CS484: Introduction to Computer Vision
Homework 2

Yasemin Akın
22101782
Section-001

Due Date: November 30, 2024

1.0 Line Based Object Matching

In this part of the homework assignment, I developed a methodology to match template images with their rotated versions by using orientation histograms. I started by preprocessing each image through grayscale conversion, Gaussian smoothing, and Canny edge detection to isolate prominent edges. Then, I used the Hough Line Transform to detect lines, applying adaptive thresholding to ensure the desired number of lines was detected within a set number of iterations. Using the detected lines, I computed orientation histograms, weighting the line lengths and categorizing them into angular bins. To match rotated images to templates, I cyclically shifted the histograms to minimize their Euclidean distance, which also allowed me to estimate the rotation angles. I evaluated the results by comparing the matched templates with the ground truth and recorded metrics such as matching accuracy, rotation angles, and distances.

Below I have given 3 experiments as bad (5 matches), intermediate (7 matches), and good (9 matches -maximum I could do-) as the best representatives of the many experiments I conducted. Experiments with the same experiment names in different parts are related to each other. In other words, when the parameters used in experiment 1 in the hough lines section and experiment 1 in the histogram section come together, I get the results in experiment 1 in the matching results section.

1.1 Results for Edge Detection for Different Parameters

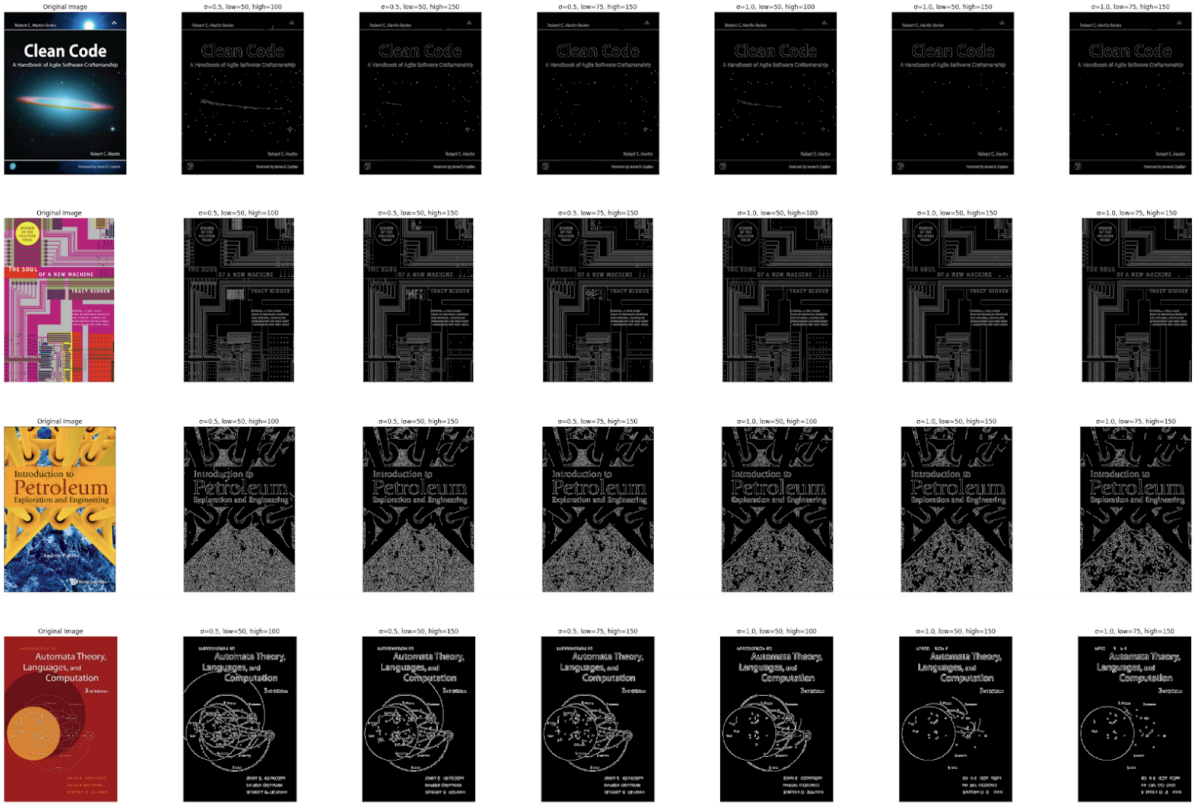

Figure 1: Results for Edge Detection for Different Parameters

Figure 1 illustrates effects of different parameters, i.e. sigma values, lower and upper threshold values, on different images. Sigma = 1.5 and lower and upper threshold tuple = (100, 200) were also tried but since these parameters demonstrated too much loss on information of edges, they were discarded. Here, we can see the effect of sigma = 0.5 and sigma = 1 on lower and upper threshold tuples = [(50, 100), (50, 150), (75, 150)] on some selected images that let us clearly differentiate between parameters and thus, let us make an effective choice of parameters. The threshold tuples are selected accordingly the common ratios of H_Threshold = 3 * L_Threshold or H_Threshold = 2 * L_Threshold. According to the outcomes and comparisons, selected parameters are: sigma = 0.5 and the threshold tuple = (50, 100).

The Canny Edge Detection implementation used here is based on OpenCV's documentation and tutorials [1].

1.2 Results for Line Detection (Hough Transform)

The Probabilistic Hough Transform implementation used here is based on OpenCV's documentation and tutorials [2].

I have used the new rotated images while doing this part of the homework.

Explanation of the desired_line_count and max_iterations parameters: Adaptive thresholding is employed to dynamically adjust the threshold parameter of the Probabilistic Hough Line Transform to achieve a desired number of detected lines. The algorithm iteratively increases or decreases the threshold based on whether the number

of detected lines falls outside a specified range (80%-120% of the desired count). The max_iterations parameter limits the number of adjustments, ensuring the process halts if the desired line count cannot be achieved within a reasonable number of attempts. This balances computational efficiency with the accuracy of line detection.

```
# Adaptive thresholding for line detection
  iteration = 0
  adaptive_threshold = desired_line_count
  while iteration < max_iterations:
  lines = cv2.HoughLinesP(edges, rho, theta, threshold=adaptive_threshold,
                          minLineLength=min_line_length, maxLineGap=max_line_gap)
    detected_line_count = len(lines) if lines is not None else 0

    if desired_line_count * 0.8 <= detected_line_count <= desired_line_count * 1.2:
      break
    elif detected_line_count < desired_line_count * 0.8:
      adaptive_threshold = max(int(adaptive_threshold * 0.8), 1)
    else:
      adaptive_threshold = int(adaptive_threshold * 1.2)
    iteration += 1
```

Experiment 1:
Parameters:
rho = 1
theta = π/180
min_line_length = 8
max_line_gap = 6
desired_line_count = 10
max_iterations = 10

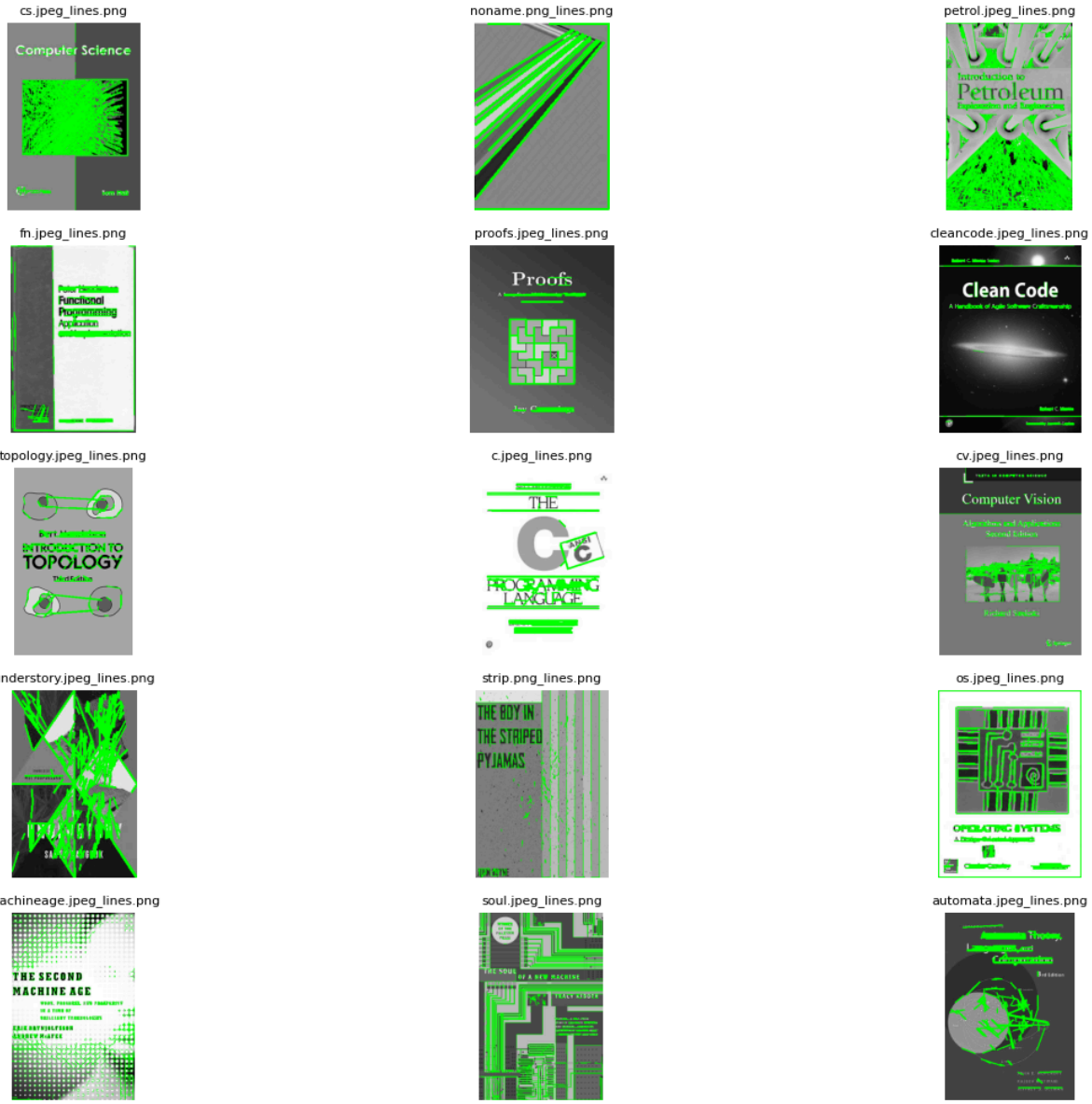

Figure 2: Experiment 1 Results for Original Images

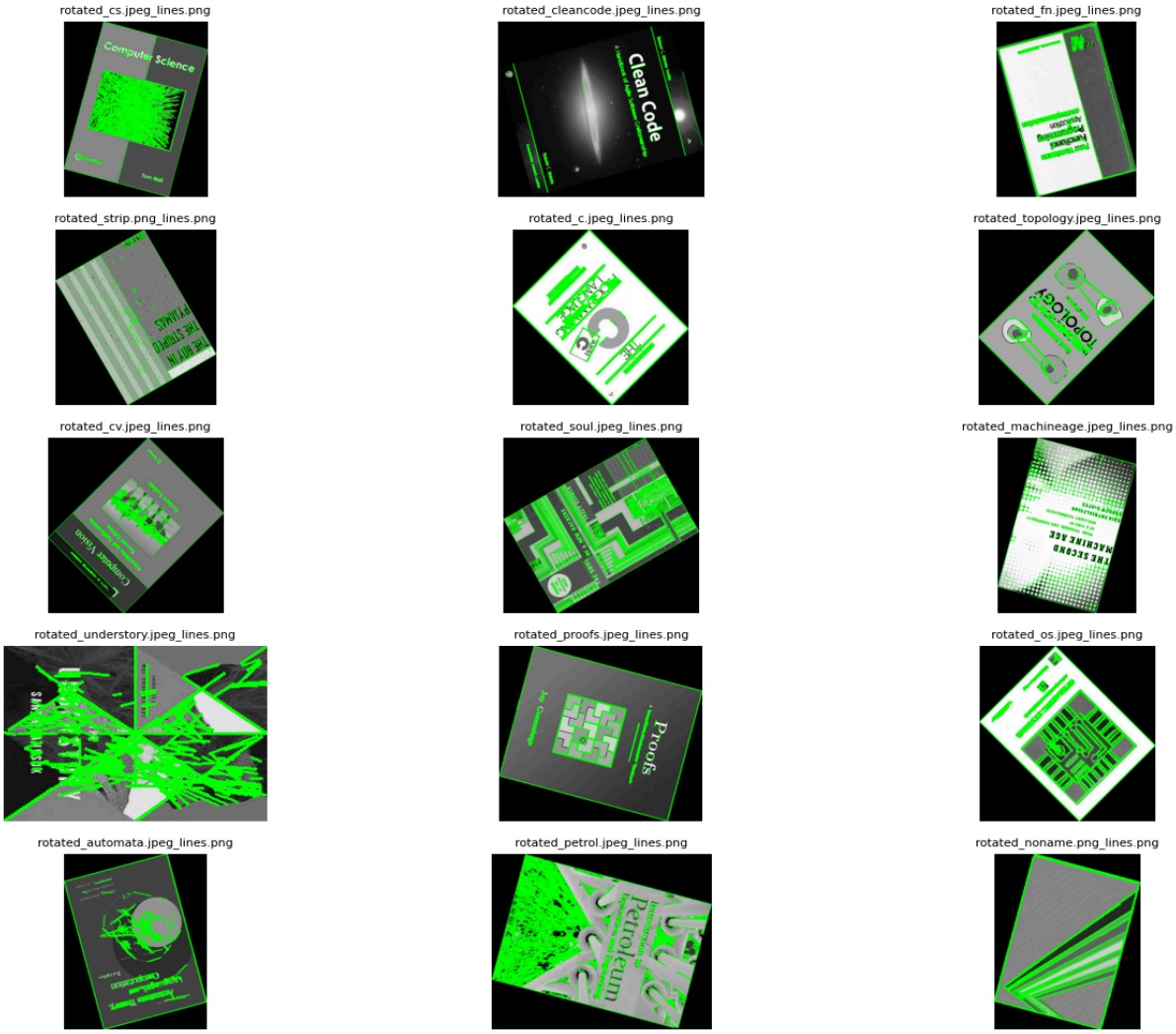

Figure 3: Experiment 1 Results for Rotated Images

Experiment 2:
Parameters:
rho = 1
theta = π/180
min_line_length = 8
max_line_gap = 8
desired_line_count = 6
max_iterations = 10

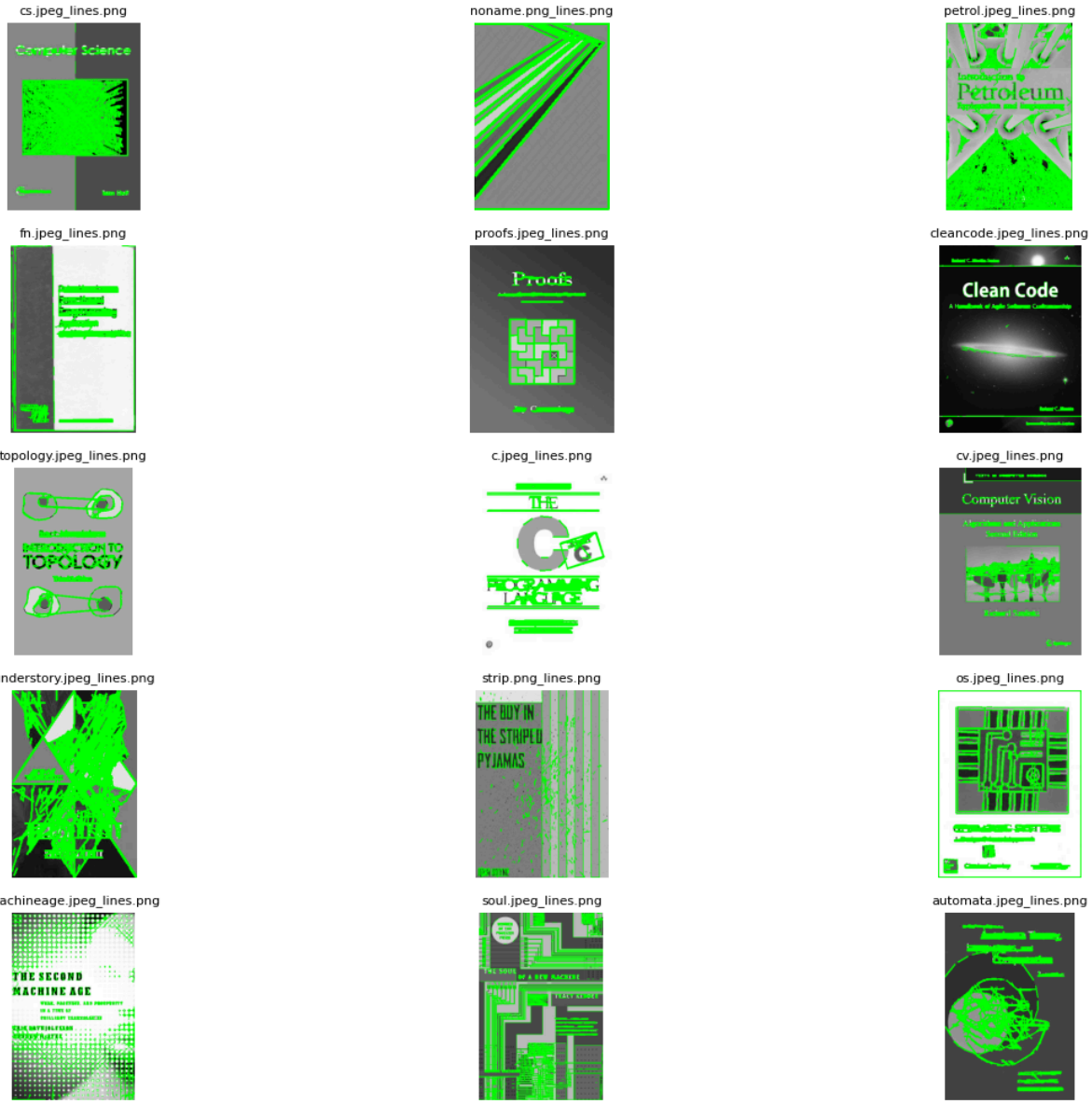

Figure 4: Experiment 2 Results for Original Images

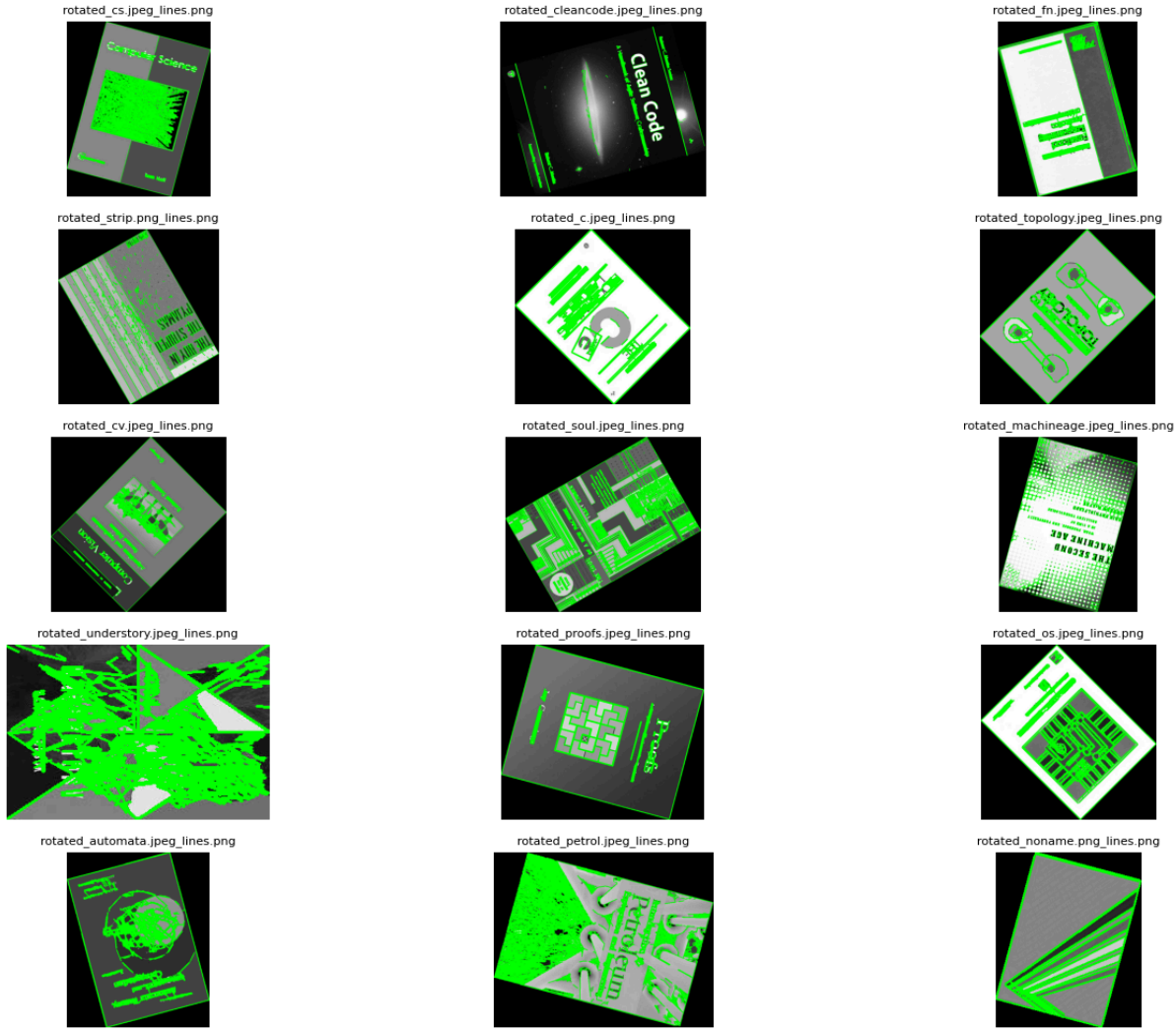

Figure 5: Experiment 2 Results for Rotated Images

Experiment 3:
Parameters:
rho = 1
theta = π/180
min_line_length = 10
max_line_gap = 4
desired_line_count = 6
max_iterations = 10

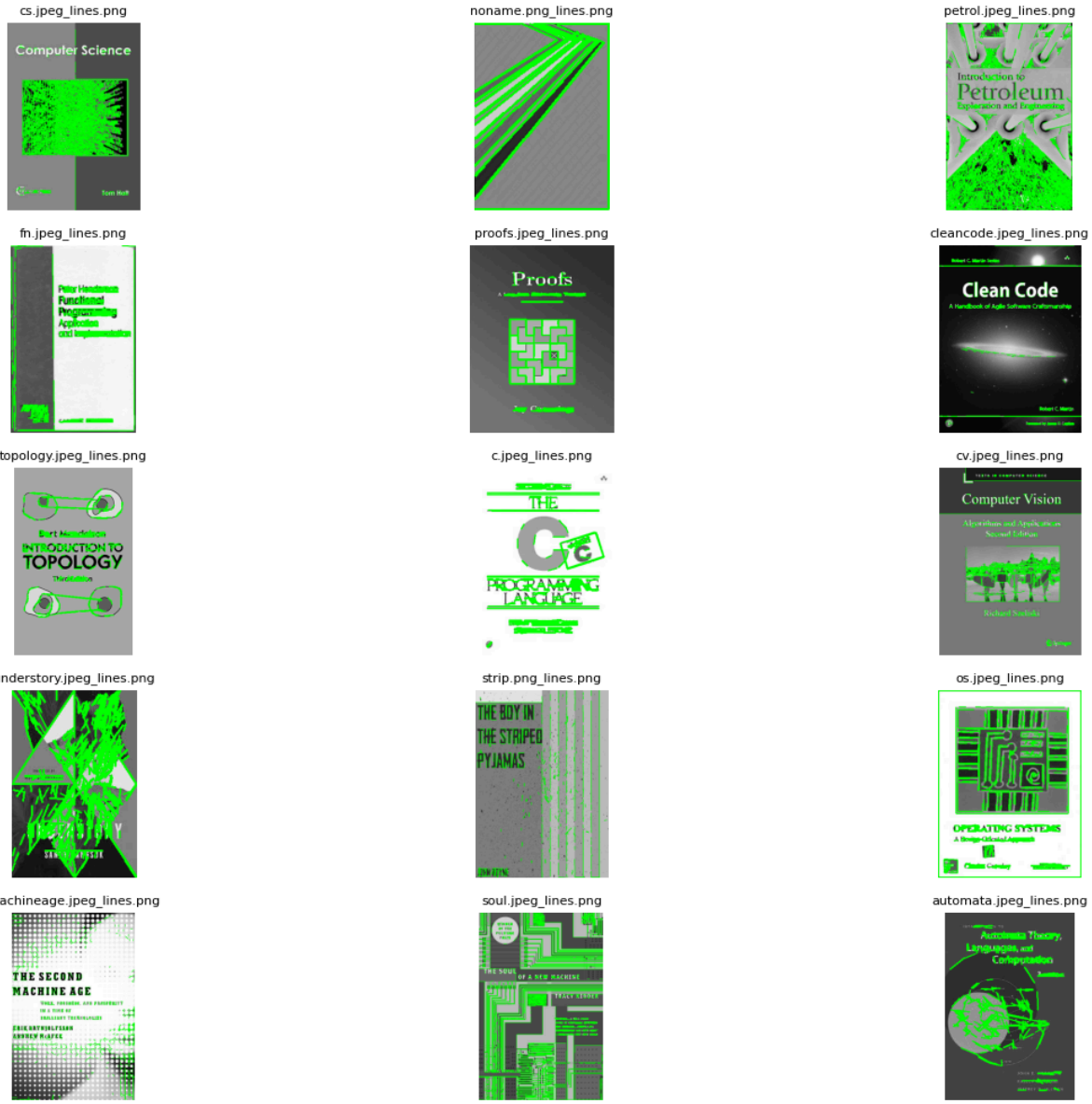

Figure 6: Experiment 3 Results for Original Images

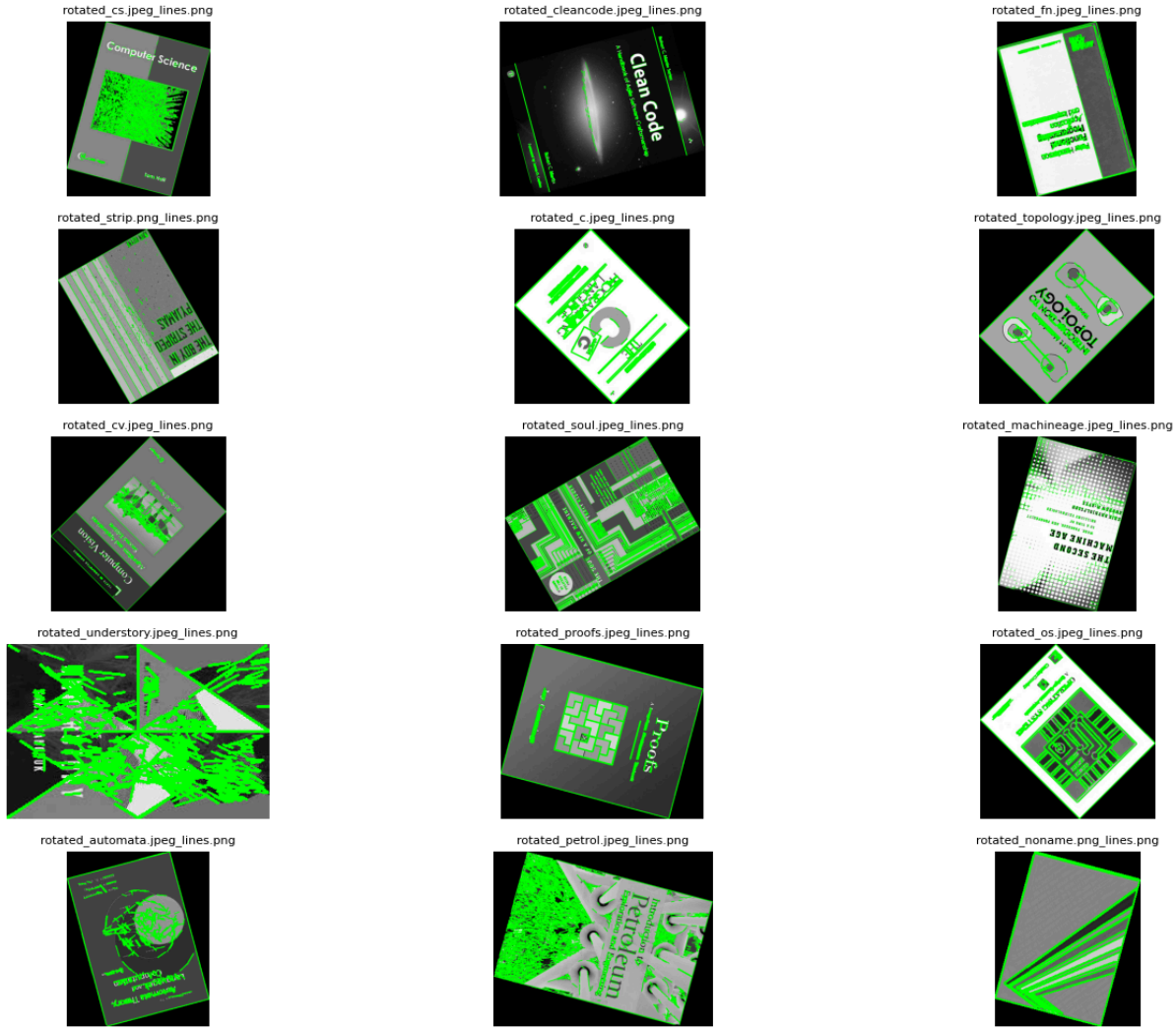

Figure 7: Experiment 3 Results for Rotated Images

1.3 Example Line Orientation Histograms

Experiment 1:
num_bins: 45

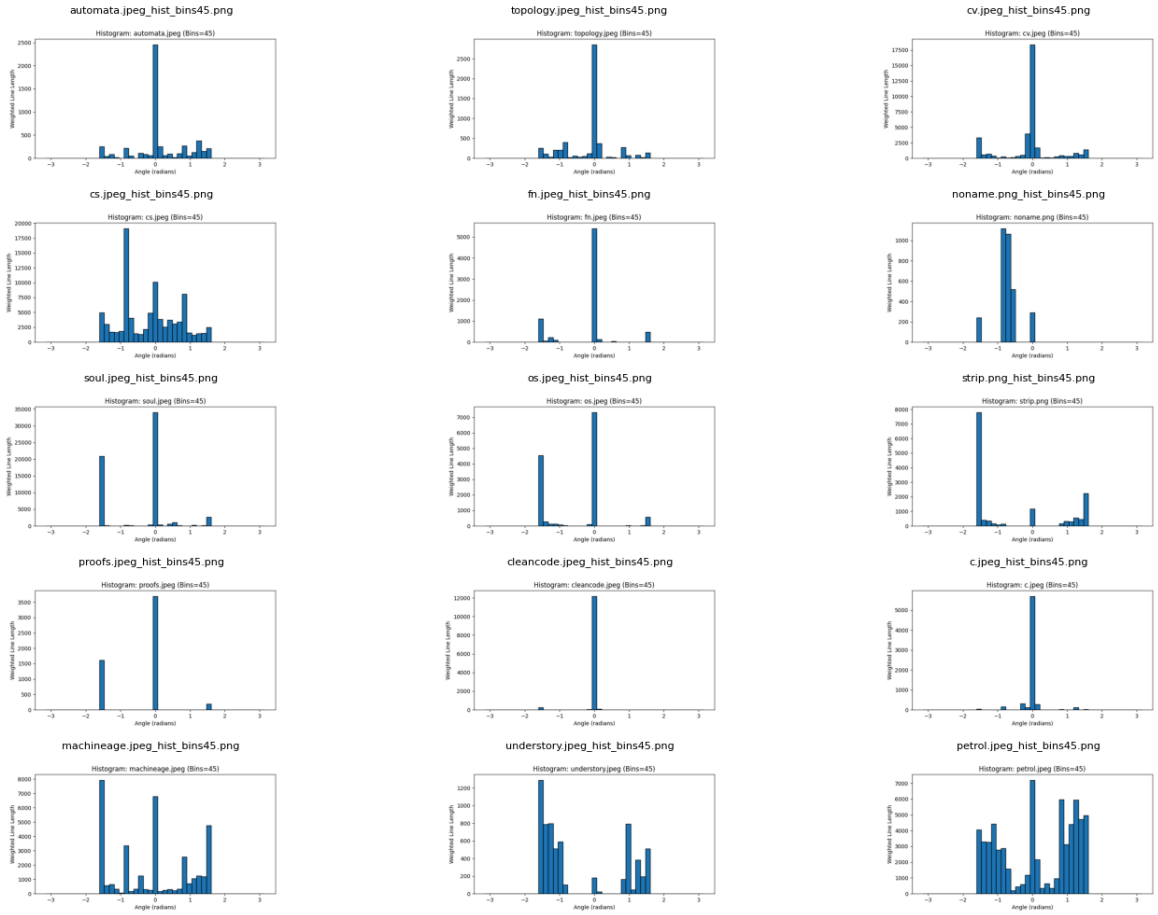

Figure 8: Experiment 1 Results for Original Images

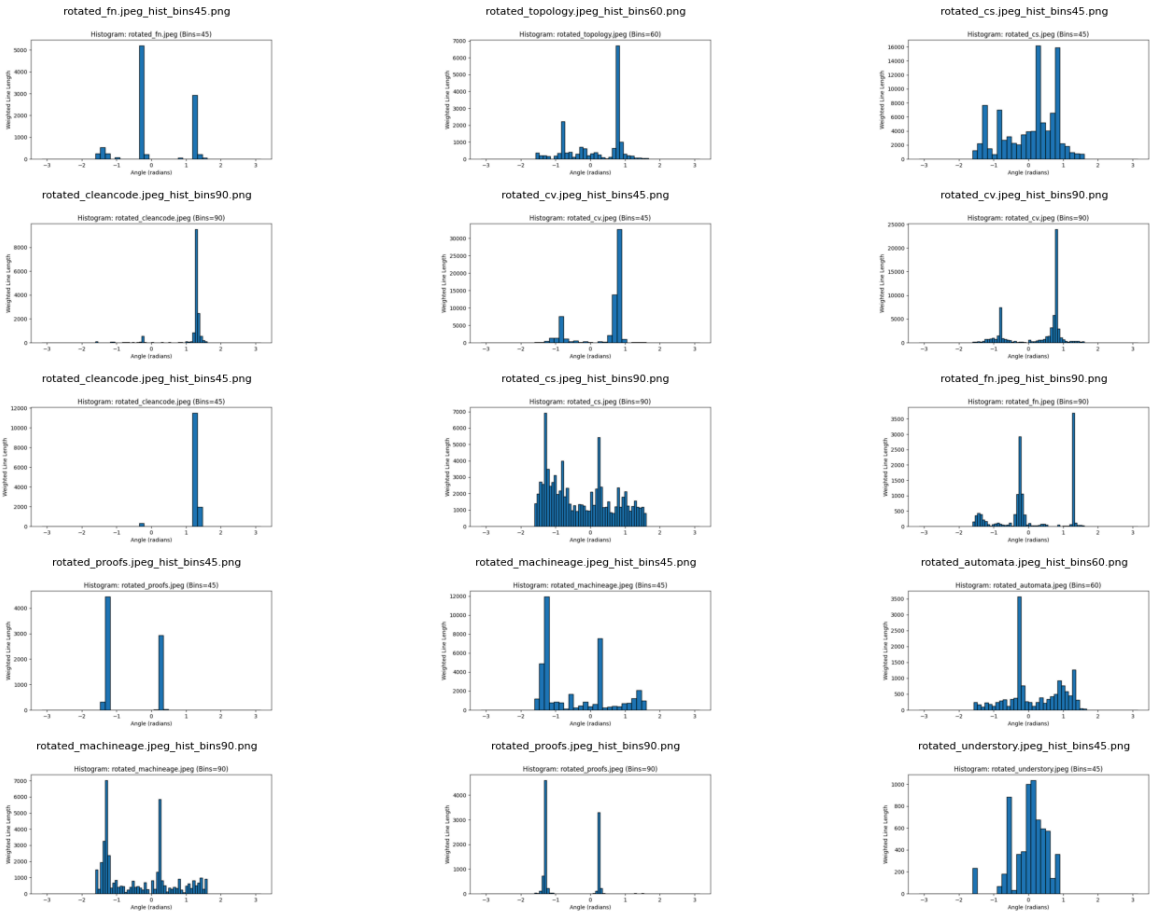

Figure 9: Experiment 1 Results for Rotated Images

Experiment 2:
num_bins: 60

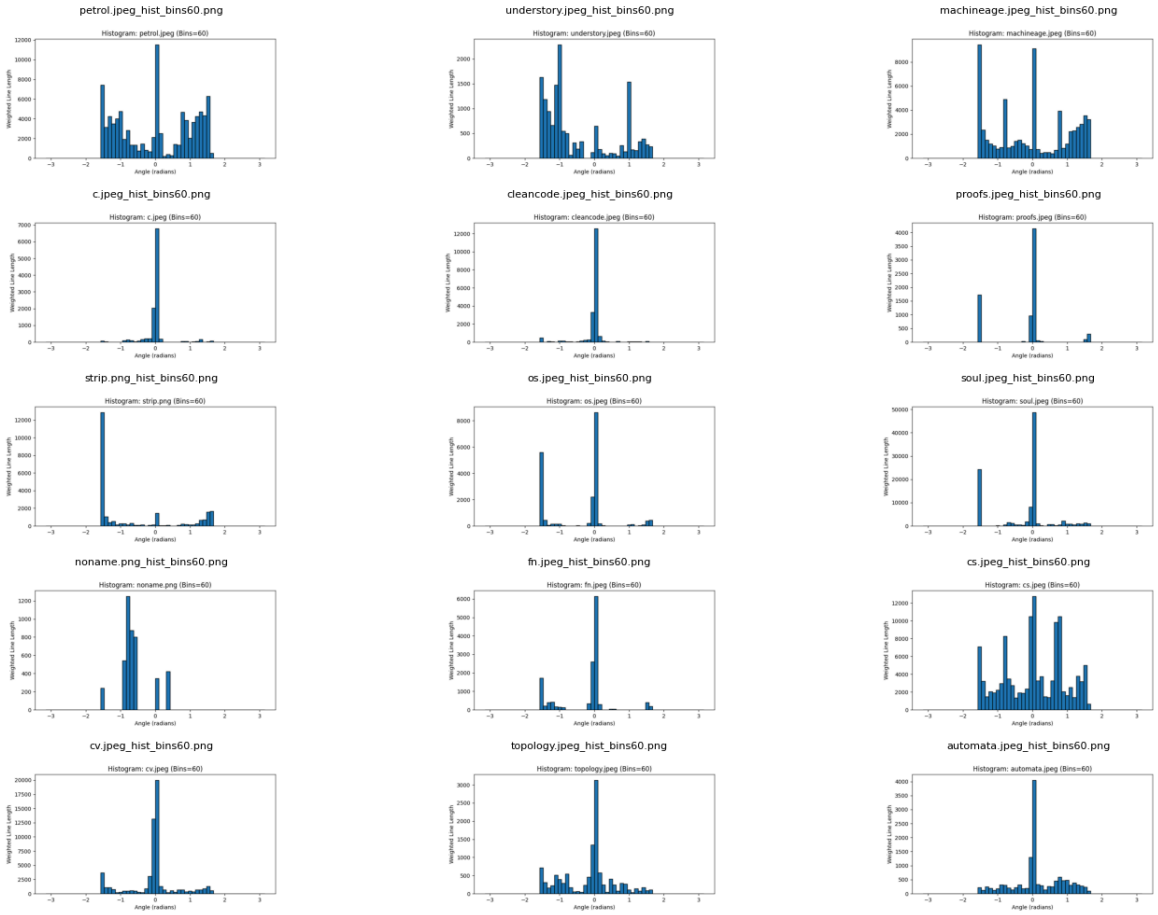

Figure 10: Experiment 2 Results for Original Images

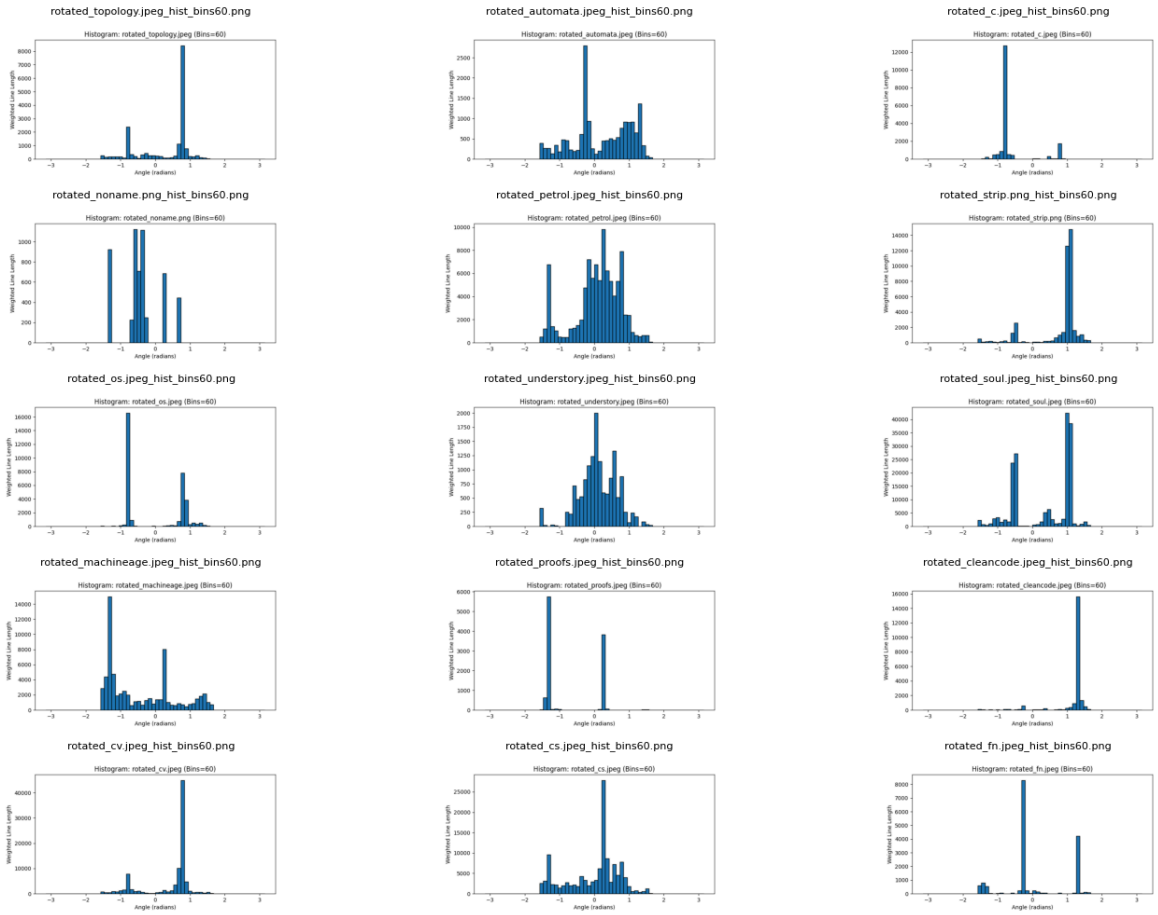

Figure 11: Experiment 2 Results for Rotated Images

Experiment 2:
num_bins: 90

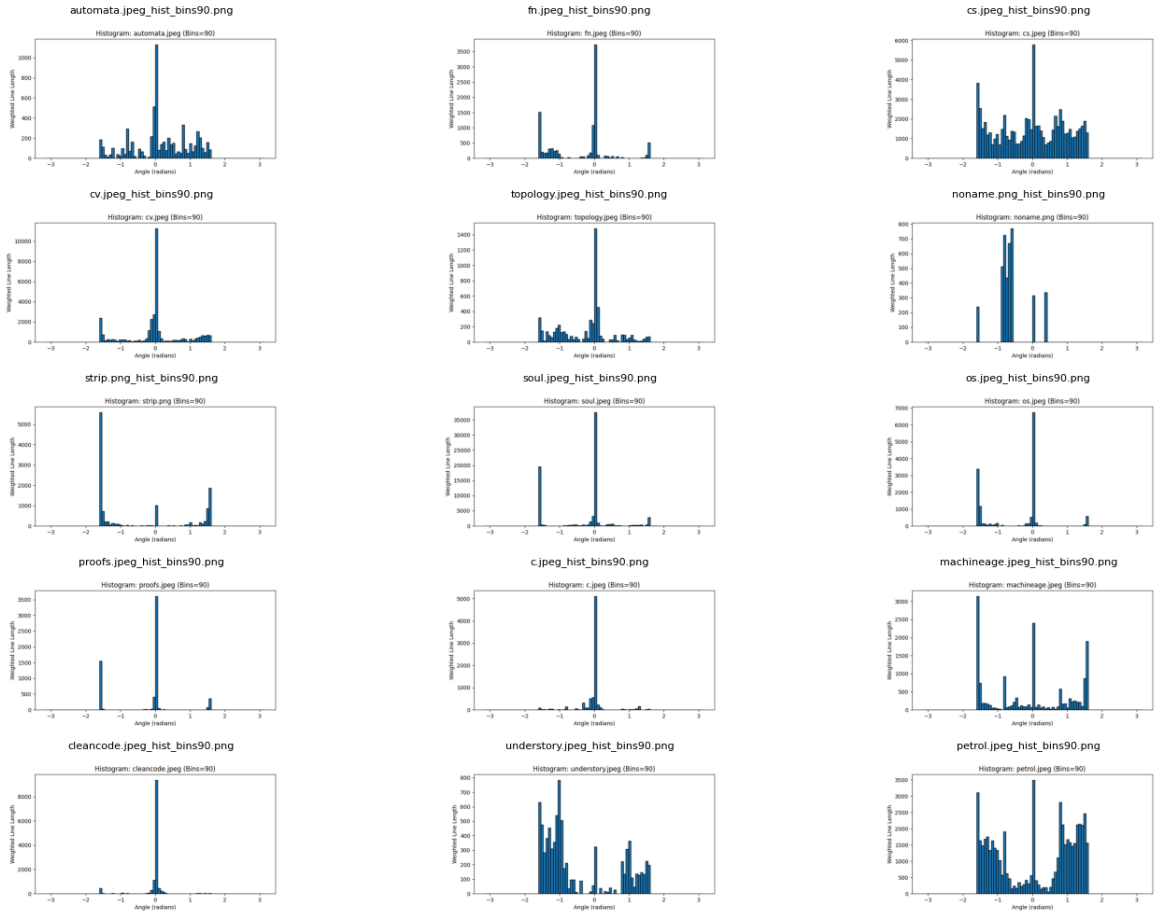

Figure 12: Experiment 3 Results for Original Images

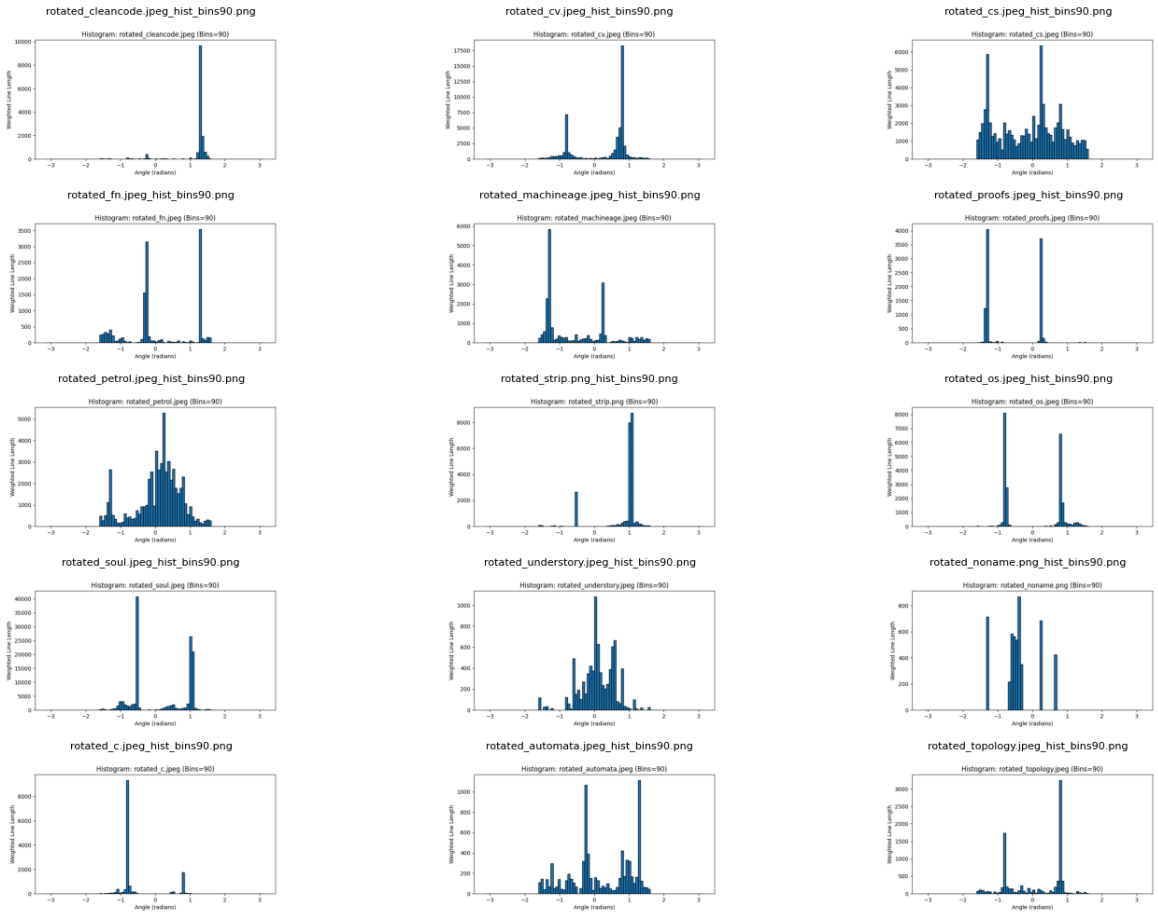

Figure 13: Experiment 3 Results for Rotated Images

1.4 Results for Matching the Books and Estimated Rotation Angles

Experiment 1: (Matching Result = 9/15 - Good Example)

Rotated Image	Matched Template	Rotation Angle (degrees)	Distance	Correct Match
rotated_automata.jpeg	automata.jpeg	16.0	2116.951839346056	TRUE
rotated_c.jpeg	strip.png	-40.00000000000001	3293.0387310671103	FALSE
rotated_cleancode.jpeg	cleancode.jpeg	-72.0	2911.6214739113266	TRUE
rotated_cs.jpeg	cs.jpeg	0.0	20373.999383801554	TRUE
rotated_cv.jpeg	cv.jpeg	-48.000000000000014	17416.191008925285	TRUE
rotated_fn.jpeg	c.jpeg	16.0	3520.5129231377564	FALSE
rotated_machineage.jpeg	machineage.jpeg	-15.999999999999972	9650.838294062092	TRUE
rotated_noname.png	noname.png	-15.999999999999972	868.68785872703	TRUE
rotated_os.jpeg	cleancode.jpeg	48.00000000000001	9674.120058127768	FALSE
rotated_petrol.jpeg	petrol.jpeg	0.0	17100.129990814192	TRUE
rotated_proofs.jpeg	proofs.jpeg	-15.999999999999972	3049.9723323977632	TRUE
rotated_soul.jpeg	soul.jpeg	-55.99999999999998	46254.17379622726	TRUE
rotated_strip.png	cleancode.jpeg	-63.999999999999986	7953.866534942909	FALSE
rotated_topology.jpeg	c.jpeg	-48.000000000000014	2105.9694967157116	FALSE
rotated_understory.jpeg	noname.png	-63.999999999999986	2118.123269541658	FALSE

Table 1: Results for Experiment 1

Experiment 2: (Matching Result = 5/15 - Bad Example)

Rotated Image	Matched Template	Rotation Angle (degrees)	Distance	Correct Match
rotated_automata.jpeg	automata.jpeg	18.0	2453.2195705130553	TRUE
rotated_c.jpeg	strip.png	-42.000000000000036	2948.298013281188	FALSE
rotated_cleancode.jpeg	cleancode.jpeg	-72.0	3966.916973747738	TRUE
rotated_cs.jpeg	cv.jpeg	-12.000000000000018	20111.4125184243	FALSE
rotated_cv.jpeg	soul.jpeg	-42.000000000000036	18251.809016354942	FALSE
rotated_fn.jpeg	c.jpeg	18.0	4936.237706206497	FALSE
rotated_machineage.jpeg	machineage.jpeg	-12.000000000000018	11198.654052238791	TRUE
rotated_noname.png	noname.png	-12.000000000000018	1124.5866517205548	TRUE
rotated_os.jpeg	strip.png	-42.000000000000036	8725.342230103295	FALSE
rotated_petrol.jpeg	machineage.jpeg	-12.000000000000018	18324.191634553525	FALSE
rotated_proofs.jpeg	c.jpeg	78.0	4188.695323779435	FALSE
rotated_soul.jpeg	soul.jpeg	-60.000000000000036	44235.785081986985	TRUE
rotated_strip.png	cv.jpeg	-60.000000000000036	6576.851027065506	FALSE
rotated_topology.jpeg	fn.jpeg	-42.000000000000036	3021.6959697327816	FALSE
rotated_understory.jpeg	topology.jpeg	0.0	2544.0153810485085	FALSE

Table 2: Results for Experiment 2

Experiment 3: (Matching Result = 7/15 - Intermediate Example)

Rotated Image	Matched Template	Rotation Angle (degrees)	Distance	Correct Match
rotated_automata.jpeg	automata.jpeg	16.0	1329.3715650478262	TRUE
rotated_c.jpeg	cleancode.jpeg	48.00000000000001	2041.3392400862308	FALSE
rotated_cleancode.jpeg	cleancode.jpeg	-72.0	1768.5232708827511	TRUE
rotated_cs.jpeg	cs.jpeg	-12.000000000000018	6658.551065320998	TRUE
rotated_cv.jpeg	cv.jpeg	-44.000000000000014	9332.342981501686	TRUE
rotated_fn.jpeg	proofs.jpeg	-72.0	3284.5560620280503	FALSE
rotated_machineage.jpeg	c.jpeg	76.0	3870.2817230721084	FALSE
rotated_noname.png	noname.png	-15.999999999999972	1053.3892676532926	TRUE
rotated_os.jpeg	os.jpeg	-44.000000000000014	5294.6073271489895	TRUE
rotated_petrol.jpeg	cs.jpeg	-12.000000000000018	7872.956114987601	FALSE
rotated_proofs.jpeg	fn.jpeg	76.0	3636.164404206257	FALSE
rotated_soul.jpeg	soul.jpeg	-59.999999999999986	36402.9776888347	TRUE
rotated_strip.png	cv.jpeg	-59.999999999999986	6504.730449823968	FALSE
rotated_topology.jpeg	proofs.jpeg	-44.000000000000014	857.5217008580512	FALSE
rotated_understory.jpeg	automata.jpeg	0.0	1344.6939064991686	FALSE

Table 3: Results for Experiment 3

The experiments highlight the influence of parameter configurations on the performance of the image matching process, specifically in terms of the number of correct matches achieved. Experiment 1 resulted in 9 matches out of 15 using moderate min_line_length (8) and max_line_gap (6) with num_binsset to 45. These settings provided a balanced approach to line detection, capturing a sufficient number of features without excessive noise. The relatively lower number of bins (45) allowed for a broad angular resolution, which might have contributed to better matching for images with less complex rotations or alignments.

In Experiment 2, the performance dropped significantly to 5 matches, largely due to the increased max_line_gap (8) and num_bins (60). The higher gap likely allowed for overly lenient line connections, introducing noise or inaccuracies in the detected line orientations. Additionally, the increase in num_bins provided finer angular resolution but may have fragmented the orientation histogram, making matching more sensitive to small discrepancies.

Experiment 3 achieved intermediate results with 7 matches, leveraging a stricter min_line_length (10) and lower max_line_gap (4) combined with a higher num_bins (90). The stricter line length requirement ensured the detection of prominent lines, while the high number of bins increased angular precision. However, the stringent parameters might have overlooked weaker features, reducing the histogram's ability to capture the full range of line orientations in complex scenarios.

Overall, the analysis suggests that a balanced configuration of parameters is critical. Parameters like max_line_gap and num_bins have trade-offs; looser settings can introduce noise, while stricter values may omit relevant features. Experiment 1's settings provided the best compromise, achieving the highest accuracy, while Experiment 2 suffered from oversensitivity, and Experiment 3 was overly conservative, leading to reduced matches. This demonstrates the need for careful tuning of parameters to adapt to specific image characteristics and matching goals.

2.0 Image Segmentation with Superpixels

In this part of the homework assignment, an image segmentation pipeline to process Swiss mountain images is implemented. The goal was to achieve accurate segmentation by over-segmenting images into superpixels using the SLIC algorithm and refining the results through clustering and contextual representation. Superpixel segmentation parameters, such as k, m, and max_iter, were carefully chosen to balance accuracy and efficiency.

Gabor texture features were extracted for each superpixel to capture patterns at multiple scales and orientations. These features were clustered using k-means to generate an initial segmentation. However, this approach lacked spatial coherence in some cases. To address this, I incorporated contextual representation by calculating features from neighboring superpixels, defined by ring radii, and concatenated them with the original features. This enhanced the spatial and contextual awareness of the clustering.

Refined segmentation maps were produced by clustering these contextual features. The results were visualized using pseudo-color overlays and boundary-marked images. Iterative parameter tuning, particularly for k and ring multipliers, allowed me to optimize the segmentation results, effectively capturing both texture and spatial structure of the images.

2.1 Parameters Used for Superpixel Segmentation

1. k: Specifies how many superpixels the image will be divided into. Higher k creates more, smaller superpixels; captures finer details, increases computation time; lower k produces fewer, larger superpixels; may miss small features, faster computation.

2. m: Controls the trade-off between spatial proximity and color similarity. Higher m emphasizes spatial proximity, superpixels are more regular and compact, less adherence to edges; lower m prioritizes color similarity, superpixels closely follow image edges, shapes may be irregular.

3. max_iter: Determine how many times the algorithm refines superpixel boundaries. More iterations lead to better-defined superpixels, increase processing time; fewer iterations lead to faster execution, and may result in less accurate segmentation.

4. S: Approximate size of each superpixel. Directly affects the superpixel size and limits the search area, by defining the local region for clustering, affecting the speed and accuracy.

2.2 Segmentation Results Using SLIC

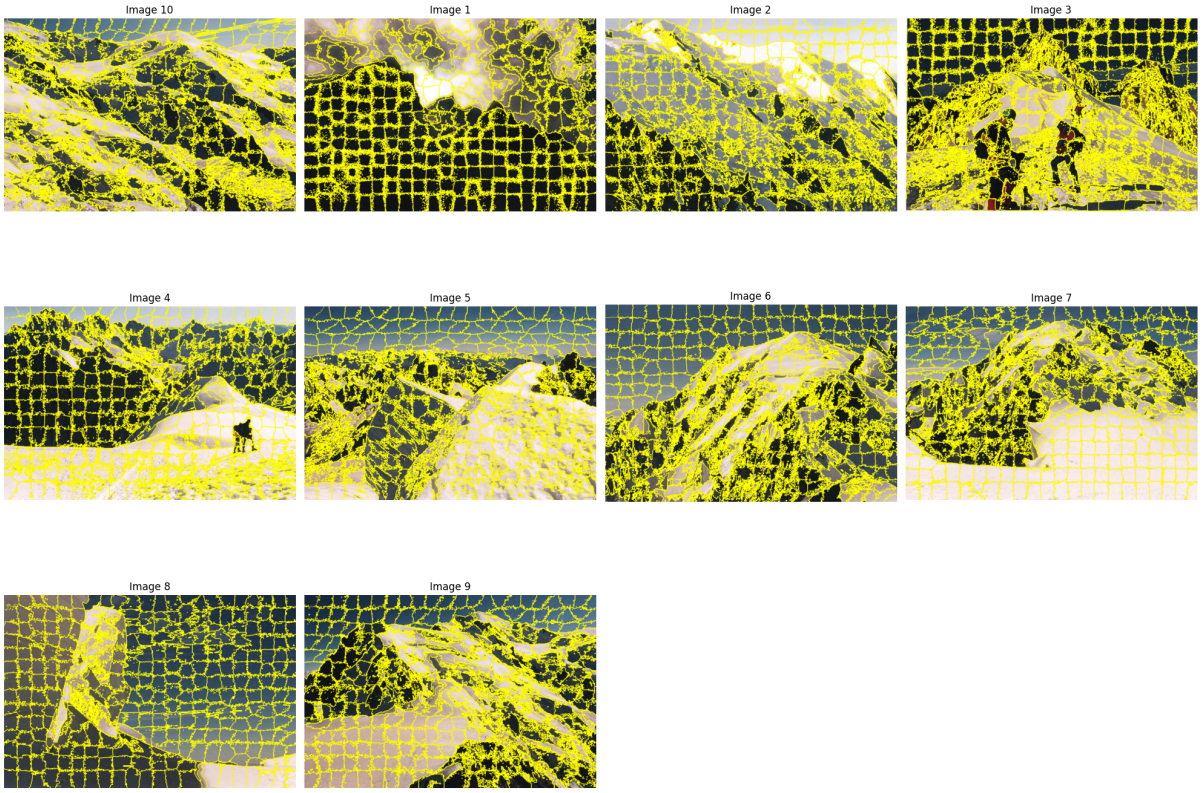

Figure 14: Segmentation Results for All Images

2.3 Parameters for Gabor Texture Feature Extraction

1. Minimum wavelength (minWaveLength): Defines the wavelength of the smallest scale Gabor filter, controlling the highest frequency (smallest detail) that can be detected.

2. Scaling factor (mult): Determines how the wavelength increases for each successive scale, effectively doubling the wavelength to cover a broad range of spatial frequencies.

3. Number of scales (num_scales): Specifies the number of different scales (frequencies) at which the Gabor filters are applied, capturing textures at multiple levels of detail.

4. Number of orientations (num_orientations): Indicates the number of orientations (directions) for the Gabor filters, enabling detection of texture patterns in different directions.

5. Frequency bandwidth (sigmaOnf): Controls the bandwidth (blurriness) of the filters in the frequency domain; higher values result in broader filters that are less selective in frequency.

6. Angular bandwidth (dThetaOnSigma): Controls the selectivity of the filters to different orientations; higher values increase the angular bandwidth, making the filters less orientation-specific.

2.4 Gabor Texture Feature Examples

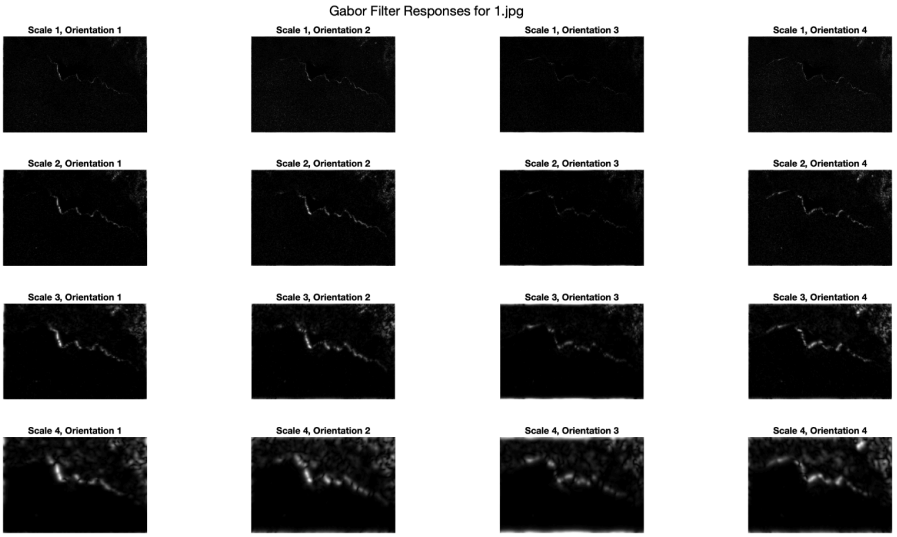

Figure 15: Gabor Filter Responses for Image 1

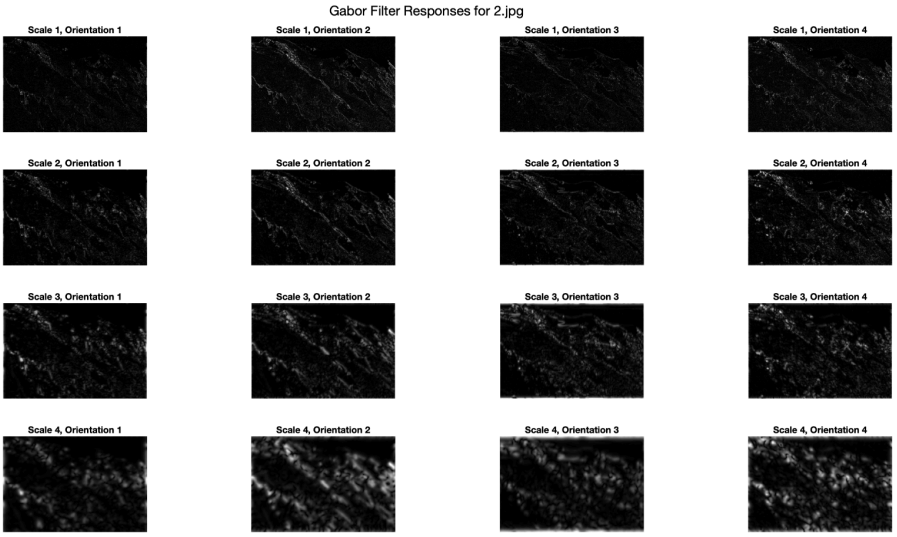

Figure 16: Gabor Filter Responses for Image 2

2.5 Clustering/Segmentation Results

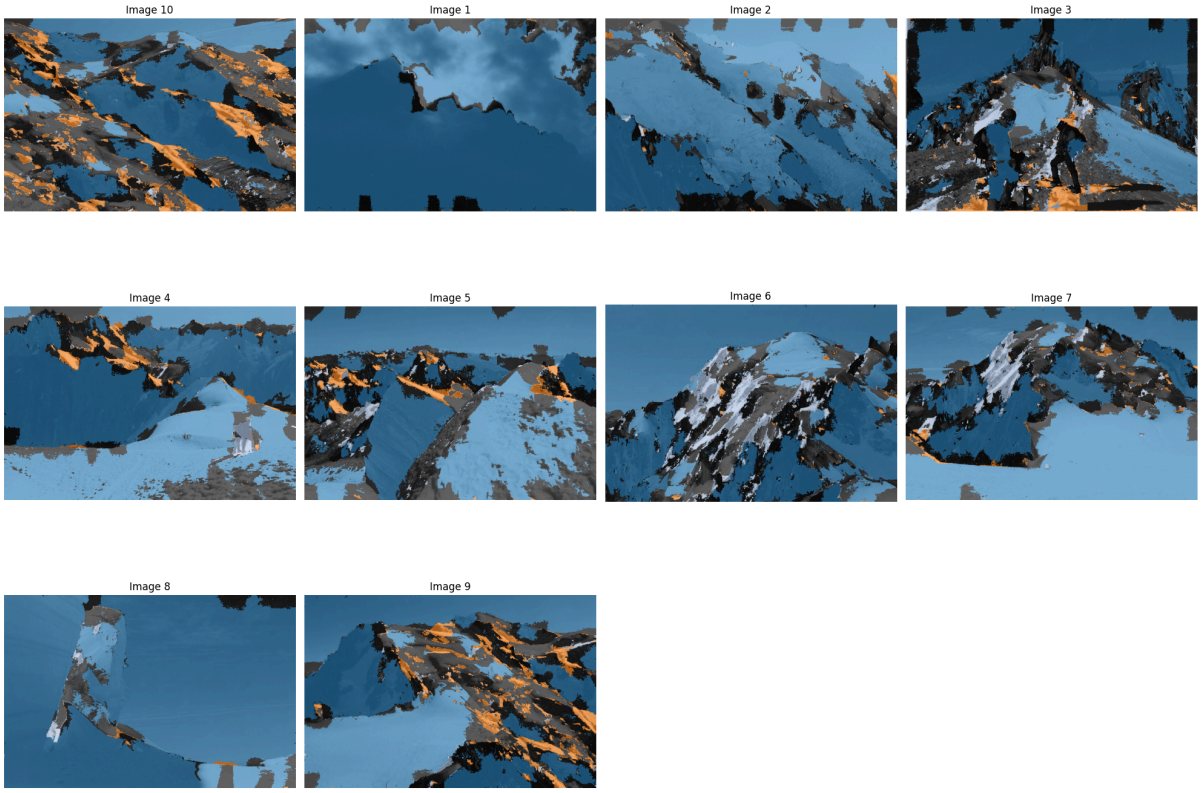

Figure 17: Results for Part 3

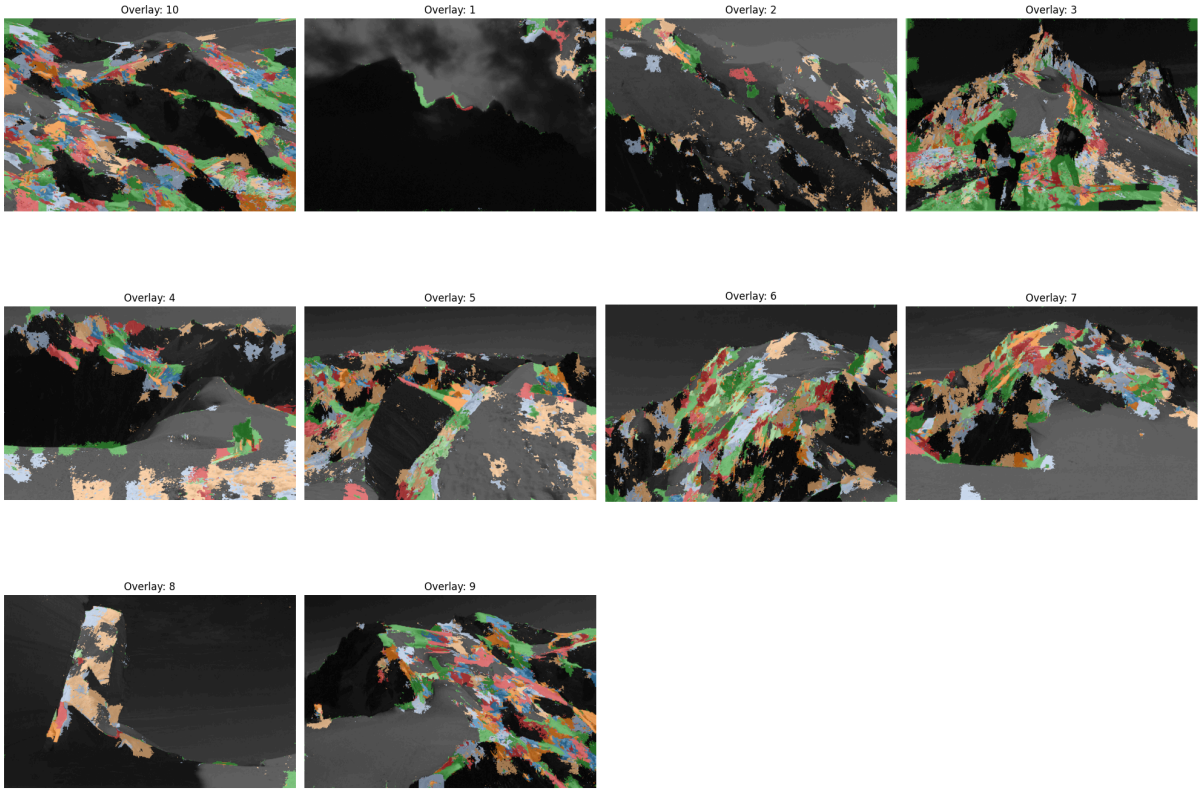

Figure 18: Results for Part 4

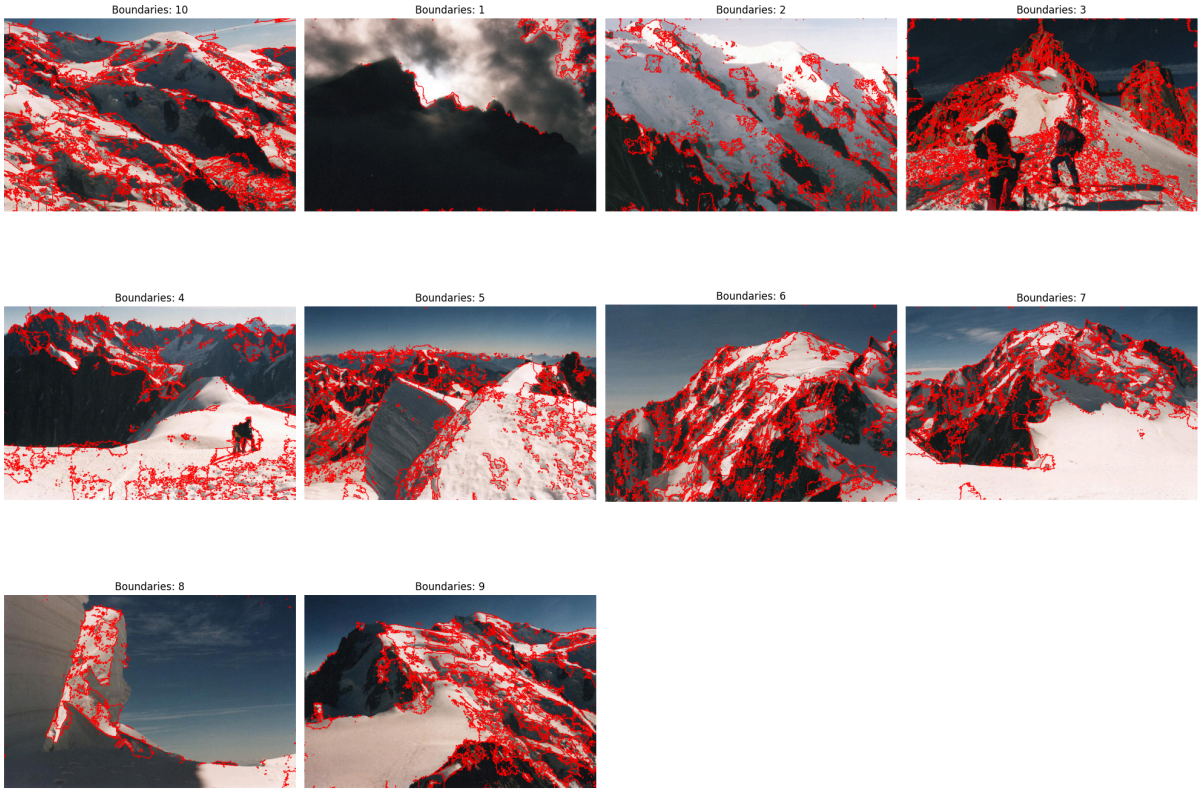

Figure 19: Boundaries of Segmentation in Part 4

2.6 Citations

[3] P. Kovesi, "Gabor Image Features," MATLAB Central File Exchange, 2023. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/38844-gabor-image-features. [Accessed: Nov. 28, 2024].

[4] Scikit-learn Developers, "KMeans clustering," Scikit-learn Documentation, Version 1.5, 2024. [Online]. Available: https://scikit-learn.org/1.5/modules/generated/sklearn.cluster.KMeans.html. [Accessed: Nov. 28, 2024].

[5] Scikit-learn Developers, "StandardScaler preprocessing," Scikit-learn Documentation, Development Version, 2024. [Online]. Available: https://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.StandardScaler.html. [Accessed: Nov. 28, 2024].

[6] Scikit-image Developers, "skimage.color module," Scikit-image Documentation, Version 2024. [Online]. Available: https://scikit-image.org/docs/stable/api/skimage.color.html. [Accessed: Nov. 28, 2024].

[7] Scipy Developers, "scipy.spatial.distance_matrix function," Scipy Documentation, 2024. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance_matrix.html. [Accessed: Nov. 28, 2024].

[8] Scikit-image Developers, "skimage.segmentation module," Scikit-image Documentation, Version 2024. [Online]. Available: https://scikit-image.org/docs/stable/api/skimage.segmentation.html. [Accessed: Nov. 28, 2024].

In addition to the libraries already cited, I utilized several other traditional image processing, computer vision, and data libraries in my implementation. These include OpenCV (cv2), NumPy, Matplotlib (pyplot), and various modules from os, scipy.io, and skimage.

2.7 Discussion of Results

Superpixel Segmentation Parameters

1. k (superpixel number) = 300, because smaller superpixels capture finer details and perform over-segmentation. More than 300 (i.e. 400) led to unnecessary and too complicated over segmentation and less than 300 did not seem overly segmented.

2. m (compactness parameter) = 15. Since the aim in the assignment is doing image segmentation, low m worked better when trying to align superpixels with natural boundaries in the Swiss mountain views dataset such as capturing intricate details like the edges of rocky areas. Since our prior goal is to detect object boundaries in a detailed way, spatial proximity is less important than color adherence and from the paper I know that in the range [1, 40], lower m values provide better adherence to color boundaries, while higher m provides spatially compact superpixels. But, there were also problems that arose as a result of using low m = 10 and 12, since these values could not prevent too much noise in the images. So, I have increased m to 15, which I have found to be optimal for this case.

3. max_iter (number of times the superpixel boundaries are refined)= 10, applied as suggested in the paper.

4. S (size of superpixels) = $\sqrt{N/k}$, where N is the number of pixels.

Gabor Texture Feature Extraction Parameters

1. minWaveLength = 3.

2. mult (scaling factor) = 2.

3. num_scales (number of scales) = 4, as suggested in the assignment.

4. num_orientations (number of orientations) = 4, as suggested in the assignment.

5. sigmaOnf (frequency bandwidth) = 0.65.

6. dThetaOnSigma (angular bandwidth) = 1.5.

I applied the default values for the gaborconvolve.m function, that is the MATLAB implementation, because the assignment suggested, just changed the num_orientations parameter from 6 to 4 since the assignment said so.

Initial Clustering Parameters

1. k (number of clusters) = 4. This value allows for sufficient differentiation between feature groups while maintaining computational efficiency. Higher values (k = 8) result in unnecessary grouping.

Contextual Representation and Subsequent Clusterings Parameters

1. k_initial (number of clusters used for initial clustering) = 8. While fewer clusters (4) simplified the segmentation, it also led to unrepresentative groupings that fail to capture relevant details. Therefore, I increased k to capture more granular features in the images.

   Here, I have also seen that In normal clustering, k = 4 suffices as it only considers local features like texture and color. However, contextual clustering incorporates additional information from neighboring superpixels, increasing feature complexity and variability. This requires a larger k, such as k = 8, to effectively separate regions with distinct contextual patterns. A smaller k results in over-generalized and noisy segmentation, while a higher k allows finer differentiation and better handling of the added variability, improving segmentation quality.

2. k_final (number of clusters used for contextual clustering) = 8. Here, I just align the parameter value with the initial one.

3. ring_threshold (minimum number of pixels required for a superpixel to be considered within a ring) = 15. A low threshold (like 5) allows inclusion of nearby neighbors, enriching contextual information. But, to ensure only significant neighbors contribute, especially in high-resolution images where many superpixels exist, I have raised the threshold to 15.

4. first_ring_multiplier = 3. Values around 2.5-3 could effectively capture relevant neighbors without overextending, balancing proximity and contextual reach for neighboring superpixels. Higher multipliers included too many distant superpixels, introducing noise. Lower (1.5-2) multipliers restricted the contextual information to a relatively small neighborhood. This led to fragmented clustering as small areas dominated the cluster assignments.

5. second_ring_multiplier = 4. Values around 4 could effectively capture relevant neighbors without overextending, balancing proximity and contextual reach for neighboring superpixels. Higher multipliers included too many distant superpixels, introducing noise. Lower (3-3.5) multipliers restricted the contextual information to a relatively small neighborhood. This led to fragmented clustering as small areas dominated the cluster assignments.

References

[1] OpenCV, "Canny Edge Detection," *OpenCV Documentation*, 2017. [Online]. Available: https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html. [Accessed: Nov. 22, 2024].

[2] OpenCV, "Hough Line Transform," *OpenCV Documentation*, 2017. [Online]. Available: https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html. [Accessed: Nov. 22, 2024].