Transformation

Transformation

This section will cover :

- Histograms
- Image Adjustments
- Image Maths
- Kernels

These topics will be **implemented** and **tested** in Python with OpenCV.

Histograms

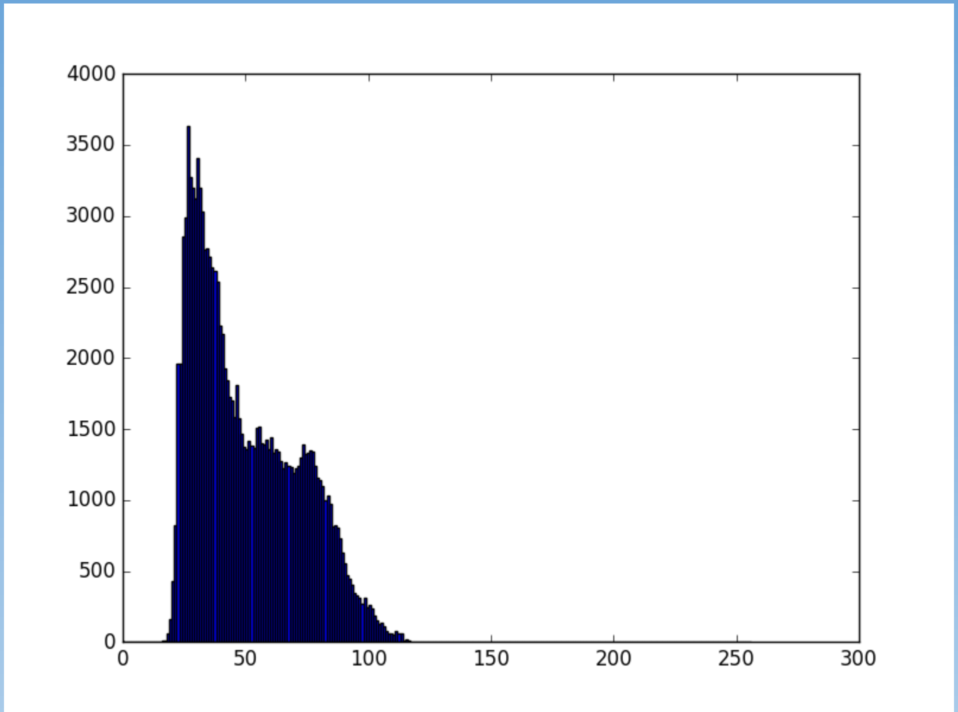
Histograms

Histograms are used to understand more about the distribution of pixel values in an image.

A histogram shows the frequency of occurrence of each pixel value from 0 to 255.

This allows the brightness and contrast of an image to be clearly seen.

Histograms



4000
3500
3000
2500
2000
1500
1000
500
0
0
50
100
150
200
250
300

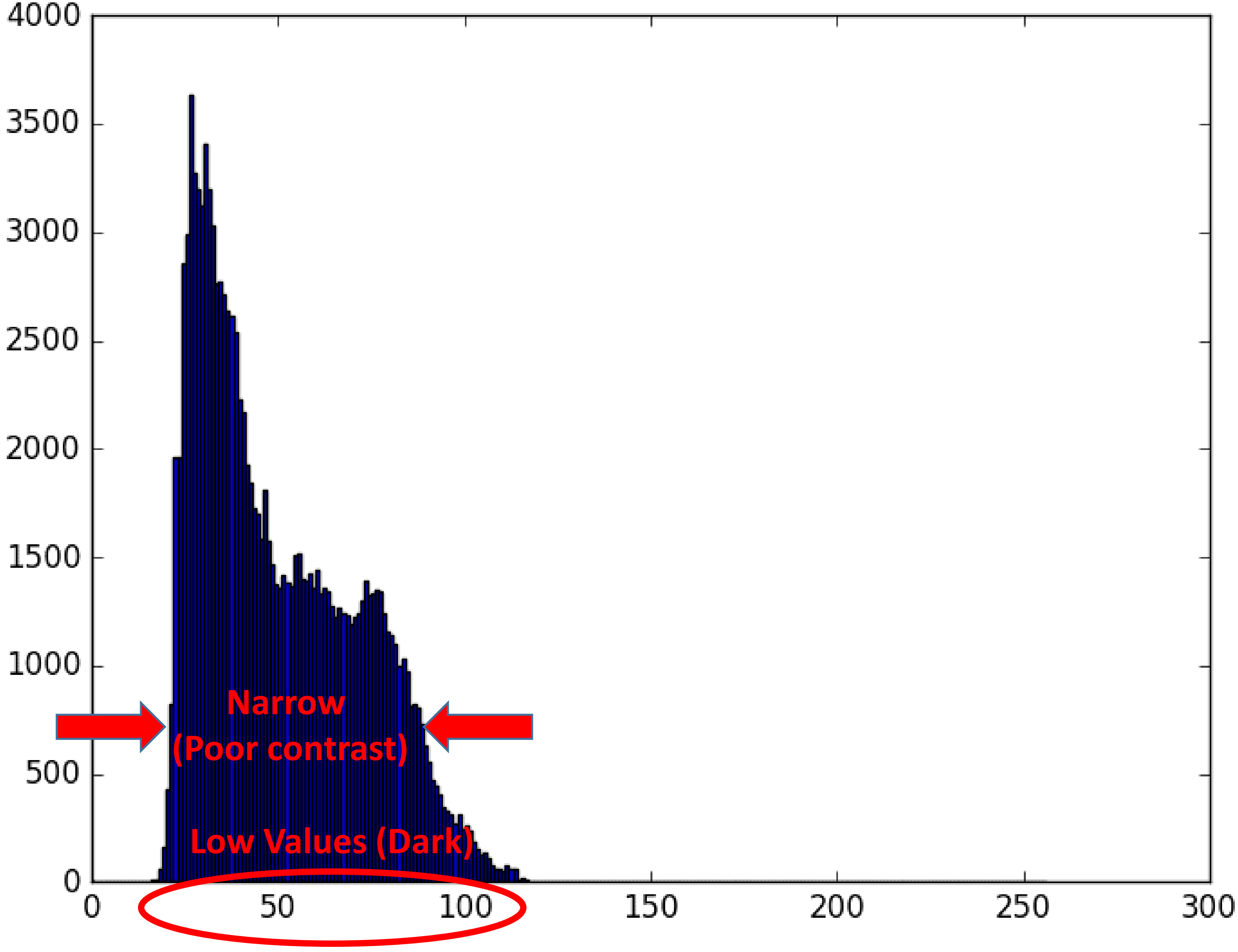
Histograms

The balance in an image can be directly observed from the histogram.

From an intensity histogram, the brightness is given by the location of the pixels on the 0 to 255 scale.

The contrast describes how well these are distributed.

Histograms



4000
3500
3000
2500
2000
1500
1000
500
0
0
50
100
150
200
250
300
Narrow
(Poor contrast)
Low Values (Dark)

Histograms

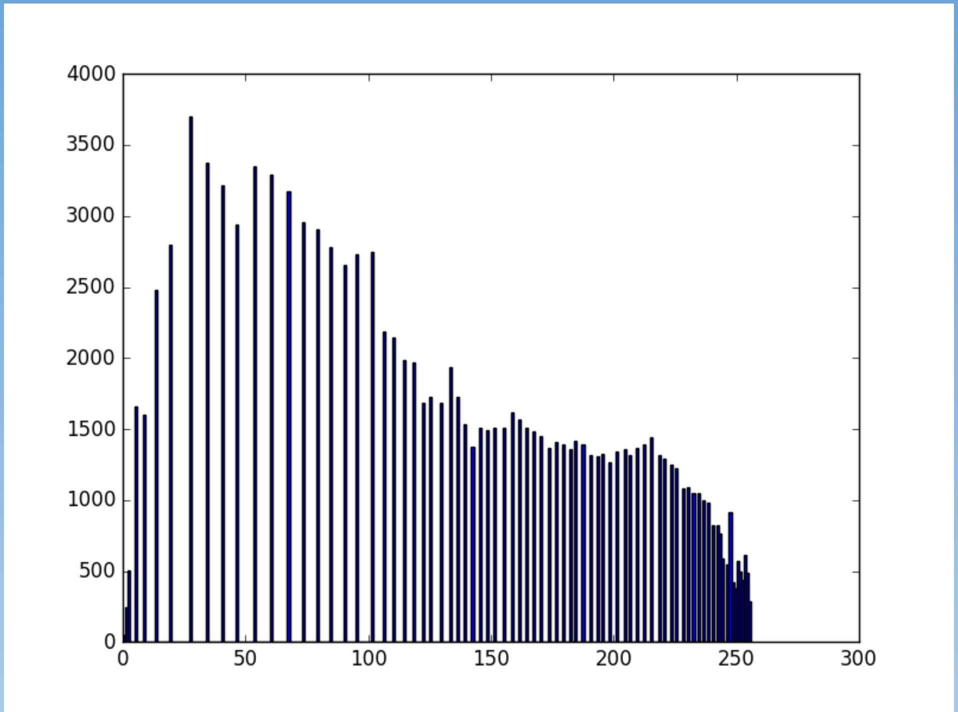
Histogram Equalisation

Any imbalances in a histogram (poor lighting or contrast) can be corrected using histogram equalisation.

The simplest form of this is also known as *contrast stretching*.

This simply redistributes the pixel values across the full range.

Histograms



The new intensity is given by:

$$\mathbf{I} = \mathbf{255} \times \frac{\mathbf{I_{IN}} - \mathbf{I_{MIN}}}{\mathbf{I_{MAX}} - \mathbf{I_{MIN}}}$$

Where $\mathbf{I_{IN}}$ is the original image intensity and $\mathbf{I_{MIN}}$ and $\mathbf{I_{MAX}}$ are the original minimum and maximum intensities.

Histograms

Applications

Don’t forget this can apply to any colour channel so can be used for other equalisations besides brightness and contrast.

Histograms are also used to construct appropriate filtering kernels or select thresholds.

In Python with OpenCV

Plotting Histograms

To plot a histogram, we use Matplotlib's *hist* function.

The pixel values need to first be unravelled from matrix form to a 1D array by using the *ravel* function :

```
Values = G.ravel()
plt.hist(Values,bins=256,range=[0,256]);
```

`Values` are the pixel values of the greyscale image `G`.

Histogram Equalisation

To equalise a histogram, we use the *equalizeHist* function:

```
H = cv2.equalizeHist(G)
```

`H` is the equalised version of the greyscale image `G`.

Task : Histograms

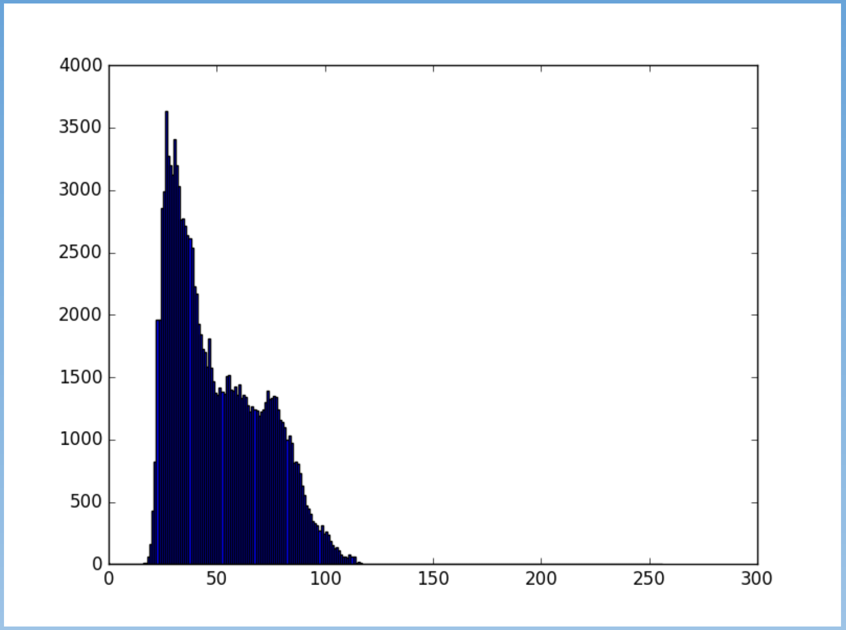
1. Open image “Wartime.jpg”;
2. View its histogram alongside the image;
3. Use histogram equalisation to improve its contrast;
4. View the new histogram alongside the new image.

Advanced Task:

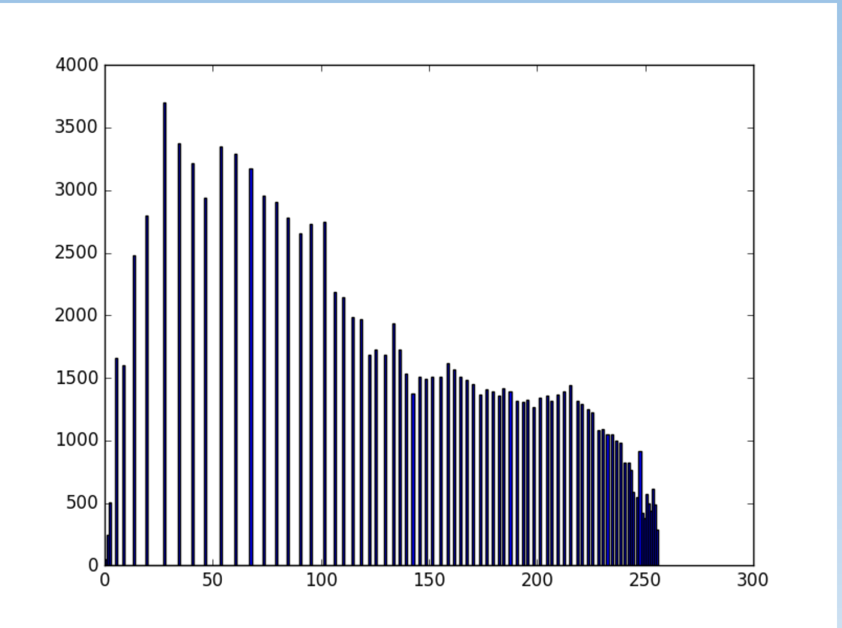
Try this on a poor quality colour image.

Which channel should be equalised?

Histograms



4000
3500
3000
2500
2000
1500
1000
500
0
0
50
100
150
200
250
300



4000
3500
3000
2500
2000
1500
1000
500
0
0
50
100
150
200
250
300

Image Adjustments

Image Adjustments

In order to transform an image, it is often necessary to do some simple adjustments to the whole image first.

Adjustments include:

- Scaling
- Cropping
- Rotating

Image Adjustments

Scaling

Scaling simply means resizing an image.

Note however that the image will not be re-sampled.

This means that the pixel values will be interpolated when expanding and averaged when shrinking.

Scaling can retain aspect ratio or warp the image to a different shape.

Image Adjustments

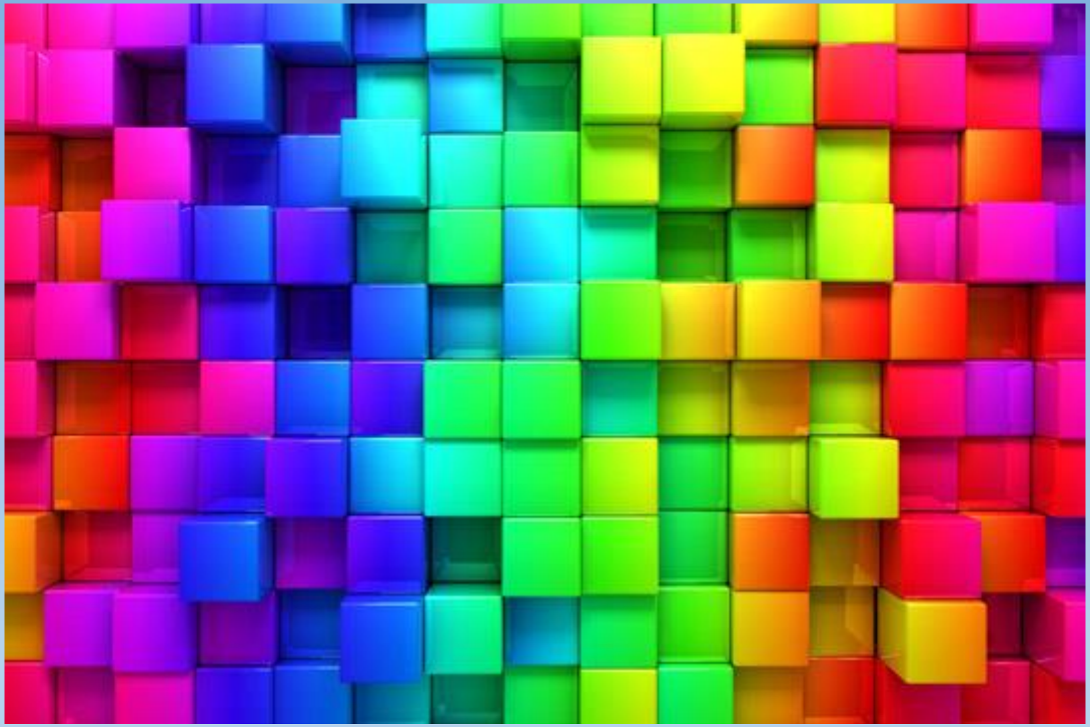


Image Adjustments

Scale = 0.5 x width, 0.5 x height

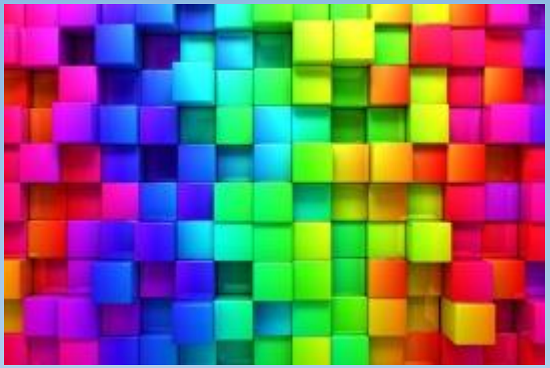


Image Adjustments

Scale = 0.5 x height

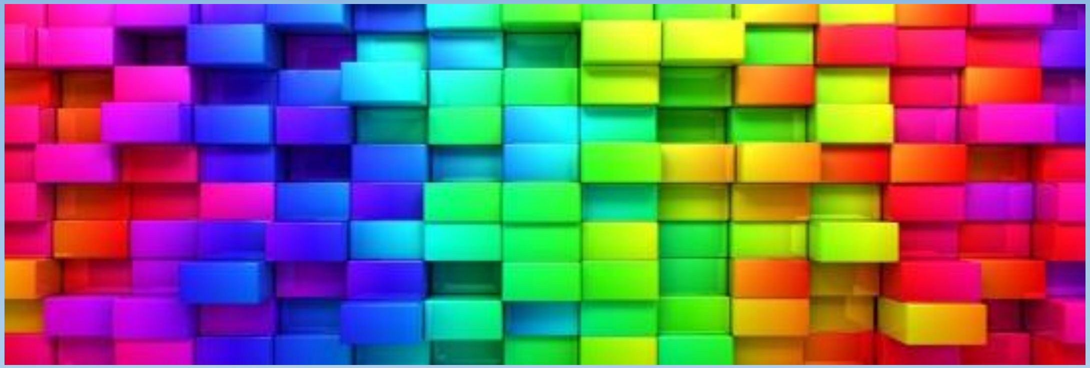
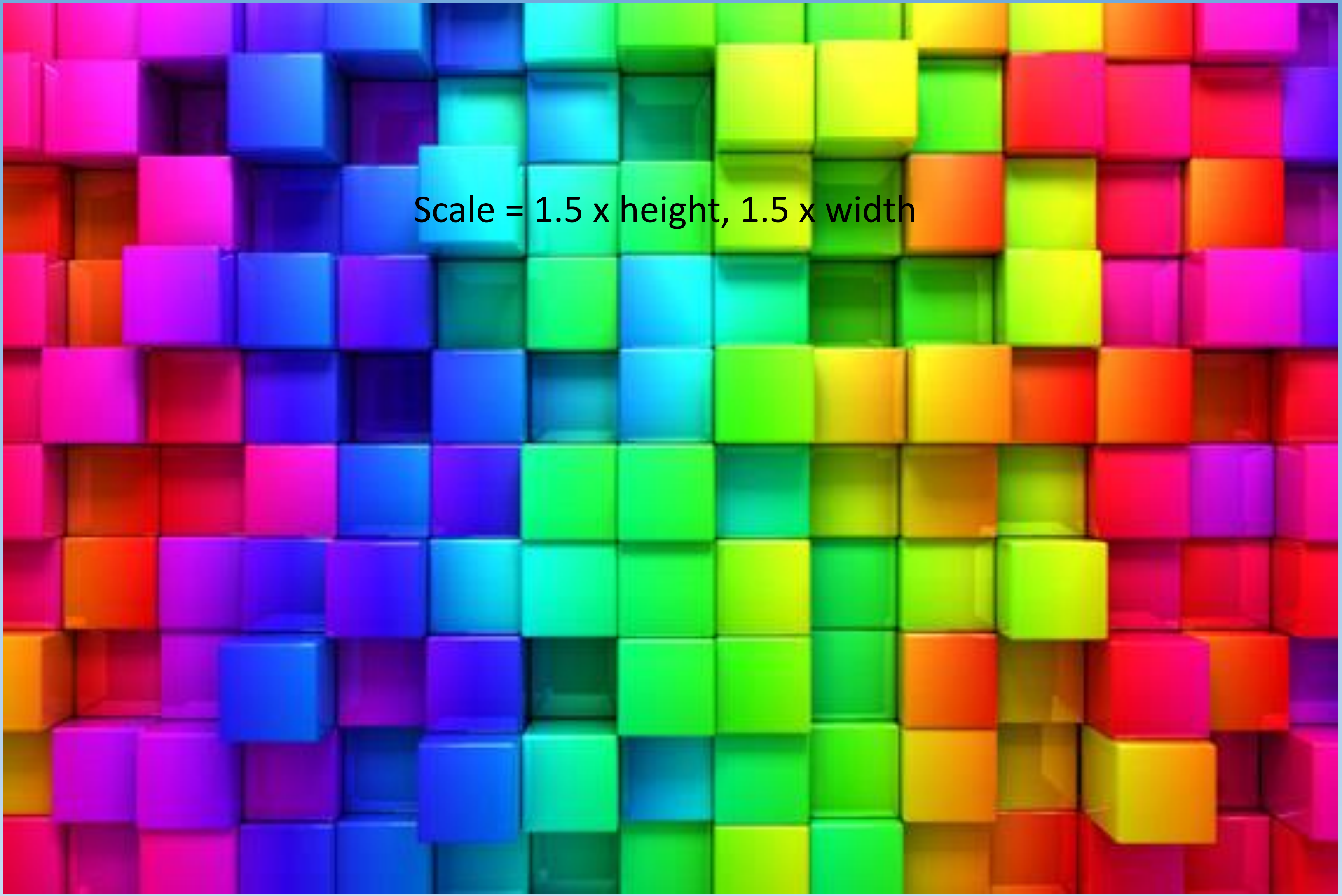


Image Adjustments



Scale = 1.5 x height, 1.5 x width

Image Adjustments

Cropping

In cropping an image, a smaller section of the image is extracted. Cropping differs from scaling in that some image information is lost. The new image can be either the size of the smaller section or can be buffered with zeros to retain the same size as the original.

Image Adjustments

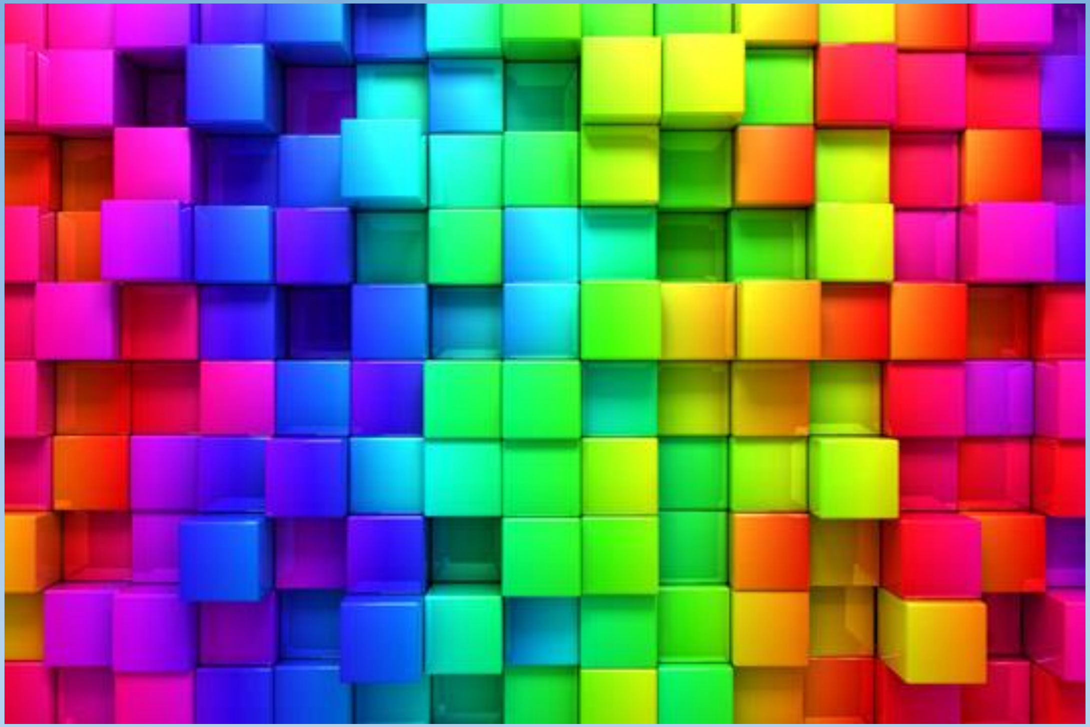


Image Adjustments

Cropped Image

Image Adjustments

Cropped image with buffer

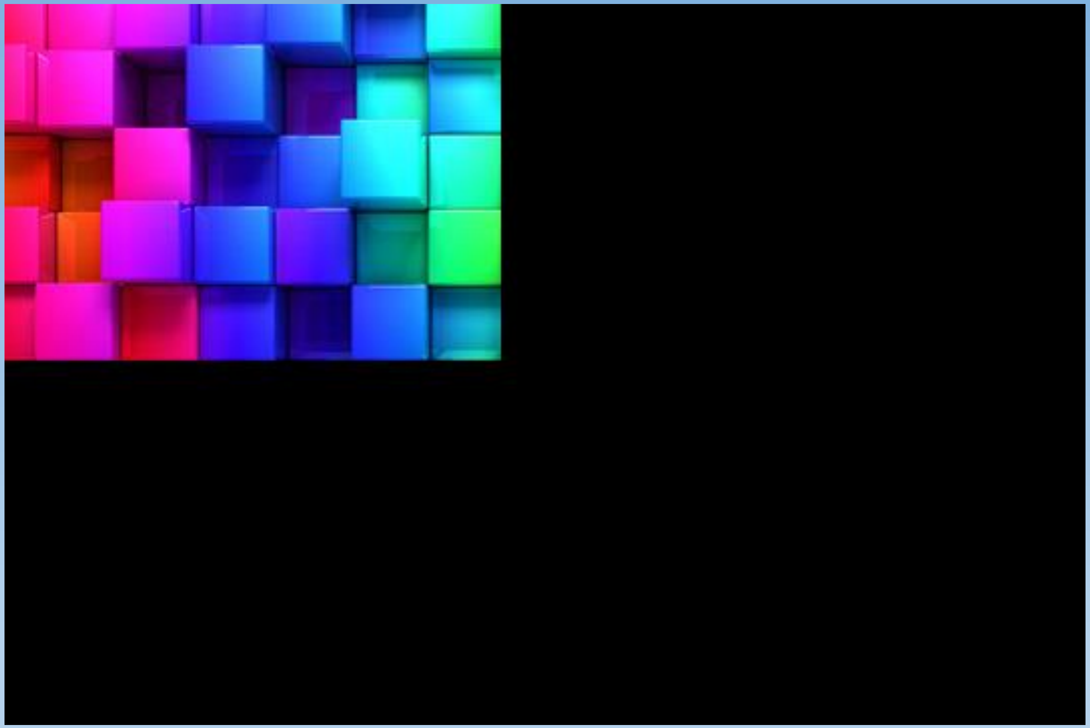


Image Adjustments

Rotating

An image can be rotated by creating a rotation matrix.

Note that a rotation may leave blank spaces.

These will be filled with zeros by default.

Image Adjustments

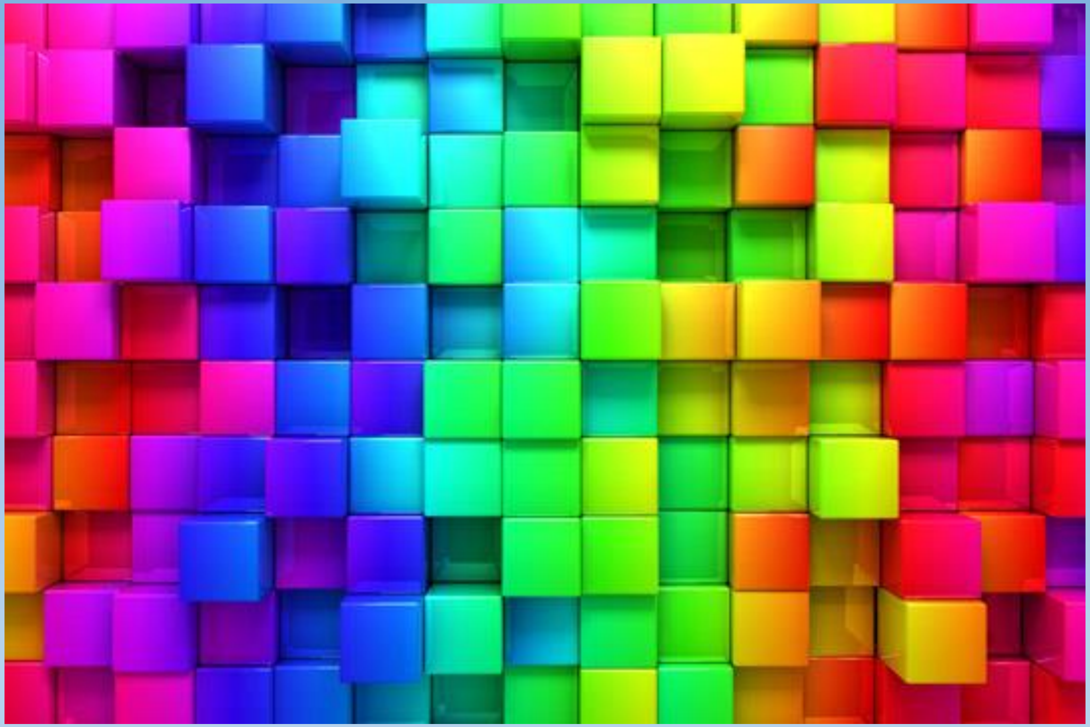


Image Adjustments

Rotation $=30^o$

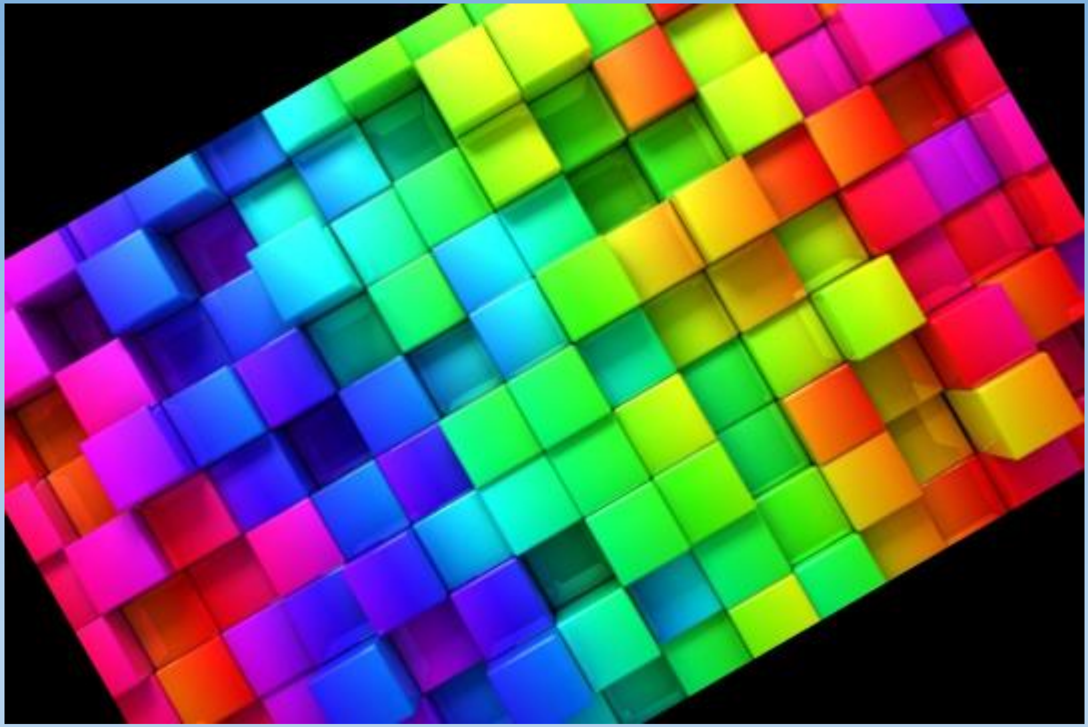


Image Adjustments

Rotation $=60^o$

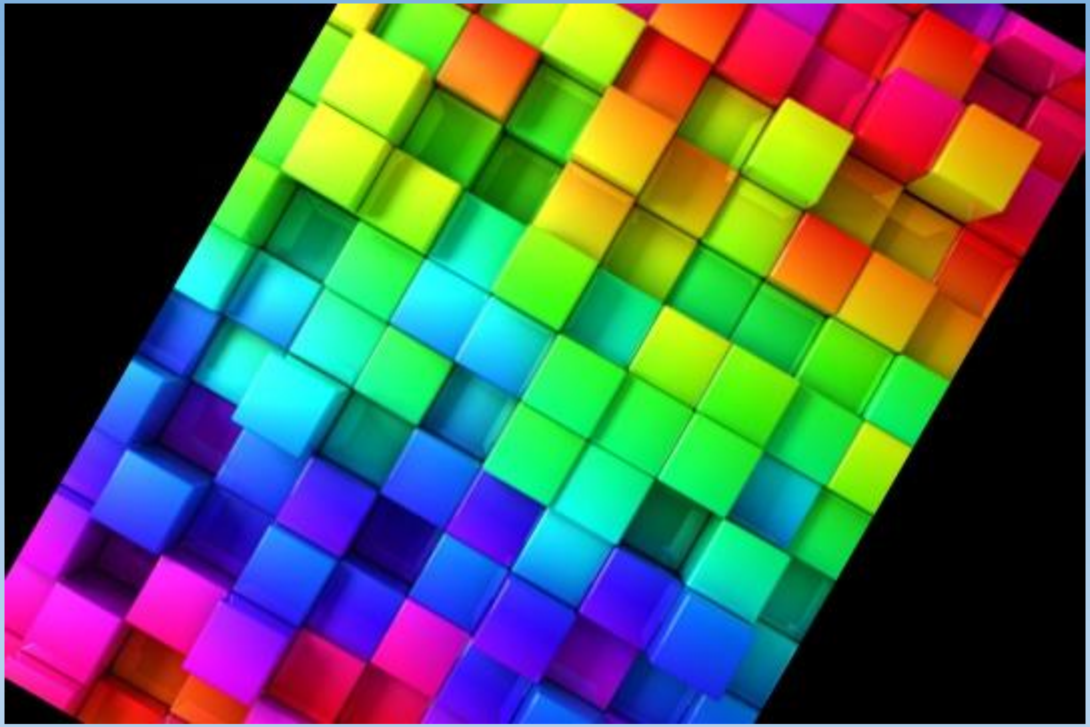
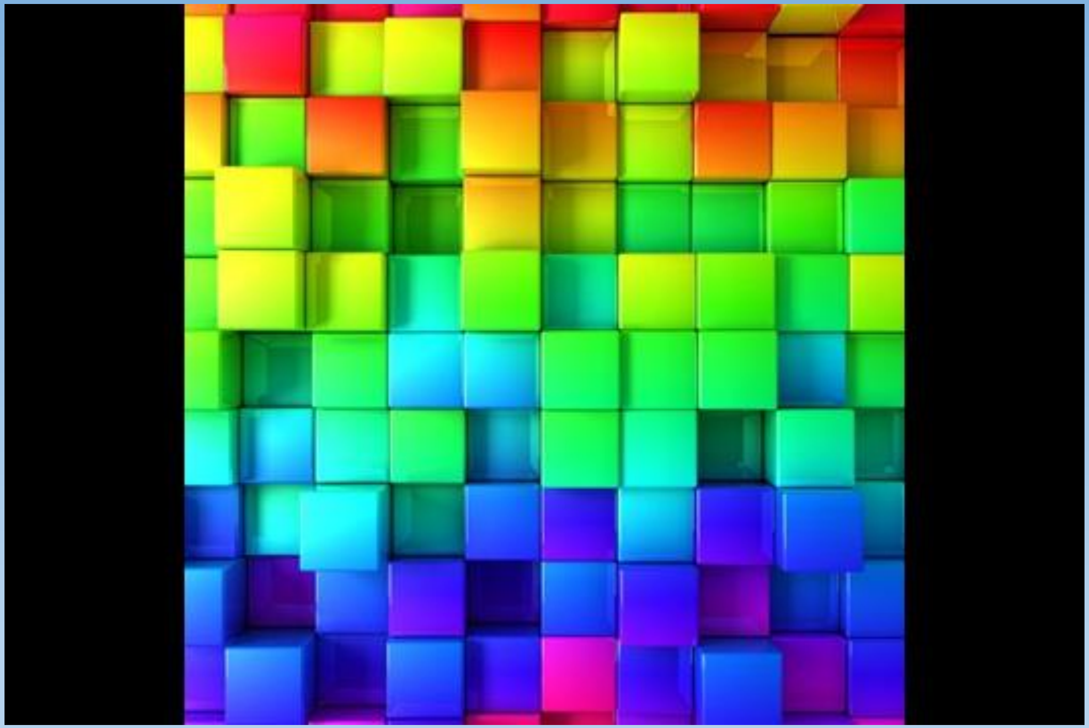


Image Adjustments

Rotation $=90^{o}$



In Python with OpenCV

Scaling

Scaling can be performed in OpenCV using the *resize* function.

It is useful to know the original size first:

```
h, w, d = I.shape
S = cv2.resize(I, dsize=(2*w, 2*h))
```

`S` is the new image, twice the size of the original image, `I`.

`h, w, d` are the height, width and depth of the original.

*Note: The new size can only be an integer number of rows and columns.

Cropping

We have actually seen cropping before when investigating pixels:

```
C = I[0:180,0:270]
```

`C` is the new cropped image, with dimensions 270 x 180.

The height and width of the original can be used here to crop to a specific fraction of the original.

Rotating

To rotate an image, we have to first create a rotation matrix using the *getRotationMatrix2D* function.

Then this is can be applied using the *warpAffine* function.

```
M = cv2.getRotationMatrix2D(center=(cx,cy),
                    angle=d, scale=s)
R = cv2.warpAffine(I, M = M, dsize=(w,h))
```

`M` is the rotation matrix with centre of rotation, `(cx,cy)`, degrees, `d` and scaling factor, `s`. `R` is the new rotated image with size, `(w,h)`.

Task : Adjusting

1. Open any image;
2. Crop out the fourth quadrant of the image;
3. Rotate this cropped section by 45^o.

Advanced Task:

Don’t fall off the edge!

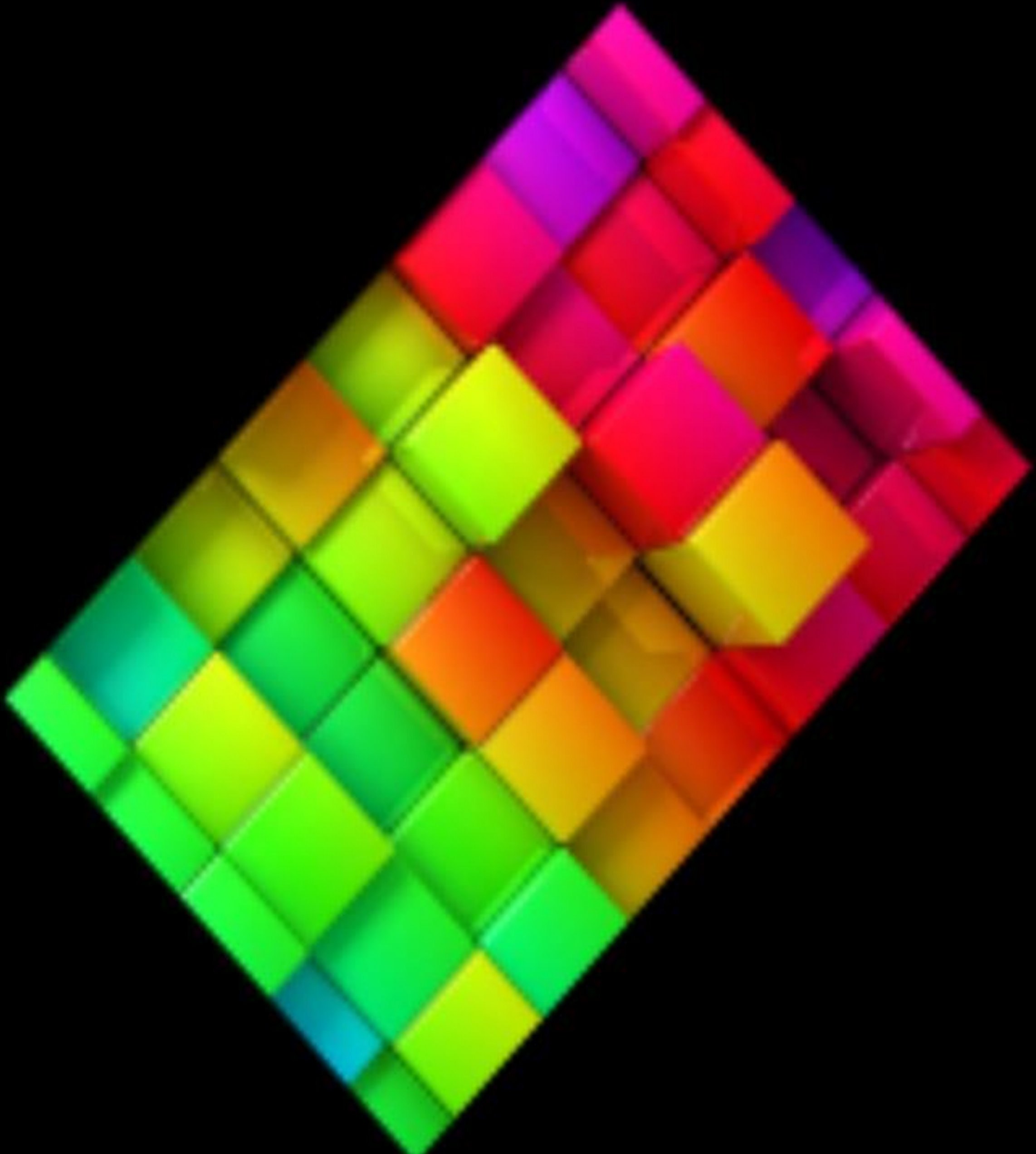


Image Maths

Image Maths

As images are matrices of numbers, they can be combined or altered using simple maths.

This is done on a pixel-by-pixel basis.

For example, multiplication of images is done as a dot-product, rather than matrix multiplication:

$$I_{OUT} = I_1 .* I_2$$

Image Maths

<u>Range</u>

The pixel range is limited to 0 to 255 for an 8-bit image (2D).

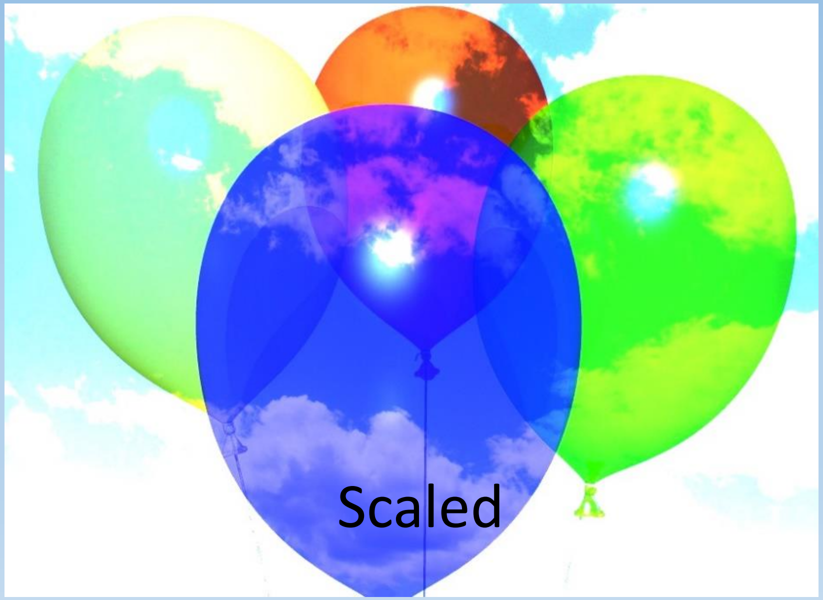
What happens when we exceed this range?

We then have the choice between *scaling* and *saturation*.

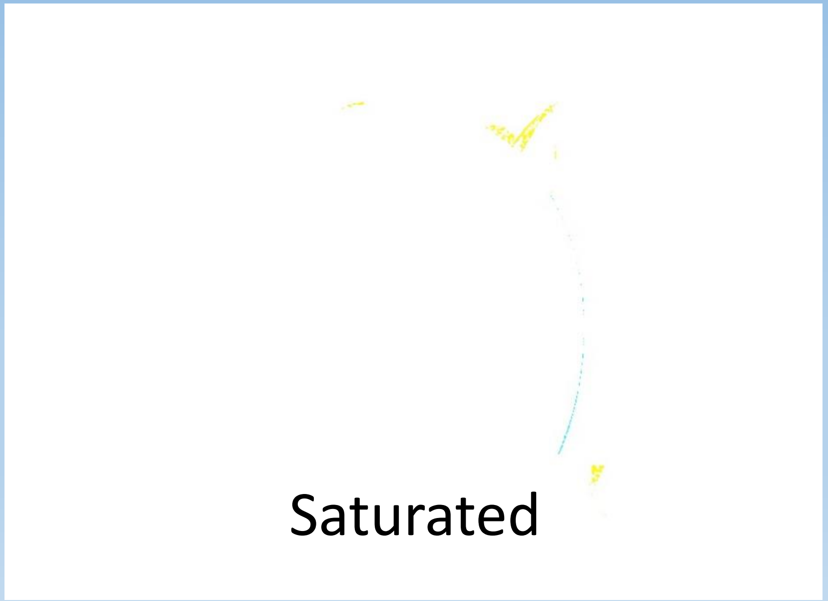
Scaling is dividing all pixels by a fixed value to bring them into range.

Saturation means cutting off at a fixed value.

X



Scaled



Saturated

Image Maths

Operations

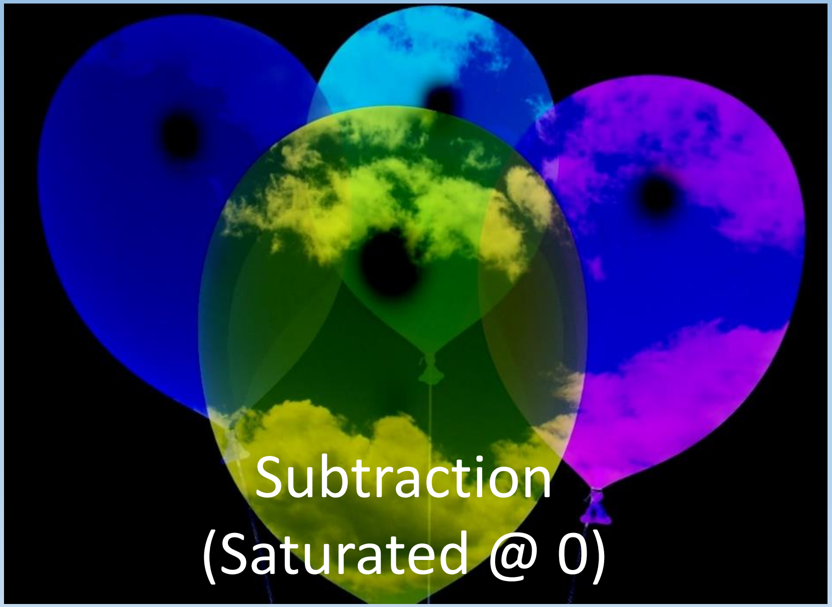
Arithmetic operations on images include addition, subtraction, multiplication and division.

Boolean operators such as AND, OR and NOT can also be used, however, Boolean operators are more normally used with binary images.

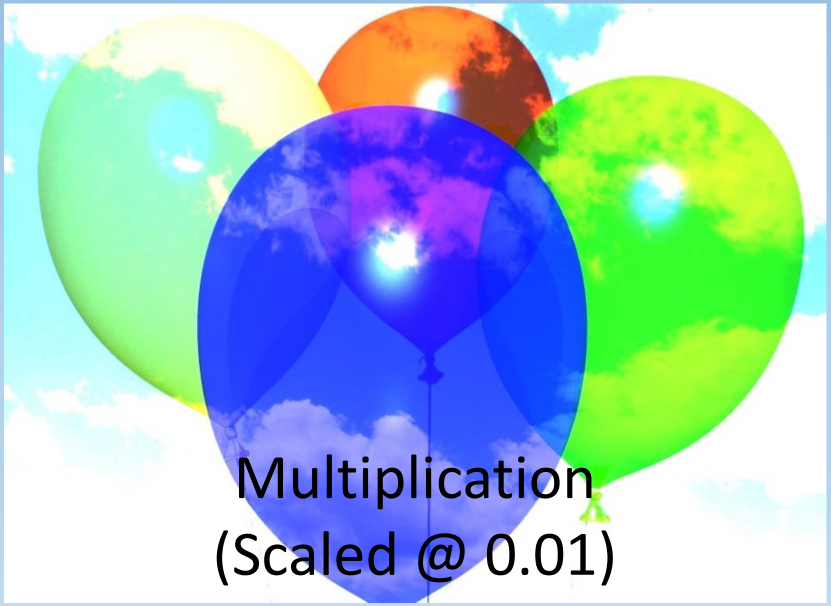
+

Addition
(Saturated @ 255)

–


Subtraction
(Saturated @ 0)

×



Multiplication
(Scaled @ 0.01)

÷

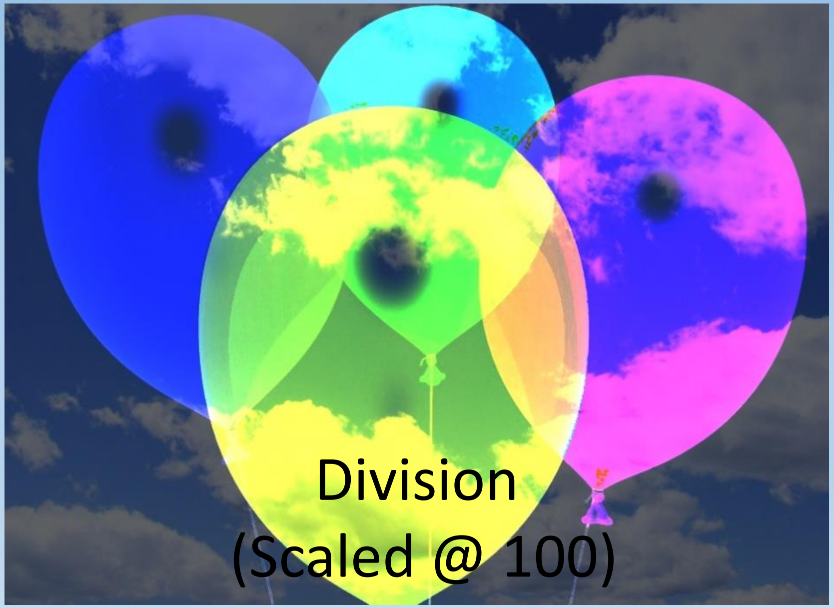

Division
(Scaled @ 100)

Image Maths

Scalars

As well as combining images with each other, simple maths can be used to apply a scalar to an image.

A scalar is just a fixed value or constant.

Scalars can be used in addition, subtraction, multiplication or division.

In Python with OpenCV

Addition

Addition can be performed in OpenCV using the *add* function:

```
A = cv2.add(I1,I2)
```

`A` is the new image, the sum of `I1` and `I2`.

By default, OpenCV will saturate the output at 255.

*Note: The source images should be the same size.

Subtraction

Subtraction can be performed in OpenCV using the *subtract* function:

```
S = cv2.subtract(I1,I2)
```

`S` is the new image, the difference between `I1` and `I2`.

By default, OpenCV will saturate the output at 0.

*Note: The source images should be the same size.

Multiplication

Multiplication can be performed in OpenCV using the *multiply* function:

```
M = cv2.multiply(I1,I2,scale = 0.01)
```

`M` is the new image, the product of `I1` and `I2`.

By default, OpenCV will saturate the output at 255 but the optional `scale` parameter can be used to scale instead.

*Note: The source images should be the same size.

Division

Division can be performed in OpenCV using the *divide* function:

```
D = cv2.divide(I1,I2,scale = 100)
```

`D` is the new image, the division of `I1` by `I2`.

By default, OpenCV will saturate the output at 0 but the optional `scale` parameter can be used to scale instead.

*Note: The source images should be the same size.

Task : Image Maths

1. Open image “Orange.png and “Water.jpg”;
2. Scale each image to 50 % by multiplying each by 0.5;
3. Add the two scaled images to get a composite image;
4. Adjust the scaling to give a nicer output.

Advanced Task:

Investigate the *addWeighted* function to achieve the same.

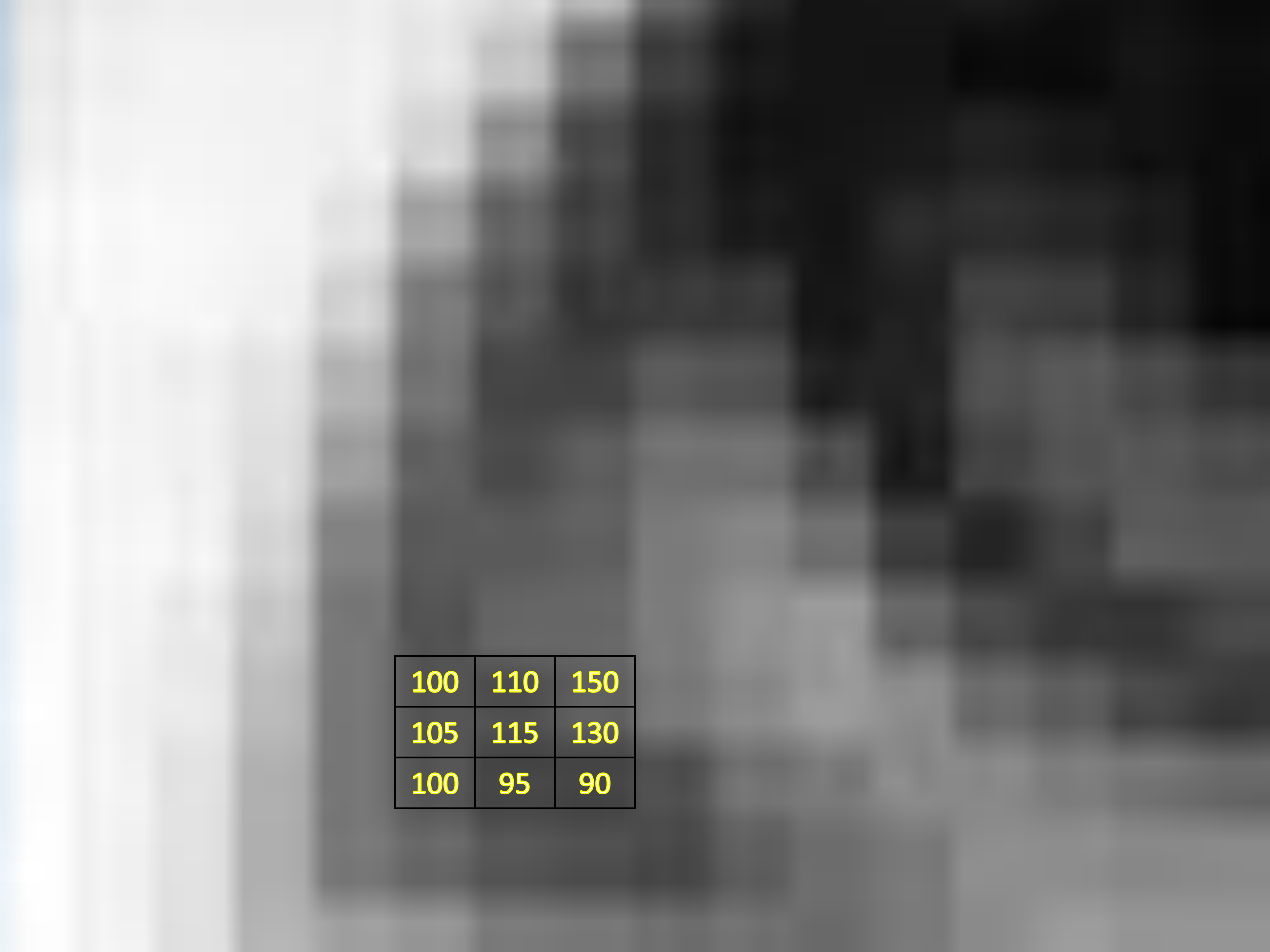
Kernels

Kernels

Many operations in Image Processing are achieved by applying a *filter* or *kernel* to the image.

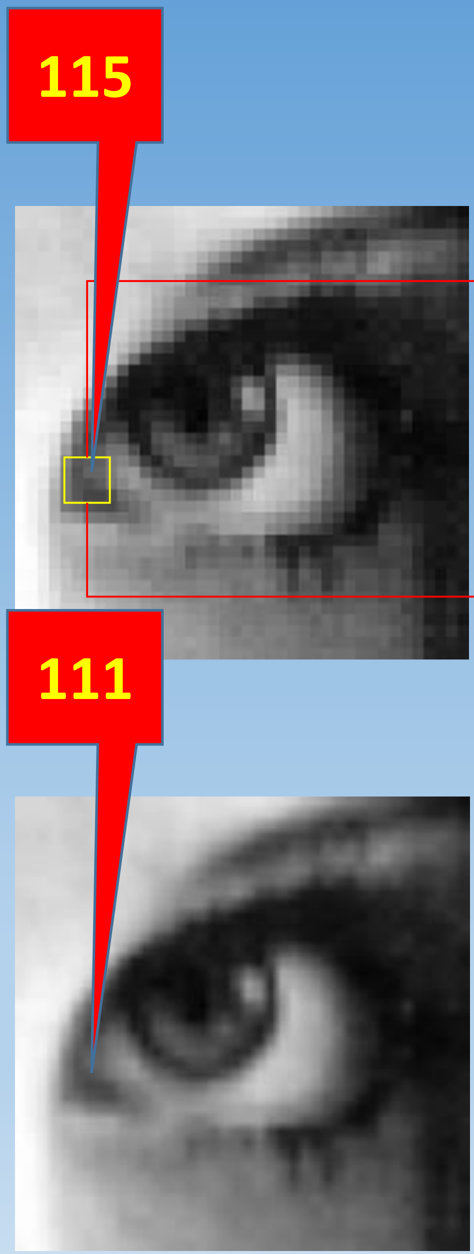
This is an operator which makes each pixel a weighted sum of its neighbours.

This process is called *convolution*.



100	110	150
105	115	130
100	95	90

Kernels



Convolution

100	110	150
105	115	130
100	95	90

Convolve this with:

$$\frac{1}{27}\begin{array}{|c|c|c|}\hline 1 & 4 & 1 \\ \hline 4 & 7 & 4 \\ \hline 1 & 4 & 1 \\ \hline \end{array}$$

New value = weighted sum =

(100 x 1 + 110 x 4 + … + 90 x 1) / 27

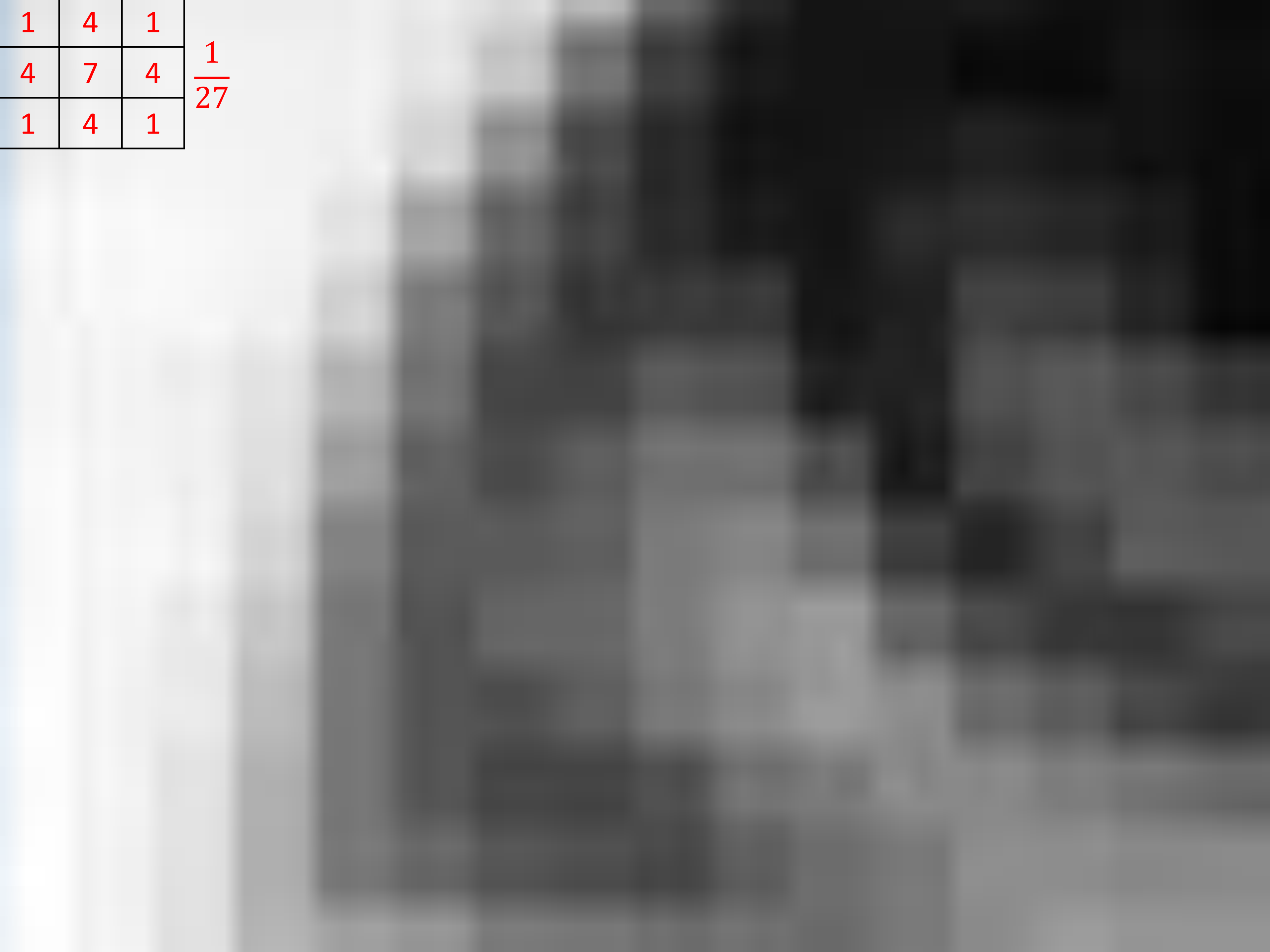
= 111

Repeat for every pixel

Result, in this case, is Gaussian smoothing

1	4	1
4	7	4
1	4	1

$$\frac{1}{27}$$



Kernels

<u>Applications</u>

Kernels are the fundamental unit of many applications.

They can be used for low-pass and high-pass filtering and gradient extraction for edge detection and feature finding.

They are also the main method of reducing noise.

Kernels

<u>Building Kernels</u>

There are many kernel designs for existing applications but it is also possible to build your own.

See if you can guess what these kernels might do….

1	1	1
1	1	1
1	1	1

$\frac{1}{9}$

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

$\frac{1}{4}$

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

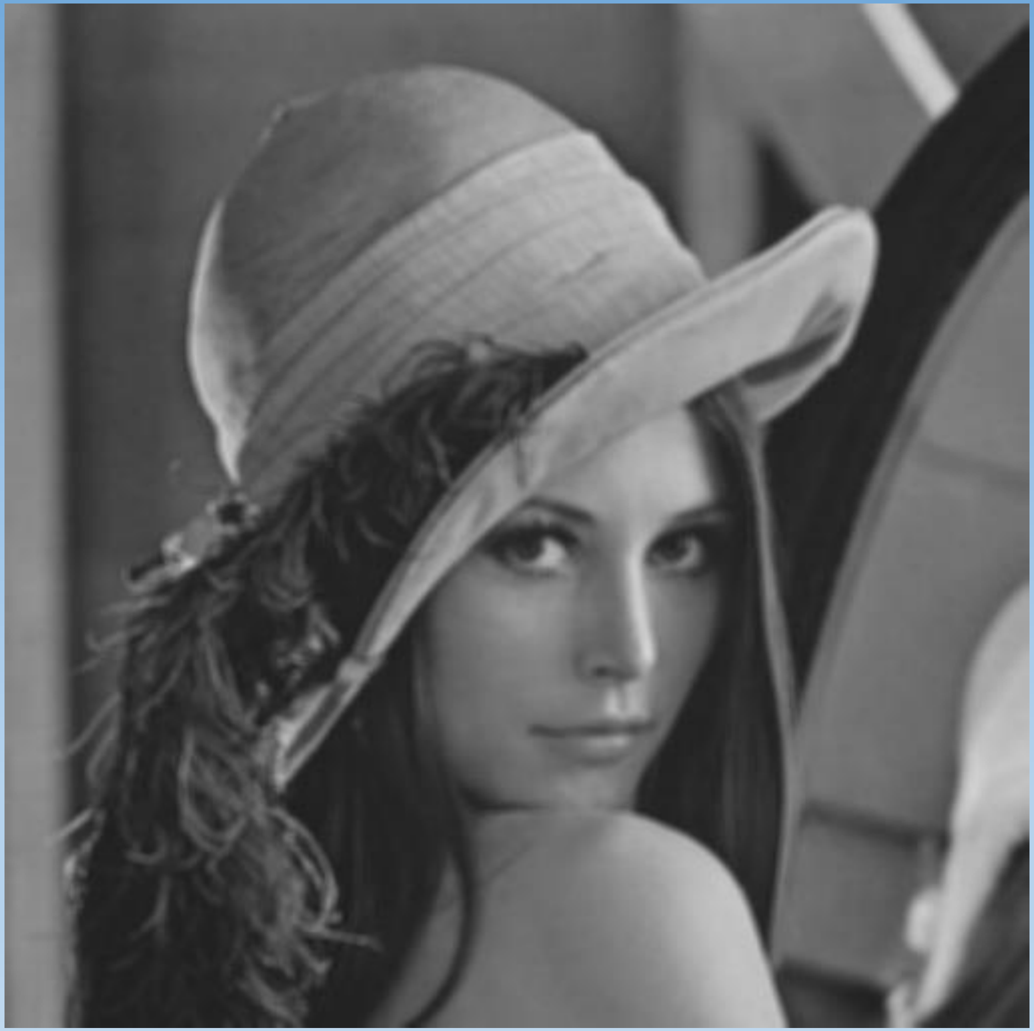
$\frac{1}{4}$

-1	-1	-1
-1	8	-1
-1	-1	-1

$\frac{1}{8}$

Kernels

Kernels



$$\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} \ \frac{1}{9}$$

Low Pass Filter

Kernels

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

$\frac{1}{4}$

Horizontal Gradient

Kernels

-1	-2	1
0	0	0
1	2	1

$\frac{1}{4}$

Vertical Gradient

Kernels

-1	-1	-1
-1	8	-1
-1	-1	-1

$\frac{1}{8}$

High Pass Filter

In Python with OpenCV

Building Kernels

To build a kernel, we use the Numpy *array* function:

```
k = np.array([[1,4,1], [4,7,4], [1,4,1]],
dtype=float)
```

`k` is the kernel.

The type of the array (`dtype`) should be specified here.

*Note: This particular kernel should be divided by 27.

Applying Kernels

A kernel is applied using the *filter2D* function:

```
F = cv2.filter2D(I,ddepth=-1,kernel=k)
```

`k` is the kernel, `I` is the original image and `F` is the newly filtered image.

*Note: `ddepth` is the bit depth of the output image (e.g. 8-bit).

Setting it to -1 means the output image has the same bit depth as the original image.

Task : Kernels

1. Open any image;
2. Build a kernel for sharpening an image

   (you will need to research this);
3. Apply this filter to the image.

Advanced Task:

Subtract the filtered image from the original to highlight changes.

Transformation

In this section you have learned about :

- Histograms
- Image Adjustments
- Image Maths
- Kernels

These topics were **implemented** and **tested** in Python with OpenCV.

Questions?