ADCS Overview

Automatic Defect Classification System (ADCS) for FS, BS & EN Model Deployment

By: `Tam Zher Min`
Email: `tamzhermin@gmail.com`

Summary

This is an indepently architected sequential system (similar to AVI recipes), threaded alongside a Tkinter GUI. It can automatically classify and sort wafer image scans locally for SSMC and can also train new machine learning models. Total ~2000 LOC (lines of code). Run the `ADCS.vbs` file to start.

Operator's Guide

Slides for operators can be found in the `/ADCS/notes/guides` folder. This guide is for operators looking to check the wafer lots with defects and for how to sort the wafer scans after they are classified by the ADCS.

System Design

System Logic

This is a full-fledged system that I planned and wrote every single line myself during my 4-month internship at SSMC (AUG 23 '21 — JAN 07 '22). This system deploys 2 CNN models locally (up to 3 needed) and performs inference on all images found from continuously polling the folder where all the wafer scans are transferred to.

The system also needs to parse through a weird file format to extract relevant information. This file is also required to be edited because SSMC's software can only understand this format. The way it is parsed is a little hacky and not 100% fool-proof but because it does not have a fixed format, there are no easy ways around it.

The Tkinter GUI was very challenging to code because UI systems are usually very finnicky. However, I managed to make it work, allowing users to change settings and logging to the GUI with a queue and using threading to run the production or training mode separate from the main Tkinter GUI thread.

All system design logic, flow, structure and considerations were by me — good in that I managed to produce something of this scale alone, bad in that I am not sure if these are the best practices or if I had missed out on any glaring problems; but I did what I could.

Training Mode

Take note, for training mode, the "Balanced no. of Samples per Class" value is important. You should derive this number by looking at the number of images you have for each class. It should be more than the number of samples in each class but lower than the most majority class.

For example, if the chipping class has only 30 images while the stain, scratch and whitedot classes have 100 images each and the AOK class has 1000 images, then you should pick between a minimum of 100 to a maximum of 1000. A good number might be 300 for this case. You can refer to the table below to get a sensing.

Hence, it heavily depends on the number of samples you have for training. As more images get sorted into the trainval folder for future retraining, this value should increase over time, otherwise you are not fully utilising the images to train the models.

eg.	aok	chipping	scratch	stain	whitedot	RANGE	# INPUT #
1.	400	10	20	40	20	40-400	100
2.	1000	30	100	150	200	200-1000	300
3.	800	100	200	150	75	200-800	400

Production System Flow

1. AVI scans wafers and generates FBE images
2. KLA files and images fed into ADC drive's "new" directory (dir)
3. ADCS continuously polls "new" dir for KLA files
4. If KLA files found, start model inference; else, poll again after some wait time
5. Model Inference
    1. Reads oldest KLA file and stores relevant information into "wafer" data structures
    2. Checks if filenames referenced in KLA file can be found in the "new" dir
    3. If all found or after timeout, feed FS/BS/EN images into their respective models
    4. FBE models classify images and modify the KLA file's CLASSNUMBERs to the predictions
    5. Results will also be saved to CSV (Excel) files for future reference
    6. Move KLA file and images to ADC drive's "old" directory and also copy them to K drive
    7. Predicted files in "unsorted" folder require manual sorting for future retraining
6. Repeat

Folder Structure (Critical Files Only)

Do follow this folder structure to ensure reproducibility

```
 [K drive]                  // modified KLA file and images copied here after inference
[ADC drive]                 // houses all wafer data and ADCS application code
│
├── /data                   // stores all KLA files and images from AVI
│   ├── /new                // unpredicted lots
│   └── /old                // predicted lots for backup and retraining
│       ├── /backside       // test/trainval/unsorted folders will have folders for all 5 classes
│       │    ├── /test      // manually sorted images for model testing to simulate new images
│       │    ├── /trainval  // manually sorted images for model training and validation
│       │    └── /unsorted  // predicted images to be sorted into /trainval for future retraining
│       │        ├── /aok
│       │        ├── /chipping
│       │        ├── /scratch
│       │        ├── /stain
│       │        └── /whitedot
│       ├── /edgenormal     // test/trainval/unsorted folders will have folders for all 2 classes
│       │    ├── /test
│       │    ├── /trainval
│       │    └── /unsorted
│       │        ├── /aok
│       │        └── /chipping
│       ├── /frontside      // any frontside scans found will be backed up here
│       └── /unclassified   // all ignored defect codes, eg. edgetop (176) and wafer maps (172)
│
└── /ADCS                   // the Automatic Defect Classification System
    ├── /assets             // miscellaneous files abstracted away from end-users
    │   ├── debug.log       // log file of the latest run of main.py for debugging
    │   ├── icon.ico        // wafer icon found online
    │   ├── requirements.txt// necessary python libraries and versions
    │   ├── run.bat         // batch file that runs main.py using specified python.exe file
    │   └── settings.yaml   // config file for users to easily change settings and modes
    ├── /models             // trained FBE .h5 tensorflow models
    │   ├── /backside
    │   ├── /edgenormal
    │   └── /frontside
    ├── /results            // FBE predictions in CSV for production and training modes
    │   ├── /production
    │   │   ├── /backside
    │   │   ├── /edgenormal
    │   │   └── /frontside
    │   └── /training
    │       ├── /backside
    │       ├── /edgenormal
    │       └── /frontside
    ├── /src                // helper modules for ADCS in OOP style
    │   ├── adcs_modes.py   // script file with the 2 modes chosen in the GUI
    │   ├── be_trainer.py   // model training code for backside and edgenormal models
    │   ├── kla_reader.py   // code to parse and edit KLA files
    │   └── predictor.py    // model prediction code generic for FBE models
    │
    ├── *ADCS.vbs           // starts the ADCS app
    ├── main.py             // python script of the ADCS GUI to START/STOP
    └── README.pdf          // this user guide you're reading saved in PDF format
```

Full Program Settings

Below are the descriptions for all of the settings found in the settings.yaml file in the assets folder. They allow users to change advanced settings for the code outside of the GUI such as the delay times and training hyperparameters.

The descriptions below help users understand what each setting does in a readable manner because the actual settings.yaml file is automatically generated in alphabetical order.

Note, there is technically no need to change anything in the settings.yaml file. Also note that all settings are case-sensitive. You can read more about the YAML syntax here.

Understanding the Descriptions

```
setting_name: [option A / option B] (default=x)
    # description
```

The setting's name will be before the colon followed by the available options in square brackets and the recommended default values in round brackets. The next indented line will be a short description of the setting. However, in the actual settings.yaml file, you would just write:

```
setting_name: setting_value
```

All Available Settings

ADCS Mode

```
adcs_mode: [PRODUCTION / TRAINING] (default=PRODUCTION)
    # either production (classification) or training mode
```

Folder Locations

```
adc_drive_new:
    # folder where all new AVI scans are transferred to
adc_drive_old:
    # folder where all old predicted wafer lots and images are stored for backup
k_drive:
    # folder where Klarity Defect finds all KLA files and wafer scans
```

Pause Times

```
pause_if_no_kla: (default=30)
    # long pause time in seconds in between checking cycles if no KLA files found
pause_if_kla: (default=5)
    # short pause time in seconds in between checking cycles if there are KLA files

times_to_find_imgs: (default=3)
    # no. of times to try and find images referenced in KLA file
pause_to_find_imgs: (default=10)
    # pause time in seconds to try and find the images referenced in KLA file
```

Model Configs

```
BATCH_SIZE: (default=8)
    # no. of images to classify at a time, higher requires more RAM
CONF_THRESHOLD: [0 - 100] (default=95)
    # min. % confidence threshold to clear to be considered confident
```

BS Predictor Configs

BS Original Code: [174] AVI_Backside Defect

```
bs_model:
    # specific model to use, leave empty to use latest model
bs_defect_mapping: # correct KLA defect codes for BS defects
    aok: 0         # Unclassified
    chipping: 188  # OQA_Edge Chipping (BS)
    scratch: 190   # OQA_BS-Scratch (Cat Claw)
    stain: 195     # OQA_BS-Stain
    whitedot: 196  # OQA_BS-White Dot
```

EN Predictor Configs

EN Original Code: [173] AVI_Bevel Defect

```
en_model:
    # specific model to use, leave empty to use latest model
en_defect_mapping: # correct KLA defect codes for EN defects
    aok: 0         # Unclassified
    chipping: 189  # OQA_Edge Chipping (FS)
```

FS Predictor Configs

FS Original Code: [056] AVI Def

`unimplemented`

BE Trainer Configs

Basic Trainer Configs

```
training_runs: (default=5)
    # no. of models to train
training_subdir: [BACKSIDE / EDGENORMAL]
    # to train either backside or edgenormal models
training_n: (default=300)
    # balanced no. of samples per class
training_saving_threshold: [0 - 100] (default=95)
    # min. % test accuracy to clear before the trained model is saved
```

Advanced Hyperparameter Configs

```
dense_layers: (default=1)
    # no. of dense layers after the layers of the pretrained model
dense_layer_size: (default=16)
    # size of each dense layer, bigger size results in a bigger .h5 model
dropout: (default=0.2)
    # % of weights to drop randomly to mitigate overfitting
patience: (default=10)
    # no. of epochs to wait before early stopping and take best model
```

Custom Testing Mode

```
training_mode: [true / false] (default=true)
    # false if you want to test a specific model
test_model: (default=empty)
    # test this model name if training_mode is false
```

Abbreviations Guide

- SSMC: Systems on Silicon Manufacturing Company (TSMC & NXP JV)
- Defect Classes (the other classes are self-explanatory)
  - aok: ALL-OK, meaning a normal image with no defect (false positive)
- Domain
  - FS: Frontside
  - BE: Back & Edge (Backside + Edgenormal)
  - BS: Backside
  - EN: Edge Normal
  - ET: Edge Top (ignored)
  - FBE: Frontside-Backside-EdgeNormal
  - AVI: Automated Vision Inspection
  - KLA: File format used by SSMC's infrastructure
- System
  - CNN: Convolutional Neural Network, the machine learning model used
  - CLI: Command Line Interface
  - GUI: Graphical User Interface
  - df: Dataframe, think of it as Excel but in code