Intelligent and Interactive Systems
Spring 2017

Tutorial 1: OpenCV

OpenCV is an open source package for computer vision with C++ and Python bindings. Most of the learning paradigms require some basic pre-processing of image and video data before we can perform recognition or training. This tutorial provides and overview of such methods and is in no way exhaustive. During the course of the tutorials you will be working with some practical aspects of such algorithms and will be introduced to online resources that can be used to further enhance your learning and prepare you for the assignments.

Every part of the exercise has some instructions, and sometimes also some questions. Read these **before** *actually performing the part. Answer the questions explaining* **why** *you got a particular result, rather than* **how**. *If you get stuck, do not spend too much time on a particular problem, it is better to continue and ask us later on. These exercises are only here to set you up for the assignments and project and hence there is no need to hand in your solutions (you can still, of course, ask for feedback).*

Formality

- You need an account for the computer systems at the Dept. of Information Technology.
- You should work together in groups of two people. It may be advantageous to have someone to discuss issues with.
- The tutorials are not mandatory but you are highly recommended to take them. The topics in the tutorials will not only complement the learning during the lectures but will prove directly useful during the assignments and project.

Getting the material

In a lab at the Dept. of Information Technology running Ubuntu 14.04:

- Log on to one of the workstations using thinClient
- Open a terminal (keyboard shortcut `ctrl+alt+T`)
- Run the command `module load opencv` in the terminal.
- This makes the OpenCV module available to all the process within this terminal environment.
- A useful IDE (Integrated Development Environment) for developing and debugging in Python is Spyder it can launched by using the command
  `spyder&` (loading can be slow so wait for it)

Download the lab tutorial material from the Student portal, 'Document'→ 'Lab01'

1 Image Processing

In this section some basic image processing techniques will be considered. If you want to go deeper, pick up any book on image analysis (e.g. [4] or [3]) or read up on wikipedia. The main point of the following is to train you in image I/O and memory representation. In most IDEs there is some way of inspecting data. Look closely at the image data as represented in matrix form. Most data output from OpenCV will be in NumPy array format (i.e. not python lists). A common source of error is that the input data is not in the expected format/data type. Come back to this regularly until you know what to expect. Note that OpenCV, at least on some platforms, reads the images as BGR and not RGB.

The file `imgHandle.py` has the basic Python code snippet to get you started on the following tasks. You can invoke it by the following command.

```
python imgHandle.py <input_image_file> <options>
```

We will be implementing the following options:

`c2g`, for converting color image to grayscale.
`bw`, for converting a grayscale image to black and white.
`resz`, for resizing the image by a scale factor.
`edge`, detecting edges in an image.

We will be using `itc.JPG` as the input file. For converting it to grayscale image use:
`python imgHandle.py itc.JPG c2g`

Color to grayscale conversion

In The file `imgHandle.py, rgb2grayFunc` is called through the option `c2g`. The code function does nothing much right now other than reading an image and writing it back to `rgb2gray.png` file. We begin with the color space conversion here, use the `cv2.cvtColor` function for successful conversion as shown here:

```
if(len(img.shape)==3):
   img = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
```

1. ***What is the purpose of the*** `if` ***condition in the above code?***

Image resizing

In The file `imgHandle.py, resizeFunc` is called through the option `resz`. As in the previous example it start by grayscale conversion. The scale factor can be passed to the function as a parameter the following code snippet guides you for rescaling

```
height, width = img.shape[:2]
res = cv2.resize(img,(int(scale*width), int(scale*height)), interpolation = cv2.INTER_CUBIC)
```

Control horizontal and vertical scaling seperately is possible here, and can be experimented with.

2. ***Explain the need for explicit type casting to*** `int`***?***
3. ***Similar results can be obtained with***
   `res = cv2.resize(img,None,fx, fy, interpolation = cv2.INTER_CUBIC)`
   can you get it to work?

Edge detection

In The file `imgHandle.py, edgeDetection` is called through the option `edge`. The edges are detected through Canny edge detector.

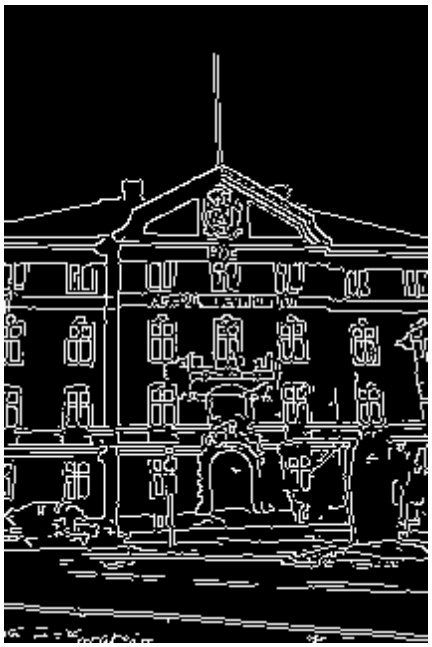

Figure 1: Edges after clearing up.

4. ***Try modifying the*** `thi,tlo` ***values to match the image above?***

2 Tracking

In this part you will be tracking the face of the animatied character *Asterix* in the videos provided using the **CamShift** algorithm. This require handling video formats in OpenCV. The point of this part is to learn how to used the tracking capabilities of OpenCV. The following algorithms are significantly behind the state-of-the-art but are relatively simple to use due to them being well implemented. If you are interested in the latest developments, try to find the web site of an academic conference lite CVPR[1] or

[1] `http://www.pamitc.org/cvpr15/`

ECCV[2]. Conferences in computer vision usually have lots of paper presentations and keynotes available as videos.

Video I/O

Use the `vidCap.py` for this task. Any video file requires a code for the encoding format within the video container file we use *MPEG4* compression which is given in the fourcc code.
`fourcc = cv2.VideoWriter_fourcc('X','2','6','4')`
A video write stream is opened with output file name, fourcc code, display frame rate (frames per second), video dimensions.
`outVideo = cv2.VideoWriter('output.mp4',fourcc, 20.0, (480,360))`

`ret ,frame = cap.read()` and `outVideo.write(frame)`
are used for reading and writing respectively.

5. ***Use the*** `cv2.resize` ***function to change the video format to*** 1024×640.
6. ***Observe that the output video is slightly delayed (duration is longer than the input file). Correct this problem.***

MeanShift/CamShift Algorithm

For this part of the tutorial, a modified version of the MeanShift algorthm [2] called the CamShift[1] will be used[3]. The camshift algorithm depends on "attractors" in the image. Hence, some feature matching scheme is needed. Here, histogram backprojection[5] will be used. A user of the algorithm defines an area to track, the algorithm then tries to follow a patch in the image with as similar colour properties as possible. The region of interest (ROI) will be rotated or scaled depending on what fits the expected colour distribution best.

Use the `camshift.py` for this task. The algorithm has been implemented here. However as you can see it does nothing significant here.

7. ***Use the first frame captured to set*** `roi` ***(region of interest) variable to surround the face of Asterix.***

Facial features detection using a cascading classifier

The current task will aim at detecting face and other facial features by constructing a Cascaded Classifier[6]. The idea behind the approach is to describe features of an object to be detected at various scales. For example a tree can be described at various scales such as `Tree -> Branches -> Leaves` all of which are visible at various levels of zoom, but with varying degree of observability. The information at each level of image scale can be captured into a feature that can be combined to detect the object. Another set of features which are also significant in detection are local features. The so called Haar-like[4] features used here are shown in figure 2.

Another central idea to the approach in [6] is the cascading of classifiers. Instead of using one classifier for object detection, running it on every candidate image patch, multiple simpler classifiers are trained. Every classifier in a cascade reduce the number of candidate patches and are trained to have as low false negative rate as possible. Running these computationally cheap classifiers in series reduces the number of candidates fast, allowing for real time face detection on older hardware (paper published in 2001). An illustration of the cascading is shown in figure 3. A crucial point here is that if your training data does not reflect all variance in a real world task, the task can't be properly trained for[5].

The file `faceHaar.py` has the basic Python code snippet to get you started on face detection.

[2] `http://eccv2014.org/`
[3] `https://en.wikipedia.org/wiki/Mean_shift`
[4] `https://en.wikipedia.org/wiki/Haar-like_features`
[5] `https://youtu.be/t4DT3tQqgRM`

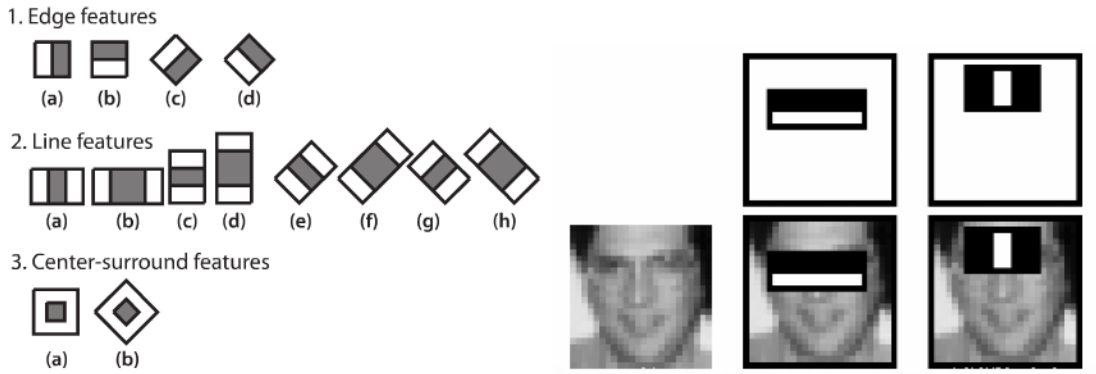

Figure 2: Haar-like features (left) and faces with feature detectors superimposed (right). Note how simple these features are. First, the pixel value in the light and dark regions are separately summed. Secondly, the areas in a full detector are summed but with the signed depending on the area colour (light is positive, dark is negative).

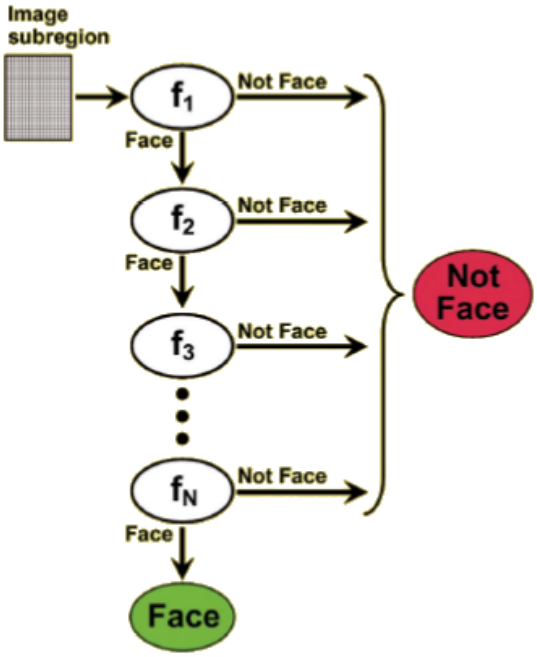

Figure 3: Illustration of how a candidate patch is inserted into the cascaded classifiers. Each stage focuses on excluding as many non target regions (in this case a face) as possible without throwing out any matches. This reduces the numbers of candidates fast with low complexity demands on each stage.

8. ***Experiment with the parameters for the cascade. What do they mean and what happens when you change them?***

Reading from a webcam is easy in OpenCV but falls outside the scope of this tutorial. Some code can be found in `camera.py`.

References

[1] G. R. Bradski. Real time face and object tracking as a component of a perceptual user interface. In *Applications of Computer Vision, 1998. WACV '98. Proceedings., Fourth IEEE Workshop on*, pages 214–219, Oct 1998.

[2] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40, Jan 1975.

[3] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing.* Prentice Hall, Inc, Upper Saddle River, New Jersey, 2nd edition, 2002.

[4] Milan Sonka, Vaclav Havlac, and Roger Boyle. *Image Processing, Analysis, and Machine Vision.* Thomson Learning, 3rd edition, 2008.

[5] Michael J. Swain and Dana H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[6] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.