NUS CS4243: Computer Vision and Pattern Recognition November 14, 2019

Finding Waldo Using A Non-deep Computer Vision Algorithm

Lecturer: Angela Yao *Student(s): Archana Pradeep, Shradheya Thakre, Wu Di*

Abstract

The aim of this project is to find three characters - Waldo, Wanda and Wizard in images from a popular book called Where's Waldo using any 'non-deep' computer vision algorithm. In this paper, we explore the different techniques we tried and our final proposed solution and how we optimised it in the context of this project.

1 Introduction

1.1 Problem statements

- Detect three characters within a densely populated image with an acceptable accuracy
- Find the characters as fast as possible

1.2 Existing Methods

1.2.1 Template Matching

It is the method of finding the location of a template image in a larger image by sliding the template image over the larger image until there is a match.

We first convert the original image to gray-scale and then extract the template using the coordinates of x and y provided in the annotated data set. Then this template is made to slide over the original image until the section of the image and the template match up to or beyond a predefined threshold value. For the template matching itself, opencv's matchTemplate() method was used.

This approach was not chosen as the final solution because:

- A different threshold value had to be set for each colored image.
- The matching was very slow as it would run through the entire image.

1.2.2 Support Vector Machine with SIFT

This method is similar to the one used in Lab 4 where we extract SIFT descriptors from the image and then get a bag of sifts which is essentially constructing histograms by assigning the local descriptors to its nearest cluster center and build a histogram to indicate the frequency at which each cluster was used.

Then using the labels we obtain from the bag of sifts we plug it into the svm machine learning model and then classify our test images.

We also tried to use HOG for feature descriptors as well since on initial research we found it to be better for feature description for characters and faces.

The reason we decided to not use this solution was because although the accuracy was good (96%) it was extremely slow due to the sliding window.

In order to improve speed we also used custom keypoint detectors which found red and white stripes but this didn't provide good result as wizard and other instances of other characters sometimes had their body hidden and could not find those true positives.

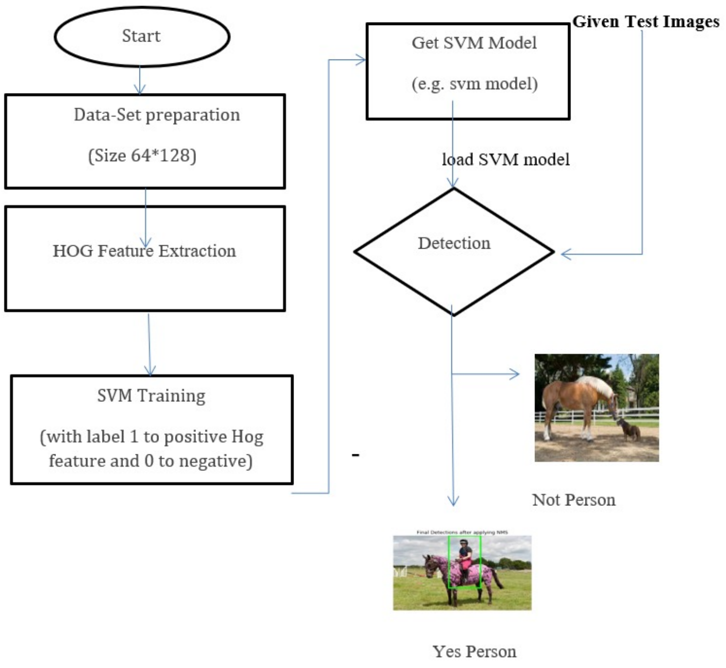

Figure 1: HOG descriptors with SVM

1.2.3 Interest Point Matching

We used SIFT descriptor to find key points. Then the waldo image was cropped to be used as a base image. Opencv's detectAndCompute() method was used to detect the key points and the corresponding descriptors by using CV brute force matcher.

However, we decided to not go with this method since there were a lot of false positive results since the features having similar key points were falsely identified as waldo.

1.3 Summary of Proposed solution

- We propose using Haar Cascades which is a machine learning object detection algorithm. This is further explained in detail in section 2 of the report.
- With this solution we achieved mAP of 0.11.

2 Proposed Solution

Haar cascade is a machine learning object detection algorithm used to detect objects in images or videos based on the paper put forth by Paul Viola and Michael Jones in 2001 titled "Rapid Object Detection using a Boosted Cascade of Simple Features". In this approach a cascade function is trained from positive (images of the character) and negative (images that are not the character) images which is then used to detect the characters in the other images.

First to train we need the Haar features. A Haar feature considers adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums. The diagram is shown in Figure 2 below.

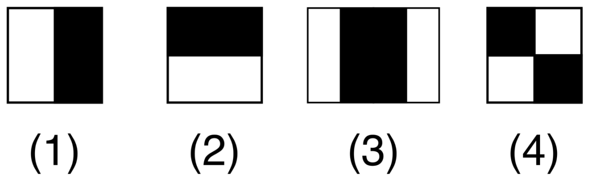

Figure 2: Haar features

This method involves the following steps:

- Select Haar Features
- Create Integral Images
- Training using adaptive boost algorithm
- Training cascading classifier [2]

3 Experiments

3.1 Data Preparation and Configuration

We did not think the provided annotations were enough for this model so we set out first to create positive and negative images for the three characters - Waldo, Wenda and Wizard.

We began by using the given annotations to crop out the images with the coordinates given. This became our positive set of images for each of the characters. We also cropped non-waldo, non-wenda and non-wizard parts of the images to use for our negative set of images. Additionally, more positive and negative images were obtained from images in github repositories (cite).

The accuracy of the model is then evaluated using mean average precision (maP) which is a well-known metric for assessing object detectors. It consists of three components : precision, recall and Intersection over Union(IoU) which was introduced during Lab 5.

Dataset	#train	#test
Waldo	130	13
Wenda	44	8
Wizard-	46	6

Table 1: Summary of datasets.

3.2 Implementation

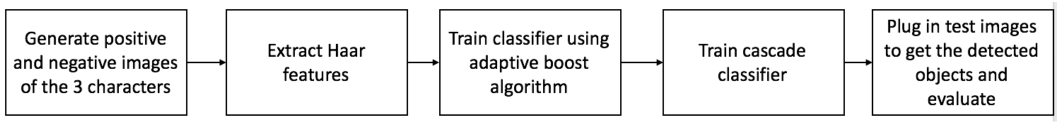
Generate positive and negative images of the 3 characters
Extract Haar features
Train classifier using adaptive boost algorithm
Train cascade classifier
Plug in test images to get the detected objects and evaluate

Figure 3: Our implementation pipeline

3.2.1 Generation of Positive and Negative Images

As mentioned in section 3.1, the positive images are those of the characters itself while negative images are non-character images, maybe the background of the image or one of the other characters, for e.g.: Waldo could be a negative for Wizard. These images were cropped by providing the x and y coordinates that made a box around the region of interest in the images into a script that would just crop these parts of the images out. The positive samples were created using opencv_createsamples method while negative samples were done using a script to generate cropped images and picked manually from the generated. [4]

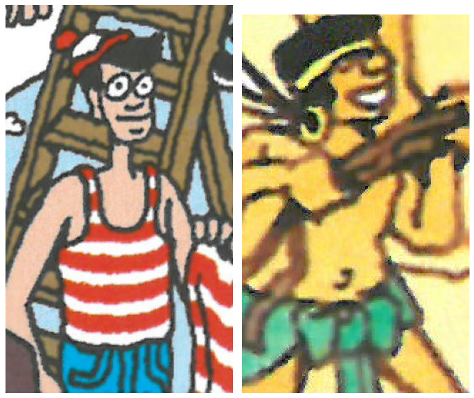

Figure 4: example of positive waldo and negative waldo

The positive samples are used by the adaptive boosting algorithm to determine what features it should choose to effectively detect the character of interest. The negative samples are randomly selected with quantity of 2000 to 4000.

3.2.2 Training Haar Cascade

OpenCV's train cascade function does step 2,3 and 4 with one command by modifying the different parameters. We decided to go with the default adaptive boosting algorithm provided which was the Gentle AdaBoost as changing this didn't seem to be affecting the accuracy of the detection by a large margin.[1]

We also trained with 15 cascade stages. The principle behind the cascade classifier is to combine weak classifiers to obtain a stronger classifier. The features obtained from the positive and negative samples are used to construct a binary decision tree. However one binary decision tree is not enough to be a weak classifier (accuracy $> 50\%$). Thus opencv combines stumps for each stage until it hits a weak classifier and multiple of these stages cascaded builds a strong classifier.

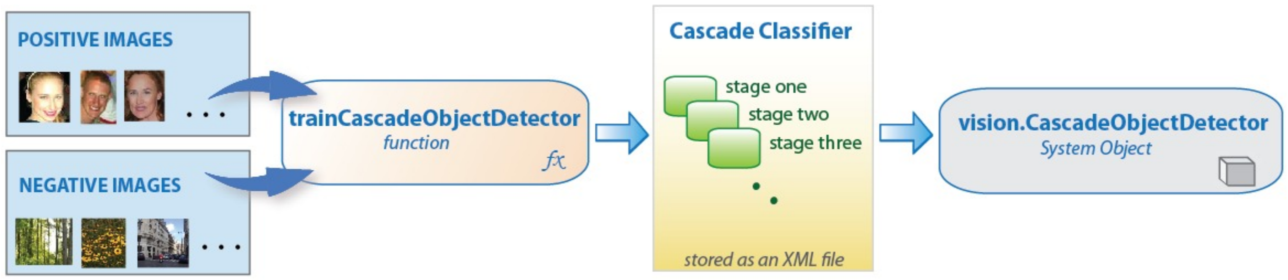

Figure 5: How cascade classifier works

3.2.3 Haar Cascade with SVM

After out initial implementation of SVM we realised that it gave a very good accuracy of 92+%. The main issue with it was the reduced speed due to trying different scales and implementing a sliding window.
However, we used a different technique where we used Haar Cascade detectMultiscale3() to get the initial 100-200 key points which most likely have waldo and then running the svm fit() to find again eliminate the false positives since the SVM model is better trained to find the classification.

This produced slightly better result than just using the detectMultiscale() for Cascade classifier. But this wasn't much faster than the sift solution alone. This was mainly because of scaling that still wasn't possible and not giving effective result when using SVM prediction.

3.2.4 Testing the classifier

To test the classifier we first use opencv's CascadeClassifer() to load the classifier from a .xml file.
Once we have the the loaded classifer, we use detectMultiScale() method. The parameters for this method includes image which takes the image to test on. We converted this to greyscale for faster computation. the scaleFactor parameter is used to create the scale pyramid. We used 1.03 which means the image's is reduced by 3%, thus increasing the chance of a matching size with the model for detection is found. [3]

The parameter minNeighbors is set to 10. This specifies how many neighbors each candidate rectangle should have to retain it. A higher value for this parameter results in less detections but with higher quality. The detector will run on a multiple scale style and at the same time following sliding window strategy. After this step, it will give multiple responses even for a single region of interest. minNeighbors tends to filter these responses just like by setting up a lower-bound threshold, i.e. it will only be counted as a valid character if the number of responses for the character is higher than minNeighbors.

The method also returns level weights which are he confidence value of the classification on that stage only. This is then saved to a .txt file and used for evaluation.

We have tested with different parameters in order to increase precision. Increasing min-Neighbors made it faster and found less keypoint hence decreasing the false positives, but we had to find a good value to stop which gave a fair result in terms of precision. Moreover, by having a low scale factor we could detect characters clearly and get more edge cases but we had to find an optimum number which was fast and accurate. This require trial and errors and many checks with different test data.

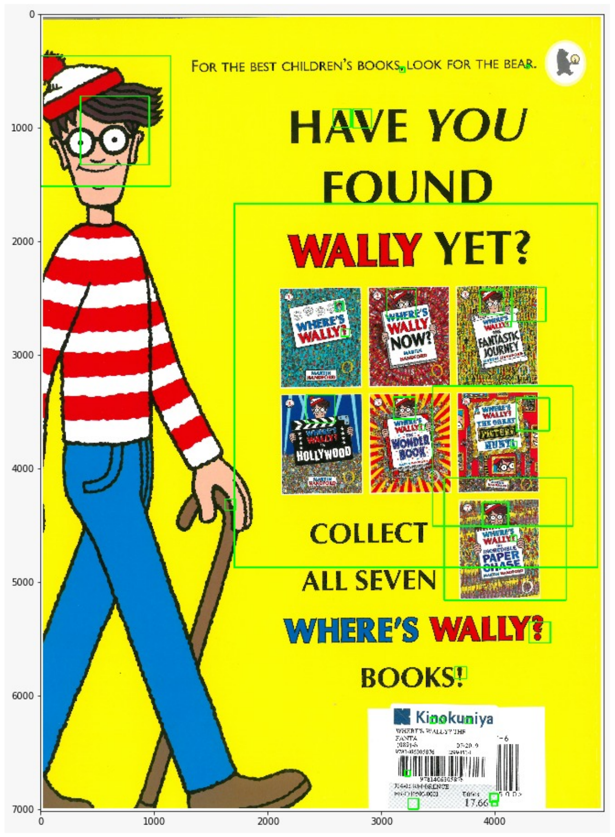

Figure 6: what the detection would look like

3.3 Results

After training with an dataset of 71 images annotations and running them on the images of val.txt we got the following AP for the characters

- Waldo: 0.14
- Wenda: 0.15
- Wizard: 0.0

This gives us a total mAP of 0.11.

Analysis

While the AP for characters is not high, the true positives are usually corrected very precisely and accurately. The main issue was finding lots of false positives which resulted in poor AP for the characters.
Despite using new annotations which just used the character's faces and were corrected manually, we realised that due to the lower number of positive training images especially for Wenda and Wizard it was tough to get good training results using Haar cascade and hence not giving us a very good result.
We got a slightly better mAP when we used svm prediction after finding the initial points through Haar cascade and then running through a second check using SVM prediction. However this was slower and the accuracy did not see a considerable improvement.

3.4 Discussion

3.4.1 SVM with SIFT Descriptors

- **Pros:** It has a great accuracy of 96% when tested on predicted labels when initial accuracy testing is done. We only need 4 classifications, waldo, wenda, wizard and none. Another training with svm we tried was to have three different classifiers each having positive and negative. This also gave us a higher accuracy when checked on test annotations with the built vocabulary

- **Cons:** For classifications of image, lot of assumptions need to be made regarding the window size and step size which might not be applicable for all types of testing images In order to effectively find waldo in test images we will have to run the svm fit() method on a lot of small images and get the closest classification. The out of the box methods don't allow us to easily test for different scales and hence we need to run the function multiple times to actually find all sizes of characters.

In order to reduce the keypoints to test for, we used ORB (fusion of the FAST key point detector and BRIEF) descriptor and SURF (speeded up robust features) keypoint detectors, but since these aren't well defined for faces detection the algorithm didn't work well to find all true positives and gave us bad results even though it was faster.

3.4.2 HAAR Cascade Detector

- **Pros:** This technique is faster than SVM and also worked very well in finding all positives.

  The function for detectMultiScale() provided us with testing for scales of different increments and also found keypoints which were more likely to be characters and hence helped us identify all true positives in much lesser time.

HAAR is also a good feature descriptor for faces and hence helps in finding character windows faster and find if its a potential character from the training set.

- **Cons:** Due to the sheer amount of training data for each character, we couldn't get good training results and this was our main reason for a lot of false positives being identified by the key descriptors.

4 Conclusion

In conclusion, we initially started of with the most basic method of template matching which we were made aware of during the lecture and realised it had issues with both speed and accuracy which didn't fit the aim of this project. In the process of looking for other methods, we drew inspiration from our lab 4 to experiment with bag of sifts and support vector machine. Since we weren't satisfied with the speed of this method we looked for alternatives until we settled on Haar cascade classifier which was faster and gave better accuracy than the methods mentioned in section 1.2 of the report. And hence our final mAP as mentioned in the results section is 0.11

5 Group Information

Member	Student ID	Email	Contribution
Archana Pradeep	A0162694W	e0140076@u.nus.edu	Report and Research for potential solutions
Shradheya Thakre	A0161476B	tshradheya@u.nus.edu	SVM Implementation, Haar Cascade and overall implementation of project
Wu Di	A0160527J	wudi@u.nus.edu	Data generation and model training

Table 2: Group member information.

References

[1] Cascade classifier training. `https://docs.opencv.org/3.4/dc/d88/tutorial_traincascade.html`, journal=OpenCV.

[2] Creating your own haar cascade opencv python tutorial. `https://pythonprogramming.net/haar-cascade-object-detection-python-opencv-tutorial/`.

[3] cv::cascadeclassifier class reference. `https://docs.opencv.org/3.4.3/d1/de5/classcv_1_1CascadeClassifier.html#a90fe1b7778bed4a27aa8482e1eecc116`.

[4] Deep learning haar cascade explained. `http://www.willberger.org/cascade-haar-explained/`, journal=Will Berger, year=2018, month=Aug.