Image Restoration Tool
User Manual

CS 499 Section 001, Group 9

Cody Leslie

Adam Brassfield

Kipling Gillespie

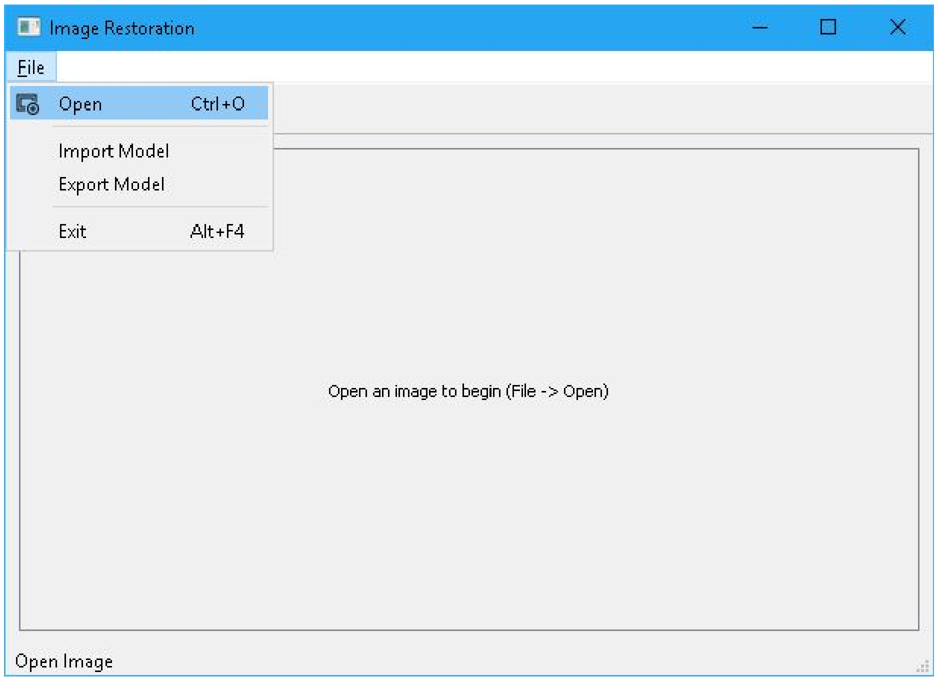
Shelby Oldfield

Table of Contents

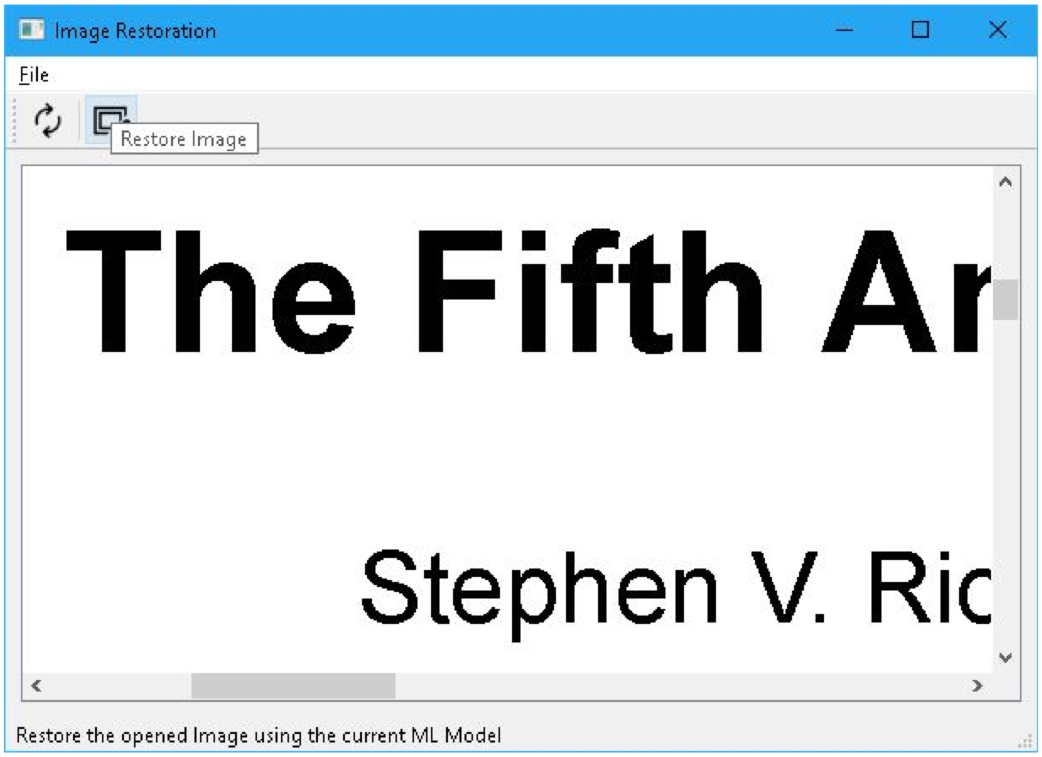
1. Training on a Data Set

a. Graphical User Interface

Run the installed file ImageRestorationApp.py to bring up the Image Restoration window.



Once here, select the File option in the top left corner of the window, which will display another menu. From this, select the Open option, and navigate to the file you want to restore in your file system.



Once loaded, the image should appear inside the large box in the main window. From here, select the Restore Image button, which has an icon with two rectangles and is located beneath the File option. From here, the image should be cleaned (Note: Currently non functional, as issues with machine learning algorithm prevent any response.)

b. Command Line User Interface

Within the command line environment, navigate to the directory with the file ImageRestoreTool.py. Alternatively, you can replace every instance of this filename with an appropriate path to the file. Then, type in the command

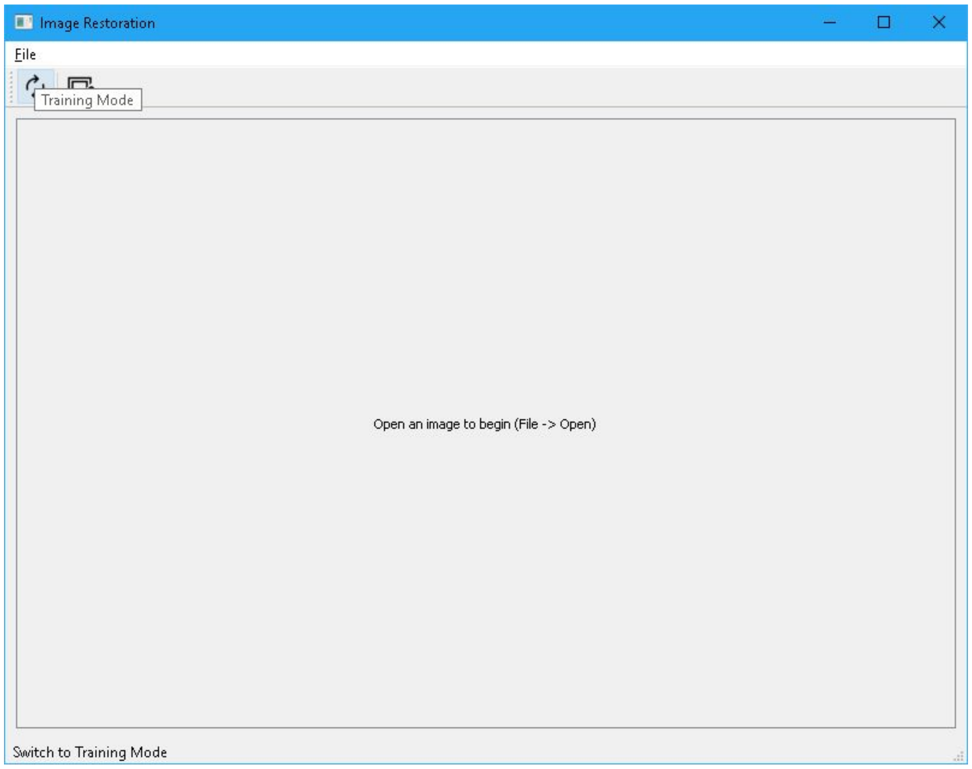
```
>> python ImageRestoreTool.py -r inputImageName outputImageName modelIn
```

Replace inputImageName with a path to the image to clean, and outputImageName with a path to the location to store the cleaned image. (Note: modelIn should be replaced with the file containing a model to execute on. However, since the machine learning is non functional, there is no way to handle this, and it is left to prevent changes being needed when it is completed.) Press the enter key, and you should recieve the Done message indicating the operation was successful.

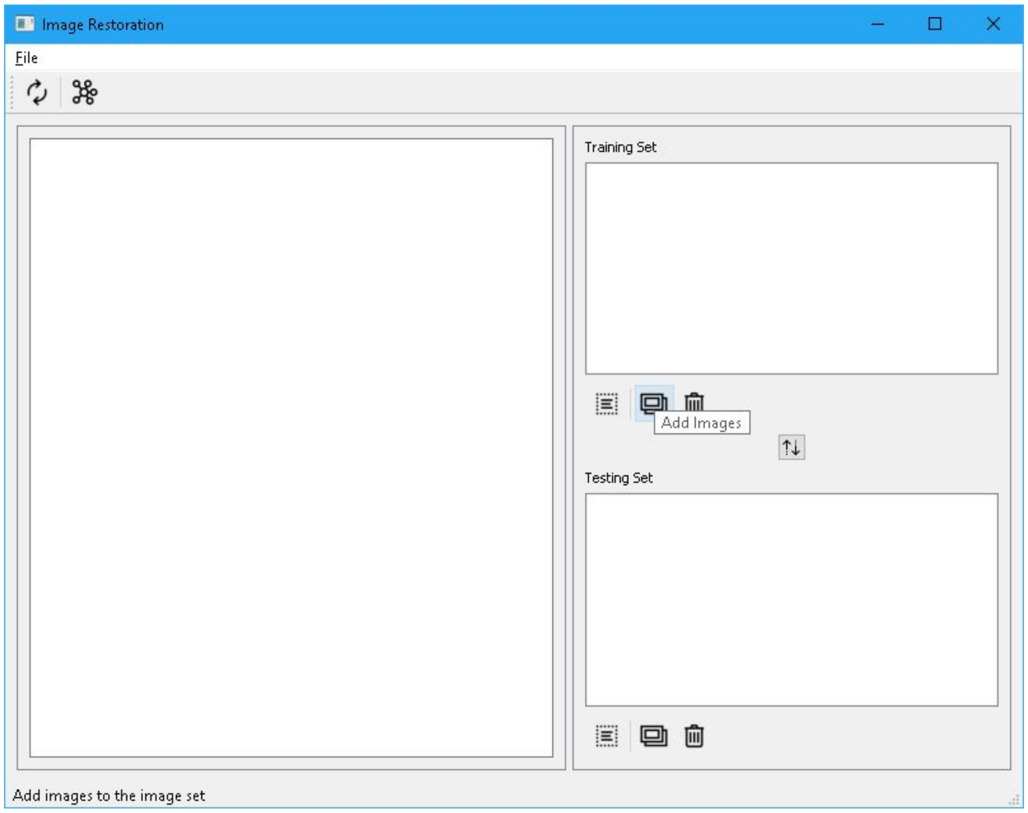
2. Cleaning an Image

c. Graphical User Interface

Run the installed file ImageRestorationApp.py to bring up the Image Restoration window.



Once here, tap the Training Mode button in the top left corner of the window, which has two cyclical arrows as an icon. This will transition the program into the training mode.



From this, tap the Add Training Images button, which has an icon of two overlapping sheets and is beneath the Training Set box, and navigate to the files you want to restore in your file system.

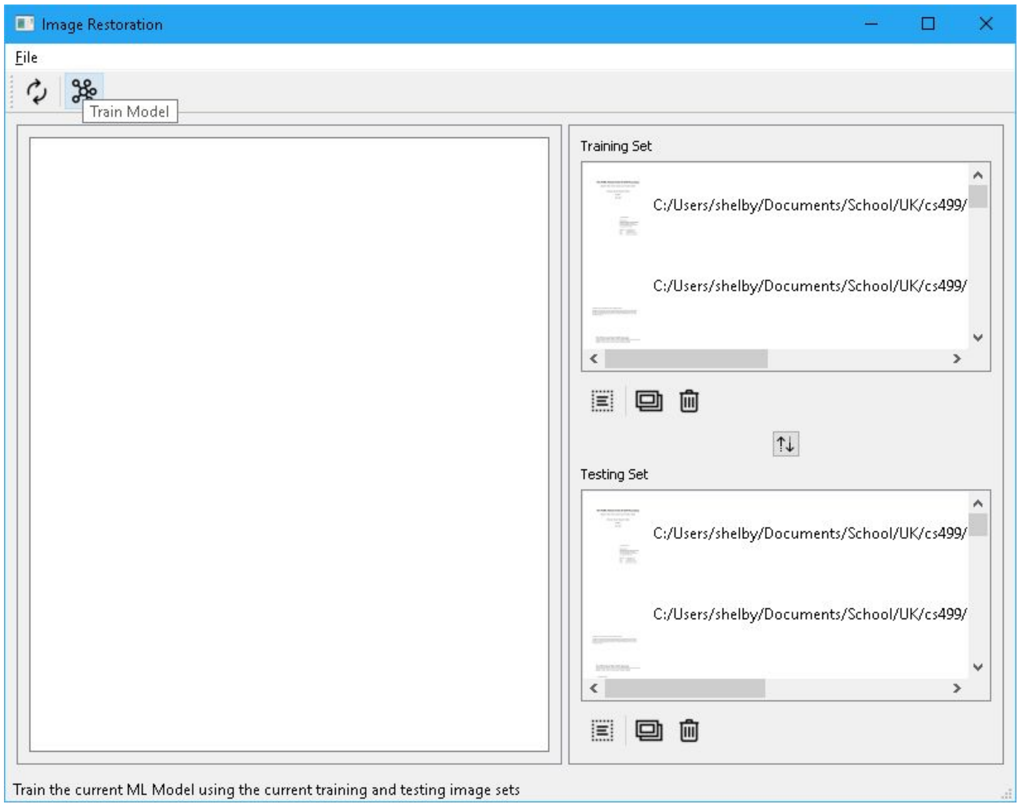


Image Restoration
File
Train Model
Training Set
C:/Users/shelby/Documents/School/UK/cs499/
C:/Users/shelby/Documents/School/UK/cs499/
Testing Set
C:/Users/shelby/Documents/School/UK/cs499/
C:/Users/shelby/Documents/School/UK/cs499/
Train the current ML Model using the current training and testing image sets

Once loaded, the images should appear inside the box directly above the button used to select them. Repeat this process for the images to be used for testing, using the Testing Set box and buttons. Then, select the Train Model button, which has an icon with a web of dots and is located beneath the File option. From here, the model should be trained on the given data. (Note: Currently non functional, as issues with machine learning algorithm prevent any response.)

d. Command Line User Interface

Within the command line environment, navigate to the directory with the file ImageRestoreTool.py. Alternatively, you can replace every instance of this filename with an appropriate path to the file. Then, type in the command

```
>> python ImageRestoreTool.py -t directoryIn directoryGnd modelOut
```

Replace directoryIn with a path to a directory containing the training images with artifacts as jpeg files, and directoryGnd with a path to a directory containing the corresponding training images with no artifacts as png files. These files should match their corresponding file in name, but not extension. (Note: modelOut should be replaced with the filename to output the model that is trained. However, since the machine learning is non functional, there is no way to handle this, and it is left to prevent changes being needed when it is completed.) Press the enter key, and you should recieve the Done message indicating the operation was successful.

3. Implementation

a. Trainer

i. Net

The program is based off of a Convolutional Neural Network that was designed for increasing quality of images when upscaling their resolutions where we heavily sourced our research. We used the Caffe Deep Learning Framework which was developed by The Berkeley Artificial Intelligence Research Lab since it has a python api that is built over a C++ open source library. This gave us the advantage of writing code in Python, but getting the performance strengths of C++.

The Neural Net itself uses 10 layers. The first layer is the input layer which takes in the training set of paired compressed and uncompressed images. A batch size must be defined in this layer and the upper bounds of that size are very dependent on the amount of memory your computer has. Large values will try and load many images at once which could fill your memory and cause your computer to thrash as it accesses its page files. We tested with a batch size of 56. The rest of the convolution layers are each followed by a Parametric ReLU activation function. The second convolution layer is for feature extraction on the image. The third layer is a convolution layer that downscales the features maps. The fourth, fifth, sixth, and seventh convolution layers each map feature. The eighth convolution layer re expands the feature maps. The 9th layer is a deconvolution layer that recombines the feature maps into a single image and does not utilize an activation function. The final layer performs Euclidean Loss calculations between the results of the network and the ground truth uncompressed image. This loss is then back feed through the network to update it’s weights.

This is all defined in a net.prototxt file that Caffe uses to define its networks. There is a lot of room for improvement in our model in this file just from modifying the output, kernel, and stride size at each layer.

ii. Solver

The solver is a feature of the Caffe Deep Learning Framework that allows you script the parameters for training a defined network. It is defined in a solver.prototxt file that defines the max number of iterations that the network should run, it tells the solver which net.prototxt file to use, it tells how often the network should output it’s loss results, and how often it should save a snapshot of the network state along with other features. The learning rate parameter for the network is defined here. This value can have serious impact on the accuracy and amount of time the network takes to run.

b. Cleaner

The purpose of the cleaner is to load the results of training the network and an image with compression artifacts and to run the image through the network for the purpose of cleaning those artifacts. The cleaner uses the same net.prototxt file that is used to define a network with two important differences. The input layer takes a batch size of 1 since only one image is being run at at time. The second difference is that the loss layer needs to be excluded since we only care about the output of the deconvolution layer.

c. Problems

We had a number of problems during the implementation of the machine learning components of this program. Caffe's documentation does not cater to people with a cursory understanding of machine learning and it took us a long time to get enough of an understanding to start coding. Many of the sources we found for learning how to use Caffe were either in C++, for using its command line utilities, or using it with MatLab. When we got python errors from Caffe the best source was StackOverflow, but we only found example uses where we had to infer how the code actually worked.

We had trouble formatting training data in a way that the correct compressed images were paired with their appropriate uncompressed image. Without being able to do this, loss could not be calculated and the network would not learn. We eventually learned how to preprocess images into HDF5 files, that would allow images to be broken down. We still had problems making sure our input was in the correct format. Images loaded into numpy arrays using OpenCV are ordered as [Height, Width, Channel], but Caffe needs data formatted as [N, Channel, Height, Width] where N is the batch size.

4. Bibliography

Notebook on Nbviewer
http://nbviewer.jupyter.org/github/BVLC/caffe/blob/master/examples/00-classification.ipynbl

V., A. (2017, June 30). An Example of a Convolutional Neural Network for Image Super-Resolution—Tutorial. Retrieved November 11, 2018, from https://software.intel.com/en-us/articles/an-example-of-a-convolutional-neural-network-for-image-super-resolution-tutorial

jia2014caffe, Author = {Jia, Yangqing and Shelhamer, Evan and Donahue, Jeff and Karayev, Sergey and Long, Jonathan and Girshick, Ross and Guadarrama, Sergio and Darrell, Trevor}, Journal = {arXiv preprint arXiv:1408.5093}, Title = {Caffe: Convolutional Architecture for Fast Feature Embedding}, Year = {2014}

C. Dong, C. C. Loy, K. He and X. Tang, "Learning a Deep Convolutional Network for Image Super-Resolution," 2014.

C. Dong, C. C. Loy and X. Tang, "Accelerating the Super-Resolution Convolutional Neural Network," 2016.