Project Report

Motivation and Impact

Earlier this semester, we learned about the frequency domain and how it could be adopted in downsampling/downscaling images. However, we did not dig much into upscaling images. In this project, I would look to get more familiar with upscaling images. More specifically, I am looking into the SRCNN method for image upscaling.

Unlike other upscaling filters, SRCNN adopts a convolutional neural network in order to predict the upscaled image from the low-resolution input. According to the paper we referred to, the result is much better than the traditional filters, bicubic or sparse-coding-based method as the examples, that could be visually distinguished in many cases.

The improvement of the upscaling came at the cost of training the models and the longer runtime of upscaling, which is the major issue preventing the method to be adopted more widely and to replace the traditional filters. Furthermore, for real world usage, the improvements brought by SRCNN compared to traditional filters are too trivial compared to the cost of implementing SRCNN. The runtime is more likely to be improved with hardware since the software part of it is straightforward and hard to be optimized.

Therefore, this project mainly focused on two parts. The first is to implement a working SRCNN in python. Despite that there are existing implementations out there, most of them are either done years ago, or using cpu only. I aim to rewrite the whole thing in tensorflow for better compatibility with modern hardware and thus faster computation.

The second is to train the model with different kinds of images. Specifically, I tried with photos of the natural environment, photos of streets, and animation artworks. This part is inspired by the well-known Waifu2x project, which adopts the SRCNN model and is trained specifically for upscaling the animation pictures. Preliminary research has shown that training models for a specific genre of pictures could further improve the upscaling result, and we would like to compare how each of these models work with each category of the images.

By looking into these two parts, we seek for improvements in both upscaled image quality and computing speed. This should allow the SRCNN to be adopted in a wider context.

Approaches

For the implementation part, I looked into the paper as well as implementation in other languages for the reference. Specifically, I looked mostly at the Waifu2x project, which was written in Lua. I've also looked into the vulkan implementation of Waifu2x, but it was too time consuming to understand the vulkan, and I eventually focused mostly on the Lua version for reference.

The model was built with keras and tensorflow. I used the earlier version of tensorflow, namely tensorflow 1.x, because I have worked with this version of tensorflow before.

The most difficult part of the implementation is to get the model correct, which took the most of the time in the implementation. Beyonds that, I wrote a set of helper functions for importing and exporting images, as well as saving and loading models. This allows the model to be reused for different testing input.

As for the training data, based on what the authors of the paper did, I picked 60 pictures for each category from what the photos I took or the images I saved from the Internet in the past years. Since I'm only taking the images for personal and experimental usage, the images from the Internet should be fine.

Each group of pictures are divided into two groups with 55 in one the other 5 in the other. The 55 pictures are used for training and the rest for testing. The 5 images are also used to cross-test the models so as to reveal the difference of the models trained with different input.

Results

For the implementation, running the model with gpu is significantly faster than running with cpu. I compared my implementation with the two other implementations I have mentioned above, the vulkan one and the lua one. Due to the training being too time consuming, I only compared the upscale run time.

[runtime pic]

My implementation takes basically the same time as the vulkan implementation, and is much faster than the lua version due to gpu acceleration.

As for the difference among models, here are some sample results I had.

Model for Nature Scenes:



Model for City Scenes:



Model for Animation Images:



By zooming in and comparing the details, we could see that the model trained for the two photo genres performed similarly on the testing image of each other. However, the model trained for animation artworks is significantly different. The latter one tends to give out sharper edges and solid colors, which are ideal for the animation artworks and looks better than the result given by the model for the photos. On the other hand, the model for the photos tends to preserve more detail when upscaling photos, while inducing more noise with the animation images.

Implementation Details

The implementation could be briefly broken into two parts. One was to implement the read image and preprocessing part, and the other was to write the model. The image reading was done with the imagio library and scipy for transcoding.

The model is built directly with tensorflow, packaged in a tensorflow session. It is thus easy to run, save, and load with the session infrastructure.

Challenge

The most challenging part of the process, as mentioned above, is to get the model implemented correctly. Despite that the model only takes a few lines of codes, it required a good amount of research, as well as trial and error, to figure out if the model was correct. The process was pretty

time consuming, as I had to go over the documentations of tensorflow to make sure the model was correct.

Other than that, the rest of the program was simple, as all I needed to do was to read the image, save and load the model. Still, the overall process was challenging and took quite some time to accomplish.