Assignment 1: Image processing and object detection

Abstract

The aim of this lab assignment is to familiarize the students of the AIS course with the OpenCV programming library that includes a range of different computer vision algorithms and enables processing of computer images and videos in real time. Additionally, students get acquainted with two algorithms for detection of faces in images, one proposed by Paul Viola and Michael Jones and the other one proposed by Zhang *et al.* The former is based on integral images and haar cascades, while the latter uses a cascade of CNNs (Convolutional Neural Networks).

1 Programming environment

Before you start coding, set up your environment by following software requirements given in the virtual classroom eFE. For this exercise you will need Python 3, text editor of your choise (e.g. Visual Studio Code, PyCharm), OpenCV for Python and Tensorflow. If you choose to work on the computer in our faculty classroom, you can skip this step, as all above mentioned software is already installed.

Once you finish with installations, set up your own virtual Python environment using the Python module called *venv*:

https://docs.python.org/3/library/venv.html

In practice, complex development projects require a lot of packages and dependencies. Moreover, different models sometimes use different versions of the same library, hence it is useful to develop all your code inside a virtual environment with project-specific settings. Such approach isolates the Python interpreter, installed libraries and scripts, so they do not interfere with the global system installations. In other words, once the virtual environment is activated, the project you are working on becomes a self contained application, independent of the system installed Python and its modules.

New virtual environment can be created by executing:

python3 -m venv /path/to/new/virtual/environment --system-site-packages

For example, *python3 -m venv ais1 --system-site-packages* will create a virtual environment named *ais1* in your current directory. The last argument (--system-site-packages) will give this virtual environment access to the system site-packages. The virtual environment *ais1* can then be activated by:

source ais1/bin/activate

Once activated, it should look similar to the example on the image:

```
marija@evadesk:~/Documents/AIS$ python3 -m venv ais1 --system-site-packages
marija@evadesk:~/Documents/AIS$ ls
ais1
marija@evadesk:~/Documents/AIS$ source ais1/bin/activate
(ais1) marija@evadesk:~/Documents/AIS$
```

Now you can install project-specific or overwrite (upgrade/downgrade) existing libraries, by executing *python3 -m pip install name_of_the_library*.

You can exit the virtual environment by simply executing *deactivate*. To delete the environment execute *rm -rf ais1*.

2 Basic image processing (1 point)

Image processing is today used in a variety of different applications to enhance images or extract some useful information. In machine vision it is used to enhance image quality i.e. improve image contrast, illumination, for image sharpening and restoration etc. In this part of the assignment, you will test some basic image processing techniques using OpenCV library. For more information about OpenCV (installation instructions and library documentation) refer to the following link:

https://pypi.org/project/opencv-python/

Start by downloading the python script named `AIS_assignment1.py` (it is available in the virtual classroom eFE). As you can see, this code imports OpenCV library in the first command line (`import cv2`) and calls OpenCV functions that activate computer's camera and acquire camera frames in real-time until the user terminates the program by pressing any key on the keyboard (Warning: To terminate the program, first click on the window where camera frames are streaming, then press any keyboard key. Otherwise, the program will not terminate). Current acquired camera image is returned in a variable called *frame*, which is then shown in a separate window and refreshed (replaced with a new image) every millisecond.
Append the baseline code such that it performs all of the following tasks. Results of each task should be visible in separate windows.

- Camera images are represented in RGB color space, where each color shade is defined as a combination of three primary colors - red, green and blue. Convert camera frames from RGB to grayscale using OpenCV function `cv2.cvtColor()` and show them in a new window using function `cv2.imshow()`.
- Improve the contrast of the grayscale frames obtained in the previous task with histogram equalization (`cv2.equalizeHist()`).
- Test the effect of two different low pass filters used for image smoothing - Gaussian filter (`cv2.GaussianBlur()`) and bilateral filter (`cv2.bilateralFilter()`). Image smoothing is often used as noise removal technique. Noise might make it difficult for edges (or objects in general) to be detected, thus it is essential to remove it to the highest degree possible, before proceeding with other image processing algorithms. Compare results of the two filters proposed in this task. Do you see any differences?

- Find edges in grayscale camera frames first by implementing Sobel operator (`cv2.Sobel()`) then repeat the task using Canny edge detector (`cv2.Canny()`). Show the results of both methods in separate windows and analyze them. What do you think, which edge detection method performs better and why?
- In some applications, we are interested in whole image areas representing some specific objects or image attributes we are looking for. The most simple method for image segmentation is image thresholding, which can be implemented using OpenCV function `cv2.threshold()`. Experiment with different threshold values applied to grayscale frames and show the results of the threshold value you have picked in a new window.

Hint: Take a look at OpenCV Python examples available in the official OpenCV documentation:

https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_tutorials.html

3 Face detection using OpenCV (2 points)

In this part of the assignment you will familiarize yourself with the Viola-Jones algorithm for fast and robust real-time object detection. Though it was primarily developed to be used for face detection, it can be trained for a variety of other object classes. You will test this algorithm using the original camera stream provided by the baseline code from the first part of the assignment.

Viola-Jones relies on the fact that all human faces share some similar properties: eyes are darker than the upper-chicks, eyebrows are darker than the surrounding area, the nose bridge is brighter than the eyes etc. All these features can be easily detected with different Haar-like filters (similar to Haar wavelets), combined into a classifier, which is trained to match them with potential faces. To speed up the search for faces, Viola-Jones implements cascades (series) of classifiers, each progressively more complex. The first in the row of classifiers is thus the weakest classifier whose primary function is to reject all obvious, non-face parts of the image. Although it significantly reduces the search area in the image, a lot of false positives are returned and the process is taken over by the next, more discriminative classifier, which continues the search only in the non-rejected areas.

Since training of haar cascades is relatively slow process and needs a lot of training data, you will use the pre-trained classifiers from the OpenCV library. Locate the root directory of the library (a folder named *opencv*) and navigate to the subfolder *haarcascades* (*opencv/data/haarcascades*). Here you can find all available cascades, trained to detect different objects. Choose one cascade for detection of frontal faces and one cascade for detection of profile faces.

In `AIS_assignment1.py` add the following command line

```
face_cascade=cv2.CascadeClassifier(cv2.data.haarcascades + haarcascade\_file\_name)
```

where you change 'haarcascade_file_name' with the exact name of the first haarcascade you have chosen. This will load the pre-trained model (stored in *face_cascade*), which can then be used for face detection by calling `face_cascade.detectMultiScale()`. Find out what kind of input parameters does `detectMultiScale()` expect and how do their values influence face localization.

Once all faces in the image are detected, the coordinates of the upper left corner of the bounding box of each face are returned along with it's width and height. Use these data points to draw a purple rectangle around each detected face. Repeat the task once more, this time using the other

haarcascade (the one trained on profile faces). Is the algorithm working properly? Is it able to find and track your face in the camera stream?

At the end, test both haarcascades using the image named *oscar.jpg* (it can be found in the virtual classroom on eFE). Outside the while loop in the baseline code, use function `cv2.imread()` to load the image. Convert the image into grayscale and run `detectMultiScale()` to get bounding boxes of celebrity faces. Draw rectangles around each found face in the same manner as you did it in the previous task (use `cv2.imshow()` followed by `cv2.waitKey(0)`). How many faces were you able to find with each of the two haarcascades?

Hint: Take a look at this OpenCV Python example for Viola-Jones face detector:

https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.htm

4 Face detection using MTCNN (2 points)

Although Viola-Jones's method remains one of the fastest algorithms for object detection, it often struggles with accuracy, especially in challenging data. Recent state–of–the–art face detectors are therefore based on various deep–learning algorithms, trained on huge amounts of face images.
In the last part of the exercise, you will test one of the most frequently used CNN face detectors MTCNN. To find out more about this model, read the paper by Zhang *et al.* available on the following link:

https://arxiv.org/abs/1604.02878

To test the model, first install MTCNN inside the virtual environment you have created, by executing *python3 -m pip install mtcnn*. The documentation for this library is available at https://pypi.org/project/mtcnn/.

Using this library, test both, real–time face detection and face detection in *oscar.jpg*. How well and how fast does it perform the detection task in comparison to the Viola–Jones detector?

Find answers to the following questions:

- Do your chosen haarcascades work well for all face poses?
- What kind of factors impact the success of face detection?
- What are the strengths and weaknesses of the Viola-Jones algorithm?