

# Finding Waldo

*Lecturer: Angela Yao*

*Student(s): Calvin Tantio, Gadupudi Mukesh, Koh Yong Hong Lawrence*

---

## Abstract

In this report we will talk about the Finding Waldo Problem and how we implemented the algorithms to find Waldo without using Deep Learning of any sort. We also show how the results from current known algorithms are poor and can be improved significantly after incorporating a novel filtering approach specifically designed to identify repeating patterns like the patterns on Waldo's shirt and Wenda's cap. With our final proposed solution we were able to achieve a way better accuracy at identifying Waldo compared to the baseline model based on known algorithms.

## 1 Introduction

"Where's Waldo" is a series of Wimmelbilder books featuring Waldo, along with his friends, Wenda and Wizard, who set off on "a-world-wide hike". The books feature a series of detailed illustrations. Our task is to use non-deep computer vision algorithms to detect Waldo, Wenda and Wizard on each of these illustrations.

In this project, we have explored some existing algorithms such as SIFT [1] feature matching, HOG [2] feature classification using SVM classifier [3] and cascade classifier [4] to try to solve this challenge. In this report, we detail how we carry out each of our object detection task and discuss the results that we obtained.

The innovation in our approach to this challenge lies in the additional filtering of Waldo's shirt and Wenda's cap. It is observed that these two objects have distinctive red and white stripes that distinguish them from the rest of the images. It is shown that by doing this filtering before running our object detection algorithm, we can reduce the testing time drastically and detect more Waldo and Wenda images more accurately.

## 2 Proposed Solution

Generally, there are 2 different approaches that we have explored:

- Feature matching
- Classical Machine Learning classifiers (Cascade and HOG-SVM)

We also did some additional filtering process that aims to point the detectors to the potential locations of the object of interest, which will be explained later.

## 3 Experiments

### 3.1 Data Preparation and Configuration

As we have decided to use a sliding window approach to search for Waldo, Wenda and Wizard (hereby collectively referred to as “objects of interest”, or simply “objects”), it is important for us to know the distribution of the widths and heights of the ground truths. This will help us to choose the appropriate window size(s) for our detectors.

We have written scripts `generate_train_val_img.py` and `plot_height_width.py` to visualise the training and validation images as well as the distribution of their heights and widths based on the given Annotations respectively. Figure 1 below shows some of the training and validation images extracted. More extracted images can be found in `./datasets/extracted_images` directory.

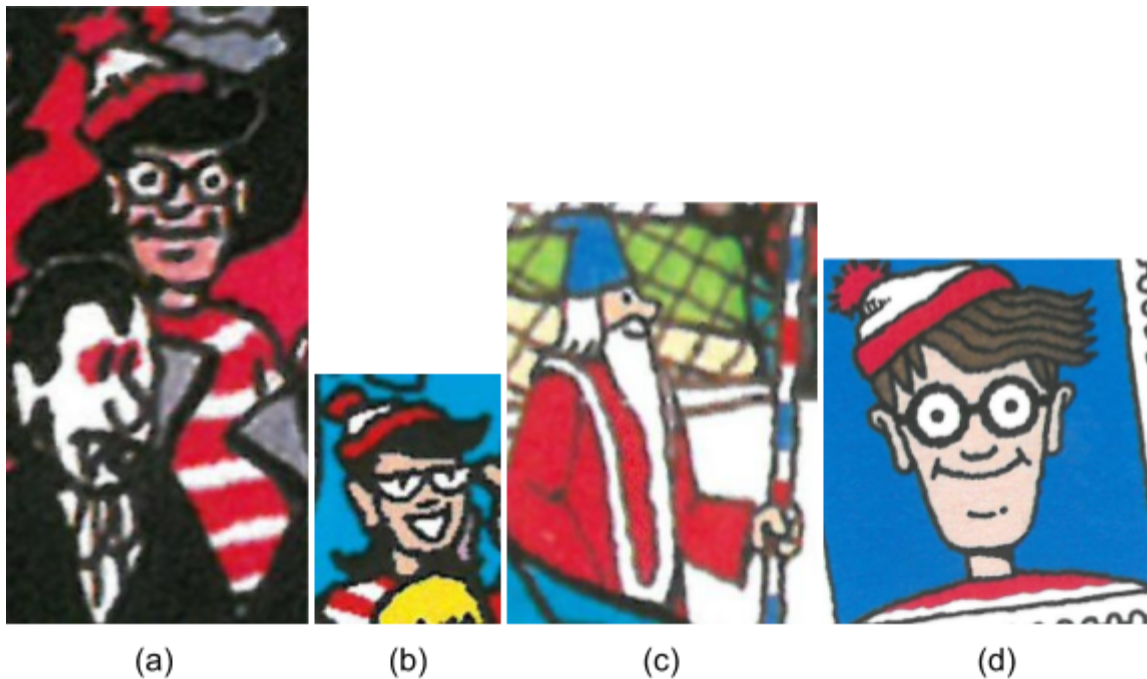


Figure 1: Samples of training and validation images

As we can see, the training and validation images have varying sizes. Figure 2 below shows a scatter plot of all the heights and widths of the training and validation images. Note that `plot_height_width.py` script not only shows the scatter plot but also saves the data in CSV format. We have used this CSV data to regenerate the scatter plot in Excel.

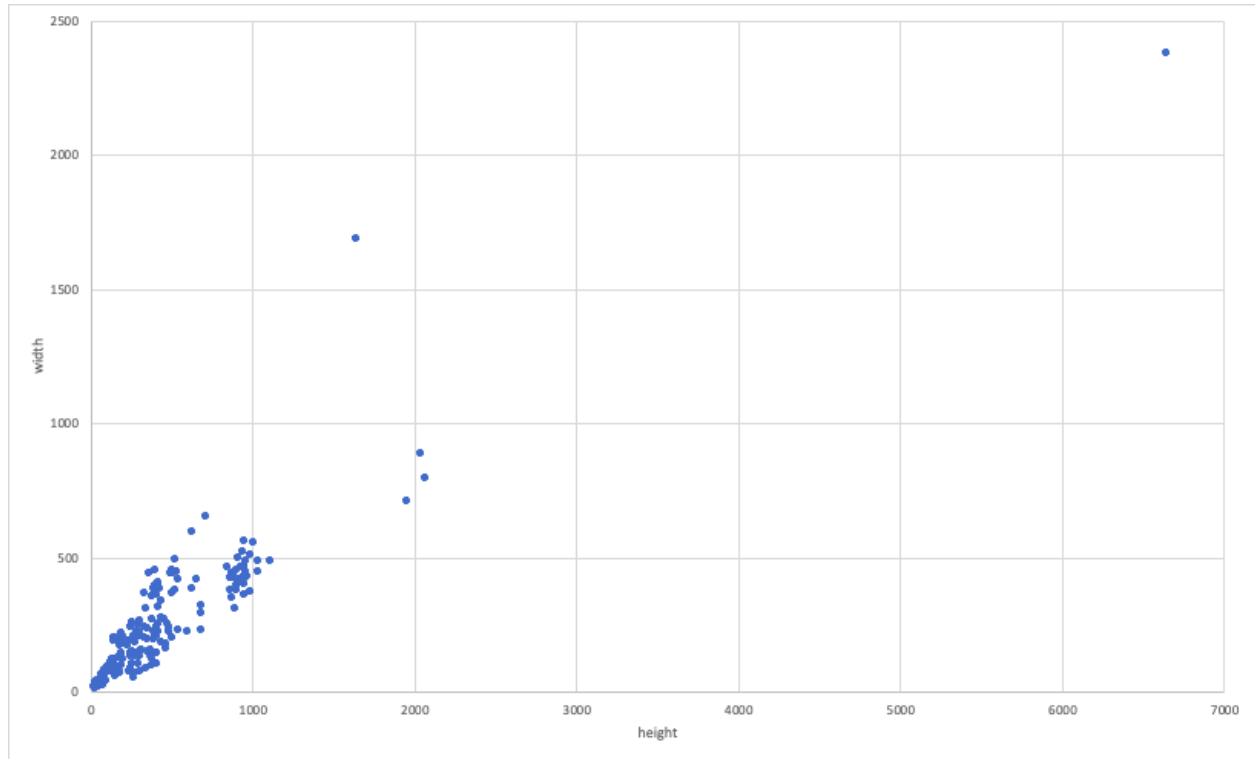


Figure 2: Scatter plot of the width against the height of the given training and validation set

We can see that the range of the heights (19 - 6647) and widths (14 - 2377), even with some extreme points removed, is still quite wide. A small detector window will not be able to detect the entirety of large objects. Similarly, large detector window covers too many surrounding noises (background) from the objects of interest. Besides that, both lead to low IoU score. The problem becomes worse when we need to use a classifier. As we resize the training images to train the classifier, large and small images will be compressed and stretched respectively. We do not want these images to be distorted too much, especially out of proportion.

Moreover, we observe from Figure 1 that the area coverage of the objects of interest also varies from an image to another. For instance, Figure 1(a) covers the face and the body of Waldo while Figure 1(d) only shows Waldo's face. This may affect the features extracted; and hence, our object detection results.

Based on the two reasons stated above, we have decided to re-extract the training images by ourselves. Our new training images only comprise of the faces of Waldo, Wenda and Wizard. With this new training images, we rerun `plot_height_width.py` to obtain the following results.

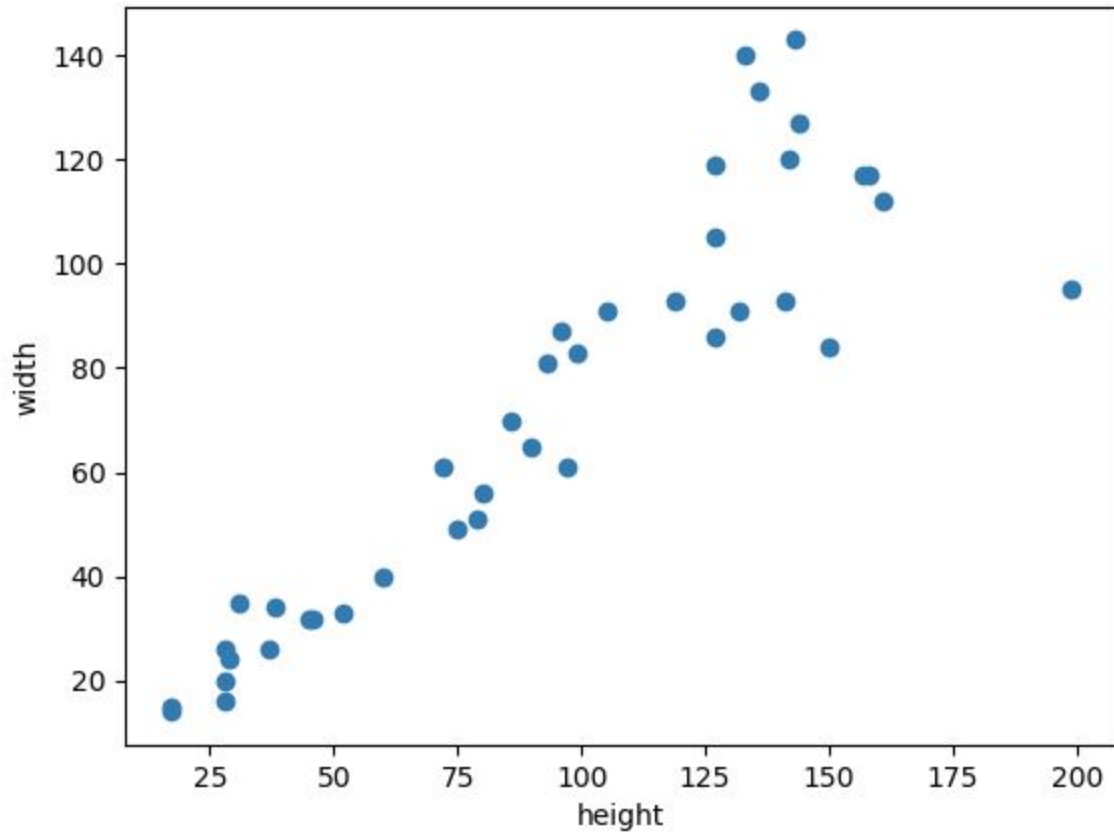


Figure 3: Scatter plot of the width against the height of the cropped Waldo faces

The results are arguably better than what we have originally. We hypothesise that this will give us a better chance of finding the objects of interest.

Finally, the following table shows the summary of our datasets:

Datasets	#train	#test	Location (Taking the group repository as the root directory)	Description
Extracted Images	184	23	.\datasets\extracted_images	Contains images for Waldo, Wenda and Wizard
Faces	390	43	.\datasets\faces	Dataset of cropped faces of Waldo, Wenda, Wizard, other faces and background images.  Actual image contains more than 433 (390 + 43) but they are just duplicates.
Specs	76	8	.\datasets\specs	A dataset containing cropped

				<p>images of Waldo's glasses.</p> <p>Part of a failed attempt and classifying by glasses.</p>
Cascade	542	0	Not in repo	<p>A dataset containing 101 waldo faces (half are flipped copies of the original) and 441 non Waldo examples. Used to train the Cascade dataset.</p>

## 3.2 Implementation

While there were many trials and errors while implementing the solution, we tried to use various approaches to get the best mAP. Several limitations like speed of training, speed of testing and training data size have led us to implement various solutions. Each of the proposed solution tries to address a limitation while setting a base for more a more complicated solution towards the end. We ended up combining various proposed solutions along with an in house developed additional filtering process to improve the location of search and speed of testing. This even improved the speed of testing by over a 1000% as mentioned below.

### 3.2.1 Proposed Solution 1: Feature Matching

As its name suggests, feature matching extracts keypoints (features) from the provided template image and tries to match them to keypoints in the test images. Feature matching is arguably better than template matching for this task as we would assume that the keypoints extracted are more resistant to changes. It is observed that the objects of interest, while similar, are not exactly the same. Template matching will unlikely to perform well given that it is not invariant to scale and rotation.

In this project, a brute force matcher is used. A brute force matcher finds all the features in the template image and the test image. Then, it takes all the keypoint descriptors from the template image and computes their distances to all other keypoint descriptors in the test image. Only the closest pairs are returned. Also, the SIFT feature is chosen as it is both rotation and scale-invariant. *We will discuss the feature matching process using Waldo as an example.*

The first step is to determine which images are going to be used as the template image. A script `show_sift_match.py` is used to display SIFT features matches between two images. From a simple Waldo faces test, we can see from Figure 4 that some matches are quite successful while others not so much.

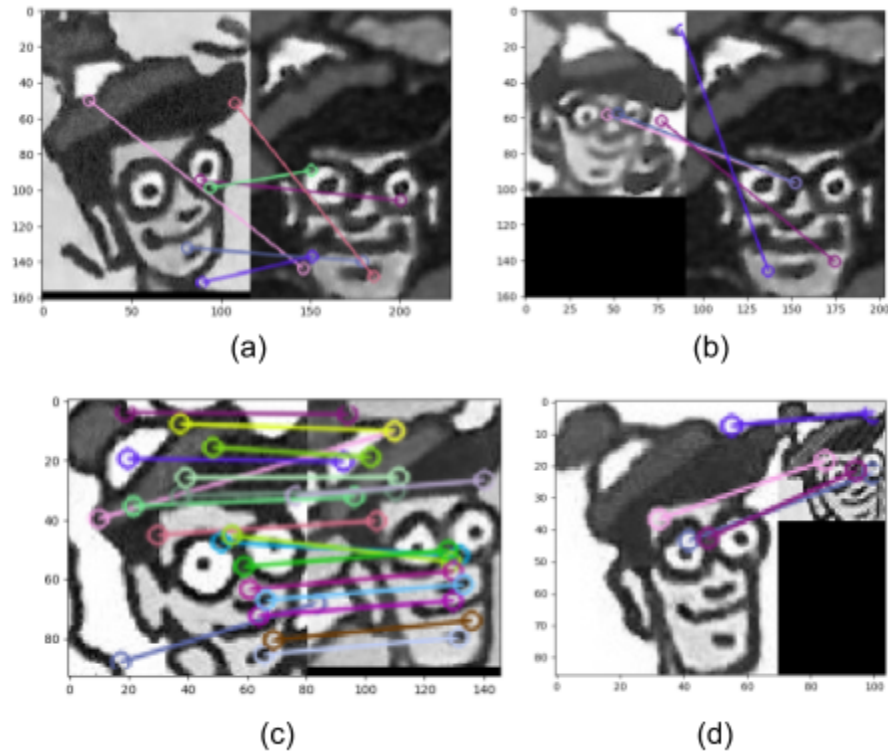


Figure 4: SIFT feature matching of Waldo's faces

We have considered several ways to extract Waldo's face that is the most representative of the Waldo population. The first option is to consider the number of matches. However, as seen in Figure 4, the number of matches is not representative of how well the images are matched. Despite having more matches, Figure 4(a) has poorer keypoint features matches than Figure 4(d). The second option is to find the average of all the Euclidean distances of the matches. This does not work either as there is a higher possibility that images with a lower number of matches have higher Euclidean distance. Thus, currently, we believe that the best option is through trial and error. We sampled some Waldo faces and non-Waldo faces (faces and non-faces) and observe the SIFT matching among them. Finally, we have chosen the following image of Waldo as a template.



Figure 5: Waldo's face template image chosen

The next step is to determine the size of the sliding window. For feature matching, there is only one consideration for the size of the sliding window, that is whether the window can capture the entire Waldo face. Looking at Figure 3, we can have the base sliding window with 40 (width) x 50 (height) dimension. We adopt the idea of scaling the sliding window to take into account the varying sizes of Waldo's faces. We will scale the sliding window by 2 and 4 times as it slides through the test images.

For the actual object detection, we compare the top 20 features with the highest response. It seems like these features are generally located around the same areas on Waldo's faces. Usually, this will be around the hat and also the spectacles. We can get the following images by running `show\_sift\_keypoints.py`

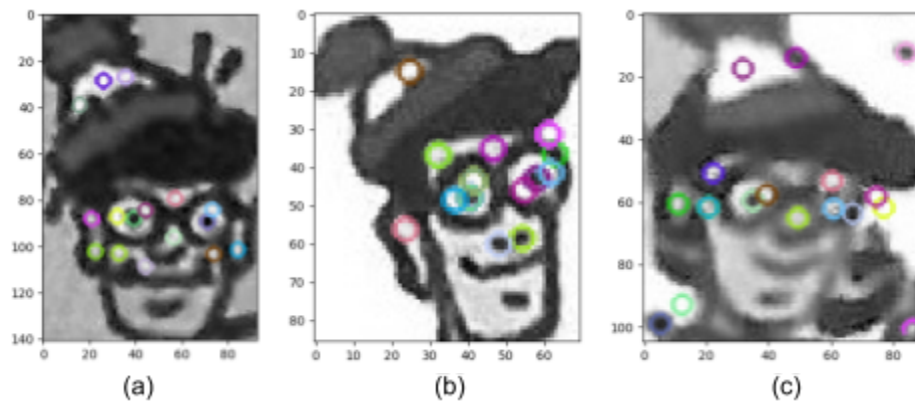


Figure 6: 20 keypoints with the highest response for Waldo's faces

The general flow of the implementation is detailed as follows (refer to `feature\_matching.py`):

1. Extract the 20 keypoints with the highest response from the template image.
2. Concatenate the 20 keypoints descriptors of the template image to form a 2560 ( $20 * 128$ ) dimension vectors. This is our template feature vector.
3. Slide the windows (3 scales) through the test image.
4. For every segment of the test image bounded by the sliding window, extract the feature vector (apply Step 1 and 2 to the test image segment).
5. Find the cosine distance between the template feature vector and the test image segment feature vector.
6. If the distance falls below a certain threshold (set to 0.45), we say that we have found the object.



What we found is that this method does not really work well for this project. For instance, for the cropped test image below, our object detector returns almost every part of the test image.



Figure 7: Cropped test image

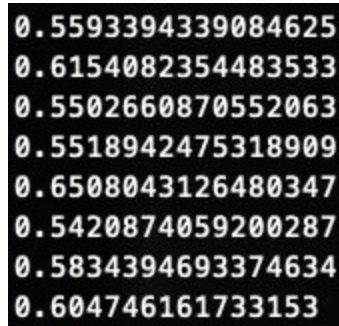


Figure 8: Some of the segments of the test image returned

Upon inspection, all the distances computed are very close to one another. Hence, it is very difficult to even set the correct threshold. There are several reasons why we think this happens:

1. The nature of the drawing is very similar. There are a lot of similar features distributed all over the test image.
2. The concatenation of the keypoint descriptors is not robust. It is very hard - if not impossible - to pinpoint exactly what keypoint descriptor comes earlier and which comes later. This means that we may be comparing 2 very different keypoints descriptors. For example, in Figure 6(a) above, the purple keypoint may have the highest response; and hence is located at the front of the test feature vector. For Figure 6(b) (the template), the keypoint with the highest response may be the blue one; and hence, it will be located at the front of the feature vector. In this case, in essence, we are comparing two unrelated features.
3. There is no objective way to choose the template image. We did it through observation and trial and error.





```
0.5593394339084625
0.6154082354483533
0.5502660870552063
0.5518942475318909
0.6508043126480347
0.5420874059200287
0.5834394693374634
0.604746161733153
```

Figure 9: The distances computed for every segment of the test image

Owing to the unpromising results obtained even from the cropped image, we did not carry the experiment further. Note, however, that our code works for Wenda and Wizard as well.

### **3.2.2 Proposed Solution 2: HOG+SVM**

With the unsuccessful attempt to find Waldo using feature matching, we tried to train classical Machine Learning models on HOG feature vectors built from the training data. We trained the various classifiers, such as Random Forest and Support Vector Machines before deciding on the one with better accuracy. Then, we tuned the hyperparameters to improve our results even further. Later, using a sliding window approach, we were able to identify Waldos, Wendas and Wizards.

Our reasoning for using HOG was that feature descriptors would represent the image or the image patch that simplifies the image by extracting useful information and throwing away the extra information. This would let us capture the important information in the required subject's face. We had to take into consideration several limitations of the following implementation including the speed of running and its tradeoff for accuracy. Hence we decided to again use a grey-scale image as opposed to a multi channel colour image as this wouldn't drastically change the accuracy.

This let us create a feature vector of size 5832. We trained our model with these feature vectors as our input data using Support Vector Classifier. Our initial tests pointed towards Support vector Classifier over Ensemble Classifiers as it ensured higher accuracy. We believe this SVC had a better accuracy as a clear boundary could be established between various test cases and also because of the limited number of examples compared to the number of parameters of the feature vector.

But even this method led to a huge number of false positives. While we were able to identify Waldo most of the time, this also led to a huge number of other people and objects being classified as Waldo, Wenda or Wizard. Apart from this, each image took almost an hour to run because of the huge number of windows we had to evaluate with the sliding window approach. This approach gave us an initial mAP of around 0.01. Considering the lack of a huge data set, we tried to augment the number of images through extra manual annotation of the characters. We even tried to flip and rotate the images in small angles to increase the number of training images for better accuracy. But this did not vastly improve the accuracy

and was taking longer time. Hence we decided to find a way to minimise our scope of search while working on improving our classifier.

### 3.2.3 Proposed Solution 3: Haar Cascade

An improvement we made to the classifier was to use a different kind of object detector. We felt HOG feature descriptors were unable to capture spatial embedding when we capture the feature embeddings. Also we were only taking into consideration low level features and hence there could be a lot of false positives. In order to overcome these two issues we decided to use a Cascade object detector which came with a several pretrained set of classifiers designed to identify faces and upper body. We also realised Waldo's face did not change it's aspect ratio significantly as it was always looking in one direction and always faced forward.

Hence we tried to implement Cascade Classifier which was an ensemble of pretrained weak learners. These weak learners provided the base over which the weighted average was taken to make a decision on whether an image was a desired character. In order to improve our accuracy here, we even increased the number of positive and negative examples by manually annotating them. We were able to get a much better accuracy of finding Waldo, Wenda and Wizard as shown in the pic below.

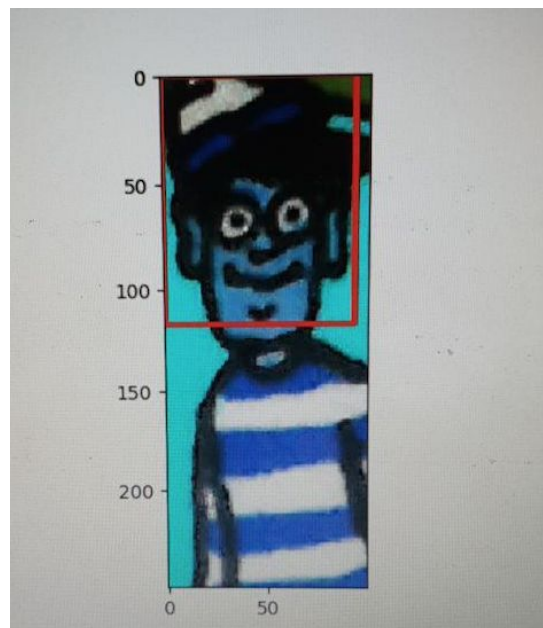


Figure 10: Finding Waldo using Cascade Classifier

The tradeoff with this approach was the training time. Training the cascade classifier took roughly 2 hours, given 100 positive images and 400 negative images. We still wanted to improve our model even further and hence combined it with an interesting in house developed idea like Minimizing scope of search with Colours.

### 3.2.4 Proposed Solution 4: Minimizing Scope of Search with Colours

#### 3.2.4.1 The Rationale for Using Colour

It seems that from both the HOG SVM and Cascade methods we explored, both learners were unable to detect with colour as they used grayscale input images. As a result, a lot of the false positive results include images with no red or white, which obviously would not contain Waldo, Wenda or the Wizard.

Furthermore, for HOG SVM, the process of sliding a window through the image was very long. For a small image with dimensions 2000x2000, it took around 5 minutes to slide the window through. The window was of size 100x100 and a step of 10 pixels along the height and width was taken. The time taken seemed to increase exponentially when applied to images of

Hence, there is a need to filter through the false positive results using colour as a metric.

#### 3.2.4.2 Using fixed filters

The initial attempt was to use fixed patterns to find Waldo and Wenda's shirt.

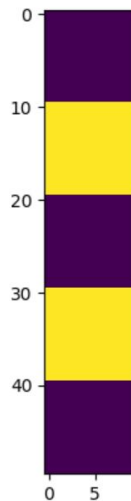
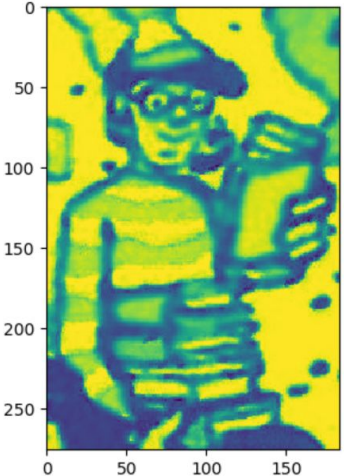
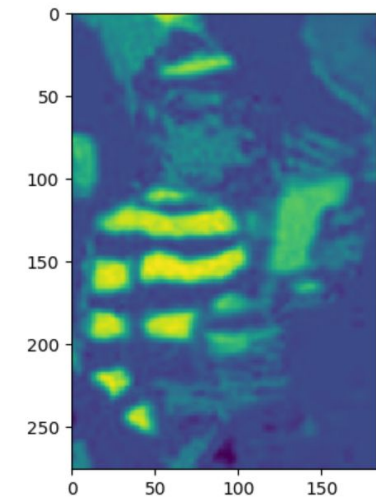


Figure 11: Pattern (kernel) used to find Waldo's Shirt

Note that the purple regions had values of -1, while the yellow regions had values of 1. The pattern is essentially a numpy array of 1s and -1s.

The red regions of the image of Waldo is then convolved with the pattern seen above. Regions where Waldo's shirt were present would then have a much higher value than other regions.

To extract out the red region, simply taking the red channel would not do.

Code	Resulting Image	Comments
<pre>import matplotlib.pyplot as plt import cv2  img = cv2.imread(&lt;path to image&gt;)  red = img[:, :, 2].astype(float) plt.imshow(red) plt.show()</pre>		<p>As seen from the image on the left. Simply taking the red channels was not a correct representation of the red regions of the image.</p> <p>Note that the dark regions show low values (on a scale from 0-255). Bright yellow regions show high values (close to 255).</p>
<pre>import matplotlib.pyplot as plt import cv2  img = cv2.imread(&lt;path to image&gt;)  red = img[:, :, 2].astype(float) gray = cv2.cvtColor(img, cv2.COLOR_BGR_GRAY).as ty pe(float)  plt.imshow(red - gray) plt.show()</pre>		<p>By taking the grayscale value of the original image, and subtracting it from the red channel, the true red regions of the image can be seen.</p> <p>From the image on the left, it is clear that only the red stripes on Waldo's shirt and hat have high values. The book he is holding orange, hence will also have a slightly higher value.</p>

The method showed positive results, being able to detect Waldo's shirt only in certain images. Naturally, pictures of Waldo in different sizes then posed to be a huge problem and different sizes of the pattern were required.

After initial experimentations, the method was abandoned as it took too long to determine a good size for the pattern. Furthermore, it took too long to process large images with this method.

### 3.2.4.3 Using The Magical CV2.bitwise\_and()

Further experiments were done using the red channel. Capturing the red regions of the image was simple. However, capturing other colours proved to be challenging.

The character Waldo has 3 main colours, red, white and beige (skin). Experiments were done to determine whether using these 3 colours could effectively determine Waldo's location. However, it is not as trivial as simply taking the red channel as seen in the previous example.

Masks were designed to then be able to extract colours from the cropped Waldo images.

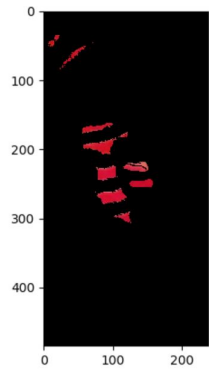
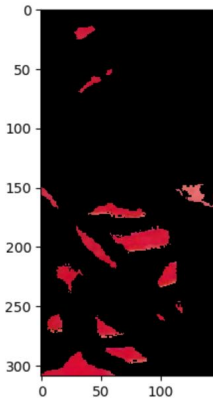
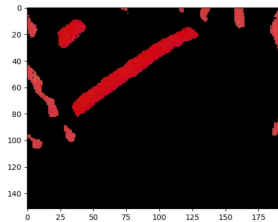
Firstly, the image is converted into the HSV colorspace. Then a mask is used to capture all the pixels that lie within the range of HSV values stated. Finally, the result is given by taking a bitwise operation of the original image with the mask. An example of the code is as shown below.

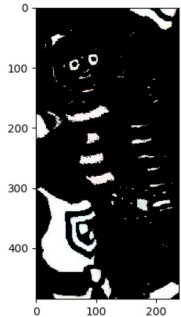
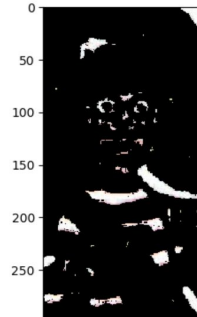
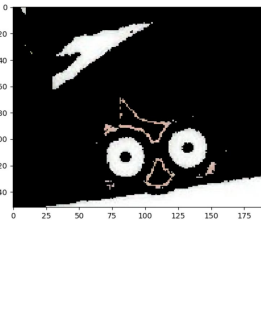
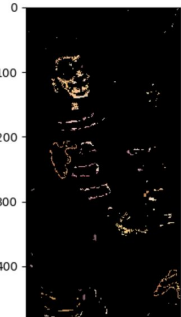
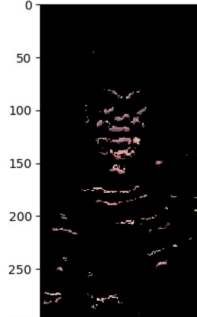
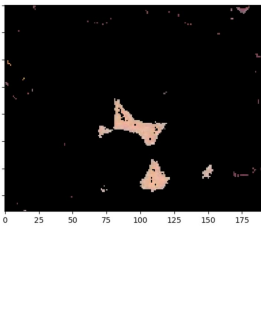
```
hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Range of HSV values to be used
w00 = (175, 70, 255)
w01 = (0, 0, 200)
mask = cv2.inRange(hsv_img, w01, w00)

result = cv2.bitwise_and(rgb_img, rgb_img, mask=mask)
plt.imshow(result)
plt.show()
```

Multiple masks could be joined together to capture variations of the color red, white and beige. The range of HSV values had to be manually determined. For Waldo's skin colour, 13 masks were used.

Filter	Image1	Image2	Image3
Red			

White			
Skin			

Extracting the regions of red, white and beige gave some very interesting results. However, we realised that Waldo's skin was not always beige. As seen in the following image, there are many times where Waldo's skin appears to be too pale, resulting in the white filter actually picking up his skin color.

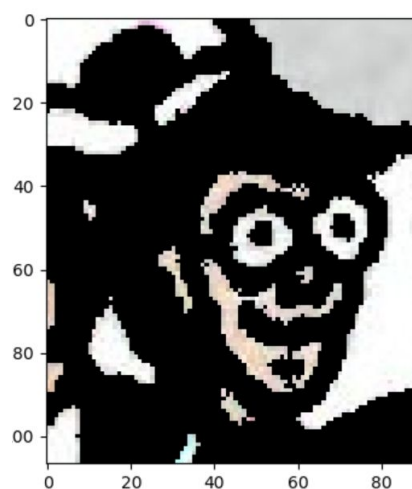


Figure 12: White mask picking up Waldo's skin colour

It can be seen that Waldo's skin is not completely white, however the white mask still captured his face. This is due to the white pixels on Waldo's shirt not being completely white as well. In many cases, the red

pixels and white pixels mix together. This is not visually noticeable however this happens to be the case for a lot of the Waldo pictures where Waldo is really small (Smaller than 50x50).

Hence, the skin mask/filter was abandoned as it overlapped with the white filter.

#### 3.2.4.4 Using sum of Red and White pixels

Using a sliding window, we then scanned through the whole region of the image, rejecting windows where pixel counts of red and white were below a certain threshold.

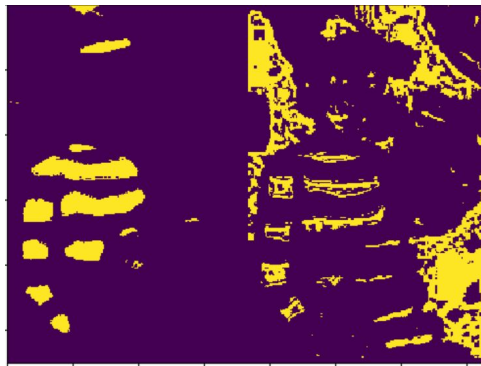


Figure 13: Left: Resulting image with red mask, Right: Resulting image with white mask

For regions where Waldo was present, the sum of red and white pixels were much higher. Hence this method was able to capture Waldo. However, there are many false positives as well.



Figure 13: Left: False positives in using red and white pixels

As seen in the example images above, the red and white pattern clearly did not resemble Waldo's striped shirt. Merely counting the pixels was not enough.



### 3.2.4.5 Using Warp Affine and Bitwise

Given that the red and white regions of the image could be extracted, there is a need to determine if the red and white regions were stripes. A trick to doing so would be to shift the red region up and down by a few pixels using a warp affine function in OpenCV and then using the bitwise AND operation.

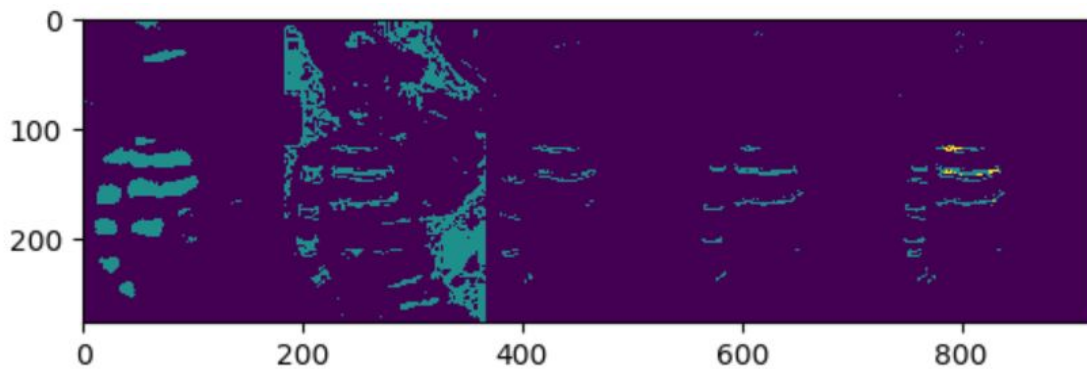


Figure 14: From the left - Red region, white region, red region shifted up and bitwise with white, red region shifted down and bitwise with white, final sum of previous 2 images

As seen from the above images, shifting red regions upwards and downwards by a few pixels (7 pixels seem to work well) and doing a bitwise AND operation with the white region managed to clearly identify the stripe patterns of Waldo's shirt, as well as removing the irrelevant white background in the image.

Following that, the red region was also shifted left and right. This is because for stripe patterns, a shift towards the left and right would not allow it to intersect with the white region. Hence, images with high values after the left and right shift would not follow the stripe pattern.

Finally, after applying this method to the whole image, the red and white patterns could be successfully extracted.

By taking a sliding window of 50x50 across the multiple images with a step size of 50, it would result in 27720 windows (Taking the largest image with the size 7015x9921). However, with the method of finding Waldo's shirt through the red and white patterns, only 16 windows remained (For the first image, 000.jpg found in the given datasets\JPEGImages folder).



Figure 15: Resulting image. Windows that were rejected have been blacked out



Figure 16: Resulting images. Waldo's shirt remains!





Furthermore, the method is fast even when taking a sliding window through the JPEG image of size 7015x9921, taking approximately 5-7 seconds. For certain images, only 4 windows remained, and Waldo's shirt was found.







### 3.3 Results

#### 3.3.1 Results for Cascade + Minimizing Scope With Colours

By combining methods found in 3.2.3, Haar Cascade and 3.2.4, Minimizing Scope of Search with Colours, the results were extremely good. However, for images where Waldo appears small (smaller than 50x50) the classifier did not manage to find Waldo.

List of JPEG Images where Waldo was found (Note that there are a lot of duplicate images. Actual results are much better!):

JPEG Image	Results (All the pictures the Classifier thinks is Waldo)
000	
005	
007	
008	

010	
011	
014	
017	
020	
021	

The classifier is able to find Waldo from the JPG images - 000, 002, 005, 007, 008, 010, 011, 014, 017, 018, 020, 021, 022, 025, 026, 027, 029, 033, 037, 042, 043, 044, 048, 060, 061, 062, 063, 068, 069, 075, 077, 078.

It is very possible to find Waldo for more images but more time would be needed to improve the Cascade classifier.

### **3.4 Discussion**

The section serves as a summary of the performance of all of our proposed solutions.

For feature matching, the detection can be done faster than using a classifier. However, it is still slow because the sliding window needs to traverse the entire image. While it is relatively easy to implement, However, the result is highly inaccurate. It returns too many false positives. We question the robustness of the feature vectors generated as well as the methodology in choosing the appropriate template to be used as the detector. We further suspect that the nature of the images in the illustrations (test images) are too similar, making it difficult for our objects of interest to have distinctive features.

For the classical Machine Learning approach, we realised that combining it with our novel filter approach drastically reduced the time for searching and testing the dataset. It also improved our accuracy of finding Waldo and Wenda. Nevertheless, this still has a lot of false positives owing to lack of sufficient data and lack of more powerful feature extractors. Our feature descriptors based on HOG are designed for low level feature extraction and may not be suited for face detection. This may be the reason behind the many false positives detected.

In order to improve the power of features extracted, we implemented the Haar Cascade Object detector. Being specifically designed for recognising faces and being an ensemble of different weak learners, it drastically improves the performance of the model. This, however, comes at the expense of the increase in training time for each character. Multiple fine-tuning attempts are also required. In order to overcome these limitations, we tried to combine our filter approach with Haar Cascade Object detector. This new addition resulted in a faster and a better model. It not only reduced the time taken to finish the testing due reduced points of check but also improved the accuracy. This in fact became our final model.

### **4 Conclusion**

In this project, we have explored several common object detection approaches. We have shown that while the performance of each approach is certainly poor, we attribute this to lack of sufficient amount of training data and necessary limitations imposed on the implementation. But regardless, we have clearly shown that we can filter the test image to direct the detector window to focus on certain regions in the test image with high probability of finding the objects of interest. This is possible because Waldo and Wenda have distinctive features that we can leverage on and that are absent from the rest of the test image (background). There is also a significant improvement in the time taken to detect objects of interest as the detector window no longer needs to traverse the entire test image.



## 5 Group Information

Member	Student ID	Email	Contribution
Calvin Tantio	A0160601X	calvin.tantio@u.nus.edu	<ul style="list-style-type: none"><li>- Data inspection (distribution and visualisation)</li><li>- Feature matching</li><li>- HOG + SVM classifier</li><li>- Test result to test file conversion (for evaluation)</li><li>- Report</li></ul>
Gadupudi Mukesh	A0161426L	mukesh.gadupudi@u.nus.edu	<ul style="list-style-type: none"><li>- HOG + SVM/Random forest</li><li>- Combining Color filtering with HOG</li><li>- mAP implementation</li><li>- Report</li></ul>
Koh Yong Hong Lawrence	A0155174E	e0031311@u.nus.edu	<ul style="list-style-type: none"><li>- Data extraction</li><li>- Template matching</li><li>- Feature matching</li><li>- Minimizing scope of search with colours</li><li>- HOG + SVM classifier (Tried multiple variations. Detecting Waldo vs Wenda vs Wizard vs Neg, Waldo face vs non-Waldo face)</li><li>- Cascade classifier</li><li>- Report</li></ul>

## References

- [1] Lowe, David. (2004). Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision. 60. 91-. 10.1023/B:VISI.0000029664.99615.94.
- [2] Dalal, Navneet & Triggs, Bill. (2005). Histograms of Oriented Gradients for Human Detection. IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2005). 2.
- [3] Evgeniou, Theodoros & Pontil, Massimiliano. (2001). Support Vector Machines: Theory and Applications. 2049. 249-257. 10.1007/3-540-44673-7\_12.
- [4] Padilla, Rafael & Filho, Cicero & Costa, Marly. (2012). Evaluation of Haar Cascade Classifiers for Face Detection.