Project 3: Real-time 2-D Object Recognition

Hardik Devrangadi devrangadi.h@northeastern.edu
Thejaswini Goriparthi goriparthi.t@northeastern.edu

Real-time 2-D Object Recognition

This project aims to perform 2-D Object Recognition, by processing the image sequence output from a webcam, to isolate objects placed in view on a white surface, and be translation, scale, and rotation invariant. This project incorporates the pre-processing of the incoming video stream by thresholding, cleaning by applying morphological features.

Since this project has been done by a group of two, Tasks 1 and 2 (Thresholding, Morphological Filtering) have been coded from scratch.

Setup

For consistent results, also so that the threshold value can be kept constant, a webcam is setup in a closed box with lighting, so that there are no external disturbances (shadows, light) at any point of the day when the program is run. There are two light sources, the overhead primary light located around the webcam as seen in Figure 1, and the secondary light source placed at the side as seen in Figure 2. A white background is attached

Figure 1: An overhead webcam is placed with a primary light source

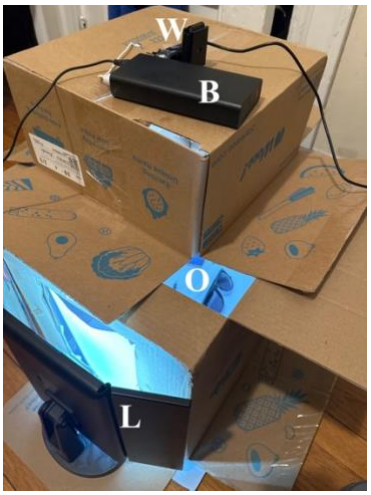

Figure 2: Setup (W-Webcam, B-Battery Bank, O-Object Surface, L-Secondary Light Source)

to the box on which objects can be placed. This is a foolproof setup, to get consistent results every time.

Tasks

Task 1: Threshold the input video

Since the webcam does not output high quality video, if the lighting conditions are not ideal, there is significant noise in the image. Before performing the thresholding on the video, a 5x5 Gaussian blur is applied to the image, to remove this inherent camera noise.

After blurring the image, the image is converted to greyscale. A threshold is then applied to the image. Initially, the output thresholded image is filled with ones i.e., it is made to be completely white. Then, thresholding is done by iterating through all the pixels, and each pixel is checked with the set threshold value. If the pixel is greater than the threshold, that pixel is made black. This gives us the separation between the foreground and the background.

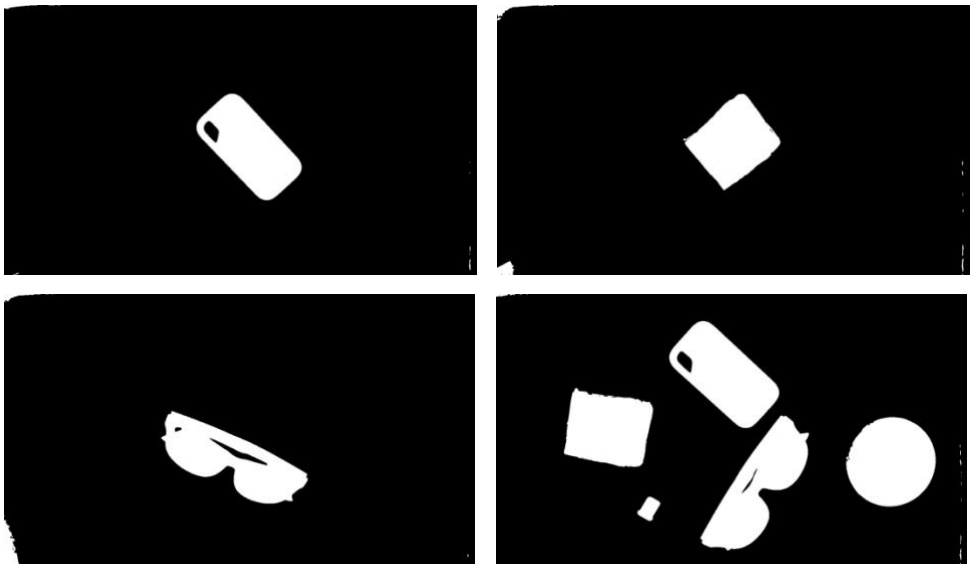

Figure 3: (clockwise from top left) Thresholded Images (a) Hard-disk (b) Cardboard Square (c) Multiple Objects (d) Sunglasses

Figure 3 shows some examples of the thresholded images after blurring and converting to greyscale.

Task 2: Clean up the binary image

After the binary image is obtained, it can be observed that there are white spots towards the corners and rough edges that interfere with the size and features of the image. To combat this, a morphological filter must be applied. To clean up the image, erosion of the image is performed, so that there are no unnecessary white pixels in the image. These erroneous pixels appear due to the edges of the white sheet background of the setup.

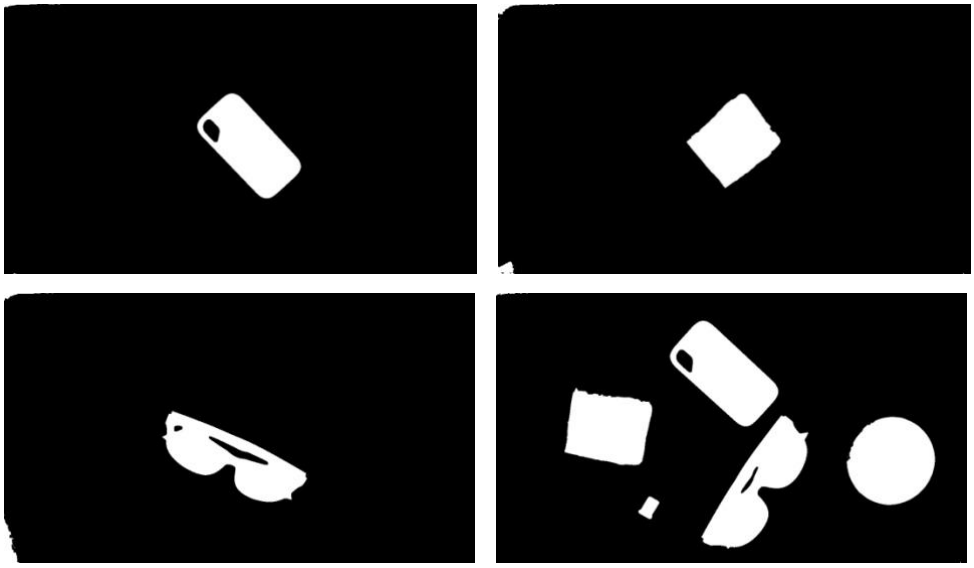

Figure 4: (clockwise from top left) Eroded Images (a) Hard-disk (b) Cardboard Square (c) Multiple Objects (d) Sunglasses

Figure 4 shows the images of the same examples after erosion has been performed to remove stray white pixels. The effect of erosion can be noticed on the bottom right of every image.

Task 3: Segment the image into regions

In this task, the image is segmented into regions by either using region growing or Two pass algorithm. For the connected components analysis of the binary image, the OpenCV inbuilt function cv::connectedComponentsWithStats is utilized which assists in identifying regions in the image. The various regions present in the image are identified and colored in with random colors so that they can be differentiated, as seen in Figure 5. These regions can be then analyzed to obtain their feature vectors.

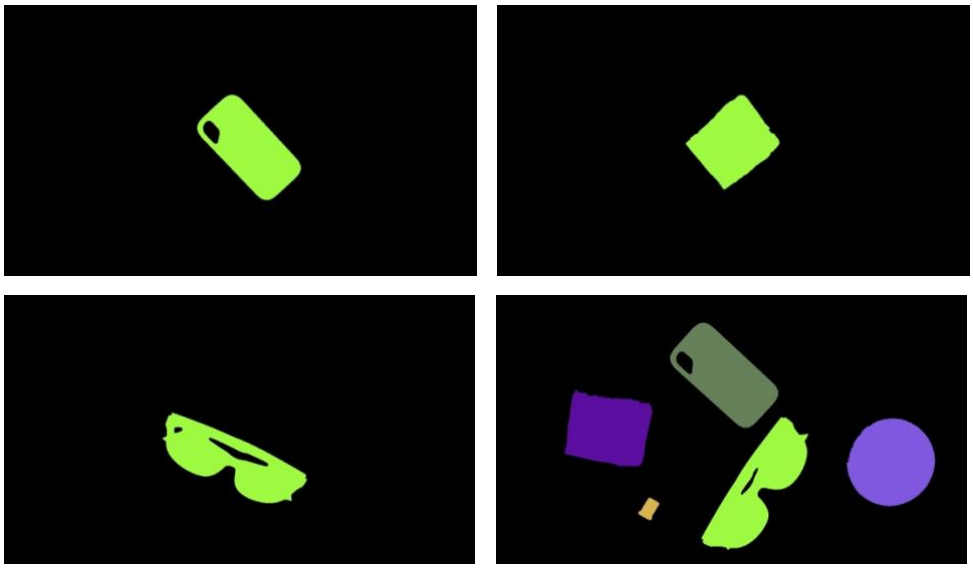

Figure 5: (clockwise from top left) Segmented Images (a) Hard-disk (b) Cardboard Square (c) Multiple Objects (d) Sunglasses

Task 4: Compute features for each major region

The extractFeatures() function extracts features from an input image using connected components analysis. The function takes two inputs - an input image and an integer value N, which denotes the number of largest regions for which feature extraction needs to be performed. Using the connectedComponentsWithStats function, the function labels all the connected components in the eroded image, computes the areas of all the components,

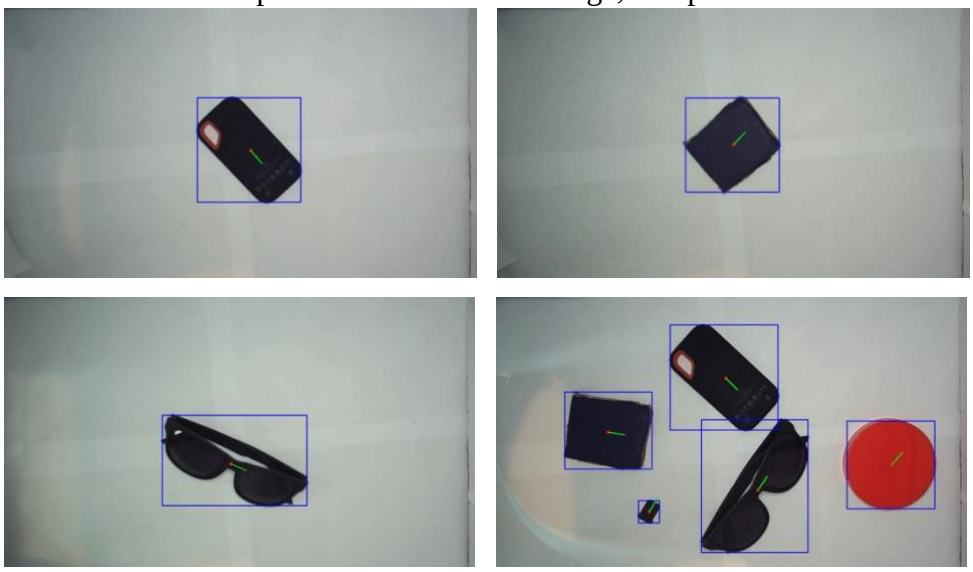

Figure 6: (clockwise from top left) Images with features and bounding box (a) Hard-disk (b) Cardboard Square (c) Multiple Objects (d) Sunglasses

and sorts them in descending order. Bounding boxes are computed for each region using the connected components using OpenCV's statistical methods. This method returns a statistical matrix for each connected

component or region in the image. The statistics matrix contains information about the coordinates, width and height of the upper left corner of the bounding box. These values are accessed using the cv::CC_STAT_LEFT, cv::CC_STAT_TOP, cv::CC_STAT_WIDTH, and cv::CC_STAT_HEIGHT flags. A bounding box is drawn on the original image using the cv::rectangle() function. A bounding box encloses the entire region as seen in Figure 6 which can be used to calculate properties of the maximum number of regions (N) in consideration.

Then, we use the cv::moments() and cv::HuMoments() functions to compute the moment feature vectors of the N largest regions. The moment features computed for each region include the percentage fill factor, and 7 Hu moments. These features are stored in a vector and returned as the output of the function. The OpenCV function cv::moments() used in the code computes the central moments of the region and is displayed on each object in the live video, while the cv::HuMoments() computes the Hu moments which is a set of seven moments that are invariant to translation, scale, and rotation.

Task 5: Collect training data

The system's training mode allows users to collect feature vectors from objects, apply labels, and save them to CSV files for later use in classifying unknown objects. A screenshot of the CSV file is shown in Figure 7. When the user presses the 'n' key, the system asks the user to enter the number of scans to collect the feature vector of the current object. The user is then prompted to enter a label for the object. This label is used later to identify the object during classification.

	A	B	C	D	E	F	G	H	I
1	tumbler	12.921	0.219459	0.0143187	5.92E-07	2.15E-05	7.60E-11	2.57E-06	8.47E-12
2	tumbler	13.2085	0.219048	0.0145665	1.98E-05	5.65E-05	1.89E-09	6.79E-06	-1.57E-10
3	tumbler	13.8254	0.209618	0.0144203	1.40E-06	5.99E-06	1.39E-11	7.15E-07	1.04E-11
4	tumbler	14.1593	0.19679	0.00969082	9.04E-06	1.78E-05	9.97E-11	1.67E-06	-2.02E-10
5	tumbler	13.4085	0.217819	0.0149753	2.47E-06	2.16E-06	3.10E-12	2.22E-07	3.91E-12
6	tumbler	13.4133	0.216401	0.0147857	1.02E-05	1.71E-06	6.06E-12	1.91E-07	3.71E-12
7	tumbler	13.5029	0.201083	0.011016	2.73E-06	2.95E-05	1.86E-11	3.06E-06	-2.65E-10
8	tumbler	12.641	0.224985	0.0156524	2.49E-06	1.09E-05	5.22E-11	1.34E-06	-2.31E-11
9	tumbler	12.9804	0.217458	0.014996	1.04E-06	3.26E-06	5.91E-12	3.88E-07	-9.09E-13
10	tumbler	15.6882	0.190716	0.00940834	2.78E-05	1.65E-06	6.85E-12	9.10E-08	-8.78E-12
11	wallet	8.39786	0.198218	0.00068531	5.01E-05	0.00016545	1.34E-08	4.14E-06	6.94E-09
12	wallet	9.83437	0.173564	0.00130905	1.72E-05	2.40E-06	-1.18E-11	-3.79E-08	-9.80E-12
13	wallet	7.99464	0.216312	0.00121314	8.20E-05	0.00024339	2.35E-08	7.77E-06	2.51E-08
14	wallet	9.12732	0.17333	0.00130734	6.03E-05	2.93E-06	8.54E-12	6.00E-08	3.80E-11
15	wallet	8.68411	0.204478	0.00116762	0.00011557	0.00018488	6.73E-09	3.76E-06	-2.62E-08
16	wallet	9.81705	0.169456	0.00078245	1.29E-05	1.86E-07	2.25E-13	2.09E-09	1.79E-13
17	wallet	6.73384	0.293761	0.00421637	2.38E-05	0.00020035	-1.21E-08	-1.19E-05	-6.67E-09
18	wallet	9.50563	0.174926	0.00159866	3.33E-05	3.41E-06	-2.68E-11	-5.80E-08	-2.46E-11
19	wallet	9.86063	0.173215	0.0006825	5.61E-05	5.42E-06	3.96E-11	1.41E-07	-8.57E-11
20	wallet	8.41884	0.188576	0.00204361	8.22E-05	2.57E-05	-1.18E-09	-8.20E-07	8.03E-11
21	clip	3.27089	0.167861	0.0011725	0.0001173	1.95E-06	2.95E-11	6.67E-08	-6.03E-13
22	clip	3.33696	0.182134	0.00331022	0.00021378	3.34E-06	4.07E-11	1.12E-07	7.95E-11
23	clip	3.6933	0.17222	0.0025686	0.00015031	4.08E-06	1.00E-10	2.06E-07	-1.30E-11
24	clip	3.29705	0.168827	0.00044637	0.00034659	1.23E-05	7.76E-10	2.41E-07	2.19E-10
25	clip	3.12723	0.169521	0.00089475	0.0003721	8.54E-06	4.30E-10	2.29E-07	-2.17E-10
26	clip	2.70268	0.168617	0.0002344	0.00018527	6.66E-06	1.46E-10	-5.07E-09	-1.83E-10
27	clip	3.07429	0.169451	0.00077575	0.00018121	7.69E-06	2.87E-10	2.14E-07	-3.46E-13
28	clip	3.20196	0.170282	0.00114522	0.00015911	3.33E-06	7.58E-11	1.08E-07	1.15E-11
29	clip	2.86848	0.163062	5.47E-05	1.49E-05	2.31E-07	4.00E-13	-1.71E-09	-1.57E-13
30	clip	2.72911	0.164644	4.76E-05	6.57E-05	4.24E-07	-2.24E-12	-2.89E-09	-4.20E-14
31	highlighter	2.11982	0.449673	0.1716	0.00066127	0.0008159	5.99E-07	0.00033655	2.62E-08
32	highlighter	2.35589	0.46397	0.18465	0.00078874	0.00057078	3.82E-07	0.00023851	-2.82E-08
33	highlighter	2.01473	0.538053	0.248666	0.00224361	0.00250793	5.95E-06	0.00125058	-9.77E-08
34	highlighter	2.15679	0.457161	0.181407	7.00E-05	5.96E-05	3.85E-09	2.54E-05	-2.91E-11
35	highlighter	2.19393	0.494923	0.213284	0.00096022	0.00083725	7.49E-07	0.00037757	4.26E-08
36	highlighter	2.52134	0.467581	0.191037	0.00035356	0.00028701	9.14E-08	0.0001249	-8.70E-10
37	highlighter	1.87991	0.498241	0.208043	0.00452462	0.00574383	2.92E-05	0.00260904	-1.98E-06
38	highlighter	1.56473	0.644365	0.346983	0.0391961	0.0426837	0.00174541	0.0251176	4.02E-05
39	highlighter	2.17062	0.470449	0.190738	0.00044923	0.00049292	2.32E-07	0.00021491	3.32E-09
40	highlighter	1.93625	0.587896	0.30029	0.00332025	0.0032716	1.08E-05	0.00179128	4.22E-07
41	speaker	7.14848	0.18234	0.00379747	0.00034119	6.93E-05	1.05E-08	4.06E-06	-1.91E-09
42	speaker	7.81116	0.177851	0.00310268	0.00027068	3.45E-05	3.10E-09	1.75E-06	-1.22E-09
43	speaker	7.12795	0.183132	0.00366803	0.00035239	9.08E-05	1.62E-08	5.19E-06	-1.75E-09
44	speaker	6.7833	0.181716	0.00364083	0.00028041	7.34E-05	1.05E-08	4.13E-06	-7.03E-10

Figure 7: A screenshot of the CSV database file of the stored feature vectors after training the classifier

Feature vectors are collected by capturing frames from the video feed and extracting features from each frame using the extractFeatures function. The extractFeatures function returns a vector of feature vectors. Each feature vector contains features extracted from one frame.

For each scan, the system waits for the user to press the 's' key to capture frames and extract features. When the 's' key is pressed, the bounds are clipped to the previously specified region of interest and passed to the extractFeatures function. The feature vector returned by this function is written to her CSV file along with the object's labels.
After all scans are completed, the CSV file is closed, and the system returns to object detection mode.

Task 6: Classify new images
To perform classification of objects we utilize the database created. Firstly, a CSV file (DB file) containing feature vectors and labels is read. It then scales the feature vector based on the standard deviation of the feature values in the CSV file. The scaled feature vector is used to find the K nearest neighbors in the CSV file using the Euclidean or Manhattan distance, depending on the value of the D (distance metric) parameter. Finally, the most common label among the K nearest neighbors is returned as the predicted class. The classification of all 15 objects is shown in Figure 8.
The user is prompted to enter ‘m’ if Manhattan distance is to be used or ‘e’ if Euclidean distance is to be used. Also, the value of K for nearest neighbor matching is taken as a user input.

Figure 8: The classifier classifying all 15 objects that were trained

The main function is classifyFeatures, which takes a feature vector, the value of K, and the distance metric as input. The function reads the CSV file and computes the mean and standard deviation of the feature values in the CSV file. It then scales the feature vector by subtracting each object's feature vector with the corresponding feature vector of the object under the live camera and dividing the difference by the standard deviation.
The scaled feature vector is used to compute the distance between the feature vector and each row in the CSV file using either the Euclidean or Manhattan distance. For Manhattan distance we calculate the sum of absolute values and for Euclidean distance we calculate the sum squared difference between feature vectors of objects divided by standard deviation. The rows with the minimum distances are selected, and the most corresponding label in the row is returned as the predicted class and displayed on the top left corner of the bounding box, whose coordinates are taken from the bbox parameters.

Task 7: Implementing a different classifier

The basic idea behind KNN is to find the K nearest neighbors in the training set that are closest to a given image under camera and assign the majority class of these neighbors to the current image. K is a hyperparameter that can be tuned to improve classifier performance. The value of K is taken as a user input and considered for classification of images. The value of K depends on the number of times a specific object is trained.

For KNN classifier with K>1, we train the object multiple times in different orientations making it more efficient. The function findMostFrequent is used to find the most common label among a vector of strings. It uses a map to count the frequency of each string and returns the string with the highest frequency. This function is used to compute the most common label among the K nearest neighbors. The classifyFeatures() function takes a feature vector and K and D (distance metrics) values to classify objects in frames. It uses a list of training features stored in a CSV file to find the K nearest neighbors of the input feature vector and returns the most frequent object label among those neighbors as the detected class.

Task 8: Evaluate the performance of the system
The performance of the system is evaluated by testing the classification over 15 classes. 6 Trials are done for each class and the confusion matrix is obtained. The rows are the classified labels, and the columns are the true labels. Euclidean Distance Metric is used for the generation of this confusion matrix, by detecting the class of the object, one object at a time.
The same test was performed for KNN (K-Nearest Neighbor) where $K = 10$. The same Euclidean Distance Metric was used. A much better accuracy was observed.

The confusion matrices for Nearest Neighbor and K – Nearest Neighbor (K = 10) can be seen in Figure 9.

Task 9: Capturing a demo of the system working
The video of the demo has been uploaded to YouTube, and can be found here:
https://www.youtube.com/watch?v=9gmSl2JLopE

True Labels / Classified labels	tumbler	wallet	clip	highlighter	speaker	straw	headphone	trimmer	keychain	spoon	harddisk	sunglasses	cardholder	case	mug
tumbler	6						2								
wallet		6													
clip			6												
highlighter				6											
speaker					6										
straw						6			1						
headphones							4						2		
trimmer								4		1					
keychain								2	5						
spoon										5	1				
harddisk											5				
sunglasses												6			
cardholder													4		
case														6	1
mug															5

True Labels / Classified labels	tumbler	wallet	clip	highlighter	speaker	straw	headphone	trimmer	keychain	spoon	harddisk	sunglasses	cardholder	case	mug
tumbler	6														
wallet		2													
clip			6												
highlighter				6											
speaker		2			6										
straw						6									
headphones							6								
trimmer								6							
keychain									6						
spoon										6					
harddisk											6				
sunglasses												6			
cardholder													6		
case														6	
mug		2													6

Figure 9: Confusion Matrices for (a) Nearest Neighbor and (b) K-Nearest Neighbor Classifier (K = 10) with Euclidean Distance Metric

Extensions
Three extensions have been added to this project. More than 10 items have been added to the database, the classifier is able to classify 15 different objects. The classifier can be run by computing either of the two distance metric implementations, Manhattan distance metric, or Euclidean distance metric. The classifier can also classify multiple images at a time in a single frame.

Extension 1
The feature vectors of more than the required 10 classes have been saved in the database file. The classifier can classify objects across 15 classes, namely tumbler, wallet, clip, highlighter, speaker, straw, headphones, trimmer, keychain, spoon, harddisk, sunglasses, cardholder, case, and mug.

Extension 2
There are two distance metric implementations, Manhattan distance metric and Euclidean distance metric. When the program is first run, it requests the user to enter the desired distance metric that must be used for classification. The user can enter ‘e’ for Euclidean distance metric to be used or ‘m’ for Manhattan distance metric.

Extension 3
The classifier can classify multiple objects accurately all at once. The bounding box can be drawn for multiple objects in the scene and the classified label is printed on the respective bounding box. An example of this can be seen in Figure 10, where the classifier is able to classify 6 objects at the same time.

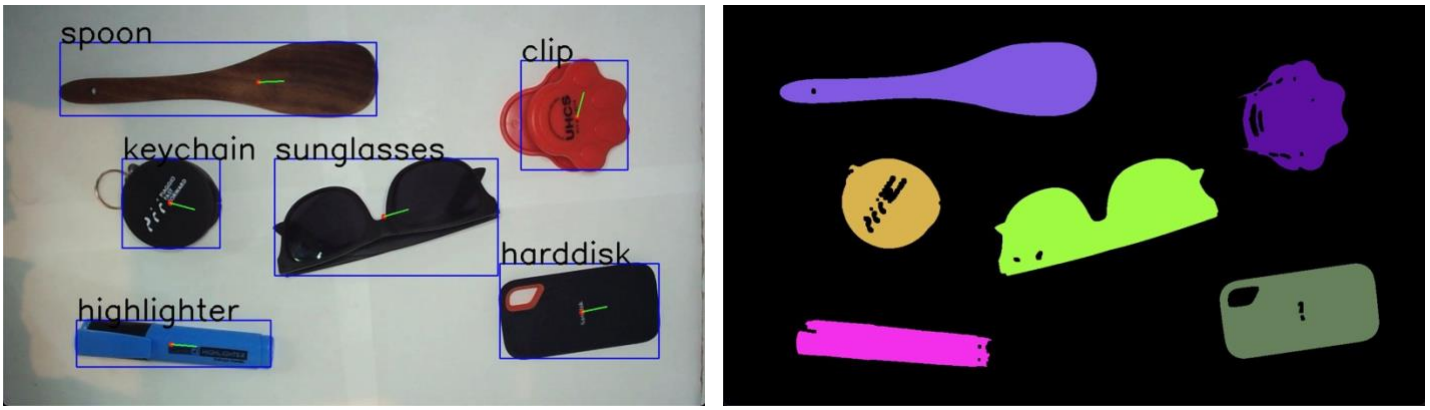

Figure 10: (a) The classifier classifying 6 objects simultaneously (b) The segmented image separating the foreground from the background

Project Learnings and Insights
Object recognition is a popular area of computer vision where the goal is to identify objects in an image or video. There are various techniques for object recognition, including methods based on deep learning and traditional methods based on image processing. In this project, the goal was to recognize 2D objects invariably in displacement, scaling, and rotation from the camera looking down. The method includes thresholding of the input video, cleaning the binary image, segmenting the image into regions, computing features for each key region, collecting training data, and classifying new images.

Additionally, we gained hands-on experience in collecting and labeling data for use in training and classification models. Overall, this project provided a valuable learning experience in key highlights of computer vision.

Acknowledgements and Resources
We would like to acknowledge Professor Bruce Maxwell and all the Teaching Assistants for their valuable insights and support throughout the project.

- OpenCV Tutorials: https://docs.opencv.org/4.5.1/index.html
- Morphological Operations: https://medium.com/@rajilini/morphological-operations-in-image-processing-using-opencv-and-c-8580de272606
- Segmentation Algorithm: https://towardsdatascience.com/implementing-a-connected-component-labeling-algorithm-from-scratch-94e1636554f
- Shape Matching using Hu moments: https://learnopencv.com/shape-matching-using-hu-moments-c-python/
- Li, Tt., Jiang, B., Tu, Zz., Luo, B., Tang, J. (2015). Image Matching Using Mutual k-Nearest Neighbor Graph. In:, et al. Intelligent Computation in Big Data Era. ICYCSEE 2015. Communications in Computer and Information Science, vol 503. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-46248-5_34