Material Summary: Working with Images

1. Image Processing

1.1 Loading and Inspecting Images

- There are many ways to read an image
- One of the easiest is using scikit-image

```
from skimage.io import imread
tiger_image = imread("tiger.jpg")
```

- Displaying the image

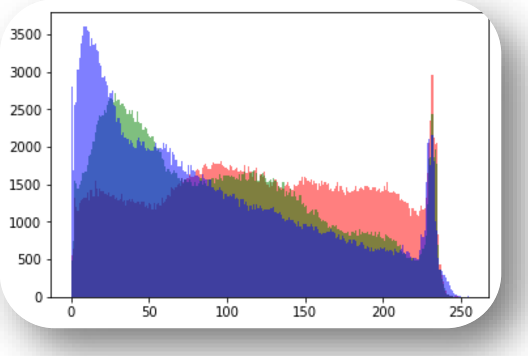
```
plt.imshow(tiger_image)
```

- The image is actually a **matrix of pixels**
  - Each pixel is an array of three values: **R, G, B** ∈ $[\mathbf{0}; \mathbf{255}]$
  - Grayscale images only have **one value per pixel**
- Most image **processing algorithms** are easier to understand on grayscale images

```
red = tiger_image[:, :, 0]
green = tiger_image[:, :, 1]
blue = tiger_image[:, :, 2]
```

1.2 Image Histogram

- As usual, histograms tell us how the values are distributed
- How many dark values, how many light values
- Maximum brightness, peaks, etc.



- Histograms need to have a **single variable**
  - Take each channel separately, e.g., red
  - Convert the **2D matrix to 1D array**: **image.ravel()**
  - Show the histogram as usual
    - It's common to use 256 bins

```
plt.hist(red.ravel(), bins = 256, color = "red")
plt.show()
```

- We can also plot **all channels on a single histogram**

1.3 Converting to Grayscale

- Sometimes working **per channel is not necessary**
- We can combine **all three channels** and get a grayscale image
- Simplest way: **get the mean of all values**

```
tiger_grayscale = np.mean(tiger_image, axis = 2)
```

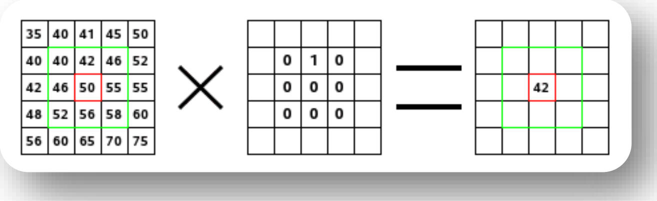
- Better way: **use coefficients for each channel**
  - The human eye **discerns colors differently**
  - Were more sensitive to green colors
  - Some formulas are given here

```
tiger_grayscale = 0.299 * red + 0.587 * green + 0.114 * blue
```

- Depending on the image, the differences **may or may not be easy to see**
- For art purposes, we can experiment with our own **coefficients for combining all channels**

1.4 Convolution

- **Convolution kernel (filter)**
  - A small, usually 3x3, matrix of numbers
- Convolution process
  - **Input**: image, kernel
  - **Output**: new image



35	40	41	45	50
40	40	42	46	52
42	46	50	55	55
48	52	56	58	60
56	60	65	70	75

×

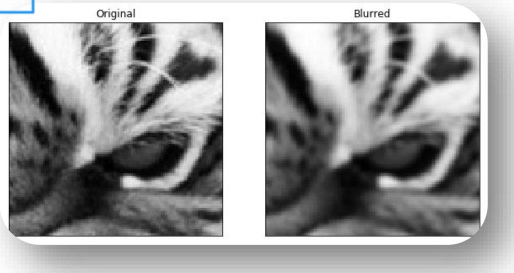
	0	1	0	
	0	0	0	
	0	0	0	

=

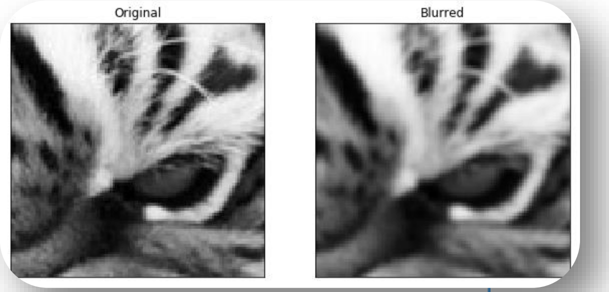
		42		

- Combining the image and a kernel:
  - Apply the kernel **over each pixel**
  - Multiply the values **element-wise** (Hadamard product)
  - Sum **all values**
  - Assign the **sum to the corresponding pixel** in the output image
    - Image corners are treated in different ways, not really important how
- The choice of kernel depends what the **output image will represent**

```
from scipy.ndimage.filters import convolve
convolve(image, kernel)
```



- Example: **box blur**

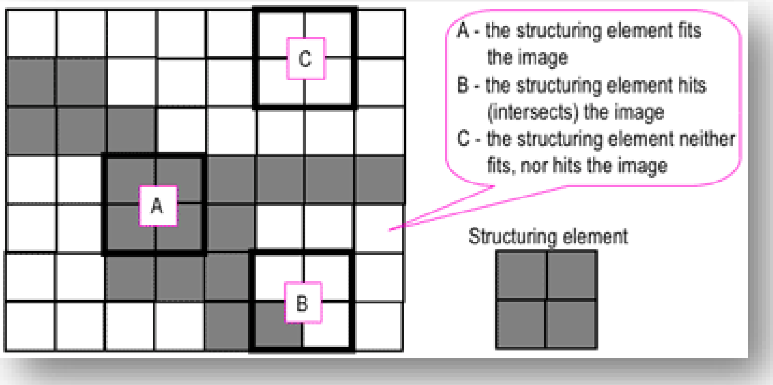


```
box_blur_kernel = np.array([
  [1, 1, 1],
  [1, 1, 1],
  [1, 1, 1]
]) / 9

blurred = convolve(tiger_grayscale, box_blur_kernel)
plt.imshow(tiger_grayscale[150:250, 300:400], cmap = "gray")
plt.show()
plt.imshow(blurred[150:250, 300:400], cmap = "gray")
plt.show()
```

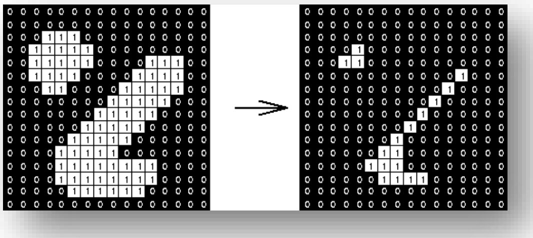
1.5 Image Morphology

- Four main operations (see this tutorial)
  - **Dilation**
  - **Erosion**
  - **Opening**
  - **Closing**
- A simple series of **algorithms for image transformation**
- Basic methodology
  - Choose a structuring element (e.g., 2x2 square or cross)
  - Move the element around the image
  - Apply an operation
- **Input**: **binary image**
  - Pixel values **0** and **1**, **not [0; 255]**
  - This is called **thresholding**
- **Output**: **transformed image**


A - the structuring element fits the image
B - the structuring element hits (intersects) the image
C - the structuring element neither fits, nor hits the image
Structuring element
A
B
C

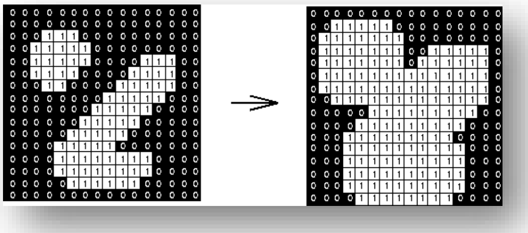
First get **all values** inside the structuring element

- **Erosion**: **replace all values with the min value**
  - Strips away a layer of pixels
  - Holes become larger
  - Small regions are eliminated
- **Dilation**: **replace all values with the max value**

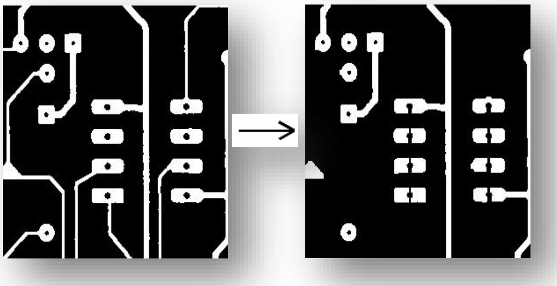


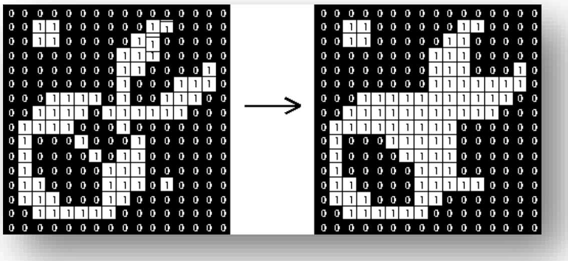
```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 0 1 1 1 0 0
0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0
0 0 0 1 1 0 0 0 0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0
0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 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 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 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 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 0 0 0 0 0 0 0 0 0 0 0 0
```



    - Adds a layer of pixels
    - Gaps become smaller
    - Small gaps are filled in
- **Opening**: **erosion followed by dilation**
    - Pixels which survived erosion are restored to their original size
    - Opens up a gap between two objects connected by thin bridges
- **Closing**: **dilation followed by erosion**
    - Fills in holes in the regions while keeping the initial region sizes





1.6 Other Operations on Images

- Matrix operations – **pixel-wise**
    - One image:
        - **Addition, Gain, Negative**
        - **Resampling, Cutting**
    - Transformations – **perspective**, **warp**, etc.
    - Two (or more) images:
        - **Addition (multiple exposure)**
        - **Subtraction (difference)**
        - **Division (normalization)**
        - **Averaging**
- Thresholding **(usually 2 levels)**
- Fourier **transform**, **filtering** and **convolution**
- Contrast **enhancement**
- Histogram **equalization**
- Stacking (many **2D images** ⇒ **one 3D image**)
- Analysis:
    - **Measurements, Segmentation, Object extraction / Identification**
    - **Enhancements, Inpainting**