Prof. Bernt Schiele, Dr. Mario Fritz
<{schiele, mfritz}>@mpi-inf.mpg.de

Rakshith Shetty, Sahar Abdelnabi
rshetty@mpi-inf-mpg.de, sahar.abdelnabi@cispa.saarland

Exercise 1: Image Filtering and Object Identification

In this exercise you will first familiarise yourself with basic image filtering routines. In the second part, you will develop a simple image querying system which accepts a query image as input and then finds a set of similar images in the database. In order to compare images you will implement some simple histogram based distance functions and evaluate their performance in combination with different image representations.

The zip file you unpacked contains the code directory with two sub-directories - filtering and identification. Each directory contains a placeholder script (`filter.py` and `identification.py`, resp.) 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 three images: `graf.png`, `kand.png` and `night.png`, which we will use for testing purposes. The identification part contains *query* and *model* images, which will be used to evaluate your implementation. The model and query images 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. Ideally, after you have implemented all the missing code you should be able to execute it without errors.

In your submission please only submit the edited code files and not the model and query images.

Question 1: Image Filtering (10 points)

a) Implement a method which computes the values of a 1-D Gaussian for a given variance σ^2. The method should also give a vector of values on which the Gaussian filter is defined: integer values on the interval $[-3\sigma, 3\sigma]$.

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

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

```
def gauss(sigma):
    ...
    ...
    return G,x
```

b) Implement a 2D Gaussian filter in `gauss_module.py`. The function should take an image as an input and return the result of the convolution of this image a with 2D Gaussian kernel of given variance σ^2. See Fig. 1 for an illustration of Gaussian filtering. You can take advantage of the `convolve2d` function from the scipy library if you don't want to implement convolution yourself.

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

```
def gaussianfilter(img,sigma):
    ...
    ...
    return outimg
```

Hint: use the fact that the 2D Gaussian filter is separable to speed up computations.

c) Implement a function `gaussdx` for creating a Gaussian derivative filter in 1D according to the following equation

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

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

The effect of applying a filter can be studied by observing its so-called *impulse response*. For this, create a test image in which only the central pixel has a non-zero value:

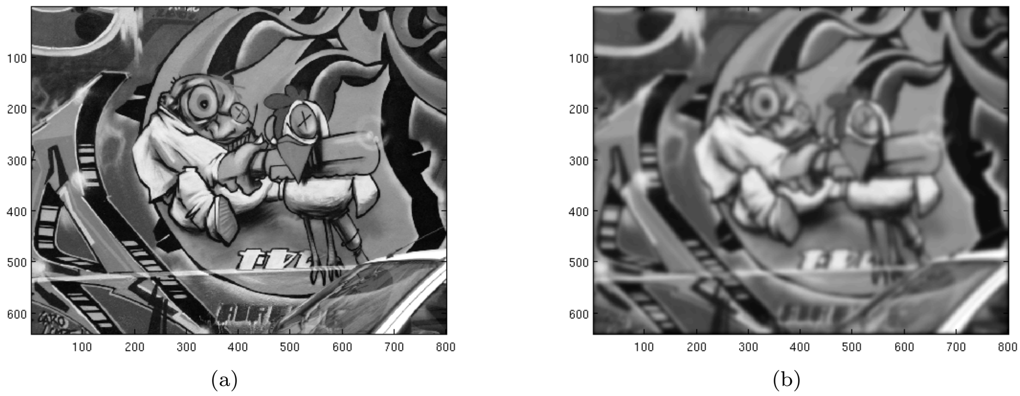

Figure 1: (a) Original image (b) Image after applying a Gaussian filter with $\sigma = 4.0$.

```
imgImp = np.zeros((27,27))
imgImp[14,14] = 1.0
```

Now, create the following 1D filter kernels `G` and `D`.

```
sigma = 5.0
G = gauss(sigma)
D = gaussdx(sigma)
```

What happens when you apply the following filter combinations?

1. first G, then G^T.
2. first G, then D^T.
3. first D, then G^T.
4. first G^T, then D.
5. first D^T, then G.

where G^T refers to the transpose of vector G. Visualize the results and put them in your report.

d) Use the functions `gauss` and `gaussdx` in order to implement a function `gaussderiv` that returns the 2D Gaussian derivatives of an input image in x and y direction. Try the function on the three test images and comment on the output. Visualize the results and put them in your report.

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

a) Implement a function `normalized_histogram`, which takes a gray-value image as input and returns a normalized histogram of pixel intensities. Compare you implementation with the built-in Python function `numpy.histogram`. Your histograms and the histograms computed with Python should be approximately the same, except for the overall scale, which will be different since `numpy.histogram` does not normalize its output.

b) Implement other histogram types discussed during the tutorial (refer intro slides). Your implementation should extend the code provided in the functions `rgb_histogram`, `rg_histogram` and `dxdy_histogram` (in `histogram_module.py`). Make sure that you are using the correct range of pixel values. For "RGB" the pixel intensities are in $[0, 255]$, for "rg" the values are normalized to be in $[0, 1]$. For the derivatives histograms the values depend on the σ^2 of the Gaussian filter; with $\sigma = 7.0$ you can assume that the values are in the range $[-32, 32]$.

c) Implement the histogram distance functions discussed during the tutorial (refer intro slides), by filling the missing code in the functions `dist_l2`, `dist_intersect`, and `dist_chi2` (in `dist_module.py`).

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 a 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 the closest model image. The function should take string parameters, which identify the distance function, the histogram function and the number of histogram bins. See the comments at the beginning of `find_best_match` for more details. Aside from the indices of the best matching images, your implementation should also return a matrix which contains the distances between all pairs of model and query images.

(a) (b)

Figure 2: Query image (a) and the model images with color histograms similar to the query image (b).

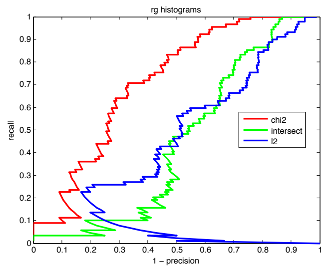

Figure 3: Recall/precision curve evaluated on the provided set of model and query images.

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 the 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.

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

Question 4: Performance Evaluation (10 points)

a) Sometimes instead of returning the best match for a query image we would like to return all the model images with a 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 the threshold on the distance between the images by τ and using the following notation:

   TP (True Positive) = number of correct matches among the images with distance *smaller* than τ,
   FP (False Positive) = number of incorrect matches among the images with distance *smaller* than τ,
   TN (True Negative) = number of incorrect matches among the images with distance *larger* than τ,
   FN (False Negative) = number of correct matches among the images with distance *larger* than τ,

   precision is given by

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{4}$$

   and recall is given by

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{5}$$

   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` that you have to 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.

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

Please turn in your solution by uploading it to the cms website. Maximum upload size is 10 MB, please make sure you remove the images before zipping the code directory. The deadline for submission is Sunday, May 17th, 23:59.