NUS CS4243: Computer Vision and Pattern Recognition November 6, 2019

Where's Our Waldos?

Lecturer: Angela Yao
Students: Joshua Lee Kai Sheng, Tian RunXin, Yoon Jia Jun Ken

Abstract

Object detection and image classification are two of the key challenges in computer vision which play a crucial role in developing Artificial Intelligence (AI) technologies, such as self-driving cars, in this increasingly automated age. This paper explores the use of an ensemble of cascading Viola-Jones detection framework and Histogram of Oriented Gradients (HOG) template matching to solve the "Where's Waldo" problem. The models that we have trained obtained a mean average precision (mAP) score of 0.294 using `ovthres` = 0.3.

1 Introduction

The "Where's Waldo" problem statement requires us to find Waldo, Wenda and Wizard in a series of Wimmelbilder books featuring them. The term characters will now be used exclusively to refer to these three humans of interest. The books feature images that contain the characters (often only Waldo) amidst a chaotic background with many other non-relevant humans. Hence, this problem can be dissected into two components: Locating humans (Object detection) and specifying the character (Image classification).

There are several common methods for object detection, generally classified into machine learning methods and deep learning methods. In this paper, we focus on non-deep, machine learning methods, and first seek to provide a comparison of such methods. Some of the popular methods that we have explored in this study include Viola–Jones object detection framework based on Haar features [3], histogram of oriented gradients (HOG) features with SVM [1] and template matching. Each of these methods studied has its downsides, which are listed below.

For the Viola–Jones object detection framework, the main weakness is that windows, once identified as false negatives, are not recoverable in subsequent stages of the cascade classifier. This results in objects of interest going undetected. Besides, Haar features used by each weak classifier may not be descriptive enough to help accurately detect objects of interest due to their over-simplicity. Furthermore, training takes time and many training examples are required for the training process.

For HOG features with SVM, the main downside is that it is extremely difficult to fine tune the hyperparameters (C etc.) and select the best kernel function. Similar to Viola–Jones object detection framework, HOG features with SVM requires many training examples and much time for training. Moreover, the actual detection process usually takes longer time than Viola–Jones object detection framework.

As for basic template matching using pixel intensities, it is highly sensitive to the variance of true object appearance due to transformations, differences in luminosity etc. Also, the actual detection process takes an extremely long time due to the brute force nature of this method.

In light of the weaknesses of the different methods, we combine them into an ensemble of methods to create a more robust object detector. Ensemble learning is proven to obtain better performance as the individual methods complement one another's weaknesses owing to diversity among the individual methods [2].

We propose an ensemble of methods solution by cascading Viola–Jones object detection framework with HOG template matching. The Viola–Jones object detection framework will first output multiple candidate windows, which potentially consist of false positives. Thereafter, HOG template matching will do further detections on these windows to filter the false positives away.

We are able to achieve decent detection results on the test image set, with a mean average precision (mAP) score of 0.294 using `ovthres` = 0.3.

2 Proposed Solution

We propose an innovative solution involving an ensemble of methods by cascading Viola–Jones object detection framework with HOG template matching shown in Figure 1.

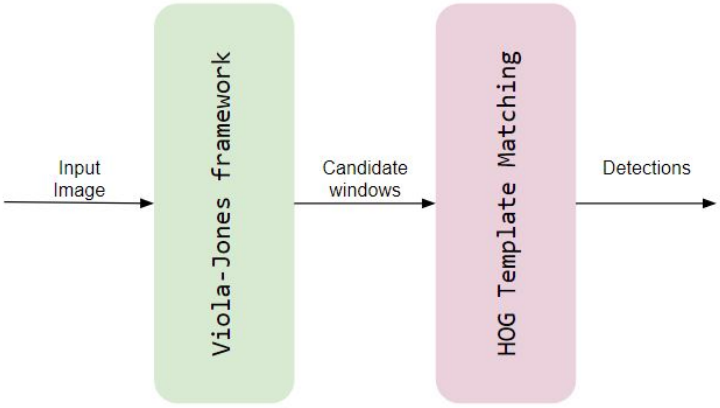

Figure 1: Diagram illustrating proposed framework.

This two-stage framework seeks to solves both components of the problem statement. In the first stage, the Viola–Jones object detection framework will output multiple candidate windows, which potentially consist of false positives. In the second stage, the HOG template matching will do further detections on these windows to filter out and eliminate the false positives, leaving only characters detected.

2.1 Technical details about Viola-Jones object detection framework

Viola-Jones object detection framework is a popular real time object detection framework prior to the advent of deep methods. It involves four main stages: Haar feature selection, integral image creation, AdaBoost training and cascade of classifiers creation.

Stage 1: Haar features selection
Haar features are used as the feature for object detection in this framework. Haar features are rectangular features which comprise of differences of sums of intensity of regions in the rectangle, i.e. $feature\ value = \Sigma(pixel\ values\ in\ white\ area) - \Sigma(pixel\ values\ in\ black\ area)$. Examples of Haar features are shown in Figure 2 below.

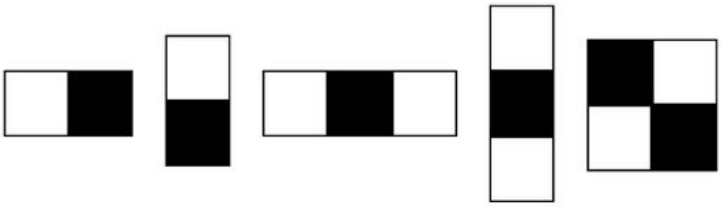

Figure 2: Examples of Haar features.

Objects of interest usually have similarities that can be described using Haar features. For example, in the case of Waldo, the hat has alternating dark and bright regions and the eye region is darker than the forehead. In our application, we use a set of upright Haar features.

Stage 2: Integral image creation
In order to speed up computation of Haar features, a prefix sum matrix, a.k.a. integral image, is used. Prior to detection, an integral image is pre-computed where each pixel is the sum of all original pixel values above and left to it, inclusive. Using the principle of inclusion and exclusion, the prefix sum matrix allows Haar features to be computed in constant time. For example, referring to Figure 3 below, the sum over the orange rectangle can be easily computed by $sum = A - B - C + D$. This gives the framework a boost in detection speed.

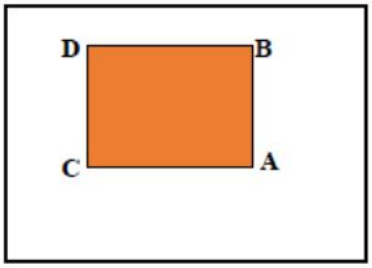

Figure 3: Usage of integral image

Stage 3: AdaBoost training
In a detection window, the number of possible Haar features is exponential in the size of the window. For example, in a 24×24 detection window, the number of possible Haar features is in the order of 10^5. As such, it would be prohibitively expensive to try all of them when training a classifier in every stage of the cascade.

In light of this, AdaBoost is employed to train each classifier. This algorithm iteratively selects the best classifier with the least weighted error at that moment, and reweighs all wrongly classified training examples by the classifier. In the end, the algorithm aggregates all the classifiers selected as a linear combination of weak classifiers to give a strong classifier. The weight of each weak classifier is proportional to its accuracy.

The final strong classifier classifies the detection window based on the formula:

$$h(x) = \sum_{j=1}^{M} \alpha_j h_j(x) \tag{1}$$

where $h(x)$ is the final classification, α_j is the weak classifier weight, $h_j(x)$ is the classification by the weak classifier.

Using AdaBoost, we are able to speed up the training process, while achieving a robust strong classifier for each stage of the cascade.

Stage 4: Cascade of classifiers creation
The Viola-Jones framework combines a number of strong classifiers as a detection cascade, where each stage consists of a strong classifier. On average only 0.01% of the detection windows contain objects of interest. In order to speed up detection, we have to skip through firm negative windows quickly and spend more time on potential positive windows.

In order to do so, earlier classifiers are less complicated and faster to compute, with a higher maximum false positive rate. Since a window is immediately discarded if it fails any stage, faster earlier classifiers can help weed out most negative windows quickly, without wasting time on them.

Thereafter, the cascade gradually consists of more complex classifiers, with lower maximum false positive rate, which take more time for classification. These later classifiers can further weed out the false positives that pass through earlier stages, further enhancing detection accuracy. Since most negative examples do not reach the later stages, these slow classifiers will not affect the detection time significantly.

Furthermore, in order to learn from mistakes made by previous stages, false positives from earlier stages are reused as negative training examples for later stages.

2.2 Technical details about HOG template matching

HOG feature is a type of robust feature that counts the occurrences of gradient orientations in a detection window. The idea behind this feature descriptor is that local object appearance can be effectively described by a distribution of intensity gradients.

The detection window is divided into blocks and smaller cells. For each cell, a histogram of gradient orientations are computed. The cell histograms are normalized across adjacent blocks to achieve better invariance to illumination and geometric transformations. The final HOG descriptor is obtained by concatenating all these histograms.

In using HOG for template matching, we first select a few good representative images of the objects of interest and compute the HOG for these images. The computed HOGs are then used as the templates for template matching.

In template matching, the detection window and HOG template are resized so that they have the same size. We then compare each detection window with the HOG templates using correlation distance, which has the formula:

$$dCor(u, v) = 1 - \frac{(u - \overline{u}) \cdot (v - \overline{v})}{\|(u - \overline{u})\|_2 \|(v - \overline{v})\|_2} \tag{2}$$

where $\overline{u}$ and $\overline{v}$ are the means of elements of u and v; and $x \cdot y$ is the dot product between x and y.

As any correlation distance is in the range of [0, 1], the score of each detection is computed by the formula:

$$score = 1 - correlation\ distance \tag{3}$$

Since our goal in this second stage is to remove false positives, we impose a threshold on the score of each detection window. Only windows with a score higher than the threshold will be outputted to the baseline as positive detections. These windows are the ones which display some correlation and hence similarity with the templates.

2.3 Ensemble of methods

2.3.1 Ensemble Pipeline

In the ensemble of methods, we first run Viola-Jones detector over the full input images. The Viola-Jones detector is able to filter away most true negative windows and outputs candidate windows that can potentially be true positives and false positives.

In order to further filter away the false positives, which will significantly affect mAP evaluation, we input these candidate windows into a HOG template matcher. The template matcher will loop through each of these candidate windows and match them with a set of chosen HOG templates and score them using Equation 3. To remove firm false positives, we impose a threshold of 0.25 on these candidate window scores. Only windows with scores higher than the threshold will be output as true detections.

2.3.2 Ensemble Implementation

To combine both methods, we created an `ensemble_detector` script. The implementation of the script is pretty straightforward. There are two stages: Viola-Jones objection detection framework and HOG template matching.

Stage 1: Viola-Jones objection detection framework
For Viola-Jones objection detection framework, there are two main aspects for implementation: training and detection. We used `openCV`'s `cv::CascadeClassifier` in both stages.

For training, we trained a custom Viola-Jones cascade classifier for our purpose. This is because the general cascade classifiers such as the `haarcascade_frontalface_alt.xml` provided by `openCV` do not fit our specific application. We researched heavily on the training process for the cascade classifier such as the theory and hyperparameters for training. The training process is detailed further in Section 3.2.

For detection, we used the existing `openCV`'s implementation of the `cv::CascadeClassifier`. We use a `scaleFactor` that varies according to the input image size, which is detailed in Table 1 below.

The rationale of choosing different `scaleFactor` for different image sizes is that the number of detections depend on both `scaleFactor` and image size. If `scaleFactor` is kept constant, the number of detections increases with larger image size. Whereas, if the image size is kept constant, the number of detections increases with smaller `scaleFactor`. Hence, we choose a larger `scaleFactor` for larger images and vice versa.

Image Size	Total Pixels	Scale Factor
Very Large	$\geq$ 9000000 pixels	1.2
Large	$<$ 9000000 pixels	1.1
Medium	$<$ 3000000 pixels	1.05
Small	$<$ 3000000 pixels	1.01

Table 1: Scale factors for different image sizes

After detection, the Viola-Jones object detector will pass the list of candidate detection windows to the second stage, the HOG template matcher.

Stage 2: HOG template matching
The HOG template matcher will receive a list of candidate detection windows from the first stage. It will first resize the candidate windows and templates to the same size. To achieve this, we use `cv2.resize()` method.

Thereafter, we compute the HOG features of the candidate windows and the templates using `skimage.feature.hog()` method by `skimage`.

After computing HOG, we compare each of the candidate windows with the set of templates using correlation distance measure as described by Equation 2. To do this, we use `scipy.spatial.distance.correlation()` method by `scipy`.

For each candidate window, we use the best (shortest) distance to the set of templates to score them. For example, if a window has distance 0.5 to template 1 and distance of 0.2 to template 2, we use the distance of 0.2 to compute the score for that window. The score computation formula is described in Equation 3.

We then impose a threshold of 0.25 on these scores. Only windows with scores higher than the threshold are outputted as true detections.

2.3.3 Rationale of Using Ensemble of Methods

For Viola-Jones object detection framework to work well as a standalone method, many training examples are needed to reduce the number of false positives output. This is because to reduce false positives, we need to set a low target maximum false positive rate and use a large number of stages. We need a large number of training examples to support this. Otherwise, the cascade classifier will overfit the training examples and potentially filter away true positives during testing. Dropping true positives in any of the stages is dire as they cannot be recovered in subsequent stages, resulting in non-detection.

In our application, we indeed have insufficient training examples. Therefore, as described in Section 3.2.1, we use relatively fewer number of stages and a higher false positive rate for the cascade classifier. This is so as to retain the true positives while allowing for potentially many false positives. However, this gives rise to another problem: the potentially large number of false positives may adversely affect the mAP performance of the detector.

On the other hand, template matching, as a standalone method, is too slow in detection. Furthermore, it may be sensitive to variances of the appearance of the object of interest.

In light of the weaknesses of both methods, we combine both of them in an ensemble. We place Viola-Jones detector in the first stage, with the goal of removing most negative windows

while retaining all the true positives. Unlike template matching, Viola-Jones object detection framework is able to achieve this relatively fast due to several engineering techniques discussed in detail in Section 2.1. Thereafter, we use HOG template matching to filter away the undesired false positives output by the cascade classifier.

3 Training

3.1 Data Preparation and Configuration

3.1.1 Data Source

On top of the training data provided in the course, we also sourced for other sources of training data, for example, from `https://github.com/vc1492a/Hey-Waldo`.

3.1.2 Manual data cleaning and preparation

The annotations provided by the course bound the positive windows of the characters of interest, Waldo, Wenda and Wizard, in each of the images. However, the definition of a positive window is broad — some windows contain the full body of the characters down to their legs, while others contain only their heads or half of their bodies. While having a wider variety of images allow for the training of a more generalized classifier, the context of this project does not allow for it this generalization. There will be an excess of false positives detected due to two reasons: the ambiguous nature of such a trained classifier, and the feature similarities between characters and the background. The latter reason occurs as intended by the designers of the book in creating an engaging problem for readers, thus we have chosen to target the former reason by designing dedicated classifiers that have stricter criteria.

Through our observations, it became apparent that all positive character windows contain the head and varying degrees of exposure of the body. Full body windows are scarcely found for all characters, leading to the inability to train a model solely based on full body images as there does not exist ample data. Hence, we processed the data through manual cropping to generate two sets of positive training data: heads and torsos. Heads are defined to enclose the face and headwear of the character; torsos are defined to enclose the top of the headwear to the estimated bottom of the ribcage as well as the shoulders. An example of positive samples from each category is depicted in Figure 4 below. Character torso models are intended to be the main means of detection as these models are the strictest, while character head models act as an auxiliary in the event that there is minimal to none of the characters' bodies exposed.

Figure 4: Positive sample examples. From left to right: Waldo head and torso, Wenda head and torso, Wizard head and torso

On top of the original image samples, we also carried out image augmentation to generate more positive data by using the `imgaug` library. Each positive image underwent contrast, brightness adjustments, Gaussian noise additions, as well as Gaussian blurring with varying sigmas as shown in Table 2.

Augmentation type	Range of values
Brightness Levels	[-20, -15, -10 -5, 5, 10, 15, 20]
Contrast Levels	[0.85, 0.9, 0.95, 1.05, 1.1, 1.15]
Sigmas	[0.5, 0.75, 1, 1.25, 1.5]

Table 2: Image augmentation types and values

Additionally, we also implemented a script to generate negative training data from the full images. We achieved this by masking the positive windows depicted by the annotations, followed by cropping the images into sizes of 64×64, 128×128 and 256×256 to capture different scales of the negative regions. The images produced were scanned to remove false negatives where characters appear in windows not defined in the annotations. Overall, the total number of training examples, including augmented samples, we have is summarized in Table 3.

Character	Body part	Number of examples
Waldo	Head	2730
	Torso	1155
Wenda	Head	1071
	Torso	483
Wizard	Head	735
	Torso	357

Table 3: Total number of examples for each category

3.2 Training Configuration

3.2.1 Training configuration for Viola-Jones Cascade Classifier

We have developed a training schedule with different combinations of hyperparameters in order to train our Viola-Jones detector. The hyperparameters include the size of the window, number of stages, maximum false alarm rate, minimum hit rate etc.

For each character, owing to the two sets of positive training data (head and torso), we trained two separate Viola-Jones detectors. Each Viola-Jones detector uses a different detection window aspect ratio: a head detector window uses a ratio of $1 : 1.2$, while a torso detector window uses a ratio of $1 : 2.5$. As we have three characters, we trained a total of $3 \times 2 = 6$ Viola-Jones detectors in total for each set of hyperparameters.

To ease the training process, we developed several scripts to automate the training process. A data generation script was implemented to prepare the data for training. A trainer script was implemented to train the models with different hyperparameters automatically. Additionally, a validation script was developed to evaluate the various trained models.

The training-validation data are those from `train.txt`. We used k-fold cross validation to validate our hyperparameters. For our case, $k = 5$. To help us split the training data into training and validation sets, we used `sklearn.model_selection.KFold`. An example training-validation split can be found in Appendix A. To prevent overfitting and shorten training time, we did not use the full set of training examples in training. Rather, we use 200 random positive examples and 600 random negative examples in each set of training.

During validation, we used `cv.CascadeClassifier.detectMultiScale3()` to get multi-scale detections with classification scores. As the classification score is in the range of $[-\infty, \infty]$, we used a sigmoid function to map it to a probability range of $[0, 1]$, in order to use it as the confidence score for mAP.

Thereafter, we used mAP to evaluate the performance for each of the hyperparameters set used. For each hyperparameters set, we averaged the evaluation results from each of the k-fold validation to give a final evaluation. The validation results for the various hyperparameters sets can be found in Appendix B.

The best set of hyperparameters is `w`=25, `maxFalseAlarmRate`=0.4, `minHitRate`=0.999, `bt`=GAB, `mode`=BASIC. We use these hyperparameters to train our final model. This set of hyperparameters is optimal for several possible reasons described as follows:

1. The maximum false alarm rate is relatively low. This means there is a larger number of weak classifiers at each stage, thus reducing false positives and improving performance. However, the maximum false alarm rate is not too low either in order to retain the true positives.

2. The number of stages is small to prevent overfitting to the small set of training examples available.

3. The minimum hit rate is relatively high to reduce the probability of missing an object of interest. However, is it not a perfect value of 1 in order to achieve the desired maximum false alarm rate at each stage.

An example set of trained weak classifiers using Haar features of a single stage of Viola-Jones detector is shown in Figure 5 below.

Figure 5: Example set of trained weak classifiers of single stage for Waldo's head

3.2.2 Parameters Tuning for HOG Template Matching

As described in Section 2.3.2, we used `skimage.feature.hog()` to compute the HOG feature descriptors for the candidate detection windows, as well as for the templates. We employed sev-

eral heuristics in determining the parameters for the computation of HOG feature descriptors. After testing several combinations of parameters, we decided to use 5 orientation bins, 6×6 pixels per cell and 2×2 cells per block. We also resized the templates and candidate windows to having a common width of 60. Examples of HOG features computed for Waldo's head are shown below in Figure 6.

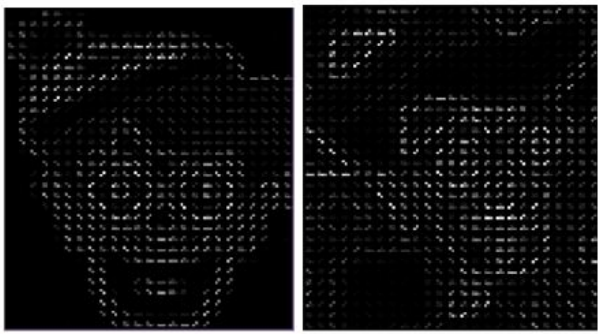

Figure 6: Examples of HOG features for Waldo's head

4 Experiments

4.1 Testing Results

After carrying out training and validation, we trained our final model of Viola-Jones detector using the best set of hyperparameters. We then combined the trained Viola-Jones detector with the HOG template matcher to give the final ensemble model.

Using the final ensemble model, we carried out testing on the images in `val.txt`. To evaluate the detection results quantitatively, we used the provided `evaluation.py` to perform mAP evaluation on the baselines generated by our final model. We tested our model using different values of `ovthresh`. The results are summarized in Table 4.

ovthresh	Waldo mAP	Wenda mAP	Wizard mAP	Average mAP
0.05	0.700	0.571	0.611	0.627
0.10	0.652	0.429	0.500	0.527
0.20	0.484	0.357	0.148	0.330
0.30	0.484	0.357	0.042	0.294
0.40	0.484	0.109	0.000	0.198
0.50	0.484	0.092	0.000	0.192

Table 4: Final evaluation results

As seen from the final evaluation results, our model performed relatively well for smaller `ovthresh` values. However, performance suffers as `ovthresh` increases. This is due to several reasons as follows:

1. As described in Section 3.1.2, we use the torsos of the characters to train the Viola-Jones detector as there are too few full bodies available. As such, we are unable to detect full

bodies of the characters, resulting in lower Intersection over Union (IoU) values for the detections. Hence, with a high `ovthresh`, our detections are likely to be marked as false positives. An example visualization of the disparity between annotations and detected windows is shown in Figure 7 below.

2. The manual annotations contain inconsistencies in the exposure of a character's body, resulting in difficulty in exactly matching the detection windows to those of the annotations.

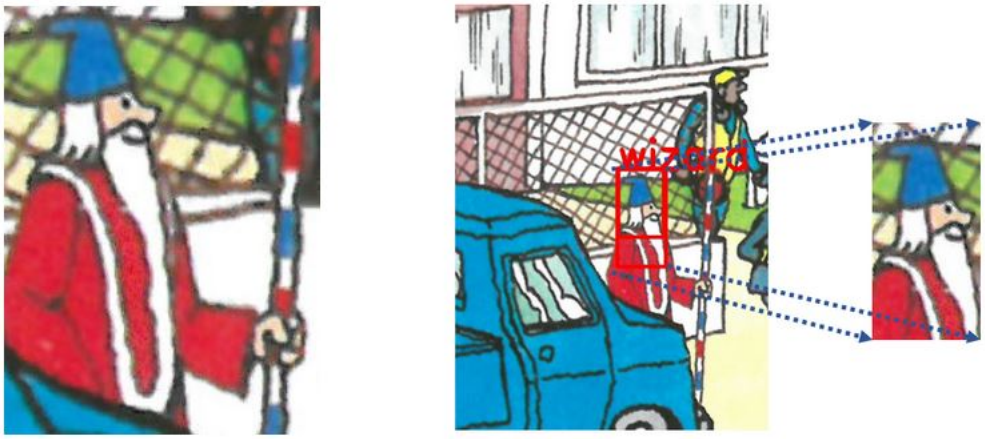

Figure 7: Annotation true positive (Left) and model detected positive (Right)

Despite not being able to match the exact the annotation bounding boxes, our model is able to localize the characters in most images as proven by our results in Figure 7. We believe that detections of this nature serves as necessary and sufficient conditions to the solution of locating the characters. Determining the position of the head or torso of a character allows us to locate the character immediately. It also opens up the possibility of future enhancements to the ensemble, where the locality of the detected character (particularly in the positive height and width directions) can be parsed to enlarge the bounding box to detect the entirety of the character.

Example visualizations of detection results by our ensemble model are shown in Figure 8 to Figure 11.

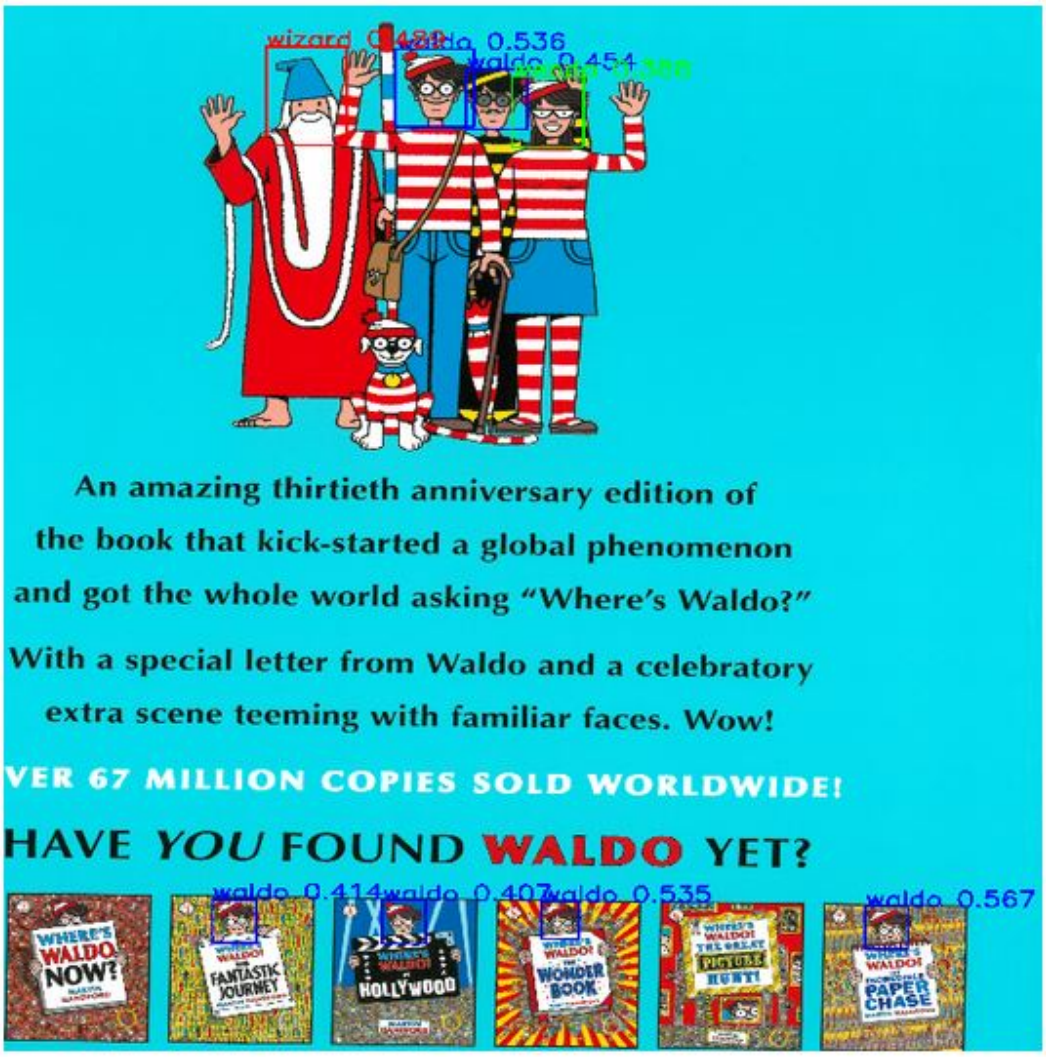

Figure 8: Detections results for image 004

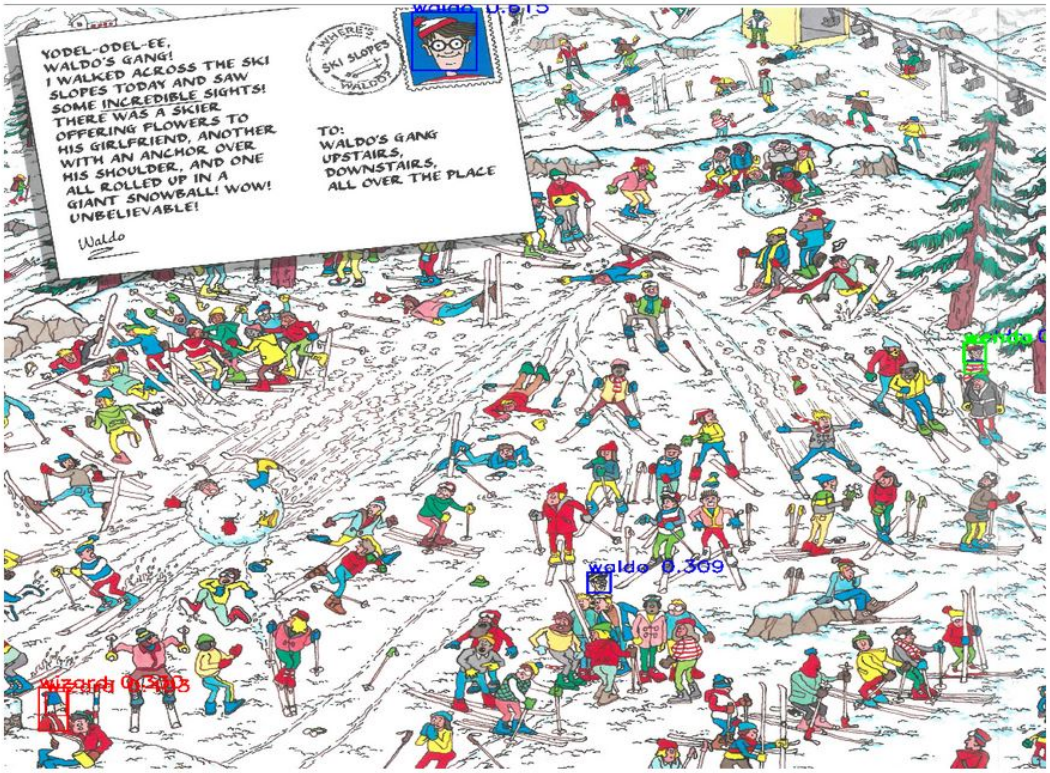

Figure 9: Detections results for image 038

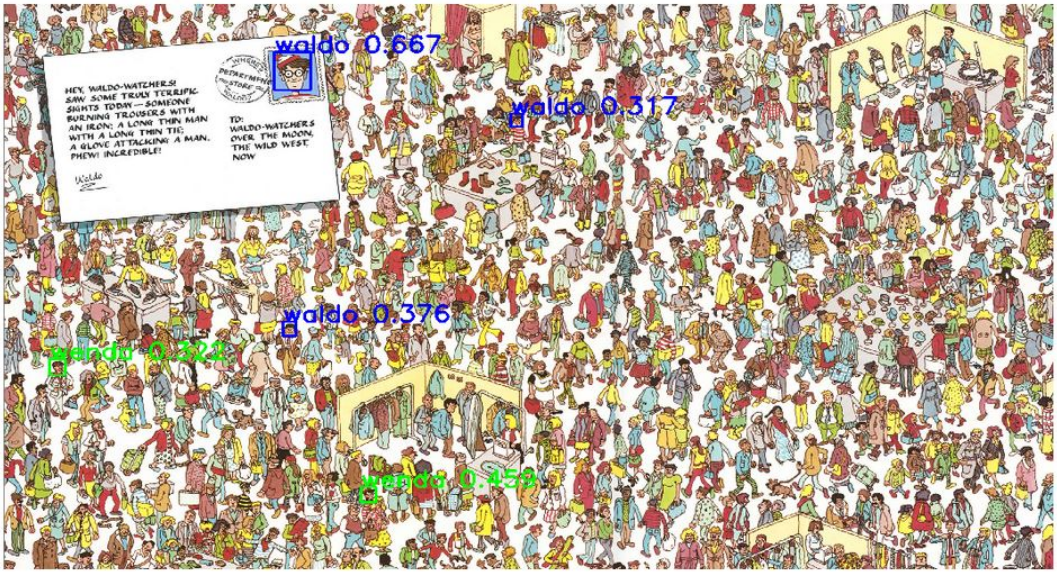

Figure 10: Detections results for image 042

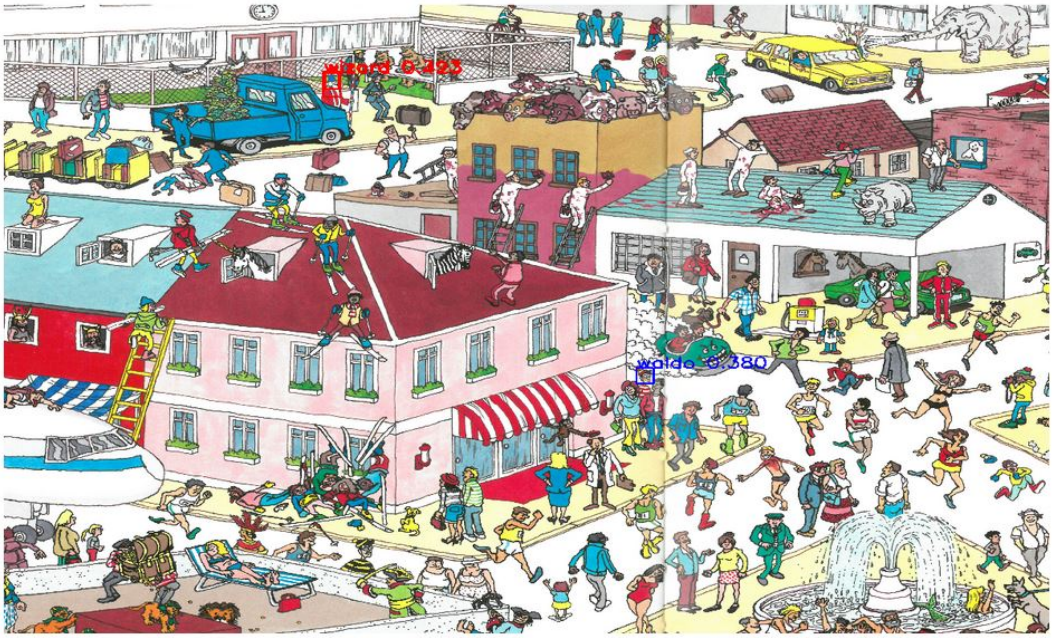

Figure 11: Detections results for image 050

5 Discussion

5.1 Strengths and Weaknesses

The strengths of our ensemble model are described as follows:

1. Our ensemble model combines the strengths of the individual models, i.e. speed of Viola-Jones detector and accuracy of HOG template matching, to achieve a faster and more accurate model than standalones.

2. Our ensemble model is fairly accurate considering it as a non-deep method. As there are relatively more examples for the heads of the characters, our model is able to detect the character heads accurately. Our model is also able to localize the characters to great accuracy, which is sufficient for the goal of finding the characters.

3. Viola-Jone detector allows for great flexibility in choosing the boosting scheme, feature type etc.

4. HOG feature used in template matching is known to be a robust feature type.

The weaknesses of our ensemble model are described as follows:

1. Our detection window aspect ratio is fixed at $1 : 1.2$ for head detection and $1 : 2$ for torso detection. However, these may not be the true ratios for all occurrences of heads and torsos of the characters.

2. Our current ensemble model does not merge head detection with torso detection if both are detected at the same location, resulting in poorer mAP scores.

3. Due to insufficient training examples, our Viola-Jones detector cannot be optimized. To ensure detection of the characters, we used only the torsos instead of the full bodies of characters. This is because there are simply too few full bodies available for useful training. As a result, our detector is only able to detect the torsos instead of the full bodies of the characters, which adversely affects mAP scores.

4. Since Waldo and Wenda look similar, HOG template matching may not be able to differentiate them well enough sometimes.

5.2 Other Methods Attempted

On top of the ensemble method, we have also attempted other methods, which are described in detail in Appendix C.

6 Conclusion

In this project, we designed and proposed an ensemble of methods for the task of "Where's Waldo". We combine both Viola-Jones framework and HOG template matching to achieve a robust ensemble object detector for our application. The two methods selected are able to complement each other's weaknesses. Hence, we are able to achieve a decent detection results on the test image set, with a mean average precision (mAP) score of 0.294 using `ovthres` = 0.3.

7 Group Information

Member	Student ID	Email
Joshua Lee Kai Sheng	A0161868R	e0104999@u.nus.edu
Tian RunXin	A0209160N	e0446367@u.nus.edu
Yoon Jia Jun, Ken	A0155677N	e0031814@u.nus.edu

Table 5: Group member information.

7.1 Contributions

Contributions by Joshua Lee Kai Sheng

1. Researched on theory of various methods

2. Implemented Viola-Jones training and detection

3. Implemented SVM with HOG training and detection (`openCV` and `sklearn`)

4. Implemented scripts for file and data generation (basic cascade classifier training, extraction from annotations `xml` files etc.)

5. Implemented scripts for proper training (data generation, training and validation)
6. Implemented scripts for ensemble detector (detection and visualization scripts)
7. Sourced for additional datasets
8. Created training plan for Viola-Jones
9. Trained Viola-Jones detector
10. Developed midterm report and final report

Contributions by Tian RunXin

1. Researched on template matching
2. Implemented basic template matching
3. Implemented HOG template matching
4. Implemented scoring for template matching
5. Experimented with ensembles
6. Tuned parameters for HOG
7. Fine-tuned HOG template matching
8. Refined templates for HOG template matching
9. Developed midterm report and final report

Contributions by Yoon Jia Jun, Ken

1. Manual cropping of positive images into head and torso
2. Generation of negative examples from whole images excluding the positive region
3. Image augmentation to generate more positive images
4. Researched on Viola-Jones method
5. Trained Viola-Jones detector
6. Tuned hyperparameters for Viola-Jones detector
7. Implemented scripts for final model training
8. Picked templates for HOG template matching
9. Coordinated training process
10. Developed midterm report and final report

References

[1] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. 2005.

[2] Ludmila I Kuncheva and Christopher J Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning*, 51(2):181–207, 2003.

[3] Paul Viola, Michael Jones, et al. Rapid object detection using a boosted cascade of simple features. *CVPR (1)*, 1(511-518):3, 2001.

A Training-Validation Data Split

Fold number	Training image indices	Validation image indices
1	008 000 072 011 052 017 002 066 023 032 071 065 057 076 009 004 007 013 033 039 068 027 010 025 049 063 050 078 026 028 021 070 001 058 062 048 031 059 014 016 064 053 075 051 069 022 019 077 061	060 006 005 029 012 055 015 042 044 079 020 037 034
2	060 006 005 029 012 055 015 042 044 079 020 037 034 076 009 004 007 013 033 039 068 027 010 025 049 063 050 078 026 028 021 070 001 058 062 048 031 059 014 016 064 053 075 051 069 022 019 077 061	060 006 005 029 012 055 015 042 044 079 020 037 034
3	060 006 005 029 012 055 015 042 044 079 020 037 034 008 000 072 011 052 017 002 066 023 032 071 065 057 063 050 078 026 028 021 070 001 058 062 048 031 059 014 016 064 053 075 051 069 022 019 077 061	076 009 004 007 013 033 039 068 027 010 025 049
4	060 006 005 029 012 055 015 042 044 079 020 037 034 008 000 072 011 052 017 002 066 023 032 071 065 057 076 009 004 007 013 033 039 068 027 010 025 049 059 014 016 064 053 075 051 069 022 019 077 061	063 050 078 026 028 021 070 001 058 062 048 031
5	060 006 005 029 012 055 015 042 044 079 020 037 034 008 000 072 011 052 017 002 066 023 032 071 065 057 076 009 004 007 013 033 039 068 027 010 025 049 063 050 078 026 028 021 070 001 058 062 048 031	059 014 016 064 053 075 051 069 022 019 077 061

Figure 12: Training-Validation Data Split for 5-fold validation

B Training Schedule and Results

w	minHitRate	maxFalseAlarmRate	mAP**
20	0.99	0.4	0.102
		0.45	0.089
		0.475	0.091
	0.995	0.4	0.102
		0.45	0.089
		0.475	0.091
	0.999	0.4	0.154
		0.45	0.143
		0.475	0.147
25	0.99	0.4	0.106
		0.45	0.095
		0.475	0.096
	0.995	0.4	0.155
		0.45	0.137
		0.475	0.140
	0.999	**0.4**	**0.182**
		0.45	0.173
		0.475	0.166

Figure 13: Training schedule and results
**mAP calculated using `ovthres` = 0.5, best across possible number of stages

C Other Methods Attempted

C.1 Dalal-Triggs-like Detection using HOG features

This method uses HOG features with SVM for object detection. A sliding window is used to detect the object of interest at all positions. An SVM is trained using HOG features of training images. Thereafter, the trained SVM is used for object detection all locations.

For this method, we tried out HOG and SVM implementation from two different libraries, `openCV` and `sklearn`.

For the `openCV` version, we had great trouble in saving the trained SVM in the right format due to poor official documentation. We managed to train and save a model using guides from online forums. However, the trained model took a very long time for detection and was highly inaccurate, with no detection at all for obvious cases.

For the `sklearn` version, we `skimage.feature.hog` for HOG calculations and `sklearn.svm.SVC` as the SVM. Unlike openCV, there is no multiscale detection method in `sklearn`. Hence, we used a self-implemented sliding window and `skimage.transform.pyramid_gaussian` to achieve multiscale sliding window. Additionally, we also used `imutils.object_detection.non_max_suppression()` to do non-maximum suppression on the candidate windows.

From rough visualization of this method's results, the method seems to be underperforming. Most of the time, Waldo is not even included in any of the windows outputted. This could be due to lack of training data or the various parameter tunings (e.g. kernel type, C etc.) that we have not tried out. Furthermore, this method seems to take a longer time for detection than the Viola-Jones method. An example visualization of our current results is shown in Figure 14

Figure 14: Detection Results for SVM with HOG features (Waldo not detected)

C.2 Basic template matching

This is a brute force method that uses basic template matching based on pixel values. Since template matching is highly susceptible to variances within the positive class, we used a set of representative templates to do the matching. Convolution was done across the image with these templates across different scales of the image. For each template, we outputted the maximum response window as the detection. We then aggregated these windows as our detection result. We used `openCV` library for this purpose.

As expected, this method is extremely slow due to its brute force nature and the slowness of sliding window convolution. Besides, it is highly sensitive to the scaling factor of the image in multiscale detection. Furthermore, there is a trade-off between accuracy and time performance in choosing the number of templates used and the smoothness of the scaling process. From rough visualization of this method's results, the method seems to be able to detect Waldo in some cases. An example visualization of our current results is shown in Figure 15.

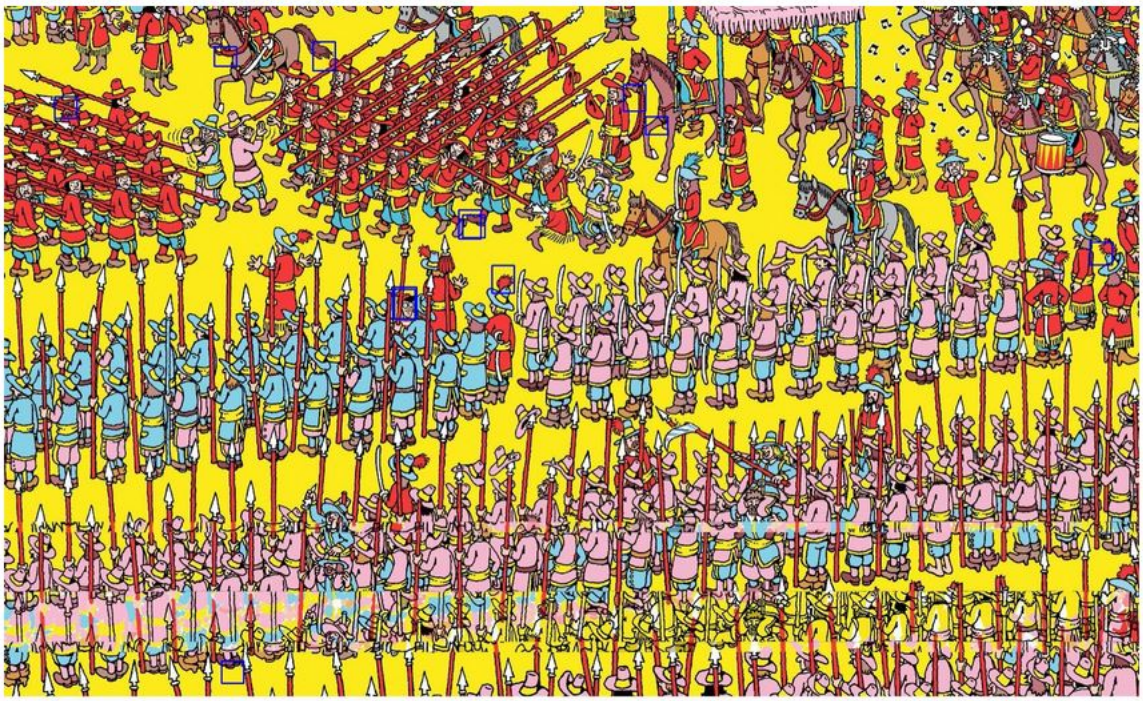

Figure 15: Detection Results for Basic Template Matching (Waldo detected)