AASD 4004
Machine Learning - II

Applied AI Solutions Developer Program

Module 10
Image Transformations
Image Histograms
Image Convolutions

Vejey Gandyer

Agenda

Image Transformations

Image Histograms

Image Convolutions

Image Transformations

What is it?

Image Transformations

Refers to the task of transforming digital images

- Translation
- Rotation
- Resizing

Resizing

When resizing an image, we need to keep in mind the **aspect ratio** (proportional relationship of width and height of the image)

Computing the ratio
Divide the new width by old width of the image

Computing the dimensions
Multiply the old height by ratio r and convert it to an integer

resize( )

Translation

Translation is the **shifting** of an image along the x and y axis.

Using translation, we can shift an image up, down, left, or right, along with any combination of the above!

NOTE: Translating (shifting) an image is given by a NumPy matrix in the form: [[1, 0, shiftX], [0, 1, shiftY]]

You simply need to specify how many pixels you want to shift the image in the X and Y direction.

Translation Matrix

Translation matrix **M** is defined as a **floating point array**

- The first row of the matrix is [1, 0, t_x], where t_x is the number of pixels we will shift the image left or right.
  - Negative values of t_x will shift the image to the left
  - Positive values of t_x will shift the image to the right
- The second row of the matrix is [0, 1, t_y], where t_y is the number of pixels we will shift the image up or down.
  - Negative value of t_y will shift the image up
  - Positive values of t_y will shift the image down

warpAffine( )

- The first argument is the image we wish to shift
- Second argument is our translation matrix M
- Third argument is the dimensions (width and height) of our image

Rotation

Rotation is rotating an image by some angle **θ**

- θ to represent by how many **degrees** to rotate the image

When we rotate an image, we need to specify **around which point** we want to rotate, mostly rotate around the **center** of an image; you can rotate around any arbitrary point.

Steps

1. Find the center of the image
2. Rotation Matrix R
3. Rotate the image

Rotation Matrix

Rotation matrix **R** is computed using getRotationMatrix2D

getRotationMatrix2D function takes three arguments

- the point at which we want to rotate the image around
- θ, the number of degrees we are going to rotate the image
- Scale of the image
    - 1.0 --> No change in dimensions
    - 2.0 --> Double the size
    - 0.5 --> Half the size

warpAffine( )

- The first argument is the image we wish to rotate
- Second argument is our Rotation matrix R
- Third argument is the dimensions (width and height) of our image

Image Histograms

What is it?

Image Histogram

Distribution of pixel intensities of an image *(# pixels Vs Pixel Intensities)*

- Grayscale histogram
- Color histogram

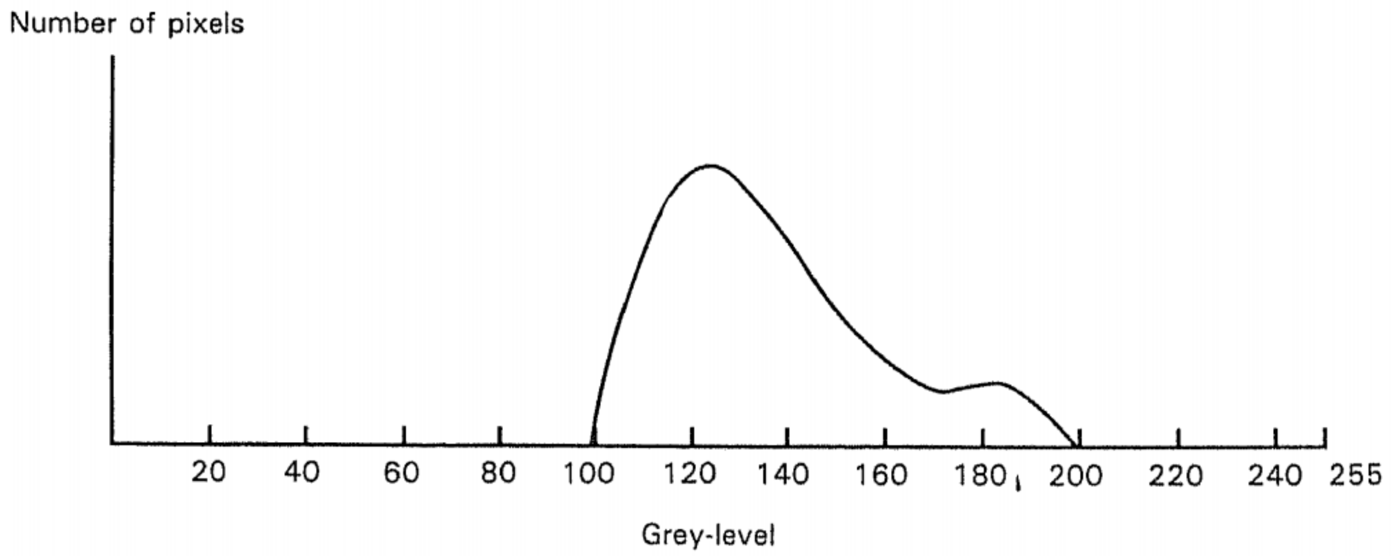


Image Histogram

When plotting the histogram, the X-axis serves as our **bins**

- If we construct a histogram with 256 bins, then we are effectively counting the number of times each pixel value occurs

- If we use only 2 (equally spaced) bins, then we are counting the number of times a pixel is in the range **[0, 128]** or **[128, 255]**

The number of pixels binned to the x-axis value is then plotted on the y-axis

calcHist( )

images: This is the image that we want to compute a histogram for

- Wrap it as a list: [myImage]

channels: This is a list of indexes, where we specify the index of the channel we want to compute a histogram for

- Grayscale image, the list would be [0].
- RGB image,, the channels list would be [0,1,2]

mask:

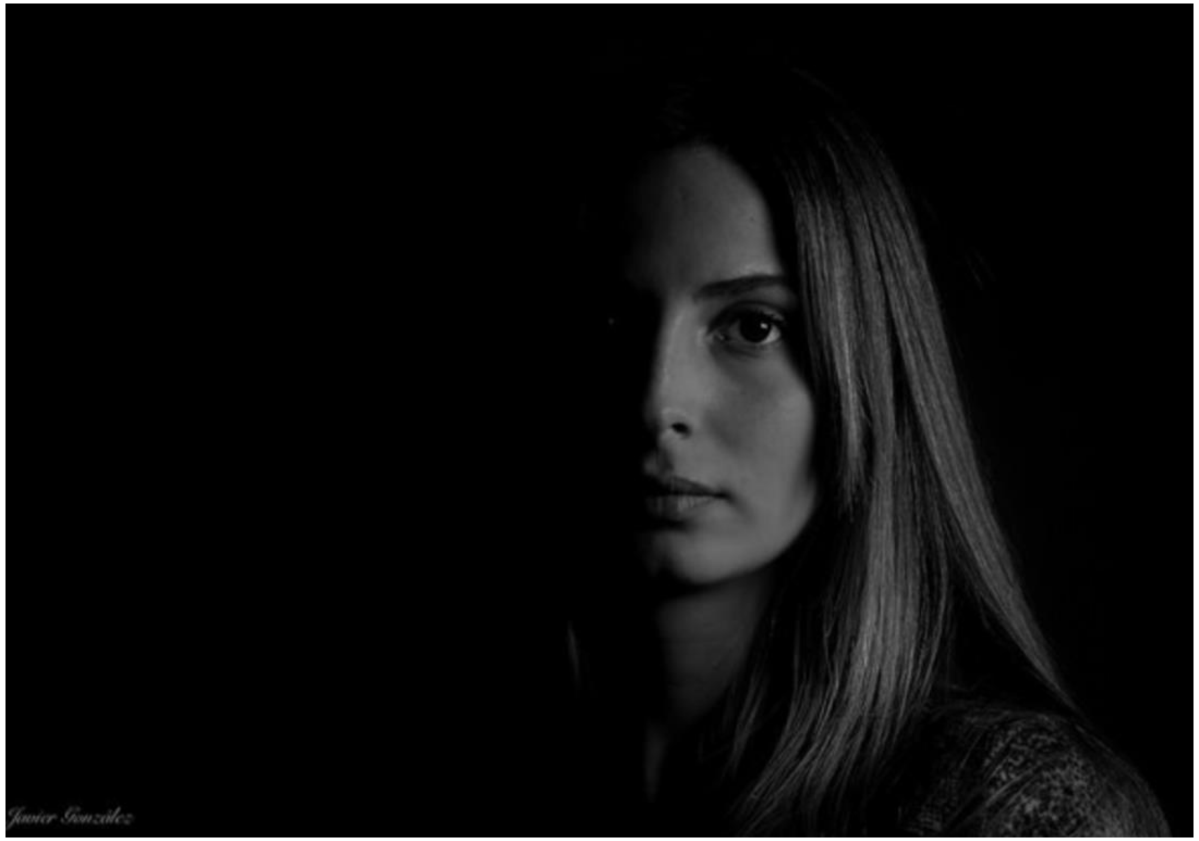
- If a mask is provided, a histogram will be computed for masked pixels only
- If we do not have a mask, we can just provide a value of None

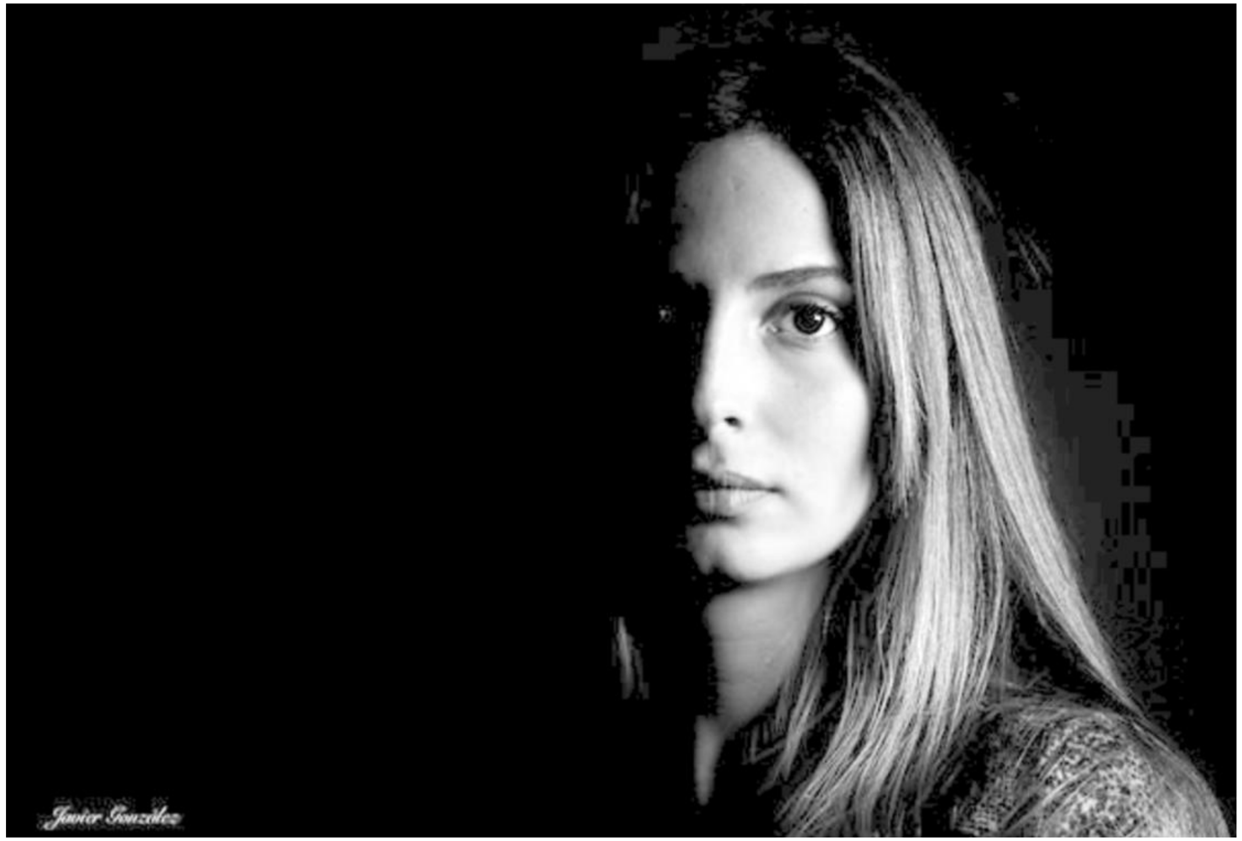
histSize: This is the number of bins we want to use when computing a histogram Again, this is a list, one for each channel we are computing a histogram for. The bin sizes do not all have to be the same. Here is an example of 32 bins for each channel: [32,32,32]

ranges: range of possible pixel values.

- [0, 256] for each channel

Low-contrast images





Histogram Equalization

Histogram equalization improves the **contrast** of an image by “**stretching**” the distribution of pixels

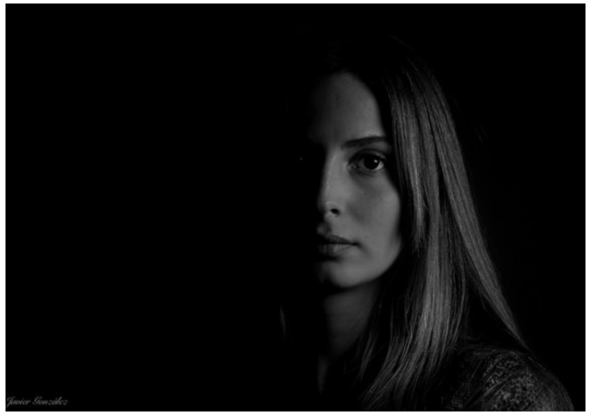
Consider a histogram with a large peak at the center of it. Applying histogram equalization will stretch the peak out towards the corner of the image, thus improving the global contrast of the image.

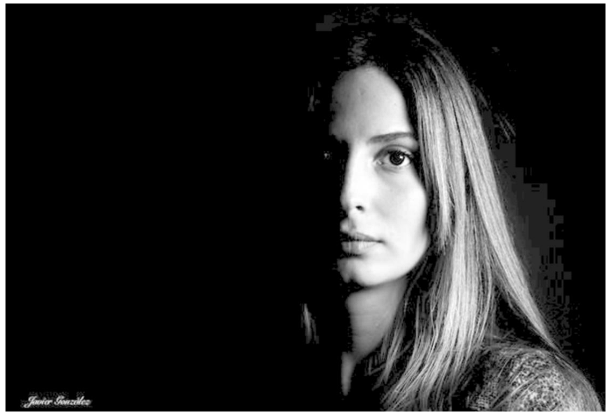
Histogram equalization is applied to **grayscale images**

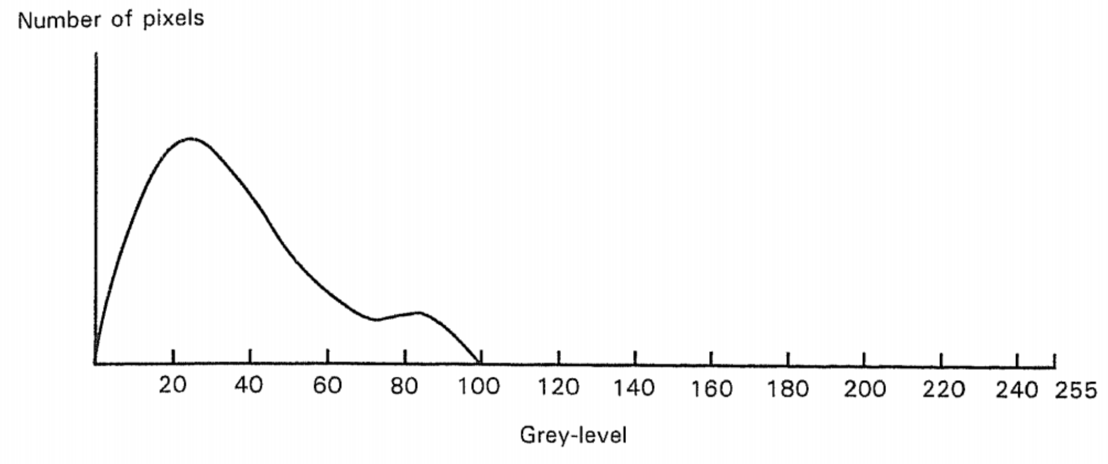
Useful when an image contains foregrounds and backgrounds that are both dark or both light

- It tends to produce unrealistic effects in photographs
- Normally useful when enhancing the contrast of medical or satellite images

Histogram Equalization







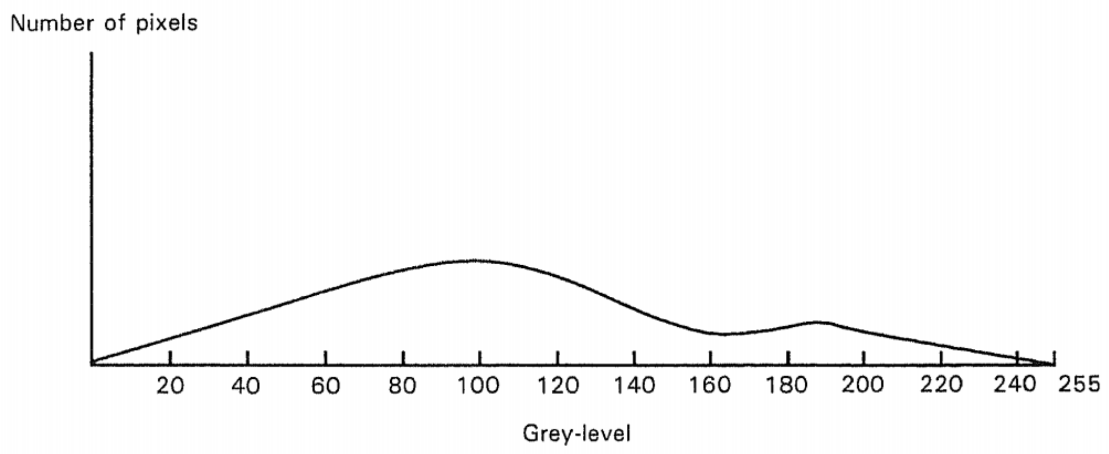


Image Convolutions

What is it?

Convolution

Convolution is **element-wise multiplication** of two matrices followed by a sum

- Take two matrices
- Multiply them element-by-element (not a dot product)
- Sum the elements together

<u>Uses</u>

- Blurring
- Edge Detection
- Convolutional Neural Network (CNN) **

** DL- I

Convolution

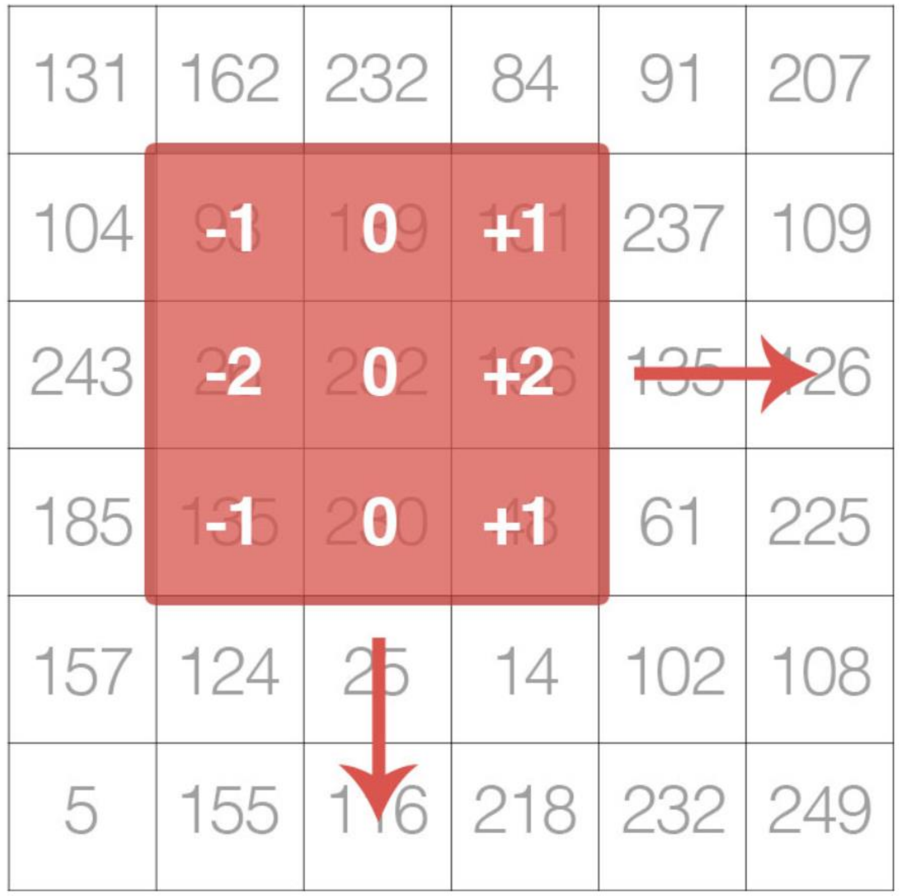


Image (White Background)

Kernel (Orange matrix)

Move the kernel from top-left of the image and move around from left to right and top to down

Convolution Steps

Select an (x, y)-coordinate from the original image

Place the center of the kernel at this (x, y)-coordinate

Take the element-wise multiplication of the input image region and the kernel, then sum up the values of these multiplication operations into a single value. The sum of these multiplications - **kernel output**

Use the same (x, y)-coordinates from Step #1, but this time, store the kernel output in the same (x, y)-location as the output image

Further Reading

Digital Image Processing 4th edition

Rafael Gonzalez & Richard Woods