Image Super Resolution Using Deep Convolutional Networks

Stephen Salerno CS 445 Fall 2015

My project was about neural networks, and how to use them to upscale images to “super resolution”, while maintaining detail and clarity in the image. My method followed the paper “Image Super Resolution Using Deep Convolutional Networks” by the Chinese University of Hong Kong and Microsoft Research. I also took inspiration from Waifu2x by nagadomi, and used his trained neural network model for my project.

Through my project I demonstrated how using a deep convolutional neural network, we can use deep learning to create an end-to-end mapping between low and high resolution images. This means that as long as we have a neural network trained with similar images, we can recover high resolution artwork using only a low resolution input.

How I did it

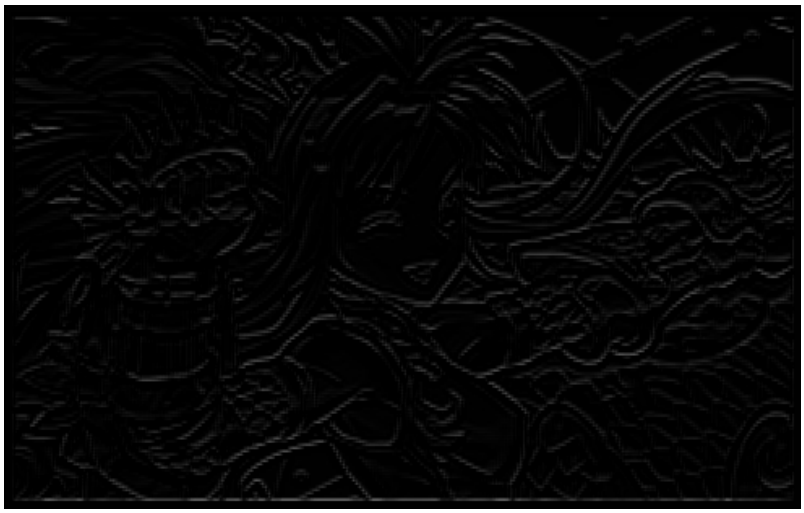
To start, I needed to make sure I had a neural network trained to use appropriate images. For this project I used the neural network from the waifu2x project, which was trained with 6,000 high resolution anime art style images. This would allow me to upscale anything in an anime art style or similar. The neural network there ran on the Torch scientific computing framework, but I used the JSON export tool to obtain the neural network into a JSON file.

Next, I tried to load the neural network into the MATLAB workspace. It turned out to be much more difficult than anticipated, as any JSON parser I used took over 20 minutes to parse the massive JSON file into MATLAB data. I ended up getting around this by saving the parsed JSON to a matfile, and in my code I now simply load the data during each execution from the matfile, which will be much faster.

The next step was to prepare the image for the algorithm. I first upscale the image naively to 2x resolution, using nearest neighbor. I then have to convert the RGB image to YCbCr, since this implementation only works on a single channel, so I work with the luminance channel when running through the network. I also add some padding to the image to fix some problems with convolutions in later steps.

The algorithm starts at the first layer of the neural network. The network is divided into 7 layers, each with nInputPlane and nOutputPlane parameters. Each layer also has a set of weighting filters, each corresponding to an output plane, and containing filters for each input plane. And there are also biases that are used after the filtering step.

In practice, the first layer works by taking the luminance image input as the first input plane, then performs 32 separate convolution filtering steps on it, adding the bias parameter, and saving those to a cell for the next layer. These output planes are known as feature maps, which appear similar to the following image:



The feature maps are what help the neural network interpolate image detail throughout the neural network.

Taking the 32 output planes from the first layer, these are used as input planes to the next layer. The second layer in the model has 32 input planes and 32 output planes, to further refine the feature mapping. However, when performing the convolution steps this time, all 32 input planes are used for each output plane. This means that each weighting parameter contains a filter for each of the input planes, which are then filtered and averaged through a partial sum, plus the bias parameter, to give us the desired output planes.

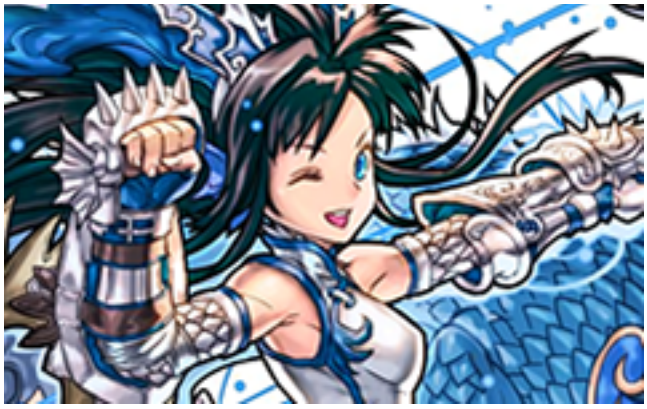
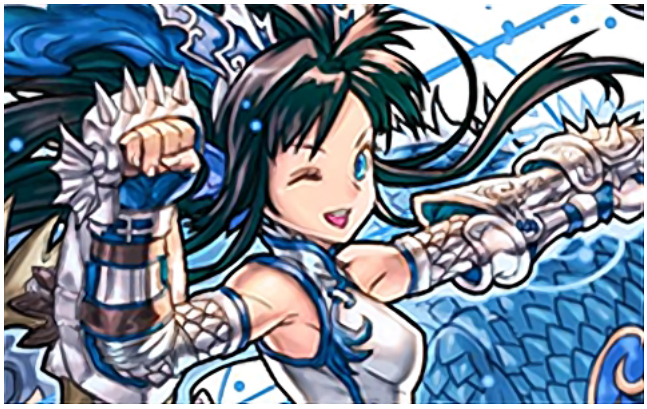
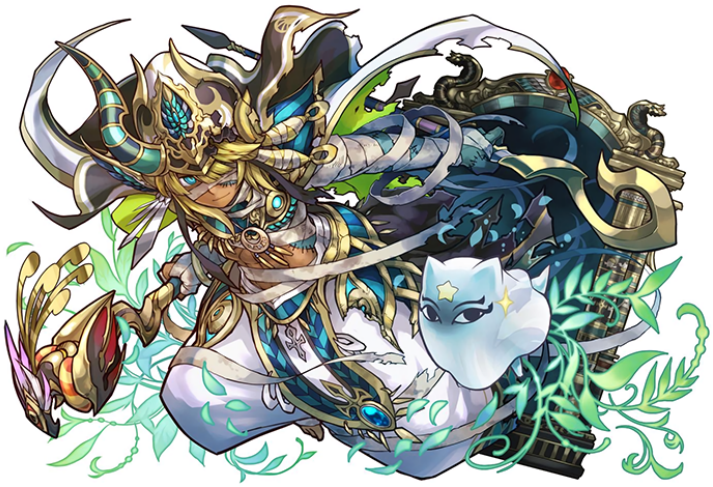
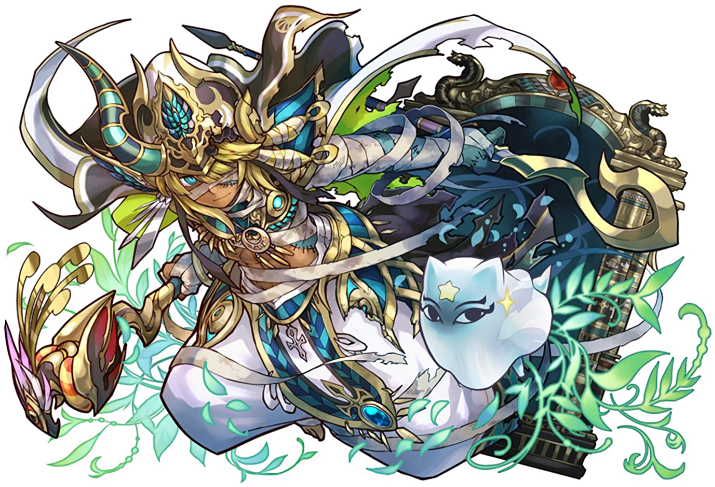
These steps repeat for the rest of the layers of the neural network. The network escalates planes until the end, which is the reconstruction step:

Layer 1: 1 input Plane -> 32 Planes (32 convolutions)
Layer 2: 32 Planes -> 32 Planes (1,024 convolutions)
Layer 3: 32 Planes -> 64 Planes (2,048 convolutions)
Layer 4: 64 Planes -> 64 Planes (4,096 convolutions)
Layer 5: 64 Planes -> 128 Planes (8,192 convolutions)
Layer 6: 128 Planes -> 128 Planes (16,384 convolutions)
Layer 7: 128 Planes -> 1 output plane (128 convolutions)

Layer 7 is where the planes are reassembled into the final luminance image output. This output will be the same resolution as the input image that was naively scaled, but with greater detail and clarity throughout, more similar to the ground truth of the higher resolution artwork.

After the final layer, the padding is removed from the image, it is replaced into the luminance channel of the input YCbCr image, and that image is converted back to RGB to give the final output super resolution image.

Results

I was overall very happy with my results. Since it is a neural network and mostly prediction based, the results can vary somewhat in quality, but as long as the input image matches what the neural network was trained for, the results should always be good, sometimes even nearing the ground truth image.

Troubles/failures

I did not notice any significant failures with any of the artwork I tried, but one notable problem is the speed of my implementation. My first problem was the sheer size of the neural network JSON file, but I managed to alleviate that problem by using a matfile as a shortcut. However, as I mentioned above when talking about the layers of the neural network, this algorithm involves an extremely large number of convolution steps, 31,904 performed on the input image. This can take very long on even the best CPUs. I found when running through my first example above (“Karin”), a 240 x 150 image, it took around 10-15 minutes to complete the algorithm. This is already pretty long, but found that when trying somewhat larger images, like the second

example (“Osiris”), a 750 x 469 image, it took me over an hour to complete all the convolution steps. I tried to alleviate the problem by using a library that used FFT for the convolutions, and also used the GPU to speed up calculations, but didn’t see much improvement. I also tried to implement multithreading for the calculations of each plane, but wasn’t able to see a significant improvement in run time. I believe that implementing this algorithm in a faster language like C++, preferably using multithreading, or even using straight GPU calculations, would’ve probably helped the runtime of this algorithm.

References and sources

Original Paper:

Chao Dong, Chen Change Loy, Kaiming He, Xiaoou Tang,
"Image Super-Resolution Using Deep Convolutional Networks", http://arxiv.org/abs/1501.00092

Inspiration, and source of neural network model:

nagadomi, “Waifu2x”, https://github.com/nagadomi/waifu2x

Matlab Libraries Used:

Qianqian Fang, JSONLAB, http://www.mathworks.com/matlabcentral/fileexchange/33381-jsonlab--a-toolbox-to-encode-decode-json-files-in-matlab-octave

Bruno Luong, FFT-based Convolution, http://www.mathworks.com/matlabcentral/fileexchange/24504-fft-based-convolution

Image sources:

Puzzle & Dragons, Gungho Online Entertainment

Digimon Adventure Tri, Saban Capital Group