Working with Images and Text

Reading, exploring and analyzing, feature extraction

Yordan Darakchiev

Technical Trainer

iordan93@gmail.com

sli.do

#DataScience

Table of Contents

- Image processing
  - Reading, exploring, manipulation
  - Convolution
  - Image morphology
- Text processing
  - Text preparation
  - Frequency analysis
  - TF-IDF

Image Processing

Understanding what people see

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: $R, G, B \in [0; 255]$
  - Grayscale images only have one value per pixel
- Most image processing algorithms are easier to understand on grayscale images

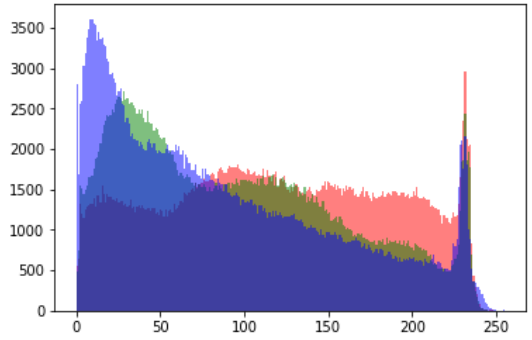
```
red = tiger_image[:, :, 0]
green = tiger_image[:, :, 1]
blue = tiger_image[:, :, 2]
```

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



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)
```

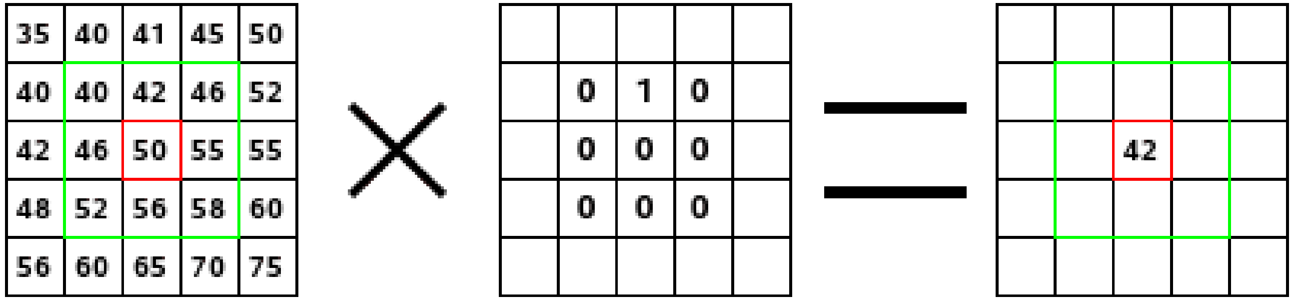
  - Better way: use coefficients for each channel
    - The human eye discerns colors differently
    - Were more sensitive to green colors
    - Some formulas are given here

```
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
    - It's easiest to see the differences when we compare the histograms
- For art purposes, we can experiment with our own coefficients for combining all channels

Convolution

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

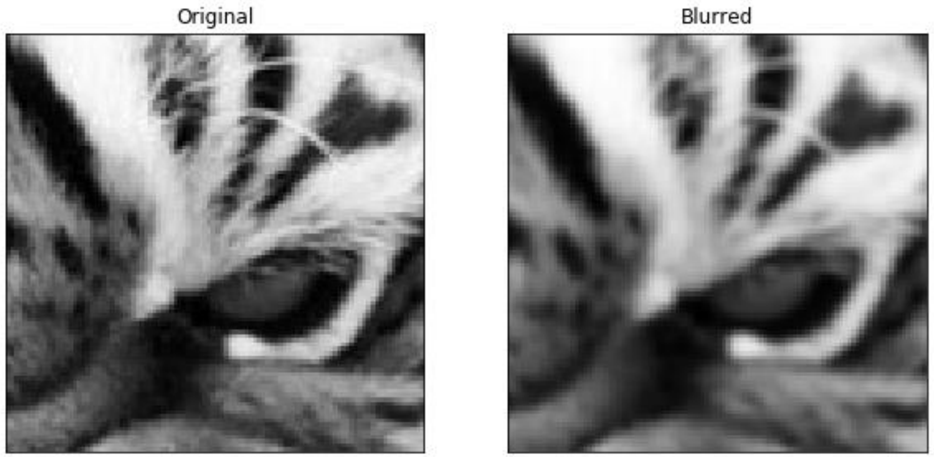


Convolution (2)

- The choice of kernel depends what the output image will represent
  - Some ideas here

```
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()
```

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

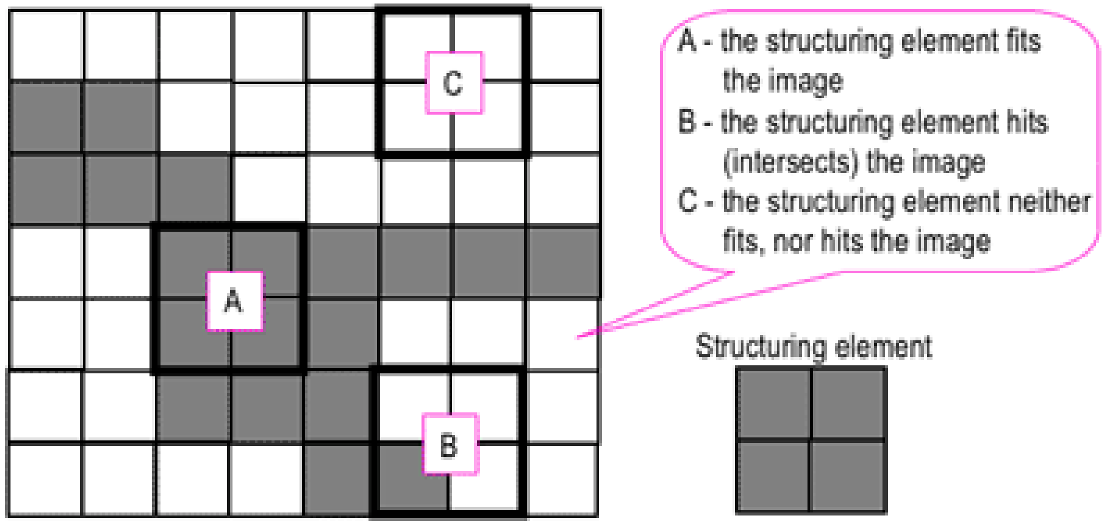
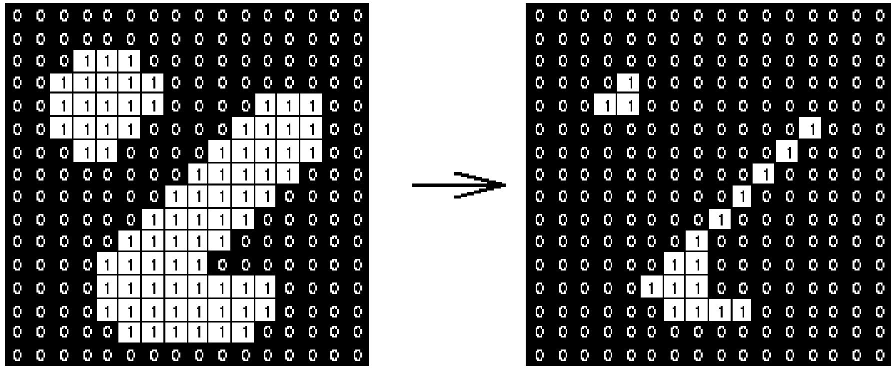


Image Morphology (2)

- 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
  - Adds a layer of pixels
  - Gaps become smaller
  - Small gaps are filled in

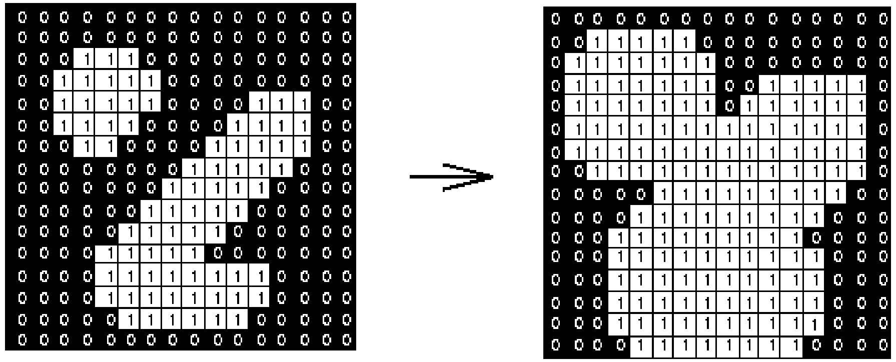
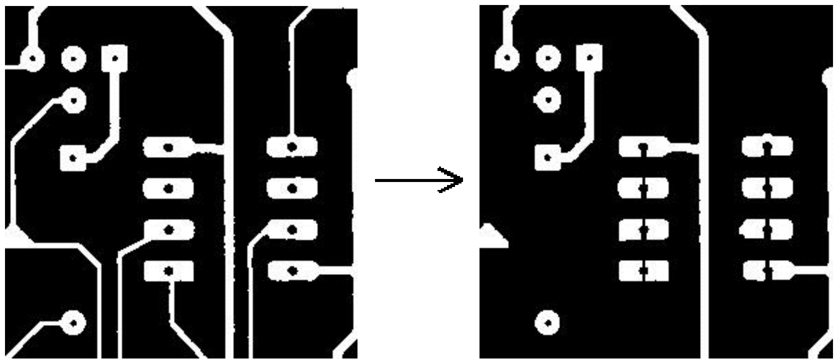
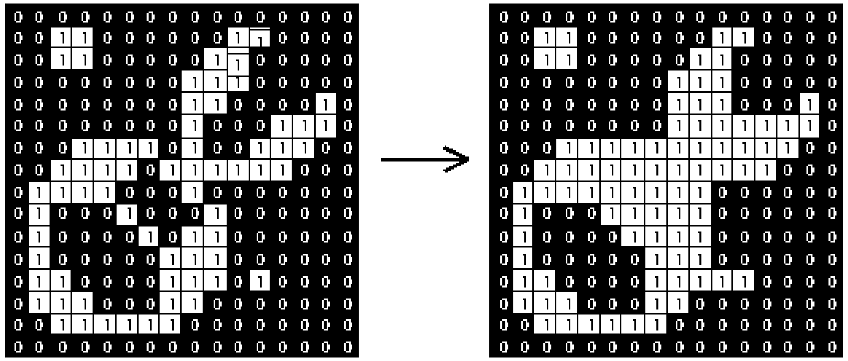


Image Morphology (3)

- **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



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

Text Processing

Understanding what people write

Text Data

- Documents, written in plain text
  - News, tweets, blog posts, poems, books, legal documents, etc.
  - May also be auto-generated (i.e., server logs)
- Objective
  - Preprocess the text data so that it's structured
    - Algorithms can analyze a table of numbers, not plain text
      - This is especially true for machine learning algorithms
- Applications
  - Sentiment analysis
  - Grouping texts – similar topics, similar authors
  - Classification (e.g., spam / fake news prevention)
  - Text summarization, etc.

Character Frequencies

- Reading is simple: open the file, read it, close it

```
text = ""
with open("alice.txt", "r", encoding = "utf-8") as f:
    text = f.read()
print(len(text))
```

- A string is a collection of characters
  - There are several ways to count them, the easiest being by using a library: `collections.Counter`

```
from collections import Counter
char_counter = Counter(text)
```

  - Most common characters ("etaoin shrdlu")

```
char_counter.most_common(20)
```

  - Similarly, most common words: split by all non-word characters

```
import re
word_counter = Counter(re.split("\W+", text))
```

Preparing Text Data

- Before we start working with the text, we have to "normalize" and clean up the messy data
  - Remove all non-letter characters
    - Numbers, punctuation, whitespace, etc.
    - If needed, apply additional rules, e.g., if we're looking at tweets, **@mention** means a username and we may want to get rid of it
  - Transform all characters to lowercase
  - Remove "stopwords"
    - Words that are too frequent in all documents and don't contain much information such as "the", "a", "is", etc.
  - Perform stemming
    - Extract the stems of all words, e.g., "connected", "connection", "connecting" should all point to "connect"

Stopwords and Stemming: NLTK

- NLTK is a library for working with natural language
  - Contains all frequently used algorithms and corpora
  - Installation: as usual, using conda: `conda install nltk`
- Getting and removing stopwords
  - Download the words first

```
import nltk
nltk.download("stopwords")
from nltk.corpus import stopwords
stop = set(stopwords.words("english"))
sentence = "this is a foo bar sentence"
print([w for w in sentence.lower().split() if w not in stop])
```

- Stemming – Porter's algorithm (includes many "manual" rules)

```
from nltk.stem.porter import *
stemmer = PorterStemmer()
words = ["caresses", "flies", "dies", "seizing", "itemization",
"sensational", "traditional", "reference", "plotted"]
print([stemmer.stem(word) for word in words])
```

TF – IDF

- Term frequency – inverse document frequency
  - A common method to preprocess the text

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

  - High score: rare, specific words
    - Hypothesis: these may be better related to the topic
    - Note: This may also include misspelled words and / or names
  - Low score: words that occur in nearly all documents

Using TF – IDF

- Read the "20 newsgroups" dataset (from `scikit-learn`)

```
from sklearn.datasets import fetch_20newsgroups
# Download only some categories to speed up the process
newsgroups = fetch_20newsgroups()
```

- Initialize the algorithm (docs) and compute the matrix

```
tfidf = TfidfVectorizer(input = "content", analyzer = "word",
    ngram_range = (1, 4), min_df = 0, stop_words = stop, sublinear_tf = True)
tfidf_matrix = tfidf.fit_transform(newsgroups.data)
```

- Get all feature names

```
feature_names = tfidf.get_feature_names()
```

- Get the IDF for each word / n-gram in one document

```
doc = 0 # Change the index to view another document
feature_index = tfidf_matrix[doc, :].nonzero()[1]
tfidf_scores = zip(feature_index, [tfidf_matrix[doc, x] for x in feature_index])
for w, s in [(feature_names[i], s) for (i, s) in tfidf_scores]:
    print(w, s)
```

Summary

- Image processing
  - Reading, exploring, manipulation
  - Convolution
  - Image morphology
- Text processing
  - Text preparation
  - Frequency analysis
  - TF-IDF

Questions?