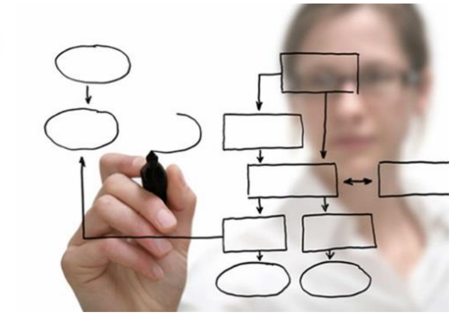
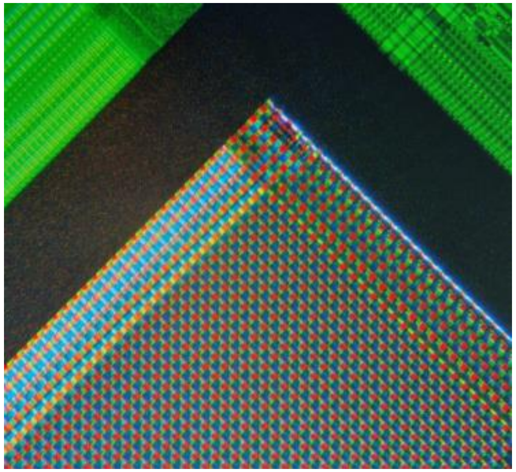
Computing@CIT



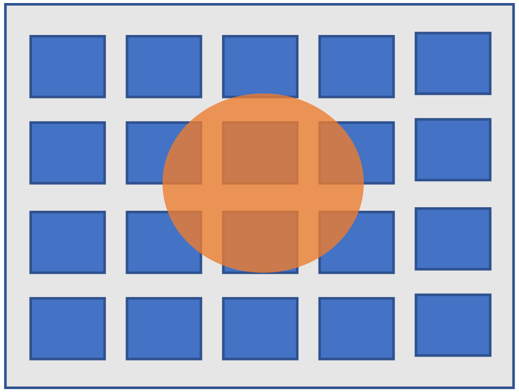
Machine Vision

Lecture 3: Image Sampling

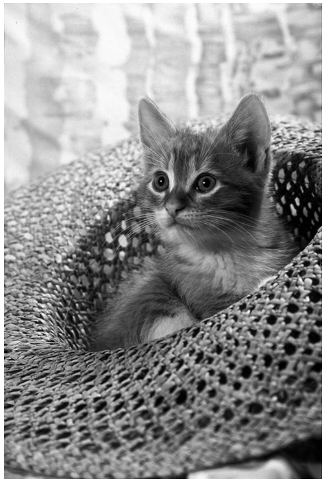
Image sensing



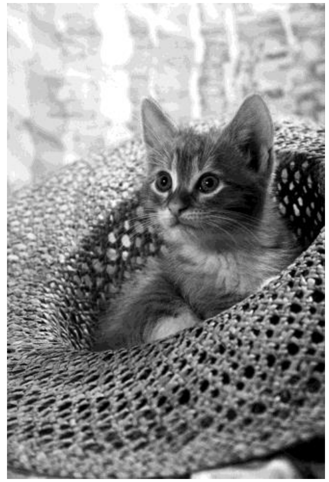
- While it is sometimes useful to consider images as continuous functions mapping the 2D image plane to some real-valued intensity, the process of image sensing is inherently discretised
- There are two types of discretisation that we need to consider:
    - **Radiometric sampling**, that is how the intensity/colour of each pixel is encoded
    - **Geometric sampling**, that is at what spatial resolution the incoming beams of light are measured



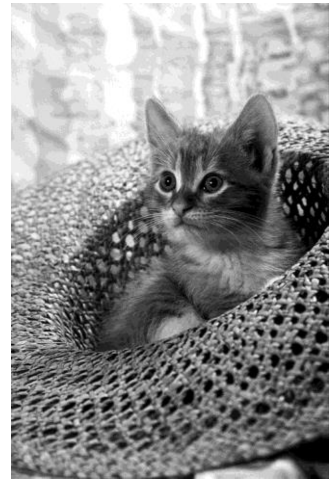
Radiometric depth



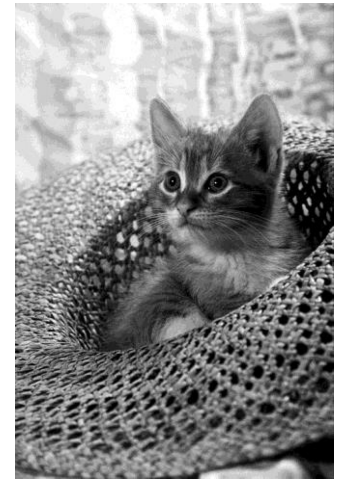
8-bit



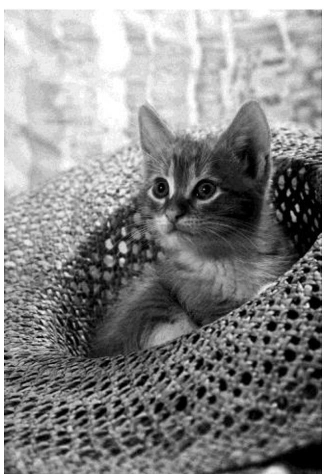
7-bit



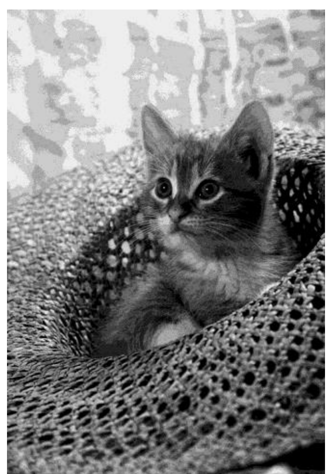
6-bit



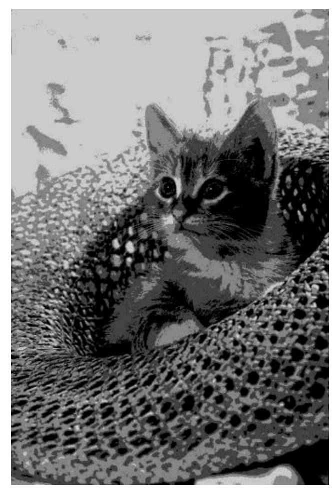
5-bit



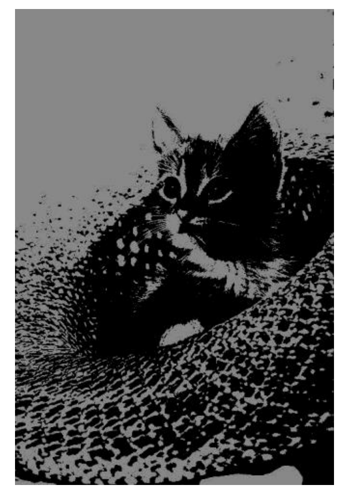
4-bit



3-bit



2-bit



1-bit

Radiometric sampling

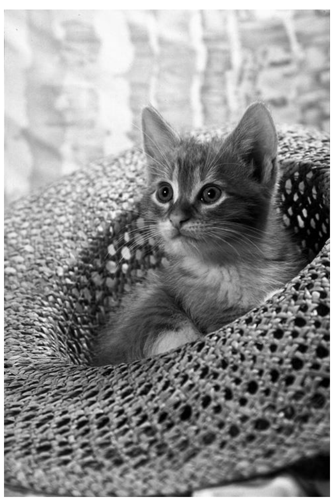
- Irradiance is the power per unit area of light falling on a surface
- Every pixel measures a brightness value by integrating irradiance over a defined amount of exposure time
- Exposure times are usually calibrated so that an optimal dynamic range between black and white is achieved
- The brightness is then discretised, typically such that 0=black and 255=white
- High Dynamic Range (HDR) cameras use more bits for discretising brightness

Image histogram

- The image brightness histogram is the function that for each brightness z counts the number of pixels in the image that take on this particular brightness value

$$h[z] = \sum_{x,y} \delta[I[x,y] - z]$$

- It shows how large the areas of a particular brightness are



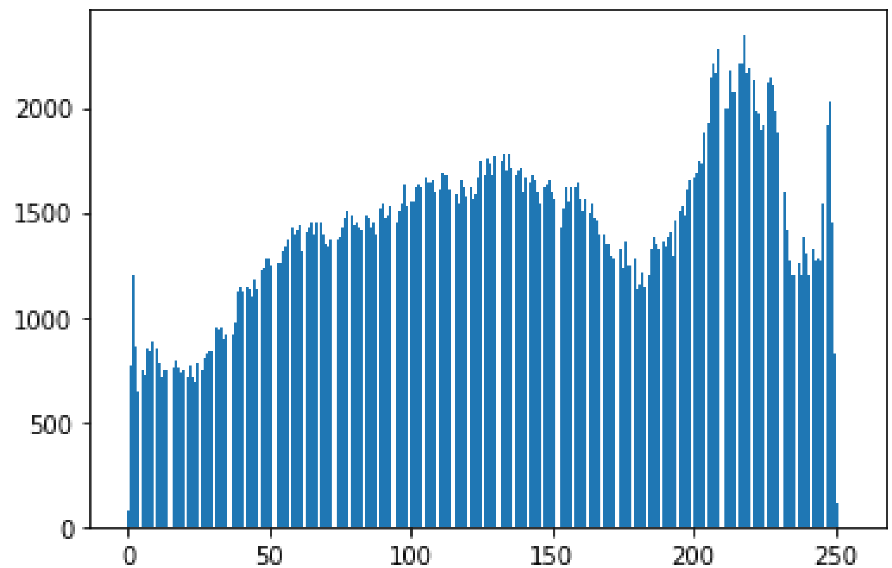


Image histogram

- The image brightness histogram is the function that for each brightness z counts the number of pixels in the image that take on this particular brightness value

$$h[z] = \sum_{x,y} \delta[I[x,y] - z]$$

- It shows how large the areas of a particular brightness are

- In OpenCV it is calculated as follows

Channels

Mask

Number of bins

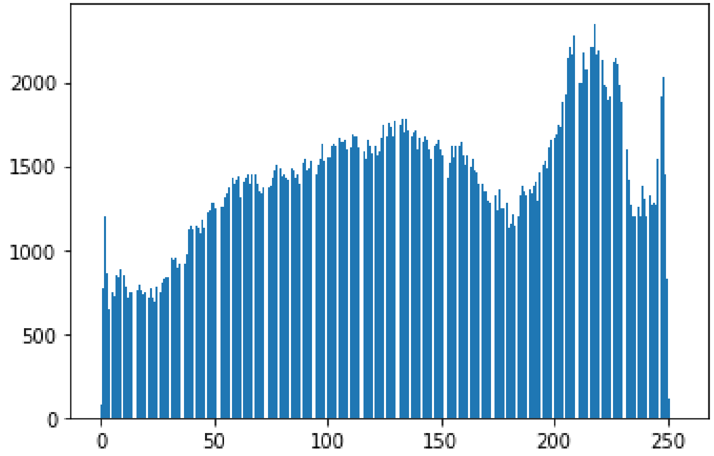
```
hist = cv2.calcHist([img],[0],None,[256],[0,256])
plt.bar(range(len(hist)),hist.flatten())
```

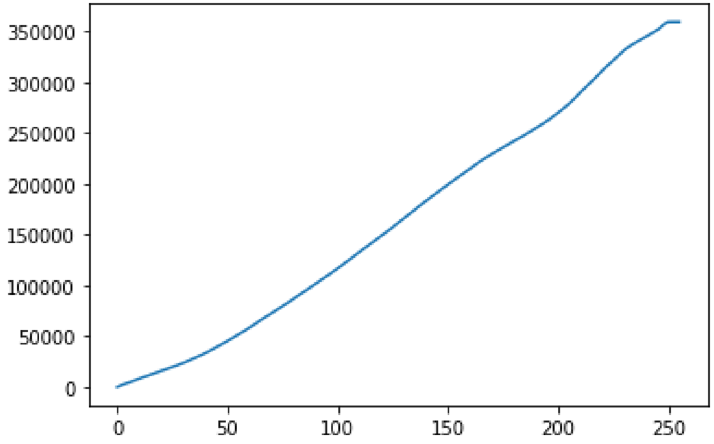
Brightness range

Cumulative histogram

- The cumulative histogram is the number of pixels in the image that have a brightness smaller than a particular value t

$$H[t] = \sum_{z \leq t} h[z]$$






```
plt.plot(np.cumsum(hist))
```

Histogram operations

- A function T applied to all pixels of an image irrespective of position and transforming its brightness is called a histogram operation

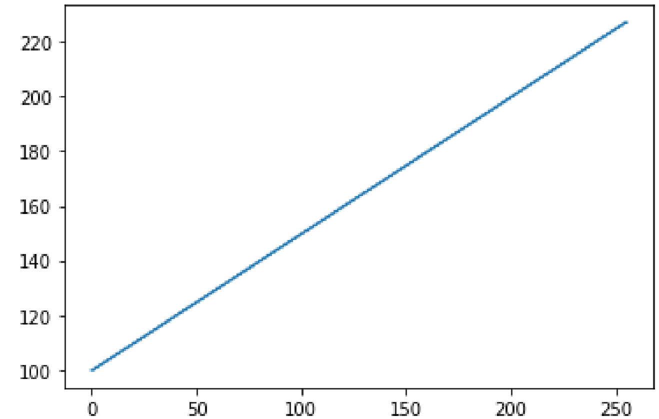
$$I'[x,y] = T[I[x,y]]$$

- Because an image is typically discretised this can be realised as a simple lookup table

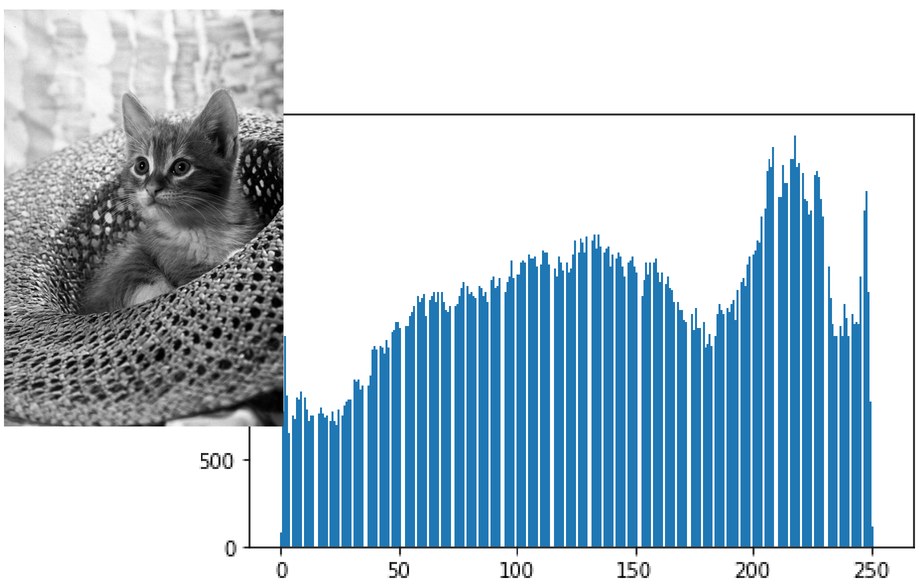
```
T = np.array(np.arange(256)/2,dtype=np.uint8)
img2 = T[img]
```

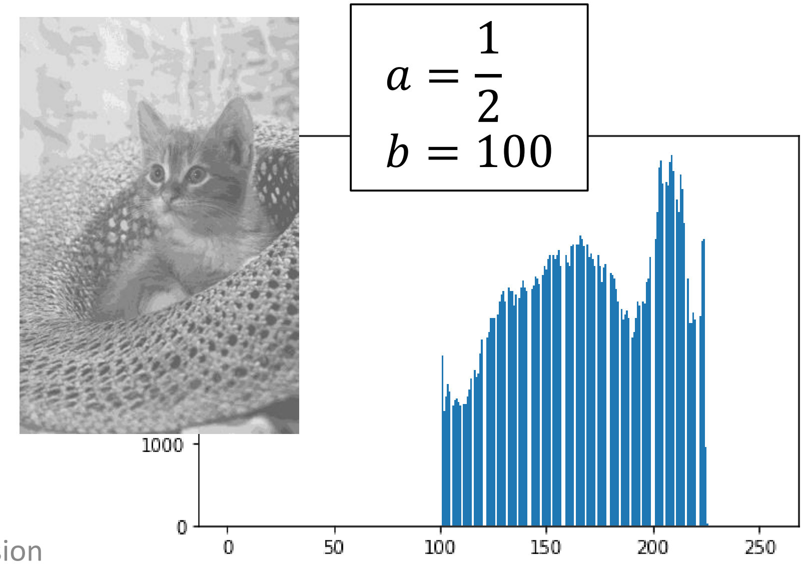
Contrast and brightness adjustment

- If T is linear the histogram is "squeezed" and "shifted" resulting in a contrast and brightness adjustment of the image



$$I'[x, y] = a\, I[x, y] + b$$



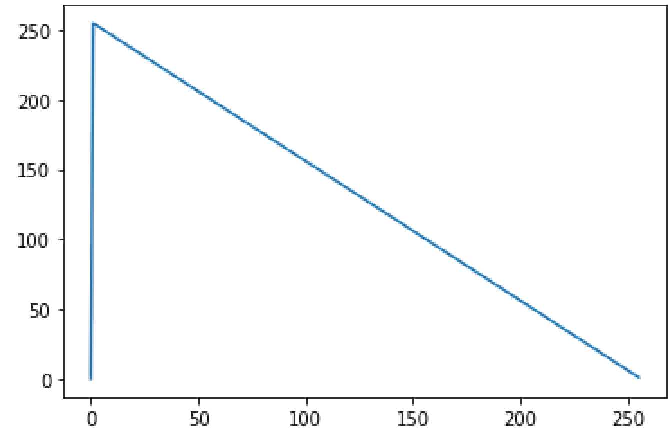


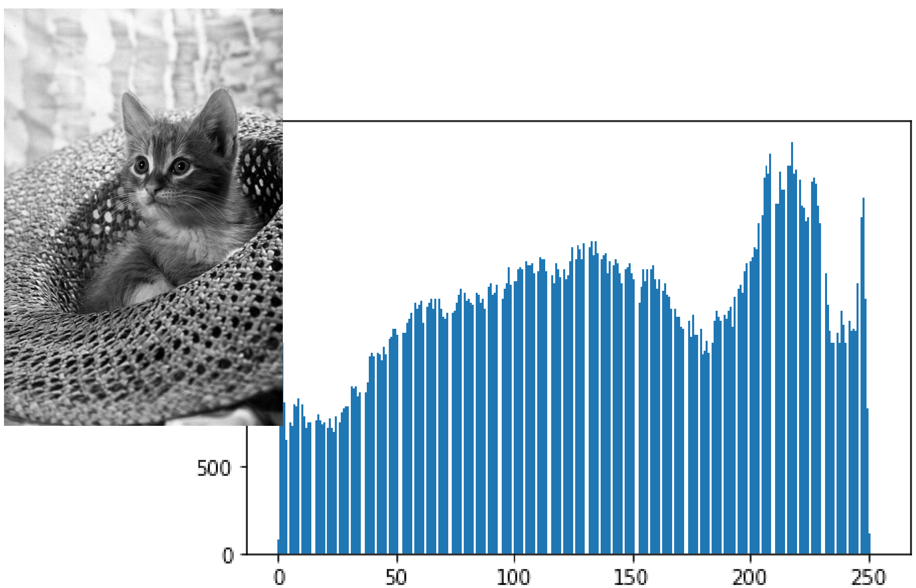
$$a = \frac{1}{2}$$
$$b = 100$$

Negative images

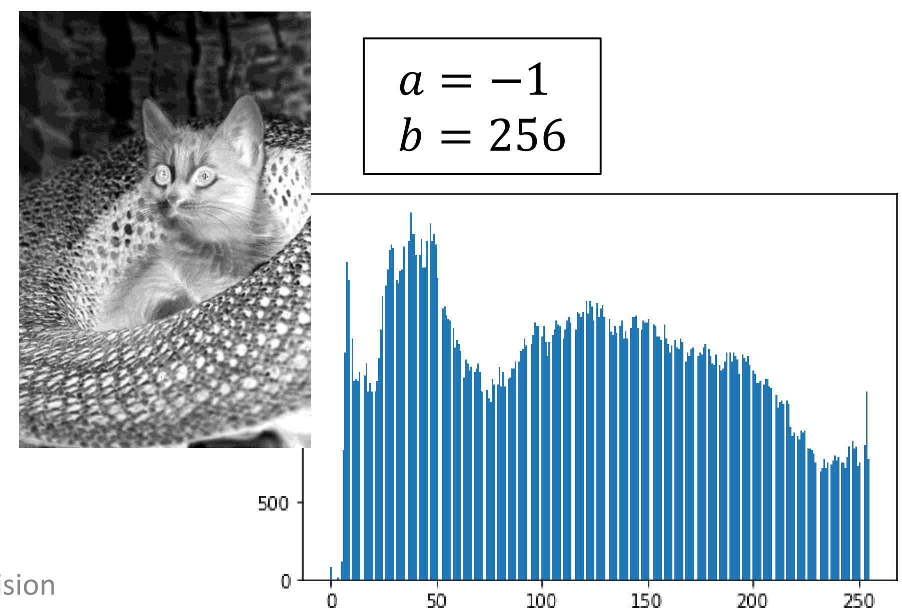
- A linear transformation with negative a transforms the image into the negative image

$$I'[x,y] = a\, I[x,y] + b$$



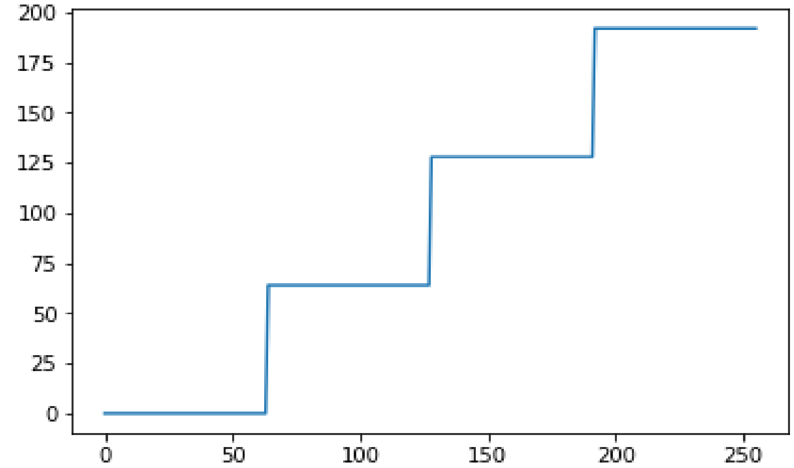


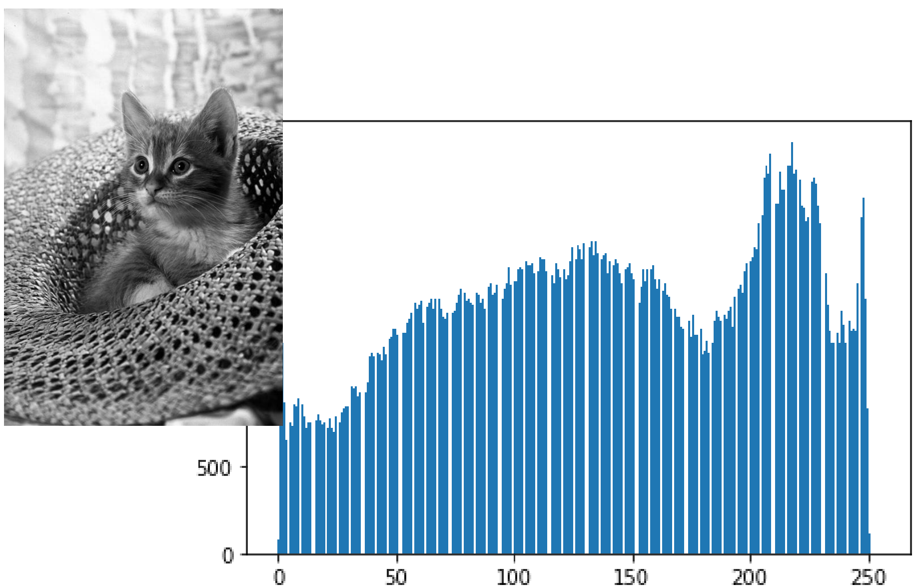
$$a = -1$$
$$b = 256$$

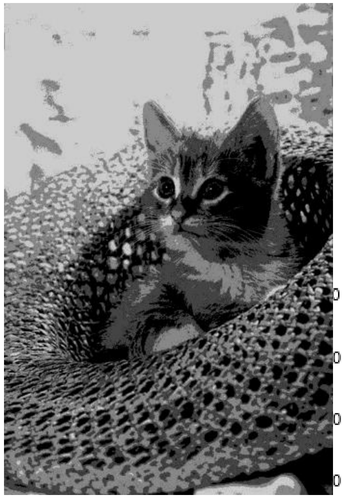


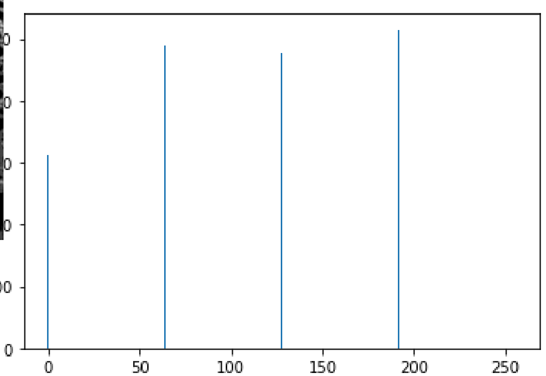
Radiometric discretisation

- If the transformation T is a step-function the radiometric discretisation can be adjusted









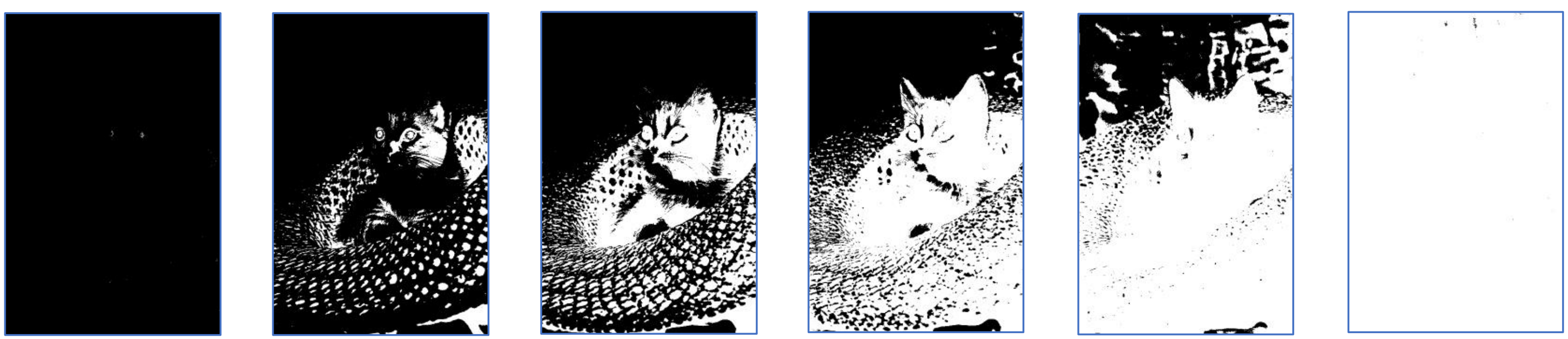
Thresholding

- An important special case is the step-function with just a single step

$$I'[x,y] = \begin{cases} 0 & I[x,y] \geq T \\ 1 & I[x,y] < T \end{cases}$$

- The result is a **binary image**, where all areas exceeding a given threshold T are white and the rest of the image is black
- This can be useful for image segmentation

$T = 0$ $T = 50$ $T = 100$ $T = 150$ $T = 200$ $T = 250$



Thresholding

- An important special case is the step-function with just a single step

$$I'[x,y] = \begin{cases} 0 & I[x,y] \geq T \\ 1 & I[x,y] < T \end{cases}$$

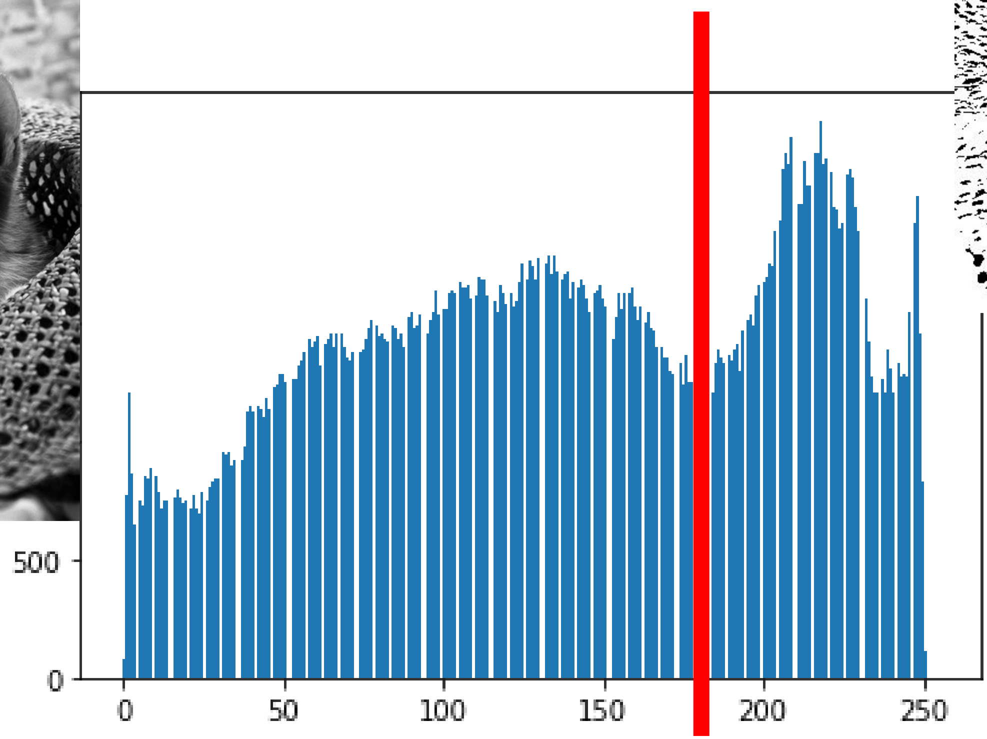
- The result is a **binary image**, where all areas exceeding a given threshold T are white and the rest of the image is black
- This can be useful for image segmentation

- Because of this there is a dedicated function in OpenCV

```
ret,binary_image = cv2.threshold(img, T, 255, cv2.THRESH_BINARY_INV)
```

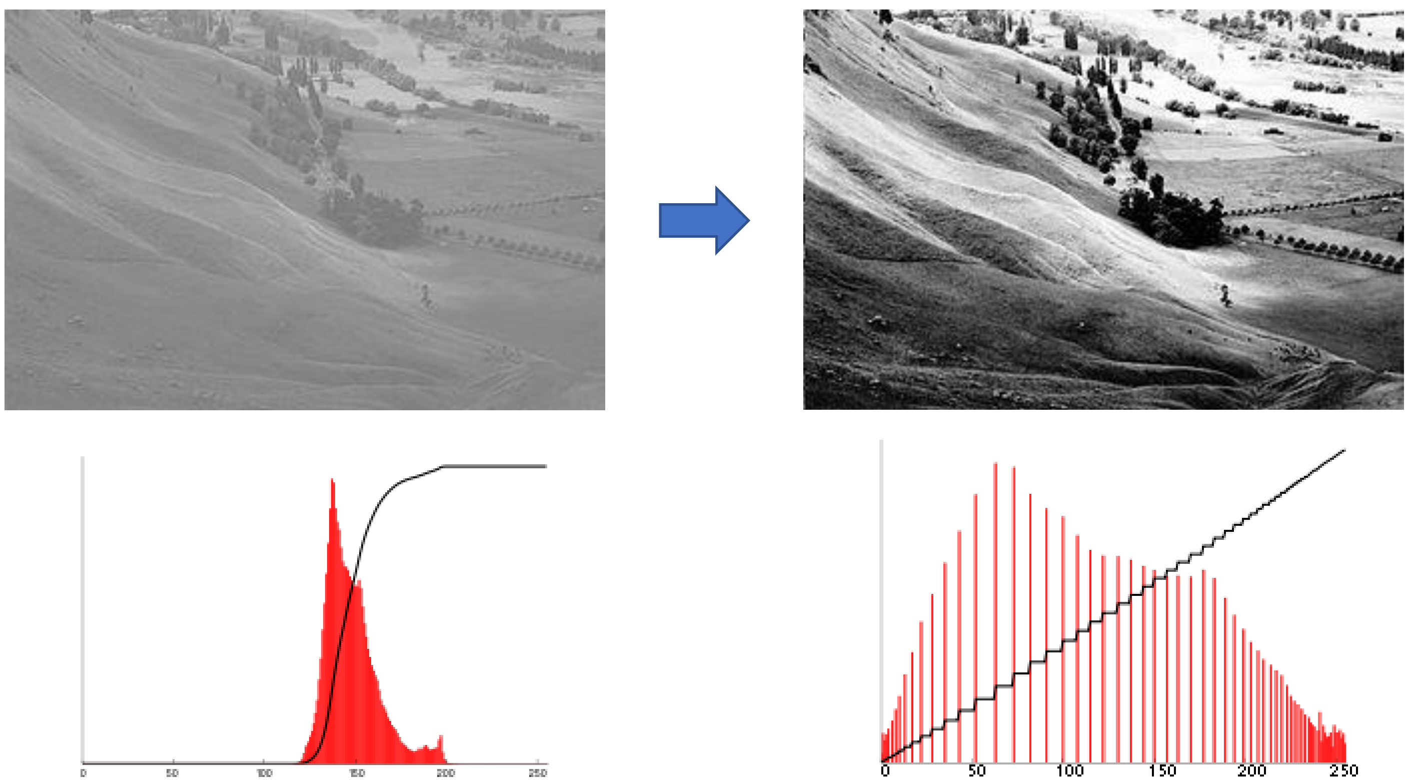
Thresholding

- Sometimes the histogram can be useful for determining a good threshold value



Histogram equalisation

- Histogram equalisation tries to improve the contrast of an image by ensuring that all brightness values occur equally frequent



Histogram matching

- A more generic operation is **histogram matching**, which takes the histogram of the input image as input

$$h[z] = \sum_{x,y} \delta[I[x,y] - z]$$

- And calculates the transformation $I' = TI$ so that the histogram of this transformed image is the given target histogram

$$h'[z] = \sum_{x,y} \delta[T[I[x,y]] - z]$$

- **Histogram equalisation** is the special case where h' is uniform

Histogram matching

- To find a transformation that matches the output histogram to a target histogram we use the cumulative histograms, which immediately show for each value how to transform it

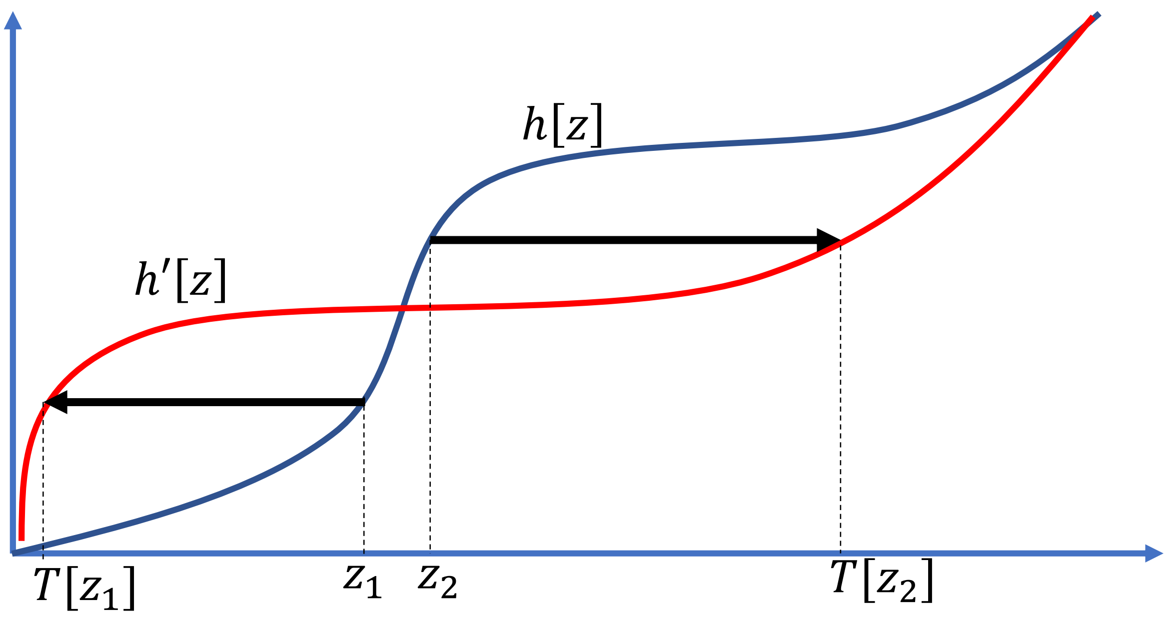


Image arithmetic

- While histogram operations work pixel-by-pixel on only one image, we can also consider operations C that work pixel by pixel on two images I_1 and I_2 resulting in a new image

$$I'[x, y] = C[I_1[x, y], I_2[x, y]]$$

- These operations can be for example arithmetic (+,-,x,/) or logical (AND, OR)

Image subtraction

- For example, we can use subtraction or division to detect change

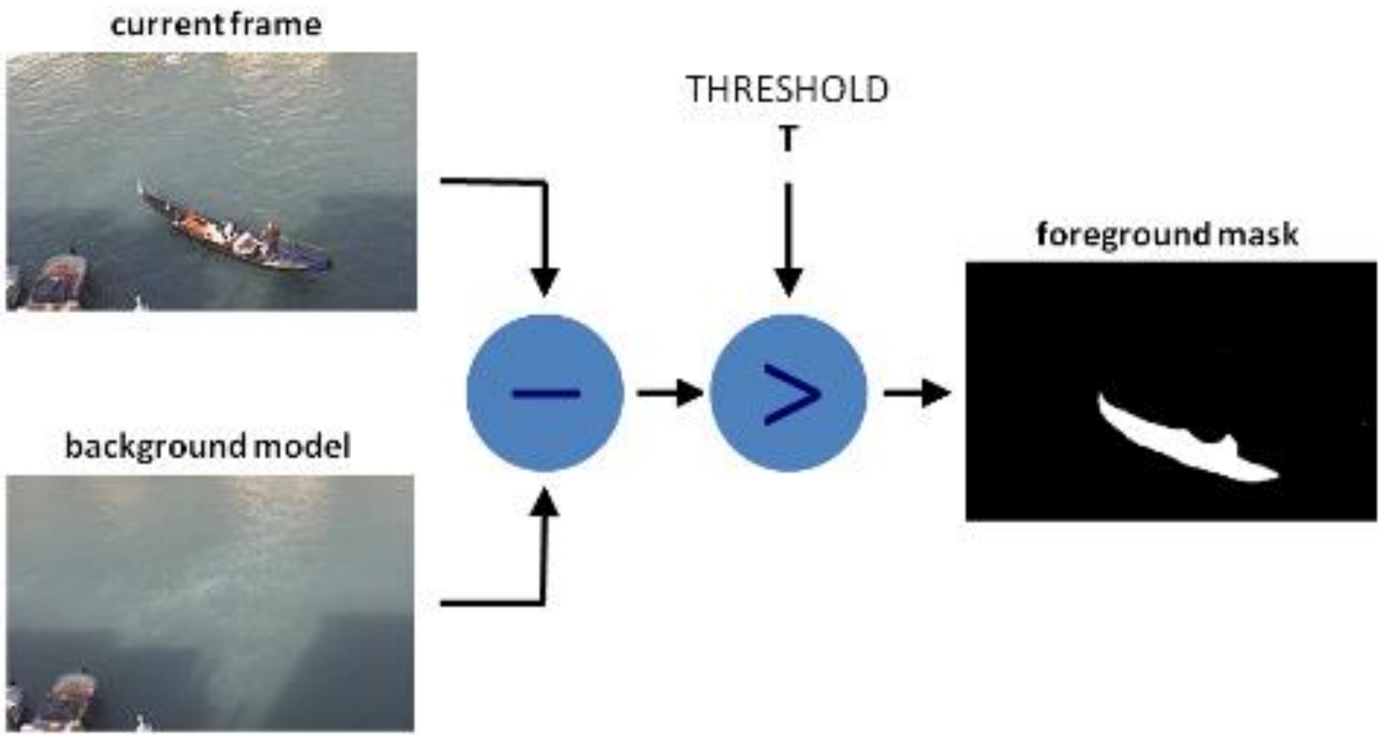
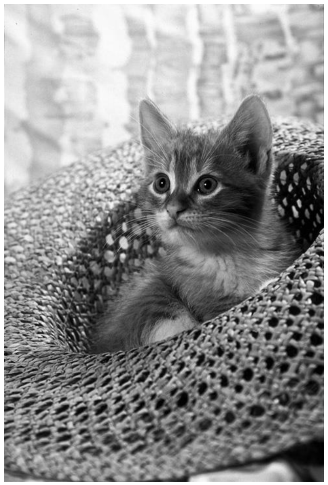


Image blending

- To blend two images together we can use a convex combination

$$I'[x,y] = \lambda I_1[x,y] + (1-\lambda) I_2[x,y]$$

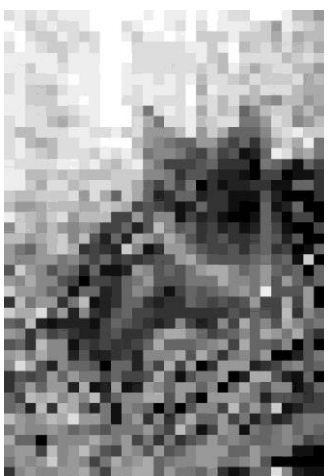
Geometric resolution

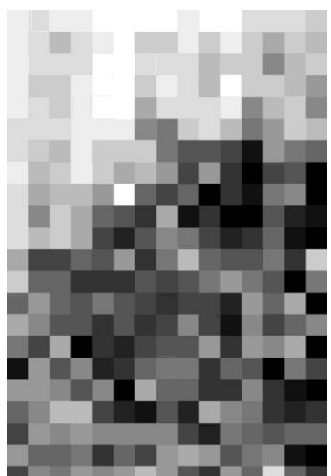

733x490

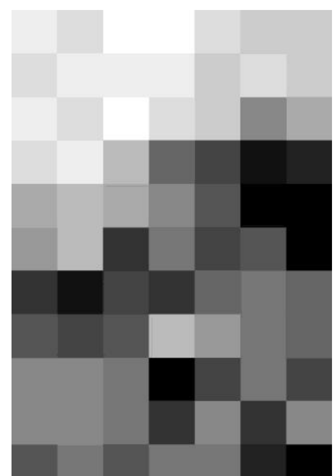

366x245

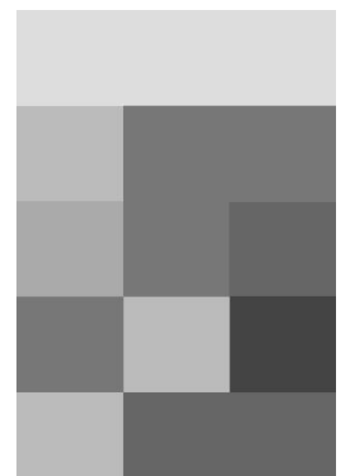

183x122


91x61

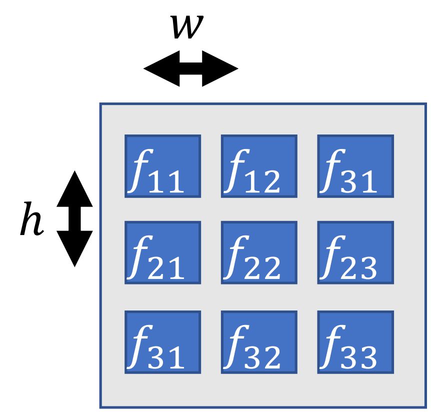

45x30


22x15


11x7


5x3

Spatial sampling



- Let's assume again the image is a continuous function

$$f[x, y] = wh \sum_k \sum_l f_{kl} \delta[x - kw, y - lh]$$

comprising unit impulses sampled on a regular grid of width w and height h

- The Fourier transform of this image is

$$F[u, v] = wh \sum_k \sum_l f_{kl} e^{-i(ukw+vlh)}$$

- This is a periodic function with period $\frac{2\pi}{w}$ and $\frac{2\pi}{h}$, which means all values of $F[u, v]$ for $|u| > \frac{\pi}{w}$ and $v > \frac{\pi}{h}$ are redundant

Nyquist frequency

- Let's now look at an image defined in the frequency domain as follows

$$F[u,v] = \begin{cases} \tilde{F}[u,v] & |u| \leq \frac{\pi}{w} \wedge |v| \leq \frac{\pi}{h} \\ 0 & otherwise \end{cases}$$

- The inverse Fourier transform is

$$f[x,y] = \sum_k \sum_l f_{kl} \frac{\sin\left[\pi\left(\frac{x}{w} - k\right)\right]}{\pi\left(\frac{x}{w} - k\right)} \frac{\sin\left[\pi\left(\frac{y}{h} - l\right)\right]}{\pi\left(\frac{y}{h} - l\right)}$$

- Which evaluates at the sampling points as

$$f[kw, lh] = f_{kl}$$

Nyquist frequency

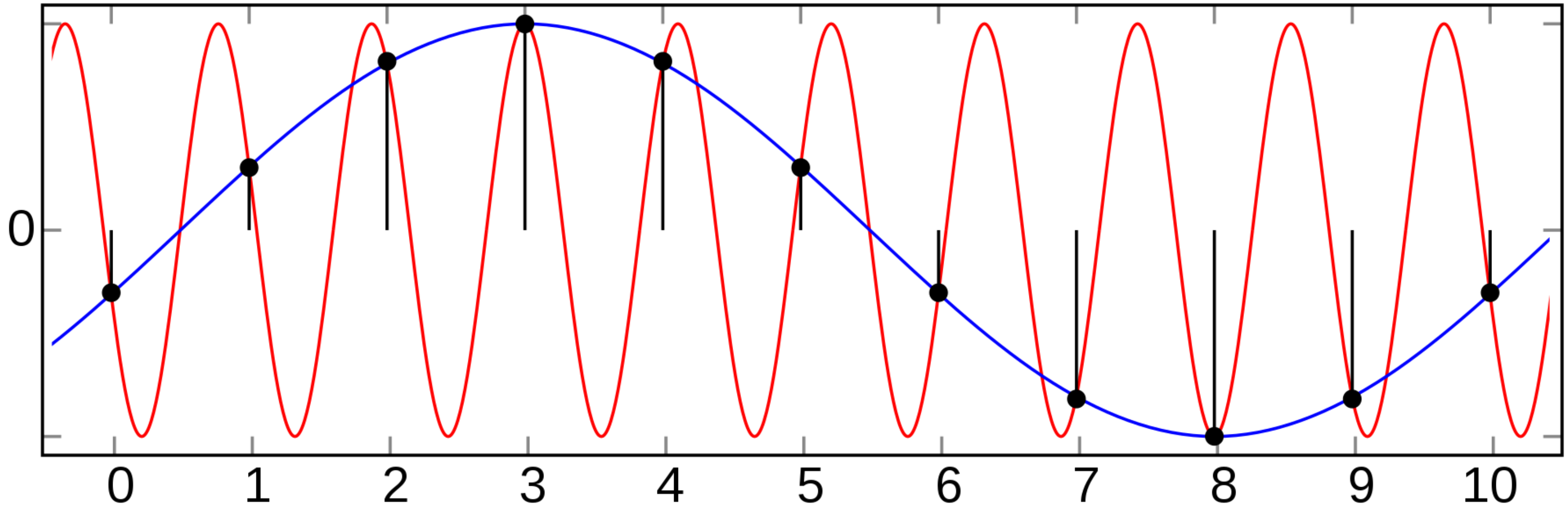
- Therefore, no information is lost when the image is smooth enough, i.e. it sufficiently band-limited
- The **Nyquist frequency**

$$B = \frac{\pi}{\max(w, h)}$$

- is the maximum permissible frequency that can occur in the image so that it is discretely sampled without loss of information
- In other words, the sampling interval has to be less than $\frac{\lambda}{2}$, where λ is the wavelength of the highest frequency that can occur
- This can be achieved by sufficiently blurring (or de-focusing) the image before it is sampled on the image plane

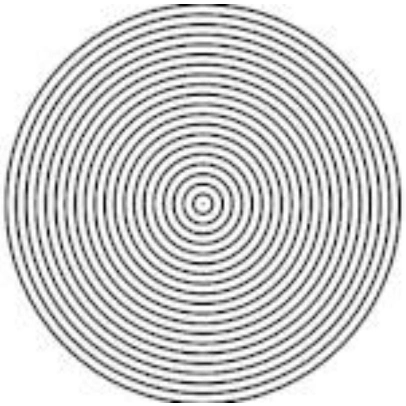
Aliasing

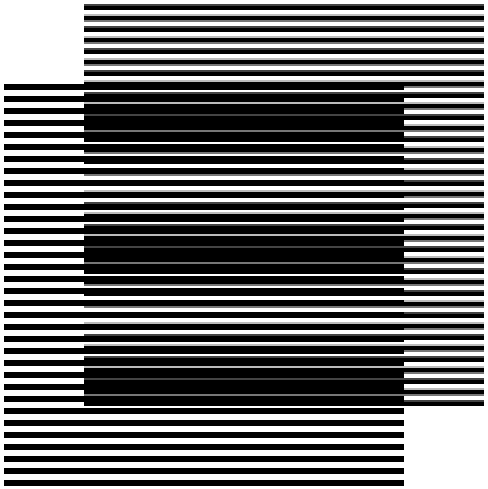
- The loss of information that occurs when sampling a signal of a higher frequency than the permissible Nyquist frequency is called **aliasing**



Aliasing

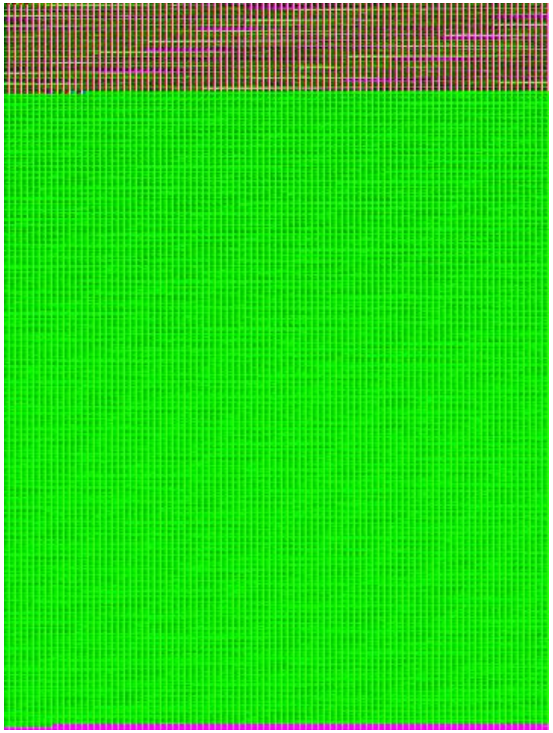
- **Spatial aliasing** leads to artefacts in images, such as lower frequency Moiré patterns on high frequency textures and problems along sharp edges





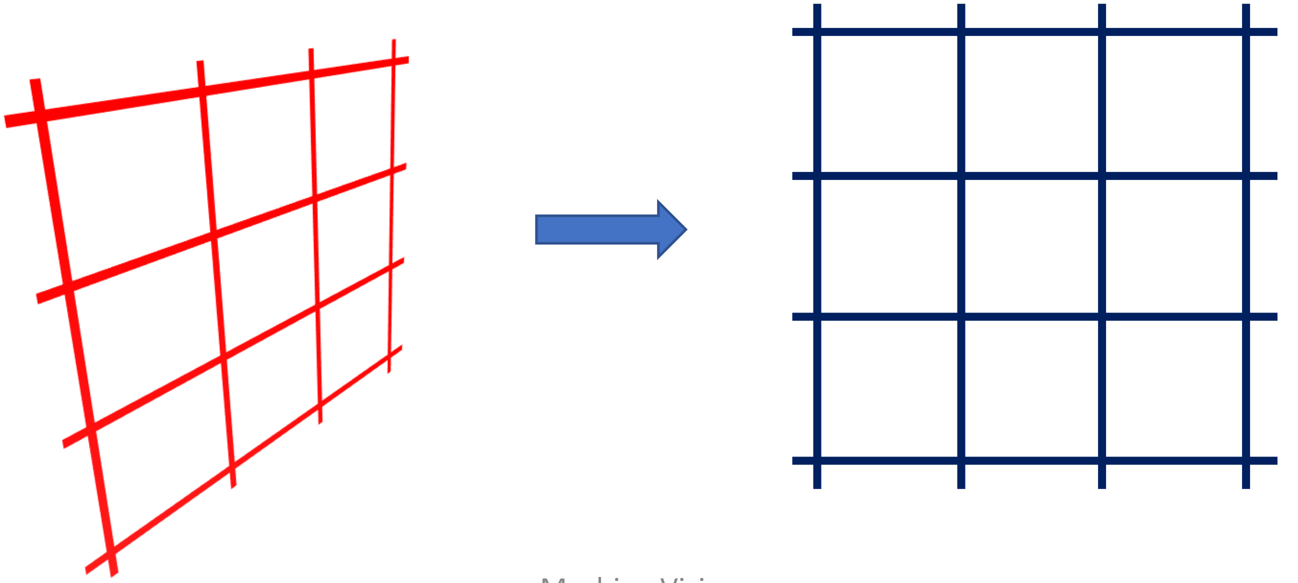
Aliasing

- **Temporal aliasing** occurs in videos when the frame-rate (i.e. the temporal sampling frequency) is too low to capture some high-frequency event



Geometric image transformations

- Geometric image transformations are used for changing the shape of an input image into a desired target shape
- Important geometric transformations are re-sizing, rotation, translation, and perspective transformations
- Because of the discretisation we need to re-sample the original image



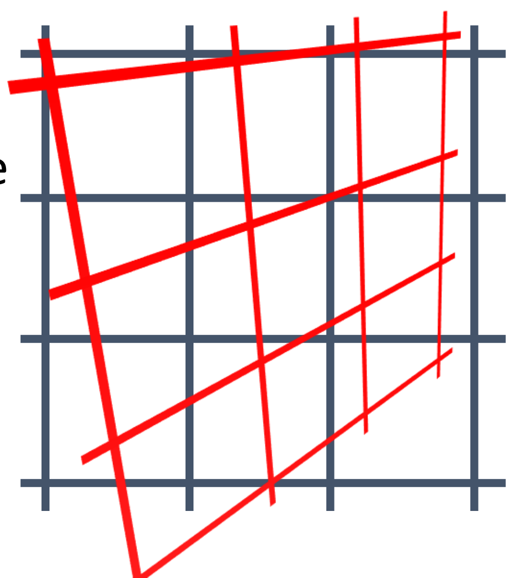
Re-sampling

- For every pixel (x_o, y_o) in the output image we calculate the coordinate in the input image

$$x_i = T_x[x_o, y_o]$$
$$y_i = T_y[x_o, y_o]$$

- These coordinates typically do not co-inside with the discrete raster of the input image

- Therefore we need to **interpolate** the brightness value

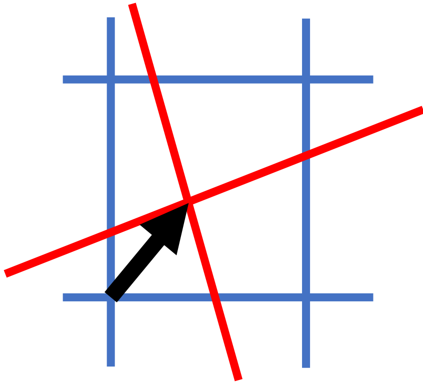


Re-sampling

- To interpolate the brightness of pixel (x_o, y_o) we look at the neighbourhood of (x_i, y_i) in the input image

- **Nearest-neighbour interpolation** uses the brightness value of the nearest raster point in the input image to determine the brightness value in the output image

$$I'[x_o, y_o] = I\left[\left\lfloor x_i + \frac{1}{2} \right\rfloor, \left\lfloor y_i + \frac{1}{2} \right\rfloor\right]$$

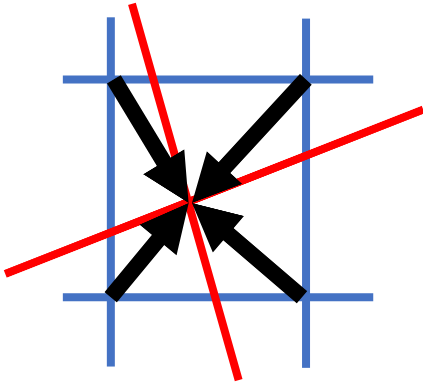


Re-sampling

- To interpolate the brightness of pixel (x_o, y_o) we look at the neighbourhood of (x_i, y_i) in the input image

- **Linear interpolation** uses the average brightness of the four nearest raster points in the input image weighted by the distance to determine the brightness value in the output image

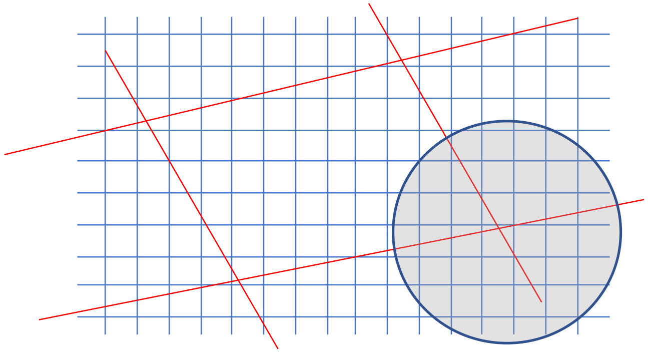
$$
\begin{gathered}
I'[x_o, y_o] = (1 - \lfloor x_i \rfloor)(1 - \lfloor y_i \rfloor) I[\lfloor x_i \rfloor, \lfloor y_i \rfloor] \\
+\lfloor x_i \rfloor (1 - \lfloor y_i \rfloor) I[\lfloor x_i \rfloor + 1, \lfloor y_i \rfloor] \\
+(1 - \lfloor x_i \rfloor) \lfloor y_i \rfloor I[\lfloor x_i \rfloor, \lfloor y_i \rfloor + 1] \\
+\lfloor x_i \rfloor \lfloor y_i \rfloor I[\lfloor x_i \rfloor + 1, \lfloor y_i \rfloor + 1]
\end{gathered}
$$



Re-sampling

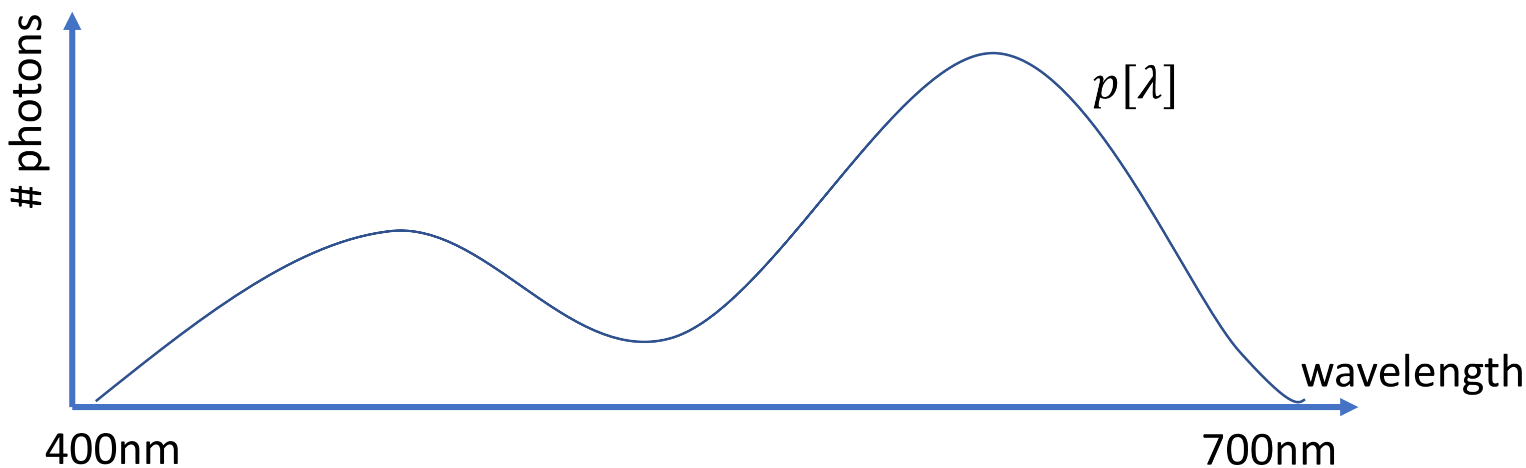
- Linear interpolation provides smoother results
- Even higher-order polynomial interpolations (e.g. bi-cubic) from larger neighbourhoods is being used

- If the input image has a higher resolution than the output image, aliasing is an issue
- This can be resolved by **Anti-Aliasing Filters**
- The input image is low-pass filtered first, i.e. a Gaussian smoothing kernel is applied to remove all high frequencies above the Nyquist frequency of the target resolution



Colour images

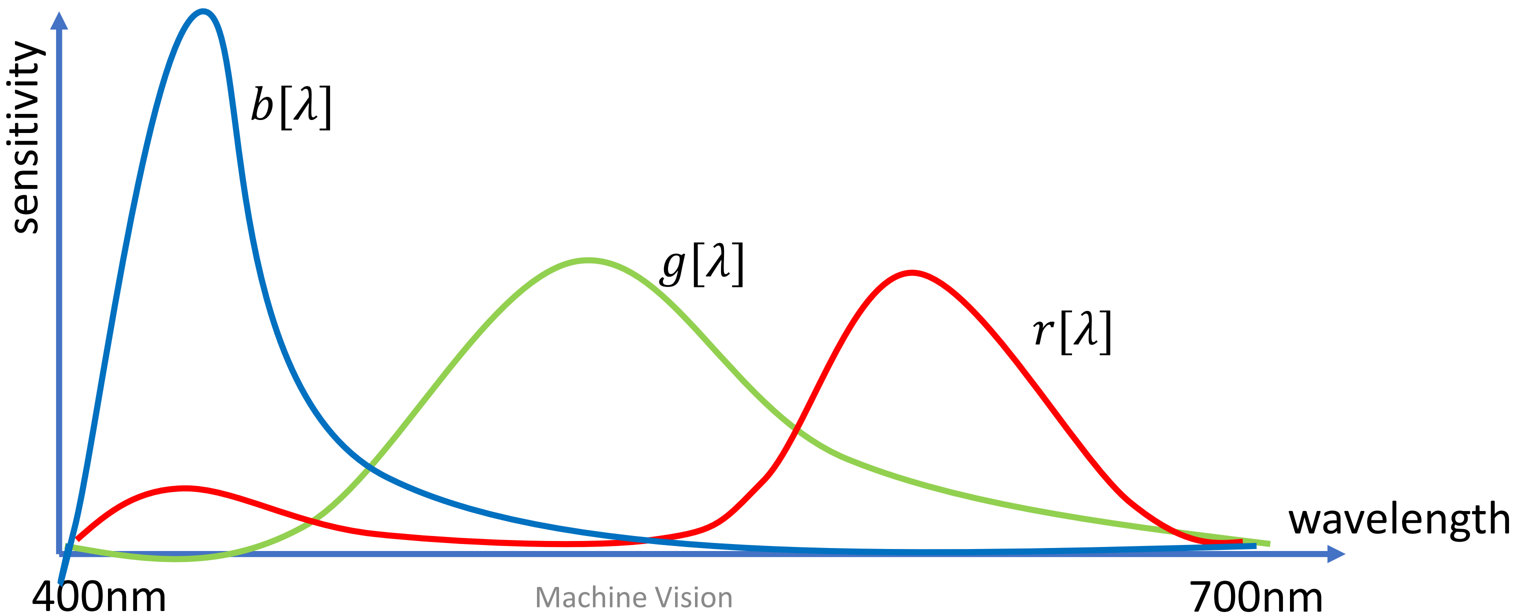
- So far we have only looked at intensity of pixels
- Every photon that is hitting a pixel sensor has a specific energy (or wavelength)
- During exposure every pixel is hit by a lot of these photons
- The wavelength distribution $p[\lambda]$ of these photons depends on the light source and the material properties of the objects off which the light has been reflected before entering the camera
- Most photons emitted by the sun and passing through the earth atmosphere have a wavelength between 400nm and 700nm
- Because the human visual system is adapted to this range it is called “visible light”



Colour images

- To describe the colour spectrum $p[\lambda]$, the human visual system uses only three distinct "sensors", which are sensitive to different parts of the spectrum
- These sensitivity curves $r[\lambda]$, $g[\lambda]$, and $b[\lambda]$ are biologically motivated and fixed and every colour is encoded as a linear combination of these three basis functions
- Only three coefficients (x, y, z) are used to encode the spectrum

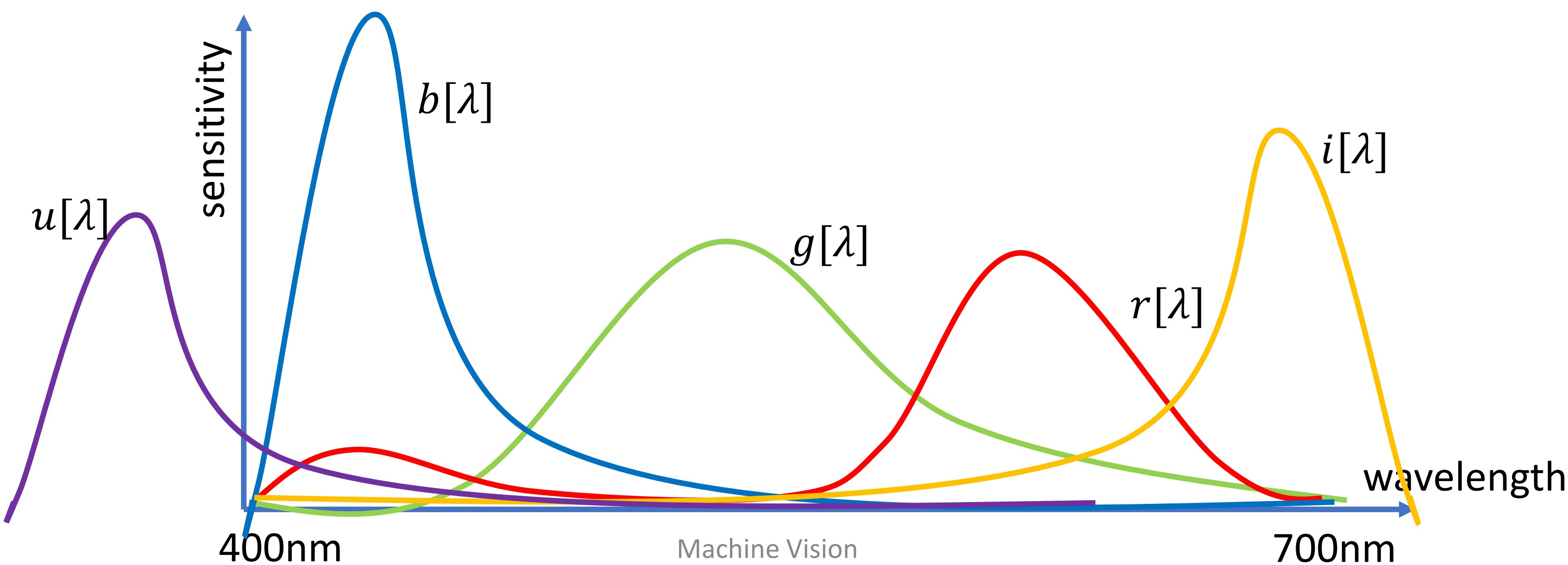
$$p[\lambda] = x\, r[\lambda] + y\, g[\lambda] + z\, b[\lambda]$$



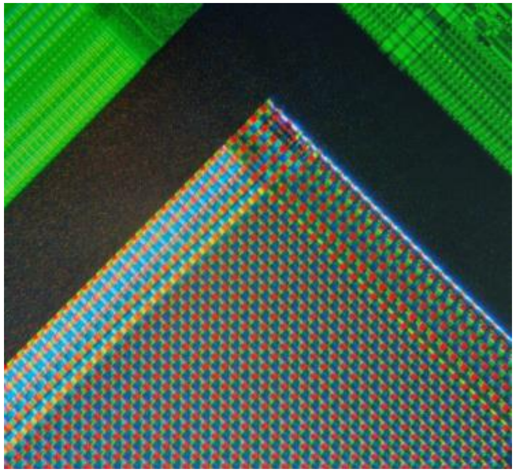
Colour images

- Only wavelength distributions that are possible to express as a linear combination of these red/green/blue basis functions can be represented this way
- This coincides with our own inability to spot the difference between those spectra, but this does not mean a machine vision system has the same restrictions
- Other basis functions can be applied, in particular to cover parts of the spectrum that are invisible to the human eye (e.g. infrared or ultraviolet)
- These **multi-spectral images** are then represented by more than three channels

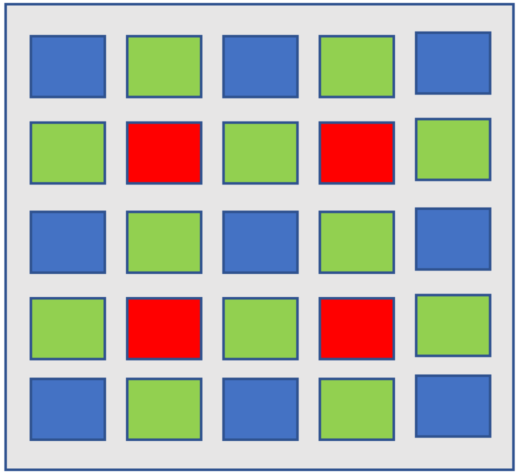
$$p[\lambda] = x\, r[\lambda] + y\, g[\lambda] + z\, b[\lambda] + a\, i[\lambda] + b\, u[\lambda]$$

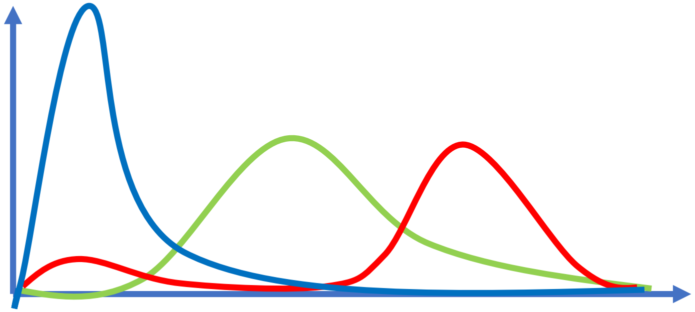


Colour sensing



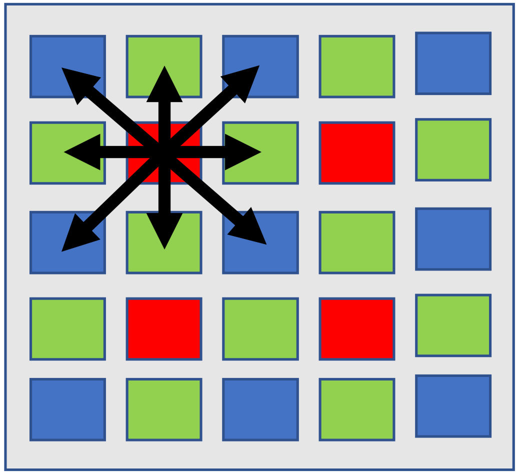
- To measure the individual coefficients of the spectrum individual colour filters are placed in front of the sensors
- Now a pixel is counting only those photons that fall within the range of a specific basis function
- The filters are arranged in specific physiologically motivated patterns, such as the **Bayer pattern** (right) which assigns 50% to sensing green and 25% each to sensing red and blue
- Note, that the full resolution is not used for all colour channels, instead a digital colour image is a mosaic of red, green, and blue components





De-mosaicking

- In order to obtain the RGB values at full resolution we need to interpolate the measurements from neighbouring cells
- The easiest de-mosaicking algorithm interpolates each pixel as the average of all neighbouring pixels



- For a red pixel the colour is calculated as

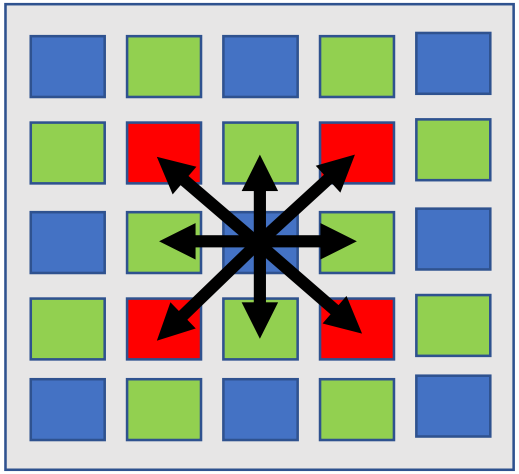
$$R = I[x, y]$$

$$G = \frac{1}{4}(I[x+1, y] + I[x-1, y] + I[x, y+1] + I[x, y-1])$$

$$B = \frac{1}{4}(I[x+1, y+1] + I[x+1, y-1] + I[x-1, y+1] + I[x-1, y-1])$$

De-mosaicking

- In order to obtain the RGB values at full resolution we need to interpolate the measurements from neighbouring cells
- The easiest de-mosaicking algorithm interpolates each pixel as the average of all neighbouring pixels



- For a blue pixel the colour is calculated as

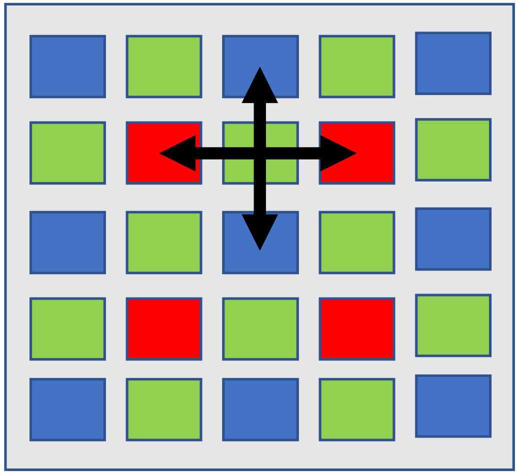
$$B = I[x, y]$$

$$G = \frac{1}{4}(I[x+1, y] + I[x-1, y] + I[x, y+1] + I[x, y-1])$$

$$R = \frac{1}{4}(I[x+1, y+1] + I[x+1, y-1] + I[x-1, y+1] + I[x-1, y-1])$$

De-mosaicking

- In order to obtain the RGB values at full resolution we need to interpolate the measurements from neighbouring cells
- The easiest de-mosaicking algorithm interpolates each pixel as the average of all neighbouring pixels



- For a green pixel the colour is calculated as

$$G = I[x, y]$$

$$R = \frac{1}{2}(I[x+1, y] + I[x-1, y])$$

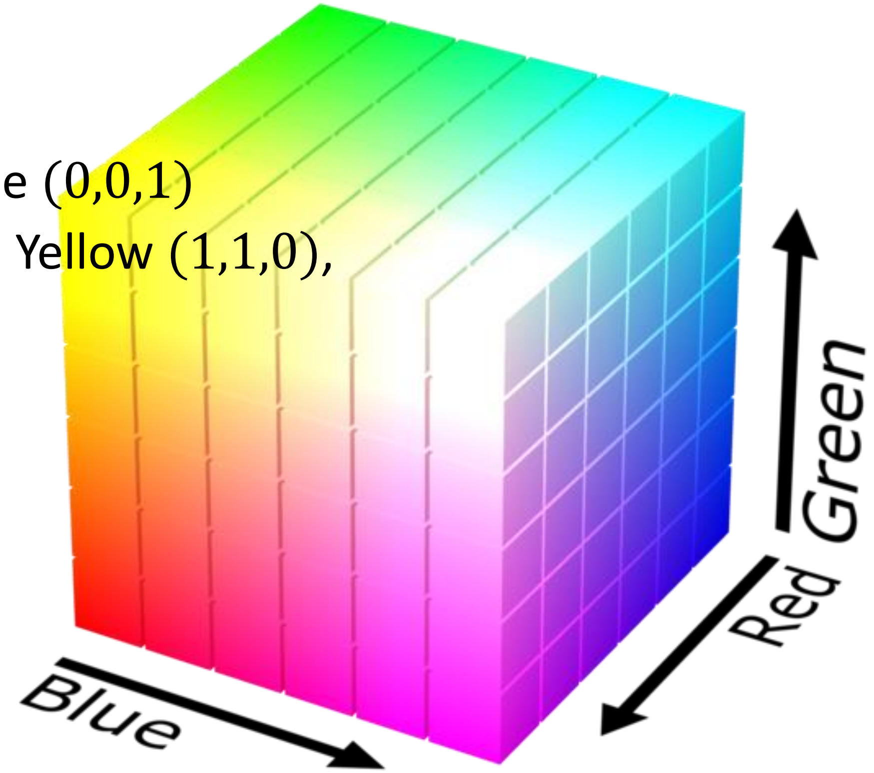
$$B = \frac{1}{2}(I[x, y+1] + I[x, y-1])$$

Aliasing in colour images

- The effective resolution of individual colour sensors is lower, therefore aliasing effects on high-frequency images can cause colour distortions

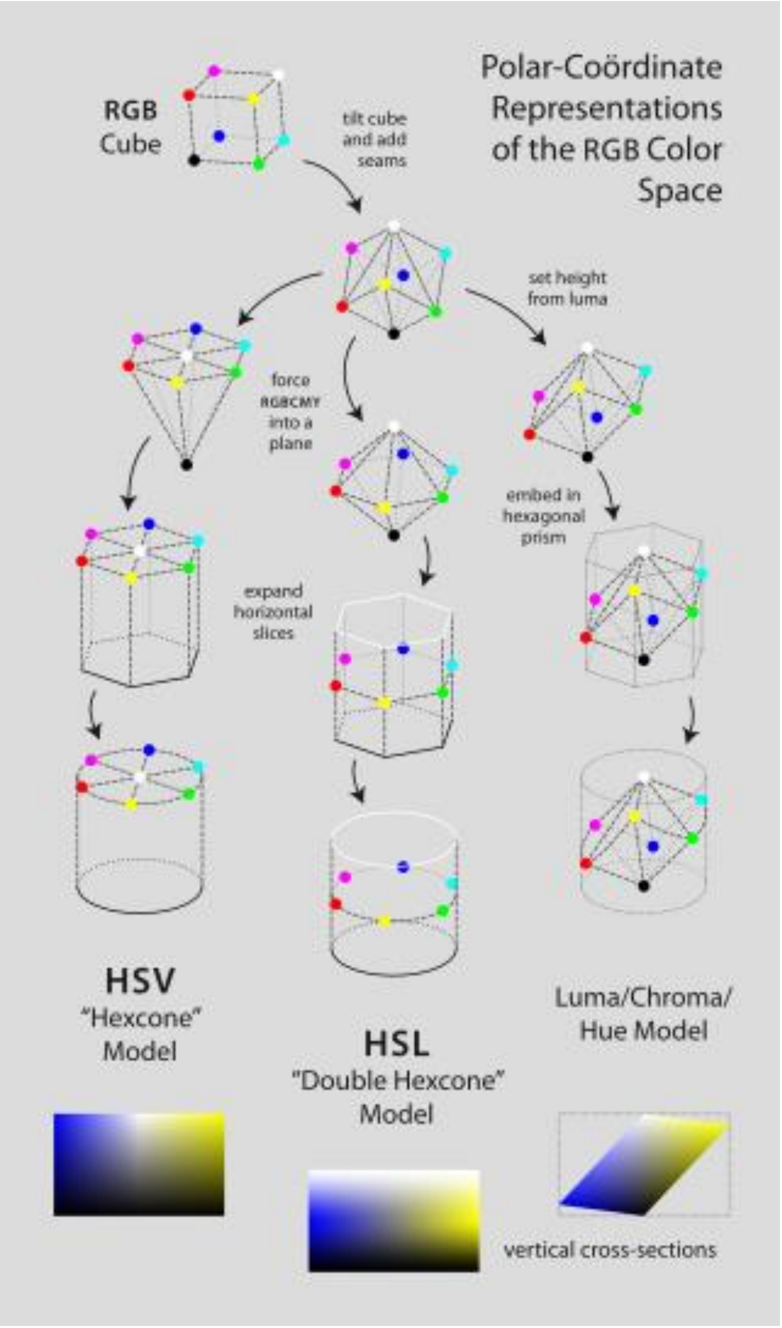
Colour spaces

- The three dimensions red, green, and blue form the RGB colour cube
- Every point in this cube represents a colour
- The eight edges of the cube are
  - Black (0,0,0),
  - Red (1,0,0), Green (0,1,0), Blue (0,0,1)
  - Cyan (0,1,1), Magenta (1,0,1), Yellow (1,1,0),
  - and White (1,1,1)



Colour spaces

- The RGB colour space is not very intuitive
- In particular, determining distance relations in RGB space lacks interpretation
- Therefore other colour models that transform the colour cube into a cylinder and use polar coordinates are useful



HSV colour model

- The HSV colour model describes each point in the 3D colour cube colour with the attributes
  - Hue
  - Saturation
  - Value

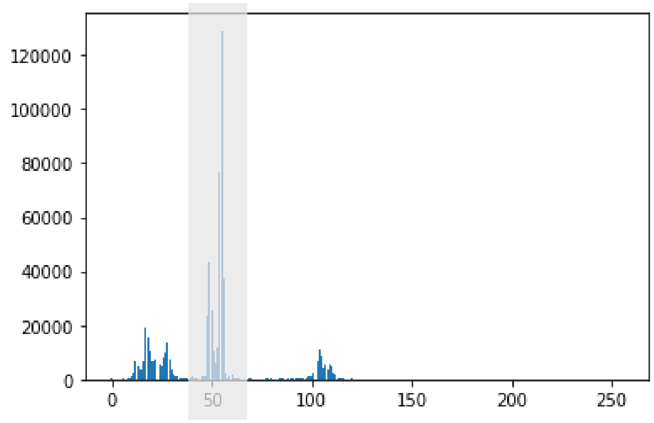
- In particular the Hue channel is interesting, as it resembles what humans perceive as a specific colour and abstracts from brightness and saturation, that we typically ignore when attributing objects

```
hsv_image = cv2.cvtColor(input_img, cv2.COLOR_RGB2HSV)
```

Chroma key

- Thresholding the Hue channel allows to cut out areas of a specific colour

- This is useful for image segmentation and widely used in TV production



Thank you for your attention!