1 Introduction

In this lab, we look at various aspects of neighborhood processing for low-level image understanding. These neighborhood operations provide important cues like corners, edges, blobs which help in image denoising, higher-level understanding like identifying shapes etc. In Section 2, we look at the basic operations of convolution and correlation. Next, in Section 3, we look at low-level filters like Gaussian and Gabor. Finally, in Section 4, we look at several applications such as image noising, denoising, edge detection and foreground segmentation.

2 Neighborhood Processing

2.1 Correlation and Convolution

1. The key difference between correlation and convolution is the way they index over the subset of image ($\mathbf{I}$) corresponding to the kernel ($\mathbf{h}$) which affects how the kernel weights are being multiplied with the input image pixels. To understand how convolution and correlation treat a kernel $\mathbf{h}$, let's consider a sample image with an impulse response s.t. $\mathbf{I} \in \mathbb{R}^{5\times5}$ has 1 at its center and all other pixels are 0. And let $\mathbf{h}$ be defined as follows:

$$\mathbf{I} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{h} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & k \end{bmatrix}$$

   Then, by definition, we would obtain,

$$\mathbf{h} \otimes \mathbf{I} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & k \end{bmatrix}, \mathbf{h} * \mathbf{I} = \begin{bmatrix} k & h & g \\ f & e & d \\ c & b & a \end{bmatrix}$$

   Thus, correlation tends to retain the kernel for such an image while convolution reverses (rotate 180^o about x and y axis) the kernel. It is much more nuanced to understand how correlation and convolution treat an image because it depends on the kernel. For e.g., $\mathbf{h}$ with 1 at the center, rest all zeros will act like an identity kernel over the image for both correlation and convolution. A box-kernel will act like an averaging (smoothing) filter over the image and so on.

2. In order to understand the case when correlation and convolution apply the same effect, we first try and understand how they are related to each other, i.e., if we consider correlation on a local image patch as a matrix multiplication (say, by $\mathbf{h}_{\otimes}$), we want to

know how it is related to the corresponding convolution operation (say, by $\mathbf{h}_*$). Let us consider a toy example where $\mathbf{h}_\otimes \in \mathbb{R}^{3\times3}$,

$$\mathbf{h}_\otimes = \begin{bmatrix} a & b & c \\ d & e & f \\ g & l & m \end{bmatrix}$$

Now, based on the definition, the corresponding convolutional matrix would be,

$$\mathbf{h}_* = \begin{bmatrix} m & l & g \\ f & e & d \\ c & b & a \end{bmatrix}$$

On careful observation, one can notice that $\mathbf{h}_*$ can be obtained by a sequence of matrix operations (row/column) on $\mathbf{h}_\otimes$.

$$\mathbf{h}_\otimes = \begin{bmatrix} a & b & c \\ d & e & f \\ g & l & m \end{bmatrix} \xrightarrow{\mathcal{R}_{1,3}} \begin{bmatrix} g & l & m \\ d & e & f \\ a & b & c \end{bmatrix} \xrightarrow{\mathcal{C}_{1,3}} \begin{bmatrix} m & l & g \\ f & e & d \\ c & b & a \end{bmatrix} = \mathbf{h}_*$$

Here, $\mathcal{R}_{1,3}$ denotes exchanging rows 1 and 3 while $\mathcal{C}_{1,3}$ denotes exchanging columns 1 and 3. Now, interestingly, these can be obtained using a simple matrix multiplication by so-called permutation matrices. For row transform, you need to pre-multiply a permutation matrix and for column, post-multiply. It is trivial to check that:

$$\mathcal{R}_{1,3} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \mathcal{C}_{1,3} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} =: T$$

Another fascinating aspect it that $T^2 = I$ because if you exchange rows 1 and 3 and re-do the same operation, you end up with the same matrix. This is also easy to verify analytically. So when would correlation and convolution be the same:

$$\mathbf{h}_\otimes = \mathbf{h}_*$$
$$\mathbf{h}_\otimes = \mathcal{R}_{1,3}\mathbf{h}_\otimes\mathcal{C}_{1,3}$$

Thus, for 3×3 kernel, the following kernel would have correlation and convolution equal:

$$\mathbf{h} = \begin{bmatrix} a & b & c \\ d & e & d \\ c & b & a \end{bmatrix}$$

Now, this toy case generalizes beautifully to $D = (2K + 1)$-dimensional case:

$$\mathbf{h}_\otimes = \mathcal{R}_{1,2K+1}\mathcal{R}_{2,2K}..\mathcal{R}_{K,K+2}\mathbf{h}_\otimes\mathcal{C}_{K,K+2}\mathcal{C}_{K-1,K+3}..\mathcal{C}_{1,2K+1}$$

We have not tried proving this result for the general case, but the summary is that a kernel where correlation and convolution coincide has to be invariant to 180^o rotation about both horizontal and vertical axes. The most common example of such masks could be the Gaussian kernel since it is symmetric in every direction.

3 Low-level filters

3.1 Gaussian filters

The 1D Gaussian filter is defined as follows:

$$G_\sigma(x) = \frac{1}{\sigma(2\pi)^{1/2}} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

where σ is the variance of the Gaussian. A 2D Gaussian kernel can then be defined as a product of two 1D Gaussian kernels:

$$G_\sigma(x,y) = G_\sigma(x) \times G_\sigma(y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

- **Answer to Question 2**: Convolving an image with a 2D Gaussian filter produces the same results as achieved by convolving an image with two 1D Gaussian filters in x and y direction. This is due to the separability property of 2D Gaussian filter which can be shown easily:

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) = \left[\frac{1}{\sqrt{2\pi}\sigma} e^{\left(-\frac{x^2}{2\sigma^2}\right)}\right]\left[\frac{1}{\sqrt{2\pi}\sigma} e^{\left(-\frac{y^2}{2\sigma^2}\right)}\right] = G_\sigma(x)G_\sigma(y)$$

  Considering an image of size $N \times N$ and a square kernel filter of size $M \times M$, then the computational complexity of 2D Gaussian filter is given by $\mathcal{O}(N^2M^2)$. This is because, for each pixel, we compute M^2 multiplications. In case of 1D Gaussian filter, for every pixel, we apply a 1D kernel, once horizontally and once vertically, thus, $2M$ number of operations per pixel. Thus, the total computational complexity is given by $\mathcal{O}(2MN^2)$.

- **Answer to Question 3**: It is interesting to design a 2nd-order kernel because they can be used for edge detection. An edge can be defined as a location of rapid intensity or color variation. If we consider image as a height map, then edges would correspond to regions of steep slope, for them the second order derivative would be zero. Thus, zero-crossings of second order kernels would give edges making them interesting. Also, 2nd-order kernels are much more stable than 1st-order kernels making them good candidates.

3.2 Gabor filters

The Gabor filters are a class of filters formulated as:

$$\mathbf{g}_{\text{real}}(x,y;\lambda,\theta,\psi,\sigma,\gamma) = \exp\left(-\frac{x'^2+\gamma^2y'^2}{2\sigma^2}\right)\cos\left(2\pi\frac{x'}{\lambda}+\psi\right) \tag{1}$$

$$\mathbf{g}_{\text{im}}(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \sin\left(2\pi \frac{x'}{\lambda} + \psi\right) \tag{2}$$

where $[x', y']$ are rotated in the $\mathbb{R}^2$ plane: $x' = x\cos\theta + y\sin\theta$, $y' = -x\sin\theta + y\cos\theta$.

- **Answer to Question 4**: The parameters of the Gabor filter are described:
  1. λ denotes the wavelength of the sinusoidal component.
  2. θ denotes the angle between the real-axis (x) and the normal direction to the parallel Gabor stripes, essentially it controls the orientation of the filter
  3. ψ denotes the phase-shift of the sinusoid; if $\psi = 0$, it means the sinusoid will pass through the origin
  4. σ denotes the standard deviation of the Gaussian envelope and controls the width of the Gaussian
  5. γ controls the aspect ratio desired in the kernel
- **Answer to Question 5**:
  1. On varying θ, we see that it controls the angle of the normals to Gabor stripes w.r.t. the x-axis. For $\theta = 0$, the filter is vertical, $\theta = \pi/2$, it is horizontal and so on as shown in Figure 1a. We keep all other parameters constant while varying θ and likewise for other cases.
  2. On varying σ, we note that it controls the overall width of the Gaussian envelope and also as this width increases, the number of stripes within the envelope increases as shown in Figure 1b.
  3. On varying γ, we see that the height of the filters varies. It basically controls the aspect ratio and the higher it is, the lesser is the height of the filter (length along y-axis assuming $\theta = 0$). This is illustrated in Figure 1c.

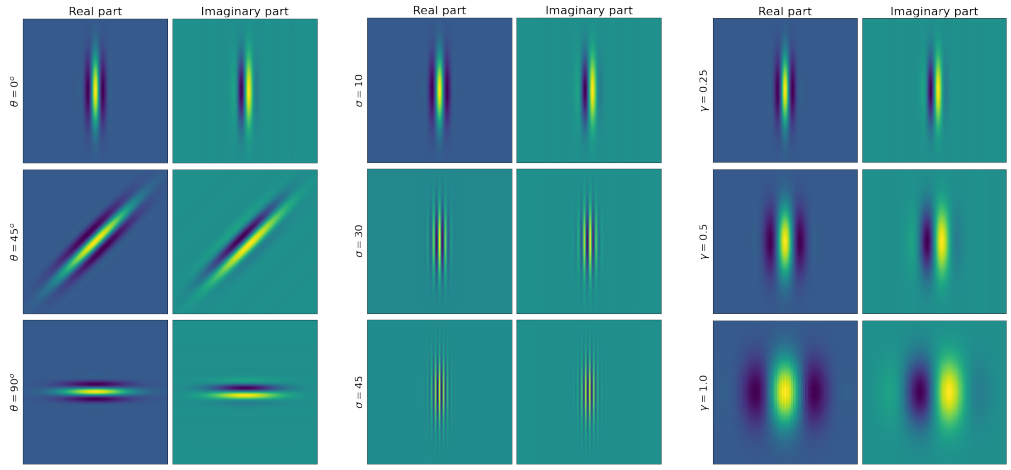
4 Applications in image processing

4.2 Image denoising

4.2.1 Quantitative evaluation

Answers to question 6:

1. The PSNR metric is a measure of how well an image has been approximated, and is calculated by finding the ratio of the maximum intensity of the original image, to the difference between the original image and the approximated image. In general, the higher the PSNR, the better. A higher PSNR implies that the approximation closely resembles the original image.



(a) Varying θ (b) Varying σ (c) Varying γ

Figure 1: Gabor filters (real, imaginary parts) with varying parameters θ, σ, γ individually.

2. The computed PSNR is 16.1079 dB.

3. The computed PSNR is 20.5835 dB. This is higher than the previous case implying that the images noised using Gaussian is closer to the original image than one with Salt and Pepper noise.

4.2.2 Neighborhood processing for image denoising

Answers to question 7:

1. The image denoising results are shown in Figure 2 for box filter and Figure 3 for median filter. We observe that for the Gaussian noisy image is better denoised by box filters while for the Sale and Pepper noisy image by median filter.

2. The PSNR between a denoised image and the original image for each of the 12 cases are reported in 1. Effect of filter size: As filter size increases, for both filter types, the PSNR decreases - smaller filters provide the best denoising since larger filters lead to too broader aggregation.

3. The median filter is better for salt and pepper noise. It can be seen in Figure 1a and Figure 1b that the PSNR scores for the median filter are greater than the PSNR scores for the box filter. Furthermore, the filtered images with salt and pepper noise in Figure 3 look subjectively clearer than the corresponding images in Figure 2. This is because salt and pepper noise is caused by sudden changes in the image signal. Choosing the median value of the surrounding pixels removes these sudden changes by replacing those intensities with less extreme ones.

Both median and box filters yield quite similar PSNR scores for images with Gaussian noise. However, the box filter appears to perform better than the median filter subjectively. This is because, as mentioned above, the median filter reduces sharp changes in intensities. However, gaussian noise does not result in sharp changes in intensity, so the median filter is not ideal in this case.

4. We experimented with varying σ for denoising the Gaussian noise image with Gaussian filter. The results are shown in Figure 4. We empirically found 5×5 kernel to give the best results in terms of PSNR and thus, we retained that.

5. Table 2 shows that for a 5×5 Gaussian filter, the PSNR improved as σ increased, until a certain point (when $\sigma = 1.0$). After this point, the PSNR remained relatively constant. This is because the greater the σ value, the more smoothing takes place. So when sigma is small, there will still be significant noise in the image because the image will not have been smoothed much. However, when sigma is large, a lot of smoothing takes place and information is lost.

6. The median, box, and Gaussian filters have fundamentally different ways of eliminating noise within an image. The box filter works by averaging neighbouring pixel values. If enough points are selected, the sum of the noise values will ideally approach zero. The Gaussian filter takes a similar approach in that it too takes an average of neighbouring pixel values. However, these values are weighted by their proximity. Pixels closer to the center are assigned a higher weight than those that are far away. Weights are assigned to pixel values according to a normal distribution. The median filter on the other hand, eliminates noise by disregarding it. Since we choose median value of neighbouring pixels, a large spike in the intensity of a neighbouring pixel has no affect on the median value. Any two of these methods may give a PSNR in the same ballpark for an image. However, the qualitative results may vary. For this reason, the PSNR metric cannot be used in isolation to determine whether one denoising method is better than another.

Filter size	3×3	5×5	7×7
Gaussian	**26.215**	23.648	21.933
Salt & Pepper	**23.386**	22.630	21.413

(a) Box filter

Filter size	3×3	5×5	7×7
Gaussian	**25.528**	23.941	22.243
Salt & Pepper	**27.856**	24.667	22.545

(b) Median filter

Table 1: PSNR (dB) comparisons between the original image and denoised image for varying kernel sizes for box and median filters.

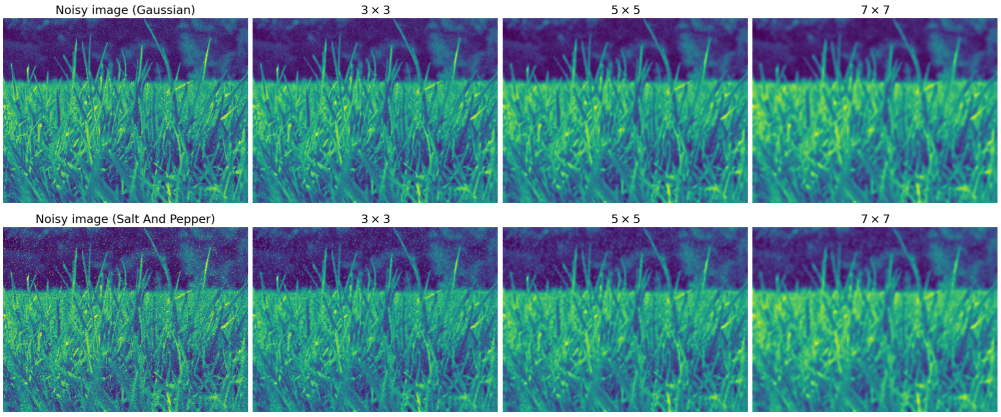


Figure 2: Box filtering for denoising given noisy images with varying filter size.

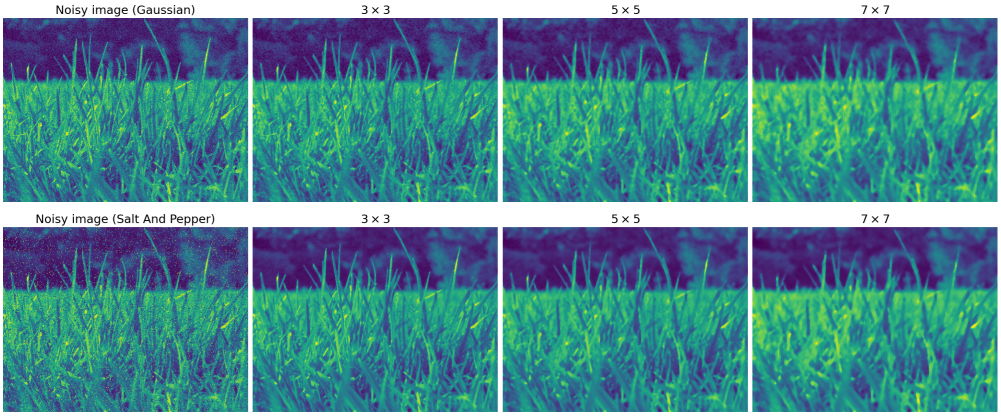


Figure 3: Median filtering for denoising given noisy images with varying filter size.

σ	0.1	0.25	0.5	1.0	2.0	4.0
PSNR	20.583	20.583	24.270	**26.408**	24.485	23.853

Table 2: σ vs PSNR for Gaussian noised image with a 5×5 Gaussian filter

4.3 Edge detection

4.3.1 First-order derivatives

Answer to question 8: The results are shown in Figure 5.

- G_x: The image of partial derivative w.r.t. x provides edges along the x-axis, i.e. points along x axis where there is a sudden change in the intensity.
- G_y: The image of partial derivative w.r.t. y provides edges along the y-axis.

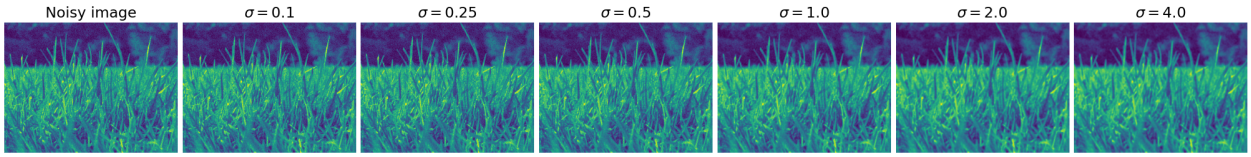


Figure 4: Denoised images for Gaussian noise images with filter 5×5 and varying σ.

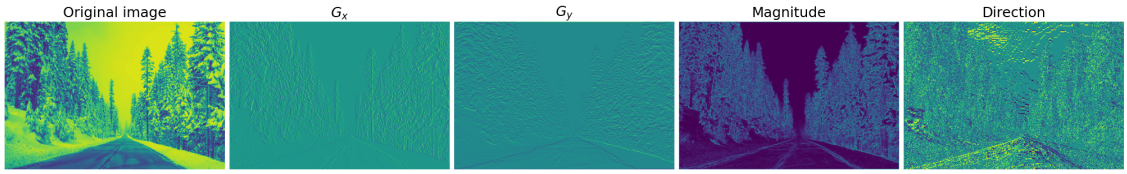


Figure 5: The partial derivatives, magnitude, direction of thegradient field of a sample image.

- Magnitude: This indicates, for every pixel, how strong the gradient is at that pixel. If it is 0, it means the intensity around the pixel is almost constant and higher values mean there is a change in intensity in some directions.
- Direction: This indices the angle of the slope of direction of derivative at a given pixel. If it is 0, it means the derivative is parallel to the x-axis, if it is $\pi/2$, it means the derivative is parallel to y-axis.

4.3.2 Second-order derivatives

Answer to question 9:

1. The results using the three methods are shown in Figure 6.
2. The first method involves two separate convolution steps and is computationally more expensive. The second method applies the two convolutions together as a single kernel because these are linear operators and linear operators are associative in nature. The third method (DoG) is actually an approximation of the second-order derivatives by a difference of a narrow Gaussian (like an impulse filter) and a wider Gaussian.
3. In the 1st method, it is important to smoothen an image before convolving with a Laplacian as images inherently have noise in its pixels and directly convolving with Laplacian will exaggerate that noise leading to a very jittery 1st-order derivative and may show every pixel as an extrema and zero crossings on the 2nd-order derivative.
4. Empirically, we found using $\sigma_1 = 0.4, \sigma = 1.0$ provides decent results as shown in Figure 6. The purpose of two standard deviations is to obtain two different Gaussians, one with a sharp narrow peak and another with a wide peak such that their difference is able to

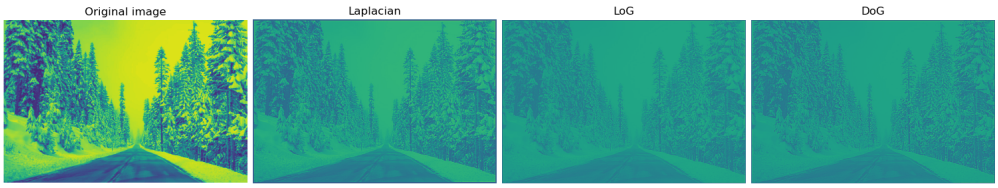


Figure 6: Laplacian of Gaussian (LoG) computed with three different ways.

approximate Laplace of Gaussians. This is useful since it prevents the computationally expensive 2nd order derivative and does the job with two Gaussians.

5. In order to isolate the road properly, on top of edge detection, we will need (a) do non-maximal suppression (to get thin edges), perform thresholding to ignore noisy edges, and (b) to run line detection algorithms (e.g. Hough transform) to actually extract it.

4.4 Foreground-background separation

Answers to question 10:

1. We implement and run the Gabor-filtering based algorithm for all the test images. With the default parameter setting, we report the sample results in Figure 7. We observe that it works fairly well for cases where the foreground object has a sufficiently different texture than the background. For a harder case where the foreground texture is not as distinguishable (e.g. Kobi in Figure 7), it fails.

2. We experiment with various hyperparameter settings for the test images and note the observations:

   - For the Kobi image, we experimented with various values of $\sigma \in \{0.001, 0.1, 1, 10\}$ (with other parameters being fixed) which drives the size of the Gabor kernel. Smaller σ gives a smaller kernel (e.g. $\sigma = 0.001 \rightarrow 3 \times 3$) and that maybe more beneficial in this case because a smaller receptive field may be better able to distinguish the dog texture from the background which is quite similar to it. Also, notice the dog shadow is causing a consistent problem in the segmentation.
   - For Kobi, we note that the other parameters do not have a significant impact on quality of mask obtained when changed from the default values.
   - For all other sample images, we observe best performance with the default parameter setting.

3. Impact of post-smoothing magnitudes: If there are noisy local variations even within region of same texture, those will be exaggerated in the magnitudes of Gabor features and will affect the segmentation. To avoid impact of such local noise, we pass the

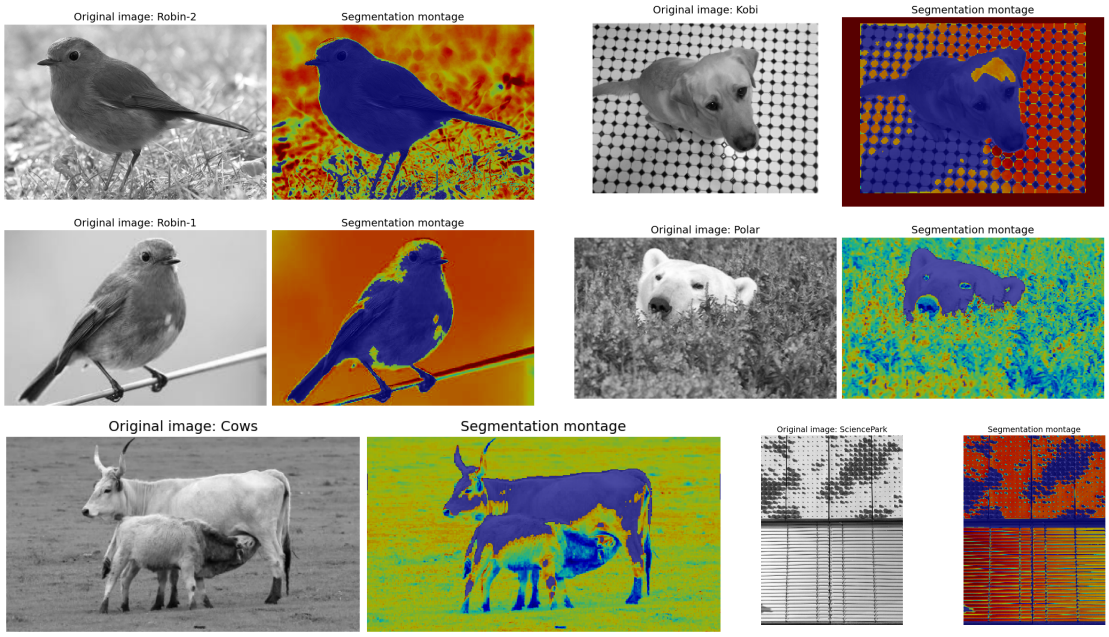
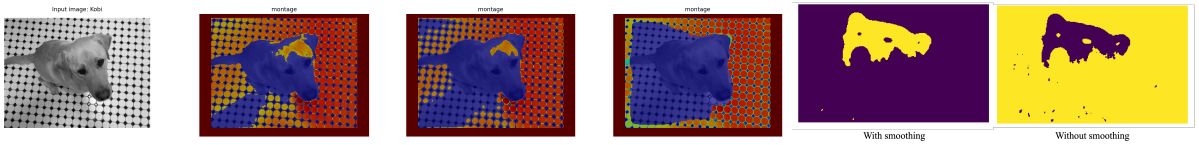


Figure 7: Sample results for Gabor-filtering based forground segmentation with the default parameters. Blue denotes the foreground.



(a) Original (b) $1e^{-3}$ (c) 1 (d) 10 (e) Gaussian smoothing

Figure 8: (L) Results on varying σ for Kobi image. (R) Impact of Gaussian smoothing on Gabor magnitudes

magnitudes by a Gaussian smoothing filter. For example, see Figure 8e, the segmask for Polar bear image. Without the smoothing, the background has spurts of noise.

Conclusion

To summarize, we studied, implemented, analyzed various methods for neighborhood processing from using low-level filters to detect edge/blob/corners to using them for a variety of applications. An important learning for us was how useful these classical methods could be even for a non-trivial task like foreground segmentation. However, it involves fair amount of parameter tuning with these carefully handcrafted features.