Project 3 Report

Name: Priyanshu Ranka (NUID: 002305396)

Semester: Spring 2025

Professor: Prof. Bruce Maxwell

Course: Pattern Recognition and Computer Vision

1. Project Overview

This project is about 2D object recognition. The goal is to have the computer identify a specified set of objects placed on a white surface in a translation, scale, and rotation invariant manner from a camera looking straight down. The computer should be able to recognize single objects placed in the image and show the position and category of the object in an output image. If provided a video sequence, it should be able to do this in real time. It includes parsing all these tasks-

Threshold the Input Video - Apply a thresholding algorithm to segment objects from the background, using pre-processing techniques like blurring and dynamic thresholding with K-means clustering (K=2) for optimal separation.

Clean Up the Binary Image - Use morphological operations like erosion, dilation, and closing to remove noise, fill gaps, and refine object boundaries for better segmentation.

Segment the Image into Regions - Perform connected component analysis to label different object regions, filtering out small and irrelevant ones while visualizing segmented objects with distinct colors.

Compute Features for Each Major Region - Extract key region features such as the axis of least central moment, aspect ratio, and percent filled area, ensuring scale and rotation invariance using OpenCV's moments.

Collect Training Data - Implement a training mode to collect feature vectors with user-assigned labels, storing them in a database for later classification from live video or pre-labeled images.

Classify New Images - Compare new objects' feature vectors with stored data using a scaled Euclidean distance metric, assigning labels via nearest-neighbor classification and displaying results in real-time.

Evaluate the Performance of the System - Test classification accuracy using a 5x5 confusion matrix by evaluating multiple images per object in different orientations, identifying misclassifications, and refining the system.

Capture a Demo of the System Working - Record a video demonstrating real-time object classification with thresholded images, segmented regions, and predicted labels, highlighting robustness across various conditions.

Implement a Second Classification Method - Develop an alternative classification method, such as K-Nearest Neighbors (KNN) with K>1, to improve accuracy by considering multiple nearest matches instead of a single comparison.

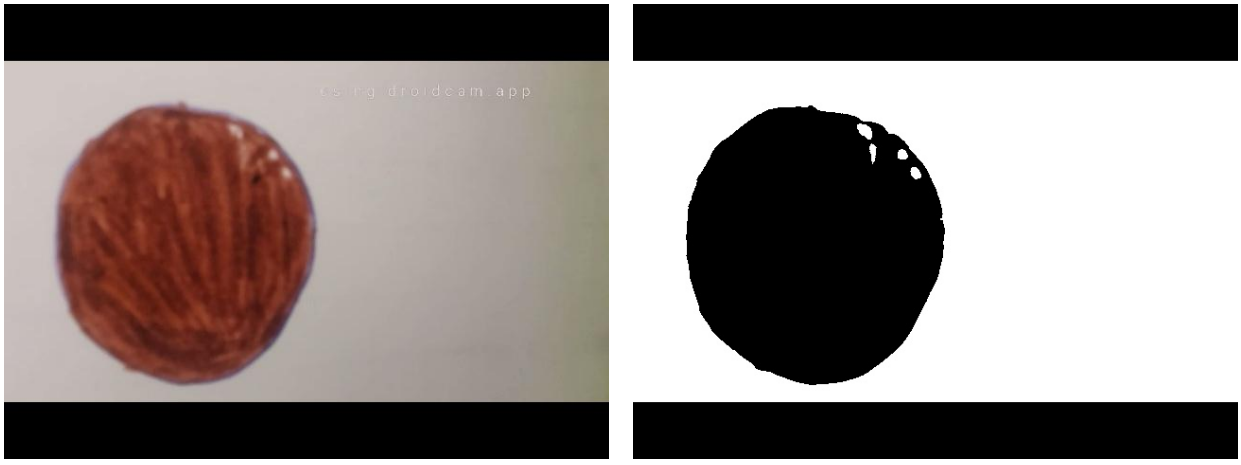
2. Tasks

2.1 Threshold the Input Video

Objective: Implement a thresholding algorithm to separate objects from the background. Use pre-processing techniques like Gaussian blurring for smoother segmentation. Apply dynamic thresholding using K-means clustering (K=2) to automatically determine an optimal threshold. Display the thresholded output and test it on different objects to ensure effective segmentation.

Approach: The thresholdImage function converts the input image to grayscale and applies a simple thresholding operation. The image is converted to grayscale using OpenCV's cvtColor() function. A Gaussian blur is applied to smooth the image, helping in noise reduction. Then, a pixel-by-pixel comparison is made, where pixels with intensity values less than or equal to a predefined threshold (110 in this case) are set to 255 (white), and the rest are set to 0 (black). This binary image helps segment regions of interest in the next steps.

Results:



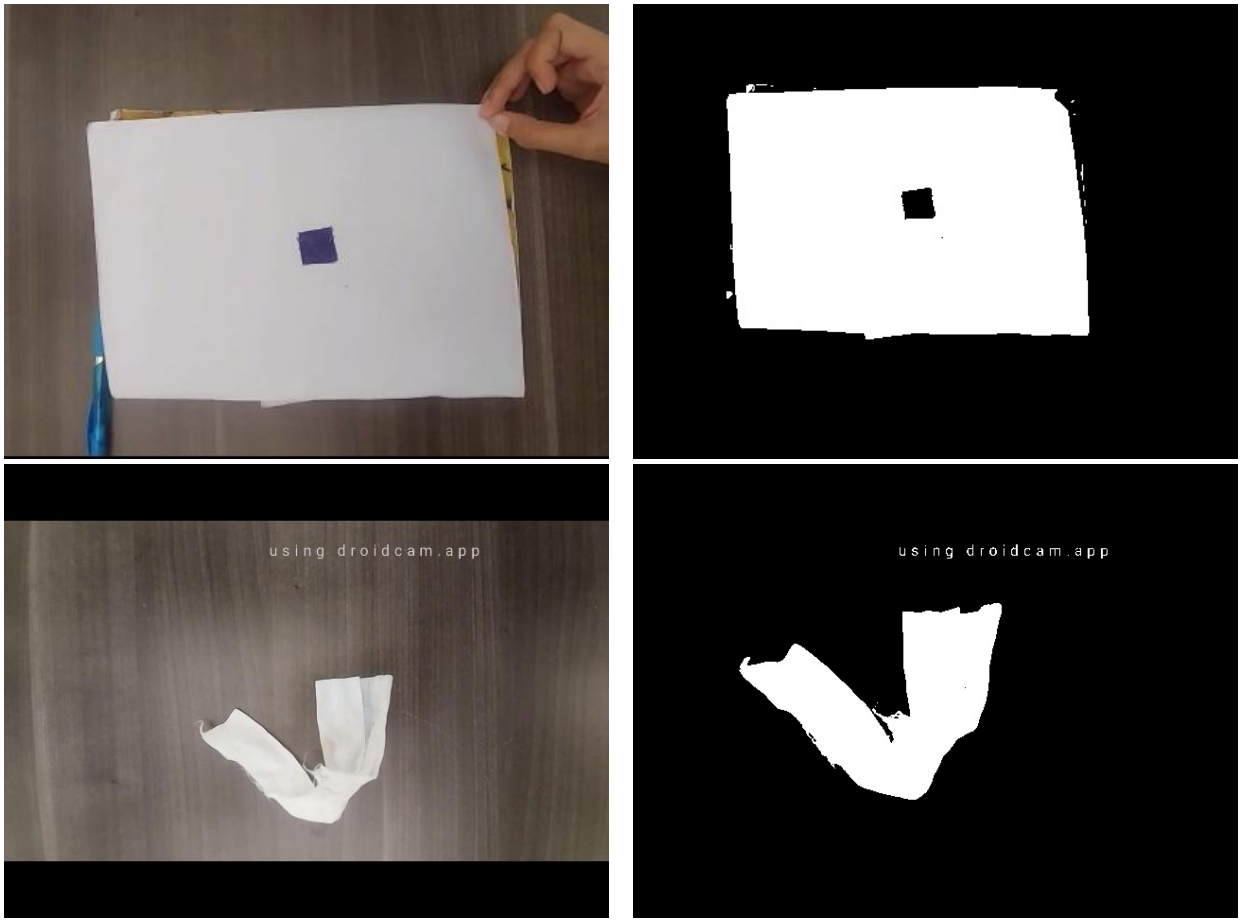


Figure 2.1: The target images are unevenly filled shapes, circle and square, and a tissue paper on dark background. The output threshold images are to the right of the target images

Note: For the 3rd example shown above, the thresholds were reversed. That is the background was considered dark, and the target/object was of lighter color. The result obtained thus had better boundaries and shapes than the previous ones. This was done to experiment and compare the results of different backgrounds and detecting the target object on them.

2.3 Clean Up the Binary Image

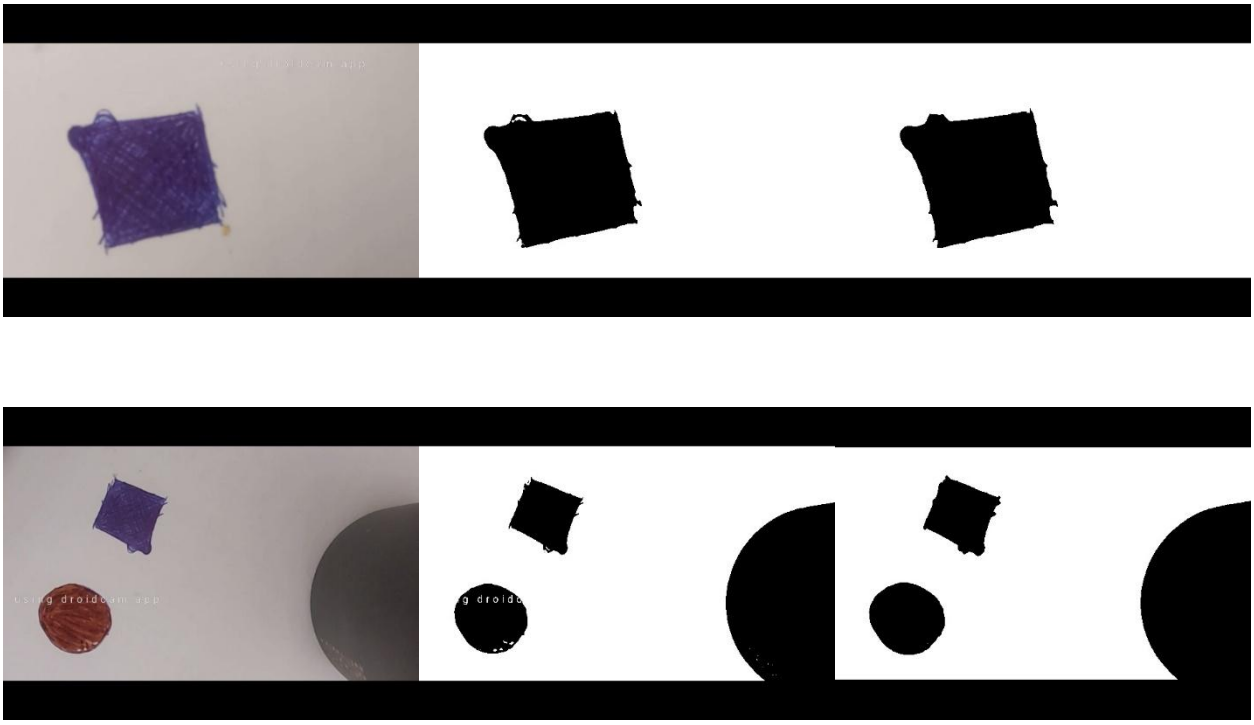
Objective: Use morphological operations like erosion, dilation, and closing to remove noise and refine object boundaries. Analyze the binary images to identify issues like small artifacts or holes and apply appropriate filtering. Ensure a cleaner, more defined object representation for accurate segmentation.

Approach: The cleanImage function cleans up the thresholded image using morphological operations, namely **dilation** and **erosion**. The image is first thresholded using

thresholdImage(). Then, dilation is applied using a kernel of size 2x2, followed by erosion with a larger kernel (2.5x2.5). This sequence helps remove small noise and fills in holes in objects, making the image more suitable for further analysis.

Alternatively, the morphologyEx() function is mentioned as a potential method to remove noise, but it's commented out in favor of the dilation-erosion approach.

Results:



Figures: Target images followed by their threshold and clean images.

Notice how in the clean images the small holes are filled up and the boundaries are clearer. This is the result of erosion function applied on the diluted image. This helps in the tasks ahead by making sure that regions are separated from one another and the detected regions are full and not have any scattered holes in them.

2.4 Segment the Image into Regions and Compute Features for Each Major Region

Objective: Perform connected component analysis to extract and label different regions. Use connectedComponentsWithStats to filter small, irrelevant regions and retain significant ones. Display segmented objects using distinct colors and ensure robust segmentation by focusing on the most central, well-defined objects.Extract key features like the axis of least central moment, aspect ratio, and percent filled area. Compute central moments using OpenCV's moments function

for rotation and scale-invariant feature extraction. Display the computed features in real-time for analysis and validation.

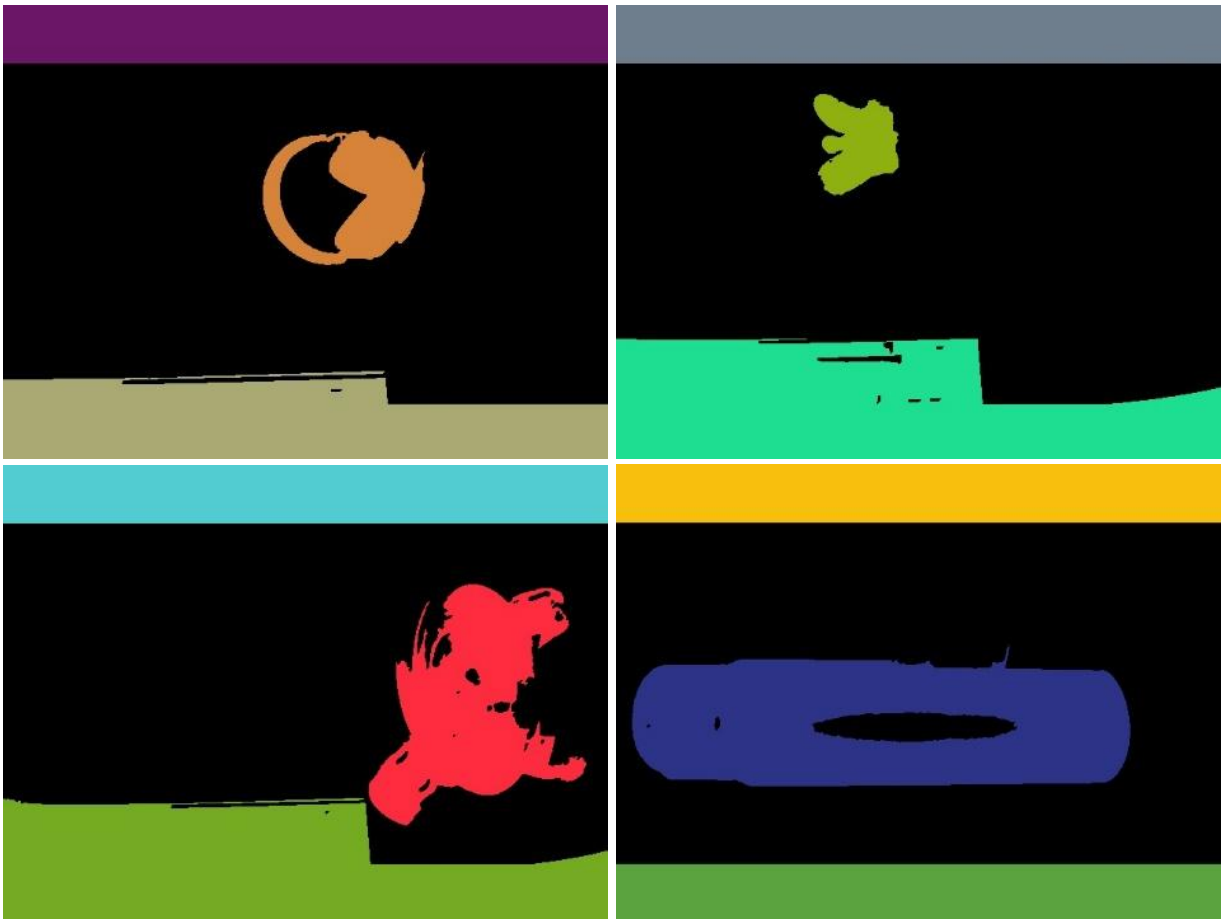
Approach: The *connectedComponentsAnalysis* function processes the thresholded image to identify connected components (regions) and assigns random colors to the largest components. The steps are as under-

Connected Components: *connectedComponentsWithStats* is used to identify connected regions in the binary image, returning the labels, statistics, and centroids of each component.

Filtering Components: A minimum area threshold (5000 pixels) is applied to ignore small regions. The remaining components are sorted by area, and only the top 5 largest regions are retained.

Color Assignment: Random colors are generated for the valid components using *generateRandomColors()*. These colors are applied to the labeled regions in a new image, visualizing the segmented regions.

Results:



Figures: The major regions marked separately with different colors

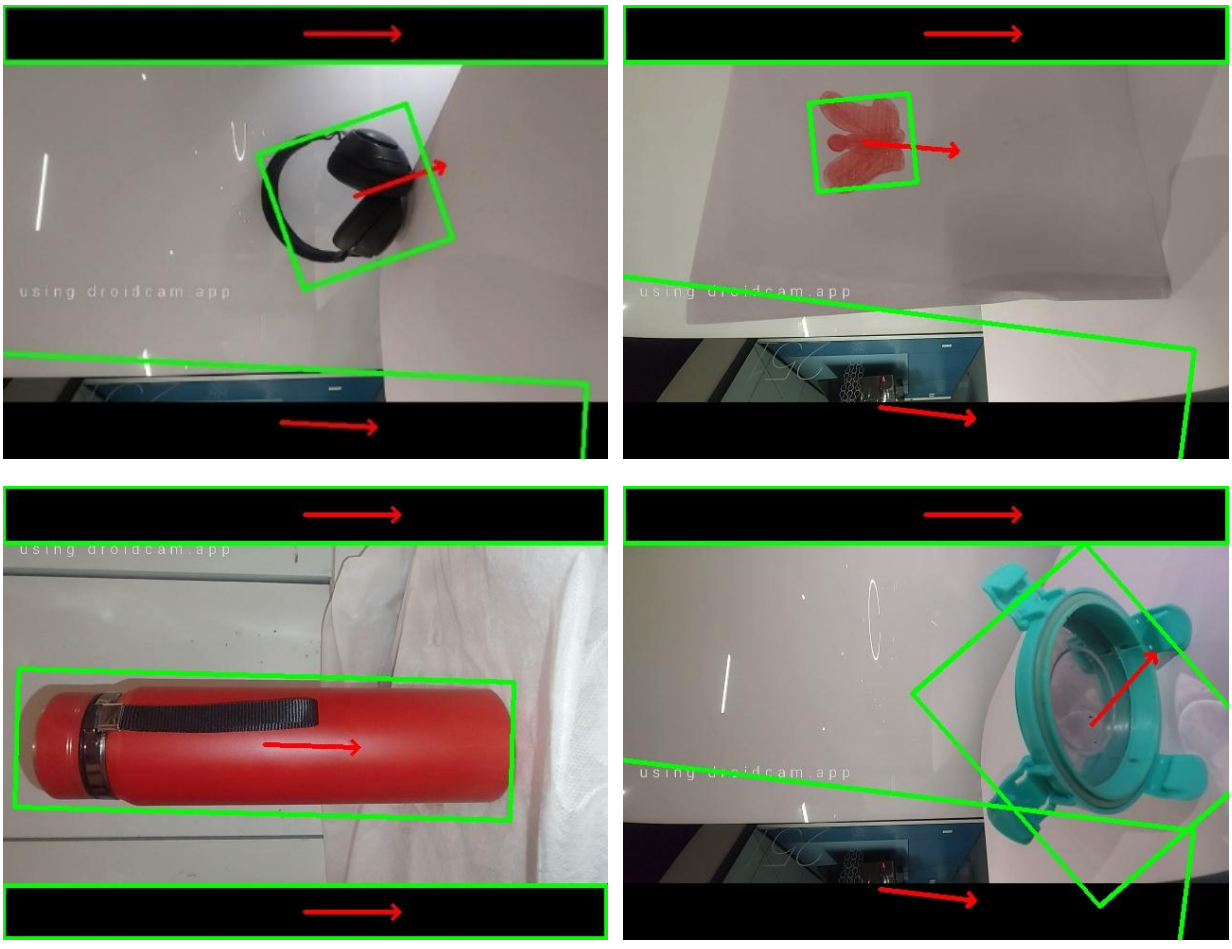
2.5 Collect Training Data and Classify New Images

Objective: Implement a training mode to collect and store feature vectors of labeled objects. Allow user input to assign labels to objects and save the data for later classification. Enable training from live video or pre-labeled images to build a reference object database.

Compare feature vectors of new objects with stored data using a scaled Euclidean distance metric. Assign labels based on the closest match using nearest-neighbor classification. Display the predicted label on the video feed and handle unknown objects effectively.

Approach:

Results:



Figures: The feature vector for each image from in the previous task

2.7 Evaluate the Performance of the System

Objective: Test the system on multiple images of each object in different orientations. Construct a 5x5 confusion matrix to analyze classification accuracy. Identify misclassifications, assess feature effectiveness, and refine the system to improve recognition performance.

Approach: A separate code file has been attached for the confusion matrix.

2.8 Capture a Demo of the System Working

Objective: Record a video showcasing real-time object classification. Display thresholded images, segmented regions, and predicted labels. Demonstrate the system's ability to handle different object orientations and lighting conditions. Provide a link to the video for external review.

Link to the video: Working_Video - Made with Clipchamp.mp4

Alt Link to the video: Working_Video

2.9 Implement a Second Classification Method

Objective: Develop an alternative classifier, such as K-Nearest Neighbors (KNN) with K>1. Train on multiple samples per object and compare features with multiple nearest matches. Improve classification accuracy by reducing errors in objects with similar features.

Approach:

Feature Extraction

The program captures video frames from a camera and processes them through several image processing steps, including thresholding, cleaning, and connected components analysis. Each detected region is analyzed to extract relevant features:

- Aspect Ratio and Filled Percentage to describe shape and size properties.
- Hu Moments to capture shape invariants.
- Orientation to determine the major axis of the detected object.

These extracted features form the basis for classification.

Training Mode

To enable supervised learning, the system provides a training mode. When activated (via key press 't'), the program prompts the user to label the detected objects. The extracted features, along with their respective class labels, are stored in features.csv. This dataset accumulates multiple samples per object over time, improving the classifier's robustness.

Classification Mode

In testing mode, the program loads the stored dataset and applies classification based on user selection:

- Pressing ‘n’ triggers a standard classification method.
- Pressing ‘k’ applies the KNN classifier (classifierKNN()), which currently operates with K = 3, assigning the label of the nearest stored feature.

You've asked to compare classification and classifierKNN functions, but you haven't provided the actual implementations of these functions. However, based on their names and typical KNN behavior, I can make a general comparison.

Comparison:

Feature	classification (Nearest Neighbor, k=1)	classifierKNN (K-Nearest Neighbors, k>1)
K Value	Fixed at 1.	Variable, determined by the K parameter.
Decision Basis	Class of the single nearest neighbor.	Majority class among the K nearest neighbors.
Robustness to Noise	Highly sensitive to noise and outliers. A single noisy neighbor can lead to misclassification.	More robust to noise. The majority vote smooths out the influence of outliers.
Decision Boundaries	Complex, jagged decision boundaries, prone to overfitting.	Smoother decision boundaries, reducing overfitting.
Variance	High variance. Small changes in training data can significantly change classification results.	Lower variance. Averaging among K neighbors makes results more stable.

Bias	Low bias, as it closely follows the training data.	Potentially higher bias, depending on the choice of K.
Computational Cost	Lower, as it only needs to find the single nearest neighbor.	Higher, as it needs to find the K nearest neighbors and perform a majority vote.
Generalization	May generalize poorly due to overfitting.	Often generalizes better due to smoother decision boundaries.

Key Differences Summarized:

- Majority Voting: The core difference is that classifierKNN uses majority voting among the K nearest neighbors, while classification relies solely on the single nearest neighbor.
- Robustness: KNN is generally more robust to noise and outliers.
- Overfitting: KNN is less prone to overfitting due to smoother decision boundaries.
- Flexibility: KNN provides more flexibility through the K parameter, allowing you to tune the classifier's behavior.

When to Use Which:

- Use classification (k=1) when:
    - The dataset is very clean and noise-free.
    - Computational speed is critical.
    - Accept the risk of overfitting.
- Use classifierKNN (k>1) when:
    - The dataset contains noise or outliers.
    - Reduce the risk of overfitting.
    - Improve generalization performance.
    - Use more computational power.