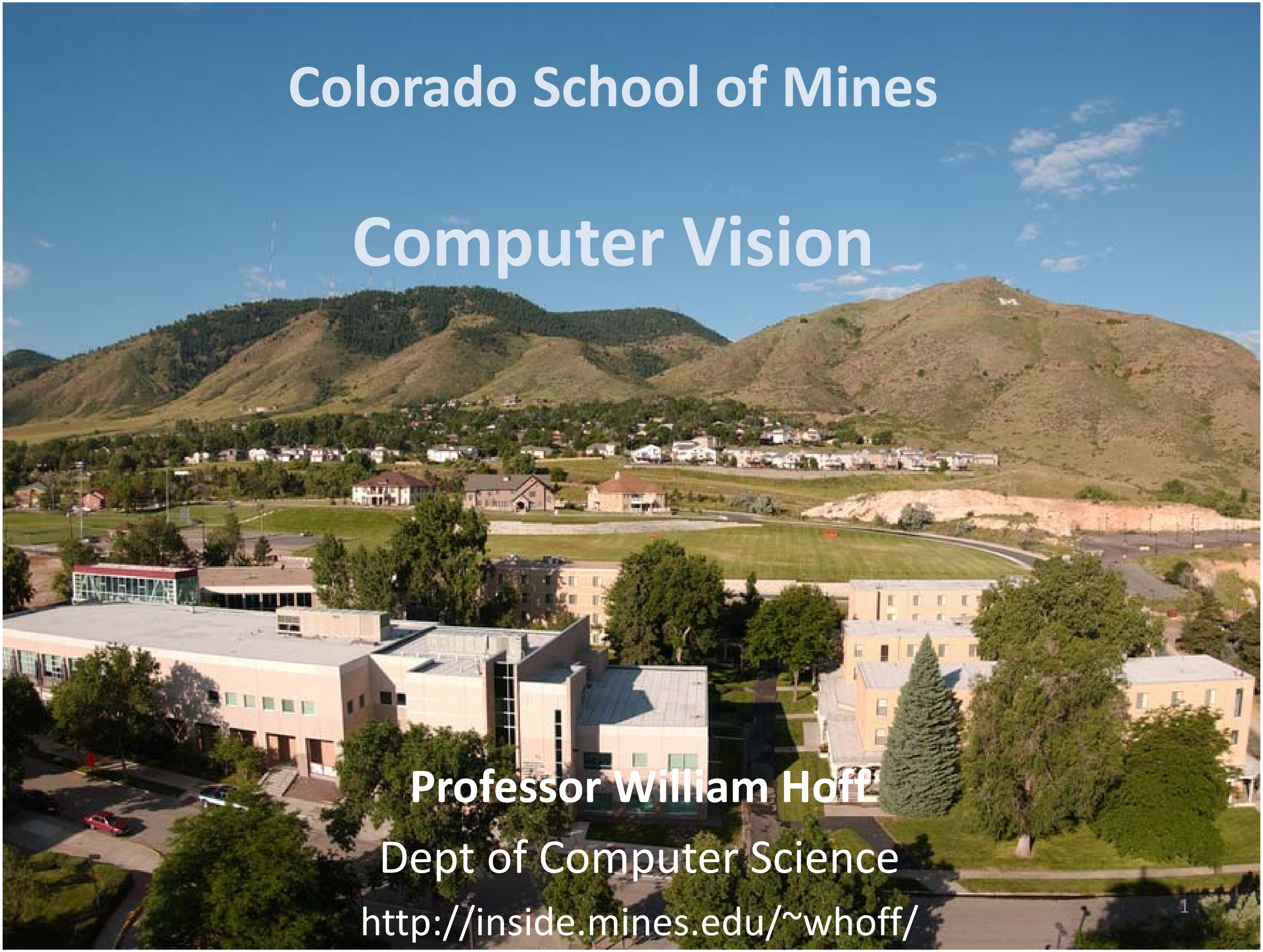


Colorado School of Mines

Computer Vision

Professor William Hoff
Dept of Computer Science
http://inside.mines.edu/~whoff/

Image Filtering

Image Filtering

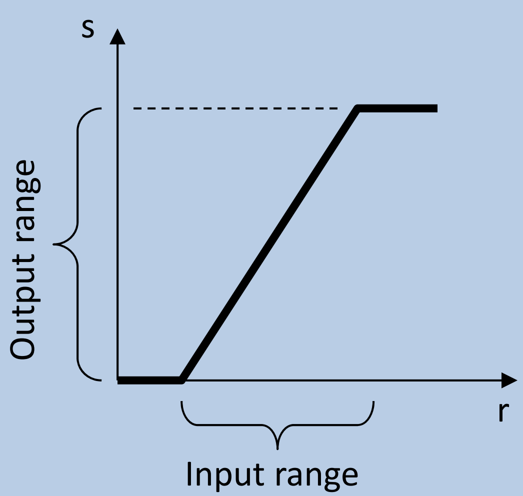
- We will briefly look at methods to reduce noise and enhance images

- Topics
  - Gray level (point transforms)
  - Spatial (neighborhood) transforms

Gray Level Transformations

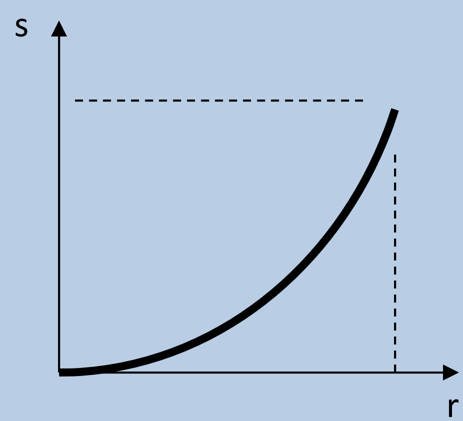
- Point operations
  - *s = f(r)*
  - Map input pixel value *r* to output value *s*
- Examples

Linear scaling



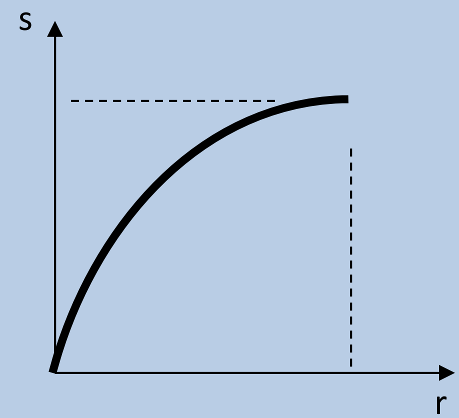
Square ($s=cr^2$)

enhances contrast of high intensities

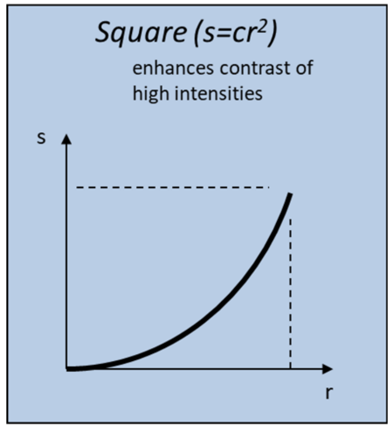


Log (s=c log(1+r))

enhances contrast of low intensities



Example

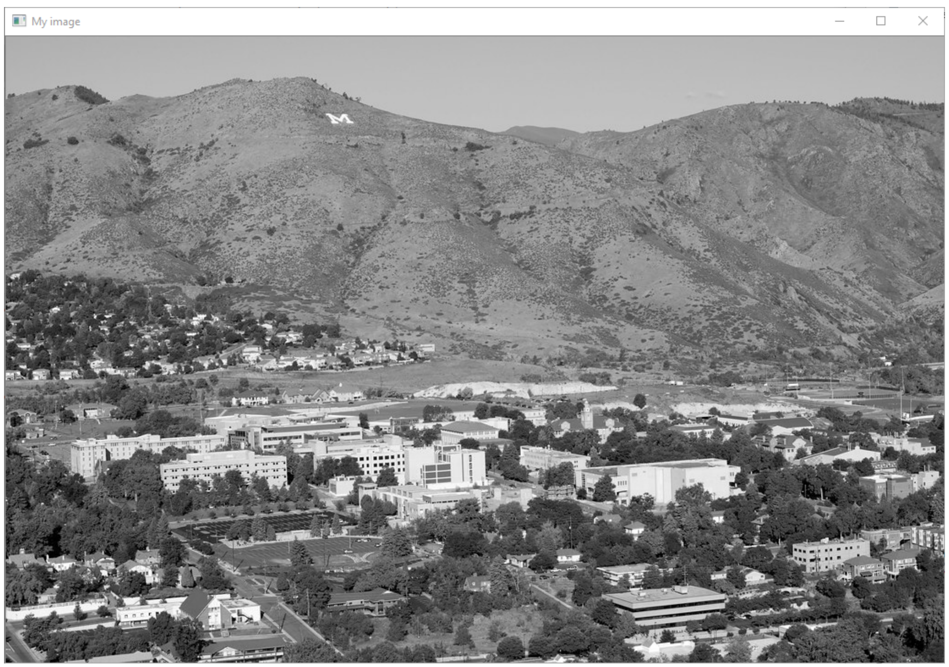


input

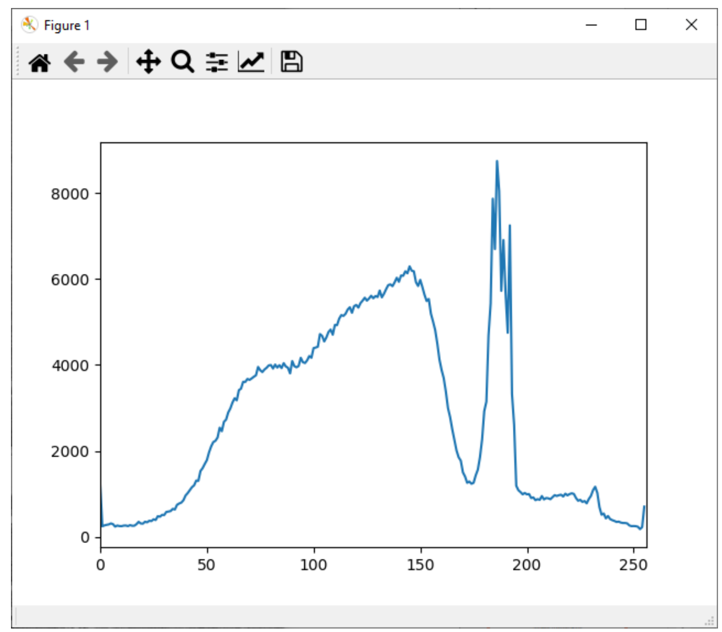
output

Image Histograms

- A plot of the count of the number of pixels with each value
- Since image values are usually in the range 0..255, we will have 256 bins



image



histogram

Computing the histogram

- Use OpenCV's "calcHist"* or Numpy's "histogram"

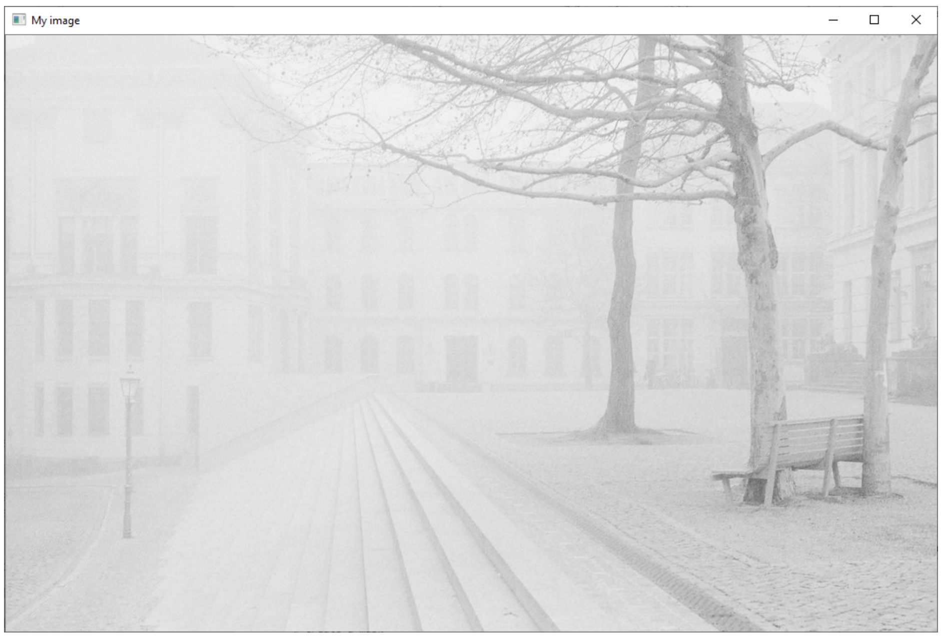
```
import cv2
import numpy as np
import urllib.request
from matplotlib import pyplot as plt

def main():
    # Download an image from the web and save it to a file.
    url = "https://live.staticflickr.com/6033/5893276634_692ed33989_b.jpg"
    urllib.request.urlretrieve(url, "myimage.jpg")

    img = cv2.imread("myimage.jpg", cv2.IMREAD_GRAYSCALE)
    img_height = img.shape[0]
    img_width = img.shape[1]

    win_name = "My image"
    cv2.imshow(win_name, img)
    cv2.waitKey(0)

    hist,bins = np.histogram(img.ravel(),256,[0,256])
    plt.plot(hist)
    plt.xlim([0, 256])
    plt.show()

if __name__ == '__main__':
    main()
```

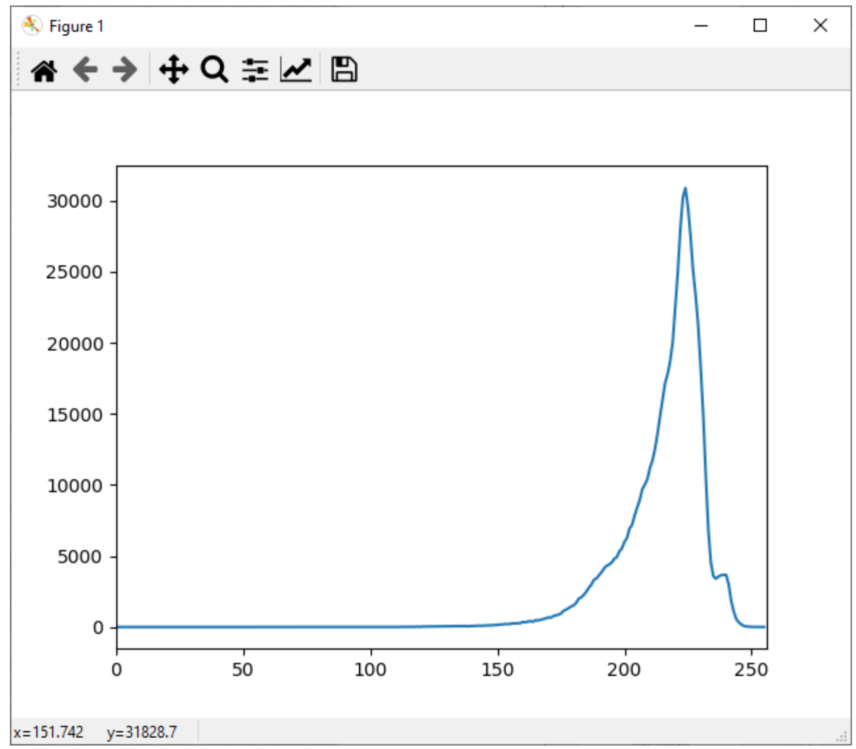
Read in image as grayscale, so we have one histogram instead of 3

*For an example, see https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_histograms/py_histogram_begins/py_histogram_begins.html

Low contrast image

https://farm66.static.flickr.com/65535/48322636587_a5ae136d80_b.jpg

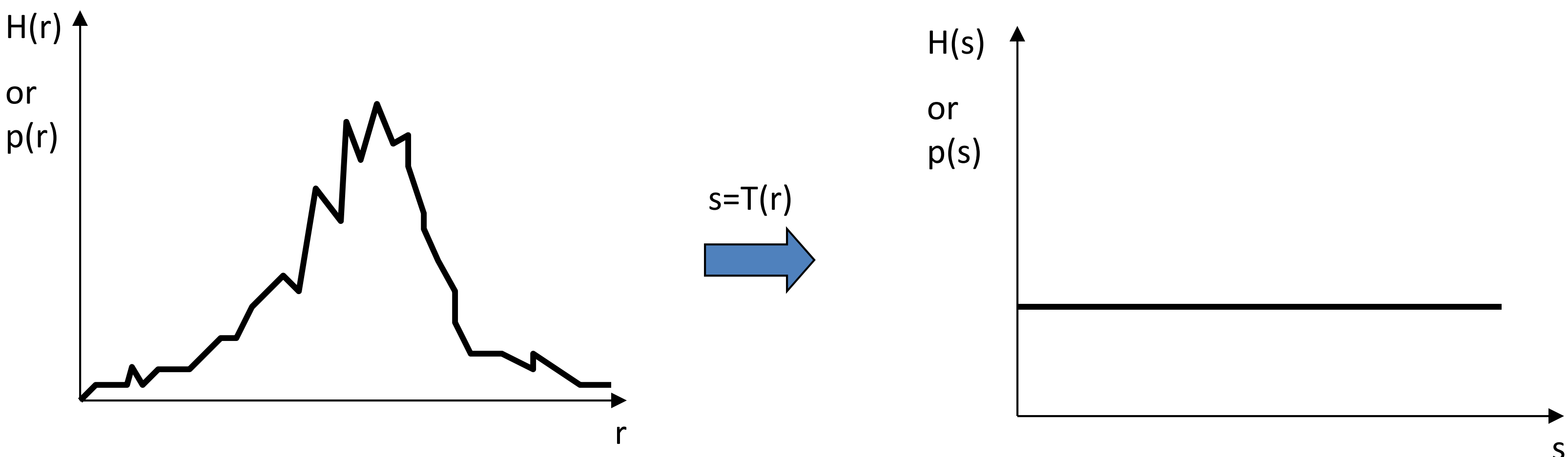




Note the uneven distribution of intensities

Histogram Equalization

- Think of the histogram H(r) as the (scaled) probability distribution of the input image values
- For good contrast, we want the histogram of the output image to be flat: *p(s)* = const

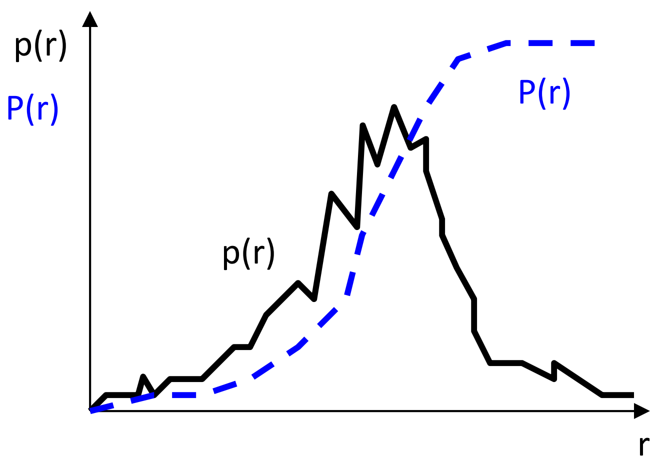


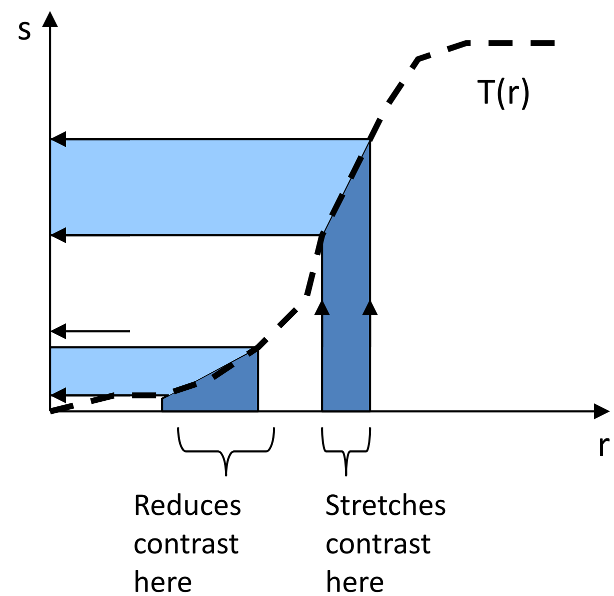
- This stretches contrast where the original image had many pixels with a certain range of gray levels and compresses contrast elsewhere

Mapping Function

- The desired mapping function is just the cumulative probability distribution function of the input image

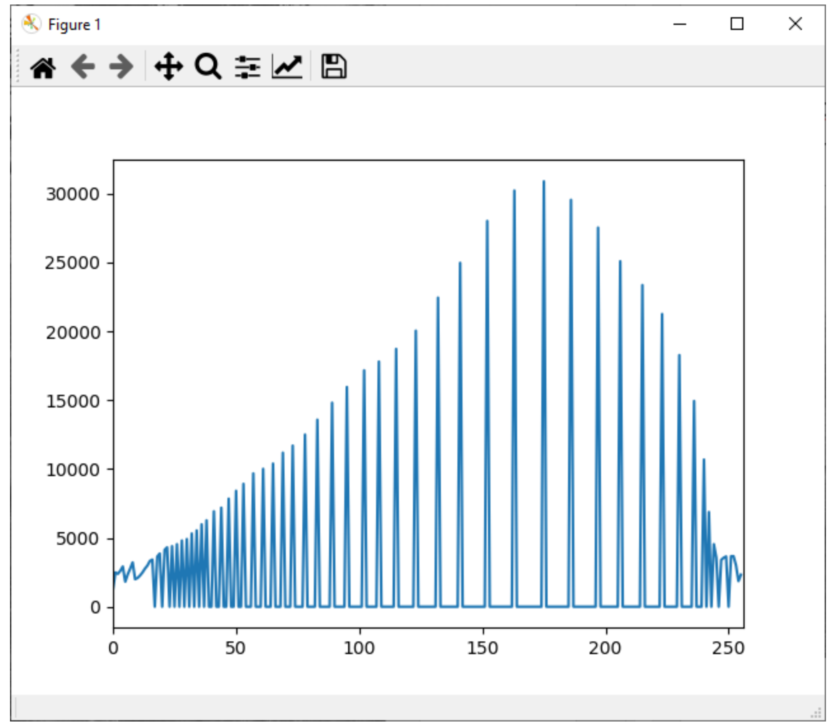
$$P(r) = \int_0^r p_r(w)\,dw$$





Example

- OpenCV histogram equalization
  - `equalizeHist()`

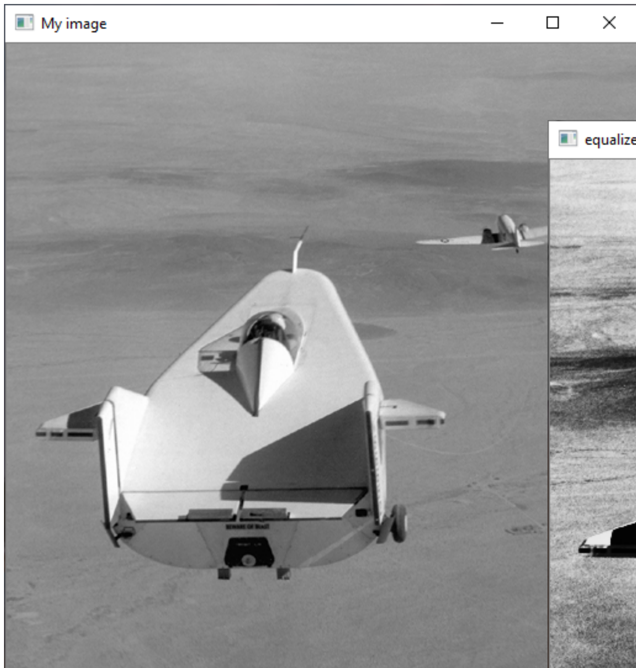


Histogram is flat (or, it would be if the bins were larger)

Adaptive Histogram Equalization

- Instead of using a single mapping function for the whole image, we use a different mapping function for each local neighborhood
- OpenCV

```
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
adapt_img = clahe.apply(img)
```



Input image “liftingbody.png”

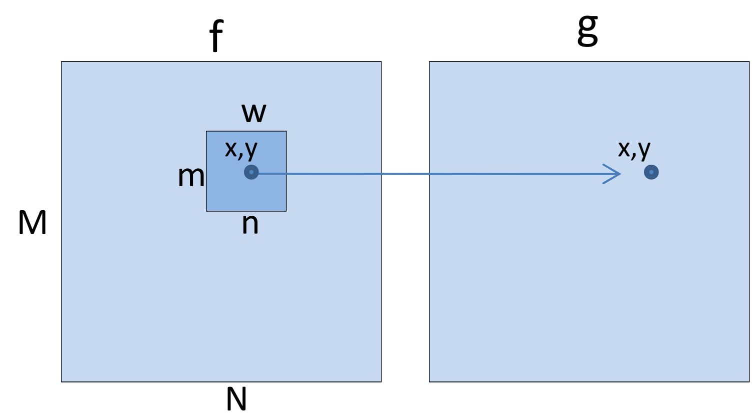
https://www.bogotobogo.com/Matlab/images/MATLAB_DEMO_IMAGES/liftingbody.png

Histogram equalization

Adaptive histogram equalization

Spatial Filtering

- Filter or mask w, size m x n
- Apply to image f, size M x N
- Sum of products of mask coeffs with corresponding pixels under mask
- Slide mask over image, apply at each point
- Also called "cross-correlation"

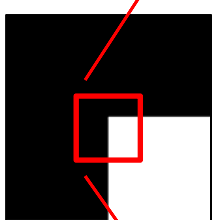


$$g(x,y) = \sum_{s=-m/2}^{m/2} \sum_{t=-n/2}^{n/2} w(s,t)\ f(x+s, y+t)$$
$$= w(x,y) \otimes f(x,y)$$

Example – box (or averaging) filter

$w(x,y)$

1	1	1
1	1	1
1	1	1



$f(x,y)$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	10	10	10	10
0	0	0	10	10	10	10
0	0	0	10	10	10	10
0	0	0	10	10	10	10

Region around a corner

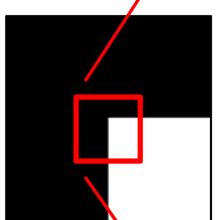
$g(x,y) = w(x,y) \otimes f(x,y)$

blurred corner

$w(x, y)$

1	1	1
1	1	1
1	1	1

result image



$f(x, y)$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	10	10	10	10
0	0	0	10	10	10	10
0	0	0	10	10	10	10
0	0	0	10	10	10	10

Region around a corner

$$g(x, y) = w(x, y) \otimes f(x, y)$$

	0	0	0	0	0	0
	0	10	20	30	**30**	30
	0	**20**	40	60	60	60
	0	30	60	90	90	90
	0	30	60	90	**90**	90
	0	30	60	90	90	90

blurred corner

OpenCV example

```
# Create an averaging filter by hand.
kernel = np.ones((15,15),np.float32)/225

# Apply it.
dst = cv2.filter2D(img, ddepth=cv2.CV_8U, kernel=kernel)
```

- *This is the "depth" of the destination image*
- *CV_8U means 8-bit unsigned*

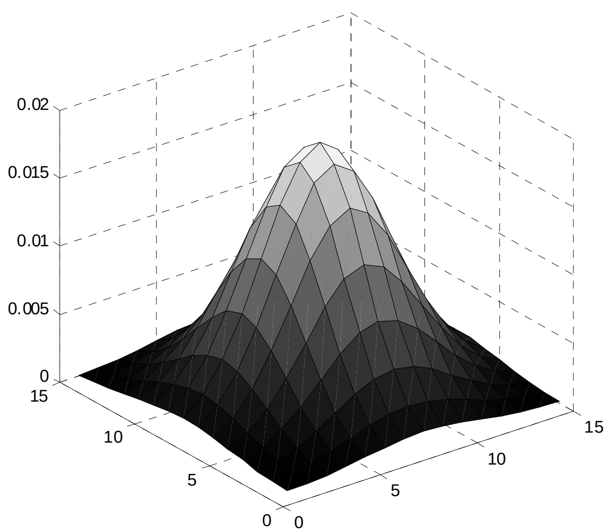
You can also apply a box filter directly like this:

```
cv2.blur(img,(15,15))
```

Gaussian Smoothing Filter

- Gaussian filter usually preferable to box filter
- Attenuates high frequencies better

$$h(x,y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/(2\sigma^2)}$$



15x15 Gaussian, σ=3 (scaled to 1)

0.01	0.02	0.03	0.06	0.08	0.11	0.13	0.14	0.13	0.11	0.08	0.06	0.03	0.02	0.01
0.02	0.03	0.06	0.10	0.15	0.20	0.24	0.25	0.24	0.20	0.15	0.10	0.06	0.03	0.02
0.03	0.06	0.10	0.17	0.25	0.33	0.39	0.41	0.39	0.33	0.25	0.17	0.10	0.06	0.03
0.04	0.08	0.15	0.25	0.37	0.49	0.57	0.61	0.57	0.49	0.37	0.25	0.15	0.08	0.04
0.05	0.11	0.20	0.33	0.49	0.64	0.76	0.80	0.76	0.64	0.49	0.33	0.20	0.11	0.05
0.06	0.13	0.24	0.39	0.57	0.76	0.89	0.95	0.89	0.76	0.57	0.39	0.24	0.13	0.06
0.07	0.14	0.25	0.41	0.61	0.80	0.95	1.00	0.95	0.80	0.61	0.41	0.25	0.14	0.07
0.06	0.13	0.24	0.39	0.57	0.76	0.89	0.95	0.89	0.76	0.57	0.39	0.24	0.13	0.06
0.05	0.11	0.20	0.33	0.49	0.64	0.76	0.80	0.76	0.64	0.49	0.33	0.20	0.11	0.05
0.04	0.08	0.15	0.25	0.37	0.49	0.57	0.61	0.57	0.49	0.37	0.25	0.15	0.08	0.04
0.03	0.06	0.10	0.17	0.25	0.33	0.39	0.41	0.39	0.33	0.25	0.17	0.10	0.06	0.03
0.02	0.03	0.06	0.10	0.15	0.20	0.24	0.25	0.24	0.20	0.15	0.10	0.06	0.03	0.02
0.01	0.02	0.03	0.06	0.08	0.11	0.13	0.14	0.13	0.11	0.08	0.06	0.03	0.02	0.01

OpenCV: `blur = cv2.GaussianBlur(img,(15,15),sigmaX=3)`

Sharpening Spatial Filters

- First derivative, numerical approximation

$$\frac{\partial f}{\partial x} \approx \frac{f(x+h) - f(x)}{h}$$

-1	+1

- Can also do central difference

-0.5	0	+0.5

$$\frac{\partial f}{\partial x} \approx \frac{f(x+h) - f(x-h)}{2h}$$

Edge Operators for 2D Images

$$\frac{\partial}{\partial x} \quad \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \qquad \frac{\partial}{\partial y} \quad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Sobel operators

- These effectively do a blurring followed by a derivative
  - Note that you would need to scale the results

Example

$w(x, y)$

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

$f(x, y)$

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0

$g(x, y) = w(x, y) \otimes f(x, y)$

Example

$w(x, y)$

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

$f(x, y)$

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0

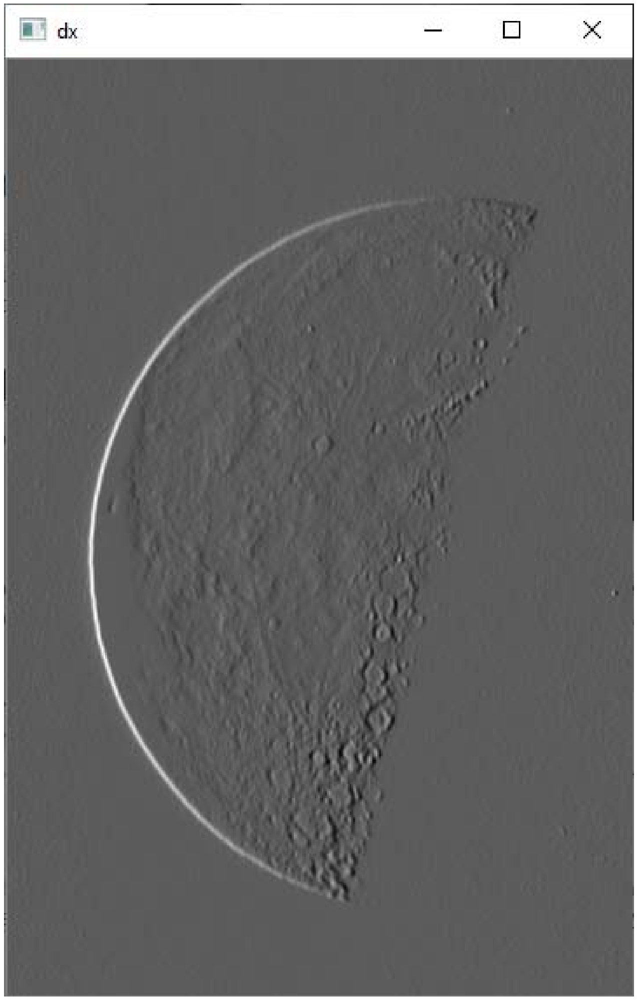
$g(x, y) = w(x, y) \otimes f(x, y)$

0	0	0	0	0	0	0	0
0	1	1	0	0	-1	-1	0
0	3	3	0	0	-3	-3	0
0	4	4	0	0	-4	-4	0
0	4	4	0	0	-4	-4	0
			⋮				
			⋮				

OpenCV Example

- Create Sobel mask in x direction, and apply it

```
sobelx_kernel = np.array([
  [-1, 0, 1],
  [-2, 0, 2],
  [-1, 0, 1],
])
# Apply mask to image. Output image can contain negative values,
# so make the depth of the output image CV_32F (ie., floating).
sobelx = cv2.filter2D(img, ddepth=cv2.CV_32F, kernel=sobelx_kernel)
```



- Scale image to 0..1 so we can see the values

```
# Scale (for display purposes only).
# Min value is set to alpha, max value to beta.
result_display = cv2.normalize(
  src = sobelx, dst = None, alpha = 0, beta = 1,
  norm_type = cv2.NORM_MINMAX, dtype = cv2.CV_32F)
```

- You can also just do

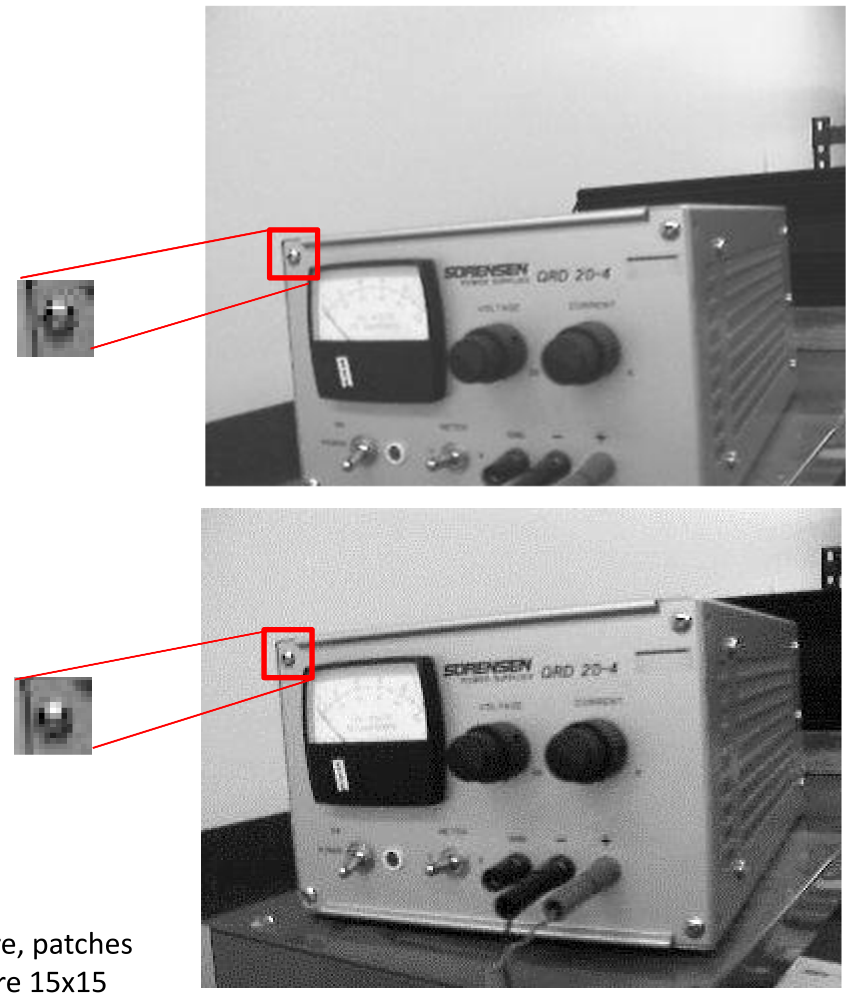
```
# Instead of creating the kernel and doing filter2D, just do this.
sobelx = cv2.Sobel(img, ddepth=cv2.CV_32F, dx=1, dy=0, ksize=3)
```

Template Matching

Image Patch Features

- We often want to find distinctive points in the image
  - Could match these points to a 3D model, for object recognition
  - Or track them from one image to another, for motion or structure estimation

- A point can be described by the appearance of a small image "patch", or template, surrounding the point



Here, patches are 15x15 pixels

Sum of Squared Differences (SSD)

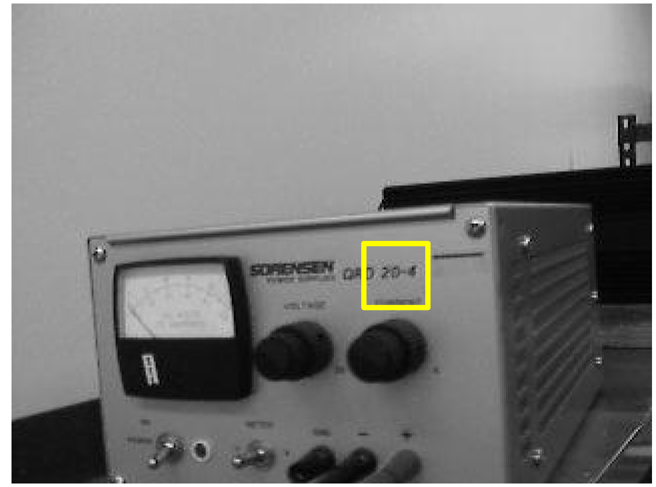
- Say we want to match a patch (template) from image I_0 to image I_1
  - We'll assume that intensities don't change, and there is only translational motion within the image
  - So we expect that

$$I_1(\mathbf{x}_i + \mathbf{u}) = I_0(\mathbf{x}_i)$$

  - In practice we will have noise, so they won't be exactly equal
- But we can search for the displacement **u**=(*x*,*y*) that minimizes the sum of squared differences (SSD):

$$E(\mathbf{u}) = \sum_i w(\mathbf{x}_i)\left[I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)\right]^2$$

w(xi) is an optional weighting function, that weights the center of the patch higher than the periphery



SSD and Cross Correlation

- Expanding the expression for SSD:

$$E(\mathbf{u}) = \sum_i \left[I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i) \right]^2$$

$$= \boxed{\sum_i I_1(\mathbf{x}_i + \mathbf{u})^2} - 2\boxed{\sum_i I_1(\mathbf{x}_i + \mathbf{u}) I_0(\mathbf{x}_i)} + \boxed{\sum_i I_0(\mathbf{x}_i)^2}$$

sum of intensities in I_1 … independent of the template

cross-correlation of I_1 and I_0 … it is high when I_1 matches I_0

sum of intensities in the template I_0, which is a constant

- So, to minimize SSD, we maximize the cross-correlation score

Vector representation of correlation

- Correlation is a sum of products of corresponding terms

$$c(x,y) = \sum_{s=-m/2}^{m/2} \sum_{t=-n/2}^{n/2} w(s,t)\; f(x+s, y+t)$$

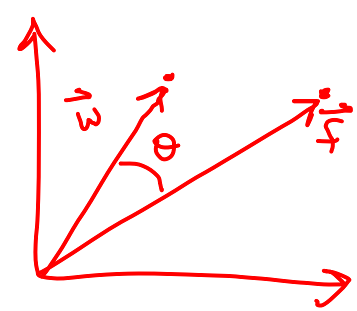
$$= w(x,y) \otimes f(x,y)$$

- We can think of correlation as a dot product of vectors **w** and **f**

$$c = w_1 f_1 + w_2 f_2 + \ldots + w_{mn} f_{mn} = \mathbf{w} \cdot \mathbf{f}$$

- If images w and f are similar, their vectors (in *mn*-dimensional space) are aligned

$$c = |\mathbf{w}||\mathbf{f}| \cos\theta$$



Template Matching (continued)

- Since *f* is not constant everywhere, we need to normalize

$$c = \frac{\mathbf{w} \cdot \mathbf{f}}{|\mathbf{w}||\mathbf{f}|} = \cos\theta$$

- We can get better precision by subtracting off means

$$c(x,y) = \frac{\sum_{s,t}[w(s,t)-\overline{w}][f(x+s,y+t)-\overline{f}]}{\left\{\sum_{s,t}[w(s,t)-\overline{w}]^2 \sum_{s,t}[f(x+s,y+t)-\overline{f}]^2\right\}^{1/2}}$$

This is the normalized cross correlation coefficient

Perfectly opposite

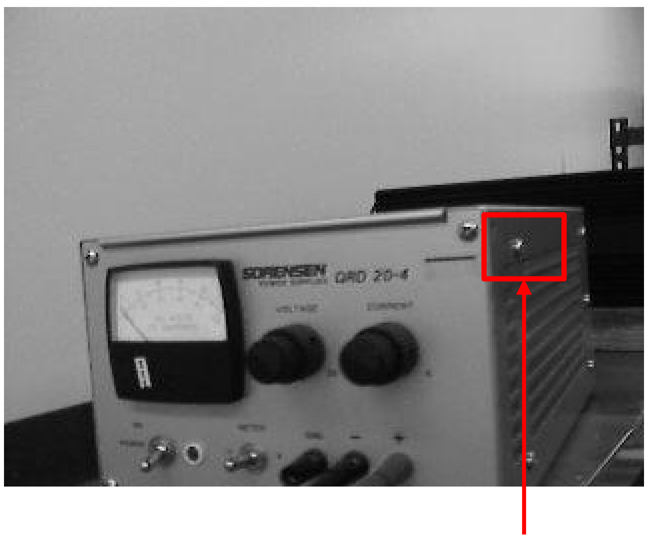
Perfect match

Range: -1.0 … +1.0

OpenCV

```
C = cv2.matchTemplate(img,template,cv2. TM_CCOEFF_NORMED)
```

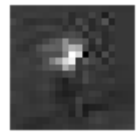
- Does a normalized cross correlation
- Inputs: template image T, and target image I
- Output: correlation scores image C (values range from -1..+1)



Template T

Target image I

Source image I0



Template T

Target image I1

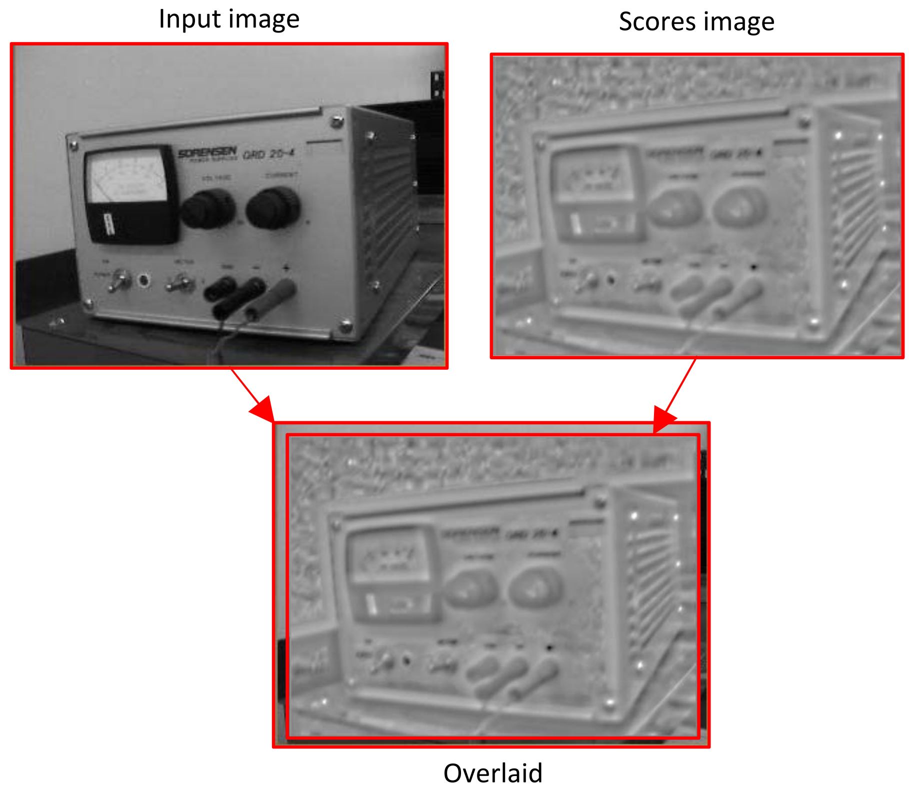
- To find the (single) highest and lowest value:
  ```
  min_val, max_val, min_loc,
  max_loc = cv2.minMaxLoc(scores)
  ```
- To find all matches greater than a threshold:
  ```
  np.where(C >= maxVal)
  ```

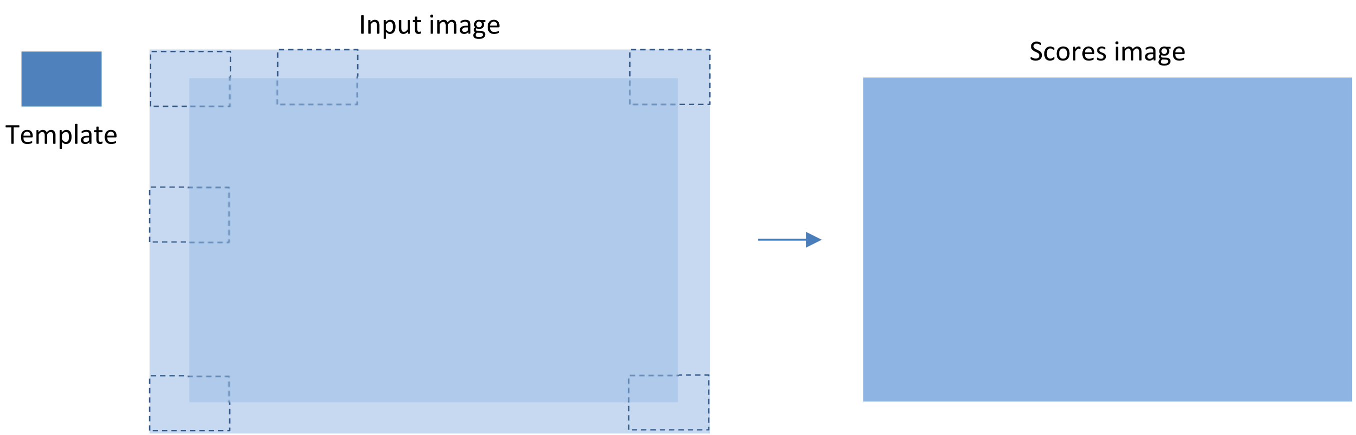
Scores image

Scores image

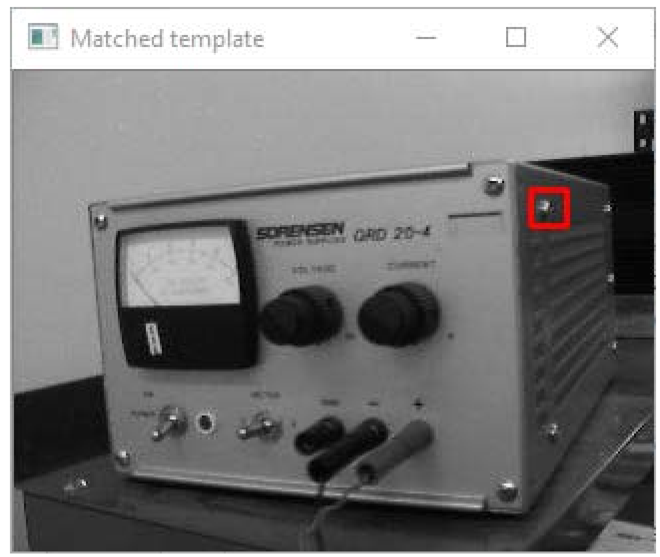
- The scores image is a little smaller than the input image! Why?

- The algorithm only outputs a score value wherever the template fits entirely within the image





- The scores image is smaller than the input image by half the template size, along each side and top and bottom
- You should add half the template size to the detected (x,y) coordinates from the scores image
- Namely, if the template size is (2*M+1)x(2*M+1), add M to the detected (x,y) coordinates



Maximum score = 0.851724 at (x,y) = (267,68)