Real-Time 2D Object Recognition

Joseph Nelson Farrell & Harshil Bhojwani

February 21, 2024

Project Description

In this project we were tasked with image processing with goal of performing real-time object recognition. There were many steps in the process starting with thresholding, where the frame is converted to a binary image where all the pixels that are part of the foreground remain white and all the background pixels are black. We then cleaned the image using erosion and dilation. Once we had a cleaned image we identified the different regions. We then filtered the regions to identify the object of interest. Next we computed the moments and from the moments we computed the oriented bounding box. This was done with the goal of creating features that were translation, scale, and rotation invariant. Once we had the features we created a database so that we could classify objects in real time.

Source Images

These are the source images of the objects that will be used as we walk through the steps of the project.

Figure 1: Chess Piece

Figure 2: Thule Key

Figure 3: Eraser

Thresholding

The first step was thresholding. To accomplish we used all home brew functions, and followed these steps:

1. Apply Gaussian Blur
2. Apply Grey Scale
3. Apply KMeans(k = 2) (Provided by Professor Maxwell)
4. Compute threshold value.
5. Create binary image

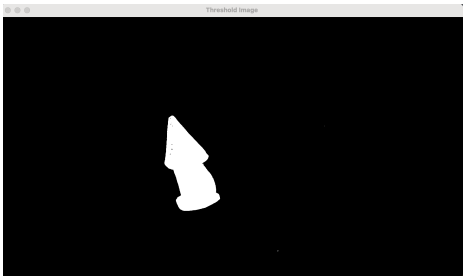

Figure 4: Threshold Chess Piece

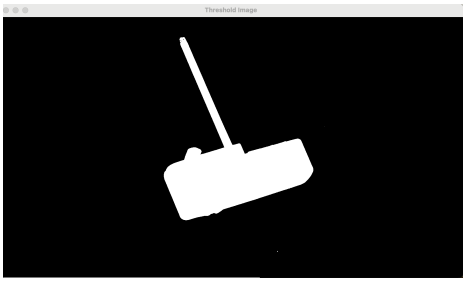

Figure 5: Threshold Thule Key

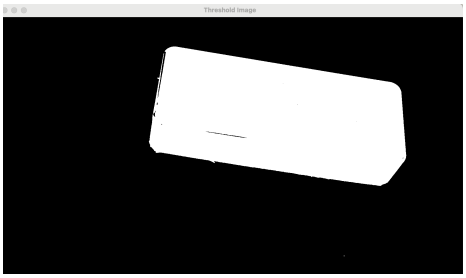

Figure 6: Threshold Eraser

Morphological Filtering

Next we cleaned the binary image by performing morphological filtering. The following procedures were carried out using home brew functions.

1. 3 4-connected dilations
2. 2 8-connected erosions

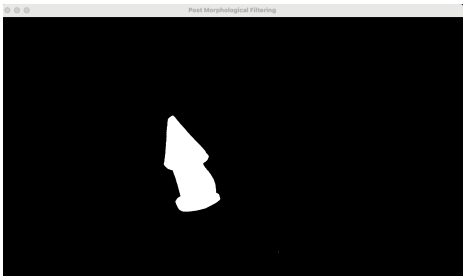

Figure 7: Post Morph Chess Piece

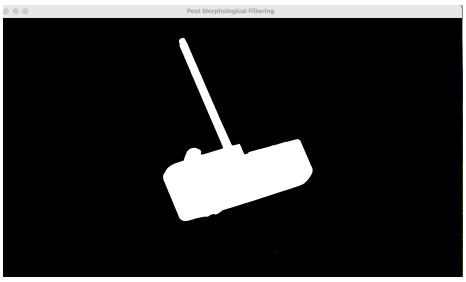

Figure 8: Post Morph Thule Key

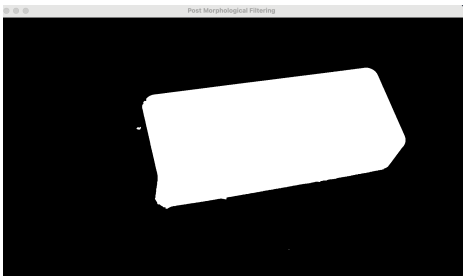

Figure 9: Post Morph Eraser

Segmentation/Regions

Next we segmented the image into regions using **cv::connectedComponentsWithStats** and applied the following filters:

1. Region Area > 2000.
2. Region does not touch an edge

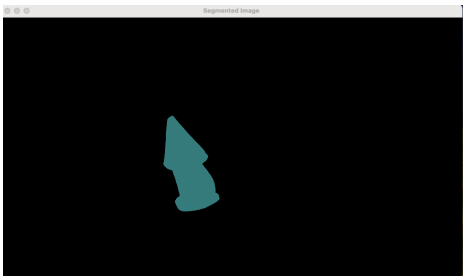

Figure 10: Segmented Chess Piece

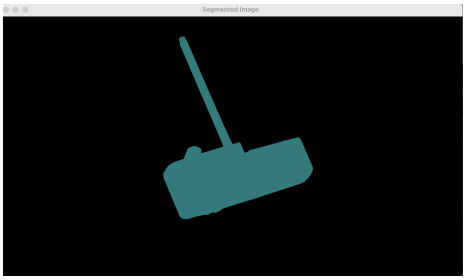

Figure 11: Segmented Thule Key

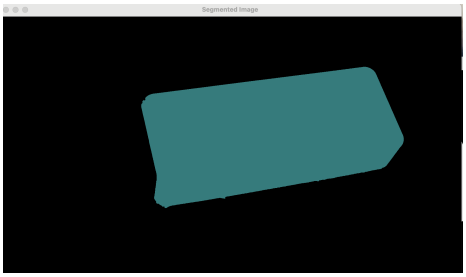

Figure 12: Segmented Eraser

Features

Next we computed features on the regoin of interest. We filtered regions for center most region and called **cv::moments** on that region. We used the output from the moments to compute the eigenvectors, and we used the eigenvectors to construct a minimal bounding box. We then used the bounding box to compute 2 features:

1. Percent Filled

2. Height Width Ratio

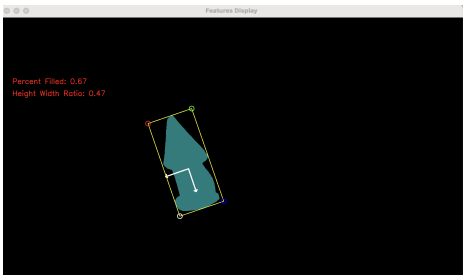

Figure 13: Axes, Bounding Box, Features - Chess Piece

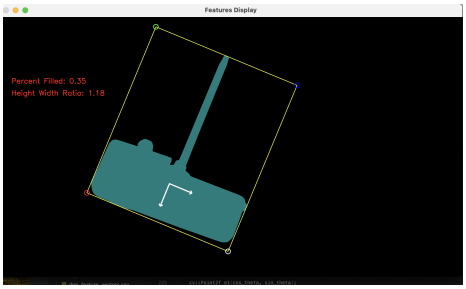

Figure 14: Axes, Bounding Box, Features - Thule Key

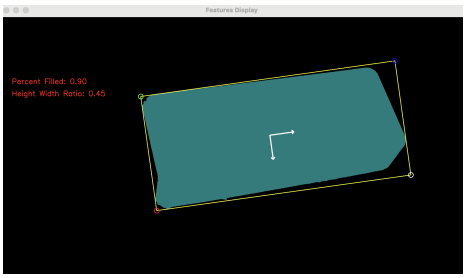

Figure 15: Axes, Bounding Box, Features - Eraser

Training

The images above are taken from the video stream. As seen, the feature vectors are being computed for every frame. Perhaps this is not the most efficient strategy. Nevertheless, when a user wants to add an image to the database i.e., enter training mode, all the user has to do is press **"t"** (or press **"d"** to enter

dnn training mode). This will enter a conditional and the user will be prompted to identify the the object.

"What is the label of this item? Please enter the name:"

The label is then stored. Because we have the associated feature vector all we do from here is append both to the database csv. The label is the first column, and the features make up the subsequent columns. The csv will take the from:

1. Col 1: Label
2. Col 2: Percent Filled
3. Col 3: Height Width Ratio

DNN funstions similarly except, the feature vector of the current frame is computed in the conditional.

The user will see this:

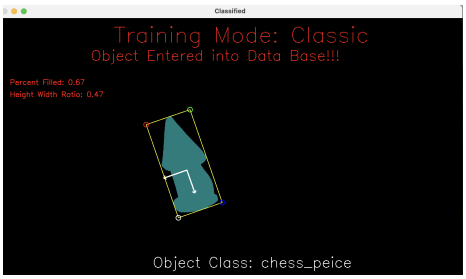

Figure 16: Training Mode

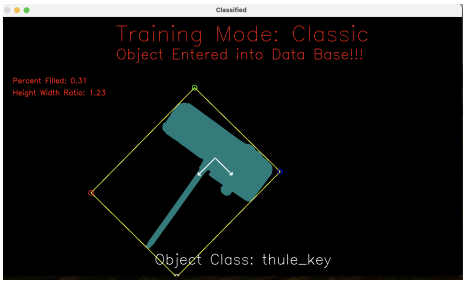

Figure 17: Training Mode

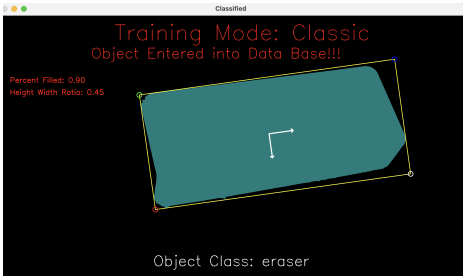

Figure 18: Training Mode

This is an example of what the user will see with **dnn training**:

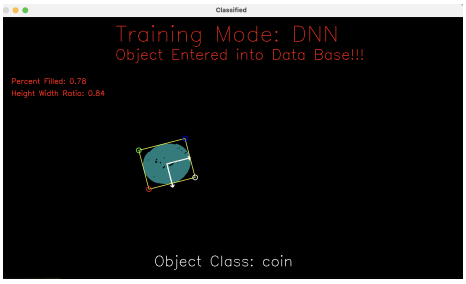

Figure 19: DNN Training Mode

Classify

To enter classify mode for classic features, all the user has to do is press **"c"**.

Then the user will prompted to enter the true label of the item. This information I used to track the results a create a confusion matrix.

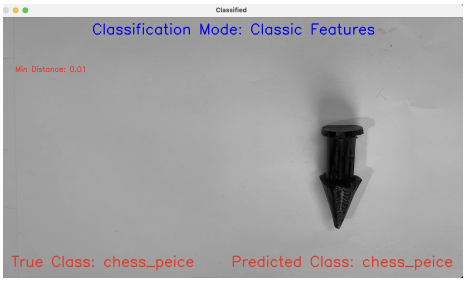

Figure 20: Classify Mode

Figure 21: Classify Mode

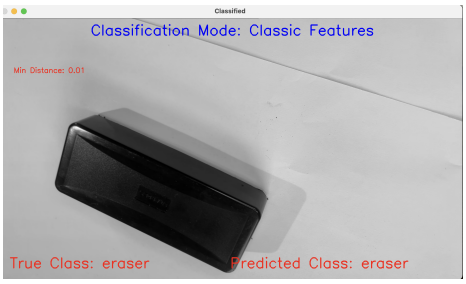

Figure 22: Classify Mode

Figure 23: Classify Mode

Figure 24: Classify Mode

To enter classify mode for DNN embedding, all the user has to do is press **"p"**.

This is an example of what the user will see in **dnn classification mode**

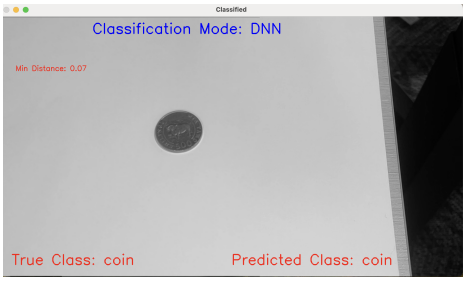

Figure 25: DNN Training Mode

Evaluation: Classic Features

To evaluate the classification functionality the user just has to run the program and enter classification mode. The user can enter classification mode as many times as they wish. When the user quits the program, the user will be asked if they wish to display the confusion matrix.

Then the user will prompted to enter the true label of the item. This information is used to track the results and create a confusion matrix.

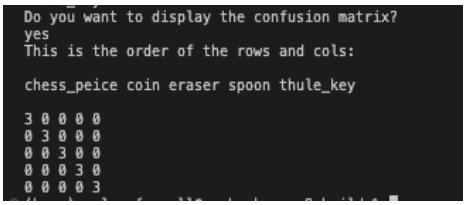

Figure 26: Confusion Matrix

As demonstrated, the classic features performed quite well, with an accuracy score of: 1. Where 1 is the best possible score.

Demo of System

This is a link to a google drive containing the demo video: CLICK HERE FOR LINK.

Implementation of Second Method

For a second classification method we used a **dnn embedding**. We implemented this second method in a similar manner as described above. The output the user sees when interacting with this second method have been displayed above.

To enter DNN training mode the user presses **"d"**.

This enters a conditional where a DNN embedding is created and entered into a database (csv file) with the label as the first column and feature vector as the subsequent columns (50 in this case). This was accomplished by modifying the code provided by Professor Maxwell.

To enter DNN classification mode, the user presses **"p"**. The DNN embedding for the current frame is then generated and compared against all the elements in the database. The current object is classified as the object it is closest to using cosine distance as a distance metric.

Comparing Results

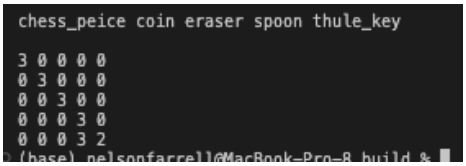

Figure 27: DNN Confusion Matrix

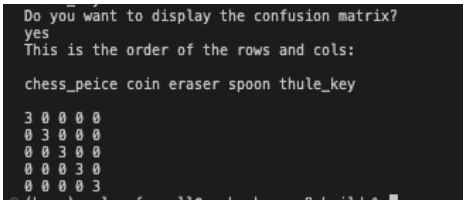

Figure 28: Confusion Matrix

As we can see, the DNN embedding performed slightly worse. It labeled the **thule_key** a **spoon** on the 3 occasions. This doesn't necessarily mean the DNN embedding is not as good as classic features for classification. Lighting variations likely played a role here as they did not remain constant.

The accuracy score for the DNN embedding was: 0.82

The accuracy score for the classic features was: 1.00

Extension

For the extension we gave our program the ability to determine when an object was not in the database. The program then informs the user and prompts the user to enter the object into the database. This was accomplished by empirically deriving a tolerance that would allow those objects that matched an object in the database to be identified, and those that did not match be parsed out.

The tolerance decided on was: distance > 0.15

This is an example of what the user sees when they attempt to classify an object not in the database.

Figure 29: Classic Feature Classification Mode

Figure 30: DNN Classification Mode

We attempted to code connected components with stats from scratch and made a lot of progress, but at the time of submission this was not yet fully functional.

We included the code for out connected components in the **until.cpp** file.

Reflection

In this project we learned many different aspects of image processing and feature generation. We learned how to threshold images, identify regions, compute region statistics, and use those statistics to create feature vectors. We not only learned how we accomplish this using **openCV** but also how we can write the algorithms from scatch. In the end we learned a great deal about image processing and object recognition.

Acknowledgements

1. OpenCV Documentation. https://docs.opencv.org/4.x/index.html
2. Stack Overflow. https://stackoverflow.com/
3. Geeks for Geeks. https://https://www.geeksforgeeks.org/
4. Bruce Maxwell. CS 5330 Class Materials and Code
    1. Kmeans Code
    2. DNN getEmbedding Code
    3. append_iamge_data_csv Code
    4. read_image_data_csv (adapted)