SAPIENZA UNIVERSITY OF ROME

Advanced Machine Learning
Summer Semester 2020

Prof. Fabio Galasso <galasso@di.uniroma1.it>

Exercise 1: Image Filtering and Object Identification

In this exercise you will first familiarise yourself with basic the image filtering routines of Python and NumPy (**Question 1**), then develop a simple image querying system which accepts a query image as input and then finds a set of similar images in the database (**Question 2, 3 and 4**).

Download the zip file. The file consists of two directories: Filtering and Identification. Each directory contains a placeholder script (`filter.py` and `identification.py`, respectively) for functions you will have to implement in this exercise; your task is to fill the missing code corresponding to each subproblem and produce brief reports on the results whenever necessary.

The filtering part contains two images: `graf.png` and `gantrycrane.png`, which we will use for testing purposes.

In the identification part, you will compare images with several distance functions and evaluate their performance in combination with different image representations. The identification part contains query and model images for the evaluation, which correspond to the same set of objects photographed from different viewpoints. The files `model.txt` and `query.txt` contain lists of image files arranged so that i-th model image depicts the same object as i-th query image. The placeholder scripts will also be used to test your solution.

As part of the exercises, write a report to answer the questions in this exercise. Specifically, questions indicated by `report` should be answered in a separate report file - preferably a pdf format file. After you have implemented all the missing code, in both exercise parts you should be able to execute those without errors. Use those to support your answers in the report.

The completed exercise should be handed in as a `zip`, `tar.gz` or `rar` format which compresses all the code + the `report`.

Submit it by sending an email to Prof. Fabio Galasso `galasso@di.uniroma1.it` including the report and the completed script files **by Wednesday March 18th, 23:59**.

Question 1: Image Filtering (10 points)

Refer to the placeholder script `filter.py`.

a) Implement a method which computes the values of a 1-D Gaussian `Gx` for a given standard deviation σ:

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

The method should also return a vector `x` of values on which the Gaussian filter is defined: integer values on the interval $\left[-3\sigma^2, 3\sigma^2\right]$.

Open the file `gauss_module.py` with your preferred editor and complete the script:

```
def gauss(sigma)
    ...
    ...
    return Gx, x
```

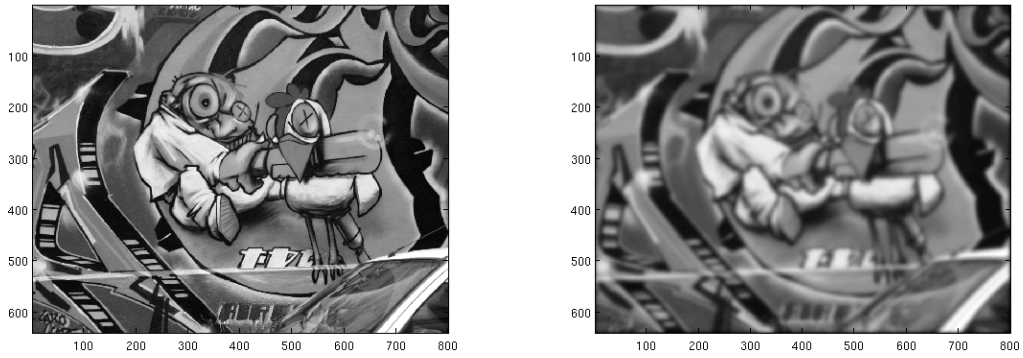
(2 points)

b) Implement a 2D Gaussian filter in `gauss_module.py`. The function should take an image as an input and return the result of convolution of this image with 2D Gaussian kernel. See Fig. 1 for illustration of Gaussian filtering. You can take advantage of the Python's `convolve2D` function if you don't want to implement convolution yourself.

Open the file called `gauss_module.py` with your preferred editor and complete the script:

```
def gaussianfilter(img,sigma)
    ...
    ...
    return smooth_img
```

Use the fact that 2D Gaussian filter is separable to speed up computations and obtain full points for this exercise. (2 points)



(a) (b)

Figure 1: (a) Original image (b) Image after applying a Gaussian filter with $\sigma = 4.0$.

c) Implement a function `gaussdx.py` for creating a Gaussian derivative filter in 1D:

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

$$= -\frac{1}{\sqrt{2\pi}\sigma^3}x\exp(-\frac{x^2}{2\sigma^2}) \quad (3)$$

As in 1a, the method should also return a vector `x` of values on which the Gaussian derivative filter is defined: integer values on the interval $\left[-3\sigma^2, 3\sigma^2\right]$.

Open the file `gauss_module.py` with your preferred editor and complete the script:

```
def gaussdx(sigma)
    ...
    ...
    return Dx, x
```

(2 points)

d) Turn to the file `filter.py` and prepare a written report for this question.

The effect of applying a filter can be studied by observing its *impulse response*. Execute the code in `filter.py` to create a test image in which only the central pixel has a non-zero value:

```
img_imp = np.zeros([27,27])
img_imp[13, 13] = 1.0
```

Now, create the following 1D Gaussian and Gaussian derivative kernels, `Gx` and `Dx` respectively.

```
sigma = 7.0
[Gx, x] = gauss_module.gauss(sigma)
[Dx, x] = gauss_module.gaussdx(sigma)
```

What happens when you apply the following filter combinations? (Consider the corresponding subplot figure in `filter.py`.)

1. first `Gx`, then `Gx`^{T}
2. first `Gx`, then `Dx`^{T}
3. first `Dx`^{T}, then `Gx`
4. first `Dx`, then `Dx`^{T}
5. first `Dx`, then `Gx`^{T}
6. first `Gx`^{T}, then `Dx`

(`report`, 2 points)

e) Turn now to `gauss_module.py` and complete the method `gaussderiv`, to return two copies of the input image, smoothed according to the standard deviation σ and derived in the directions x and y respectively:

```
def gaussderiv(img, sigma)
    ...
    ...
    return imgDx, imgDy
```

Apply the method to the provided example images `graf.png` and `gantrycrane.png` and comment on the output in your report. Consider also why smoothing an image is important before applying the derivative filter.

(`report`, 2 points)

Question 2: Image Representations, Histogram Distances (10 points)

a) Turn to the script `identification.py`, which showcases the context for all functions and modules of this second Identification part. For this question, consider the module `normalized_module` and implement the function `normalized_hist`, which takes gray-value image as input and returns *normalized* histogram of pixel intensities. When quantizing the image to compute the histogram, consider that pixel intensities range in $[0, 255]$.

Compare your implementation with built-in Python function `numpy.histogram`. Your histograms and histograms computed with Python should be approximately the same, except for the overall scale, which will be different since `normalized_hist` does not normalize its output. (3 points)

b) Implement the histogram types also missing within the `normalized_module`. Your implementation should complete the scripts provided in the functions `rgb_hist`, `rg_hist` and `dxdy_hist`.

As above, when quantizing the images to compute the histogram, consider that "RGB" and "RG" range in $[0, 255]$. For the case of `dxdy_hist`, the actual image value ranges depend on the chosen Gaussian filtering variance, used when computing the derivative. For this exercise, set the standard deviation $\sigma = 3.0$ and cap the pixel values to be in the range $[-6, 6]$. (3 points)

c) Implement the histogram distance functions within the `dist_module`. In more details, implement the function `dist_intersect`:

$$\bigcap(Q, V) = \frac{1}{2}\left(\frac{\sum_i \min(q_i, v_i)}{\sum_i \min q_i} + \frac{\sum_i \min(q_i, v_i)}{\sum_i \min v_i}\right) \tag{4}$$

the `dist_l2` function:

$$d(Q, V) = \sum_i (q_i - v_i)^2 \tag{5}$$

and the `dist_chi2`:

$$\chi^2(Q, V) = \sum_i \frac{(q_i - v_i)^2}{q_i + v_i} \tag{6}$$

(4 points)

Question 3: Object Identification (10 points)

a) Having implemented different distance functions and image histograms, we can now test how suitable they are for retrieving images in query-by-example scenario. Implement a function `find_best_match`, in `match_module.py`, which takes a list of model images and a list of query images and for each query image returns the index of closest model image. The function should take string parameters, which identify distance function, histogram function and number of histogram bins. See comments in the beginning of `find_best_match` for more details. Additionally to the indices of the best matching images your implementation should also return a matrix which contains distances between all pairs of model and query images. (3 points)

b) Implement a function `show_neighbors` (in `match_module.py`) which takes a list of model images and a list of query images and for each query image visualizes several model images which are closest to the query image according to the specified distance metric. Use the function `find_best_match` in your implementation. See Fig. 2 for an example output. (3 points)

c) Use the function `find_best_match` to compute recognition rate for different combinations of distance and histogram functions. The recognition rate is given by a ratio between number of correct matches and total number of query images. Experiment with different functions and numbers of histogram bins, try to find combination that works best. Submit the summary of your experiments in a `report` as part of your solution. (`report`, 4 points)

Figure 2: Query images in the first column and respectively retrieved model images on their right, i.e. the 5 closest pictures according to the intersection distance between RG-histograms with 30-binned histograms (the titles report the distance).

Question 4: Performance Evaluation (bonus of 10 points)

a) Sometimes instead of returning the best match for a query image we would like to return all the model images with distance to the query image below a certain threshold. It is, for example, the case when there are multiple images of the query object among the model images. In order to assess the system performance in such scenario we will use two quality measures: *precision* and *recall*. Denoting threshold on distance between images by τ and using the following notation:

TP = number of correct matches among images with distance *smaller* then τ,
FP = number of incorrect matches among images with distance *smaller* then τ,
FN = number of correct matches among images with distance *larger* then τ,

precision is given by

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{7}$$

and recall is given by

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{8}$$

For an ideal system there should exist a value of τ such that both precision and recall are equal to 1, which corresponds to obtaining all the correct images without any false matches. However in reality both quantities will be somewhere in the range between 0 and 1 and the goal is to make both of them as high as possible.

Implement a function `plot_rpc`, defined in `rpc_module.py`, where you compute precision/recall pairs for a range of threshold values and then output the precision/recall curve (RPC), which gives a good summary of system performance at different levels of confidence. See Fig. 3 for an example of RPC curve. (5 points)

b) Plot RPC curves for different histogram types, distances and number of bins. Submit a summary of your observations as part of your solution. (`report`, 5 points)

Exercise credits: Prof. Bernt Schiele and Prof. Mario Fritz, Max Planck Institute for Informatics.

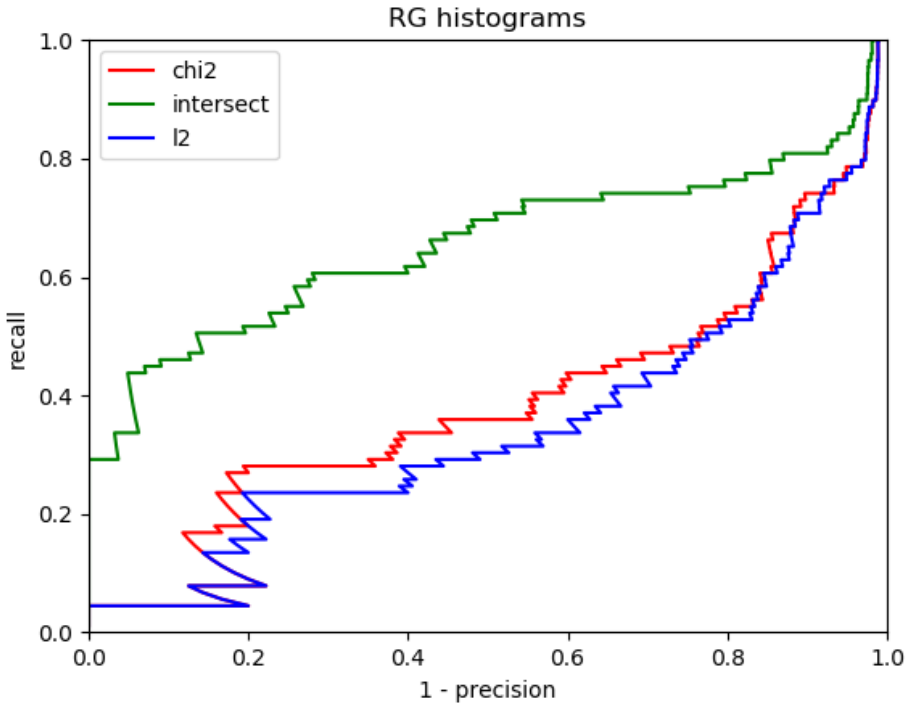


Figure 3: Recall/precision curve evaluated on the provided set of model and query images, when using RG histograms.