CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

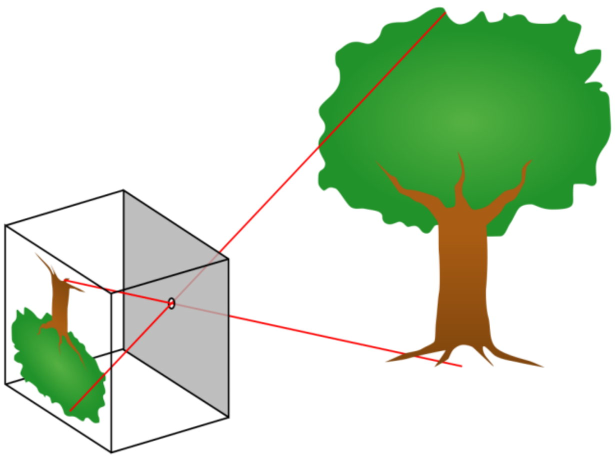
Computing@CIT

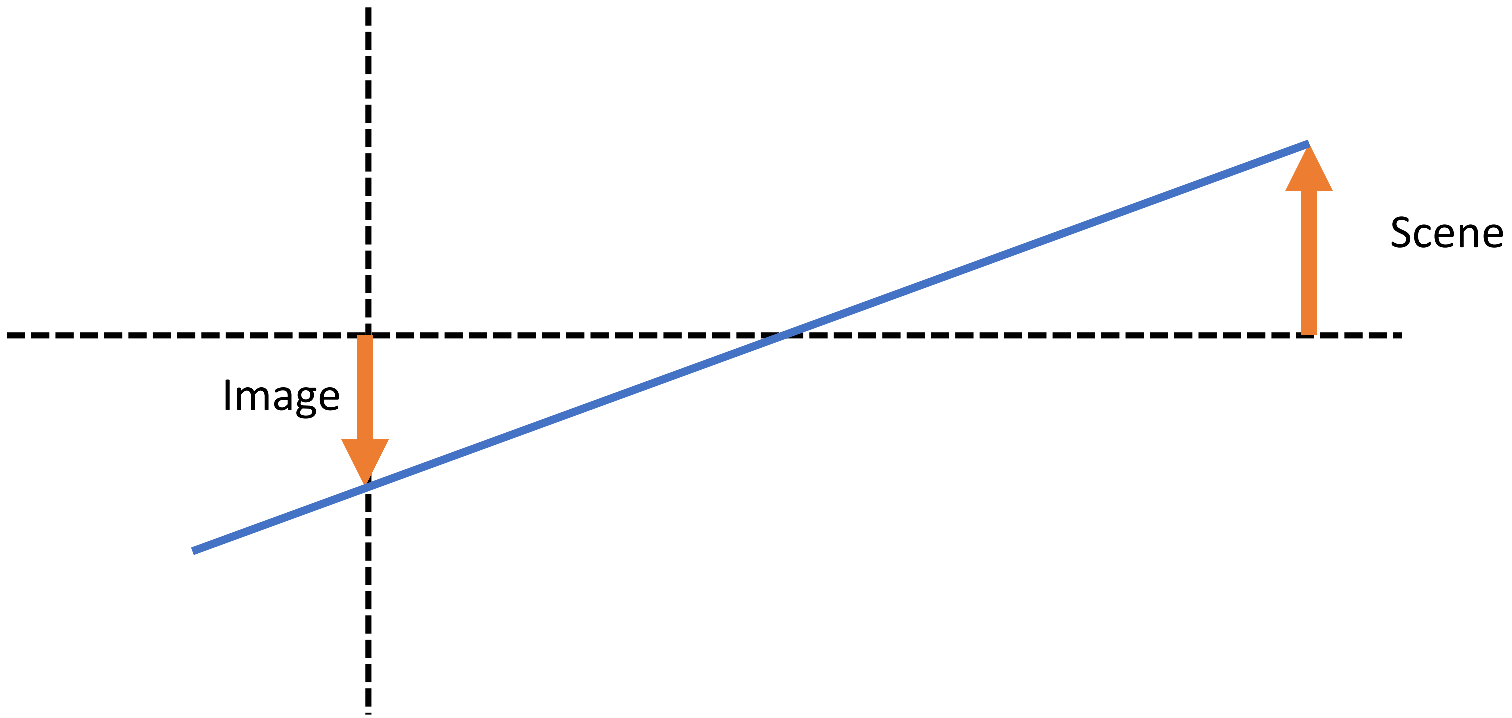
Machine Vision

Lecture 2: Linear Filters

Pinhole camera model

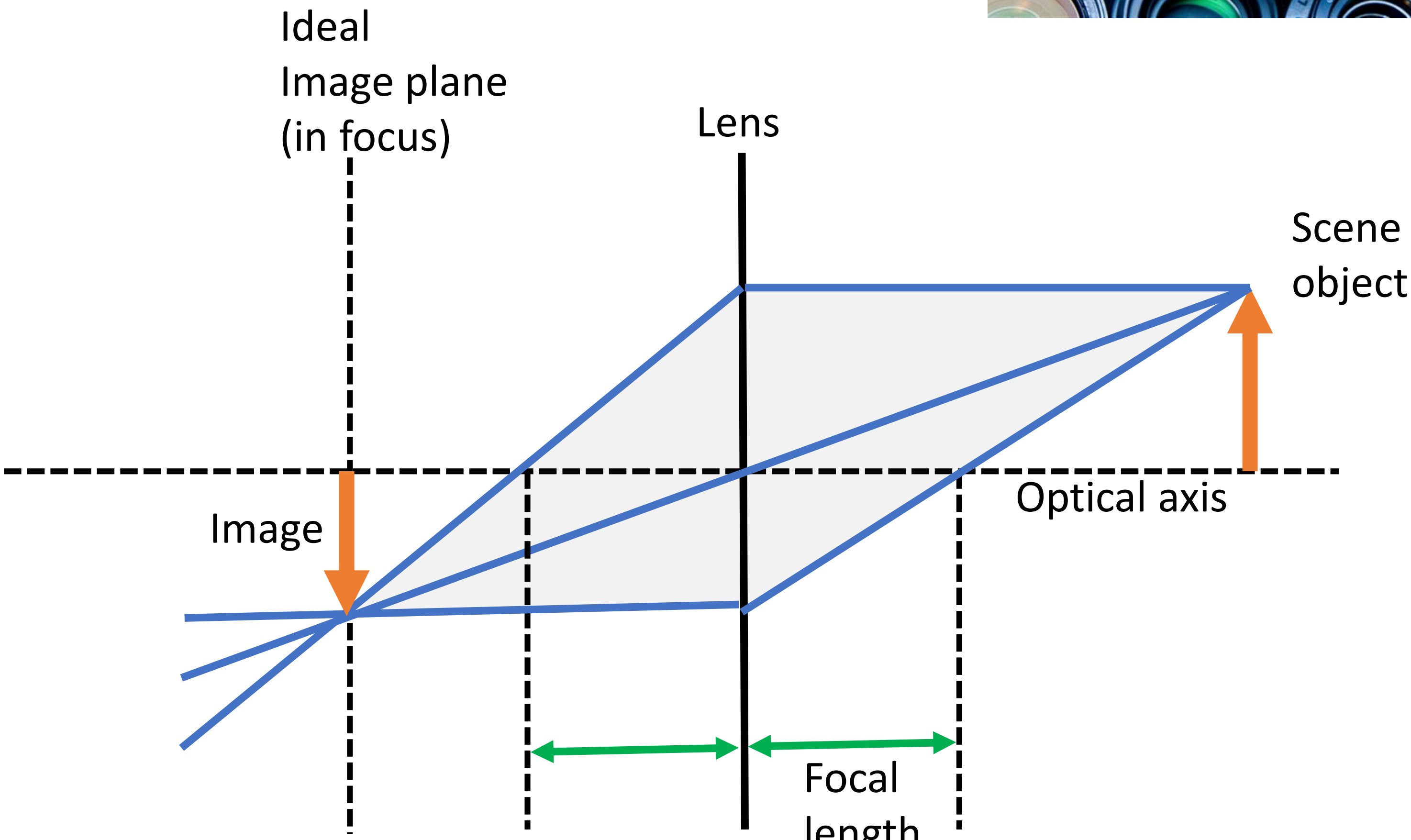
- Light travels on a straight line from the object through then centre of projection
- The image is then created at the intersection with the image plane





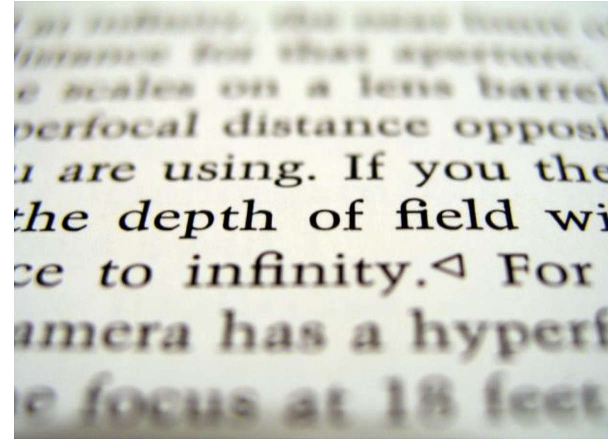
Simple lens optics

- In reality lenses have to be used to gather light
- A sharp image is only produced for objects at a specific distance



Simple lens optics

- Unless perfectly focused a blurred image of every scene point is projected onto a circle on the image plane



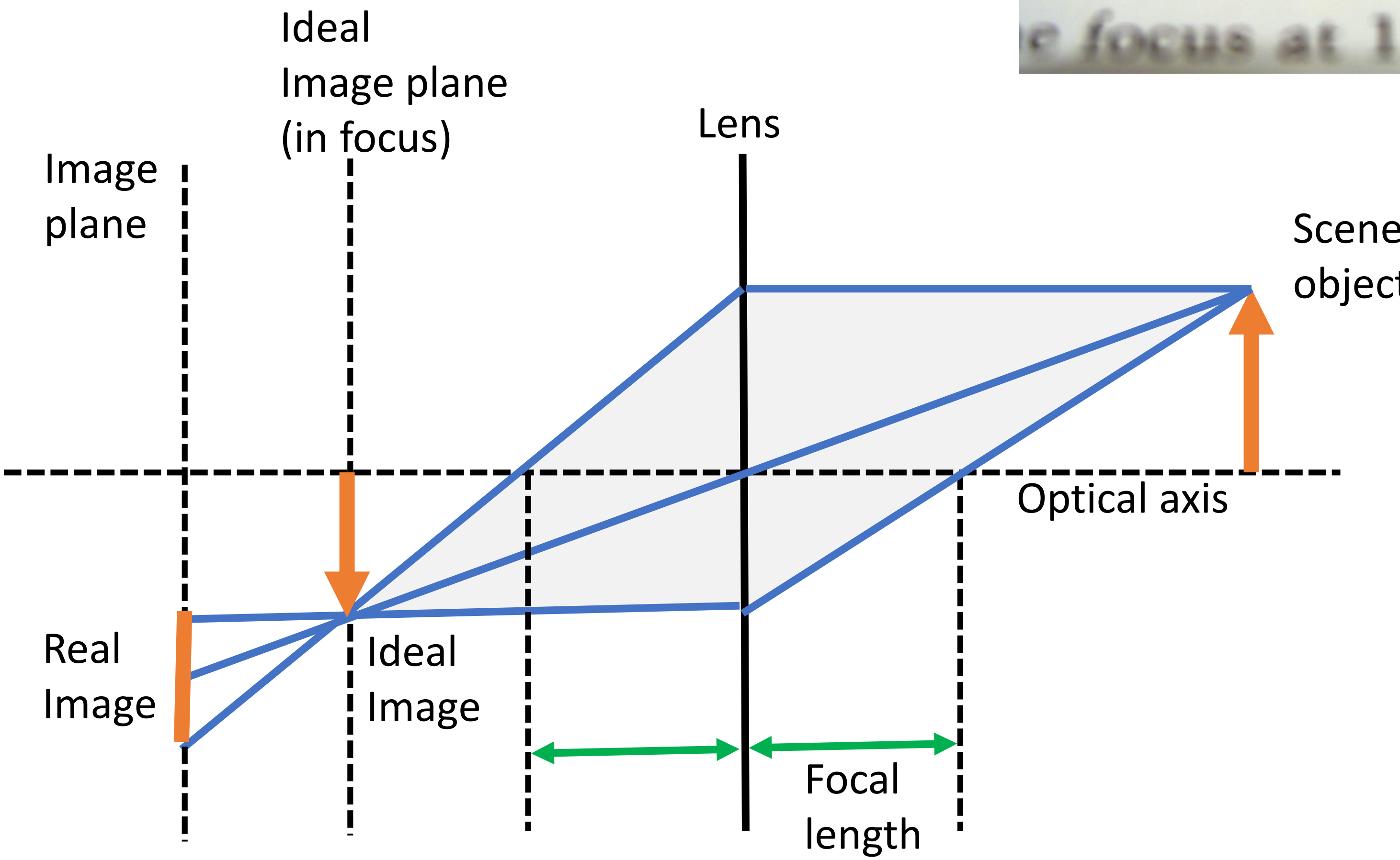
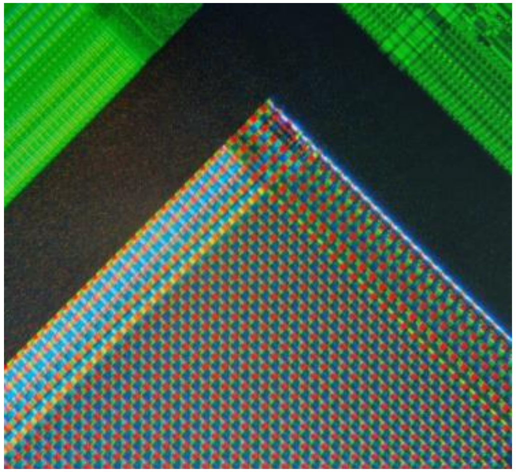
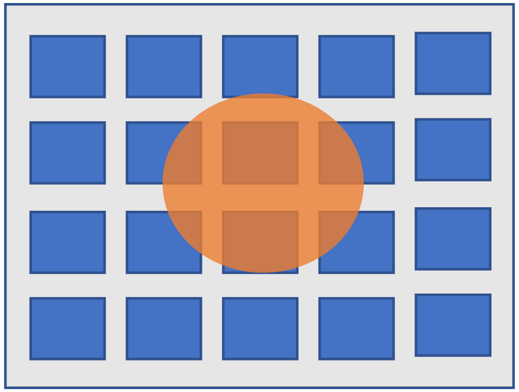


Image sensing



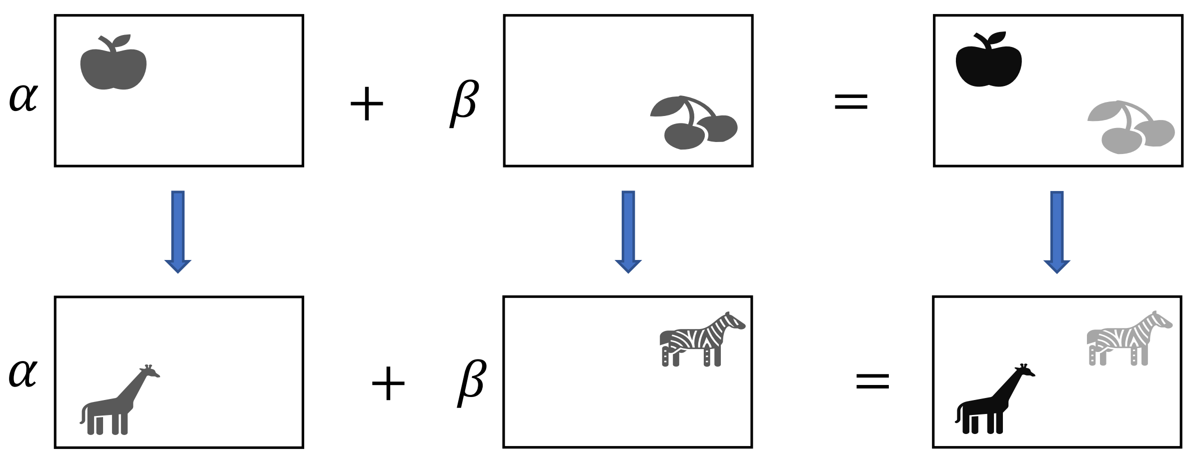
- The image plane is covered with a grid of **picture cells (aka pixels)** that “count” the number of photons during exposure
- The probability of a photon originating from a scene point being detected by a pixel is proportional to the sensor area covered by its blurred image and the exposure time



- The number of detected photons, and therefore the **intensity value** of each pixel, is affected by quantum fluctuations causing independent **white noise** in each pixel

Linear, shift-invariant systems

- For now we consider an intensity image as 2-dimensional continuous signal
$$f\colon \mathbb{R}^2 \to \mathbb{R}$$
- An image processing algorithm is taking images as input and produces a processed image as output
- Let's assume the algorithm would produce the following outputs
$$f_1 \to g_1$$
$$f_2 \to g_2$$
- The algorithm is called a **linear system**, if
$$\alpha f_1 + \beta f_2 \to \alpha g_1 + \beta g_2$$



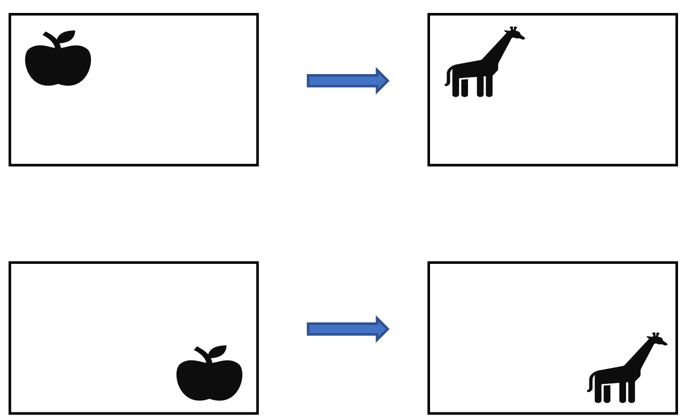
Linear, shift-invariant systems

- An algorithm is called **shift-invariant**, if the output of a shifted input image is the shifted output image, i.e. if

$$f[x, y] \to g[x, y]$$

- then

$$f[x-a, y-b] \to g[x-a, y-b]$$



Linear, shift-invariant systems

- An algorithm that is both linear and shift-invariant is called a **linear, shift-invariant system**
- Every linear, shift-invariant system can be calculated as follows

$$g[x,y] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f[x-\xi, y-\eta]h[\xi,\eta]d\xi d\eta$$

- For some function $h[\xi,\eta]$, called the **point-spread function** or **convolution kernel**
- This operation is called a **convolution**

$$g = f \otimes h$$

Convolution

- Intuitively the convolution moves the (flipped) kernel over the input image, multiplies the values and sums them up

1	3	5	3	5
3	4	5	3	5
3	5	2	1	3
1	2	3	4	1
5	7	3	2	6

$f[x, y]$

$\otimes$

0	1	0
1	2	1
0	1	0

$h[-\xi, -\eta]$

$=$

	24			

$$\begin{aligned} &1 \times 3 \\ &+1 \times 3 \\ &+2 \times 4 \\ &+1 \times 5 \\ &+1 \times 5 \\ &= 24 \end{aligned}$$

Convolution

- Intuitively the convolution moves the (flipped) kernel over the input image, multiplies the values and sums them up

$$
\begin{array}{|c|c|c|c|c|}
\hline
1 & 3 & 5 & 3 & 5 \\ \hline
3 & 4 & 5 & 3 & 5 \\ \hline
3 & 5 & 2 & 1 & 3 \\ \hline
1 & 2 & 3 & 4 & 1 \\ \hline
5 & 7 & 3 & 2 & 6 \\ \hline
\end{array}
\otimes
\begin{array}{|c|c|c|}
\hline
0 & 1 & 0 \\ \hline
1 & 2 & 1 \\ \hline
0 & 1 & 0 \\ \hline
\end{array}
=
\begin{array}{|c|c|c|c|c|}
\hline
 & & & & \\ \hline
 & 24 & 24 & & \\ \hline
 & & & & \\ \hline
 & & & & \\ \hline
 & & & & \\ \hline
\end{array}
$$

$f[x,y]$ $\quad$ $h[-\xi,-\eta]$

$$
\begin{aligned}
&1 \times 5 \\
&+1 \times 4 \\
&+2 \times 5 \\
&+1 \times 3 \\
&+1 \times 2 \\
&= 24
\end{aligned}
$$

Convolution

- Intuitively the convolution moves the (flipped) kernel over the input image, multiplies the values and sums them up

$$
\underset{f[x,y]}{\begin{array}{|c|c|c|c|c|}\hline 1 & 3 & 5 & 3 & 5 \\ \hline 3 & 4 & 5 & 3 & 5 \\ \hline 3 & 5 & 2 & 1 & 3 \\ \hline 1 & 2 & 3 & 4 & 1 \\ \hline 5 & 7 & 3 & 2 & 6 \\ \hline \end{array}}
\otimes
\underset{h[-\xi,-\eta]}{\begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & 2 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}}
=
\begin{array}{|c|c|c|c|c|}\hline & & & & \\ \hline & 24 & 24 & \boxed{20} & \\ \hline & & & & \\ \hline & & & & \\ \hline & & & & \\ \hline \end{array}
$$

$$
\begin{aligned}
&\ \ 1 \times 3 \\
&+1 \times 5 \\
&+2 \times 3 \\
&+1 \times 5 \\
&+1 \times 1 \\
&= 20
\end{aligned}
$$

Convolution

- Intuitively the convolution moves the (flipped) kernel over the input image, multiplies the values and sums them up

$$
\underset{f[x,y]}{\begin{array}{|c|c|c|c|c|}\hline 1 & 3 & 5 & 3 & 5 \\ \hline 3 & 4 & 5 & 3 & 5 \\ \hline 3 & 5 & 2 & 1 & 3 \\ \hline 1 & 2 & 3 & 4 & 1 \\ \hline 5 & 7 & 3 & 2 & 6 \\ \hline \end{array}}
\otimes
\underset{h[-\xi,-\eta]}{\begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & 2 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}}
=
\begin{array}{|c|c|c|c|c|}\hline & & & & \\ \hline & 24 & 24 & 20 & \\ \hline & 21 & & & \\ \hline & & & & \\ \hline & & & & \\ \hline \end{array}
$$

$$
\begin{aligned}
&\ \ 1 \times 4 \\
&+1 \times 3 \\
&+2 \times 5 \\
&+1 \times 2 \\
&+1 \times 2 \\
&= 21
\end{aligned}
$$

Convolution

- Intuitively the convolution moves the (flipped) kernel over the input image, multiplies the values and sums them up

1	3	5	3	5
3	4	5	3	5
3	5	2	1	3
1	2	3	4	1
5	7	3	2	6

$f[x, y]$

$\otimes$

0	1	0
1	2	1
0	1	0

$h[-\xi, -\eta]$

$=$

	24	24	20	
	21	18		

$$\begin{aligned} &1 \times 5 \\ &+1 \times 5 \\ &+2 \times 2 \\ &+1 \times 1 \\ &+1 \times 3 \\ &= 18 \end{aligned}$$

Convolution

- Intuitively the convolution moves the (flipped) kernel over the input image, multiplies the values and sums them up

$$
\underbrace{\begin{array}{|c|c|c|c|c|}\hline 1 & 3 & 5 & 3 & 5 \\ \hline 3 & 4 & 5 & 3 & 5 \\ \hline 3 & 5 & 2 & 1 & 3 \\ \hline 1 & 2 & 3 & 4 & 1 \\ \hline 5 & 7 & 3 & 2 & 6 \\ \hline \end{array}}_{f[x,y]}
\otimes
\underbrace{\begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & 2 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}}_{h[-\xi,-\eta]}
=
\begin{array}{|c|c|c|c|c|}\hline & & & & \\ \hline & 24 & 24 & 20 & \\ \hline & 21 & 18 & 14 & \\ \hline & & & & \\ \hline & & & & \\ \hline \end{array}
$$

$$
\begin{aligned}
&1 \times 3 \\
&+1 \times 2 \\
&+2 \times 1 \\
&+1 \times 3 \\
&+1 \times 4 \\
&= 14
\end{aligned}
$$

Convolution

- Intuitively the convolution moves the (flipped) kernel over the input image, multiplies the values and sums them up

$$
\underset{f[x,y]}{\begin{array}{|c|c|c|c|c|}\hline 1 & 3 & 5 & 3 & 5 \\ \hline 3 & 4 & 5 & 3 & 5 \\ \hline 3 & 5 & 2 & 1 & 3 \\ \hline 1 & 2 & 3 & 4 & 1 \\ \hline 5 & 7 & 3 & 2 & 6 \\ \hline \end{array}}
\otimes
\underset{h[-\xi,-\eta]}{\begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & 2 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}}
=
\begin{array}{|c|c|c|c|c|}\hline & & & & \\ \hline & 24 & 24 & 20 & \\ \hline & 21 & 18 & 14 & \\ \hline & 20 & & & \\ \hline & & & & \\ \hline \end{array}
$$

$$
\begin{aligned}
&1 \times 5 \\
&+1 \times 1 \\
&+2 \times 2 \\
&+1 \times 3 \\
&+1 \times 7 \\
&= 20
\end{aligned}
$$

Convolution

- Intuitively the convolution moves the (flipped) kernel over the input image, multiplies the values and sums them up

1	3	5	3	5
3	4	5	3	5
3	5	2	1	3
1	2	3	4	1
5	7	3	2	6

$f[x, y]$

$\otimes$

0	1	0
1	2	1
0	1	0

$h[-\xi, -\eta]$

$=$

	24	24	20	
	21	18	14	
	20	17		

$$\begin{aligned} &\ 1 \times 2 \\ &+1 \times 2 \\ &+2 \times 3 \\ &+1 \times 4 \\ &+1 \times 3 \\ &= 17 \end{aligned}$$

Convolution

- Intuitively the convolution moves the (flipped) kernel over the input image, multiplies the values and sums them up

1	3	5	3	5
3	4	5	3	5
3	5	2	1	3
1	2	3	4	1
5	7	3	2	6

$f[x,y]$

$\otimes$

0	1	0
1	2	1
0	1	0

$h[-\xi,-\eta]$

$=$

	24	24	20	
	21	18	14	
	20	17	15	

$$\begin{aligned} &1\times 1\\ &+1\times 3\\ &+2\times 4\\ &+1\times 1\\ &+1\times 2\\ &=15 \end{aligned}$$

Point-spread function

- The Dirac δ-function is a generalised function which is zero everywhere, except at the origin where it is "infinite"

$$\delta[x,y] = \begin{cases} \lim_{\epsilon \to 0} \frac{1}{4\epsilon^2} & if\ |x| < \epsilon, |y| < \epsilon \\ 0 & otherwise \end{cases}$$

- It represents a unit impulse (or a single point) at the origin
- Applying a convolution kernel to this single point yields the kernel itself as result

$$h[x,y] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \delta[x-\xi, y-\eta] h[\xi,\eta] d\xi d\eta$$

Point-spread function

- The convolution of a single point with a kernel yields the kernel as result

$$h = \delta \otimes h$$

- This is the reason it is called the point-spread function, because the kernel determines how a single impulse is spread

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

$\delta[x, y]$

$\otimes$

0	1	0
1	2	1
0	1	0

$h[-\xi, -\eta]$

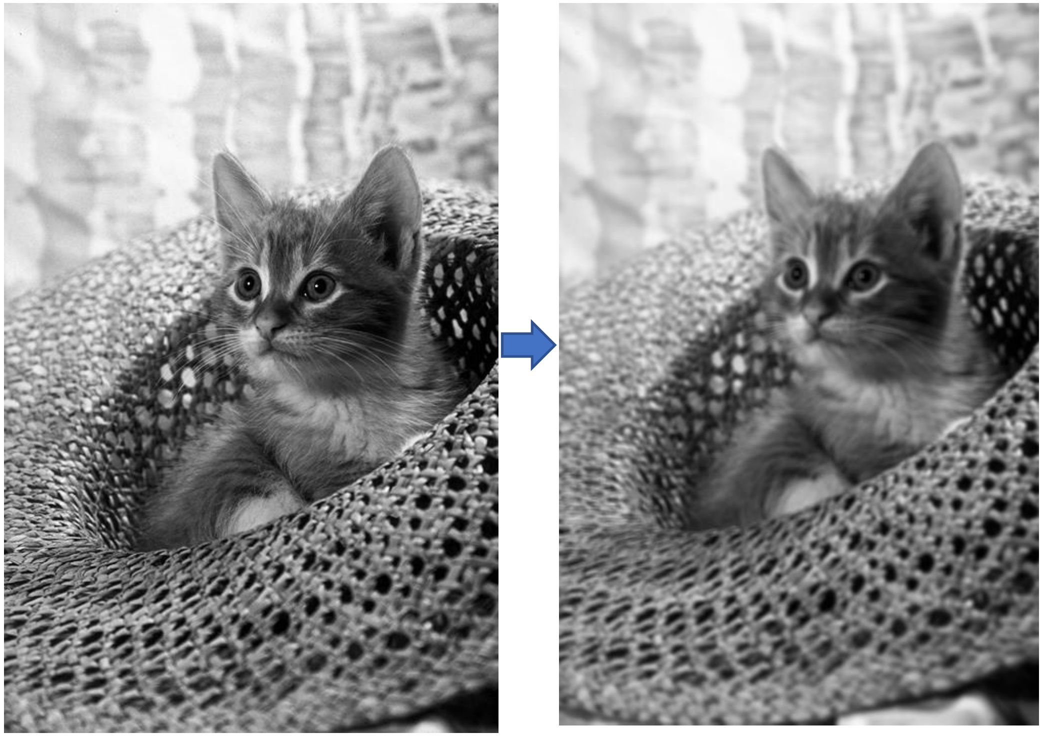
$=$

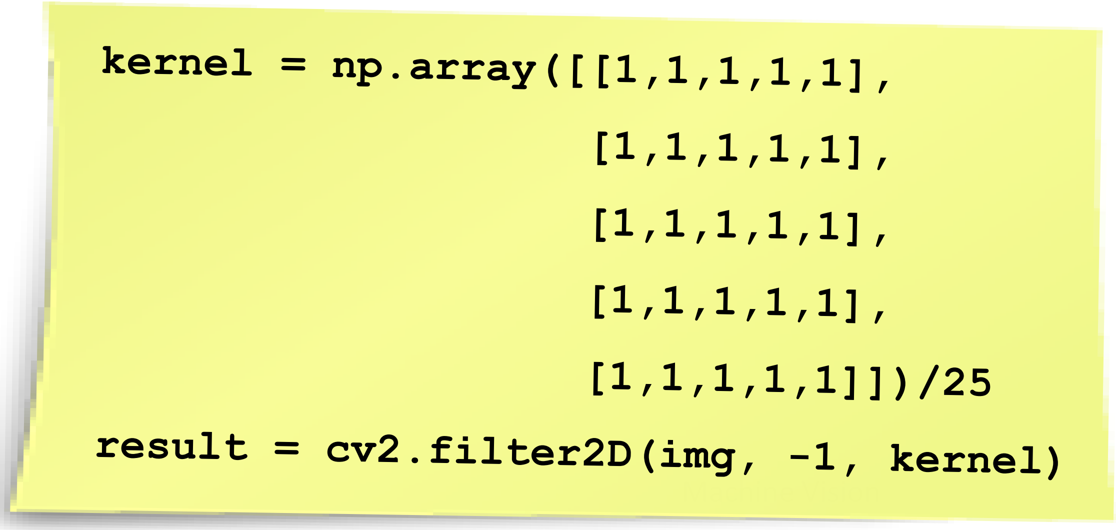
	0	1	0	
	1	2	1	
	0	1	0	

$h[x, y]$

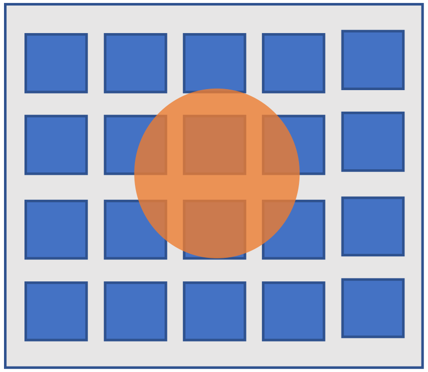
Blurring

- Applying a kernel with only positive numbers spreads out intensity
- These kernels are the basis for all blurring operations



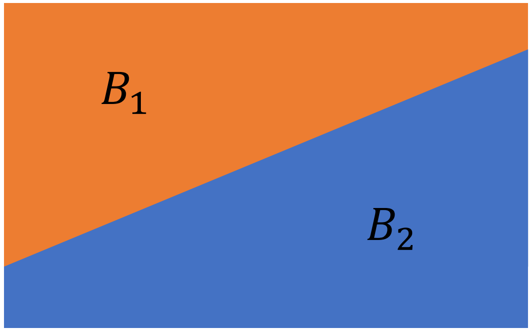


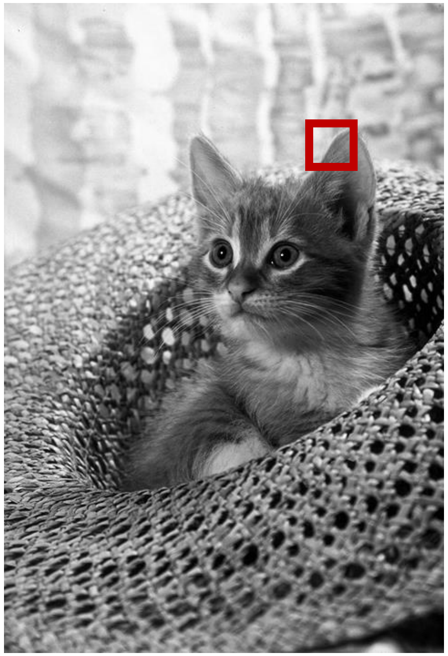
```
kernel = np.array([[1,1,1,1,1],
                   [1,1,1,1,1],
                   [1,1,1,1,1],
                   [1,1,1,1,1],
                   [1,1,1,1,1]])/25
result = cv2.filter2D(img, -1, kernel)
```



Edge detection

- Edges are local image features where brightness changes between two areas
- Edges in images typically derive from occluding contours, i.e. where one object is in front of another object





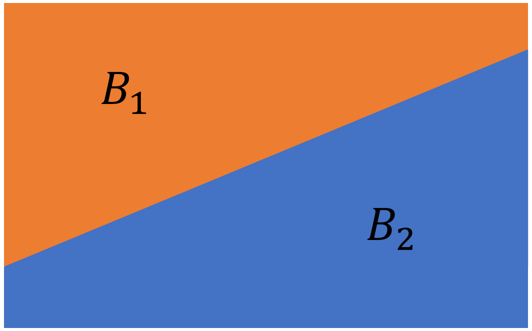
- A brightness edge is locally

$$E[x, y] = B_1 + (B_2 - B_1)u[x \sin\theta + y \cos\theta + \rho]$$

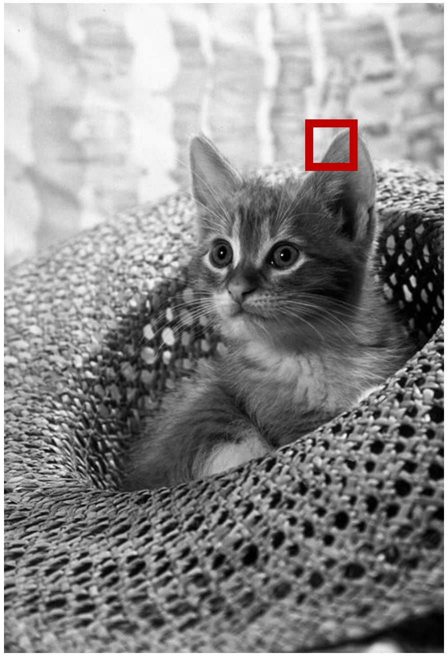
- Using the step function

$$u[z] = \int_{-\infty}^{z} \delta[t]dt$$

Edge detection



- Edges are local image features where brightness changes between two areas
- Edges in images typically derive from occluding contours, i.e. where one object is in front of another object



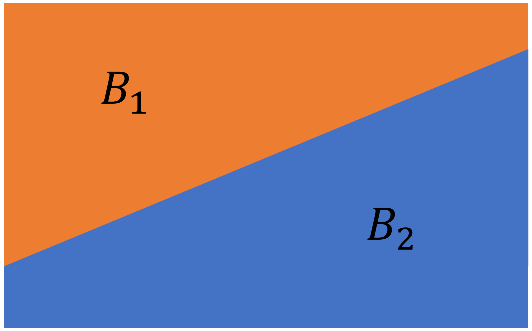
- The derivatives are only non-zero on the edge and proportional to the contrast of the edge

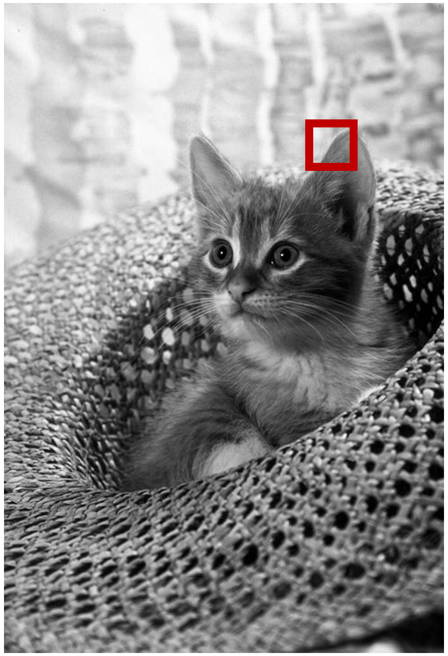
$$\frac{\partial E}{\partial x} = \sin\theta (B_2 - B_1)\delta[x \sin\theta + y \cos\theta + \rho]$$

$$\frac{\partial E}{\partial x} = -\cos\theta (B_2 - B_1)\delta[x \sin\theta + y \cos\theta + \rho]$$

Edge detection

- Edges are local image features where brightness changes between two areas
- Edges in images typically derive from occluding contours, i.e. where one object is in front of another object





- The derivatives are only non-zero on the edge and proportional to the contrast of the edge
- Therefore, to find edges in the image (i.e. the object boundaries) it is useful to look at the image gradient

Image gradients

- The derivative is a linear, shift-invariant operation, therefore it can be computed using convolution

$$\frac{\partial E}{\partial x} = \lim_{\epsilon \to 0} \frac{E[x+\epsilon, y] - E[x-\epsilon, y]}{2\epsilon} = E \otimes \delta_x[x, y]$$

$$\frac{\partial E}{\partial y} = \lim_{\epsilon \to 0} \frac{E[x, y+\epsilon] - E[x, y-\epsilon]}{2\epsilon} = E \otimes \delta_y[x, y]$$

with

$$\delta_x[x, y] = \lim_{\epsilon \to 0} \frac{\delta[x+\epsilon, y] - \delta[x-\epsilon, y]}{2\epsilon}$$

$$\delta_y[x, y] = \lim_{\epsilon \to 0} \frac{\delta[x, y+\epsilon] - \delta[x, y-\epsilon]}{2\epsilon}$$

Edge detection

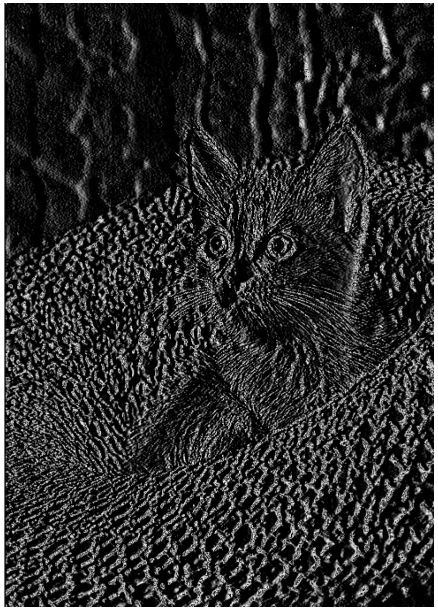
- The discrete version of the two doublets

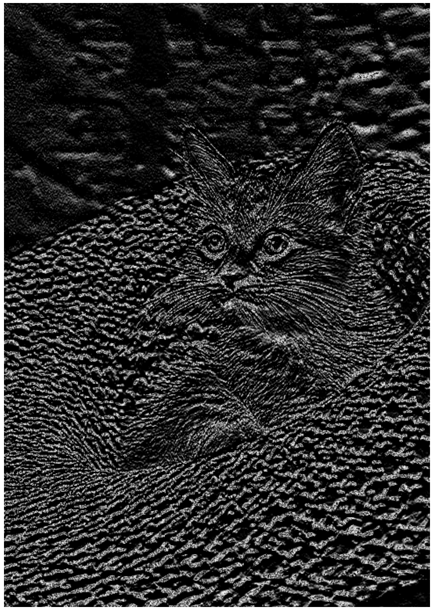
$$\delta_x[x,y] = \lim_{\epsilon\to 0}\frac{\delta[x+\epsilon,y]-\delta[x-\epsilon,y]}{2\epsilon}$$

$$\delta_y[x,y] = \lim_{\epsilon\to 0}\frac{\delta[x,y+\epsilon]-\delta[x,y-\epsilon]}{2\epsilon}$$

- Allows to calculate derivatives of images using convolution filters

```
dx = np.array([[1, 0, -1]])
dy = np.array([[1],
               [0],
               [-1]])
fx = cv2.filter2D(img, -1, dx)
fy = cv2.filter2D(img, -1, dy)
```





Properties of convolutions

- Convolutions are commutative, i.e.

$$g = f \otimes h = h \otimes f$$

- and associative

$$g = (f \otimes h_1) \otimes h_2 = f \otimes (h_1 \otimes h_2)$$

- Therefore, the order in which convolution operations are applied does not matter
- In particular, convolutions can be pooled together and applied as one single convolution to an input image

The modulation-transfer function

- Applying a convolution to the complex function

$$f[x,y] = e^{i(ux+vy)} = \cos[ux+vy] + i\sin[ux+vy]$$

- Yields

$$g[x,y] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{i(u(x-\xi)+v(y-\eta))} h[\xi,\eta]d\xi d\eta$$

$$= e^{i(ux+vy)} \underbrace{\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-i(u\xi+v\eta)} h[\xi,\eta]d\xi d\eta}_{H(u,v)}$$

- The **modulation transfer function** $H(u,v)$ does not depend on x or y, therefore the convolution operation has a multiplicative effect on individual frequencies (i.e. $e^{i(ux+vy)}$ is an eigenfunction of convolution in two dimensions)

Fourier transformation

- Every function can be considered as a sum of an infinite number of sinusoidal waves

$$f[x, y] = \frac{1}{4\pi^2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F[u, v] e^{i(ux+vy)} du dv$$

- This frequency representation $F[u, v]$ is called the **Fourier transformation** of the function $f[x, y]$

- Using the modulation transfer function the convolution is then

$$g[x, y] = f \otimes h = \frac{1}{4\pi^2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} H[u, v] F[u, v] e^{i(ux+vy)} du dv$$

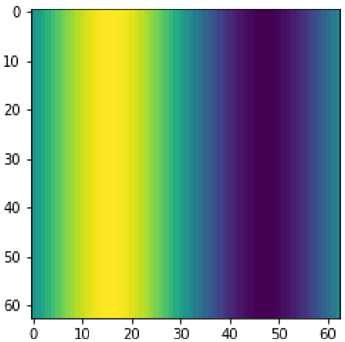
- Now looking at the Fourier transformation of $g[x, y]$ one can see that

$$G[u, v] = F[u, v] H[u, v]$$

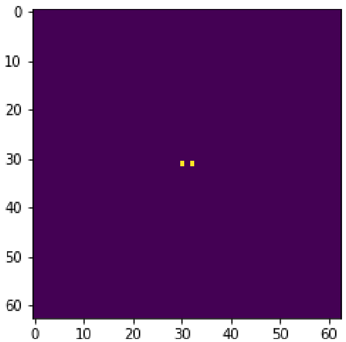
- **Convolution is simply a multiplication in the frequency domain!**

Fourier spectrum

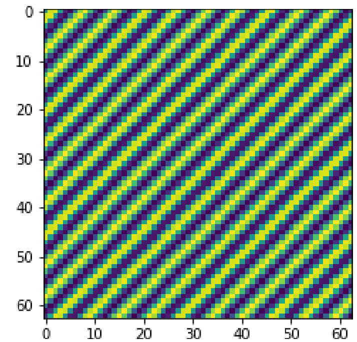
$f[x,y]$



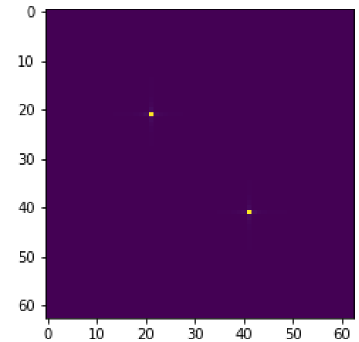
$|F[u,v]|$



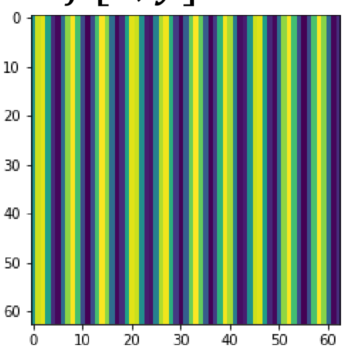
$f[x,y]$



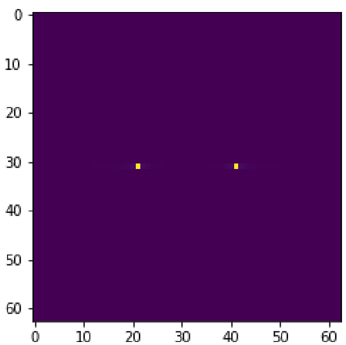
$|F[u,v]|$



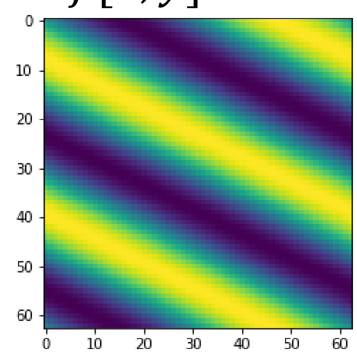
$f[x,y]$



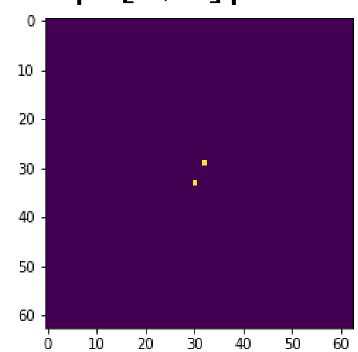
$|F[u,v]|$



$f[x,y]$

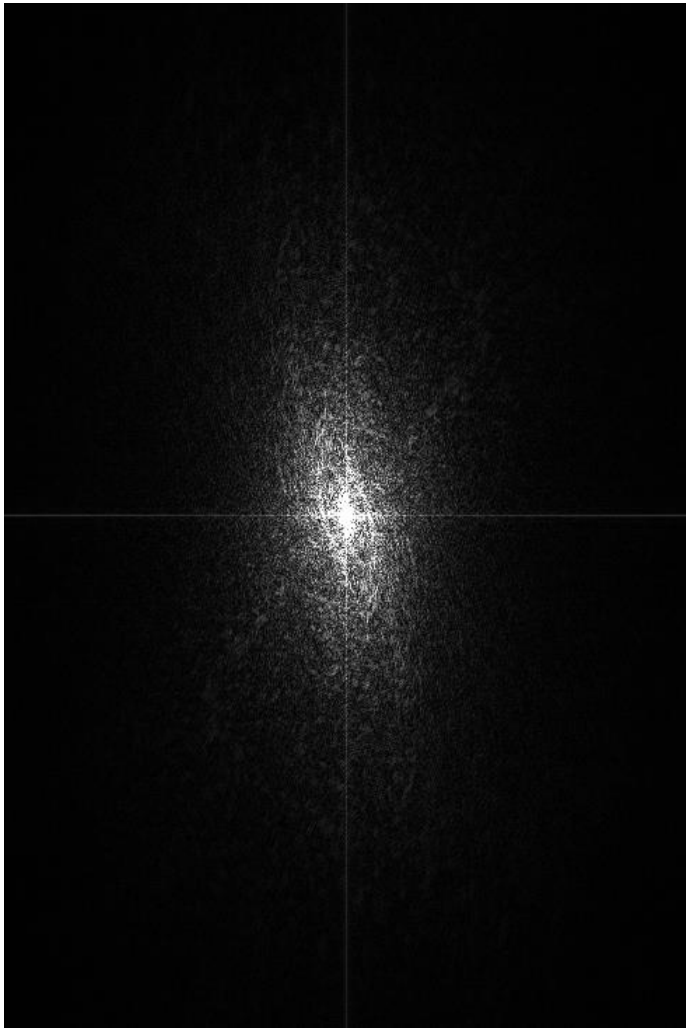


$|F[u,v]|$



Fourier spectrum

$f[x, y]$



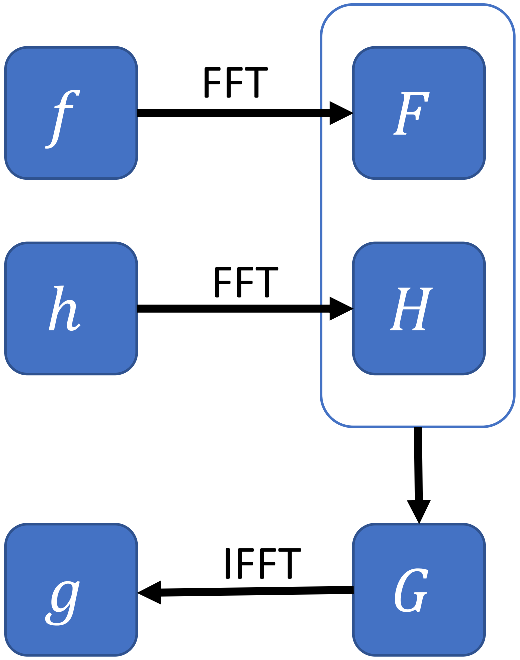
$|F[u, v]|$

Convolution in the frequency domain

- Convolution in the spatial domain
$$g[x, y] = f[x, y] \otimes h[x, y]$$
- Becomes multiplication in the frequency domain
$$G[u, v] = F[u, v]H[u, v]$$

- The modulation transfer function attenuates/dampens certain frequencies and directions
- White noise is uniform across the spectrum, so the modulation transfer function also tells us what noise components are attenuated and what noise components are dampened by the application of a filter
- We can also use this to understand how convolution filters can be inverted by observing that this is only possible if the kernel spectrum is non-zero everywhere, in which case the inverse is simply
$$F[u, v] = \frac{G[u, v]}{H[u, v]}$$

Linear Filters and FFT



- To calculate the convolution

$$g = f \otimes x$$

- We can do the following
  - Calculate Fourier transform of f and h
  - Calculate $G = FH$ by multiplication
  - Calculate the inverse Fourier transform of G

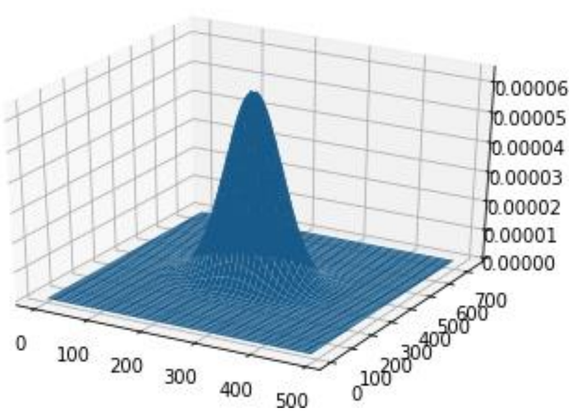
- Because spatial convolution requires $O(n^2)$ multiplications and the Fast-Fourier-Transform algorithm runs in $O(n \log n)$ this strategy can be faster (usually depending on the size of h)

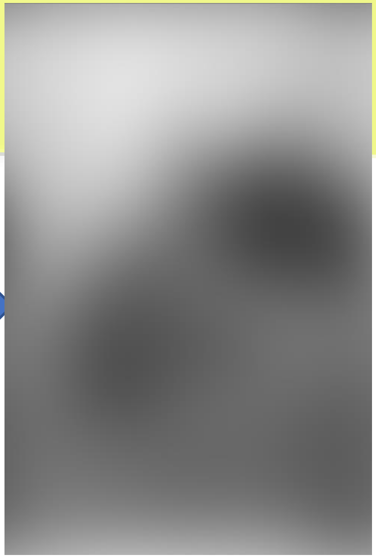
FFT in Python

```
sigma = 50
x,y = np.meshgrid(np.arange(0,len(img[0])),np.arange(0,len(img)))
kernel = np.exp(-((x-len(img[0])/2)**2+(y-len(img)/2)**2)/(2*sigma**2))/(2*np.pi*sigma**2)

ft_img = np.fft.fft2(img)
ft_kernel = np.fft.fft2(np.fft.fftshift(kernel))

result = abs(np.fft.ifft2(ft_img * ft_kernel))/255
# result = cv2.filter2D(img, -1, kernel)
cv2.imshow("result", result)
```

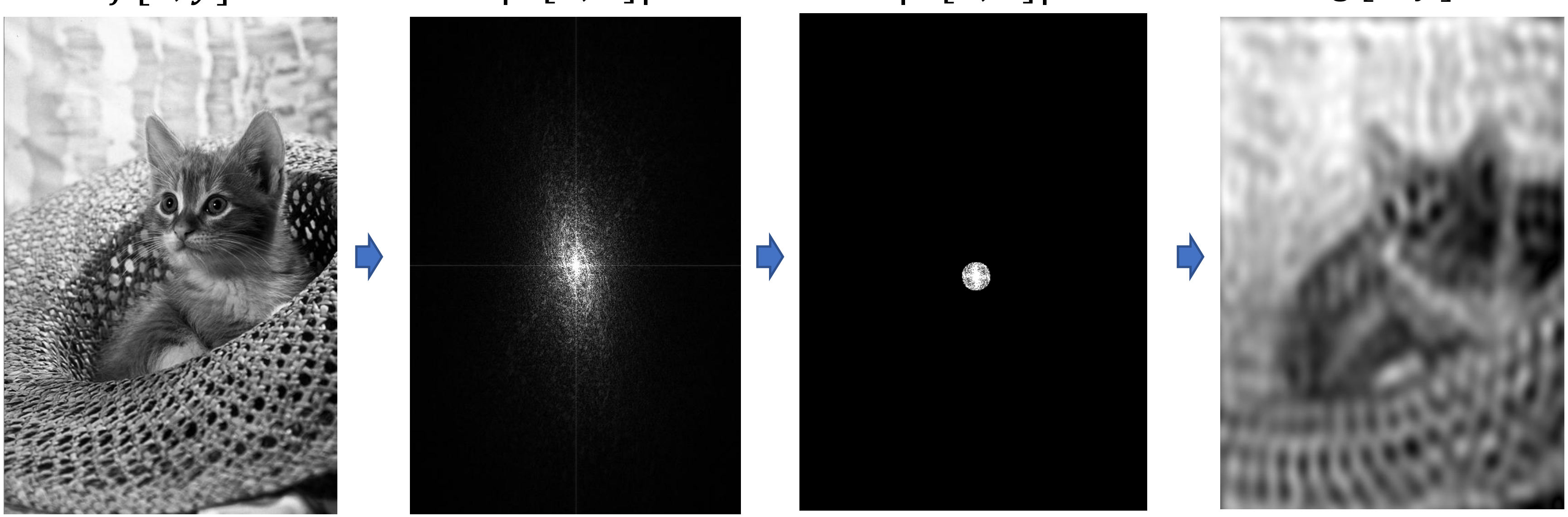




Low-pass filter

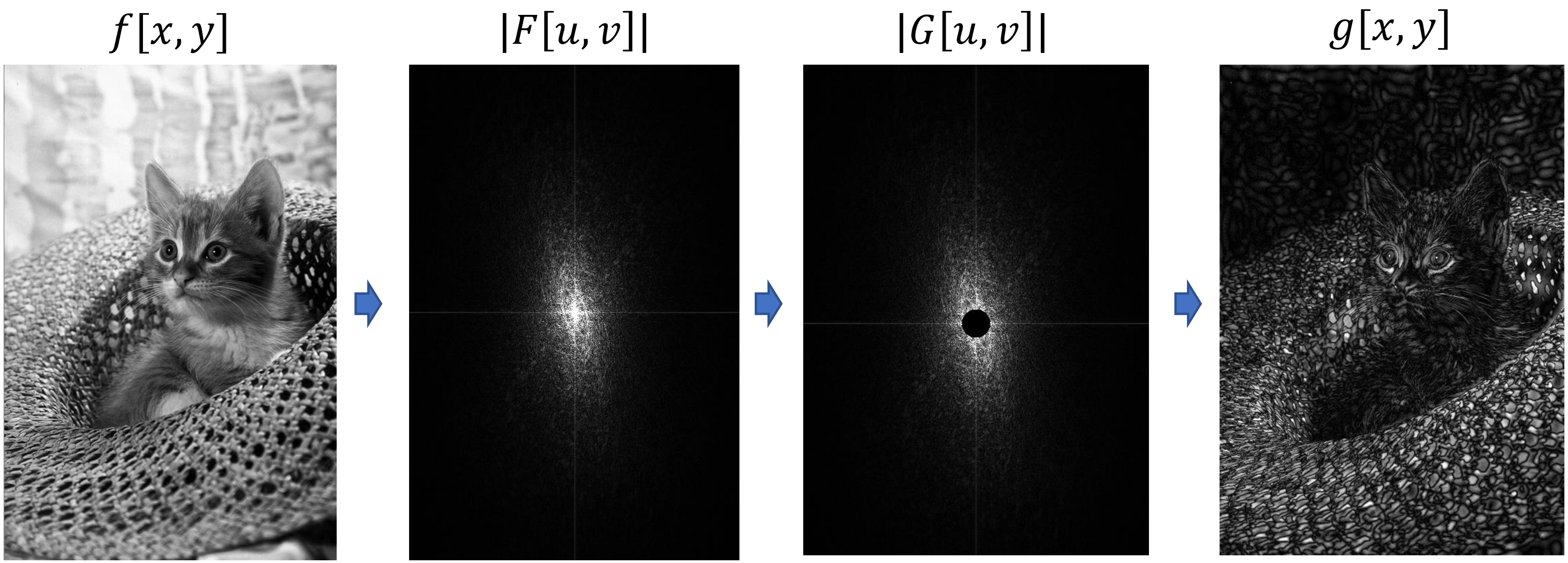
- Removing high frequencies from the image is called **low-pass** filtering
- It only retains homogeneous areas and removes all high-frequency edges
- The result is a blurred image

$f[x, y]$ $|F[u, v]|$ $|G[u, v]|$ $g[x, y]$



High-pass filter

- Removing low frequencies from the image is called **high-pass** filtering
- It only retains edges and removes all low-frequency homogeneous areas
- The result is an image with homogeneous areas removed



Smoothing at different scales

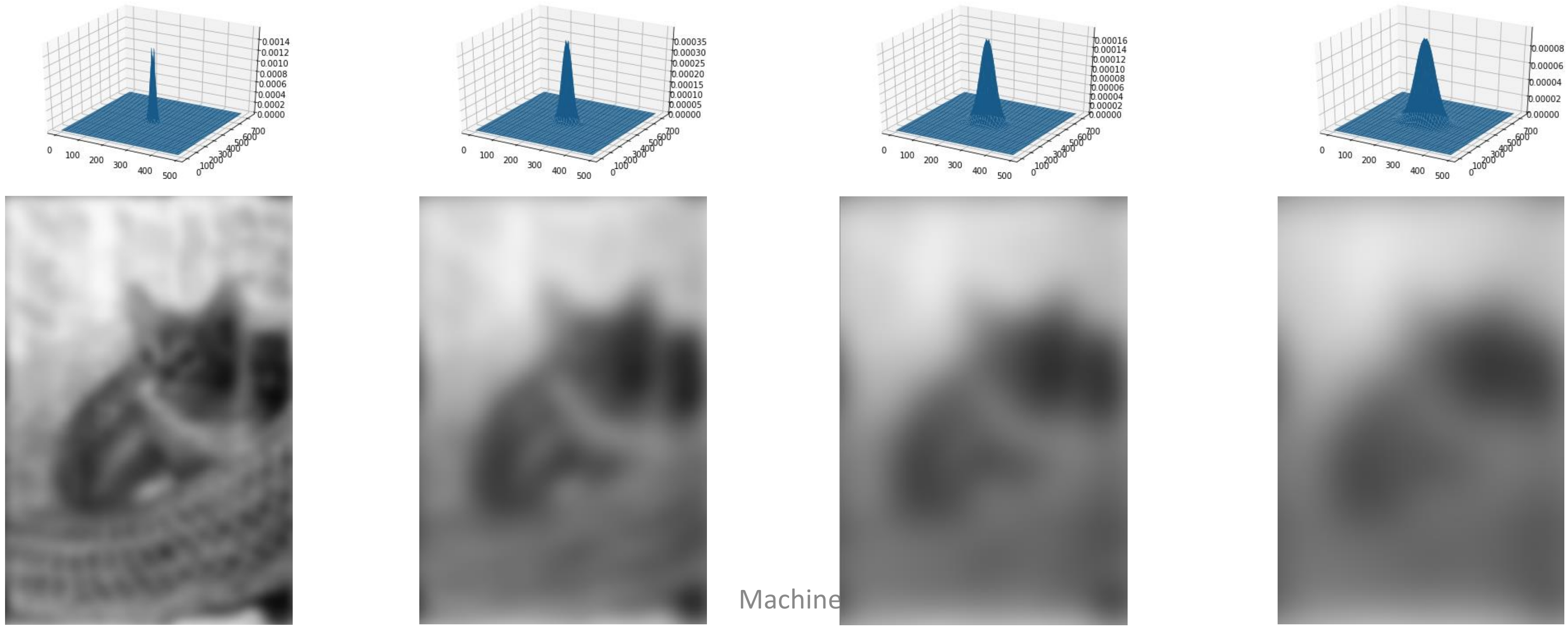
- We can filter the image with Gaussians of different size

$$g_\sigma = \frac{1}{2\pi\sigma^2} \exp - \frac{x^2 + y^2}{2\sigma^2}$$

- To generate images at different smoothing scales

$$\bar{f} = g_\sigma \otimes f$$

$\sigma = 10$ $\sigma = 20$ $\sigma = 30$ $\sigma = 40$



Derivatives at different scales

- Calculating derivatives on a discrete image using convolution filters is not practical (because it would require infinitely small pixels)

$$\delta_x[x, y] = \lim_{\epsilon \to 0} \frac{\delta[x+\epsilon, y] - \delta[x-\epsilon, y]}{2\epsilon}$$

$$\delta_y[x, y] = \lim_{\epsilon \to 0} \frac{\delta[x, y+\epsilon] - \delta[x, y-\epsilon]}{2\epsilon}$$

- Instead, we have to calculate derivatives on a smoothed version of the image (remember, convolution is associative and commutative)

$$\bar{f}_x = (\delta_x \otimes g_\sigma) \otimes f$$

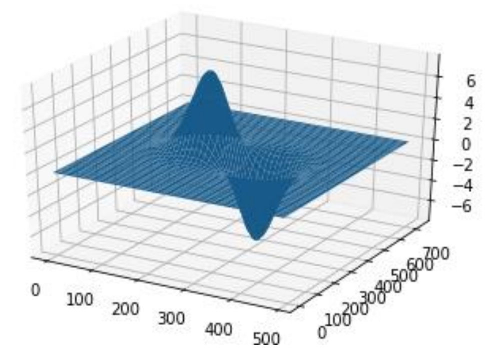
$$\bar{f}_y = (\delta_y \otimes g_\sigma) \otimes f$$

Derivatives at different scales

- We can calculate the derivative of the Gaussian function directly

$$\frac{\partial g_\sigma}{\partial x} = -\frac{x}{2\pi\sigma^4}\exp -\frac{x^2+y^2}{2\sigma^2}$$

$$\frac{\partial g_\sigma}{\partial y} = -\frac{y}{2\pi\sigma^4}\exp -\frac{x^2+y^2}{2\sigma^2}$$



- And calculate the (scale-dependent) derivatives using the following convolutions

$$\bar{f}_x = \frac{\partial g_\sigma}{\partial x} \otimes f$$

$$\bar{f}_y = \frac{\partial g_\sigma}{\partial y} \otimes f$$

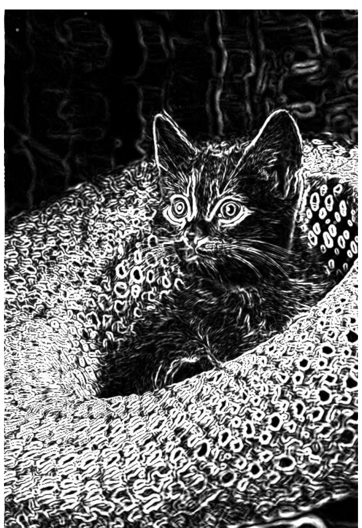
Edge detection

- If we want to have a direction independent edge detector we can look at the squared gradient image

$$\left(\frac{\partial g_\sigma}{\partial x} \otimes f\right)^2 + \left(\frac{\partial g_\sigma}{\partial y} \otimes f\right)^2$$

- The Derivative of Gaussian operator is still scale-dependent, which suggests that every edge does not only have a particular direction, but also a particular scale
- We will see later how scale invariant feature detection can be achieved (SIFT)

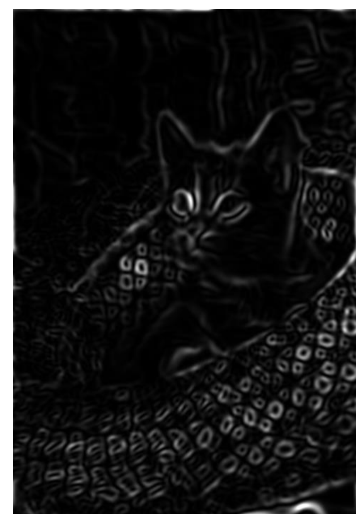
$\sigma = 1$



$\sigma = 2$

$\sigma = 3$

$\sigma = 4$

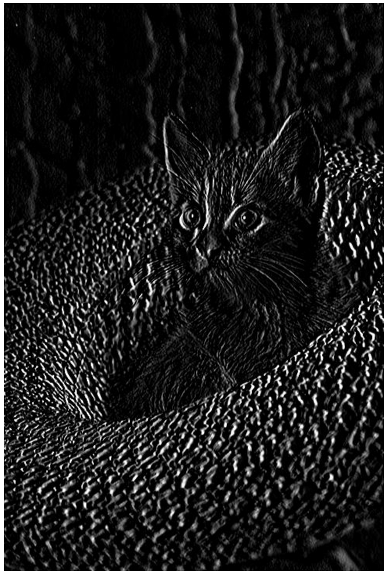


$\sigma = 5$

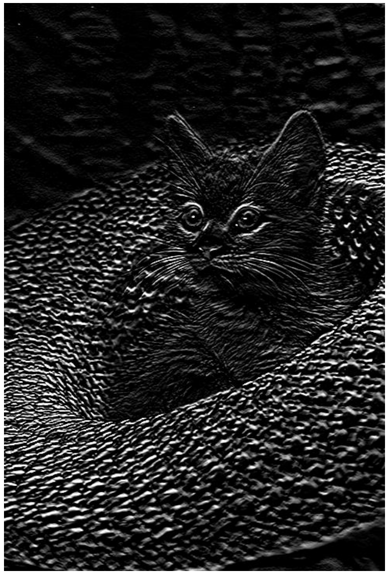
Sobel operator

- In practical applications it makes sense to choose convolution kernels that approximate derivatives of Gaussians while at the same time are efficient to compute
- The **Sobel operator** applies a 3x3 convolution filter which applies a difference operation in x-direction and a smoothing operation in y-direction (and vice versa) similar to a derivative of Gaussian operator
- The scale in this case is fixed to the image resolution

1	0	-1
2	0	-2
1	0	-1



1	2	1
0	0	0
-1	-2	-1



Summary

- Every linear, shift-invariant system can be expressed as convolution
- A convolution in the spatial domain corresponds to a multiplication in the frequency domain
- Image smoothing and image derivatives are two important examples of linear, shift-invariant operations
- Occluding object boundaries create edges in images, which can be detected using such linear, shift-invariant filters
- These edge features are characterised by their direction and by their scale

Thank you for your attention!