Machine Learning Based Microscopic Imaging Quality Enhancement

Wanjun Gu

University of California, San Diego

University of Tokyo
Department of Chemistry

UTRIP 2018

ABSTRACT

Convolutional neuron network (CNN), as a category of machine learning techniques, is used to enhance the quality of microscopic imaging. For this project, a network is designed for enhancing the image quality of human tissue pictures. The objective neuron network can enhance the image quality by increasing the image sharpness and adding in more details to the pixelated input images. Network also creatively uses "Hour-Glass" model and "Convoluted Feature loss" function to maximize performance and speed.

General Background Information

A big problem encountered by concurrent microscopic imaging technique is the lack of high-quality images. The motion blur of images due to fast cell flow, the low-resolution image output due to limitation of sensor size and insufficient light intensity due to compromises made with the choice of excitation beam frequency can cast problems to the later steps of data analysis. The accuracy of flow cytometry, cell classification and other research objectives will dramatically decrease if certain resolution standard is not met. In response to this problem, this project introduces a machine-learning based image quality enhancing algorithm to enhance the quality of the microscopic images.

Machine learning is a statistical data analysis technique which can automate model building and model fitting. A typical machine learning procedure requires training data collection, model framework design, model training and model prediction. In this specific project, convolutional neuron network is the key component of the model framework.

Convolutional Neuron Network (CNN), as suggested by name, is a bionic-inspired network design specializing in image processing and computer vision. The fundamentals of convolutional neuron network are not different from those of mathematical convolution operation. Nevertheless, more than convolving digits, the framework parameterizes the convolution matrices to adjacent image pixel information to form convolutional layers. Hidden layers are then formed by stacking multiple convolutional layers together. In this way, non-linear statistical connections are made among pixels in the image subjective to researchers' interest. Over training, the network can be optimized to find a certain pattern within the given training data base. With the pattern established, the model can be then applied to new input images to introduce desired change to the input.

For this project, a desired model should be able to increase the image resolution as well as its sharpness and finally, able to increase the accuracy the later sorting algorithm. The objectives of the project can be achieved in 6 steps:

- Basic Statistical image processing algorithm design
- Tissue Data set construction
- Deep Learning Model Design
- Loss Function Design
- Model training
- Parameter Adjustment

Statistical Image Processing algorithm

Signal noise cancellation and enhancing the signal from primary data are usually the very first steps of any data processing processes. For the specific purpose of flow cytometry and high-speed microscopic imaging, a high-pass filtration was performed on the acquired images. The cut-off signal intensity was determined by manually collecting the cell image signal intensity and its percentile. In this way, all the non-cell pixels will be dimmed. On top of that, the algorithm will then normalize the signal, which can most of times increase the brightness and contrast of the image as shown in figure 1.

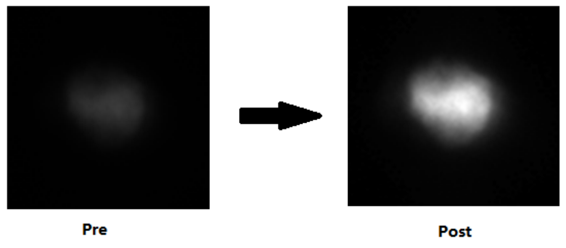


Figure 1
Pre and post Signal noise cancellation and normalization

Besides the most basic signal filtering and noise reduction, there are two most prevalent ways to increase the image digital resolution.

Algorithmic Enhancement:
In this algorithm, the resized image can be achieved by resizing each pixel. For instance, to obtain an image that is 2 times of its original size, each pixel in the image needs to be expanded to 2 times of their original sizes. The process can be visualized in figure 2.

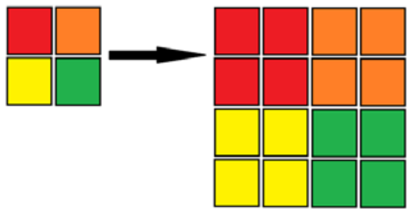


Figure 2

Bicubic Enhancement:

In this algorithm, when an image is resized, the algorithm takes the average value of the 2 adjacent pixels to generate the pixel in between. A visualization of this process is shown in figure 3

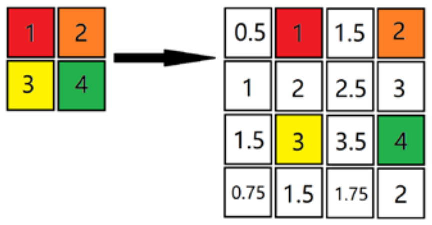


Figure 3

Tissue Dataset Construction

To train model to best fit the future input images, the dataset with the best representation must be found. In this case, the training dataset must satisfy a few requirements. First, the dataset must be acquired from open-source repositories. For this experiment, the image data was acquired from open source image databases including Google image, Bing image, ImageNet and Baidu image. The image acquisition process is completed automatically using an internet crawler algorithm which generate URL batches for download. Second, the dataset must be well classified. For this experiment, as the objective is to increase the resolution of microscopic images of cells and tissues, the whole database is consisted of pictures of human cells and tissues. To achieve the accuracy of classification, not only the keywords inputted to the search engine has been scrutinized, the downloaded pictures were as well all manually checked to make sure they belong to the same valid category. After the acquisition and quality check of the check, one more crucial step for data pre-processing is to reshape the data. To be able to feed the data into the neuron network, we need to reshape the image data to array-lists with certain dimensions.

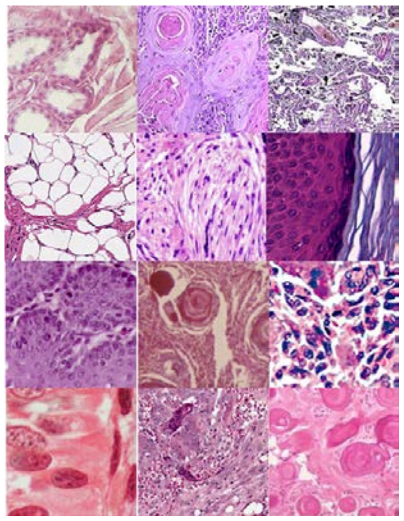


Figure 4
Sample pictures from dataset HRLR300 Human Tissue dataset

Model Design

As convolution layers are predominant components of the network, the most paramount parameters of the networks are the parameters in the convolutional layers. With that acknowledged, the rest part of the research is about try-and-error. In the research, we tried different network design with convolutional layers with different number of filters and different convolutional kernel sizes. Among all the models that have been tried, some are well-known models mentioned in various research papers that are said to be advantageous in performance. We tabulated the speed and accuracy comparison of the models in the appendix. Besides convolutional layers, another essential component of the model is the activation functions. During our modelling processing, we tried the combination of different activation functions and ended up using “Relu” activation function for the middle layers and “arctan” activation function for the output. This way, we can effectively add in variations to the model without overfitting. What has been determined by experimental trials is a model combining the advantages of various image super-resolution network models. We name our model “Hour-Glass” model, which is a model consisted of decreasing number of filters and olive-shape distributed sizes of convolution kernels. The advantage, which will be parameterized later, of this model can be concluded by high-speed as well as high performance.

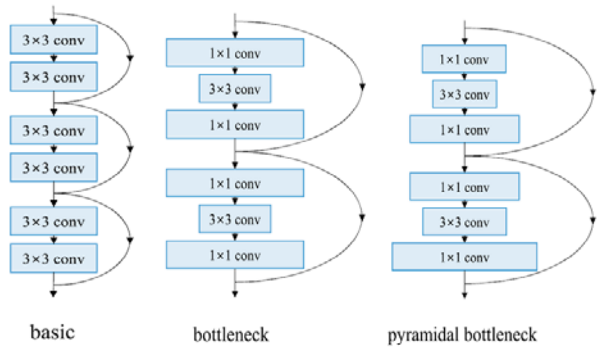


Figure 5
Some other Neuron Network structures that are claimed to have good performance

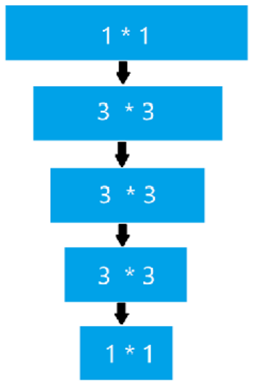


Figure 6
Hour-Glass Model

Loss function Design

For the model to be optimized after generations and generations of training, a function is needed to evaluate the level of optimization of the model by looking at the predicted values of the model and the real values of the model. This function is known as the loss function. At the same time, the function that works along the lost function to correlate the numerical lost and overwrite of the parameters in the model is called the optimizer function. For this project, the most commonly used optimizer “Adam” is used. But for the lost function, I tried different lost function including the mean-squared-error (MSE) lost function (Formula shown in Figure 7)

MSE Loss Formula:

$$l^{SR}_{MSE} = \frac{1}{r^2WH}\sum_{x=1}^{rW}\sum_{y=1}^{rH}(I^{HR}_{x,y} - G_{\theta_G}(I^{LR})_{x,y})^2$$

Figure 7

and a customized lost function, the convoluted-feature-map-loss function (Shown in Figure 8). As a result, the convolutional-feature-map-loss function yields marginal increase of performance while the mean-squared-error function is faster.

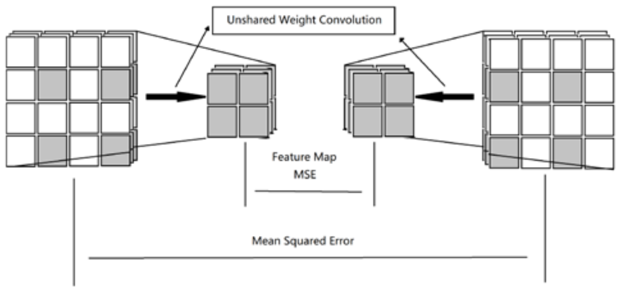


Figure 8
convoluted-feature-map-loss function

Result and Comparison:

After applying the trained model to the test images with low resolution, a predicted high-resolution image is generated and compared to the original picture. (Figure 9)

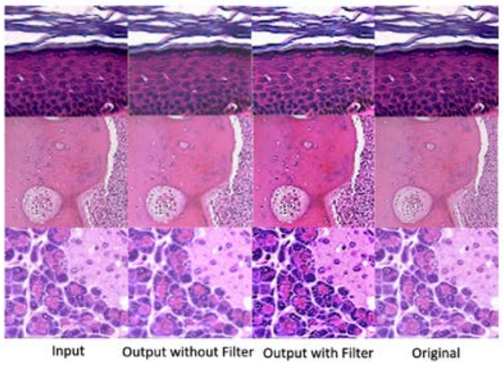


Figure 9

As we can see from the sample output, the output image, compare to the original picture, is less pixelated with more details around the edges. (Comparison shown in figure 10) Most Importantly, we notice that the network can add some details that do not exist to the output image. Most of the times, the added information can well correspond to the output image. The output image is sharper, containing more details and sometimes with higher contrast.

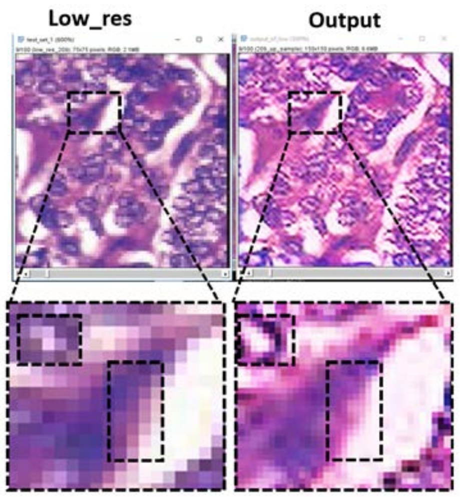
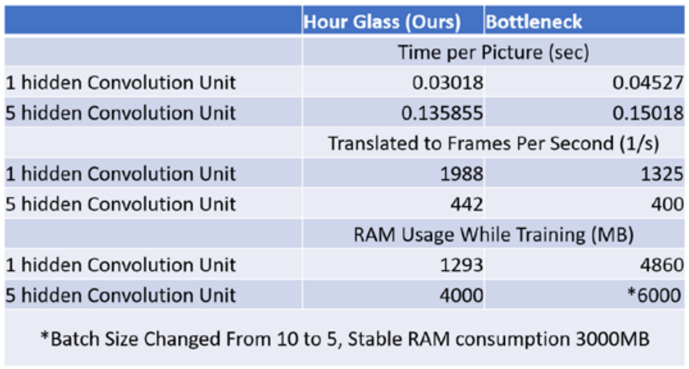


Figure 10

Speed wise, the Bottleneck network structure is one of the fastest networks when it comes to the image super-resolution. With that said, when compared to the Hour-Glass model, it is still slower with higher RAM consumption as shown in table 11.



	Hour Glass (Ours)	Bottleneck
	Time per Picture (sec)	
1 hidden Convolution Unit	0.03018	0.04527
5 hidden Convolution Unit	0.135855	0.15018
	Translated to Frames Per Second (1/s)	
1 hidden Convolution Unit	1988	1325
5 hidden Convolution Unit	442	400
	RAM Usage While Training (MB)	
1 hidden Convolution Unit	1293	4860
5 hidden Convolution Unit	4000	*6000
*Batch Size Changed From 10 to 5, Stable RAM consumption 3000MB		

Table 11
Speed and RAM usage Comparison

However, it is undeniable that both models can achieve real-time image(video) quality enhancement as the frames count per second for both models are larger than 100 even with 5 or more hidden convolution units. This is to say, both networks can be theoretically used as real-time video quality enhancement tools.

Future Research Plan

The next step of the research is to continue optimize the model. The focus of optimization will be about the neuron network and the loss function. For the neuron network, different activation functions should be tried and compared as well as different hidden layer counts. Theoretically, assuming nearly identical performance, model is more optimized with less hidden layers since more hidden layers are corresponded to heavier RAM consumption and longer calculation time. As for the loss function, it is necessary to try and compare mean-absolute-error loss function and its derivatives to see if linear loss functions can increase the sharpness of the image more. Another important thing to work on based on this neuron network is to build user interface for training, prediction and model optimization.

In a long run, the model should be redesigned to be capable of 3-dimensional image quality enhancement and most importantly, the enhanced images should theoretically have higher object classification accuracy.

Acknowledgement:

This project can never be finished without the help of my Mentor Ph.D student Kangrui Huang, my team leader Dr. Hideharu Mikami and Prof. Keisuke Goda. Besides, I want to also thank Dr. Hirofumi Kobayashi for guiding me through the basics of deep learning and image linear filter, Fangzheng Zhang for helping me with network design, Mortis Huang for helping me setting up the GPU development Environment and all the summer interns for giving me tremendous support.

References

Dongyoon Han. et.al Deep Pyramidal Residual Networks. 2017

Jin Yamanaka. et.al Fast and Accurate Image Super Resolution by Deep CNN with Skip Connection and Network in Network. 2017

Chao Dong. et al. Image Super-Resolution Using Deep Convolutional Networks. 2015

Yair Rivenson et.al. Deep learning microscopy. 2017

Khizar Hayat Member. Super-Resolution via Deep Learning, IEEE, 2018

Eric M. et.al. In Silico Labeling: Predicting Fluorescent Labels in Unlabeled Images, 2018

3rd party R packages used: *Keras, ggplot2, imager, EBImage, raster, magick, Rstudioapi, fields, plotly*

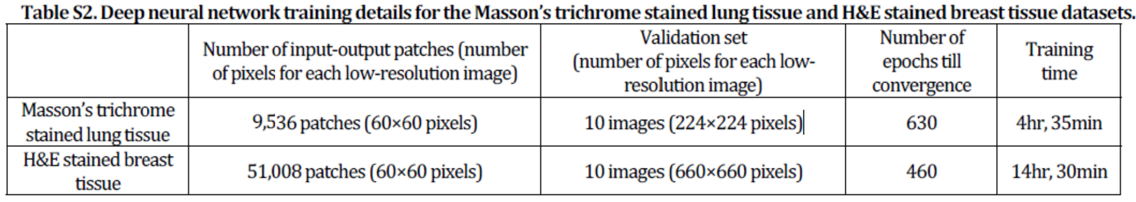
ImageJ Github Repo: https://github.com/imagej/imagej

Neural-enhance-master Github Repo: https://github.com/alexjc/neural-enhance

Srgan-master Github Repo: https://github.com/tensorlayer/srgan/blob/master/README.md

Appendix:

Table S2. Deep neural network training details for the Masson's trichrome stained lung tissue and H&E stained breast tissue datasets.



	Number of input-output patches (number of pixels for each low-resolution image)	Validation set (number of pixels for each low-resolution image)	Number of epochs till convergence	Training time
Masson's trichrome stained lung tissue	9,536 patches (60×60 pixels)	10 images (224×224 pixels)	630	4hr, 35min
H&E stained breast tissue	51,008 patches (60×60 pixels)	10 images (660×660 pixels)	460	14hr, 30min

Optica CNN Training Time Using Bottleneck Neuron Network structure

The time cannot be directly compared to our experimental result because the devices used were different.

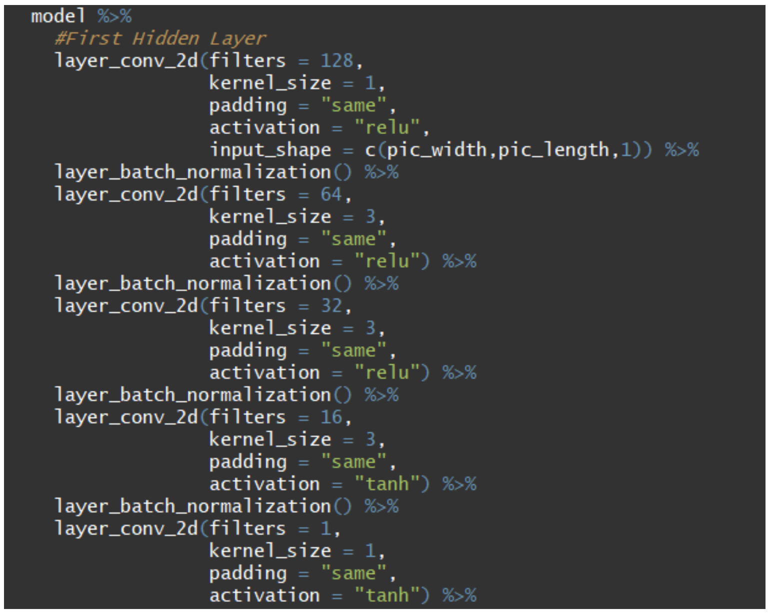
Specifications for the Machine used for Neuron Network Training:

- CPU: Intel Xeon E5-2660 16 Core
- RAM: 16GB DDR3
- GPU: NVIDIA Quadro P2000
- VRAM: 5GB (5120MB) DDR5
- Default Batch Size: 10

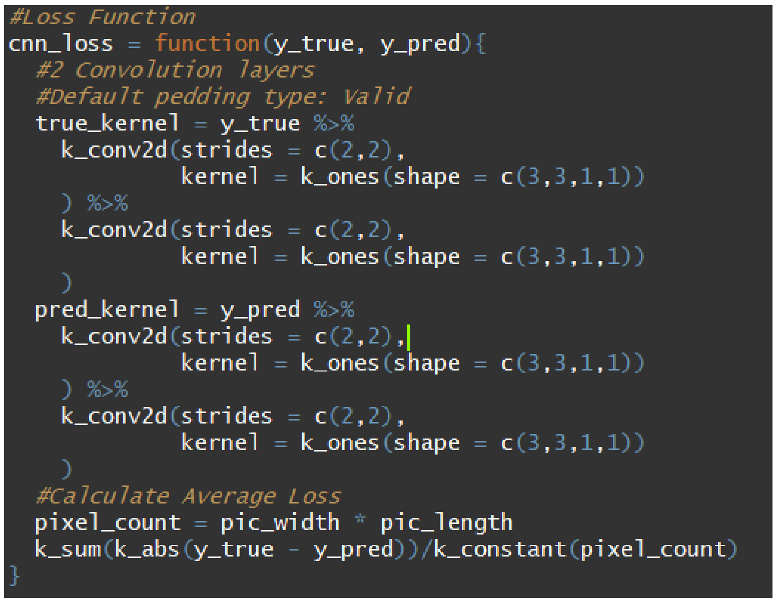
Part of the Code in R:

(Full Project Code available on GitHub: https://github.com/Broccolito/CNNTissueSR)

Model Definition:


```
  model %>%
    #First Hidden Layer
    layer_conv_2d(filters = 128,
                  kernel_size = 1,
                  padding = "same",
                  activation = "relu",
                  input_shape = c(pic_width,pic_length,1)) %>%
    layer_batch_normalization() %>%
    layer_conv_2d(filters = 64,
                  kernel_size = 3,
                  padding = "same",
                  activation = "relu") %>%
    layer_batch_normalization() %>%
    layer_conv_2d(filters = 32,
                  kernel_size = 3,
                  padding = "same",
                  activation = "relu") %>%
    layer_batch_normalization() %>%
    layer_conv_2d(filters = 16,
                  kernel_size = 3,
                  padding = "same",
                  activation = "tanh") %>%
    layer_batch_normalization() %>%
    layer_conv_2d(filters = 1,
                  kernel_size = 1,
                  padding = "same",
                  activation = "tanh") %>%
```

Loss function:


```
#Loss Function
cnn_loss = function(y_true, y_pred){
  #2 Convolution layers
  #Default pedding type: Valid
  true_kernel = y_true %>%
    k_conv2d(strides = c(2,2),
             kernel = k_ones(shape = c(3,3,1,1))
    ) %>%
    k_conv2d(strides = c(2,2),
             kernel = k_ones(shape = c(3,3,1,1))
    )
  pred_kernel = y_pred %>%
    k_conv2d(strides = c(2,2),
             kernel = k_ones(shape = c(3,3,1,1))
    ) %>%
    k_conv2d(strides = c(2,2),
             kernel = k_ones(shape = c(3,3,1,1))
    )
  #Calculate Average Loss
  pixel_count = pic_width * pic_length
  k_sum(k_abs(y_true - y_pred))/k_constant(pixel_count)
}
```