Horse Detection Using Haar Like Features

M. S. Uddin and A. Y. Akhi

Abstract—**A commonly used approach for detecting objects is based on the techniques of "boosting" and "cascading", which allow for real-time detection. In this paper I have developed a classifier for detecting horses from images or from real time video sources. For that *purpose* the Haar-like features were used to discriminate horses. Those features were used as input in a learning algorithm, based on AdaBoost, which selects a small number of critical visual features from a larger set and yields an extremely efficient classifier.**

Index Terms—**Horse detection, object detection, haar-like features, adaboost.**

I. Introduction

Object detection is an important element of various computer vision areas. The basic goal is to find an object of a predefined class in static images or video frames. Sometimes this task can be handled by extracting certain image features, such as edges, color regions, textures, contours, etc. Afterwards, some heuristics is applied to find configurations and/or combinations of those features characteristics of the object that one wants to detect. But, for complex objects, such as horses, it is hard to find features. Thus, horse detection in cluttered environment is an open problem. The major difficulties are:

- The size, color and breeds are different (see Fig. 6)
- A Horse is a non-rigid body. In other words, the shape and size of a horse varies greatly, and therefore the model of a horse is much more complex than that of rigid objects.
- Illumination and weather conditions vary greatly.

Another possible approach is to use the statistical models (classifiers). These models can be obtained by analyzing a set of training images, which will then be used to detect the horses. Statistical model-based training takes multiple instances of horses and multiple "negative" samples, i.e., images that do not contain horses. Different features are extracted from the training samples and distinctive features, that can classify the horses, are selected [1]. The Haar-like features (so called because they are computed similarly to the coefficients of Haar wavelet transforms) and a large set of very simple "weak" classifiers, that use a single feature to classify the image as horse or without horse, were used to extract the features characteristics of the horses.

Manuscript received October 29, 2014; revised May 26, 2015.

Mohammad Salah Uddin is with Dipartimento di Ingegneria Informatica Automatica e Gestionale Antonio Ruberti, Sapienza Universit`a di Roma, Rome, Italy (e-mail: uddin@dis.uniroma1.it).

Afroza Yesmin Akhi is with the Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh (e-mail: ankhi.ayaz@yahoo.com).

An approach to this technique was originally developed by Viola and Jones [2] and then analyzed and extended by Lienhart *et al.* [3], [4]. In one image sub-window, the total number of Haar like features is very large, far larger than the number of pixels. In order to ensure a fast classification, the learning process must exclude a large majority of the available features, and focus on a small set of critical features. A similar methodology combining Haar-like features and the AdaBoost algorithm, proposed by Viola *et al.* to detect faces, is proposed here to detect horses.

II. Object Detection

The object detection system uses a strong classifier to determine if sub-windows in an image contain a specified object. The strong classifier is composed of a set of weak learners with associated weights. Each weak learner uses a single image feature to produce a hypothesis. Viola and Jones show that AdaBoost can be used to both select a small subset of features and train the classifiers [2]. There are four steps to building an object detection system with AdaBoost: select a dataset with positive and negative training examples, train the threshold values for each feature, select and train a subset of the classifiers and train the attentional cascade. Once the detector is built, images are exhaustively scanned at all locations and scales to identify objects.

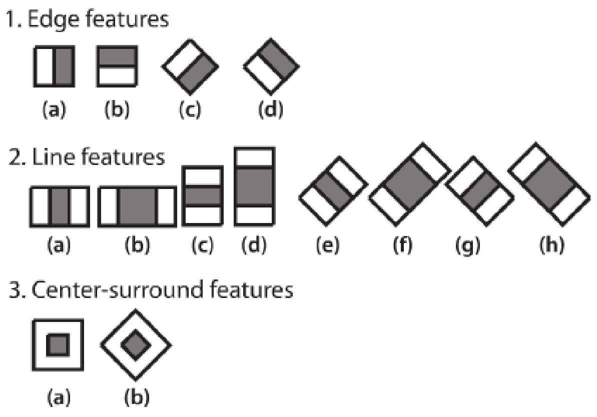


Fig. 1. Haar-like feature.

A. Image Features

Each feature is represented by a template (shape of the feature), its coordinate relative to the search window origin and the size of the feature (its scale). A subset of the features prototypes used is shown in Fig. 1.

Each feature is composed of two or three "black" and "white" rectangles joined together - these rectangles can be upright or rotated by 45 degrees. The Haar-like features value is calculated as a weighted sum of two components: the pixel gray level values sum over the black rectangle and the sum over the whole feature area (all black and white areas). The weights of these two components are of opposite signs and for normalization purpose, their absolute values are inversely proportional to the areas.

B. Integral Image

An integral image representation is used to provide constant lookup times for the sum of pixels in rectangular regions of an image. This representation enables rectangular features to be computed using a minimal number of array references. The value of an integral image at location *x*, *y* contains the sum of the pixels above and to the left of *x*, *y*.

The value of an integral image at location *x*, *y* is computed as:

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x'y') \tag{1}$$

where *i*(*x*, *y*) is the original pixel value and *ii*(*x*, *y*) is the integral image [2]. Viola and Jones show that the following recurrences can be used to efficiently compute the integral image in a single pass over the original image:

$$s\ (x\ ,y) = s(x, y-1) + i\ (x,y)$$
$$ii\ (x\ ,y) = ii(x-1, y) + s\ (x,y)$$

where *s*(*x*, *y*) is the cumulative row sum and *s*(*x*, 0) = 0 and *ii*(*0*, *y*) = 0. The top left corner of an integral image is shown in Fig. 2. The variable $P_{x,y}$ refers to the pixel value of an image at location *x*, *y*. Viola and Jones show that the sum of pixels in a rectangular region can be found in four array references. The value of *D* in Fig. 3 can be computed as 4 + 1 – (2 + 3). Therefore, the time required to compute a feature is not dependent on its size.

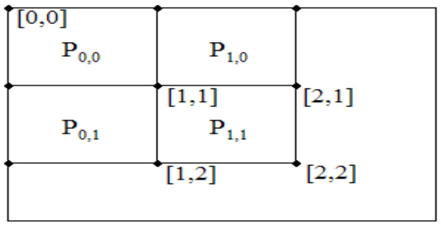


Fig. 2. Encoding integral image.

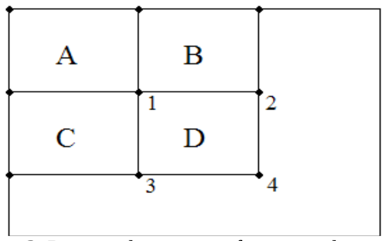


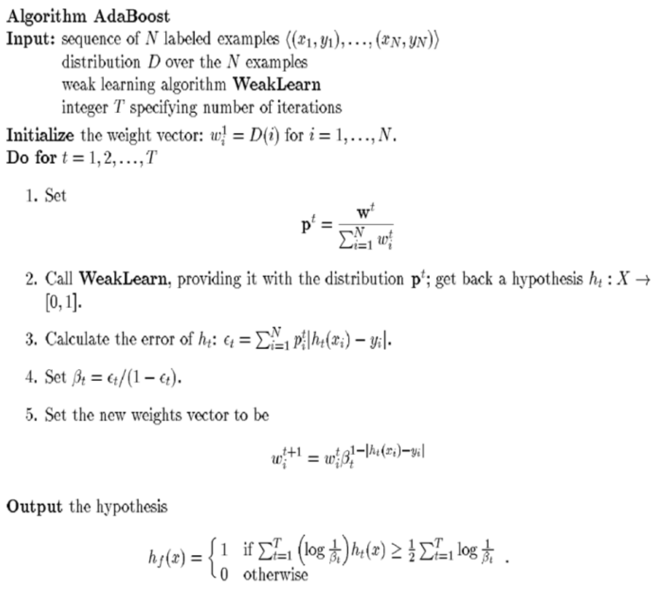
Fig. 3. Rectangular regions of an integral image.

C. Weak Classifier

The object detection system uses weak learners constrained to evaluating a single feature. For each feature, the weak learner determines the optimal threshold classification function, such that the minimum number of examples are misclassified [2]. A weak learner consists of a feature, f_j, and a threshold, θ_j:

$$h_j(x) = \begin{cases} 1,\ if & h_j(x) < \ \theta j \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The best weak learner has a misclassification rate of approximately 0.07. The strong classifier combines several weak learners to produce more accurate hypotheses.



```
Algorithm AdaBoost
Input: sequence of $N$ labeled examples $\langle(x_1, y_1), \ldots, (x_N, y_N)\rangle$
       distribution $D$ over the $N$ examples
       weak learning algorithm WeakLearn
       integer $T$ specifying number of iterations
Initialize the weight vector: $w_i^1 = D(i)$ for $i = 1, \ldots, N$.
Do for $t = 1, 2, \ldots, T$
  1. Set
       $\mathbf{p}^t = \frac{\mathbf{w}^t}{\sum_{i=1}^N w_i^t}$
  2. Call WeakLearn, providing it with the distribution $\mathbf{p}^t$; get back a hypothesis $h_t : X \to [0, 1]$.
  3. Calculate the error of $h_t$: $\epsilon_t = \sum_{i=1}^N p_i^t |h_t(x_i) - y_i|$.
  4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$.
  5. Set the new weights vector to be
       $w_i^{t+1} = w_i^t \beta_t^{1-|h_t(x_i)-y_i|}$
Output the hypothesis
  $h_f(x) = \begin{cases} 1 & \text{if } \sum_{t=1}^T \left(\log \frac{1}{\beta_t}\right) h_t(x) \geq \frac{1}{2} \sum_{t=1}^T \log \frac{1}{\beta_t} \\ 0 & \text{otherwise} \end{cases}$ .
```

Fig. 4. Pseudo-code for the adaboost algorithm. adapted from [2].

D. Classifier Training

AdaBoost is used to find the best weak learners and the corresponding weights for these classifiers. The boosting algorithm maximizes the margin between a set of positive and negative examples. Pseudo code for the boosting algorithm is shown in Fig. 4. The algorithm is first given a set of positive and negative examples. Each of the examples is converted to gray-scale, scaled to the base resolution of the detector and annotated with a 1 or 0 for positive and negative examples respectively. Next, the algorithm creates a weight vector for the examples. The initial weights are dependent on the number of positive and negative examples. If the number of positive and negative examples is equal, then the algorithm starts with a uniform weight vector.

The boosting algorithm performs a series of trials, each time selecting a new weak learner. At the beginning of each trial, the weights are normalized to sum to 1. Next, the algorithm selects the weak learner that produces the smallest misclassification error with respect to the weight vector. This step requires classifying all of the examples with the over 270,000 weak learners and is computationally expensive. Fortunately, the threshold value for each weak learner needs to be computed only once, because the hypothesis of a weak learner does not consider the weight vector. The best weak learner is then selected and used to update the weight vectors. The weights of correctly classified examples are multiplied by *β*, while the weights of misclassified examples do not change. Combined with normalization, this update results in most of the weight being placed on hard to classify examples. Therefore, as the number of trials increases, the error rates also increase. This leads to smaller α values for weak learners selected later in the training process. The final classification function is the sum of the predictions of the selected weak learners multiplied by the corresponding *α* values.

E. Attentional Cascade

The object detection system exhaustively scans all sub-windows in an image. Evaluating all sub-windows becomes intractable when the strong classifier selects several thousand features. A method to overcome this problem is the use of a degenerate decision tree to limit the number of features computed for each sub-window. Several filters are used to build decision nodes for the tree. If any filter in the tree rejects a sub-window, then the sub-window if rejected. The structure of the decision tree is shown in Fig. 5. The decision tree is constructed such that the first filter evaluates a small number of features and the later filters add more and more features. The threshold values of features in each filter must be modified to avoid discarding positive sub-windows. This is achieved by increasing the threshold value, but this process leads to higher false positive rates. Therefore, the purpose of each filter in the decision tree is to progressively discard the harder to classify false positives. A well-constructed decision tree significantly reduces the number of features evaluated for each sub-window while maintaining accuracy close to the exhaustive approach.

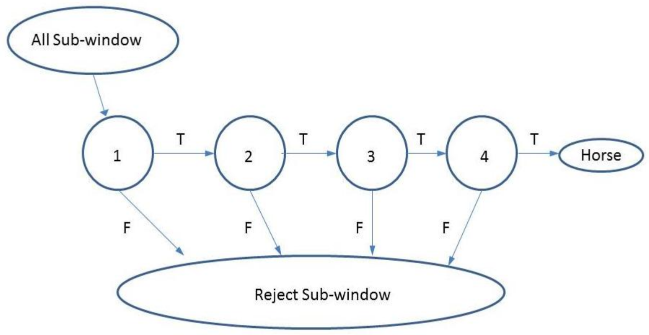


Fig. 5. Attentional cascade. adapted from [2].

F. *Scanning for Objects*

The system detects objects by exhaustively scanning images. For each input image, the detector first converts the image to gray-scale and computes the integral image. Next, the detector starts with an initial scale of 1.0 and evaluates every sub-window with the strong classifier. The scale is then increased and all sub-windows are evaluated at the new scale. The detector efficiently computes features at different scales by scaling the features, not the image. The scale is increased until the detection window is larger than the image width or height. A sub-window is marked as an object occurrence if the strong classifier returns a value of 1.

III. Building Horse Detector

In this section it is described an algorithm for constructing a cascade of classifiers [2] which achieves increased detection performance while radically reducing the computation time. The key insight is that smaller, and therefore more efficient, boosted classifiers which reject many of the negative sub-windows while detecting almost all positive instances, can be constructed. Simpler classifiers are used to reject the majority of sub-windows before more complex classifiers are called upon, in order to achieve low false positive rates.

A cascade of classifiers is a degenerated decision tree where, at each stage, a classifier is trained to detect almost all objects of interest (horses or other objects) while rejecting a certain fraction of the non-object patterns [2] (see Fig. 5).

A. *OpenCV Software*

An open-source computer vision library called OpenCV was used to train the classifiers. OpenCV implements Viola and Jones' [2] original boosted cascade algorithm, but includes an option to use the extended Haar feature set of Lienhart and Maydt [4]. The software works on Windows, Mac OS X and Linux, and comprises a series of command line programs with many parameters that can be specified by the user. It is written entirely in C/C++ and can be edited by anyone.

TABLE I: Number of Training Examples Used for Our Classifiers

	Horses	Source
Positive images	650	INRIA Horses Database and Google.com
Negative images	900	

Fig. 6. Example of horse images used for training.

B. *The Training Set*

The detection algorithm requires two set of training images; a positive set containing the object of interest (horses), and a negative (or 'backgrounds') set. For our training set we used INRIA Horses Database [6] plus collect horse images from google.com. Table I shows the number of positive and negative examples used for our classifiers. Some examples of positive training images are shown in Fig. 6.

There are many challenges in object recognition, such as variations in viewpoint, illumination, scale as well as interclass variation. For this reason, the training set should include many images of the object (generally thousands), which capture all possible sources of variation. The training set should be well chosen; otherwise it confuses the learning algorithm.

In principle, negative samples can be arbitrary images which do not contain the object of interest (such as horse). Initially, we used images from INRIA Horses and google.com. We choose negative set very carefully and assure that not to include positive examples in the negative set. Since the actual choice of negative images does not matter, as long as they don't contain the object of interest.

C. *Preparing the Images for Training*

Viola and Jones' algorithm is a supervised learning algorithm, so the computer must be 'told' which images are positive and which are negative. For the negative examples, all that is required is a text file containing the locations of the image files. For positive examples, the objects must be manually segmented from the background, and their locations and their locations within the image recorded in a text file with the following format:

```
<path>image_name_1 count1 x11 y11 w11 h11 x12 y12 w12
h12 ... <path>image_name_2 count2 x21 y21 w21 h21 x22 y22
w22 h22 ...
```

Each line contains the full filename of the image, followed by a count of how many objects are in the image, along with a list of rectangles containing their top-left coordinates, width and height in pixels. An open-source program called ObjectMarker.exe was used to draw bounding rectangles around the object(s) in each image, automatically creating a file in the above format, later to be read by OpenCV. It took many hours to go through several thousands of images, drawing bounding boxes around the objects.

Before being presented to the learning algorithm, the positive samples must be reduced to a standard size. For horses we used a size of 72 $\times$ 48 pixels. The reasoning is that the width of horse is more than height. The OpenCV program CreateSamples.exe was used to size normalize the positive images and compress them into a vector file.

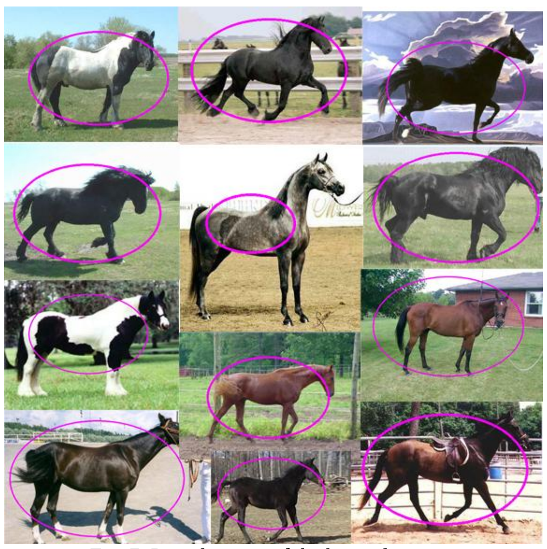


Fig. 7. Partial output of the horse detector.

D. *Training a Cascade of Classifiers*

The cascade training process involves two types of tradeoffs. In most cases, the classifiers with the most features will achieve higher detection rates and lower false positive rates. At the same time classifiers with more features require more time to compute. In principle one could define an optimization framework in which: i) the number of classifier stages, ii) the number of features in each stage, and iii) the threshold of each stage, are traded off in order to minimize the expected number of evaluated features. Unfortunately finding this optimum is a tremendously difficult problem [2].

In practice a very simple framework is used to produce an effective classifier which is highly efficient. Each stage in the cascade reduces the false positive rate as well as the detection rate. A target is selected for the minimum reduction in false positives and the maximum decrease in detection. Each stage is trained by adding features until the target detection and false positives rates are met (these rates are determined by testing the detector on a validation set). Stages are added until the overall target for false positive and detection rate is met. The complete horse detection cascade has 18 stages with 4769266 features.

IV. Experimental Results

The classifier described in the paper was tested by using image sequences with horses in different size, and breeds. This set consists of 200 images. The system achieves a horse detection rate of **63%** with 74 false positives. Some examples of the Horse detection are presented in the Fig. 7. The output of the classifier is represented by the pink eclipses, meaning that in the search sub-window corresponding to the pink eclipse the output of the cascade of the classifier was true, in other words, it has detected a horse. Note that, the detector can only detect the side view horses.

V. Conclusion

In this paper we introduced an algorithm for detecting horses, based on Haar like features. The detection rate is 62% which is further improved by increasing the test images. Due to the shortest of time we are unable to train our cascade over more training images, which improved the performance of our detector.

References

[1] G. Bradski, A. Kaehler, and V. Pisarevsky, "Learning-based computer vision with intel's open source computer vision library," *Intel Technologies Journal*, vol. 9, iss. 2, 2005.

[2] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," presented at IEEE Conference on Computer Vision and Pattern Recognition, 2001.

[3] R. L. A. Kuranov and V. Pisarevsky, "An empirical analysis of boosting algorithms for rapid objects with an extended set of Haar-like features," Intel Technical Report MRL-TR-July 02-01, 2002.

[4] R. Lienhart and J. Maydt, "An extended set of Haar-like features for rapid object detection," presented at the IEEE International Conference on Image Processing, 2002.

[5] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Proc. 13th International Conference on Machine Learning*, 1996.

[6] INRIA Horses. A dataset for object class detection. [Online]. Available: http://www.lear.inrialpes.fr/data

Mohammad Salah Uddin is a PhD student in Dipartimento di Ingegneria Informatica Automatica e Gestionale Antonio Ruberti, Sapienza Universit`a di Roma, Rome, Italy. He previously worked as a lecturer in the Department of Computer Science and Engineering, Central Women's University, Dhaka, Bangladesh. He published several journal and international conference papers from his research work. He received his B.Sc. degree in computer science and engineering from East West University, Dhaka, Bangladesh in 2012. He is interested in computer vision, human computer interaction (HCI), web service composition, semantic web service, and mobile apps.

Afroza Yesmin Akhi is a B.Sc. student in the Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh. She is a member of Web Engineering Research Group, East West University. She is currently working towards web application development. She is also interested in mobile apps.