DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

LAB1: Image Processing

Computer Vision 2023-24
P. Zanuttigh, M. Caligiuri

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CV 2023-24: HW / Projects

1. Intro to Python/ OpenCV

No HW

2. **Histogram proc. and Filtering**
3. Road Line Detection

- Select 1 among 1 and 2
- First «simple» HW: 3 points
- Deliver the jupyter notebook file with code and comments
- On/off mark
- Deadline 6/12

4. Feature Descriptors
5. Deep Learning for CV

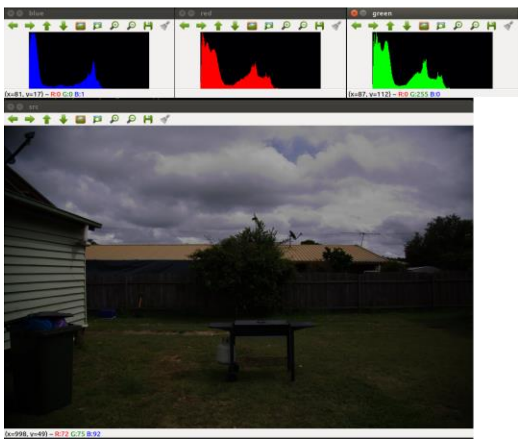
- Select 1 among 3 and 4
- Second «advanced» HW: 6 points
- Deliver code, report and results
- Mark based on solution quality

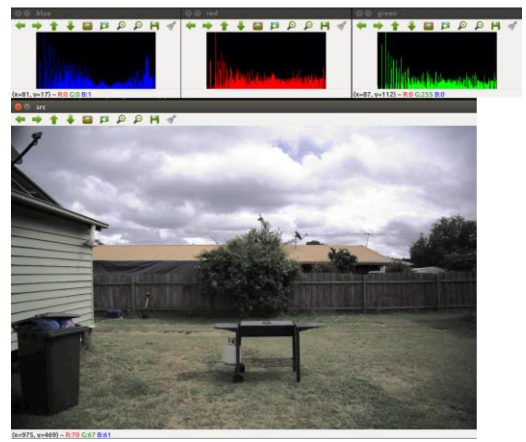
Final Exam (written in classroom): ~23 points

Final mark: **3** + 6 + 23 = max 32 points

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

LAB1: Image Processing





Three main tasks:

1. Compute the histogram of an image and equalize it
2. Perform histogram specification across 2 images
3. Remove the noise using low pass filtering

Recall: Histogram of an Image

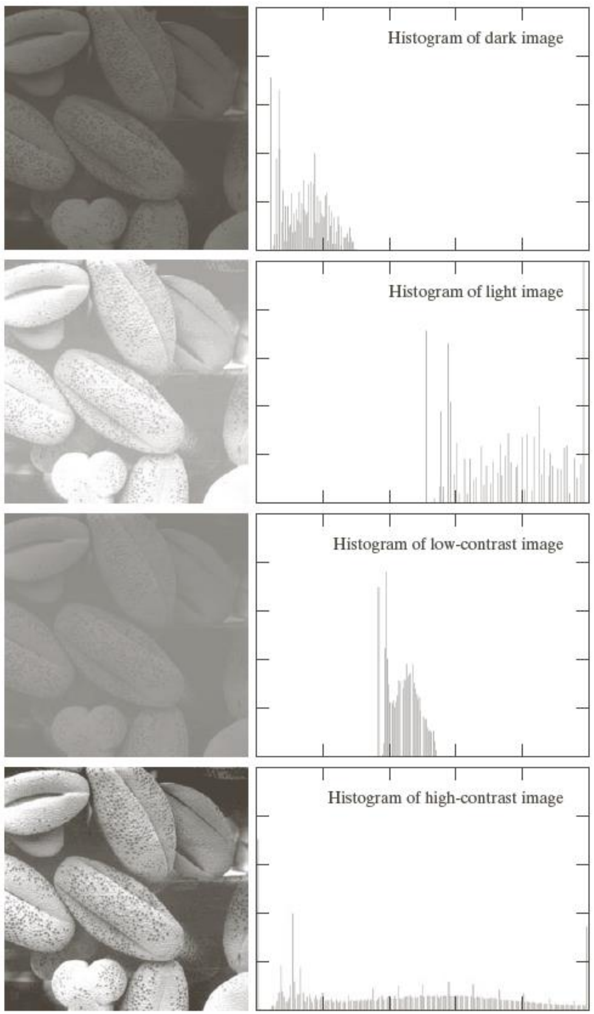


FIGURE 3.16 Four basic image types: dark, light, low contrast, high contrast, and their corresponding histograms.

Normalized Histogram

$$p(r_k) = \frac{h(r_k)}{MN} = \frac{n_k}{MN}$$

$h(r_k) = n_k$ = number of pixels with intensity equal to r_k

Can be viewed as a *probability density*

Usage:

1. Image statistics
2. Compression
3. Segmentation
4. Image enhancement

cv2.calcHist

```
hist= cv2.calcHist (  images,
                      channels,
                      mask,
                      histSize,
                      ranges,
                      uniform = true,
                      accumulate = false
       )
```

- *images*: The source array(s) (i.e., images, can work on multiple images)
- *channels*: The channel (dim) to be measured. In case of grayscale = 0 (1^{st} channel)
- *mask*: A mask to be used on the source array. If not defined it is not used
- *histSize*: The number of bins per each used dimension
- *histRange*: The range of values to be measured per each dimension
- *uniform*: If true the bin sizes are the same (uniform quantization)
- *accumulate*: if false clear the histogram at start (true to accumulate on multiple images)

The function with more than one channel computes an n-dimensional histogram, not 3 histograms for the 3 channels. To work on color images ***split the image into the 3 channels and work on each channel by itself***

Histogram Equalization

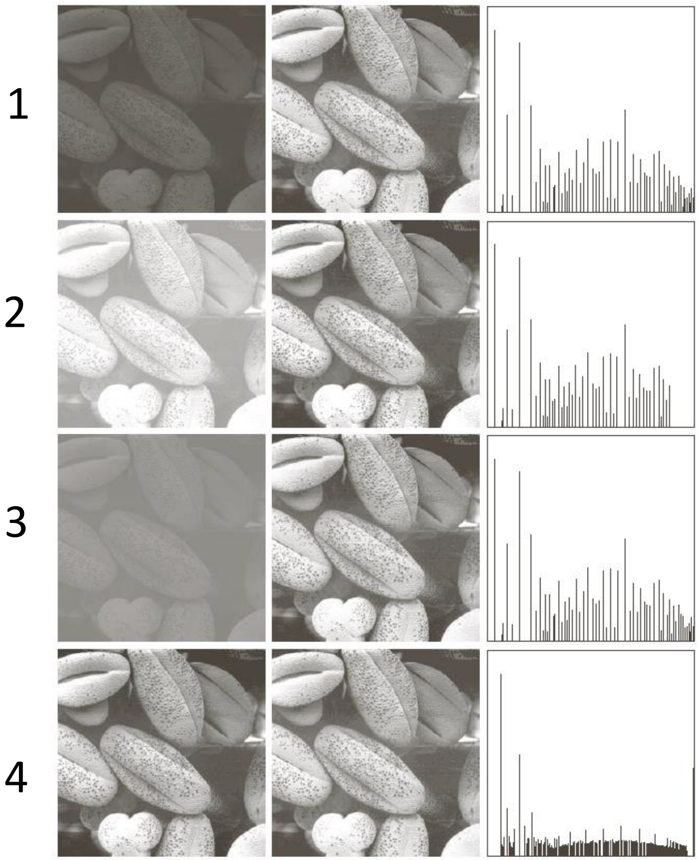


FIGURE 3.20 Left column: images from Fig. 3.16. Center column: corresponding histogram-equalized images. Right column: histograms of the images in the center column.

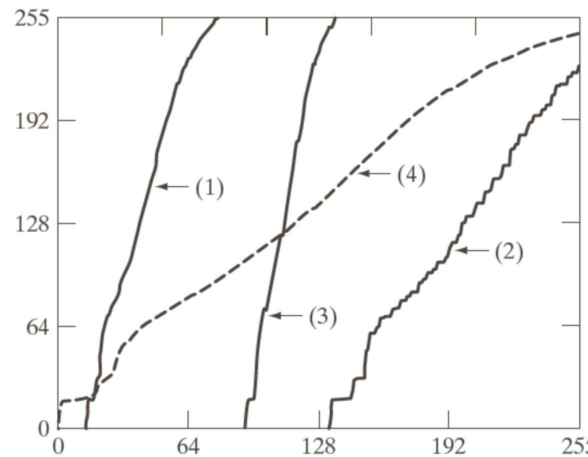


FIGURE 3.21 Transformation functions for histogram equalization. Transformations (1) through (4) were obtained from the histograms of the images (from top to bottom) in the left column of Fig. 3.20 using Eq. (3.3-8).

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j)$$

Best Strategy: change the color space to LAB and equalize only the luminance

```
output_img = cv2.equalizeHist(input_img)
```

Histogram Specification

- The two images have different exposure and colors
- Apply histogram specification to align them in order to be able to fuse them
- Histogram specification is not provided in OpenCV but you have the function in the notebook

Discrete case

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j)$$

$$G(z_q) = (L-1)\sum_{i=0}^{q} p_z(z_i)$$

$$z_q = G^{-1}(s_k)$$

1. Compute $s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j)$
2. Compute $G(z_q) = (L-1)\sum_{i=0}^{q} p_z(z_i)$ and round the values
3. Create table $z_q \leftrightarrow G(z_q)$
4. For each r_k get the corresponding s_k and search for the closest $G(z_q)$
5. Build the equalized image by mapping each r_k in the corresponding z_q

Image Denoising

- Generate a denoised version of the equalized image
- You should try different filters and parameter values (*see table*)
- Write a program that computes the filtered images and shows them
- Pass the parameters using some trackbars
  - cv2.createTrackbar("parameter_name", "window_name", default_value, max_value, callback_func)
  - To pass the image and the parameters to the callback of the trackbar you can create a class containing the image and parameters
  - Trackbars have integer values, use some rescaling e.g. value=sigma*10
- Use a base filter class and create subclasses for the various filters

Filter	***Parameters***
`cv2.medianBlur()`	• Kernel size (square and odd)
`cv2.GaussianBlur()`	• Kernel size (square and odd) • σ (assume $\sigma_X = \sigma_Y$)
`cv2.bilateralFilter()`	• Kernel size (trackbar not required, use a fixed value or use the $6\sigma_s$ rule) • σ_r (range std. dev.) • σ_S (spatial std. dev.)

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Extend the Filter Class

```
class Filter:
    def __init__(self, size: int):
        if size % 2 == 0:
            size += 1
        self.set_filter_size(size)

    def set_filter_size(self, size: int) -> None:
        if size % 2 == 0:
            size += 1
        self.filter_size = size

    def get_filter_size(self):
        return self.filter_size

    def __call__(self, image: np.ndarray) -> None
        raise NotImplementedError


class GaussianFilter(Filter):
    def __init__(self, size: int, sigma_g: float):
        super().__init__(size)
        self.sigma = sigma_g

    def __call__(self, image, sigma_g):
        self.result_image = #add code
# Then implement the other things for the Gaussian
# and the other filters ...
```

Extend the filter class and implement the 3 filters inside the derived classes

Derived classes

- Extra parameters:
  - *Median* : no new parameters, only the win size !
  - *Gaussian*: contains also σ
  - *Bilateral*: σ_r, σ_s
- Redefine *doFilter* with corresponding operations

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Recall: Gaussian Filter

$$H(u,v) = e^{-D^2(u,v)/2D_0^2}$$

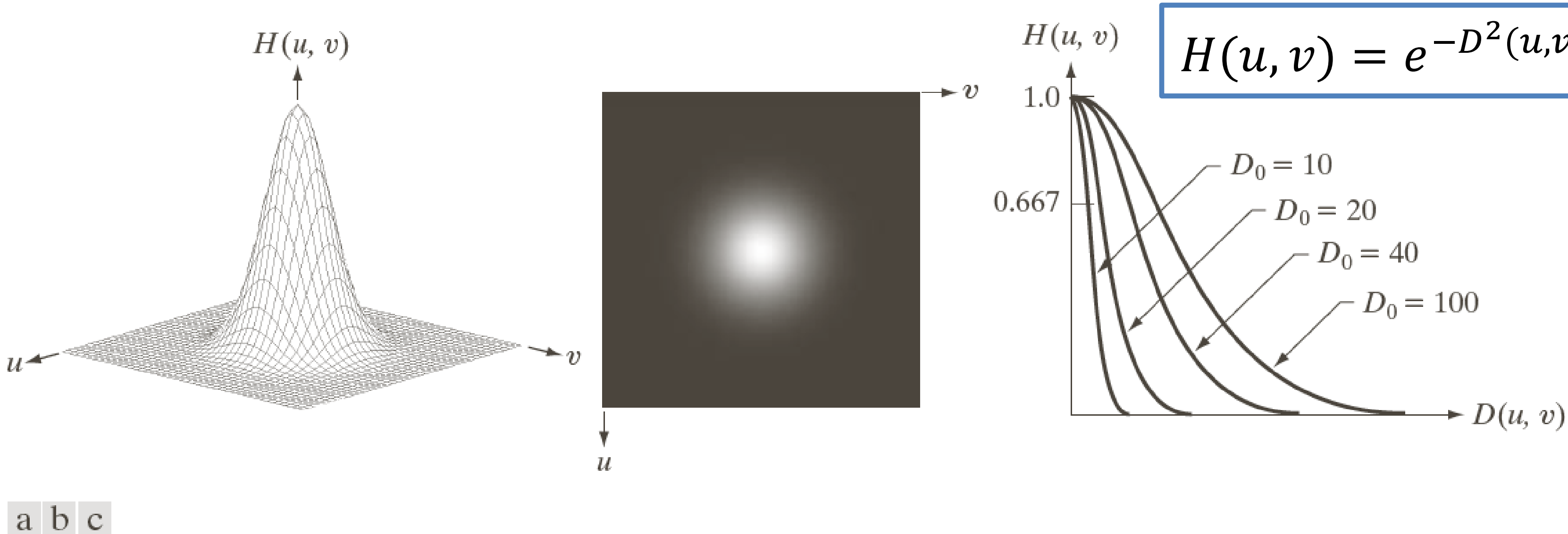


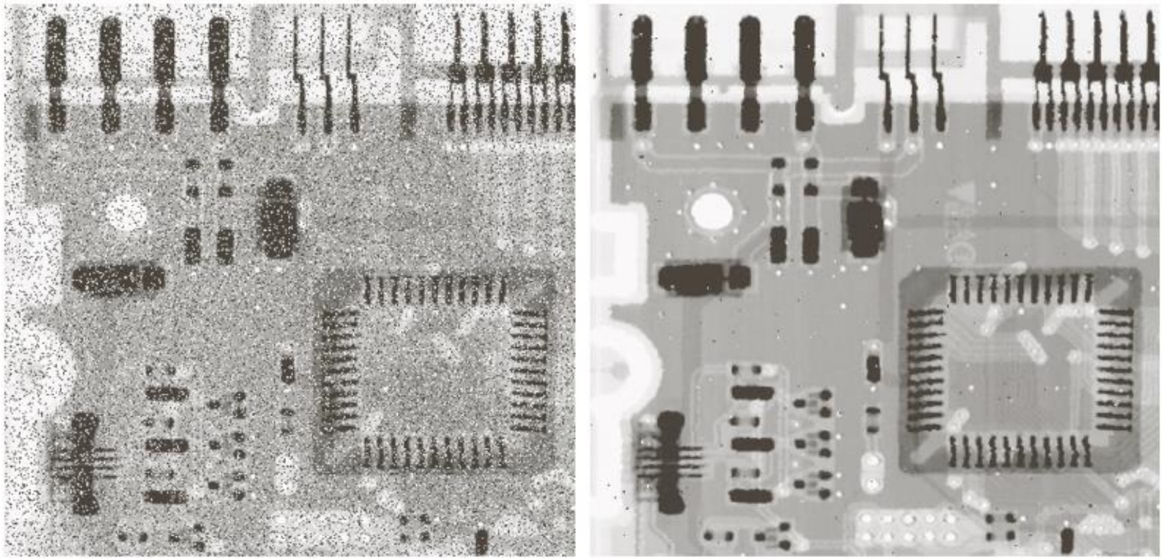
FIGURE 4.47 (a) Perspective plot of a GLPF transfer function. (b) Filter displayed as an image. (c) Filter radial cross sections for various values of D_0.

$D_0 = \sigma_{freq}$ *(standard deviation in the frequency domain)*

- There's no ringing !
- Not very precise frequency selection
- Very commonly used in practice

```
dst = cv2.GaussianBlur(
        src,
        ksize,
        sigmaX,
        sigmaY = 0,  // set to 0 to get σx= σy
        borderType = BORDER_DEFAULT
)
```

Recall: Median Filter



$$g(x,y) = \underset{(s,t)\in R}{median}\{f(s,t)\}$$

3	7	4
8	2	92
10	10	5

Remove Salt and Pepper noise
Only parameter: window size

2	3	4	5	7	8	10	10	92

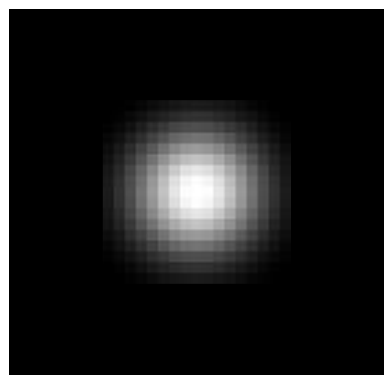
```
dst = cv2.medianBlur (input_img, win_size)
```

Recall: Bilateral Filter

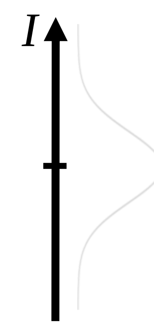
$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in S} G_{\sigma_s}(||\mathbf{p} - \mathbf{q}||)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)\, I_{\mathbf{q}}$$

normalization factor

space weight



range weight



- Weighted average of pixels, but the weights depend on both the spatial distance and the color/intensity difference
- Space σ_s : spatial extent of the kernel
- Range σ_r : “minimum” amplitude of an edge

```
dst = cv2.bilateralFilter (input_img, win_size, sigmaColor, sigmaSpace)
```