COMS30068 - Image Processing and Computer Vision with Coursework (2021-22)

Assignment Weeks 9-11
The No Entry Sign Challenge

Overview: This assignment is the summative piece of unit coursework for COMS30068 for those who elected to the CW version of COMS30030. It focusses on classical computer vision algorithms for object and shape detection. It runs over 3 weeks and must be completed individually.

Submission: Each student is required to upload their full piece of work (incl all source code files, and your 4-page -report PDF) as a single ZIP file to Blackboard under the submission point "Image Processing & Computer Vision CW" **FOR UNIT COMS30068** before the ***deadline at 13:00 on Friday 10th December 2021.*** Make sure you submit it an hour early or so (not last minute!) to avoid upload problems. **Even if your upload is a single second late you will immediately have 10% deducted (per day) from your mark by the system (a universal BB imposition - out of our control).**

Assessment: You will be marked on your code and report. Both elements must be submitted. Please do not attempt to copy code or report text, etc. Apart from the Hough transform functions, you are allowed to use any of OpenCV's functionalities – but, in any case, you must understand the algorithmic workings of the used functions. If you do use other code/text that is not your own, please declare this in your report with appropriate citation. **Note, plagiarism is taken very seriously based on university plagiarism policies and processes and plagiarism detection software will be used. Please avoid it**.

Managing your assignment: Please note that the coursework is akin to a Take-Home Exam. Therefore, very limited support can be provided in the **labs and the forum (**on **COMS30068** BB page, not COMS30030). There will be 1 hour of assignment support time per week during Weeks 9, 10, and 11, but the TAs cannot provide academic solutions to the coursework, as per CS Dept's instructions.

Task Overview: Detecting No Entry Signs ⛔

Introduction. Detecting (locating & classifying) instances of an object class in images is an important application in computer vision as well as an ongoing area of research. This assignment requires you 1) to experiment with the classical Viola-Jones object detection framework as discussed in the lectures and provided by OpenCV, and 2) to combine it with other detection approaches to improve its efficacy. Your approach is to be tested and evaluated on a small image set that depicts aspects of an important traffic sign – No Entry.

Basic Task Structure: The task consists of 4 subtasks which incrementally build upon each other:

- ***First Subtask*** ***(up to max 10 marks)***: This introductory part will ask you to run and understand the usage of the Viola-Jones detector as provided by OpenCV. In particular, you will experiment with the pre-built off-the-shelf frontal face detector and apply it so some example images. This part lets you gain experience on how and how well this classical detection framework can operate in practice.

- ***Second Subtask*** ***(up to max 20 marks):*** In this second task you will build your own object detector by training the Viola-Jones framework on a customised object class, that is `no entry signs'. You will then test the detector's performance and interpret your results on an image set. This part lets you gain experience on how to train and evaluate the Viola Jones framework in practice.

- ***Third Subtask*** ***(up to max 30 marks):*** In this subtask you will combine your detector with **your own implementation** of the Hough transform to detect component configurations of the shape of no entry signs in order to improve detection performance. (**Do not use OpenCV's implementations of the Hough transform!**) It is up to you to decide if you opt to implement the Hough transform for lines, circles, ellipses, your own parameterized shape or the generalised Hough transform or combinations of them for this task. This task will test your engineering skills of practically applying taught concepts and integrating them within an application context.

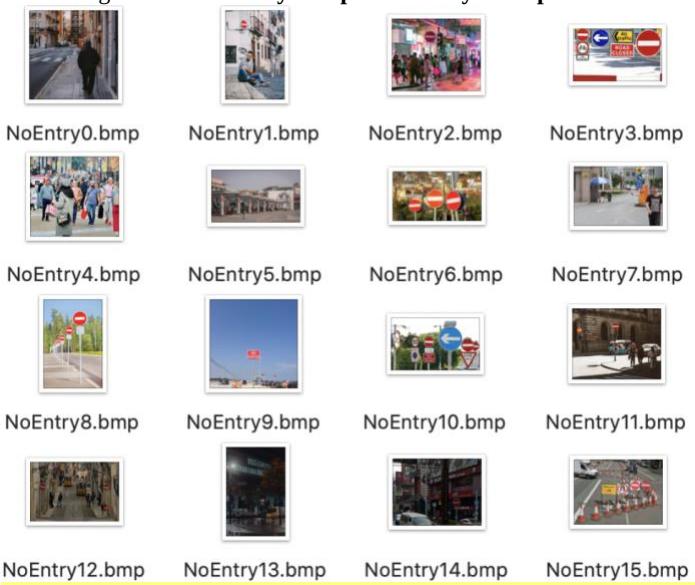
- ***Fourth Subtask*** ***(up to max 10 marks):*** In order to be able to reach marks close to and above the first-class boundary, we ask you to research, understand and use in OpenCV at least one other appropriate vision approach to further improve the detection efficiency of the system.

The ***final 30 marks for this coursework*** are reserved for excellence throughout your work, how well you inform your development on evidence and evaluation, and how well you demonstrate your comprehension of the techniques used. Here, we look for outstanding understanding, concise presentation, excellent write-up and clean implementation. For significant marks in this category, it should become clear in the 4-page report that the student has reached a level of understanding that allows for contextualisation and reflection well beyond the taught content. Very high marks in this category are reserved for work that is novel and publishable in principle.

To obtain a mark, both code and report elements must be submitted via the **Assessment and Submission menu on COMS30068**. A 5th page in the report is possible but ONLY for References/Citations/etc. **Strictly nothing on the 5th page will be used for marking, so please do not let your subtask descriptions spill onto page 5.**

Subtask 1: The Viola-Jones Object Detector

You are given some first sample code in the file **face.cpp** and 16 real-world images called **NoEntry0.bmp** to **NoEntry15.bmp**:



First, understand, compile and build the provided **face.cpp** source program to generate an executable. (You may need to adjust *include* paths if you use your own machine). On lab machines you would compile and build the provided file like this, providing all parameters in one single line:

g++ face.cpp /usr/lib64/libopencv_core.so.2.4 /usr/lib64/libopencv_highgui.so.2.4 /usr/lib64/libopencv_imgproc.so.2.4 /usr/lib64/libopencv_objdetect.so.2.4

Make sure the built program reads the provided XML file **frontalface.xml,** which contains a strong classifier trained using AdaBoost for detecting frontal human faces. The program also expects the name of an example image file to which it applies the classifier. For instance, given the image **NoEntry1.bmp** and the file **frontalface.xml** in the directory where you built and run your program in, you can run your compiled and built program as follows:

./a.out No_entry/NoEntry1.bmp

The program outputs the number of faces found to the console and the resulting detections are finally visualised in the produced output image called **detected.jpg**.

The First Page of your Report (strict limit):

a) ***GROUND TRUTH AND VISUALISATION:*** Manually annotate all the test images and generate ground truth in form of bounding box (x,y,width,height) coordinates for all valid frontal faces and store these annotations (e.g. in a file or array). Then test the detector's performance (with the given parameters as provided by **face.cpp**) on six given example images: **NoEntry1.bmp, NoEntry2.bmp, NoEntry4.bmp, NoEntry5.bmp, NoEntry7.bmp** and **NoEntry11.bmp**. Produce the six result images with bounding boxes of the ground truth (in red) and actually detected instances (in green) drawn around frontal faces and include them in your report.

b) ***IOU, TPR, F1-SCORE:*** Implement some code using a manually fixed threshold on the Intersection-Over-Union (IOU) between bounding boxes to judge which faces given the ground truth were successfully detected. Calculate and note in your report in table form the TPR (true positive rate) **for all test images,** that is the fraction of successfully detected faces out of all valid faces in an image. Discuss briefly in your report 1) some practical difficulties in assessing the TPR meaningfully, 2) why it is always possible to achieve a TPR of 100% on any detection task, and 3) implement a small piece of code to calculate the F1-score of your face detection system accurately and meaningfully for all test images given the ground truth and include it as a table in the report.

Subtask 2: Building & Testing your own Detector

The second subtask requires you to build an object detector that recognises no entry signs. The initial steps of this subtask introduce you to OpenCV's boosting tool, which you can use to construct an object detector that utilises Haar-like features.

In order to give you a smooth start, you are provided with readily compiled versions of two OpenCV tools you can use to build your object detector: **opencv_createsamples** and **opencv_traincascade**. (These binaries are part of all standard OpenCV installations.) You are also given **no_entry.jpg** containing a no entry sign that can serve as a prototype for generating a whole set of positive training images.

In addition, a large set of negative images (do not contain any no entry signs) is provided in a directory called `**negatives'** and a text file **negatives.dat** listing all filenames in the directory.

no_entry.jpg

First, create your positive training data set of 500 samples of no entry signs from the single prototype image provided. To do this, you can run the tool **opencv_createsamples** via the following single command if you use lab machines and execute it in a folder that contains the **negatives.dat** file, the **no_entry.jpg** image and the **negatives** folder:

./opencv_createsamples -img no_entry.jpg -vec no_entry.vec -w 20 -h 20 -num 500 -maxidev 80 -maxxangle 0.8 -maxyangle 0.8 -maxzangle 0.2

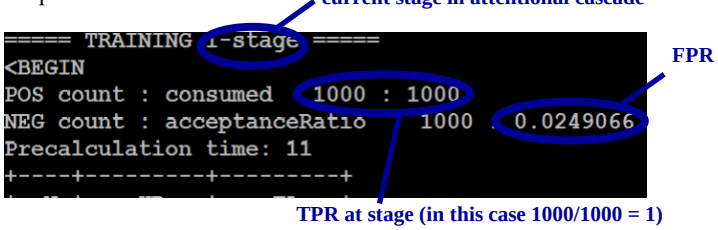
This will create 500 tiny 20×20 images of no entry signs (later used as positive training samples) and create the file **no_entry.vec**, which contains all these 500 small sample images. Each of the sample images is created by randomly changing viewing angle and contrast (up to the maximum values specified) to reflect the possible variability of viewing parameters in real images better.

Now use the created positive image set to train a no entry sign detector via AdaBoost. To do this, create a directory called **NoEntrycascade** in your working directory. Then run the **opencv_traincascade** tool with the following parameters as a single command in your working directory:

./opencv_traincascade -data NoEntrycascade -vec no_entry.vec -bg negatives.dat -numPos 500 -numNeg 500 -numStages 3 -maxDepth 1 -w 20 -h 20 -minHitRate 0.999 -maxFalseAlarmRate 0.05 -mode ALL

This will start the boosting procedure and construct a strong classifier stored in the file **cascade.xml,** which you can load in an OpenCV program for later detection as done in Subtask 1 of the assignment.

During boosting the tool will provide updates about the machine learning in progress. Here is an example output when using 1000 instead of 500 samples…



The boosting procedure considers all the positive images and employs sampled patches from the negative images to learn. The detector window will be 20×20. To speed up the detection process, the strong classifier is built in 3 parts (numStages) to form an attentional cascade as discussed in the Viola-Jones paper. The training procedure may take up to 15min for reasons discussed in the lectures – stop the training and restart if it exceeds this time.

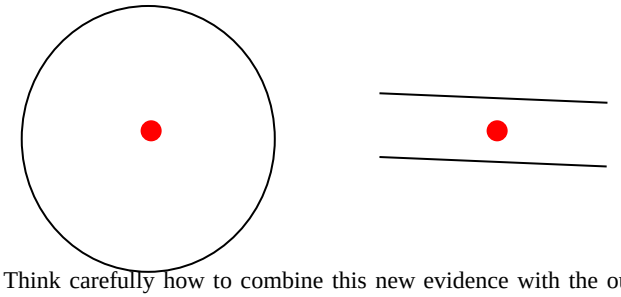
The Second Page of your Report (strict limit):

a) ***TRAINING PERFORMANCE:*** The training tool produces a strong classifier in stages. Per stage the tool adds further features to the classifier and prints the achieved TPR and FPR (false positive rate) for that point on the training data (see Figure). Collate this information into a graph that plots TPR vs FPR on the training data for the three different stages. Produce this graph in your report and briefly interpret what it shows.

b) ***TESTING PERFORMANCE:*** Test the no entry sigh detector's performance on all given example images. Produce the result images with bounding boxes drawn around detected no entry sign candidates (in green) and ground truth (in red) and include 3 of them in your report. In tabular form, calculate the overall TPR and F1 score per image and the average of these scores across the 16 images. Briefly discuss the performance achieved and give reasons for the different TPR values compared to the performance achieved in a).

Subtask 3: Integration with Shape Detectors

For the third subtask, use at least one Hough Transform on the query images in order to detect important shape configurations of no entry signs. Feel free to use and/or adapt your implementation of the Hough Transform from former formative tasks. You must implement your own Hough transform(s). Utilize the information (e.g. on lines, circles, ellipses) to aid no entry sign detection. For instance your system could consider the center of concentric circle or ellipse, and points locating between two parallel lines:



Think carefully how to combine this new evidence with the output of your Viola-Jones detector in order to improve on results. Implement and evaluate this improved detector.

The Third Page of your Report (strict limit):

a) ***HOUGH DETAILS:*** Show in your report for two of the given example no entry sign images which best exhibit the merit and limitations of your implementation: 1) the thresholded gradient magnitude image used as input to the Hough Transform, 2) a 2D representation of the Hough Space(s), 3) the result images showing final detections using bounding boxes (in green) and the ground truth (in red).

b) ***EVALUATION:*** Evaluate your detector on all of the example images. Provide the TPR and F1-score for each of the test images and their average across the 16 images in a table. Note in extra table columns the difference w.r.t. to the detection performances using only the Viola-Jones detector. Briefly note in bullet points the key merits and shortcomings of your now enhanced implementation.

c) ***DETECTION PIPELINE:*** In a flow diagram, depict how you have combined evidence from the Hough Transform(s) and Viola-Jones detector. In bullet points, explain briefly your rationale behind the way you have combined evidence.

Subtask 4: Improving your Detector

The final subtask allows you to develop the no entry sign detector further into a direction you choose. We ask you to identify and utilise some image aspects/features able to improve detection results further. This will generally include identifying, researching, understanding and using in OpenCV one other appropriate vision approach to further improve the detection efficacy and/or efficiency.

The Fourth Page of your Report (strict limit):

a) ***IDEA:*** In bullet points, explain briefly your rationale behind selecting the approach you have taken.

b) ***VISUALISE:*** Visualise important aspects of your technique in two of the given example images of no entry signs selected to best exhibit the merit of your approach.

c) ***EVALUATE:*** Evaluate your final detector on all of the example images, show the improvements in TPR and F1-score compared to previous approaches. Briefly note in bullet points the key merits and shortcomings of your final implementation.

Program Structure for Submission

We ask you to submit a zip file to Blackboard before the deadline. The file should contain all your source code and your 4-page PDF report, and if needed, include a readme.txt to explain how to compile/build. Your final detector program should take one parameter, that is the input image filename, and produce at least an image **detected.jpg**, which highlights detected no entry signs by bounding boxes. You can use your own machines or lab machines to develop your program, but please test that it runs on the lab machines seamlessly when it comes to marking. Make sure you regularly save/backup your work and monitor your workload throughout the duration of the project.

Administrative Notes from the Department of Computer Science

Deadline

The deadline for submission of all optional unit assignments is 13:00 on Friday 10th of December 2021. However, please make sure you submit it an hour early or so (not last minute!) to avoid upload problems. Submit on coursework unit COMS30068! Students should submit all required materials to the "Assessment, submission and feedback" section of Blackboard - it is essential that this is done on the Blackboard page related to the "With Coursework" variant of the unit.

Time commitment

The expectation is that students will spend 3 working weeks of no more than 40 hours, e.g. 5 days of 8 hour on their two assignments. The effort spent on the assignment for each unit should be approximately equal, being roughly equivalent to 1.5 working weeks each.

Academic Offences

Academic offences (including submission of work that is not your own, falsification of data/evidence or the use of materials without appropriate referencing) are all taken very seriously by the University. Suspected offences will be dealt with in accordance with the University's policies and procedures. If an academic offence is suspected in your work, you will be asked to attend an interview with senior members of the school, where you will be given the opportunity to defend your work. The plagiarism panel are able to apply a range of penalties, depending on the severity of the offence. These include requirement to resubmit work, capping of grades and the award of no mark for an element of assessment.

Extenuating circumstances

If the completion of your assignment has been significantly disrupted by serious health conditions, personal problems, periods of quarantine, or other similar issues, you may be able to apply for consideration
of extenuating circumstances (in accordance with the normal university policy and processes). Students should apply for consideration of extenuating circumstances as soon as possible when the problem occurs on eVisions.