**Introduction**

Created base on Python TKinter and Halcon. The app first intake user’s inputs as parameters, then call and operate respective Halcon procedures and commands for DL.

Ultimate aim is to achieve classification, object detection and anomaly detection functions within the single GUI.

Halcon procedure files should be prepared in the first place, as these Halcon codes are the working core.

There are 3 major steps being executed by Halcon: pre-process, training, and evaluation.

In this document, ClassificationPage.py will be explained in detail with natural sequence of the codes.

**Detailed Explanation**

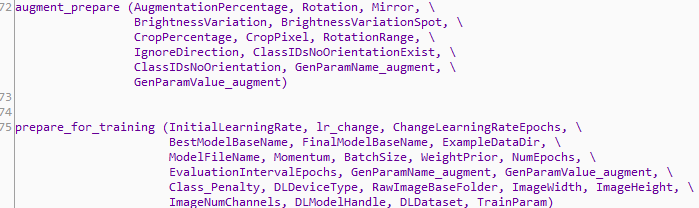
**Setting Up**

1. Initialize Halcon by setting HDevEngine.
2. Set path for Halcon procedure library. Note that other than the mandatory path of default Halcon procedures, path where newest DL library files (filenames end with \_PK.hdpl) are located should also be set.
3. Set path for the exact Halcon program(.hdev) we are using.

program = ha.HDevProgram('C:/Users/930415/Desktop/DL\_train\_CL\_seagate.hdev')

Then, set respective Halcon procedures in the program, which will be used later. E.g. augment\_prepare.

aug\_call = ha.HDevProcedureCall(ha.HDevProcedure.load\_local(program, 'augment\_prepare'))



*(Procedures inside the Halcon program)*

1. Note that we could skip setting Halcon program and use the procedures directly from Halcon libraries. Notice the load\_external. E.g.

augment\_proc\_call = ha.HDevProcedureCall(ha.HDevProcedure.load\_external('augment\_prepare'))

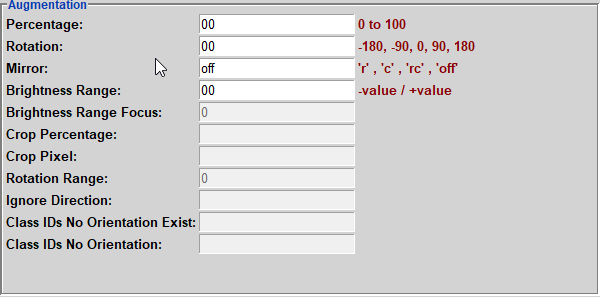
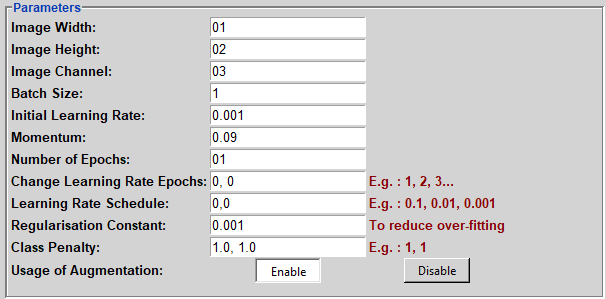
**GUI Layout**

1. Frames. Create several individual frames to separate segments.
2. Operational buttons.



Each button is linked to dedicated method. Operational sequence is from left to right, as ruled in Halcon program.

1. Parameter entry. User key in values that control the DL process. Augmentation segment is being controlled by a switch.

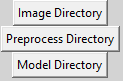


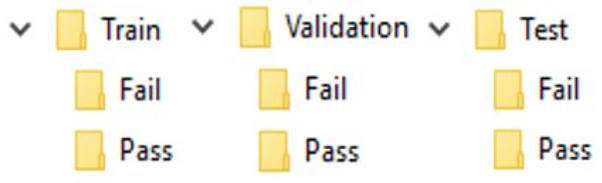
1. Directory, runtime and pretrained model selection. Note that for classification, three directories will be needed.

Image: Images were split to folders. See picture below.

Preprocess : To store Hdict files created for training later.

Model: Folder storing resulting DL models.

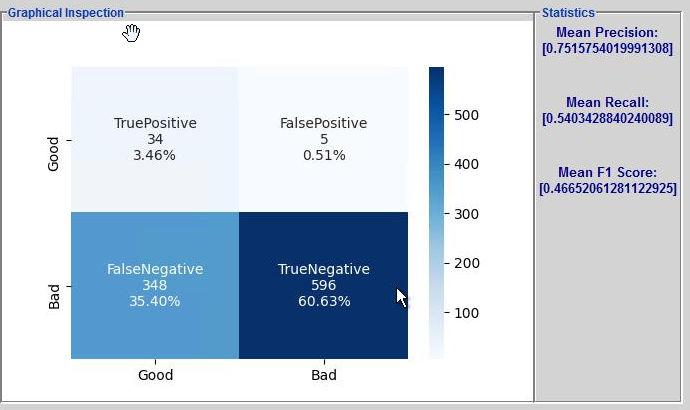
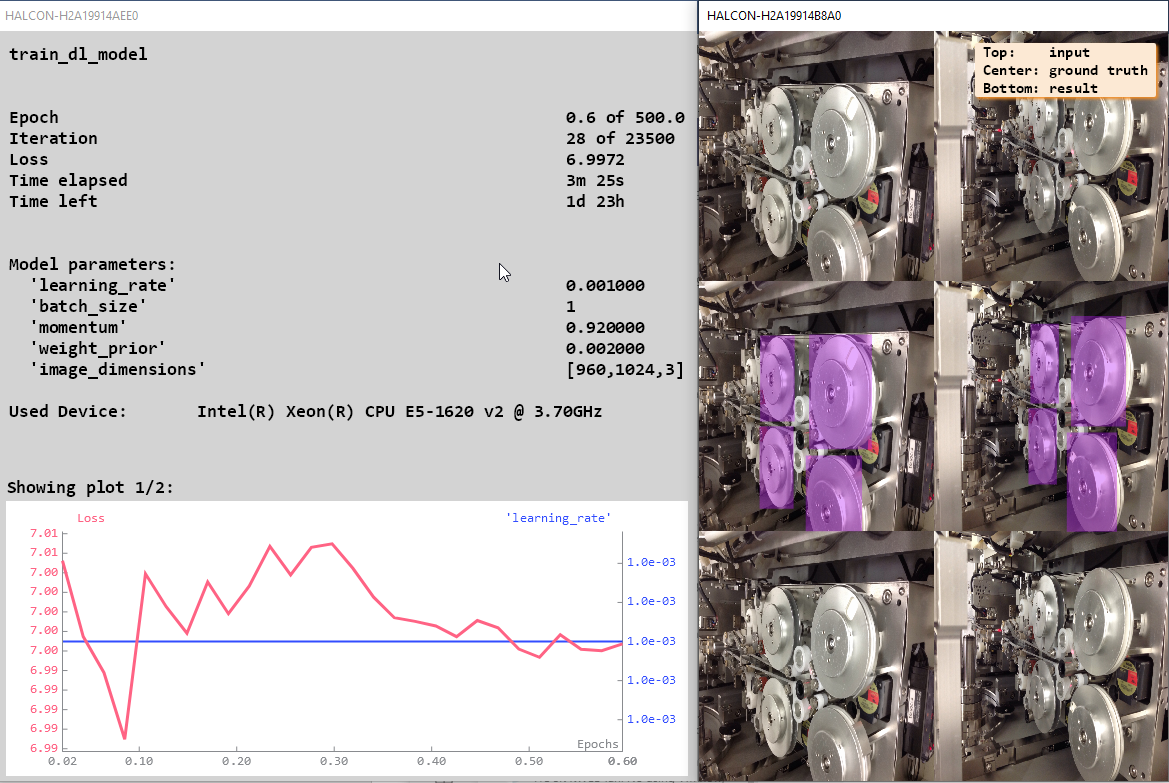




*An example of Image Directory prepared manually prior to DL training process.*

1. Progress Indication. Progress bar and labels showing the current condition of pre-process will be shown bottom right corner.

1. Evaluation. After training, confusion matrix and important training data will be shown. 
2. Native Halcon visualization window will be shown during training process. Note that these windows were called directly by Halcon codes. They will be working on individual threads and we foresee that some development works are needed to adapt this into WebGUI. In this part, we would like IT team to advise/propose proper integration method. We are currently working with MVTec on an alternative way to show this visualization through Python itself.

**Methods**

In this class, there are 2 types of methods.

1. Operational methods that interact with Halcon commands. *startAugmentation(), startPreproc(), startTraining(), startEvaluation().*
2. Methods that perform respective operational methods with threading. *preprocess\_Run(), training\_Run(),evalutation\_Run().*

*startAugmentation()* : Using user inputs to call Halcon procedure *augment\_prepare().* Some outputs will be used in pre-process.

*startPreproc()* : Pre-process. Inside this method, augmentation is executed. Using the outputs from augmentation and more other relative user inputs, Halcon procedure in *prepare\_for\_training()* procedure will be called. Some outputs will be used in training.

*startTraining()*:Training. Using outputs from preprocess to conduct DL training. Will form trained models in selected folder.

*startEvaluation()*: Evaluation. Results could be extracted from Hhandles as python tuple, and then make lists for confusion matrix.

*preprocess\_Run() & training\_Run()*:Utilize multi thread to perform the tasks.

*evalutation\_Run()*: Other than utilizing multi threads, confusion matrix will be drawn by calling makeCM class, which involves using matlibplot to draw the matrix and showing labels.

Note that message boxes will be shown when the threads are running. They function not only as reminders, but also triggers for the threads to run properly.

