CS231A Find Mii Project

Allan Joshua
Stanford University
allan84@stanford.edu

Dylan Shinzaki
Stanford University
shinzaki@stanford.edu

Abstract

This paper details an attempt to automate the Find Mii challenge. We use several techniques like face detection, face recognition and other computer vision algorithms. We attempt to construct algorithms that are capable of: identifying a given avatar in a group of avatars; identifying two look-alike avatars in a group of avatars; identifying an avatar whose behavior differs from that of the rest of the avatars in a group and identifying the fastest moving avatar in a group of avatars. Each of these tasks has three levels of difficulty easy, medium and hard where each of the levels pose unique algorithmic challenges.

1. Introduction

Recognizing particular objects in a scene, finding the odd ones out of a group and isolating the fastest moving object are all problems which extend to several real world applications. The objective of this project is to leverage these and build an agent to automatically play the popular video game Find Mii. The goal is to accomplish several tasks at increasing levels of difficulty by performing a set of operations using faces of Wii digital avatars (colloquially called a "Mii") extracted from a video stream by processing the fewest number of frames as possible. There are several problems which manifest themselves as the level of difficulty increases which ought to be solved efficiently. Scaling, translation, rotation, occlusion and motion detection are a few worth mentioning. In this paper, we detail the way we propose to solve each of these problems. We split the video stream as a sequence of frames and use several computer vision techniques to perform these tasks. We use Matlab and OpenCV for the implementation of these algorithms.

1.1. Find Mii

To successfully play Find Mii we must accomplish four tasks at varying levels of difficulties which comprise of : Identifying a particular face in a scene, identifying a pair of similar avatars from a group, identifying the odd avatars out of a group and identifying the fastest moving avatar. There are three levels of difficulties for each task which introduce new challenges like rotational variance, partial occlusion and scaling. Our System effectively solves ten of the twelve tasks. The technical approach we took will be explained in the following sections.

2. Background and Related Work

It was beneficial to use a few off-the-shelf implementations available and customize them for the project needs. The ***Find this Mii*** task could be accomplished by using Scale-invariant Feature Transform (SIFT) [1] and a derivative approach using Speeded Up Robust Features (SURF) [2]. SURF is a high-performance scale and rotation-invariant interest point detector and descriptor claimed to approximate or even outperform previously proposed schemes with respect to repeatability, distinctiveness and robustness. It relies on integral images for image convolutions to reduce computation time, builds on the strengths of the leading existing detectors and descriptors (using a fast Hessian matrix-based measure for the detector and a distribution-based descriptor). The ***Find 2 look-alikes*** task requires a face detection algorithm. Multiple approaches exist for detecting human faces. Viola and Jones accomplished human face detection using Adaboost and Haar wavelets [3]. Rowley, Baluja, and Kanade used neural networks [4]. Mii faces differ from human faces in some aspects. More generic object detection approaches could be applied to detect face-like objects such as Mii faces. [5, 6] proposed a method for training-free object detection based upon locally adaptive regression kernels (LARK) to detect novel objects which are similar to an image of a query object. Since the ***Find n odd Miis*** and the ***Find the fastest Mii*** tasks involve motion detection, sparse optical flow [7] was chosen and to pick features for optical flow the Shi and Tomasi, good Features to track [8] algorithm was used.

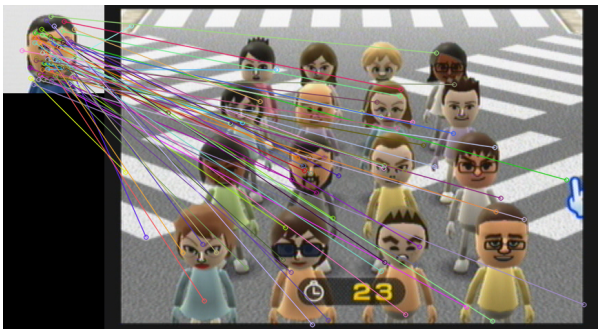

Figure 1. Example Feature Matches

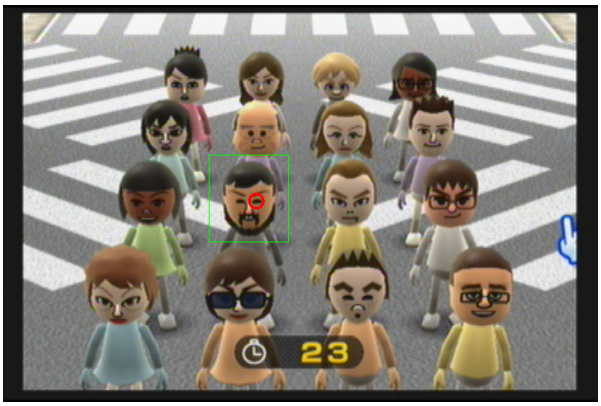

Figure 2. Hough Transform on the accumulated points leads to a good click but the solution does not generalize well.

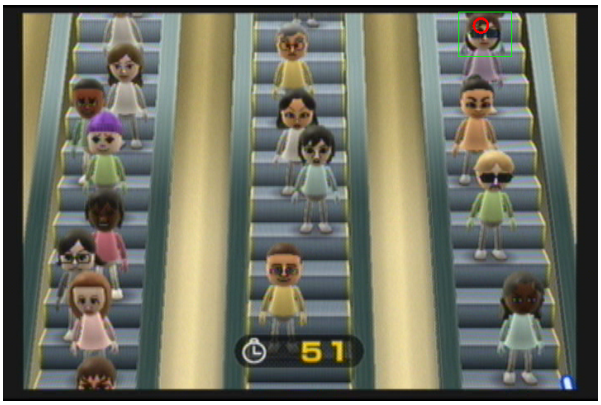

Figure 3. Final Click for Task1 Level2.

Figure 4. BackGround Subtraction.

3. Technical Approach and Experiment

3.1. Task1: Find This Mii

We propose to use the SURF interest point detector / descriptor with the intent of matching the avatar on the reference image to the specific location of the same avatar on the extracted video frame. The local features extracted would be robust to scaling, translation, rotation and occlusion. However, considering the number of interest points detected and the fact that the difference in the gradient along the *x-axis* and the *y-axis* for all of the Miis faces being quite similar, the good matches even after applying a threshold were still not discriminative enough to be confident to click. Although the number of feature matches as shown in figure 1 might suffice to pick the correct Mii by running a hough transform [9] in this case (see figure 2), it did not generalize well. On trying out other reference images on the same video, the algorithm performed poorly. To combat this, we loop through each of the frames up to a specific *threshold* and keep accumulating the good feature matches. Once we are confident that we have enough frames, we run Ransac [10] and compute the homography matrix. Once we have the homography matrix, we compute the center of the reference image and find the corresponding point on the frame which is our click point. Although the number of features on the video frame is quite large OpenCV's FLANN based matcher (Fast Approximate Nearest Neighbor Search Library) was very effiecient in finding the good matches.

This approach worked well for both levels 1 and 2 (refer figure 3). There were several interest points being picked from the background on level 3. The matching of interest points was so noisy that it was really hard to get good matches. One way we tried to combat this is to apply background subtraction and pick only the foreground. Refer figure 4.

After applying Background subtraction we compute the SURF descriptors of this image and compare this with the descriptors of the other image. We continue to pick multiple frames with the background removed and eventually run Ransac to compute the homography matrix. This brought down a lot of false matches.

3.2. Task2: Find 2 Look-alikes

The high level approach of solving this task is described below.

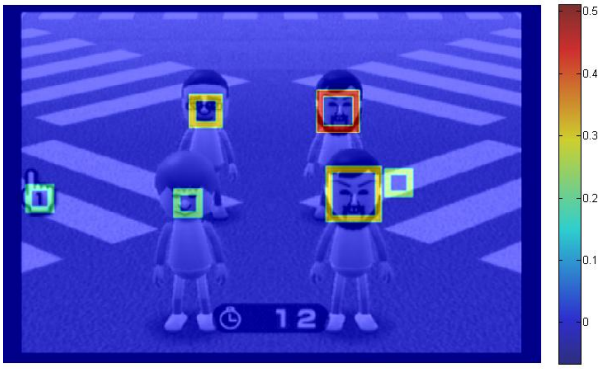

Figure 5. Example face detection with LARK detector

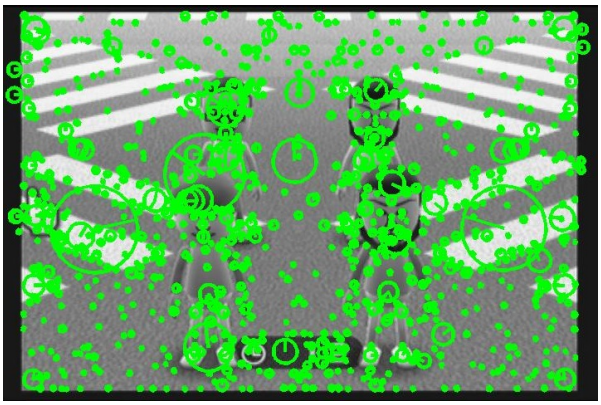

Figure 6. Example SIFT features

1. Using a face detector to locate the possible faces
2. Use SIFT to generate descriptors for the facial regions
3. Use the SIFT descriptors to match areas find the most similar facial regions.

3.2.1 Facial detection

The facial detection portion of the algorithm used a training-free generic object detector based upon LARKs. [6] and [5] proposed such an object detector and provided code to implement the detector. Given a query image, the detector identifies similar objects. For this task, an image of a human face was included with for the purposes of face detection. It also had good results when applied to scenes with Mii faces. An example of this can be seen in Figure 5.

This approach has the advantages of requiring no prior training and limited assumptions with regards to the Mii's appearance

3.2.2 Features

The SIFT portion of this task was accomplished using the MATLAB version of the VLFeat library [11]. This library was used to extract 128-dimension SIFT descriptors from each frame in the input video. The descriptors in a given frame were grouped based upon the detected face boxes. Any descriptor that was not located in a box was eliminated. An example of this step can be seen in Figures 6 and 7.

3.2.3 Heuristics

The approach described above acted as the basis for execution of Task 2 at all three levels of difficulty. A number of heuristics were added to improve the performance at higher difficulties.

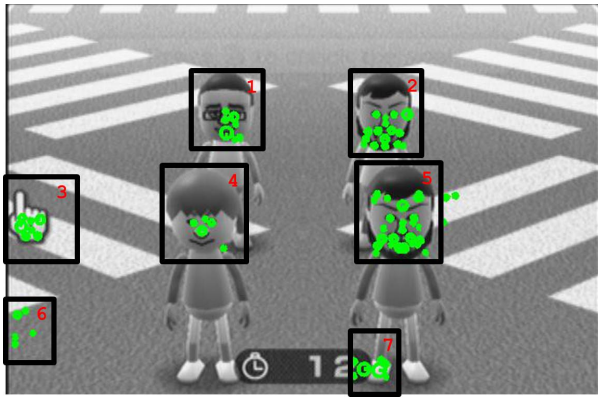

Figure 7. Example SIFT features with the face boxes

One heuristic is based upon the assumption that the movement of the Mii faces between frames was small compared to the size of the head and ground truth associated with it. For small n, groups of n consecutive frames could be treated as if they were the same frame. This expanded the previously described approach by allowing for the matching of bounding boxes that are not in the same frame. For example, it is possible to match between bounding box i in frame t with bounding box j in frame $t+n$ when n is small. Face detection boxes and SIFT descriptors were extracted from n consecutive frames. For every bounding box pair i and j in the n frames, the SIFT features matches and the similarity measure were calculated in the same way. To assure that the matching is not taking place on the same face in different frames, a bounding box pair had to be at least m pixels apart. $m = 50$ was used in the code.

This heuristic was useful because it provided more possible matches for a bounding box. This made the feature matching more robust to possible disruptions such as a complicated background or low Mii face quality. Figure 8 shows an example output for the original, 1 frame algorithm for level 3. Figure 9 shows an example output with the heuristic

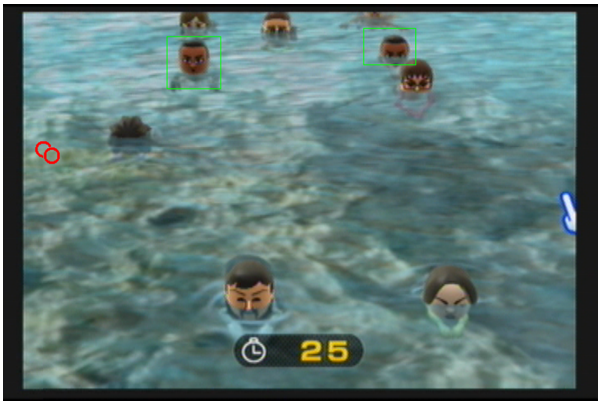

Figure 8. Output for frame groups of size 1

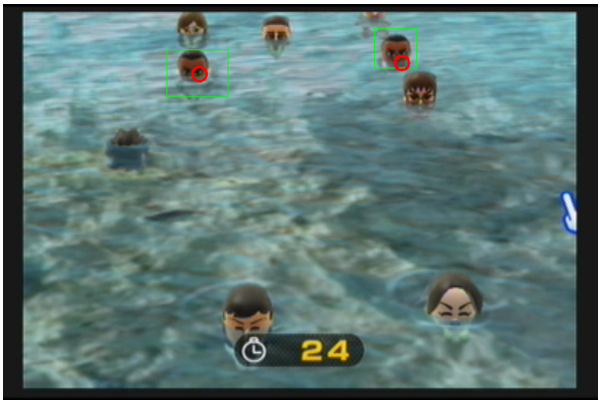

Figure 9. Output for frame groups of size 4

when $n = 4$. A downside of this heuristic is that it generally required more frames to operate. At a result, it was deemed unnecessary for level 1.

A second heuristic was used to limit false positives. Any match found by the algorithm was considered a candidate click. A candidate click would only be returned if k "similar" candidate clicks had been found in the past. The code used $k = 1$ and classified similar clicks as being within 100 pixels. This increased the number of frames consumed but greatly reduced the false positive rate. This was deemed a good trade off given the penalty of an incorrect answer versus the penalty of using more frames.

3.3. Task 3: Find n odd Mii's out

The problem in task 3 is to identify the odd Miis out from a video stream of Miis. The oddity in movement involves the Miis turning or moving their heads, where a few Miis are turning or moving their heads in the opposite direction in comparison with the other Miis. Since this task involves detecting motion, we chose to use the optical flow method.

Figure 10. This image represents an active frame in an active window. Note that the majority of interest points are move in one direction and only a very few are oriented towards the opposite direction.

We use the sparse optical flow [7] method in OpenCV. Optical flow attempts to detect motion within a scene by tracking individual features across frames. We use sparse optical flow instead of dense optical flow because of the increased efficiency provided by sparse optical flow. OpenCV provides an implementation of sparse optical flow, as well as a function that returns good features to track [8] for a particular frame.

In order to solve task 3, we apply optical flow to compare each frame to the frame two frames ahead of it. The optical flow algorithm, combined with the good-features-to-track method, yields a set of point correspondences between the two images. We then filter the point correspondences, removing pairs of points that are too far apart, which signals an error in the optical flow results and pairs of points that are too close together, which signals a lack of movement. We use the points to compute the angle of displacement. Since we are given a biased assumption that the Mii's turn their head horizontally, we pick only the points which have been displaced horizontally. This thresholding specifically picks the points that we are interested in.

One of the key observations is that the there is a period of activity which we define as an "active window". The Mii's move all at once and then stop for a while. After the period of inactivity, they again move all at once. We leverage this to pick points during this "active window". The thresholding that we have done would help with picking the specific feature points which move. During an "active window", since all of the Mii's turn their head, there would be a much higher number of feature points which would pass through our filter in comparison to an inactive frame. An active window might last just a few frames. During this period most of the feature points would be moving in one direction and

Figure 11. This image represents the accumulated points at the end of an active window. These are the most likely locations that could be clicked.

Figure 12. The feature points on the darker Mii's not being picked up.

a very few would be moving in the opposite direction. The fig 10 shows a snapshot of Task 3 Level 1 on an active frame in an active window.

We do not just pick the points from one frame. We accumulate the points throughout the "active window". Our stop criteria is when the number of feature points drop drastically. This sudden drop in feature count which satisfy our filter would indicate that the video has gone back into an "inactive window". At this juncture we will have accumulated the most likely points which could be clicked. We only accumulate the points which were moving in the opposite direction.

Figure 11 represents the accumulated points at the end of an active window. At this time we have the most likely points that could be clicked.

The original plan was to use a clustering algorithm to find the cluster centroids. Since we know the number of clicks, initializing of K-means would not be an issue. K-means, however, was not very robust to outliers and the noise in this case was way too much for it to come up with the proper cluster centroids that we wanted.

The next option that was tried was a mixture of gaussian model expectation maximization but this did not give proper results either. Mean-shift was considered but we realized that it was not going to give the proper output either. The final solution was a hough transform in a 2d space. Hough transform was very robust to noise and since most of the votes would be for the most likely Mii faces, it came out with the proper click points.

The algorithm as a whole was very abstract that it was easily portable to Level2. One of the problems which was faced during the implementation of level 2 was the fact that feature points on darker Mii's were not getting picked up. It was intuitively obvious that the gradient change was so minimal that those points were not picked up. Figure 12 depicts this. The way this was solved was by parameterizing the good-features-to track algorithm to pick 10 times more features than it originally did and adjusting the threshold accordingly. Both levels 1 and 2 worked well.

We were not able to come up with an algorithm for Task3 Level 3.

3.4. Task 4: Find the Fastest Mii

The stated problem in task 4 is to identify the fastest moving Mii among a group of Miis within a video stream. As in task 3, we approach this problem using optical flow. Unlike task 3, where detecting motion over a short time is sufficient, in task 4, it is necessary to detect continuous motion over a long period of time. The reason for this is that many objects within the scene, such as the Miis hands, could be moving faster than the fastest Mii over short periods of time. Unfortunately, the optical flow algorithm is not very robust when calculated between two frames that are far apart in time. Therefore, we try to track features across time by tracking them between adjacent frames using the optical flow algorithm.

One of the initial thoughts was to follow the points. We intended to use the good-features-to-track algorithm [8] to pick the points in one frame. We then intended to get the new locations (the ones these points would have traveled to, after one round of forward optical flow) and use those as the features to send into Lucas kanade [7] to compute the flow for the next frame which would give the locations on the frame after that and so on. Doing this would help in figuring out how far a particular feature point had traveled.

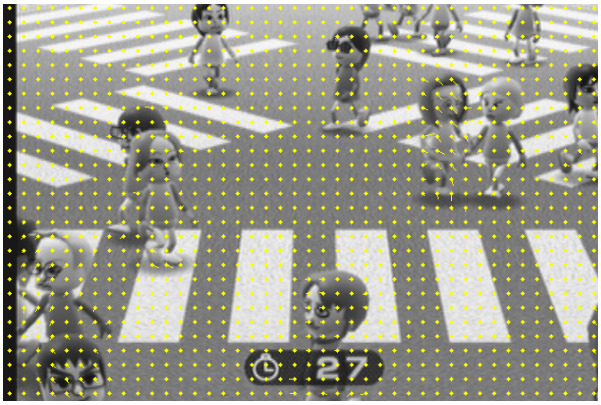

Figure 13. Velocity vectors of all of the points.

We could then use that information to figure out the points which have moved the farthest.

The biggest disadvantage of this method is that with optical flow, it is not guaranteed that we would find a point on the next frame. The error level was way too much that it was hard to follow the points over just a few frames. We could trace it as much as we can and then resort to getting the average based on the number of frames. But the problem with this approach is that we would not know which point to click. Although we might know which frame this feature point is at, it might not be the last frame. Since we cannot access any frame after clicking, this method did not seem like a viable solution. One other alternative was to use the Camshift algorithm to track the group of points which are close to each other and moving the fastest. We could figure out which set of points have moved the most and then try to apply Camshift to track it along to obtain the click point. This did not end up working well.

The other alternative we tried was to use the dense optical flow to get the velocity vectors. Fig 13 shows the velocity vectors of all of the points (not just the feature points). This alternative was promising because the features picked across different frames were not the same and there is a possibility that a feature picked in this frame will not get picked in the next one by the Shi and Tomasi algorithm [8] . Although this seems like a good option we still resorted to solving the problem using sparse optical flow.

We used an approximation. We segmented the image into small boxes as shown in Fig 14. We start off in a frame and use the Shi and Tomasi algorithm [8] and run optical flow. We pick the points by thresholding like we did for Task3. We maintain a cumulative belief net. The belief net will have the distance traveled by the feature points as values. From the second frame onward we use something very similar to a Markov assumption. The location of this feature point in this frame is only dependent on the location of this feature point in the previous frame and is independent of everything before it. If we are in the second frame, we pick feature points of this frame using the Shi and Tomasi algorithm [8]. We then use these feature points and track it one frame backwards.

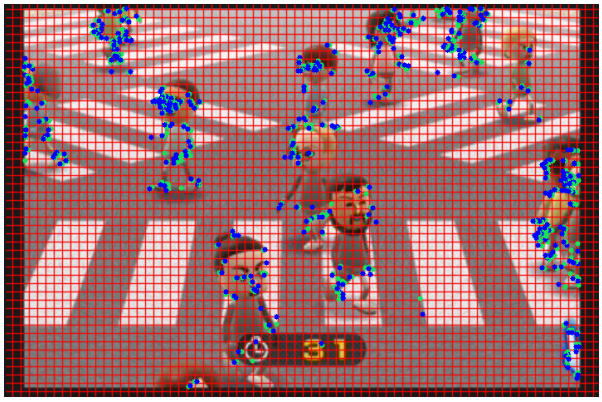

Figure 14. The green circles represent the current location and the blue circles represent the location that it was in, in the previous frame

This would give us the location of this feature point in the previous frame. Since we have discretized the image as small 10x10 grids, we can easily figure out which grid it was part of by binning it appropriately. We pick the value in that grid (which is the cumulative value of this feature point so far) add the value of how much this point will have moved to, in the next frame by running a forward optical flow and updating the belief net appropriately. We threshold the total distance we expect a particular point to have moved for us to make a confident prediction and then run a hough transform to select the click point from the list of candidate points which are above the set threshold. However, there were a few problems during implementation where features of the previous iterations were getting dropped off. If for some reason, the video is stuck or good-features-to-track does not return feature points that don't satisfy our filter, the cumulative distance so far would be dropped off. We combat this problem by maintaining an angle net, similar to the cumulative belief net. As and when a feature is picked, we compute the angle it will have moved to, and store that in the grid. As part of the update step in the algorithm we check the current belief net value and compare it with the expected bin computed by the use of the angle value of this grid. If the value is lower than the current belief net value, the belief net value is retained or moved to the appropriate bin. This helps with solving that problem. Tuning the threshold appropriately worked well for all 3 levels. Fig 15 shows the click for level 3

Figure 15. Click for Level 3

Future Distribution Permission

The author(s) of this report give permission for this document to be distributed to Stanford-affiliated students taking future courses.

4. Conclusion

This paper describes a system for automatically and effectively playing the Find Mii video game using computer vision techniques.While face detection seems like an intuitive tool to use for almost all these tasks, we found it to be inaccurate and often times not as helpful as it would seem. This may have been due to the strength of our classier or the nature of Wii faces. We found that avoiding use of a face detector when possible improved our success rate. Additionally, we found sparse optical flow and hough transform to be extraordinarily useful tools.

References

[1] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, pp. 91–110, nov 2004.

[2] T. T. H. Bay and L. V. Gool, "Surf: Speeded up robust features," *In ECCV*, p. 404 417, 2006.

[3] P. Viola and M. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, pp. 137–154, 2004.

[4] S. B. Henry A. Rowley and T. Kanade, "Neural network-based face detection," *Pattern Analysis and Machine Intelligence*, 1998.

[5] H. J. Seo and P. Milanfar, "Training-free, generic object detection using locally adaptive regression kernels," in *IEEE Trans. on Pattern Analysis and Machine Intelligence*, pp. 1688–1704, 2010.

[6] H. J. Seo and P. Milanfar, "Using local regression kernels for statistical object detection," in *Proceedings of IEEE International Conference on Image Processing (ICIP)*, pp. 2380–2383, 2008.

[7] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," *Proceedings of the 1981 DARPA Imaging Understanding Workshop*, 1981.

[8] J. Shi and C. Tomasi., "Good features to track," 1994.

[9] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Comm. ACM, 15, 1*, 1972.

[10] *Multiple View Geometry in Computer Vision (2nd ed.)*. Cambridge University Press, 2003.

[11] A. Vedaldi and B. Fulkerson, "Vlfeat: An open and portable library of computer vision algorithms." http://www.vlfeat.org, 2008.

5. Contribution

1. Task4, Task 3 Task1– Allan Joshua
2. Task2, Task1, FaceDetection Haar Training – Dylan Sinzaki.

Face detection was one of the tools which seemed like an intuitive tool to use. Although we have not explained about the details of that in this document, we implemented it and learnt that it was not as effective as the alternatives that we came up with. We had the algorithms work with the Face detector but the timings we achieved by not using them were way better.