EE106A Discussion 5: Vision

1 Convolutions

A **convolution** is the treatment of one matrix M (the original image) by another usually smaller one K (the kernel). The result of a convolution is a filtered image matrix $K * M$. The convolution is performed by sliding the kernel over the original image and taking the matrix dot product of the kernel and the part of original matrix that it covers. Let's look at an example:

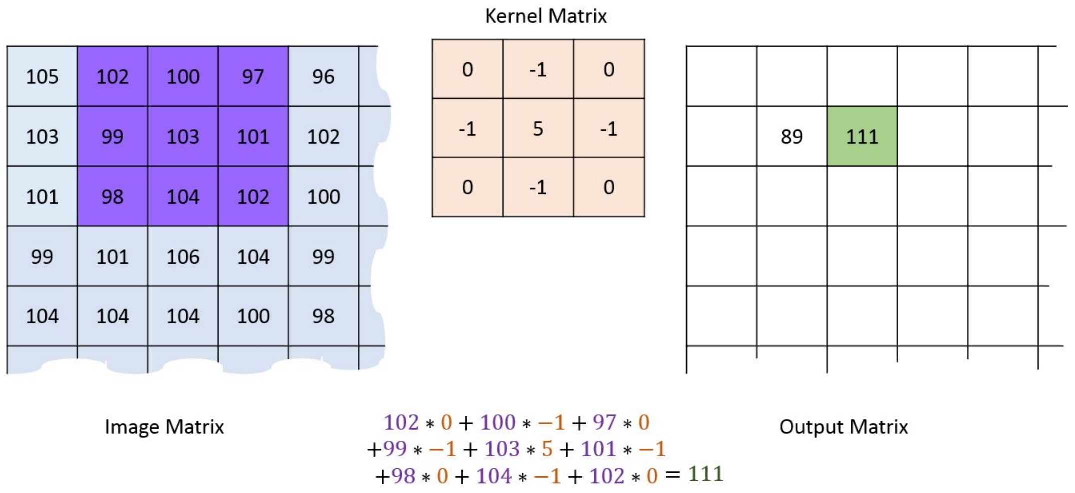
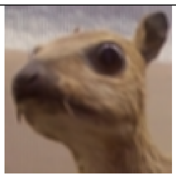
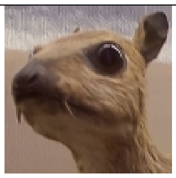
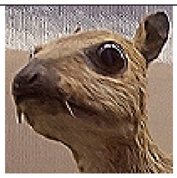


Figure 1: Example of a convolution.

Problem 1. *If the original matrix has dimensions $m \times n$ and the kernel has dimensions $p \times q$, what will be the size of the matrix resulting from the convolution?*
The size of the resulting matrix is exactly how many times the kernel can cover a unique portion of the original image. Thus, the resulting matrix will have size $(m - p + 1) \times (n - q + 1)$.

Problem 2. *Convolution kernels are useful for applying effects to images and extracting features for machine learning. Match the resulting image to the kernel applied as well as the name of its effect.*

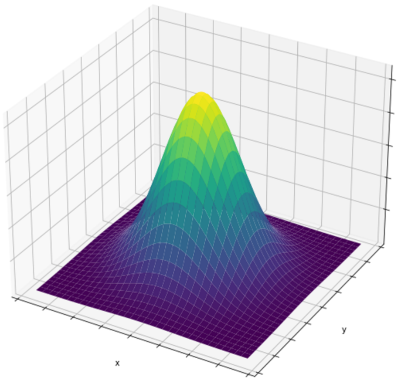
Result			
Kernel	$\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$	$\begin{bmatrix}0&0&0\\0&1&0\\0&0&0\end{bmatrix}$	$\begin{bmatrix}0&-1&0\\-1&5&-1\\0&-1&0\end{bmatrix}$
Name of effect	Box Blur	Identity	Sharpen

Kernels:

$$\left\{\begin{bmatrix}0&-1&0\\-1&5&-1\\0&-1&0\end{bmatrix},\begin{bmatrix}0&0&0\\0&1&0\\0&0&0\end{bmatrix},\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}\right\}$$

Names of effects: {Identity, Box Blur, Sharpen}

We can obtain a **Gaussian kernel** by sampling the values of a 2D Gaussian at discrete (x, y) points.



$$K_g = \begin{bmatrix}4.4e{-5}&1.3e{-3}&4.0e{-3}&1.3e{-3}&4.4e{-5}\\1.3e{-3}&3.8e{-2}&1.2e{-1}&3.8e{-2}&1.3e{-3}\\4.0e{-3}&1.2e{-1}&3.6e{-1}&1.2e{-1}&4.0e{-3}\\1.3e{-3}&3.8e{-2}&1.2e{-1}&3.8e{-2}&1.3e{-3}\\4.4e{-5}&1.3e{-3}&4.0e{-3}&1.3e{-3}&4.4e{-5}\end{bmatrix}$$

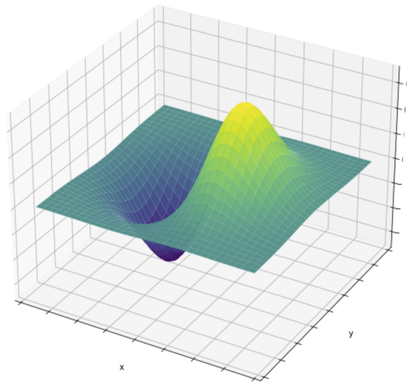
Problem 3. *What effect would a Gaussian kernel have on an image?*
Similarly to the box blur filter, the Gaussian kernel blurs/smooths the image. However, it has a number of nice properties that mean you should always use it for smoothing instead of the box blur. Intuitively, these properties stem from the fact that pixels that are further away from the center pixel have less influence on the final result.

1.1 Derivative-of-Gaussian Kernels

In addition to applying effects to images, convolution kernels can be used to approximate the horizontal and vertical derivatives at each point in an image. The simplest choice of filter for approximating the horizontal derivative has the form:

$$K_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix},$$

But this tends to be very sensitive to noise. To smooth out the noise, we could apply a Gaussian blur kernel as well as the derivative kernel. But since the derivative and convolution operations commute, it's more computationally efficient to combine these into a single operation by making a new filter based on the directional derivative of the Gaussian kernel:



$$K_{gx} = \begin{bmatrix} -7.2\text{e}{-5} & -1.1\text{e}{-3} & 0.0\text{e}0 & 1.1\text{e}{-3} & 7.2\text{e}{-5} \\ -2.1\text{e}{-3} & -3.1\text{e}{-2} & 0.0\text{e}0 & 3.1\text{e}{-2} & 2.1\text{e}{-3} \\ -6.5\text{e}{-3} & -9.5\text{e}{-2} & 0.0\text{e}0 & 9.5\text{e}{-2} & 6.5\text{e}{-3} \\ -2.1\text{e}{-3} & -3.1\text{e}{-2} & 0.0\text{e}0 & 3.1\text{e}{-2} & 2.1\text{e}{-3} \\ -7.2\text{e}{-5} & -1.1\text{e}{-3} & 0.0\text{e}0 & 1.1\text{e}{-3} & 7.2\text{e}{-5} \end{bmatrix},$$

Problem 4. *How could we change K_{gx} to obtain K_{gy} for the vertical derivative?*
We can simply take the transpose: $K_{gy} = K_{gx}^T$

We can combine the horizontal and vertical derivatives into G to approximate the overall gradient of the image:

$$G = \sqrt{K_{gx}^2 + K_{gy}^2}$$

Problem 5. *How is G useful in edge detection?*
Edges are points on an image at which the gradients are high (i.e. large changes in neighboring pixels). Thus, we can use G to read off the gradients at each pixel location, and the ones with higher gradients are points that are more likely to be on edges.

2 Pinhole Camera Model

When we take a 2D picture, we are essentially transforming points in the real 3D world to points on a 2D image. Let's see if we can work out what this transformation is, which we call the *intrinsic camera matrix*. To do so, we model our camera as a standard pinhole model camera, in which light rays from the real world are projected onto an image plane and there is no lens involved.

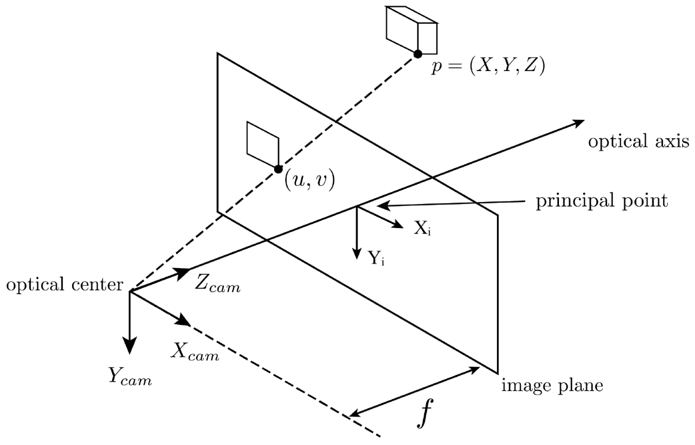


Figure 2: A pinhole camera projects a 3D object into a 2D image.

Let's look at Fig. 2 to derive our matrix. We have two frames of reference here. One is the camera frame with axes $X_{cam}, Y_{cam}, Z_{cam}$ from which 3D locations can be expressed. For example, point p has position (X, Y, Z).

We have another 2D frame of reference in the image plane with axes X_i, Y_i. The point p projected onto the image frame has coordinates (u, v) in this image frame.

Problem 6. *Write expressions for u and v as functions of the 3D position (X, Y, Z) and the focal length f.*
By similar triangles,

$$u = \frac{X}{Z} f$$

$$v = \frac{Y}{Z} f$$

2.1 Scaling

In the real world, there may be scaling effects present that may differ for the horizontal and vertical directions. This is as if f is scaled by a factor s_x when doing projections in the x direction, and scaled by s_y when doing projections in the y direction. Thus, we replace the f in the expressions for u and v by $f_x := s_x \cdot f$ and $f_y := s_y \cdot f$ respectively.

Problem 7. *Update the expressions for u and v using f_x and f_y instead of f.*

$$u = \frac{X}{Z} f_x$$
$$v = \frac{Y}{Z} f_y$$

Note that f_x and f_y are *not* the focal length in the x and y directions - the focal length is a constant! Instead, s_x and s_y capture the idea that pixels may not be square due to physical properties of the camera.

2.2 Translation of origin

In computer vision, the origin of a 2D image may not actually lie on the principal point. Thus, let's allow the image frame with axes X_i, Y_i be free to move around, such that the principal point is now at an arbitrary coordinate (x_o, y_o).

Problem 8. *Update the expressions for u and v to take into account this arbitrary origin shift.*

$$u = \frac{X}{Z} f_x + x_o$$
$$v = \frac{Y}{Z} f_y + y_o$$

2.3 Homogeneous coordinates

We define a clever way to express 2D coordinates in the image plane with three dimensions. We do this by appending a w to the end of the vector and dividing u and v by this w. That is, $\begin{bmatrix} u \\ v \end{bmatrix}$ turns into $\begin{bmatrix} u' \\ v' \\ w \end{bmatrix}$ where $\begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{w} \begin{bmatrix} u' \\ v' \end{bmatrix}$.

Problem 9. *In homogeneous coordinates where w is set to be Z, find the intrinsic camera matrix that maps a real 3D point to a point in the image plane.*

$$\begin{bmatrix} u' \\ v' \\ Z \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & x_o \\ 0 & f_y & y_o \\ 0 & 0 & 1 \end{bmatrix}}_{\text{intrinsic camera matrix}} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Problem 10. *What happens if the 3D point is scaled by a factor of lambda?*
When we multiply the point by the intrinsic camera matrix, we get $[\lambda u', \lambda v', \lambda Z]^T$ for our image plane point. However, since we divide the first two elements by the third to recover the image plane coordinates, this doesn't affect the location of the point on the image plane! Intuitively, this is because larger objects further away are indistinguishable from smaller ones closer up.