CMPEN/EE455: Digital Image Processing I

Computer Project # 2:

Connected-Component Labeling and Set Operations

Naisong Chen, Andy Luo, William Yoshida

Date: 09/21/2018

---

1. Please compress all your files into one zip file and name it by project number, group number and the last names of all group members (e.g. "project1-group1-Li-Higgins.zip").

A. Objectives

In this project, we:

- study the application of a sequence of simple image-processing operations on an image,
- explore binary-image conversion and connected-component labeling, and
- apply the concepts of logical (set) operations between pixels of different images.

B. Methods

1. Bright Region Extraction

Relevant Files:

- main.m
- binaryimage.m
- top2.m

main.m - for parts 1(a) and 1(b) of the project, main.m will call functions to perform simple image-processing operations to produce a binary image and label connected-components of an input image. To run the code for parts (a) and (b), we read in an image and call the binaryimage() function with parameters f1 and thresh. Parameter f1 is the image to produce a binary image from, and thresh is the chosen pixel intensity threshold. The returned result is the image, fthresh, the resulting binary image. Using the provided function bwlabel() with parameters fthresh and 8, denoting the connectivity type, we acquire flabel, an image with labeled components given 8-connectivity. Then, using the provided function label2rgb() with parameters flabel, we color-code the labels to produce an output RGB image, fRGB.

Flow:

1. Load input image

2. Produce binary image of input image
3. Find and label connected-components
4. Color-code the labels for connected-components of binary image
5. Create copy of the labeled image with only the two largest regions

In steps 2 and 4, we save the output images fthresh and fRGB using the function imwrite(). In step 3, we are actually given two outputs from bwlabel(), flabel as discussed above, but also num, which is the number of components found. The value of num is printed out on the console in step 4. Step 5 takes the labeled image from step 3 and sets all but the two largest connected components to 0, leaving the largest two intact. It does this using the custom top2() function.

binaryimage.m - creates a binary image of an input image. This function takes in parameters "f", the input image, and "thresh", the chosen pixel intensity threshold. The returned result is the image, "fbinary". The threshold value can be changed for a user's preference. Different thresholds will produce different larger/smaller or more/less connected-components, dependent on the image used.
Flow:

1. Cycle through MxN array and fill with background 0 values for pixels below threshold and 1 otherwise

The function denotes pixels with intensities below the threshold as background pixels and assigns them a value of 0. The rest are foreground pixels and are assigned values of 1.

top2.m - creates a new version of the input labeled image where all but the two largest connected components have their pixel values set to 0. This produces an output image that only has the two largest regions distinctly visible, from the background and from each other.
Flow:

1. Cycle through each pixel in labeled, track the count of pixels corresponding to each connected component.
2. Determine the two largest components

3. Cycle through each pixel in labeled, and set output image pixel value to labeled pixel value only if the value is one of the labels for the two largest components.

2. Logical (Set) Operations

Relevant Files:

- main.m
- operator_and.m
- operator_or.m
- operator_xor.m
- operator_not.m
- operator_min.m

main.m - For part 2 of this project, the main script first reads in the 4 input images necessary for the operators: “match1.gif”, “match2.gif”, “cameraman.tif”, “mandril_gray.tif”. Next the AND, OR, and XOR operations are performed on the two binary match images, creating the output binary images. The operators are implemented in correspondingly named functions. Next the NOT operation is performed on the match1 image, which produces a “negative” of the original image, where all pixel values of 255 become 0 and all pixel values of 0 become 255. Finally, the MIN operation is performed on the cameraman and mandrill images, forming a blend of the two images. For part 2(a), the quantities of A AND B will be intersection, A OR B will be set union, A XOR B is the quantities of NOT (A AND B). The quantities of NOT A will be reverse of origin A. So, 0 will be 1 and 1 will be 0.

Flow:

1. Read in the pair of match images, the cameraman, and the mandrill images.
2. Perform the pairwise operators AND, OR, XOR and the pair of match images.
3. Perform NOT on the first match image.
4. Perform MIN on the cameraman and mandrill images.

operator_and.m, operator_or.m, operator_xor.m - The pairwise logical operators are implemented using these functions. Each will take two binary images of the same spatial dimensions and perform the operation on them, pixel by pixel, generating a result image.

Flow:

1. Take the two input images and cycle through each pixel, for each:
    a. perform the logical operation corresponding to the function.
    b. set the pixel value for that coordinate in the output image equal to the result of the operation, where low is 0 and high is 255.
2. Return the result image.

operator_not.m - This function implements the urnary NOT operation for a single binary image. Essentially it will invert each pixel value.

Flow:

1. Take the input image and cycle through each pixel, for each:
    a. perform the not operation so 255 becomes 0 and 0 becomes 255.
    b. set the pixel value for that coordinate in the output image equal to the result, where low is 0 and high is 255.
2. Return the result image

This following table provides an overview for the logic operators:

A	B	AND	OR	XOR	NOT(A)
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

operator_min.m - This function will take two grayscale input images and compare pixel by pixel, setting the output pixel value to be the minimum of the two. This essentially is another way of implementing the set intersection operation.

Flow:

1. Take the input images and cycle through each pixel, for each:

    a. Determine for that given position what the value of the lower grayscale value is between the two images.
    b. Set the pixel value for that position in the output image to that value.
2. Return the result image.

C. **Results**

Here is the original 512x512 grayscale “lake.tif” image that was used for our operations (Figure 1).

Figure 1: Original “lake.tif” image (512x512)

Running main.m script will generate all of the necessary images in the ‘output_images’ directory.

1. **<u>Bright Region Extraction</u>**

Figure 2: Binary image of Figure 1 with threshold of 150 - “lake_fthresh.png” (512x512)

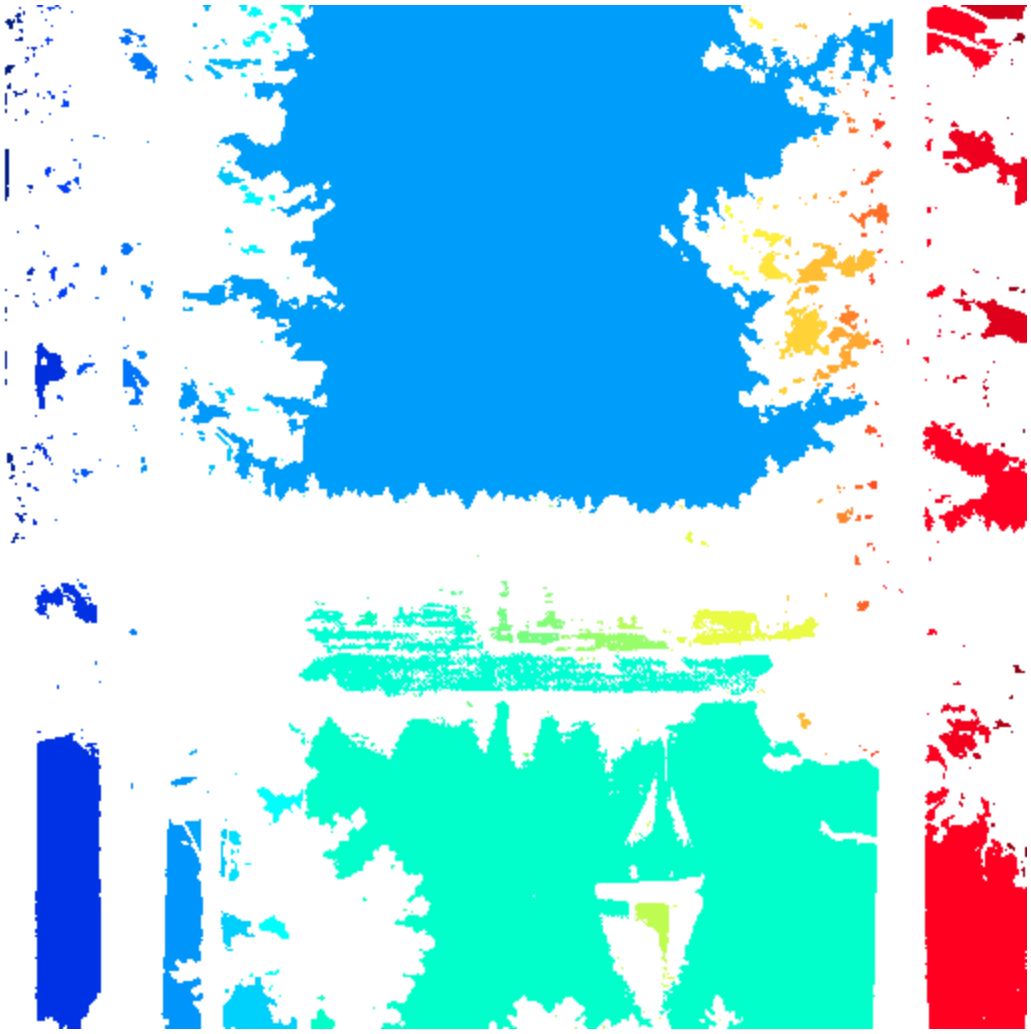

Figure 3: Color-labeled components of Figure 2 - "lake_bright_components_colored_low_threshold.png" (512x512)

Figure 4: Top 2 labeled regions preserved, rest set to 0 - “top2.png”

In Figure 2, we have the result of producing a binary image of Figure 1. As expected, dark regions of the original image, such as dark objects and shadows, were coded as background pixels and assigned a 0 value. On the other hand, light regions, such as the sky, ground, and water, were coded as foreground pixels and assigned a 1 value. The threshold value for this output was chosen experimentally by running the binaryimage() function and analyzing what components were produced. A lower-value threshold allows some more semi-dark regions to be considered “interesting”, increasing the aggregate amount of foreground region pixels. On the other hand, a higher-value threshold cuts off less bright regions, resulting in a smaller foreground.

In Figure 3, we see the result of applying the provided functions bwlabel() and label2rgb() to the binary image, Figure 2. The image shows individual components color-coded with distinct RGB values on a now white background. As expected, there is a varied difference in component size. Some are large and occupy ¼ or ⅕ of the image, since they were large bright regions in the original image, while several others are significantly smaller. They mostly appear due to random bright spots within large background regions, in this image's case, clearings amongst the foliage of the trees. Also, it is worth noting that we do know the total number of components as it is one of the outputs from bwlabel(), the function that finds and labels connected-components. With our chosen threshold of 150, we have a total of 504 individual components. This large amount was not expected, but given that there are multiple small components, as discussed above, this does make sense. As an observation, this amount value was also used to experimentally decide on a threshold, as we were able to check if our results yielded sufficient components.

In Figure 4, we see that it is fairly straightforward to extract the two largest connected components from a labeled image. The two largest regions for the lake image is the sky and the water. This makes sense because the those regions, though they may contain some variations in grayscale intensity, would all have the same value after passing through the thresholding operation. For example, the bright sky region pixels have values that range around 200 and if the threshold is 150, then all of the pixels are given the high value, and they become one large connected component. It's interesting to note that for the lake component, the boat outline is somewhat discernable, meaning that the boat region has pixels that have high enough contrast with the lake pixels to have been placed on the other side of the threshold. All in all, this series of procedures not only allowed for the extraction of components, large or small, but also for the largest components to be highlighted.

2. Logical (Set) Operators

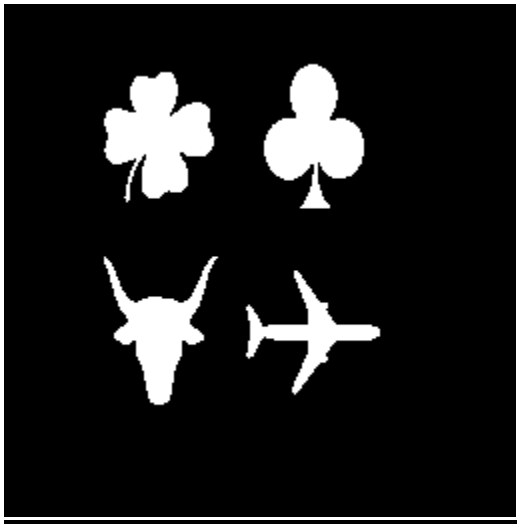

Figure 5: The original match1 image - “match1.gif”

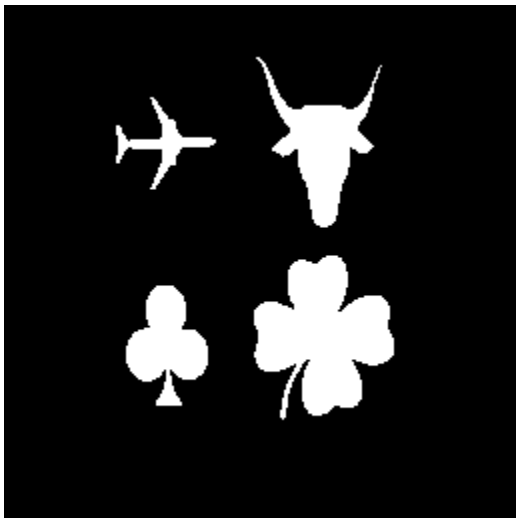

Figure 6: The original match2 image - “match2.gif”

These two binary images were very well suited for pairwise logical operators because they have easily identifiable shapes in 4 locations that are the same for both images. It is easy to intuitively verify whether or not the operators performed correctly.

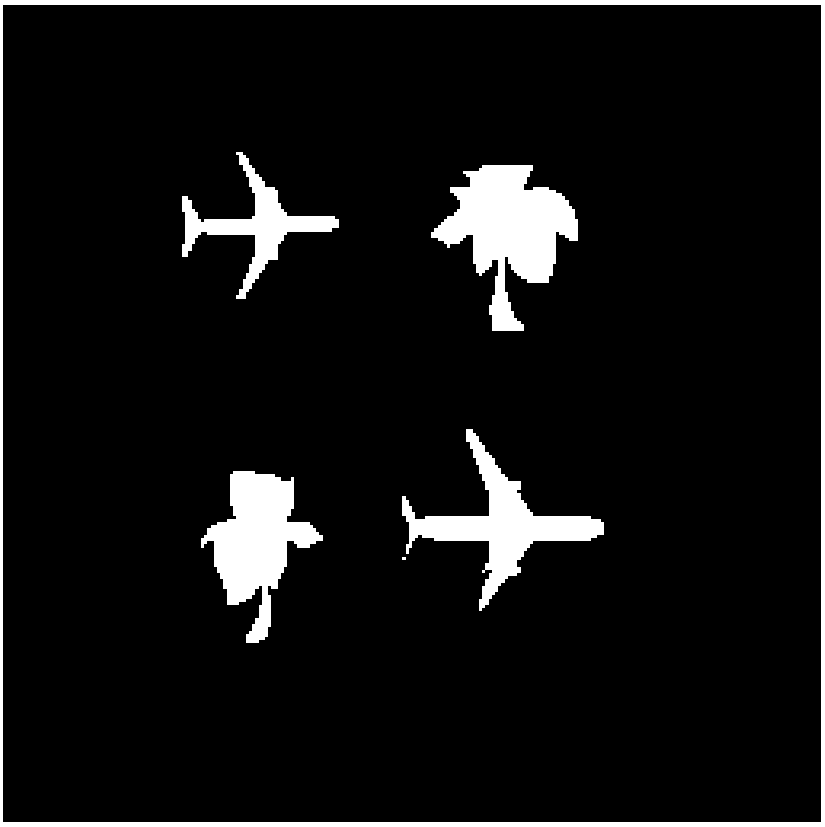

Figure 7: Applying the AND operator to the two match images - “f_and.png”

Applying the AND operator means that for any given position, the output will be high only if the two inputs are also high. The top right icon is the best indicator of this, as only the intersecting parts of the clover and the longhorn are high in the resulting image. The AND of a and b represents the intersection of a and b.

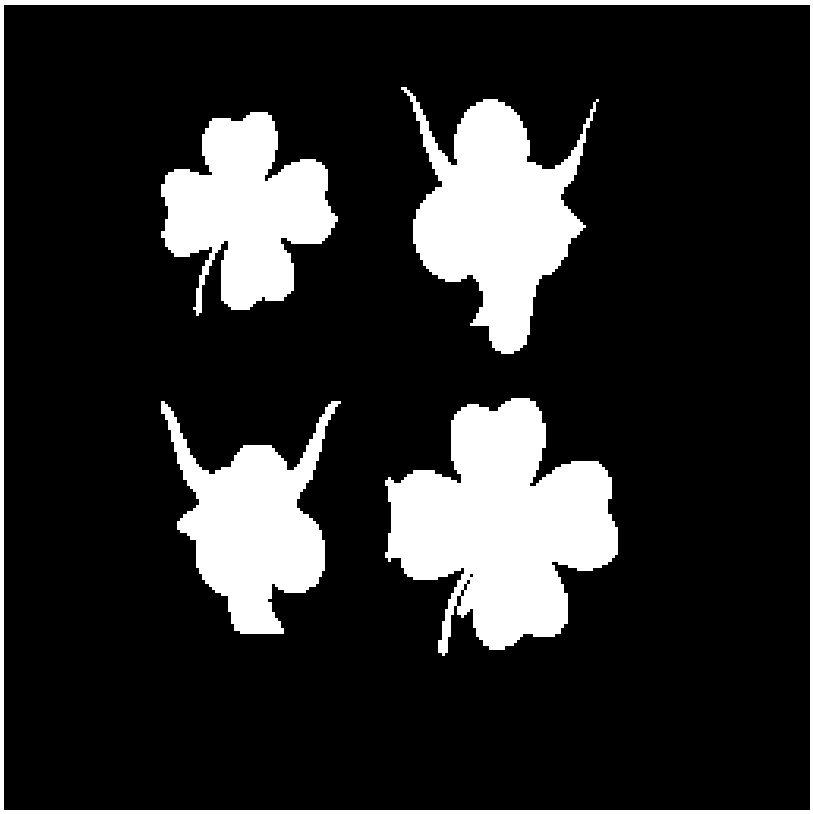

Figure 8: Applying the OR operator to the two match images - “f_or.png”

Applying the logical OR operator means that for any given position in the output image, the value is high if one or both of the inputs at that position is high. Once again, this is best verified to be correct by the top right location, where the resulting icon is the combination of the clover and the longhorn. Unfortunately, some of the other locations are less apparent because the icon in one image completely covers the icon in the other image, but still some protruding non-overlapping elements are seen. The OR of a and b implements the union of a and b.

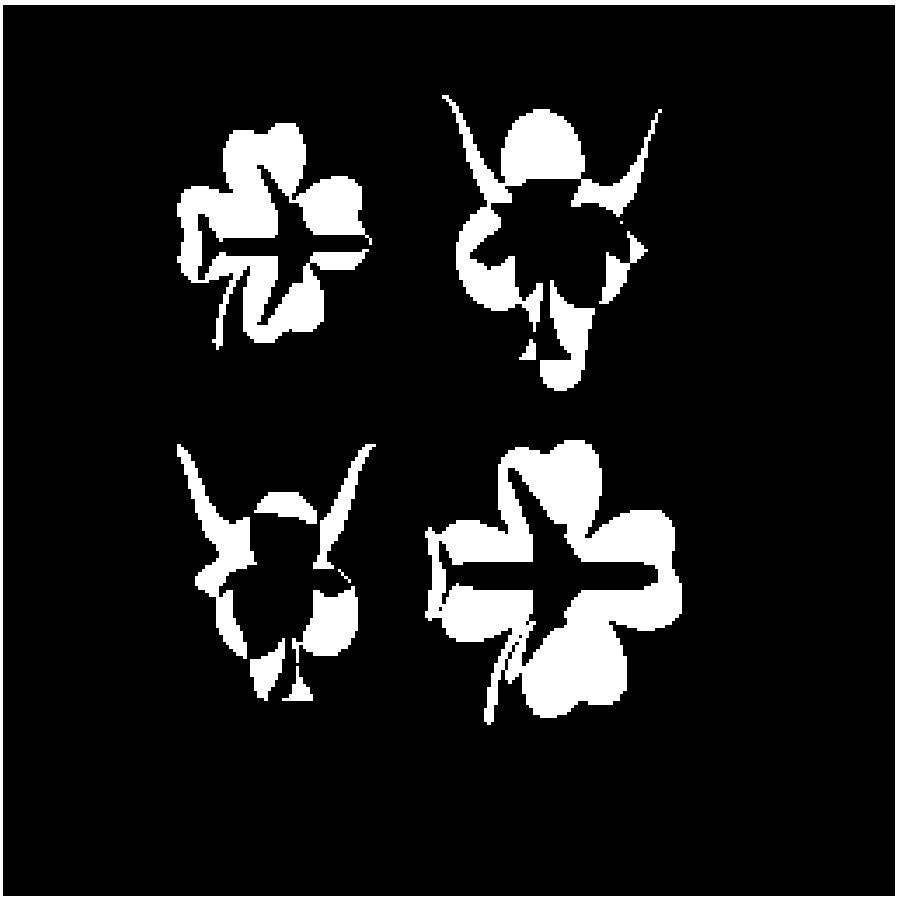

Figure 9: Applying the XOR operator to the two match images - “f_xor.png”

Applying the logical XOR operator means that for any given position in the output image, the value is high only if the value is high in exactly one of the input images. This can be verified by any of the four icon regions. Interestingly enough, there should be a dark spot in the center of each of the icons, exactly in the same shape as seen in the output image of the AND operator. This makes sense because when both inputs are high for a given location, that location in the output image should be low. This is correctly verified. The operation of a XOR b implements (a union b) intersect (complement of a union complement of b).

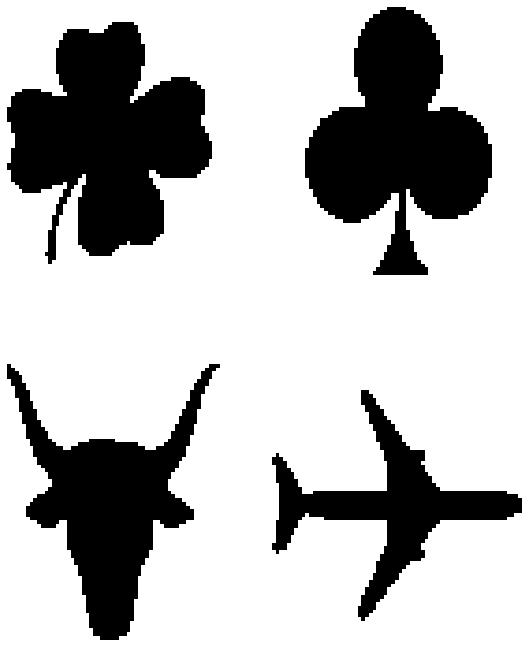

Figure 10: Applying the NOT operator to the image match1 - “f_not.png”

Applying the NOT operator to a binary image means that for any position, the value is the opposite. This essentially inverts the image, which is very apparent in figure 10, where you can see that previously white regions are now black and vice versa. The operation of NOT b implements the complement of b.

Figure 11: The original cameraman image - “cameraman.tif”

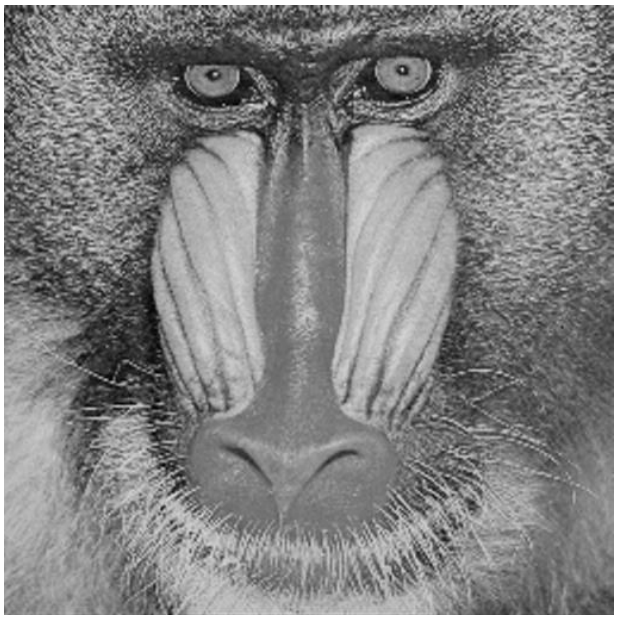

Figure 12: The original mandril image used - “mandril_gray.tif”

These two grayscale images, cameraman and mandril, are great choices for the minimum operator. They differ significantly on scene structure, but both are similar in the sense that most of the foreground or features of interest are dark and the background is mostly light.

Figure 13: Applying the MIN operator to the cameraman and mandrill images - "f_min.png"

Applying the MIN operator to the above pair of images yields the composite image as seen in Figure 13. This composite image demonstrates the fact that the MIN operator acts as a set intersect. Although the image seems to be more of an union of the two inputs than an intersect, as the "foreground" of both images are apparent in the final image, this is because the main foreground/features of interest are dark, which have lower grayscale values. As a result, taking the intersection ends up preserving these dark regions. This follows DeMorgan's Rule: (A*B)' = A' + B'. All in all, the results line up with the expected behavior.

D. **Conclusions**

The process of bright region extraction is very versatile and can be tuned, even in a simple way, to fit a variety of applications. Using simple metrics such as chessboard distance, different regions of an image can be isolated and extracted for analysis. Since regions that appear dark or bright actually contain pixel values that are almost, but not exactly, the same, the thresholding technique is needed mark out these regions. Although this lab used a binary thresholding technique, more grayscale resolution can be used as seen in the previous lab for more granular region labeling. Depending on the composition of the image, there could be small and large regions, with the smallest containing just a handful of pixel and largest containing half or more of the pixels of the entire image. The smaller connected components mostly represent small features or even noise while the largest connected components represent big, and often universally present, features such as the sky. There is value in extracting the biggest features to interpret the structure of the scene. Bright region extraction seems to be the basis of more advanced computer vision techniques.

The logical bitwise logic operators are useful tools for comparing two images, especially ones that are similar in scale and orientation. It's especially useful for highlighting the differences between two images, if that is the desired analysis. Once again, most images in the wild are not binary so some type of thresholding is used to determine the high and low levels for these binary operators. Additionally, the OR operator is a great way to create the union of two images, especially if the features or the image are high valued pixels, and the background is low valued pixels, creating an image that looks like a blend of the two images. Used in conjunction with bright region extraction, these logical operators can serve as a versatile method of basic image analysis.