REPORT
PROJECT 03

(Real-time 2-D Object Recognition)

By – Veditha Gudapati
&
Tejasri Kasturi

Introduction:

The objective of this project is to create a real-time 2D object recognition system that can recognize objects positioned on a white surface when viewed from above. The goal of this research is to create an object classification system by combining cutting-edge computer vision algorithms and methodologies. Firstly, the system converts color images into binary representations by thresholding, which allows foreground objects to be separated from the background. Next, the segmentation is further refined using morphological processes, which improve the accuracy and standard of object identification. The process of segmenting the binary image into sections enables the detection and labeling of connected components, which prepares the image for feature extraction. Since feature extraction requires calculating relevant characteristics for each detected region, it is essential for classification process. These characteristics function as elements that help separate and classify items. In addition, the incorporation of a training mode facilitates real-time object categorization by allowing the system to learn and categorize things according to labels given by the user.
Parameters like accuracy, precision, and recall are used to assess the classification system's performance; the evaluation process provides information that helps algorithm modifications and enhancements. In addition, the project uses a secondary classification method based on deep neural networks (DNNs), using their ability to autonomously build hierarchical representations of features from raw data.

TASK – 1: Threshold the Input Video

In this task the "**thresholdImage"** function plays an important role in image processing by converting color images into binary representations. First, it uses OpenCV's "**cvtColor"** function to transform the input color image to grayscale so that it may concentrate only on intensity values. Next, a blank canvas is created by initializing the output binary picture to match the size of the input by setting all pixel values to zero. The thresholding procedure, which compares each pixel's intensity value to a predetermined threshold, is the core of the function. A pixel in the binary picture output turns black if the intensity is higher than the threshold and white otherwise.

During the thresholding process, the function iterates through every pixel in the grayscale image. It compares the intensity value of each pixel to a preset threshold. The corresponding pixel in the binary picture output is set to 0 (black), indicating background, if the intensity is greater than the threshold. On the other hand, the output pixel is set to 255 (white), denoting the foreground or object of interest, if the intensity is less than or equal to the threshold.

Once all the pixels are processed, the function produces a binary image that successfully separates the objects of interest from the background.

The **"dynamicThresholding"** function dynamically adjusts the threshold value based on clustering and saturation values.

The following is a thorough description of the flow of the function:

1. **Blur Operation:** The function first blurs the input image / video using a Gaussian blurring technique. This stage is crucial for reducing noise and generating a smoother image, both of which can raise the precision in the next stages.
2. **Color Space Conversion:** The blurred image is then transformed from the BGR (Blue-Green-Red) color space to the LAB (Lightness-A/B) color space using the color space conversion technique. Because of its ability to separate color and brightness information more effectively due to its visual consistency, the LAB color space is used.
3. **Flattening and Conversion**: To prepare the LAB picture for clustering analysis, it is reshaped into a single channel. The image data type is also changed to make further operations easier.

4. **K-means Clustering:** The reshaped LAB image's distinct color clusters are found using K-means clustering. Through the specification of the number of clusters (two in this case), the algorithm divides the image into areas that share comparable color properties.
5. **Threshold Calculation**: Each cluster's average color values are represented by the centers of the resulting clusters, which are computed. The threshold value that is obtained by combining and weighing these centers is essential for the segmentation process that will be implemented in the following steps.
6. **Blur and Color Space Conversion (HSV):** To improve segmentation quality, the input image undergoes one more round of Gaussian blurring. After that, the image is blurred and transformed to the HSV color space, where hue, saturation, and value components are used to represent each pixel.
7. **Saturation-based Thresholding:** The HSV image's saturation component is used for thresholding. Pixels with high saturation values are attenuated, reducing their intensity.
8. **Reconversion to BGR**: The processed image is transformed back to the BGR color space, retaining the original color information while incorporating thresholded adjustments.
9. **Final Thresholding**: To create the final segmented image, the resulting image is further thresholded using the calculated threshold value that was obtained from the clustering stage**.**

Fig 1: Original Image (dog)

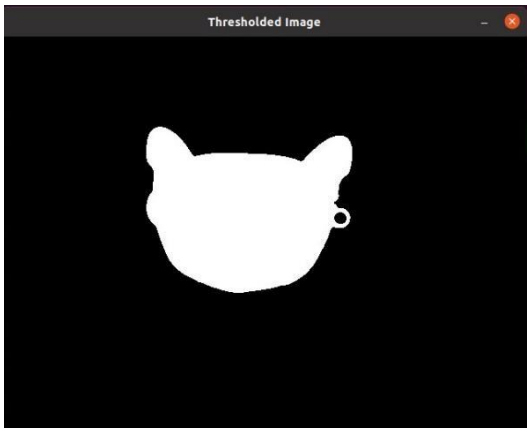


Fig 2: Threshold Output of Fig 1

Fig 3: Original Image (mouse)

Fig4: Threshold Output of Fig 3

TASK – 2: Clean Up the Binary Image

In this task we implemented "**applyMorphologicalOperations"** function to enhance the precision and quality of binary picture segmentation. The function accepts three parameters: the source binary image (src), the destination image (dst) to store the result of morphological operations, and the structuring element (kernel) used for defining the neighborhood around each pixel.

We then implemented the morphological (erosion and dilation) operations:

1. Erosion involves deleting pixels from the edges of the objects in the binary image, hence shrinking the foreground objects. Erosion calculates the lowest pixel value inside the kernel-defined neighborhood for each pixel in the image. This process works well for eliminating thin features or small noisy areas from the foreground items.
2. Dilation is the inverse of erosion and aims to expand the foreground objects by adding pixels to the object boundaries. Dilation calculates the maximum pixel value inside the kernel neighborhood for each pixel in the image. It is useful for joining disparate parts and filling in spaces or gaps inside objects.

Applying the required morphological procedures with the given kernel, the function iterates over each pixel in the input binary picture.
Depending on the intended cleaning effect, conditional statements are utilized to decide whether erosion or dilation should be used.

Fig 5: Morphological Output of Fig 1

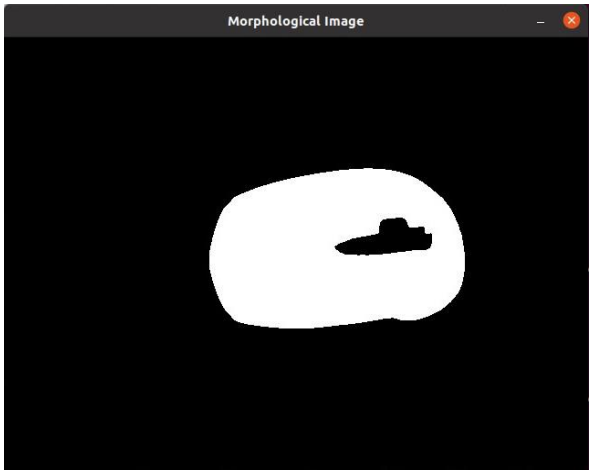


Fig 6: Morphological Output of Fig 2

TASK – 3: Segment the Image into Regions

In this task the **"segmentRegions"** function is implemented to identify and label connected regions within the binary image. These regions correspond to distinct objects or components present in the input image/video. The implementation of the function is as given below:

1. Connected Component Analysis (CCA): First, the program uses OpenCV's **"connectedComponentsWithStats"** function to perform connected component analysis (CCA) on the input binary picture. This procedure creates a labeled image, together with statistics and centroids for each labeled region, by labeling connected regions in the binary image.
2. Region Visualization: The function initializes a region map, which is an output image used to visualize the segmented regions. To differentiate each labeled region from other regions in the binary image, each one is given a distinct color.
3. Bounding Box Drawing: The function uses the **"findContours"** function to determine the contours of each designated region. The extent of each region is then visualized by drawing a bounding box around each contour using the rectangle function. The boundaries and geographical distribution of segmented objects can be seen with the help of this step.
4. Region Filtering: Optionally, the function may filter out regions based on certain criteria, such as size or shape. Large regions might be given priority for further analysis, while small regions might be disregarded to reduce noise. Customizing the filtering criteria is possible in accordance with the application's particular needs.

The main output of the "**segmentRegions"** function is the region map, which visualizes the segmented regions in the binary image.

Fig 7: Region Map of Fig 1

Fig 8: Region Map of Fig 3

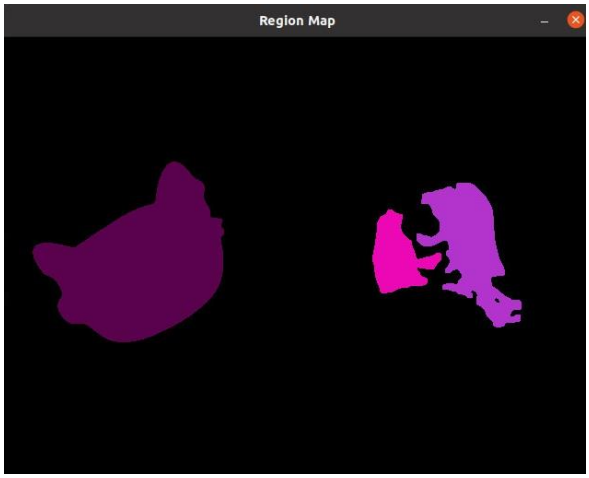


Fig 9: Region Map with each region shown in a different color.

TASK – 4: Compute Features for Each Major Region

In this task **"extractFeatures"** is implemented to analyze each segmented region within the input image, extract relevant features from these regions, and visualize the results on the output image. The **"extractFeatures"** function starts by making a clone of the input image to keep it intact during processing. This cloned image ensures that any changes made do not have an impact on the original image/video. The program then reduces the cloned image to a single intensity channel by converting it to grayscale. Converting to grayscale makes further processing stages easier, making feature extraction and contour detection possible. The **"cvtColor"** function in OpenCV effectively handles this conversion. The analysis that is then performed in the image/video is as follows:

1. Contour Detection :

   The contours, or outlines, of segmented regions within the input image are found and extracted by the "**findContours"** function . The borders dividing areas of varying intensity are represented by these contours. "**FindContours"** finds continuous lines enclosing distinct areas by examining pixel connectivity and intensity gradients. For processes like object detection, shape analysis, and feature extraction to be implemented in the next steps, this data is essential. The function returns a list of contours, each represented as a series of points, which can be further analyzed or visualized to understand the spatial distribution and shape characteristics of segmented regions within the image.

2. Calculating Moments:

   The **"moments"** function calculates spatial moments of a contour, providing statistical information about its shape and distribution. These moments include the contour's centroid, area, and orientation, all of which are critical for describing objects inside an image during feature extraction.

3. Calculating Hu Moments:

   The **"HuMoments"** function computes Hu Moments, which are a set of seven rotation, scale, and translation-invariant moments derived from image moments. These moments capture essential shape characteristics of the contour, providing an efficient representation that is invariant to changes in rotation, scale, and position. These features play a crucial role in distinguishing and characterizing objects within an image, facilitating accurate and reliable analysis in computer vision applications.

4. Computing Bounding Rectangle:

   The bounding rectangle, calculated using the **"minAreaRect"** function, is the smallest rectangle that can contain the dimensions of the contour. This rectangle is helpful for shape analysis and object characterization since it gives important details about the contour's size, aspect ratio, and orientation. The bounding rectangle simplifies the contour's geometry.

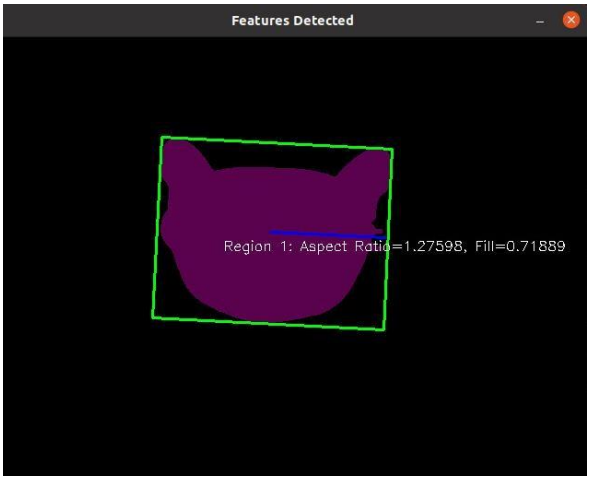


Fig 10: Computed Features of Fig 1

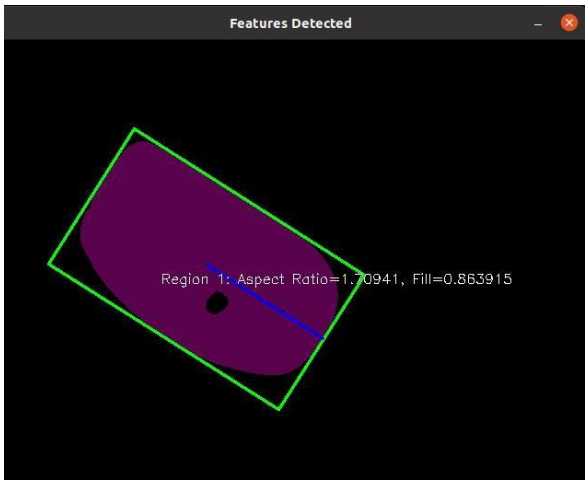


Fig 11: Computed Features of Fig2

TASK – 5: Collect Training Data

In this task the training mode is introduced. The "**trainingMode"** flag indicates that the system enters training mode when "N" key is pressed. The training mode is activated by the **"handleKeyPress"** function. When the user selects this mode, the system asks them to give a label for the detected object , which is then saved in the **"currentLabel"** variable. The feature vectors and labels that go with them are stored in a file stream (featureVectorFile) using the **"storeFeatureVector"** function. Every feature vector is written to the file as a std::vector, separated by commas, and is accompanied by its corresponding label. During training mode, integration into the main loop guarantees that feature vectors are recorded and stored. This makes it easier to collect and access the feature information in an organized manner.

The functionality of the task is as follows:

1. Training Mode Activation:

   The program allows the user to activate the training mode by pressing the 'N' key during runtime. Upon activation, the system prompts the user to enter the label for the current object being observed. Once the label is provided, the system enters the training mode, ready to capture feature vectors for the labeled object.

2. Feature Vector Extraction and Storage:

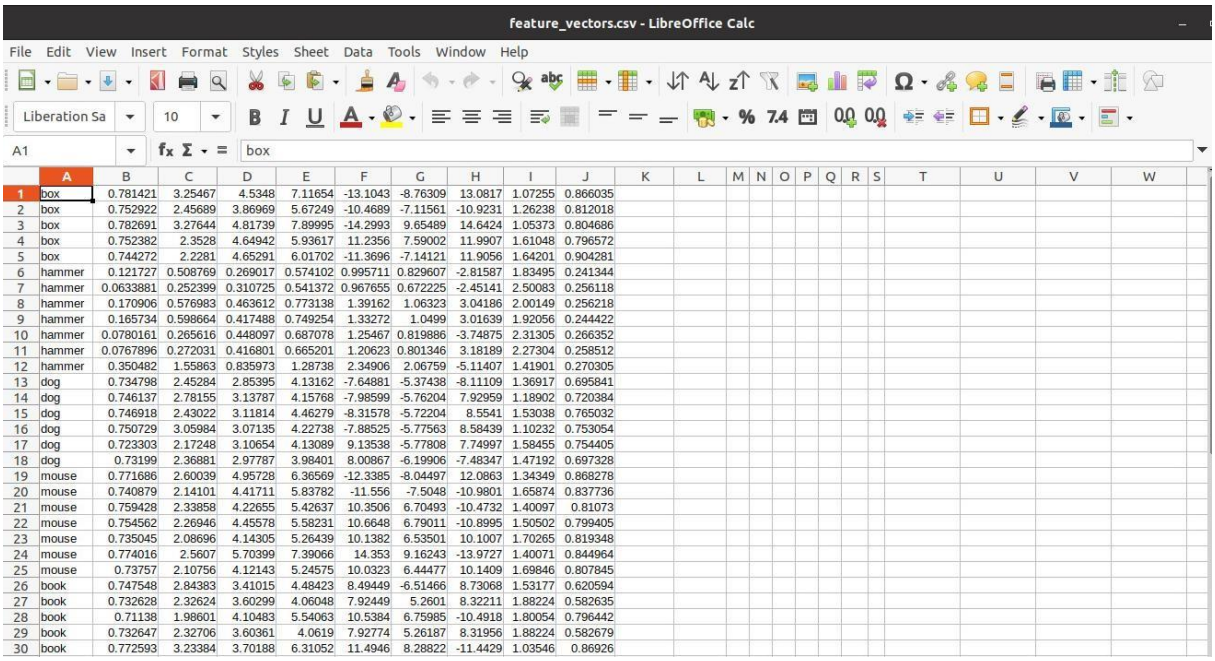
   As the system captures video frames, it applies thresholding and morphological operations to preprocess the images. Regions of interest are segmented from the processed images, and features are extracted from these regions using various techniques such as Hu moments, aspect ratio calculation, and fill percentage computation. During training mode, the extracted feature vectors are stored along with their corresponding labels in a CSV file named "feature_vectors.csv". The feature vectors are stored in a comma-separated format, with each row representing a labeled object's features.

3. Classification of Objects:

   To classify objects in real-time, the system utilizes the k-Nearest Neighbors (k-NN) algorithm. During classification, the program reads the existing feature vectors and labels from the CSV file. It calculates the Euclidean distance between the features of the input object and those in the training dataset. The system then assigns the label of the nearest neighbor to the input object, effectively classifying it.

TASK – 6: Classify New Images:

To classify a picture, the extracted characteristics are compared to those in the existing dataset of feature vectors, stored in a CSV file (feature_vector.csv). The application initiates the classification process when the 'r' key is pressed. The scaled Euclidean distance between each feature vector in the dataset and the features of the captured image is computed. Next, the classification algorithm finds the feature that most closely matches the features in the dataset and the features that were collected in the image. It precisely matches the object's expected label in the picture. Lastly, using the closest match discovered in the dataset, the algorithm outputs the expected label for the object in the picture.



	A	B	C	D	E	F	G	H	I	J
1	box	0.781421	3.25467	4.5348	7.11654	-13.1043	-8.76309	13.0817	1.07255	0.866035
2	box	0.752922	2.45689	3.86969	5.67249	-10.4689	-7.11561	-10.9231	1.26238	0.812018
3	box	0.782691	3.27644	4.81739	7.89995	-14.2993	9.65489	14.6424	1.05373	0.804686
4	box	0.752382	2.3528	4.64942	5.93617	11.2356	7.59002	11.9907	1.61048	0.796572
5	box	0.744272	2.2281	4.65291	6.01702	-11.3696	-7.14121	11.9056	1.64201	0.904281
6	hammer	0.121727	0.508769	0.269017	0.574102	0.995711	0.829607	-2.81587	1.83495	0.241344
7	hammer	0.0633881	0.252399	0.310725	0.541372	0.967655	0.672225	-2.45141	2.50083	0.256118
8	hammer	0.170906	0.576983	0.463612	0.773138	1.39162	1.06323	3.04186	2.00149	0.256218
9	hammer	0.165734	0.598664	0.417488	0.749254	1.33272	1.0499	3.01639	1.92056	0.244422
10	hammer	0.0780161	0.265616	0.448097	0.687078	1.25467	0.819886	-3.74875	2.31305	0.266352
11	hammer	0.0767896	0.272031	0.416801	0.665201	1.20623	0.801346	3.18189	2.27304	0.258512
12	hammer	0.350482	1.55863	0.835973	1.28738	2.34906	2.06759	-5.11407	1.41901	0.270305
13	dog	0.734798	2.45284	2.85395	4.13162	-7.64881	-5.37438	-8.11109	1.36917	0.695841
14	dog	0.746137	2.78155	3.13787	4.15768	-7.98599	-5.76204	7.92959	1.18902	0.720384
15	dog	0.746918	2.43022	3.11814	4.46279	-8.31578	-5.72204	8.5541	1.53038	0.765032
16	dog	0.750729	3.05984	3.07135	4.22738	-7.88525	-5.77563	8.58439	1.10232	0.753054
17	dog	0.723303	2.17248	3.10654	4.13089	9.13538	-5.77808	7.74997	1.58455	0.754405
18	dog	0.73199	2.36881	2.97787	3.98401	8.00867	-6.19906	-7.48347	1.47192	0.697328
19	mouse	0.771686	2.60039	4.95728	6.36569	-12.3385	-8.04497	12.0863	1.34349	0.868278
20	mouse	0.740879	2.14101	4.41711	5.83782	-11.556	-7.5048	-10.9801	1.65874	0.837736
21	mouse	0.759428	2.33858	4.22655	5.42637	10.3506	6.70493	-10.4732	1.40097	0.81073
22	mouse	0.754562	2.26946	4.45578	5.58231	10.6648	6.79011	-10.8995	1.50502	0.799405
23	mouse	0.735045	2.08696	4.14305	5.26439	10.1382	6.53501	10.1007	1.70265	0.819348
24	mouse	0.774016	2.5607	5.70399	7.39066	14.353	9.16243	-13.9727	1.40071	0.844964
25	mouse	0.73757	2.10756	4.12143	5.24575	10.0323	6.44477	10.1409	1.69846	0.807845
26	book	0.747548	2.84383	3.41015	4.48423	8.49449	-6.51466	8.73068	1.53177	0.620594
27	book	0.732628	2.32624	3.60299	4.06048	7.92449	5.2601	8.32211	1.88224	0.582635
28	book	0.71138	1.98601	4.10483	5.54063	10.5384	6.75985	-10.4918	1.80054	0.796442
29	book	0.732647	2.32706	3.60361	4.0619	7.92774	5.26187	8.31956	1.88224	0.582679
30	book	0.772593	3.23384	3.70188	6.31052	11.4946	8.28822	-11.4429	1.03546	0.86926

Fig 12 : Classified Images

TASK – 7: Evaluate the Performance of Your System

An essential part of the classification process is the confusion matrix. By comparing the system's predictions to the actual actual labels, it evaluates the system's performance. It enables the identification of true positives, true negatives, false positives, and false negatives, which help us identify the strengths and weaknesses of the classification algorithm.

1. Evaluation of Classification Performance: The confusion matrix allows us to evaluate how well the classification algorithm performs in terms of accuracy, precision, recall, and other performance metrics. By comparing the predicted labels with the actual labels, we can identify areas where the algorithm may need to be improved.
2. Finding Misclassifications: When the classification system predicts something incorrectly, it can be found using the confusion matrix. We can determine which classes are frequently confused with one another and make the required changes for the improvement of the system’s performance.
3. Classification Algorithm Adjustment: The classification algorithm's parameters can be adjusted with the use of the confusion matrix's results.

```
Confusion matrix mode enabled.
Enter the Object Label:
dog
Reading the CSV file
Reading /home/kasturi/PRCV/Project_3/feature_vectors.csv
Finished reading CSV file
dog
Reading /home/kasturi/PRCV/Project_3/feature_vectors.csv
Finished reading CSV file
2
2
Matrix elements:
0 0 0 0 0
0 0 0 0 0
0 0 1 0 0
0 0 0 0 0
0 0 0 0 0
```

Fig 13 : Confusion matrix for mouse

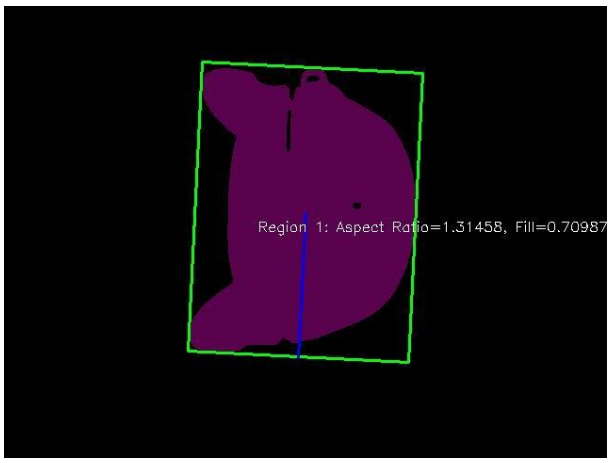


Fig 14: Detected Region of Object

```
Confusion matrix mode enabled.
Enter the Object Label:
dog
Reading the CSV file
Reading /home/kasturi/PRCV/Project_3/feature_vectors.csv
Finished reading CSV file
dog
Reading /home/kasturi/PRCV/Project_3/feature_vectors.csv
Finished reading CSV file
2
2
Matrix elements:
0 0 0 0 0
0 0 0 0 0
0 0 2 0 0
0 0 0 0 0
0 0 0 0 0
```

Fig 15 : Confusion matrix for mouse

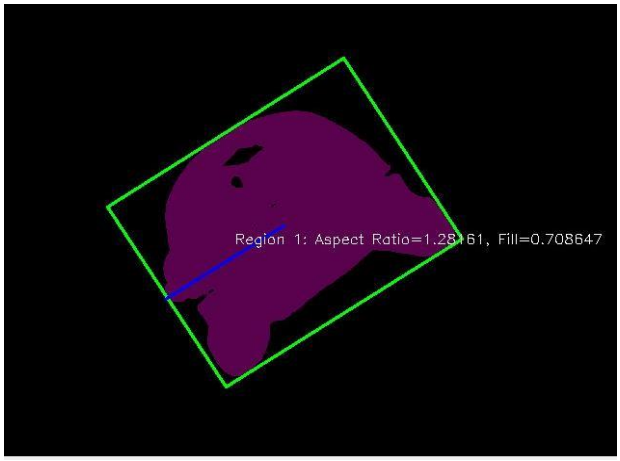


Fig 16: Detected Region of Object

```
Confusion matrix mode enabled.
Enter the Object Label:
dog
Reading the CSV file
Reading /home/kasturi/PRCV/Project_3/feature_vectors.csv
Finished reading CSV file
dog
Reading /home/kasturi/PRCV/Project_3/feature_vectors.csv
Finished reading CSV file
2
2
Matrix elements:
0 0 0 0 0
0 0 0 0 0
0 0 3 0 0
0 0 0 0 0
0 0 0 0 0
```

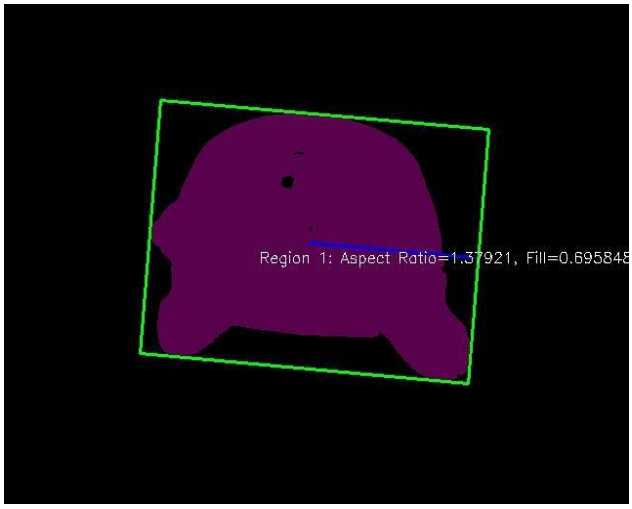


Fig 17 : Confusion matrix for mouse

Fig 18: Detected Region of Object

TASK – 8: Capture A Demo of Your System Working:

The video of the demo of the system working :
https://drive.google.com/file/d/1xEVFqubC7ECCdt3tsk6ZaXfRTDjoJR0c/view?usp=sharing

TASK – 9: Implement A Second Classification Method:

For this object classification project, two machine learning algorithms were suggested - deep neural networks (DNNs) and k-nearest neighbors (kNN). We used the DNN algorithm in this project because:

1. Firstly, DNNs perform well in automatically learning hierarchical representations of features from raw data, eliminating the need for manual feature extraction. Which is helpful in tasks involving images, where extracting meaningful features can be difficult. In contrast, kNN relies on predefined feature vectors and closeness in feature space for classification, which limits its flexibility.
2. Additionally, DNNs can effectively model high-dimensional, non-linear interactions in visual data through multiple trainable layers and activation functions. This makes them adept at handling variances and noise in real-world images. kNN on the other hand operates under a linear assumption, making it less reliable. The greater modeling capacity of deep nets helps them better generalize to unfamiliar data.

Despite training the data multiple times, we were not able to get the desired results. While the program detects the object correctly most times, it fails to give the correct identification in some cases. Additionally , the desired result is obtained when the object is subjected to multiple attempts for detection.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	toy	-2.2681	-1.122	-0.7372	0.6331	2.7661	0.4357	0.1233	0.9438	1.1026	-0.0064	0.1472	1.0514	0.9795	0.3666	0.8867	-0.7812	1.6325	-0.0543	2.9027
2	toy	-2.3288	-1.5947	-0.3694	-0.1345	2.7636	0.4459	-0.0785	-0.5343	0.6582	-1.0411	0.0073	0.9825	0.3052	1.2048	0.023	-1.5922	0.5413	-1.5733	1.6758
3	toy	-0.9041	-1.7459	-1.1793	2.1209	1.884	-0.4376	-0.0534	-0.712	1.2198	-0.9964	0.2428	0.4782	1.0321	0.5372	1.3536	-1.5026	2.505	-1.411	1.4465
4	toy	-0.5606	-4.069	-1.19	2.982	-2.356	-2.1683	2.7732	-4.6884	0.172	-2.2076	1.5776	-3.0032	-1.0113	0.4959	4.6535	-1.637	2.8736	-4.4433	-3.7403
5	toy	-1.8463	-3.1681	-1.2411	3.2919	-1.1189	-3.1064	3.1691	-3.6077	0.5987	-1.7655	1.1905	-1.3487	-0.7344	0.5008	3.5994	-1.7792	2.0381	-4.1442	-2.3315
6	toy	-1.6904	-2.7394	-2.3386	1.2097	2.2297	-2.3349	1.0492	-1.5709	0.8569	-0.977	1.3175	1.2542	0.0313	0.7546	1.3236	-1.3303	0.7102	-2.8609	1.4064
7	glove	1.5005	-1.2861	1.4409	11.0422	-17.469	0.8222	8.5133	-4.6523	2.0223	0.7549	-4.7116	-23.052	-7.9443	1.1908	6.2115	4.0622	7.9467	-6.6052	-12.719
8	glove	3.7472	-0.0838	-2.0689	8.8311	-3.7364	-3.0931	3.7767	-2.7493	1.1591	3.2202	0.049	-6.4704	-2.3857	-0.8436	3.8505	-1.4355	8.5911	-3.0782	-4.0095
9	glove	3.4087	1.7777	0.5573	0.7144	-1.6328	-0.1818	-1.364	0.5631	-1.5913	3.2778	-2.4818	-2.9671	-2.3752	-0.3763	2.6244	1.7843	0.7932	0.2109	-2.3269
10	glove	13.5652	2.9925	-2.1733	1.7964	-9.6221	-7.3554	-0.8935	-6.1045	-2.0373	10.9786	-1.9558	-12.555	-6.7081	-4.7147	4.1403	0.9277	-0.7952	-4.8754	-8.4925
11	glove	8.3161	1.647	-4.5612	13.3059	-9.3917	-5.7972	6.0299	-5.5652	1.9229	7.503	2.5994	-12.684	-2.0884	-6.4338	7.4611	-2.7382	9.5493	-3.8908	-8.4565
12	glove	8.2881	1.6509	-4.5173	13.3495	-9.4606	-5.7875	6.114	-5.5655	1.892	7.4885	2.5824	-12.72	-2.0836	-6.4692	7.4193	-2.7001	9.5319	-3.8506	-8.5315
13	glove	21.4491	-1.6187	0.6347	8.1712	-16.076	-7.7722	-2.5434	-6.8234	-2.1367	9.8333	-11.17	-22.738	-13.485	-5.0339	16.0628	0.0478	1.0246	-9.5873	-15.849
14	hammer	9.895	13.7188	4.1602	8.0003	-12.673	2.5	6.3327	2.701	-0.3843	7.4186	-0.9395	-18.86	-9.9853	-5.9449	4.9579	3.6019	10.6634	0.6934	-10.23
15	hammer	8.594	13.3544	3.798	5.3128	-12.475	1.9893	5.7527	1.8556	-2.2101	9.1881	0.8768	-15.79	-8.8188	-7.0027	4.9298	4.7759	10.6084	1.7561	-8.7628
16	hammer	7.6812	7.6941	1.7488	5.1574	-7.0608	-0.7591	1.045	-0.1794	-0.8169	6.8502	-0.6721	-7.7421	-2.9441	-5.1596	4.7346	2.2174	6.75	1.7128	-5.464
17	hammer	4.6311	10.8078	1.9501	4.4769	-4.0748	2.623	0.6096	4.8621	1.9999	4.9605	-1.232	-5.0053	-0.468	-4.432	1.4375	3.1127	7.809	5.1634	-1.4056
18	hammer	5.242	11.8594	4.8284	5.807	-11.123	4.1017	5.7833	3.8733	-0.9838	6.9449	-1.1393	-14.54	-5.6329	-5.4329	2.6233	5.2587	8.4239	3.3315	-7.3506
19	mouse	8.5152	1.9528	-5.2331	7.0665	-2.6111	-6.1813	2.5322	-3.1263	0.4739	5.5089	3.1229	-4.0583	0.0048	-4.8389	5.7449	-3.5723	8.298	-2.8853	-4.6601
20	mouse	1.5063	-2.1197	-4.0468	0.9568	2.1922	-4.5209	2.7427	-2.243	-0.5661	2.1999	3.6946	2.2621	0.9222	-0.9506	2.4182	-2.8317	2.2493	-2.3464	-0.5723
21	mouse	0.7659	-3.2449	-4.4383	0.0405	1.808	-4.8357	1.5354	-3.9506	-0.5567	1.7946	3.9039	4.2314	1.6693	-1.072	1.9621	-3.097	0.7861	-2.5955	-0.7131
22	mouse	5.2413	-2.212	-6.0786	5.5505	0.3576	-6.6718	3.6884	-4.6079	-0.887	2.7357	4.0675	-3.1024	-1.3221	-1.3972	5.812	-4.3155	4.5421	-5.1817	-4.7821
23	mouse	-1.1481	-1.3177	0.1428	0.874	0.515	-1.8649	1.8273	-2.5339	0.371	0.3655	3.4986	1.2553	1.2314	-0.3876	-1.6395	-1.0842	0.1517	-0.6049	0.3754
24	wallet	12.8101	2.8347	-3.4885	18.4184	-14.253	-9.9556	9.5905	-9.735	1.2757	11.9283	2.6701	-15.653	-5.3737	-8.2915	7.1929	-2.3078	16.023	-6.5226	-12.981
25	wallet	3.3204	3.0012	-0.6164	8.7326	-5.8412	-2.57	6.7132	-3.3496	1.7411	4.5236	0.595	-6.7077	-2.5057	-2.8934	1.1616	-0.6101	7.3041	-0.3184	-5.7852
26	wallet	0.2656	-0.275	-1.8716	2.6655	-1.5217	-2.6057	3.4137	-1.6369	0.7674	2.5856	2.5262	-1.3529	0.1808	-1.1379	0.0152	-1.2865	2.4877	-0.6878	-1.4781
27	book	6.2414	6.7196	-0.5668	6.0621	-19.19	-1.2721	12.9419	-5.7097	-5.9167	7.7537	4.6635	-30.626	-21.676	0.1444	6.4908	-1.5718	9.059	-7.7603	-19.942
28	book	8.3071	3.8859	-3.4754	11.1109	-12.91	-6.2422	9.9606	-7.1306	-0.7421	9.2502	3.3451	-20.268	-12.339	-4.1466	7.4192	-2.0035	9.943	-8.0498	-13.647

Fig 19: Dataset for DNN

```
Reading /home/kasturi/PRCV/Project_3/feature_vectors_dnn.csv
Finished reading CSV file
31
Label: toy
Dist: 30.2749
Label: toy
Dist: 29.1591
Label: toy
Dist: 27.8616
Label: toy
Dist: 27.5501
toy
```

Fig 20: Output for DNN

EXTENSIONS:

1. Add More Than the Required Five Objects to The DB So Your System Can Recognize More Objects

For the second extensions, we have added more than five objects to the database for the system to recognize. The objects that we have added are :

- Toy
- Glove
- Hammer
- Book
- Mouse
- Wallet

Through extensive training, we were able to get the system to identify additional objects. The program attempts to detect the objects correctly most times, but in some cases detects the objects incorrectly. In such cases upon multiple attempts for detection, the desired results are received The database for all six objects is in the form of a CSV (feature_vectors_dnn.csv).

REFLECTION ON WHAT WE LEARNT:

1. Gained proficiency in implementing various image processing techniques, such as thresholding algorithms morphological operations (erosion & dilation).
2. Importance of Feature Extraction: Feature extraction plays a crucial role in machine learning-based classification tasks. We learned how to extract meaningful features from images, such as Hu moments, bounding rectangles, and aspect ratios, which are important for accurately classifying objects.
3. Explored the nearest-neighbor classification, including the selection of appropriate distance metrics (Euclidean distance) and the optimization of classification performance through parameter tuning.
4. Gained hands-on experience in implementing machine learning algorithms, specifically the k-Nearest Neighbors (k-NN) algorithm for classification. Additionally, we learned about the potential of deep neural networks (DNNs) for image classification tasks and their advantages over traditional algorithms like k-NN.
5. We learned about the importance of evaluating the performance of classification algorithms using metrics such as accuracy, precision, recall, and the confusion matrix. Furthermore, we gained insights into optimizing algorithms by adjusting parameters and improving feature selection to enhance performance.

ACKNOWLEDGEMENTS:

I am grateful to the OpenCV documentation, which has been a vital resource during this project. Understanding and putting different computer vision algorithms and techniques into practice has been madepossible by the comprehensive explanations given in the OpenCV documentation.

I would also like to thank my peers for their help and feedback during this project. Throughout this process, exchanging knowledge, working together to troubleshoot challenges, and sharing ideas have all been extremely beneficia