Week 1: Introduction to the basics

Efstratios Gavves [*]

October 29, 2013

1 Summary

1.1 Provided:

- Handout
- Template report
- Basic code and functionality
- Dataset

1.2 Requested:

- A report. The report should be written according to the template that is provided in the package you downloaded. **The questions which you need to answer in the report will be marked with red font in this handout.** The report should be delivered after week 2, together with the assignment of week 2. The **deadline** for delivering report 1 (for the assignments of week 1 and 2) is Wednesday 13th of November, 23:59:59. Delay in delivering the report will be penalized with **minus 1 point per day**.

- A zip file with the implemented code. For this week that is only one file, that is *week1.py*. The places where you implement code (this time in *week1.py*) will be marked with comments

  `[# WRITE YOUR CODE HERE].`

  You **should not** write code where there is a comment

  `[# DO NOT TOUCH]`

  We do so to ensure that the final code will look as clean as possible. That way we will use our time in evaluating your code and giving you feedback, and **not debugging it** or trying to understand it. We are not responsible for debugging, please make sure the code you send us works. If there are bugs you will be asked to re-submit once more **the most**.

[*]Email: `efstratios.gavves@gmail.com`, Room: C. 3.244, Office hours: Mon 17.00-18.00

1.3 Useful links:

- `https://www.enthought.com/` Canopy Enthough environment
- `http://effbot.org/imagingbook/pil-index.htm` Python Imaging Library Handbook
- `http://docs.scipy.org/doc/` Python Numpy and Scipy Library documentations
- `https://piazza.com/uva.nl/fall2013/uva/home` Piazza.com, the electronic embodiment of our class, where you can ask questions and we will answer. Asking questions here is good, as everybody can profit from the answer.

2 Basic image representations, Gaussian blurring, Gaussian derivatives

In the first week's assignment we will get more familiar with the Python Image Library, or for shorter *PIL*. PIL contains most of the functionalities that you are going to need regarding the manipulation of images. Another very, very important library is *numpy*. *Numpy* contains a bunch of useful matrix operations that you are going to need. We will also work on generating basic image representations, such as color histograms. Last, we will apply gaussian filters to blur images.

2.1 Part 1. Download the provided package for Week 1

First download the assignment package from

`http://staff.science.uva.nl/~gavves/files/multi_week1.zip`.

Extract the package in your working directory, e.g.,

`C:\Users\username\multimedia_course`

Let's call this directory from now on $main_dir. You should then have inside $main_dir three directories: *docs*, *data* and *python*.

In the directory *$main_dir\docs* you will find the file *assignment_week1.pdf*, which is a copy of this week's handout. Also, you will find a template file *report_week1.docx*, which you can use to compile the report for the assignment of week 1. (*Note, you will deliver the report after week 2 together with the assignment of week 2.*)

In the directory *$main_dir\python* you will have all your code for the course, inside subdirectories *week1*, *week2*, ... You can find the starting code for week 1, already in the subdirectory *week1*. It contains three python scripts, that is `tools.py`, `week1\week1.py` and `week1\main_script_week1.py`.

- The `main_script_week1.py` file is your sandbox file, where you will be guided through to complete the assignment. You can start writing your code here, once you verify it works, you can transfer it to your "library" file, that is `week1.py`.

- The `week1.py` file will be the "library" you are going to build with the functions you implemented during the assignment. The purpose of `week1.py` is to be able to reuse the functions that you implemented for the assignments later on during the course. So, after successfully writing some piece of code in the sandbox file, you will be asked to transfer it to `week1.py`. That way your code will be reusable and you will not need to reinvent the wheel every time. If you change the `week1.py`, the Canopy environment will not automatically be aware of the changed if you try to use. **To update Canopy's memory you should run in the command line the command**

  ```
  reload(week1)
  ```

- The `tools.py` file contains some useful functions that you will need and will be provided by the assistants. Example functions are for downloading the datasets or getting a list of the images in the dataset.

Finally, the directory *$main_dir\data* will be the place where your datasets will be stored. We have already implemented some python scripts so that the data are automatically downloaded and extracted to this directory. After downloading the data, feel free to browse the images and get a glimpse of the problem we are dealing with.

2.2 Part 2. Start the assignment

2.2.1 Step A Downloading the data

Open the `main_script_week1.py`. As you see in the top of the script, we import several libraries, amongst which also PIL. **Before you continue with the rest of the assignment, do not forget to change the working directory of your Canopy python environment to the** *$main_dir*. If not, the code will not work, as it contains relative paths to the files you are going to use.

Now, after having changed the working directory, you can check where the data is stored. The dataset is downloaded and extracted in *$main_dir\data\objects*. Today we will work with flower images, because we are in Holland :P

Goal At the end of this step you should have successfully located the data.

2.2.2 Step B.1-4 Reading an image

Read an image using the python provided tools in the `main_script_week1.py`. Display the image to make sure the installation of all components has been successful. You should get something like Fig. 1 for *1.jpg* in the flowers directory.

In Python your image is stored in the form of an array. For a colored image, this array will have 3 dimensions, that is the height, width and the number of color channels. Since our image is now essentially an array, we can treat it as such. For example, you can compute the dimensions of your image/array. Then, try to print some pixel values of your image/array.

Note that in Python the counting for the array starts from 0 and ends in the position $H-1$ for the height and $W-1$ for the width. If you call something

Figure 1: Showing an image in Python.

like `im[H][W]`, you will get an error, as you ask Python to return a value of the matrix that is located outside *the matrix.*

Goal At the end of this step you should be able to tell how big your image is.

2.2.3 Step C.1-3 Computing histograms

Now we will create the first representation for our flower image. *What is an image representation?* An image representation is a mathematical structure, that *a)* computers can understand and *b)* summarizes what an image shows. One of the most convenient representations is the *histogram*.

A histogram is composed of bins, which get increased by *+1* for every time a certain observation is made. In our case, and since we have flowers of many different colors, our histograms will "count colors". Our colored image contains three color channels, that is "red", "green" and "blue". Hence, we are going to compute one histogram per color channel. To define a histogram, we need to define its bins. For example, let's take the "red" channel. We define the "red histogram" to have as many bins as the different values of red we can get. For a standard image these values are in the range $[0, 255]$, hence we define one bin for every of these values. Whenever such a red color value is found in the image, we add "+1" in the respective bin for that color channel. A visual explanation on how to compute a color histogram is given in Fig. 2.

We start from the red channel. Get all the pixel values for the red channel by using the command `im[:, :, 0]`. The ":" symbol asks from Python to return *all* the values from that dimension. You can compute the histogram of such an array using the `bincount` function from the `numpy` library. This function takes two inputs. The first is a 1-D array, or else a *vector*. Our red channel, however, is 2-D, hence we first need to vectorize it into a 1-D vector, by moving all columns one under the other. To do so you can use the `flatten` command, e.g., `myarray.flatten()`. The second argument is optional, if we want to apply certain weights to specific bins. If we do not want to have any special weights, as in our case, we say so by using as a second argument the *None* command. The third argument of the `bincount` is the number of bins. Since we have color images, the bins should span from 0 to 255, hence the third argument should

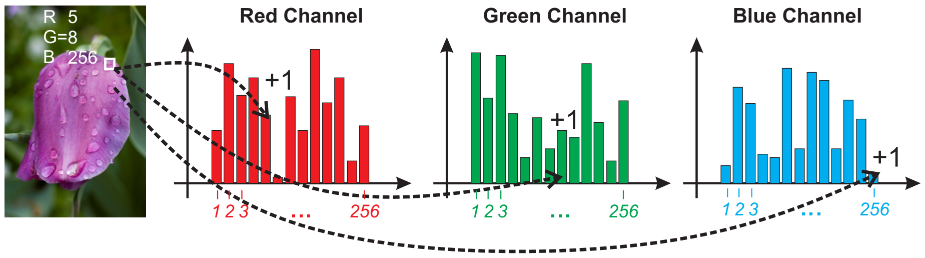

Figure 2: Computing a color histogram for the red, green and blue color channels.

be 256.

After you compute the histogram for the red channel, repeat for the green channel and for the blue channel. Then concatenate the histograms to a single one that contains all the elements in a single vector. This is our RGB color histogram representation for our flower image.

After having implemented the code for computing the histograms from an image, **transfer the code to the function**

```
def extractColorHistogram( im ):
    ...
    return histo
```

in `week1.py`. This function will take one input argument and will return one output argument: *im* will be the input image. The function should return the histogram representation of the image.

Goal At the end of this step you should be able to compute the color histogram of the flower image *1.jpg*. **Save the histogram and paste it in the report**.

2.3 Step D.1-6 Comparing histograms

Being able to compute a color histogram representation given an image, we can now compare different images using their respective histogram representations. For comparisons we need some distance function that takes as input two histograms and returns a single value that tells us how close the two histogram/images are. In computer vision there are some very popular distances functions, more specifically:

- **Euclidean**: $d(x, y) = \sum_i (x_i - y_i)^2$. The smaller the d, the more similar the histogram/vectors x and y.
- **ℓ_2**-distance: $d(x, y) = \sum_i x_i y_i$. The larger the d, the more similar the histogram/vectors x and y. The Euclidean and the ℓ_2 distances are closely connected. *Note here that ℓ_2-distance makes sense only for vectors whose length is equal to 1. We will elaborate on this later in the course.*
- **Histogram**: $d(x, y) = \sum_i min(x_i, y_i)$. The larger the d, the more similar the histogram/vectors x and y.

- $\boldsymbol{\chi}^2$-distance: $d(x,y) = \sum_i \frac{(x_i - y_i)^2}{x_i + y_i}$. The smaller the d, the more similar the histogram/vectors x and y.
- **Hellinger** distance: $d(x,y) = \sum_i \sqrt{x_i}\sqrt{y_i}$. The larger the d, the more similar the histogram/vectors x and y.

After having implemented all these distance functions, copy paste the code to the function

```
def computeVectorDistance( vec1, vec2, dist_type ):
    ...
    return dist
```

in `week1.py`. This function will take three input arguments and should return one output: *vec1* will be the first histogram/image, *vec2* will be the second histogram/image, and *dist_type* will be the type of distance measure you want to compute. The function should return as output the distance between vec1 and vec2.

Note that in the literature you may find also the concept of kernel measures *for comparisons between histograms. Kernels are closely related to distance measures, however, explaining them further is beyond the scope of the course.*

Goal At the end of this step you should be able to compare how similar are two vectors/images/histograms using the function `computeVectorDistance` you implemented.

2.3.1 Step E.1-4 Ranking images

Compute distances between images and rank them according to how similar they are. First, find out the file paths for all the flower images. Then use the functions you implemented to compute the histograms for the flower images. Then you can calculate the pairwise distances for all the pairs of images. Given a query image, you can sort all other images and find out the most similar ones. To sort the images according to their similarity, you can use the functions `sort` and `argsort` from the `numpy` library. Remember to sort them in ascending or descending order, depending on your distance measure. Naturally, the query image itself will be ranked as the most similar one. Last, you can visualize the results using the provided code.

After having implemented all these distance functions, **transfer the code** in `week1.py` (2 functions). The first function is

```
def computeImageDistances( image_paths ):
    ...
    return imdists
```

This function takes a single input, that is all the paths to the images we want to compare. The output is a square matrix, that contains all the pairwise distances between images.

The second function is.

```
def rankImages( imdists, query_id ):
    ...
    return ranking
```

and takes two input arguments. The first one is the matrix of distances, as computed from the function `computeImageDistances`, and the second is the id of the image we want to compare against. The output will be a ranked list of our images, based on their similarity to the query image.

Goal At the end of this step you should be able to rank the images according to how similar they are. **Save the figure of the 5 top ranked images for the flower pictures *5.jpg, 10.jpg, 15.jpg* and paste them in the report, together with your observations.** If the implementation is correct, the most similar images should have similar colors. For example for *1.jpg* the result should be something like Fig. 3

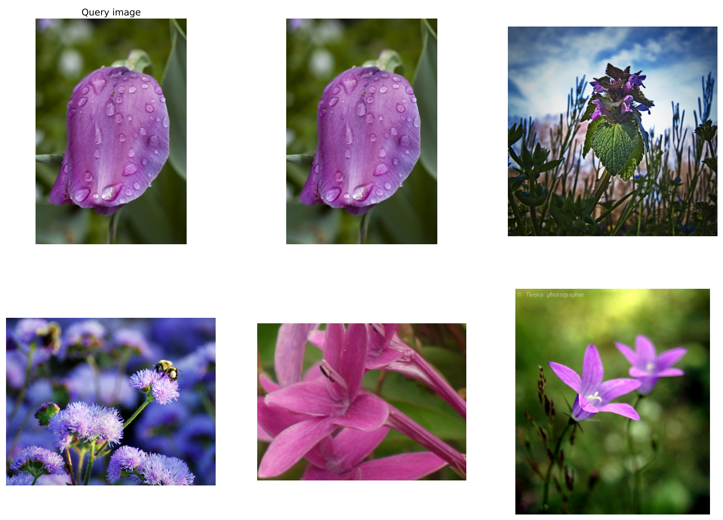

Figure 3: The most similar images for *1.jpg*.

2.3.2 Step F.1-6 Gaussian blurring: Convolution using Gaussian filters and Gaussian derivative filters

In the final part of the assignment you should implement a simple Gaussian filter. The filtering via convolution, as well the blurring is **already implemented** and you only need to report your observations.

First, you need to implement a Gaussian filter. To do so you need the implement the following function

$$G_\sigma = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{x^2}{2\sigma^2}) \quad (1)$$

Note that x stands for the displacement from the place where the filter is applied. Since we have relative coordinates for each pixel, the starting point given a pixel will be $(0,0)$. So, the x will range in the $[-d, d]$, where d is the **half size** of the filter. **Select the half size of the filter** to be $d = \lceil 3\sigma \rceil$, as we want the filter to contain about 99.7% of the energy of the gaussian function. Put the implementation of eq. (1) in `week1.py` in

```
def get_gaussian_filter( sigma ):
    ...
    return G
```

Apply the gaussian convolution for the image for $\sigma = 1, 5, 10$. To apply the convolution use the already implemented function `apply_gaussian_conv`. Check that your result is similar to the result using the Python's precomputed function. Save the results and paste them in your report. The result of the gaussian blurring for $\sigma = 10.0$ should be something like in Fig. 5.

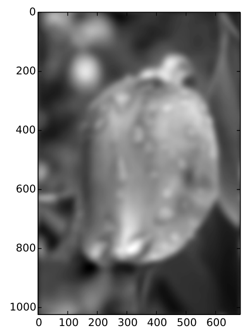

Figure 4: Blurring the image with gaussian convolution using $\sigma = 10.0$.

After having implemented the gaussian filter, you can easily implement also the derivatives of the gaussian. The first order derivative is

$$\frac{d}{dx} G_\sigma = -\frac{x}{\sigma^2} G_\sigma \qquad (2)$$

that is you can use the gaussian function you just implemented. Put the implementation of the first order derivatives in `week1.py` in

```
def get_gaussian_der_filter(sigma, order):
    sigma = float(sigma)
    acme = # WRITE YOUR CODE HERE

    x = np.arange(-acme, acme + 1)
    if order == 1:
        # WRITE YOUR CODE HERE
    elif order == 2:
        # WRITE YOUR CODE HERE

    return dG
```

where the second argument, *order*, defines the order of the derivative. In the current, you only need to implement the first order derivatives. This gaussian filters (the gaussian, and the gaussian derivatives) are merely a 1D vector. Hence, we can convolve the image with these filters along the row direction (i.e., row-wise) or along the column direction (i.e. column-wise).

Apply the first order gaussian derivative filters for the image for $\sigma = 1.0, 5.0, 10.0$ along the row and column directions. To apply the filters use the already implemented function `apply_filter`. Save the figures and paste them in the report. The result for the first order gaussian derivatives along the row and the column direction should look something like

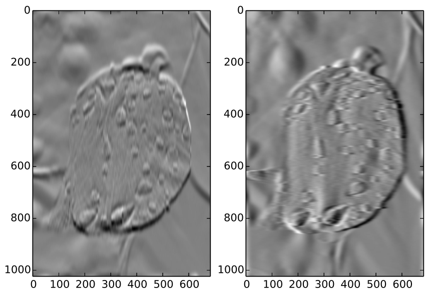

Figure 5: Getting the image gradients using gaussian derivatives with $\sigma = 10.0$.

After you have calculated the first order derivatives, you can now compute the magnitude of the gradient along the rows and columns. To compute the magnitude, you need to implement the function `gradmag` in `week1.py`, that is

```
def gradmag(im_dr, im_dc):
# WRITE YOUR CODE HERE
    return im_dmag
```

Compute the magnitude of the gradient for $\sigma = 1, 5, 10$. Note that the magnitude of the gradient kind of concentrates around the edges of the image. Save the figures and paste them in your report. Propose a way one can get just hard edges, so that you have a value 1 when the pixel lies on an edge and 0 otherwise. The result of the magnitude of the gradient should look something like in Fig. 6

An interesting case is when we have an impulse image. Impulse image is an image where all pixels are set to 0, except for the central one that is set to 1. Apply the gaussian filter convolution from above also for the impulse image and visualize the result. Compare it to the visualization of the 2d gaussian filter (**already implemented**). **Compare the visualizations. Save the figures and paste them in the report. What kind of relation do you observe between the gaussian 2d filter and the result of the convolution of the impulse image with the gaussian derivative? Report your findings.**

Goal At the end of this step you should have implemented a gaussian, a first order gaussian derivative and a second order derivative filters. The implementa-

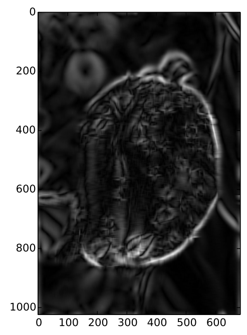

Figure 6: The magnitude of gradients for $\sigma = 10.0$.

tion should be in `get_gaussian_filter` and `get_gaussian_der_filter`. Based on these implementations you should report results in your report.

3 Bonus

As a bonus, we would like you to explain how, starting from eq. (1) of the gaussian, we arrive at its derivative of eq. (2). Furthermore, make the calculations for obtaining the second order derivative. Implement the second order gaussian derivative. and update the `get_gaussian_der_filter` accordingly. After you have implemented the second order gaussian derivatives, please visualize the result along the column and row direction for $\sigma = 1.0, 5.0, 10.0$.