Automatic image categorization

Objective: the goal of this project is to develop a vision-based system to automatically categorize the visual contents of images. In particular, we will focus on traffic signs.

1. Problem definition

Given an input image and a set of predefined categories, the developed computer vision system has to be able to correctly classify the input image into the right category.
We summarize below the main steps of a *minimal* system to achieve this goal:

1) Local feature extraction: for each image, compute its LBP descriptor (grid configuration to be chosen by the student, e.g. 5x5) to obtain a feature vector.
2) Classification: train a multiclass classifier by using the LBP descriptors of all the training images along with their labels. Suggestion: start using a *k*-NN classifier. At test time, use the trained classifier to assign a category to the target test image.

Note that the previously described pipeline is a *minimum* to start. It is expected that the student uses a more sophisticated approach to improve the results as much as possible. Tips will be provided during the lectures.

2. Dataset

For training and validating our system, we will use a simplified version (see Moodle) of the "German Traffic Sign" dataset, which is available at the following URL:
https://sid.erda.dk/public/archives/daaeac0d7ce1152aea9b61d9f1e19370/published-archive.html

It contains more than 40 categories, as it can be seen in the figure above. See the corresponding website for details.

In order to speed up the experiments, at execution time, the involved images have to be resized in such a way that width is 64 pixels, keeping their original aspect ratio.

3. Evaluation protocol

Three data partitions have been defined on the dataset: training, validation and test. Where each partition will contain a set of samples defined in its corresponding configuration file. For example, in order to select the hyperparameters of your classifier, you should use: *train_train.csv* and *train_valid.csv*. Once you have selected your best combination of hyperparameters, use *train_all.csv* in order to train your final model.
Then, the final accuracy of your system will be evaluated on the test partition. Note that you are not allowed to use the test partition until you have trained your final model.

How to compute the system performance?
You should report both the **mean accuracy** (or mean recognition rate) on all the classes and the **confusion matrix**. For the latter, you could also draw a bar plot where each bar corresponds to the accuracy of each category.

4. What to deliver?

The student has to develop, at least, the following two source files:

- *train.cpp*: this program is in charge of training a traffic sign classifier using the desired configuration (LBP grid, classifier type,...) and testing it following the protocol defined above. The trained models and the test statistics have to be saved into files (XML format).
- *test.cpp*: this program is in charge of applying a trained model to a single image or a set of images randomly sampled from the input categories.

The students will upload to Moodle a single zip file (*zip* file extension is compulsory) with the following filename '`Surname1_Surname2_Name_taskTraffic.zip`'. It must contain the following structure of directories and files:

- `README.txt`: plain text file describing how to compile and to run the provided software;
- `CMake files`: needed to build the binary files, note that at least two main files must be generated `train` and `test`, for training and testing the models, respectively.
- `src`: directory containing the source files (`.hpp`, `.cpp`);
- `models`: trained models obtained by the student for the different experiments, so they can be loaded for testing purposes (e.g. `model_N15_classifierType_vX.xml`);

- `data`: small set of images (e.g. 10) downloaded from the Internet that are correctly classified by the developed system.
- `docs`: a pdf file named `results.pdf` has to be included summarizing both graphically and in tables the experimental results obtained with the final (best) models. A section devoted to discussion of the results and conclusions has to be included as well.

5. When to deliver?

See Moodle for the exact deadline. Once the limit date has passed, the maximum possible grade will be reduced per week.

6. What is expected?

The following table summarizes the maximum grade the student can obtain depending on its development:

Points (up to)	***Item***	***Details***
7	Basic functionality (compulsory)	● The user can select the LBP grid configuration. ● The user can select whether to use uLBP or not. ● Input images can be normalized (min/max, mean/std). ● The number of neighbours *k* for classification (*k*NN) can be chosen both during training and testing. ● A SVM classifier with linear kernel can be used, allowing the user to select the margin C. ● A program called '*test*' that receives as input an image and the model file and returns the most likely category for that image and/or a ranked list of possible categories along with the corresponding score.
+1.25	Random Trees	The Random Tree classifier can be used for the task, allowing the user to select its configuration parameters.
+1.25	SVM extra	Additional kernels can be selected for SVM, along with their corresponding parameters.
+0.5	Graphical interface	The system outputs graphical results both for the training and test programs, or any other GUI facilities. E.g. the training program shows a visual confusion matrix of the test, and the test program shows the input image and displays on top of it the inferred category.

Additional details

Positive items:

- Well-documented code.
- Code properly modularized.

Negative items:

- Any partially copied lines of code will be considered as PLAGIARISM as will be graded as 0.

FAQ

A. **On the implementation of the function "*compute_confusion_matrix()*"**: Note that this is a general function that should work for any range of labels. We propose a simple implementation where the size of the CM is (*N*+1)x(*N*+1), where *N* is the maximum value on the labels found on the input vector. For example, if *N=33*, the size of CM would be 34x34. Given that, in order to fill your output CM, you only have to accumulate values as follows: CM[gt_lab][pred_lab] += 1
Tip: initialize CM with all zeros.

B. **How can I compute the test accuracy of my trained model?**: Once you have trained and saved your model to a file, use the option "*-t*" of the program "***train***" (not the *test.cpp* one) to compute the accuracy of your pre-trained model on the **test partition of the dataset**. As a reference, you should be able to reach a mean accuracy of at least 90%.

C. **How can I obtain the best combination of hyperparameters of my model?**: We recommend using a *grid search*. See further information here: https://en.wikipedia.org/wiki/Hyperparameter_optimization#Grid_search