Collaborators: Aniketh Devarasetty, Janvi Patel, Boting Lu

1 Image Pyramids

1.1 Design Decisions: Gaussian and Laplacian Pyramids

- For Gaussian blurring, I used a 5x5 kernel with $\sigma = 1.0$. I chose $\sigma = 1.0$ because in class, we learned that typically, $\sigma = \sqrt{s/2}$ where s is the downsampling rate. Since we are downsampling by a factor of 2, $\sigma = \sqrt{2/2} = 1.0$. Furthermore, we learned in class that typically, we set the filter size's half-width to be 3σ. $3\sigma = 6$ but the filter should be odd so that there is a center pixel. Thus, I chose a 5x5 kernel.

- When scaling an image dowm, I used the cv2.INTER_LINEAR interpolation method. I chose this method because it is a commonly used method for scaling down images. Also, it is a good balance between speed and quality. It essentially computes a weighted average of the neighboring pixels to determine the new pixel value.

1.2 FFT Decisions

- I first compute the 2-D Fourier transform of the image.

- I also shift the zero frequency component to the center of the spectrum.

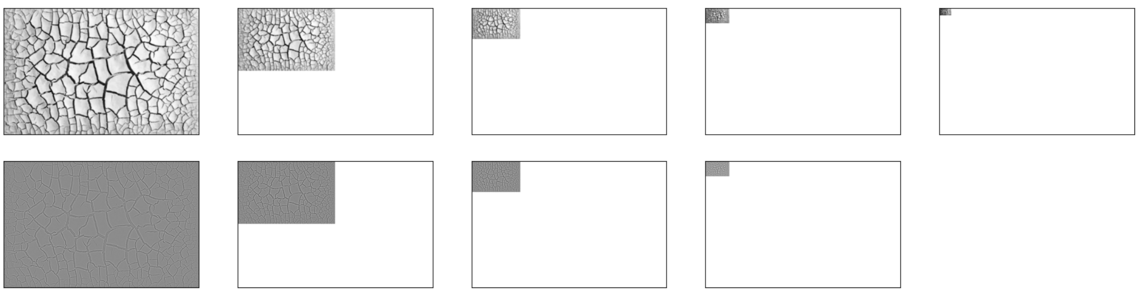
- Finally, I compute the magnitude and then convert it to the logarithmic scale. This enhances visibility of the spectrum.

1.3 Explain what the Laplacian and Gaussian pyramids are doing in terms of frequency.

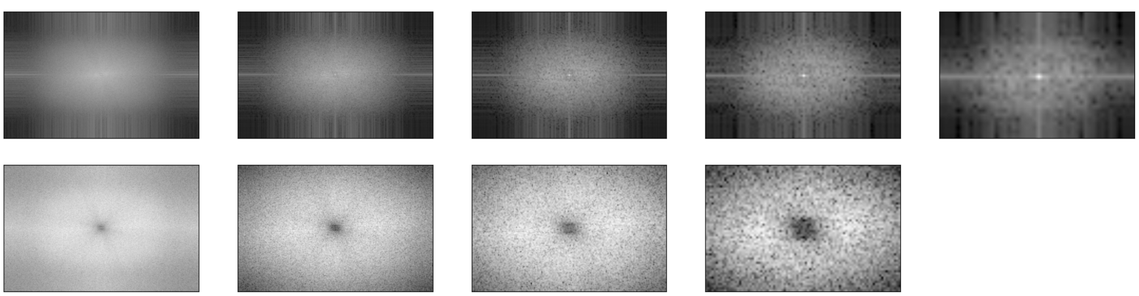
The Gaussian pyramid represents the blurred version of the image (by applying Gaussian filter) at different scales. Specifically, the Gaussian pyramid captures the low frequency components of the image. The Gaussian filter acts as a low pass filter by filtering out high frequencies. This can be seen in the frequency domain, where the Gaussian images have a whiter center (corresponding to higher values) and darker edges (corresponding to lower values). This shows that the Gaussian pyramid images have stronger low frequencies and weaker high frequencies. It is also noticable that there are faint white streaks (horizontal and vertical) that go through the center in the frequency domain. This makes sense because the original image has many horizontal and vertical edges.

On the other hand, the Laplacian pyramid represents the high frequency components of the image at different scales. These high frequencies capture the details of the image, such as edges and textures. Specifically, the Laplacian pyramid captures the residuals of the image by subtracting the blurred image from the pre-blurred image. The residuals capture the high frequencies of the original image. This can be seen in the frequency domain, where the Laplacian images have a darker center (corresponding to lower values) and whiter edges (corresponding to higher values). This shows that the Laplacian pyramid images (residuals) have stronger high frequencies and weaker low frequencies.

In essence, the Gaussian pyramid captures the low frequency components and smooths out fine details, while the Laplacian pyramid captures the high frequency components that are lost during the blurring process.



(a) Gaussian and Laplacian Pyramids



(b) Frequency Domain of Gaussian and Laplacian Pyramids

Figure 1: Pyramids and Frequency Domain

2 Edge Detection

2.1 Design Decisions: Simple Gradient-Based Edge Detector

- I first apply a Gaussian filter to the image, using $\sigma = 1.0$ and a 5x5 kernel. I chose $\sigma = 1.0$ and a 5x5 kernel for the same reasons as in the previous section.
- Then I split the image into its R, G, and B channels
- Then, I applied the X, and Y sobel filters to each channel
- Then, I computed the magnitude and theta of the gradients for each channel
- Then, I combined the magnitudes of each channel using L2 norm to get the final magnitude.
- Finally, I computed the theta of the final magnitude by using the theta of the channel with the highest magnitude. I did this using the *np.argmax* and *np.take_along_axis* functions.

2.2 Design Decisions: Oriented Filters

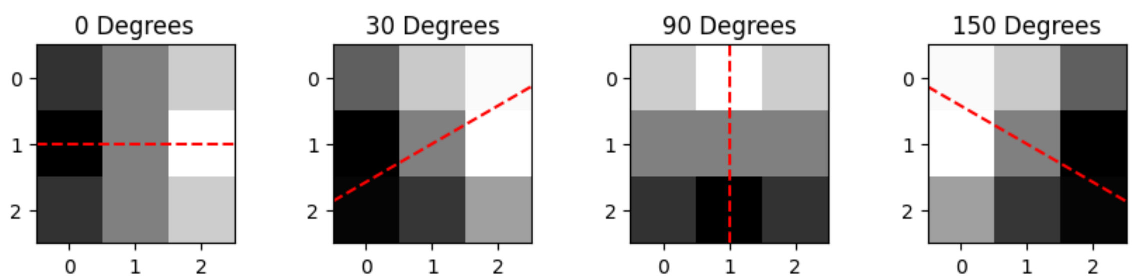
- Two of the filter orientations I chose were the 0 and 90 degrees. This corresponds to the traditional Sobel filter directions. I chose these because any angle can be easily represented as a combination of these two angles.
- The other two filter orientations I chose were 30 and 150 degrees. Although a combination of 0 and 90 degrees can capture any degree, adding 30 and 150 degrees can help increase the edge detection sensitivity.
- I first apply a Gaussian filter to the image, using $\sigma = 1.0$ and a 5x5 kernel. I chose $\sigma = 1.0$ and a 5x5 kernel for the same reasons as in the previous section.
- Then, I split the image into its R, G, and B channels
- Then, I applied the 0, 30, 90, and 150 degree oriented filters to each channel
- Then, I computed the magnitude for each oriented filters by computing the L2 norm of oriented filter applied to the R, G, and B channels. This results in 4 magnitude matrices, one for each oriented filter.

- Finally, I computed the final magnitude by taking the maximum magnitude of the 4 magnitude matrices. I did this using the *np.max* function. The final theta matrix was computed by using the orientation associated with the maximum magnitude. I did this using the *np.argmax* and *np.take_along_axis* functions.

- Note: the *orientedFilterMagnitude* takes an image and a *filter_type_dist* (which is a boolean) as an input. This value is used to determined how the oriented filter is created. I have implemented 2 methods of creating the oriented filter. Essentially, *getOrientedFilterUsingDistribution* samples a 2-D distribution at certain points, whereas *getOrientedFilterUsingCombination* uses a combination of the X and Y Sobel filters to create the oriented filter. By default, *getOrientedFilterUsingDistribution* is used.

- Note: the degree corresponds to the direction of the gradient.

2.3 Bank of Filters used for Problem 2.2

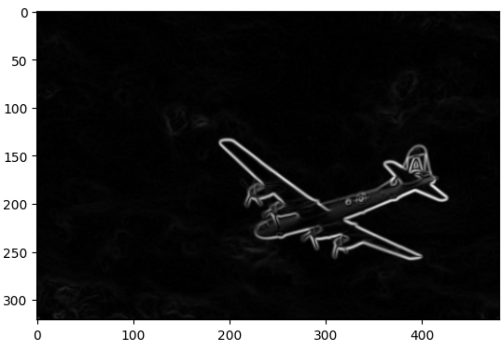


2.4 Comparison of Detected Edges in Problem 2.1 and 2.2

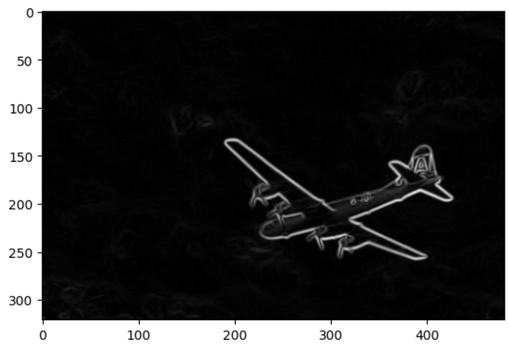
The edge detection without the non-maxima suppression look very similar for Problems 2.1 and 2.2. This makes sense since X and Y directions are sufficient enough to capture the gradient magnitudes, since any angle can be represented as a combination of X and Y directions.

However the edge detection with non-maxima suppression looks different. This is because there are more filters in 2.2 than 2.1. The non-maxima suppressed image for 2.2 has a thin and connected (closed) edge, whereas the non-maxima suppressed image for 2.1 has a thin, but more dotted edge. As a result, many of the edges in 2.2 are closed, in contrast to 2.1, which is sign of better edge detection. One possible explanation is that the increase in oriented filters for 2.2 (by including 30 and 150 degrees) helps the edge detector be more sensitive to edges at different orientations.

In summary, the edge detection with non-maxima suppression for 2.2 shows a thin and connected edge. In contrast, the edges for 2.1 have sections that are dotted and not connected. On the other hand, the edge detection without non-maxima suppression looks very similar for 2.1 and 2.2.

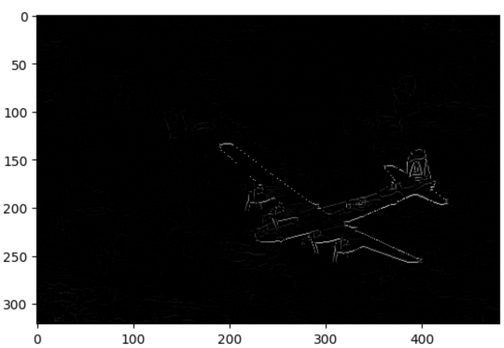


(a) 2.1 without non-maxima suppression

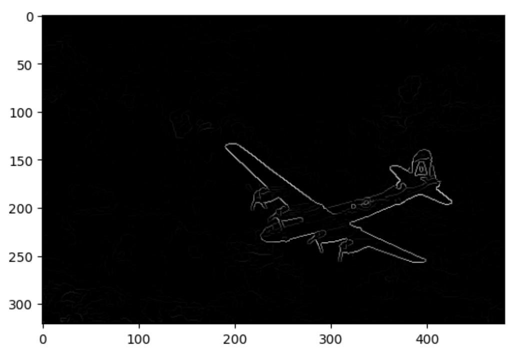


(b) 2.2 without non-maxima suppression

Figure 2: Edge detection without non-maxima suppression for 2.1 and 2.2



(a) 2.1 with non-maxima suppression



(b) 2.2 with non-maxima suppression

Figure 3: Edge detection with non-maxima suppression for 2.1 and 2.2

2.5 Ideas for Improvement

1. In class, we learned that 'Hysteresis' Thresholding is a part of Canny edge detector. It is useful in connecting disconnected edges, resulting in closed edges. Implementing this may improve the edge detection.

Algorithm:

- Implement the same steps as Sections 2.1 and/or 2.2.
- Apply Hysteresis Thresholding to the non-maxima suppressed image, to connect disconnected edges.
- The result is the edge detection.

2. For part 2.2, I chose 30 and 150 degrees as the angles for the oriented filters, in addition to 0 and 90 degrees. This selection may be improved by using PCA to find the principal directions of the edges in the image. Those princpal directions can be used to generate oriented filters. This may allow for a more accurate selection of angles for the oriented filters, and may result in a more accurate edge detection.

Algorithm:

- First, apply the X and Y Sobel filters to the image, to get the gradient direction and magnitude for all pixels.
- These directions and magnitudes can be used to plot the each pixel as a point in a 2-D space, where the x and y coordinates are the gradient magnitudes in the X and Y directions, respectively.
- Then, use PCA to find the principal directions of the points in the 2-D space.
- Use the principal directions to generate oriented filters.
- Finally, apply the oriented filters to the image, and compute the magnitude and theta of the gradients for each oriented filter.
- Then, compute the final magnitude and theta by performing vector calculus on the magnitudes and thetas of the oriented filters.
- Finally, apply non-maxima suppression to the final magnitude and theta.
- Apply Hysteresis Thresholding to the non-maxima suppressed image, to connect disconnected edges.
- The result is the edge detection.

3. Furthemore, using more than 4 oriented filters may improve the edge detection. Using more oriented filters may allow the edge detector to be more sensitive to varying edge orientations, potentially capturing better edges.

- First, suppose n represents the number of oriented filters.

- Then, create n oriented filters at different angles. These angles can be determined by dividing the 180 degrees into $n-1$ equal parts. For example, if $n = 6$, then the angles would be 0, 30, 60, 90, 120, and 150 degrees.
- Then, apply the n oriented filters to the image.
- Then, compute the final magniture and theta by taking the maximum magnitude and its corresponding theta from the n oriented filters.
- Then, apply non-maxima suppression to the final magnitude and theta.
- Apply Hysteresis Thresholding to the non-maxima suppressed image, to connect disconnected edges.
- The result is the edge detection.

4. In Canny Edge Detector we apply a Gaussian filter and then apply Sobel filters. This is effectively "Derivative of Gaussian". Edge detection could be potentially improved by using a "Laplacian of Gaussian" filter, which is basically a second derivative filter. This can help creating thinner edges. However, it is important to note that the non-maxima suppression in Canny Edge Detector is designed to thin the edges, so the Laplacian of Gaussian filter may not be necessary. This algorithm would look exactly like the Canny Edge Detector, but with a Laplacian of Gaussian filter instead of a Derivative of Gaussian filter.