**CHaDLE Web UI Explanatory Document** *Ver 1.0*

**General concept**

The Web UI firstly received user input. Then, it controls local Halcon procedures/scripts to conduct deep learning process. During the process, TrainingInfo.hdict and EvaluationInfo.hdict files which record training data, will be produced and updated periodically. The UI reads these two files and plot real-time graph.

**Folder structure**

The folder accessed by this web UI shall follow the structure below.

**Chadle\_Projects**: Main folder containing everything. Folder path will be used in source code.

**Chadle\_Data**: Project data, separated by project titles. Under each project title, there should be 3 folders :Image,Model,Split.

**Chadle\_Data /Project Title/Image**: Images for training, there should be Test, Train and Validation, and all containing images of different categories.

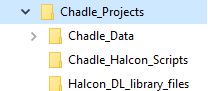
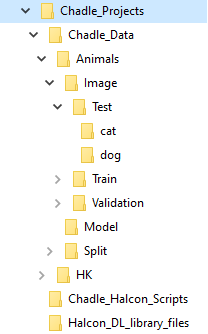
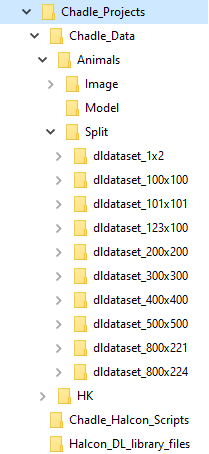
**Chadle\_Data /Project Title/Model**: Contains the best and final models of training.

**Chadle\_Data /Project Title/Split**: Contains pre-process data. Does not require user’s action.

**Chadle\_Halcon\_Scripts**: Contains Halcon scripts and files produced by them.

**Halcon\_DL\_library\_files**: Contains modified Halcon procedure libraries, in addition to mandatory ones.

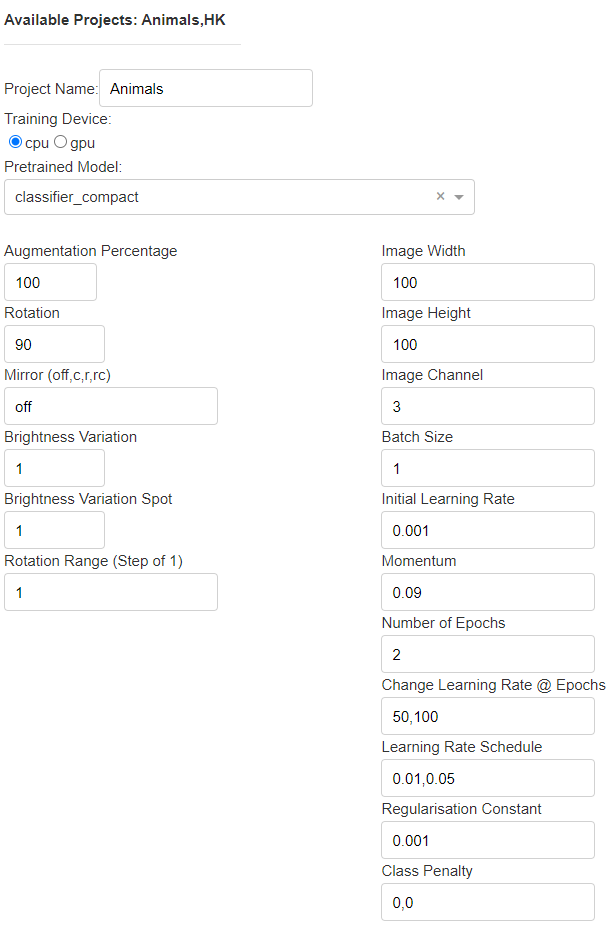
Note that these folder names are case sensitive except for project title(Animals) and image categories(cat&dog).

**Code Structure**

There are two Python files: app.py & run.py; app.py is mainly for web layout and graph plotting, while run.py includes all Halcon related methods.

*app.py*

**

**Parameters** will be inputted by user, to operate Halcon procedures.

The “Project Name” input may not be exact and is not case sensitive. E.g. type “animal” instead of “Animals”.

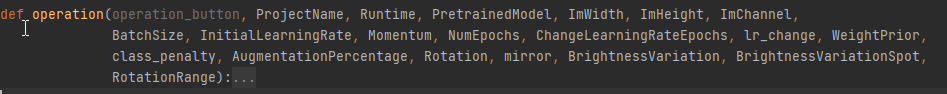
Some of the values have modified format. E.g. “Rotation” is restricted between -180 and 180, divided by 90 degrees.

Worth to mention: on top, there is label showing available projects.

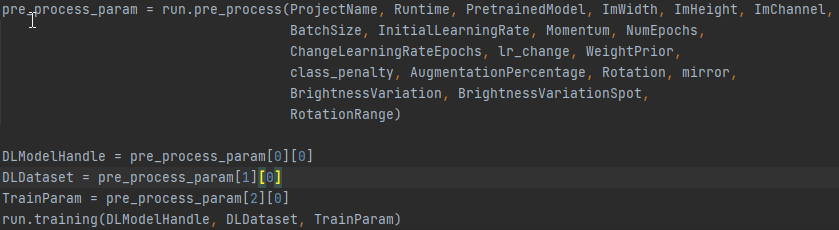
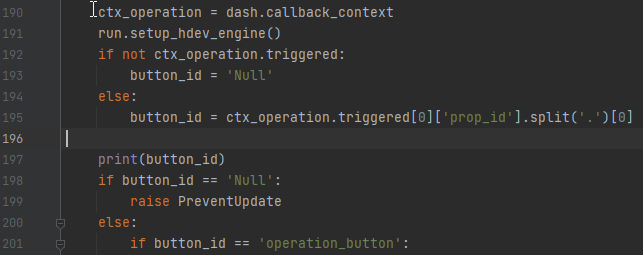
It is initialized when web starts, values fetched from run.py.



** “START TRAINING”** Button is associated with method operation(…), which combines pre-process and training-process.

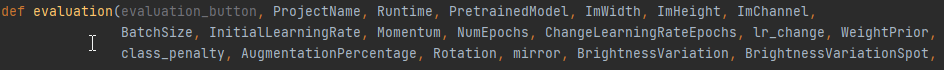


Inside operation(…), ctx\_operation acts as button listener, and button\_id indicates which button has been pressed. When the button is pressed, run.pre\_process method will start running, and output from which will be input for run.training method.



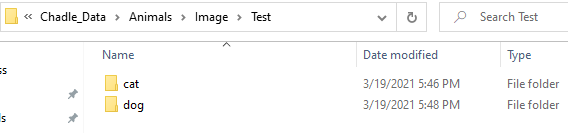
Note that the run.setup\_hdev\_engine() method is for setting up Halcon environment and must be called before pre-process/training/evaluation.

**“EVALUATION”** Button is associated with method evaluation(…), which conducts evaluation.



Concept of evaluation(…) is similar with operation(…). Just with additional graph-plotting component.

Results of evaluation will be extracted, together with local folder names of each categories, to plot confusion matrix and show important stats, e.g. mean\_precision.



*In Animals project, categories will be cat and dog.*

In the **Graph Plotter**, there are 3 parts: metrics, loss graph and top1 error graph.

Metrics are updated by update\_metrics(), which get values from run.get\_TrainInfo().

Loss and top1 error graphs are plotted by plotly.tools.make\_subplots. Required values are obtained from run.get\_TrainInfo() and run.get\_EvaluationInfo().

The graph plotter component updates by built-in interval. 

In this app, interval is set to 1000 milli sec, a rather long time to avoid compatibility errors with Halcon.

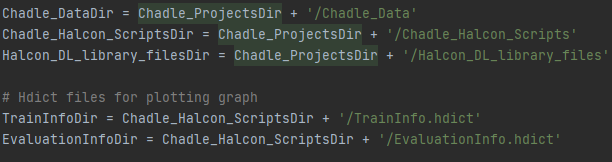
All the real-time data were collected by reading Hdict files produced during training. Details will be explained in run.py part.

*run.py*

At the beginning, location of CHaDLE project main folder must be set.



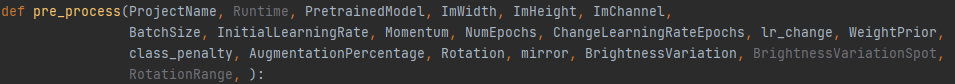
If the folder structure is strictly followed, all rest of required directories will be read automatically.



**setup\_hdev\_engine()** defines and builds the base environment of Halcon.

Note that other than mandatory Halcon procedure library, we must add the customized library to comply with our modified Halcon script. 

**pre\_process()** performs several tasks, augmentation process is included.



Notice the mutex.dat file being produced.  It is for Halcon pause-resume training function which has not been fully developed. Just ignore the mutex.dat file and do not delete it.

First task of pre\_process() is to delete existing training data files, in order to empty the graphs when starting new pre-process and training session.

Second task is to search for project name inputted by user.



Augmentation and pre-process will start when there is matched result. Outputs will be collected at the end.

**training()** and **evaluation()** are simpler methods.

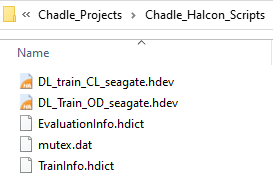
After pre-process, outputs will be directed to training() to start deep learning training in Halcon.





evaluation() performs with similar logic, just that it will return list containing of evaluation results for graph plotting.

During training, Halcon will produce TrainInfo.Hdict and EvaluationInfo.Hdict under the folder where Halcon scripts exist.



As the training goes, new data will be written into these two files with certain interval. For EvaluationInfo.Hdict will be every epoch, and for TrainInfo.Hdict will be affected by batch size.

**getImageCategories()** does not require Halcon. It reads path of folder names under ProjectName\Image\Train and return as categories.