Random Forests (RFs) Trainer

Instructor: Otmar Hilliges, Fabrizio Pece, Benjamin Hepp

1 Setting-up the Environment

In this repository, you will find a sub-folder named "cpp", with a fully functional Random Forest framework that will help you during this exercise. Please work on these files, but **DO NOT** push in the repository any changes you will make to them. Additionally, **please DO NOT push in the repository the data used to produce your results, nor your forests.**

You can get the data required for training and testing from here. Once you have downloaded the dataset, unpack the zip file somewhere on your hard drive, then run the MATLAB script "convert_mat_file_to_images.m" that you can find in the ".\data" sub-folder. When prompted, select the mat file you want to process, and then choose the folder where the data will be unpacked. To keep things organised, and assuming you are processing the training data, you should have a folder structure that resembles the following:

a2\UIERandomForest\data\train\images\

a2\UIERandomForest\data\train\images\images.csv

1.1 Compiling the Code

The framework was tested under Unix systems (Linux/Mac OS X), and builds via CMake. If you do not have access to a Unix environment, we have created a virtual machine image to run via Oracle VM VirtualBox. You can find the image here ($\sim$ 2GB).

If you are running your own Unix environment, once you have checked out the code you will need to install the following dependencies:

- CMake (3.1.3 or later)
- Eigen3 (3.1.2 or later)
- Boost (1.4.0 or later)

You can install these packages via apt-get on Linux and Homebrew on Mac OS X. To ensure you install the correct version of CMake, follow the instructions here. The virtual machine image should already have all the required packages installed. The

root password for it is "*uie2016*".

Once you have installed the required libraries, you can build the source via the usual CMake work flow:

```
cd PATH_TO_THE_CPP_FOLDER
mkdir build
cd build
cmake ../.
make
```

The last command will compile the source and will produce two binaries. Alternatively, you can run `cmake-gui` for a graphical version of CMake. On OSX, use the `-G Xcode` to generate an XCode project. More information on CMake generators can be found here.

1.2 Training

The framework comes with a training application. Assuming you have implemented steps 1–2 in Section 2.1, you can train your model by issuing the following command in a shell:

```
./depth_forest_trainer -f TRAIN_PATH/images.csv \
-j TRAIN_PATH/forest_name.json \
-l 7 -n 7 \
-c TRAIN_PATH/config.yaml
```

Here, the flag `-l` specifies the number of classes, while the flag `-n` the label assigned to background pixels. Please do not change these flags when using the provided dataset.

1.3 Testing

The framework also comes with a testing application to evaluate your model and produce performance metrics (i.e., per-pixel and per-frame confusion matrix and accuracy). You can run the evaluation application via the command:

```
./forest_predictor -f TEST_PATH/images.csv \
-j TRAIN_PATH/forest_name.json \
-l 7 -n 7
```

2 Exercise

2.1 RF Training

In this assignment you will experiment with several aspects pertaining Random Forests (RFs) training, in a multi-class classification task. Similarly to th first assignment, we will work with static hand gestures data. To do so, we are going to extract, from a dataset of real hand-gestures images, custom-engineered features, as described in Song

et al. [1]. Before starting working on the code, we strongly recommend to read, understand, and get familiar with the paper.

In the SVN you will find skeletal code that you are required to fill. There's also a `README` file that summarises the files you will need to modify. The following files have to be adapted to complete the various tasks: `weak_learner.h`, `image_weak_learner.h` and `histogram_statistics.h`. Places where code should be placed are marked with a "TODO UIE A2" comment block.

2.1.1 Step 1: Information Gain

In this first step, you are required to implement the information gain function in order to perform and asses node splits of your tree. You will need to implement:

2.1.1. The information gain metric as described in Song et al. [1]. See section "Hand State Classification Method" in the paper and class "WeakLearner" in the code;

2.1.2. The Shannon Entropy. See the "HistogramStatistics" class in the code.

Please refer to the comments in code for more details.

2.1.2 Step 2: Feature Response and Selection

In this second step, you are required to implement the feature response and selection introduced by Song et al. [1]. You will need to implement:

2.1.1. The feature response as per Song et al. [1]. See section "Hand State Classification Method" in the paper and the struct "ImageFeature" and class "ImageSplitPoint" in the code.

2.1.2. The random sampling of the features and thresholds. See the class "ImageWeakLearner" in the code.

Please refer to the comments in code for more details.

2.1.3 Step 3: Fine Tuning

In this step, you will learn how to fine-tune your RF model. The trainer application allows you to specify a variety of parameters at runtime and through a configuration file. For this step, we are interested in exploring only the effect of forest size (i.e., number of trees) and individual tree depth on the model accuracy. These parameters can be set via the attributes `num_of_trees` and `tree_depth` in the `training_parameters` block of the .yaml config file (see data\config_test.yaml for an example).

Plot performance-vs-depth and performance-vs-forest size. What is the most impacting factor for this task? Why do you think this is the case?

Additionally, you can also experiment with number of random features used for training. To do this, you will need to change the value assigned to the variable `samples_per_image_fraction`

in `image_weak_learner.h`.

In order to fine-tune your model, we suggest you write a script (using the language of your choice) that:

- Iterates over sensible ranges for tree depth and forest size;
- Creates a configuration file with the current parameters and pass it to the trainer application to generate a model. **Make sure the new configuration file has all the values which are in the template, or the application will not run correctly**.
- Evaluates the model and saves on disk the accuracy and parameters set. As the test application already computes and prints the accuracy, you can either modify the source to save this on disk, or you can grab it from the console directly in your script.

Please be aware that with increasing tree depth, training time (and memory requirement) will grow. Here's some stub Python code to illustrate this process:

```
...

#specify the parameters to train the forest
the_tree_depth = [16, 18, 20, 22]
the_forest_size = [1, 2, 3, 4, 5]

...

for _forest_size in the_forest_size:
        for _depth in the_tree_depth:

                maximum_depth = _depth;
                num_of_trees = _forest_size;

                config_file_name = base_directory + "/configuration_" + \
                "_D" + str(maximum_depth) + \
                "_T" + str(num_of_trees) + ".yaml"

                forest_file_name = base_directory + "/forest_" +  \
                "_D" + str(maximum_depth) + \
                "_T" + str(num_of_trees) + ".json"

                print("Writing config file...")
                write_config_file(config_file_name, \
                maximum_depth, \
                num_of_trees)

                print("Training...")
                run_train( train_data_path, config_file_name, \
                forest_file_name, \
                num_gestures, \
                num_gestures)

                print("Testing...")
                pix_acc, frame_acc = run_test(  train_data_path, \
```

```
                forest_file_name , \
                num_gestures , \
                num_gestures)

                #write out the accuracy
                ...
```

References

[1] Jie Song, Gábor Sörös, Fabrizio Pece, Sean Ryan Fanello, Shahram Izadi, Cem Keskin, and Otmar Hilliges. In-air gestures around unmodified mobile devices. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 319–329, New York, NY, USA, 2014. ACM.