 1: Color Feature Boosting Training Result

Color Space Type	n	x	y	z	No. of weak learners	correction rate
RGB($x=r, y=g, z=b$)	1	10	10	10	250	60.8%
RGB($x=r, y=g, z=b$)	3	10	10	10	250	71.9%
HSV($x=h, y=s, z=v$)	1	10	10	10	250	66.8%
HSV($x=h, y=s, z=v$)	1	10	10	10	500	66.7%
HSV($x=h, y=s, z=v$)	3	10	10	10	250	72.7%
HSV($x=h, y=s, z=v$)	5	10	10	10	250	72.7%
HSV($x=h, y=s, z=v$)	5	8	8	8	250	72.1%

of the animal area varies. To classify a new picture, it makes more sense to check if the picture contains more typical dog colors or more cat colors instead of check the absolute number of pixels with typical dog colors or typical cat colors.

We use 10000 pictures to train our classifier. Half of them are dog pictures and half of them are cat pictures. Then we use our classifier to classify 2000 test pictures. Also, half of the test pictures are cats, and the other half are dogs. For each picture, we get a feature vector with length $n^2 N_x N_y N_z$. Then we run boosting algorithm with a logistic loss to train the classifiers and test on our test examples.

From Table1, we can see that the highest correction rate we achieve is 72.7 %, with the feature HSV($n=3, x=10, y=10, z=10$) or HSV($n=5, x=10, y=10, z=10$). We can also notice that feature vector generated from HSV color space is better than feature vector from RGB color space. A possible explanation is that HSV color space is more closer to human perception of color.

4 Texture Feature

Image Texture represents the information about the spatial arragement of colors or intensities within an image. For this problem, we will use MATLAB's built-in Gray-Level Co-Occurence Matrix (GLCM) to analyse a picture's texture. In a $n \times n$ Gray-Level Occurence Matrix G, $G(i,j)$ represents how often a pixel with intensity i occurs with an offset to a pixel with intensity j.

1	3	3	2	2	1
3	2	3	5	1	4
4	2	2	2	1	1
3	1	3	4	5	4
6	4	3	1	2	3
4	5	7	3	2	1

For example, the above picture shows the intensity level of each pixel in a 6-by-6 pixel images. If we set offset to be $(0, 1)$, which means the second pixel is 0 pixel down and 1 pixel left comparing to the first pixel. So a pixel with intensity 3 occurs three times with this offset to a pixel with intensity 2. So $G(3, 2) = 3$.
To get the GLCM, first, we transform the picture into a gray-level picture. Then each pixel should have its own intensity (gray-level). Then we use scaling to reduce the number of intensity values to be n, so we will get an $n \times n$ GLCM.
So if we use N_f offsets, we can get N_f GLCMs. If the o-th GLCM matrix is G_o, our first texture feature $TF_1(o, i, j)$ represents $G_o(i, j)$.
To describe the picture's GLCM more accurately, we introduce four parameters, Contrast, Correlation, Energy and Homogeneity. Contrast is a measure of the intensity contrast between two neighbor pixels; Correlation is a measure of how correlated a pixel is to its neighbor; Energy is a measure of the whole image's intensity; Homogeneity is a measure of how close the elements in the GLCM are to the diagonal elements. The formulas are:

$$\text{Contrast} = \sum_{i,j} |i - j|^2 G(i, j)$$

$$\text{Correlation} = \sum_{i,j} \frac{(i - \mu_i)(j - \mu_j)G(i, j)}{\sigma_i \sigma_j}$$

$$\text{Energy} = \sum_{i,j} G(i, j)^2$$

$$\text{Homogeneity} = \sum_{i,j} \frac{G(i, j)}{1 + |i - j|}$$

Have those four properties, we can get a new feature $TF_2(o, i)(1 \leq i \leq 4)$. $TF_2(o, 1)$, $TF_2(o, 2)$, $TF_2(o, 3)$, $TF_2(o, 4)$ denote G_o's Contrast, Correlation, Energy and Homogeneity separately.
We also use the same 10000 pictures to train our classifier. Then we use our classfier to classify the same 2000 test pictures. For each picture, feature vector TF_1 has a

length of $n^2 \times N_f$ and TF_2 has a length of $4 \times N_f$. And by $N_f = 15$, the offsets are $(i, j)(0 \leq i, j \leq 3$ & at least one of $i, j \neq 0)$. By $N_f = 24$, the offsets are $(i, j)(0 \leq i, j \leq 4$ & at least one of $i, j \neq 0)$. Then we run boosting algorithm with a logistic loss and 250 weak learners to train the classifiers and test on our test examples.
From Table2, we can see that all of the classifiers have similar correction rate. The highest correction rate we achieve is 67.1 %, with the feature TF1($n = 8, N_f = 24$) or TF1($n = 8, N_f = 15$). We can also notice that TF1 feature (using GLCM matrix directly) is better than TF2 feature (using statistics of GLCM).

Table 2: Texture Feature Boosting Training Result

Feature Type	n	N_f	correction rate
TF1	8	15	67.1%
TF1	8	24	67.1%
TF1	16	15	66.8%
TF1	16	24	66.5%
TF2	8	24	65.4%
TF2	16	24	65.4%

5 Combination of Color and Texture Feature

We need to come up with a way to combine color and texture features to get a better classifier. Assume that color feature vector has a length of N_c, the texture feature has a length of N_t. Our first approach is to concatenate the two feature vectors into a longer feature vector with a length of $N_c + N_t$. Then we will run boosting algorithm using the new feature vector.
The second approach is combining two classifiers instead of combing two feature vectors. Recall that for boosting classifier, each picture will have a voting. Assume that classfier 1 gives the picture a voting v_1, and classifier 2 gives the picture a voting v_2. Given parameter α, we calculate a new voting $v = \alpha \times v_1 + (1 - \alpha) \times v_2$. If $v \geq 0$, we classify the picture to be a cat picture; otherwise, we classify the picture to be a dog picture. We use different α to form different classifiers and use them to test on our training examples. Then we find one with the highest correction rate, and use that classifier to classify test examples to calculate correction rate.

From Table 3, we can see that both of the approaches achieve a better correction rate, and approach 2 is better than approach 1.

Table 3: Combined Feature Training Result

Two feature vectors F_1 and F_2	Correction Rate			
	F_1	F_2	approach1	approach2
F_1 = RGB($n = 1, x = 10, y = 10, z = 10$), F_2 = TF1($n = 16, N_f = 24$)	71.9%	66.5%	73.1%	73.8 %
F_1 = HSV($n = 3, x = 10, y = 10, z = 10$), F_2 = TF1($n = 8, N_f = 24$)	72.7%	67.1%	73.6%	74.5 %

6 Attack on Head Detection and Classification

In this section, we will introduce some work we have done in cat and dog head detection and classification. Strictly speaking, we have not completely solve the problem due to time and computation limitations, but we did find this a possible approach for the original problem. Our goal is to build a head detector to find cat and dog head and then focus our attention on the head images to get a high performance classifier. The first step is to train a classifier to distinguish head and background images, using boost algorithms. We tried two kinds of feature vectors here: rectangular features and Haar of histogram oriented gradients and compared their performance. The second step is to build a 'cascade' of classifiers to enhance the performance, especially reducing the false positive rate. Then, we simply search among all sub-windows in all scales, locations and directions in the image and using the cascade detector to test if the sub-window is a head. Notice that there would be thousands of sub-windows in a single image, thus rapid processing speed and extremely low false positive rate are two key points here. This is also the advantages of boost algorithms and the cascade technique – boosting helps us select a small amount of 'significant' features from a large amount of ones(usually over $10,000$) and cascade leads to low false positive rate. Finally, we train a classifier to classify cat heads and dog heads, using boost algorithms.

6.1 Training Head Detector

We have two sets of feature vectors to train our head detector- rectangular features, and HOG like features, for example HOG, EOH, and our choice is the combination of Haar and oriented gradients, known as HOOGs. [**?**]. While we generally believe the second sets of features yield higher accuracy in detecting the head, they are much more time consuming to compute feature vectors???? and running boosting. ??

6.1.1 Rectangular Features

The following figure comes from [**?**], which demonstrates the canonical two-rectangle features, three-rectangle feature, and a four-rectangle feature, more or less capturing all rectangular features. We employ their technique of image representation, *integral*

image, to allows the features be computed very quickly. The same technique might be applied to the computation of HOOG features as well, but with limitation due to existence of multiple bins and renormalization procedure of HOOGs. So we do not use integral image to generate HOOGs.

XIANGCY add figure 1 in viola and jones.

6.1.2 HOOG

For image gradient $\vec{g}(x)$, given K orientation bins, oriented gradients channel $g_0^k(x)$, $k = 1, \cdots, K$, is defined as $|\vec{g}(x)|$ if the orientation of $\vec{g}(x)$ agrees with k. Our two inspecting HOOG features are defined as followed. The in-channel feature and cross-channel feature are defined as

$$HOOG^i(R_1, R_2, k) = \frac{S^k(R_1) - S^k(R_2)}{S^k(R_1) + S^k(R_2)}, \ HOOG^c(R_1, R_2, k) = \frac{S^k(R_1) - S^{k+K/2}(R_2)}{S^k(R_1) + S^{k+K/2}(R_2)}.$$

6.1.3 Boost Algorithms

We manually cropped out 466 cat heads and 380 dog heads and generated 26000 background images. We tried both adaBoost and logitBoost. Result shows logitBoost is usually better than adaBoost, and such performance difference will be enlarged in the cascade process.

some result

Analysis of result. Not satisfying possibly because not aligned and normalized.

6.2 Cascade of Classifier

The idea is similar to a decision tree. At each layer of the cascade is a classifier and an example will be rejected immediately if it is rejected by one layer. When training the classifier, at each layer, we use boosting algorithms to select important features and find a reasonable threshold that almost all positive training examples will be left and a decent amount of negative training examples will be rejected. This way, negative examples will be rejected quickly while only a small number of positive examples will be misclassified. For the same reason, a large amount of negative examples are required. In [**?**], Viola and Jones trained 9544 face images and 350 million background images to get a 37 layers of cascade classifier. In our project, we trained 400 cat heads, 400 dog heads, and 20 thousand background images with a 15 layers of cascade. We arrived at 4% false positive rate and 1% false negative rate.

some result

6.3 Classification

7 Results

8 Conclusion

References

[1]

[2]