Modular MetaCog Via Docker Containers

File Structure:

Main

- ***Containers***
  - **Discriminator**
    - Classifier Model
    - Python Code
  - **Evaluator**
    - Python Code
  - **FA_CD**
    - Classifier Model
    - Cognitive Detector and Job Control
    - Python Code
  - **FA_GLRT**
    - Classifier Model
    - Cognitive Detector and Job Control
    - Python Code
  - **FA_IDEAL**
    - Classifier Model
    - Cognitive Detector and Job Control
    - Python Code
  - **Results**
    - Python Code
- ***Datasets***
  - Mat File
- **Args Class Module**
- Base Image Dockerfile
- Discriminator Dockerfile
- Evaluator Dockerfile
- FA_CD Dockerfile
- FA_GLRT Dockerfile
- FA_IDEAL Dockerfile
- Results Dockerfile
- Text File for Terminal Commands

The Containers folder has all the files required for each individual Docker Image, this includes the specific python file to run the code segment and any requirements such as models or locally imported class files.

These files are separated from the Dataset and Args Class Module so that a single copy can be shared between all of the containers. This allows a single change in the Args Class Module, including the input data file or configuration settings, to be used for updating the whole system. Alternatively this could be changed to allow for more specific modular configuration, but is initially set up for ease of use.

A base image has been created to increase the efficiency of building the system, a great idea credited to Zak. The base image installs Ubuntu and all of the required libraries, this includes TensorFlow and Numpy. Then, rather than having to install all of this again for each container, the MetaCog containers all inherit from this base image. This decreased the build time from 1219 seconds per container down to 24 seconds per container!

Each container first copies the common Dataset folder and the common Args Class Module, then copies its corresponding subfolder in the Containers directory. Then it simply runs the specified python file. All of this can be done by building the image using the correct Dockerfile and then running the resulting Docker Image. These commands are documented in the Text File for Terminal Commands (“system_setup_commands.txt”)

Each image runs on Ubuntu 22.04 currently. This can be changed by updating the ‘FROM ubuntu:22.04’ line of the Dockerfile_base file in the main directory. The installation of cuda for GPU usage was commented out, but is included if needed for future projects.

To ensure data is passed between each container, a volume must be created prior to building the Docker Images. A volume is a long term file system managed by Docker which allows storage of custom data that is retained after a Docker Image is no longer running. This is used in the --mount type=volume option of the docker run command.

1.) The specified data file in the Args Class Module is read in the Discriminator container. Then, it is passed into the discriminator model where a probability distribution is generated for each data point. This is then saved to the file ‘distribution_tensors.csv’ inside the Volume.

Input : MAT File

Output : (7,N) Dimensional Numpy Array

7 elements in each list

N number of data points / lists

The N number of data points is specified in args.max_test of the Args Class Module.

2.) This file is then read in the Evaluator container, where each data point is processed. Currently, this just selects the max element in each array from the discriminator’s probability list and compiles it in a list. This data is saved to the file ‘max_disc_list.csv’

Input: (7,N) Dimensional Numpy Array

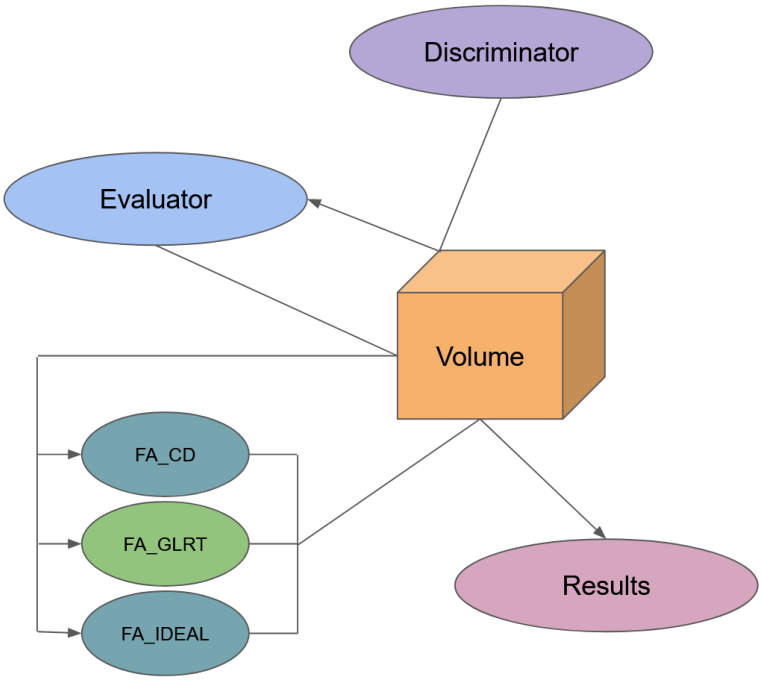
Output: (1,N) Dimensional Numpy Array

3.) The ‘max_disc_list.csv’ data is read into the FA containers to calculate the rate of False Alarms using various metrics. The three are all separate but use the same input. Each uses the input MAT File and selects a model to use for calculations based on the max disc list and preset methods. Following the calculation in each container, they save the False Alarm rate to a corresponding file: ‘FA_cd.csv’, ‘FA_glrt.csv’, ‘FA_ideal.csv’.

Input: (1,N) Dimensional Numpy Array, Arg Class Module Data

Output: Integer

4.) Using the output of the FA containers, the Results container simply reads the data from each file and then outputs the numbers to the terminal with labels.



Known Improvements to be made:
- Currently TensorFlow outputs warning messages about the version of the model being outdated and cuda is not being used, this generates a lot of unnecessary text in the terminal.
- The images do not automatically delete themselves. I am not sure if it would cause this, but I think it would be good to not have the potential issue of large memory usage over time.
- If needed, a script to run all the container configuration commands may be helpful.
- Kubernetes will be used next to orchestrate the system of Docker Containers.