Version and Modules:

Code uses Python 3.6 and MATLAB 9.7 (for analysis only).

In python, modules used are:

- glob
- sklearn
- PIL
- matplotlib
- argparse
- Keras (2.3.0)
- Tensorflow (2.0)

It is possible to download these using a combination of conda and pip. For University of Washington student, and staff: it is possible to run this in the DAIS machine with the use of a conda environment and venv.

Files and Code:

NOTE: more information about specific methods and classes can be found in their specific file.

"Outputs" folder contains:

- Outputs for cases analyzed in the paper, and the outputs related to them, and the cases themselves.

"Model Software" folder

project.py:

If the file is executed as main, it is possible to pass arguments related to the creation, training, and testing.

The following are the arguments that can be passed:

--in	-> input file -- directory or single file
--in_validate	-> input file -- directory or single file
--sample	-> sample file directory for sample images to be saved
--results	-> result file directory for text related output
--epochs	-> number of epochs to train the neural network(int)
--save_file	-> file to save/load model
--batch_size	-> number of images per batch to train the neural network

--info_interval	-> Intervals where information is saved

For using the file for model creation, training, and saving (more complex example later):

1) When initializing, pass batch size. Model will be created

2) call .train_model with the input data file, test file, sample file, result file, number of epochs, and interval for sample

3) Call .save_file with the file to save information

For loading model, when initializing, pass batch size, test = 0, file for loading model, validation file input, validation file output

utils.py

File contains helper methods used by all other python files. More instructions for each method in file.

create_datases.py

Execution of file creates text files used for the testing of the project. Uses the 'utils.py' file.

Recommended Directories for Training:

During execution, it is recommended the use of the following directories:

1) data	-> Store images in .float format.
2) model	-> Store created models
3) names_log	-> directory for saving text files with split data information saved in it
4) results	-> saved text file with loss during execution
5) sample	-> saves images of test during execution

"Analysis Software" folder contains:

applys_brisque_specific.m

Given a float file, it calculates its brisque score, and returns it as an int.

apply_brisque_average.m

Calculates the average brisque score between all float type images in a folder. Returns the average, and the brisque scores. List ordered based on the order of creation of the input.

ADAM_comparison_script.m:

Script used for printing the different BRISQUE tables and analysis. Can be executed directly. Used to print tables for Case 0, 1, 2 for 300 data points , and the one for Case 3,4,5 for 3000 data points. Uses applys_brisque_specific.m and apply_brisque_average.m

"Training Information" contains:

X_trainning_data_bY.txt:

Text files with image names used for training. These are the files mentioned before as having the proper division between data. X is the number of images used for creating the file. Y is the batch size associate with the data inside of it.

Running_file.txt

File contain use inputs used at different points for training.

Model for 3000 Images and 200 epochs

x.model

Contains a model Generator and Discriminator trained with 300 Images and 200 epochs.

Report_iteration.txt

File with report of loss during the training of the model

'samples sample' folder

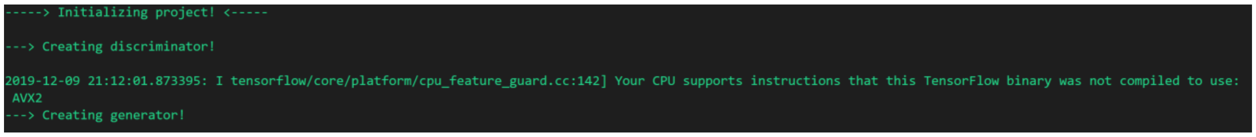
A sample of how 2 sample images type output would look like

Demo Progression:

1) When running from command line, and input is given to the model, such as:

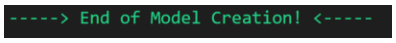
```
python project.py --in ./names_log/3000_trainning_data_b3.txt --
in_validate ./names_log/3000_validate_data_b3.txt --info_interval 30 --
epochs 200
```

2) Message letting user know of the start of model creation is given:



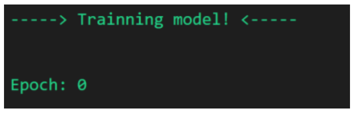
```
-----> Initializing project! <-----

---> Creating discriminator!

2019-12-09 21:12:01.873395: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled to use:
 AVX2
---> Creating generator!
```

3) If success, this message is given:



```
-----> End of Model Creation! <-----
```

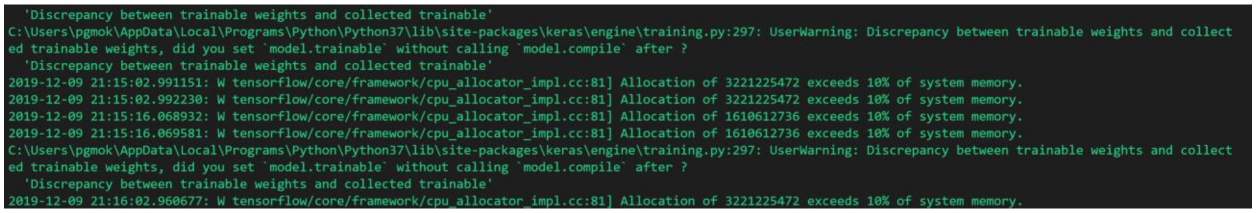
4) As model starts training, this message is printed:



```
-----> Trainning model! <-----


Epoch: 0
```

   a. This lets the user know the model is training
   b. Epoch: X let the user know what epoch they are in.

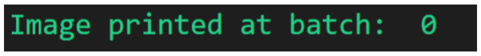
5) This message is printed after epoch 0:



```
  'Discrepancy between trainable weights and collected trainable'
C:\Users\pgmok\AppData\Local\Programs\Python\Python37\lib\site-packages\keras\engine\training.py:297: UserWarning: Discrepancy between trainable weights and collect
ed trainable weights, did you set `model.trainable` without calling `model.compile` after ?
  'Discrepancy between trainable weights and collected trainable'
2019-12-09 21:15:02.991151: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 3221225472 exceeds 10% of system memory.
2019-12-09 21:15:02.992230: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 3221225472 exceeds 10% of system memory.
2019-12-09 21:15:16.068932: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 1610612736 exceeds 10% of system memory.
2019-12-09 21:15:16.069581: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 1610612736 exceeds 10% of system memory.
C:\Users\pgmok\AppData\Local\Programs\Python\Python37\lib\site-packages\keras\engine\training.py:297: UserWarning: Discrepancy between trainable weights and collect
ed trainable weights, did you set `model.trainable` without calling `model.compile` after ?
  'Discrepancy between trainable weights and collected trainable'
2019-12-09 21:16:02.960677: W tensorflow/core/framework/cpu_allocator_impl.cc:81] Allocation of 3221225472 exceeds 10% of system memory.
```

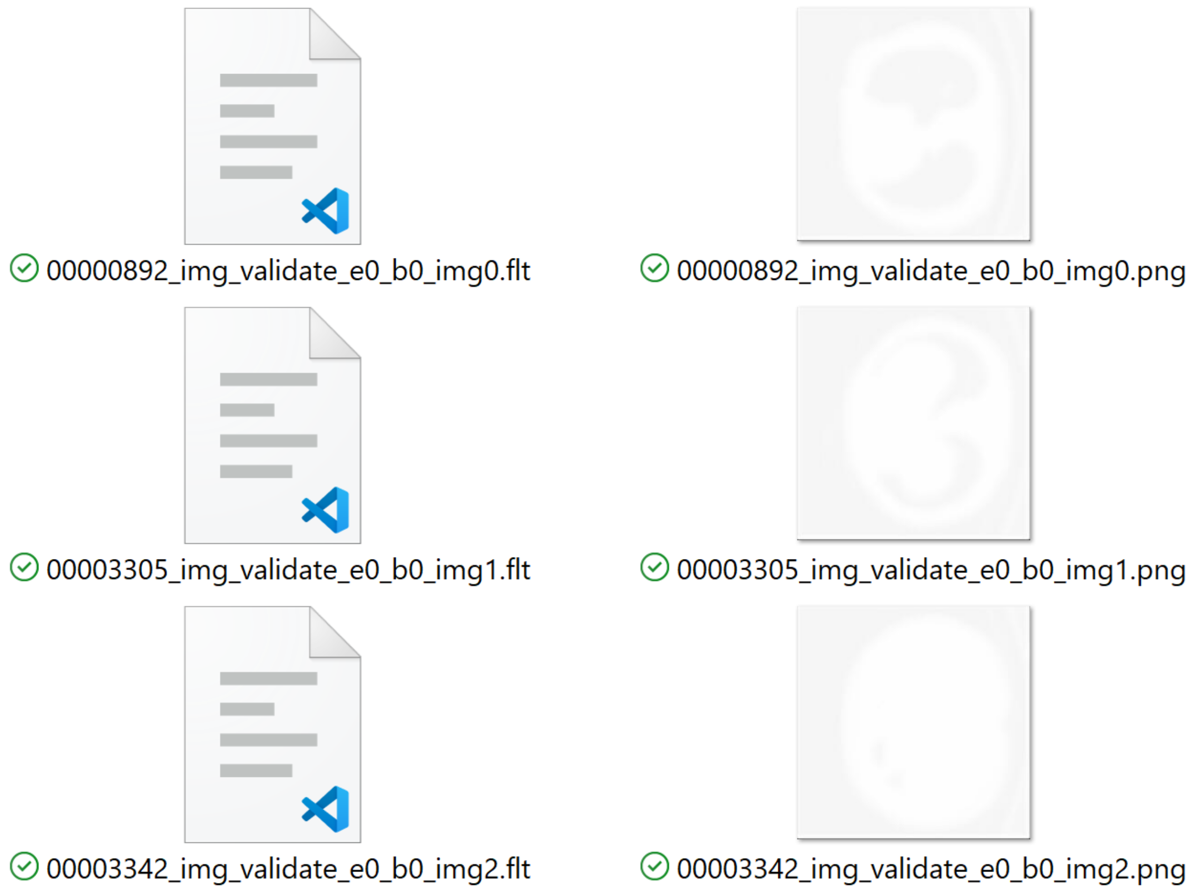
This is due to the use of memory by the model, and how it is being split. This is not relevant to the model.

6) After the user specified number of epochs, this image is printed:



```
Image printed at batch:  0
```

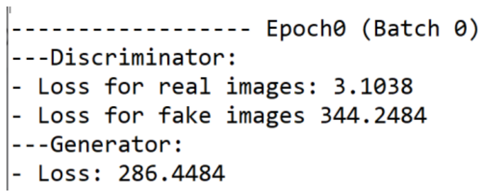
This lets the user know that the sample image has been printed in the sample file (png and float file). The user can at this point see the specific output. An example of how that looks like is:



00000892_img_validate_e0_b0_img0.flt
00000892_img_validate_e0_b0_img0.png
00003305_img_validate_e0_b0_img1.flt
00003305_img_validate_e0_b0_img1.png
00003342_img_validate_e0_b0_img2.flt
00003342_img_validate_e0_b0_img2.png

These images how the name of the image it comes from, the epoch printed at, its batch number, and the image it comes from.

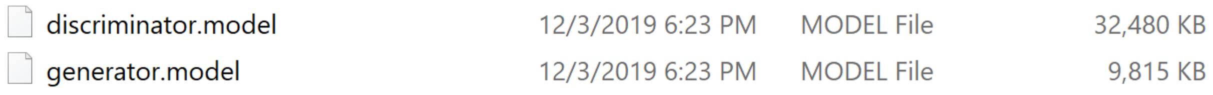
7) After all image are printed, a text report with loss is printed in a text file. This looks like this:



```
------------------ Epoch0 (Batch 0)
---Discriminator:
- Loss for real images: 3.1038
- Loss for fake images 344.2484
---Generator:
- Loss: 286.4484
```

In here, we have the loss for the discriminator and the discriminator at a specific epoch and batch, for all epochs and batches during those epochs.

8) An message is given to the user, letting them know the program is over. The Generator and the Discriminator are saved in a different file. This is how they look like:



discriminator.model	12/3/2019 6:23 PM	MODEL File	32,480 KB
generator.model	12/3/2019 6:23 PM	MODEL File	9,815 KB