Additive Manufacturing Defect Detection using Neural Networks

James Ferguson
Department of Electrical Engineering and Computer Science
University of Tennessee Knoxville
Knoxville, Tennessee 37996
Jfergu35@vols.utk.edu

***Abstract*—Currently defect detection in additive manufacturing is predominately done by traditional image processing, approximation, and statistical methods. Two important aspects of defect detection are edge detection and porosity detection. Both problems appear to be good candidates to apply machine learning. This project describes the implementation of neural networks as an alternative method for defect detection. Results from current methods and the neural networks are compared for both speed and accuracy.**

I. Introduction

The goal of this project is to apply neural networks to efficiently and more accurately detect flaws within 3D printed objects compared to currently used methods. Detecting these flaws are important for both quality control and certification of 3D printed objects. There are two main parts to this project, edge detection and porosity detection. The edge detection portion is used to detect geometric accuracy within the printed object. Examples of porosity and geometric features are highlighted in the near-IR image in Figure 1. The input data for both problems are the same, a StereoLithography (STL) slice image for each layer and the corresponding Near-IR image of the layer that is captured after it is printed. An example of a STL slice is shown in Figure 3 with the matching near-IR image is shown in Figure 4. Both images are too large to be able to be fully displayed in this paper so subsections are shown instead.

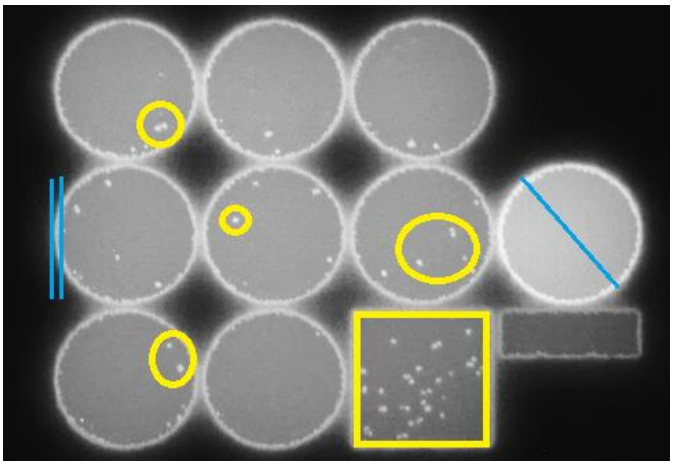

Fig. 1. Examples of porosity (yellow) and geometric features such as edge thickness and object dimensions (blue)

II. Caffe

An important component to this project is the deep learning framework Caffe [1]. Caffe is developed by the Berkeley Vision and Learning Center (BVLC) and is open source hosted at http://github.com/BVLC/caffe. Caffe was chosen to be used for this project for two main reasons – ease of use and speed. Neural Networks can be defined using a plain text prototxt format which allows for quick testing of different network architectures. An example of a Caffe prototxt file is shown in Figure 2.

Caffe also handles a highly optimized implementation of the network for both the CPU and GPU. This project uses the GPU version along with NVIDIAs CUDA Deep Neural Network library (cuDNN) which boosts Caffe's performance. Officially only the Linux and OS X platforms are supported. For compatibility with other software the first task of this project was to port Caffe to Windows. Unofficial documentation for installing an older version on Windows was located on BVLC's GitHub and served as a guide.

```
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  data_param {
    source: "input_leveldb"
    batch_size: 64
  }
}
layer {
  name: "ip"
  type: "InnerProduct"
  bottom: "data"
  top: "ip"
  inner_product_param {
    num_output: 2
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip"
  bottom: "label"
  top: "loss"
}
```

Fig. 2. Simple prototxt file with a input layer, fully connected layer, and an output layer.

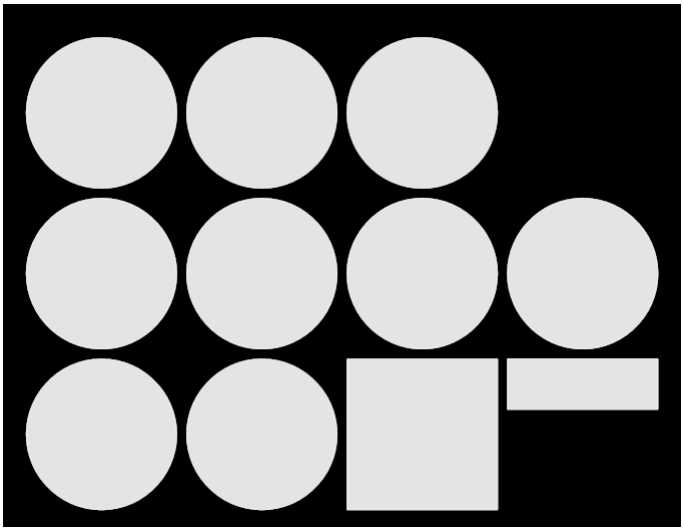

Fig. 3. Section of STL slice.

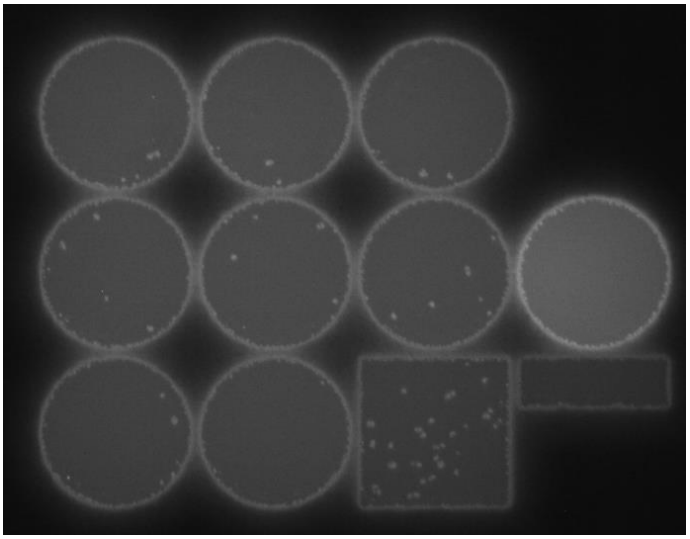

Fig. 4. Matching Section of near-IR image.

III. Edge Analysis

Edge analysis is important for two main reasons; geometric accuracy and for more accurate porosity detection. The overall goal is to be able to efficiently and more accurately detect the edges using neural networks compared to current methods. The edge detection is a four step process that is described below. The steps can be divided into 2 categories, preprocessing and detection. For the detection stage two separate methods were implemented, the current way edges are detected and the neural network detection.

A. Preprocessing steps

The preprocessing stage contains three steps, extract the contours, compute the local normal to the contour, and extract the pixel intensity profiles along the local normal. All three steps are described below and with results shown in Figure 5.

1) Extract Contours: The first step of the preprocessing phase is to extract the contours from the STL slice. This step was primarily handled by the C++ OpenCV library. OpenCV has a function that returns both a list of points for each contour as well as a hierarchy of the contours. The hierarchy of contours is important for determining the direction of inside and outside of objects with holes. The left most image in Figure 5 shows the STL slice and the extracted contour.

2) Compute local normal to contours: The second step of the preprocessing stage is compute the local normal. In order to compute the normal of a point A, a line is calculated between two points, B and C, that are three positions on each side of A. From this line a normal vector can be calculated by equations (1) and (2). In the equations the *x* and *y* coordnates of a point are represented in the format of *Point.Coordate.*

$$normal.x = B.y - C.y \quad (1)$$
$$normal.y = C.x - Bx \quad (2)$$

The next step is to compute and store the angle between the normal vector and the X axis, this can easily be done by using the atan2 function with the normal vector The results are visualize by color coding the angle and are shown as the third image of Figure 5.

3) Extract Intensity Profile: The final step of the preprocessing phase is to extract pixel intensity profiles across the normal vector calculated in the previous step. An intensity profile is the set of values that are taken along a line segment. The value of each pixel in the near IR image ranges from 0 (black) to 255 (white). The profile is taken along the normal that was calculated in the previous step and is centered on contour point. A fixed length of 15 pixels was chosen as it sufficiently spanned the entire edge. Finally, the value of each pixel is then converted to between 0 and 1 and stored for the detection phase.

B. Downhill Simplex

The current method of approximating an edge of an object is to fit a sigmoid curve to the half of the profile then selecting the index that the sigmoid crosses a determined threshold. The sigmoid curve used is calculated using two parameters, α and β, and the hyperbolic tangent function as shown in equation (3).

$$sigmoid = tan\,h(\alpha - (\beta * i)) \quad (3)$$

The downhill simplex method, also known as the Nelder-Mead method [2], is the algorithm used to fit the sigmoid to the profile. The downhill simplex method is commonly used to search over multiple dimensions of variables with the goal of minimizing a cost function. For our problem the variables to be optimized are α and β. The cost function shown in equation (4) is the sum of distance between the sigmoid and the profile at each index.

$$cost = \sum_i^8 (sigmoid(i) - profile[i])^2 \quad (4)$$

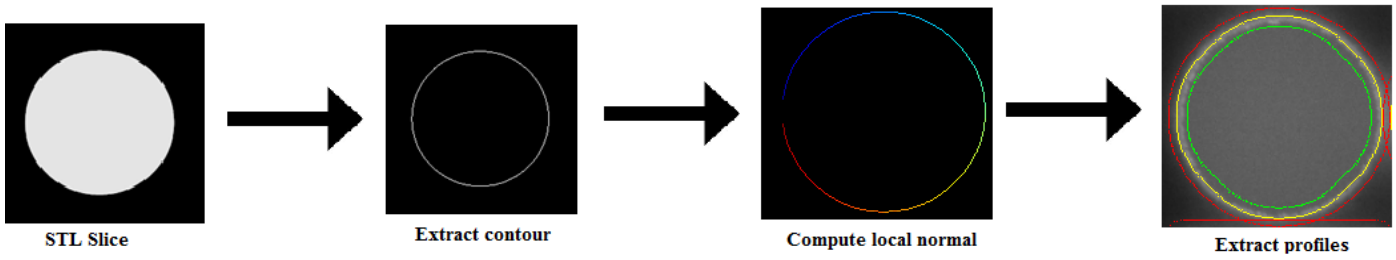

Fig. 5. Preprocessing method with visualization of the intermediate steps results. The extracted profile is shown with the beginning of the profile as green, the middle as yellow and the end as red.

The downhill simplex algorithm works by creating a simplex, which is a shape with $n+1$ vertices in n dimensions, and moving the simplex through the search space until a local minimum is reached. This problem only has two variables so the simplex is a triangle. Each vertex of the triangle represents a sigmoid with α and β corresponding to the x and y coordinates of the vertex.

Each vertex is evaluated according to the cost function and the worst vertex is removed and replaced at a new location. There are three operations of reflection, contraction, and expansion that are used to determine where the new vertex is placed. The details of these operations are discussed in [2] but are beyond the scope of this project.

Once the sigmoid converges the index within the profile of the edge must be found. A threshold value of .15 was chosen by trial and error. Each index of the sigmoid is surveyed and the index closest to the threshold is selected as the edge. The threshold can be adjusted to change the "tightness" of the edge. A visualization of the profile and sigmoid is shown in Figure 6. The algorithm must be repeated for the other half of the profile in order to find both the inside and outside edges.

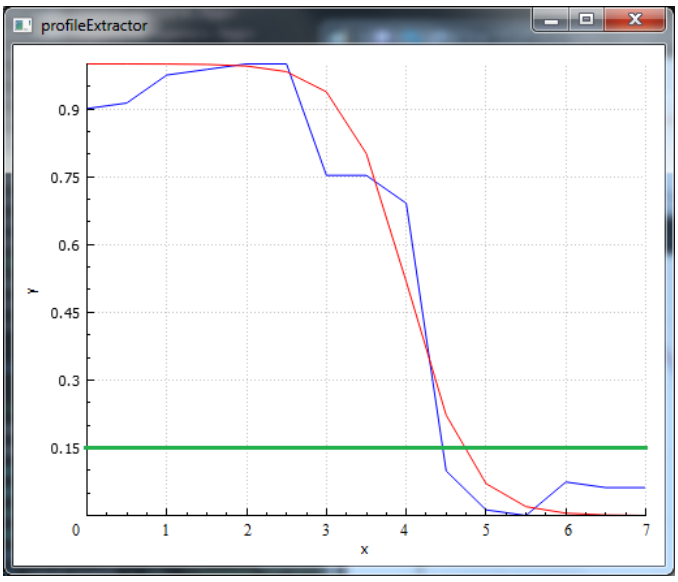

Fig. 6. Sigmoid (red) fitted by the downhill simplex to the profile (blue). The threshold is shown in green. Index 5 was chosen as it is the closest index to the point of intersection between the threshold.

C. Neural Network

The new method of edge detection using an artificial neural network was selected for multiple reasons including speed, performance, and ease of implementation. Neural Networks can easily be parallelized and are able to leverage GPU's for speed. Multiple libraries exist that assist in the creation and deployment of neural networks. As discussed early the library used in this project is Caffe. The accuracy of the neural network is limited by the accuracy of the training data which is generated by the downhill simplex method.

The network architecture is a simple feed forward fully connected network. A feed forward network consists of three types of layers – an input layer, hidden layers, and an output layer. Every neuron in each layer is connected to every neuron in the next layer. Each connection has a weight w_j associated with it. The value of each neuron of each neuron is calculated equation (5) where the value of the neuron in the previous layer is x_j. The activation function is the *tanh* function. After each forward pass the weights are adjusted by gradient descent backpropagation

$$neuron_i = \sum activation\_func(x_j w_j) \quad (5)$$

The architecture chosen is 15 neurons in the input layer, one hidden layer with 50 neurons, and 15 output neurons. The 15 input neurons map to the 15 profile values. Only one hidden layer was chosen due to the relatively small number of inputs and outputs. The 15 output neurons represent the 15 possible indices within the profile where the edge can occur. The index of the output neuron with the maximum value is determined to be the edge.

By default, Caffe only support inputs that are in an image format. A small program was written that accepts a csv (comma separated variable) file as an input and converts the data to a format compatible with Caffe. Each row in the csv file is a profile extracted from the preprocessing phase saved along with the label calculated from the downhill simplex algorithm. Each value is separated by a comma and the label calculated from the downhill simplex algorithm as the last value on the line. Next profiles are randomly separated into training and testing sets. Since many of the profiles are very similar only 20% are selected as part of the training set and the other 80% make up the testing set. After each epoch the testing set is evaluated and the training

process is stopped after performance has not improved for three consecutive times. Because the network is small it converges quickly and terminates after ten epochs, or approximately 15 seconds. Finally, the trained network is saved to be used for processing all the profiles.

D. Edge Analysis Results

Both downhill simplex and neural network methods were tested on a single layer. The layer contained 21,325 extracted profiles. Both versions were tested on a computer with an Intel Xeon ES-1650 v3 CPU and a Nvidia Quardo K2200 GPU. The neural network version was significantly faster than the downhill simplex method to process all of the profiles. The time it took to process an entire layer excluding the preprocessing steps was 21.6 seconds for the downhill simplex method compared to the 0.7 seconds for the neural network method. The neural network version had an approximate speed up of 31x over the current edge detection method. A subsection of the near-IR with the detected edge overlaid is shown in Figure 7. In Figure 7 the downhill simplex results are shown on the left and the neural network results are shown on the right. The downhill simplex results show both inside and outside edges detected, the neural network only displays the inside edge. The outside of the edge can be calculated by reversing the profile and using the same neural network. The edge produced by the neural network is much smoother than the downhill simplex version. One possible reason for this is that the downhill simplex often gets stuck in local minima while the neural network is able to generalize and handle profiles that are irregularly shaped.

Once the edges are detected the further edge analysis was completed. Figure 8 shows a visualization of the distance from the detected edge and the contour from the STL slice. This measurement can be used to analyze geometric accuracy of the printed object.

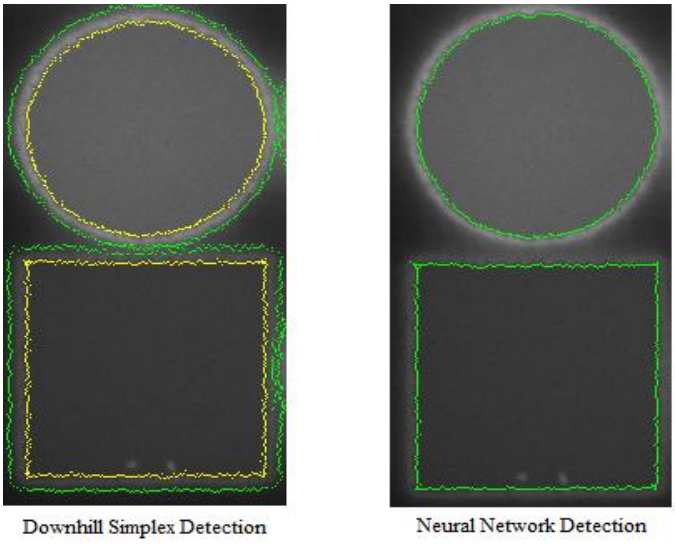

Fig. 7. Edge detection results for the downhill simplex and neural network methods

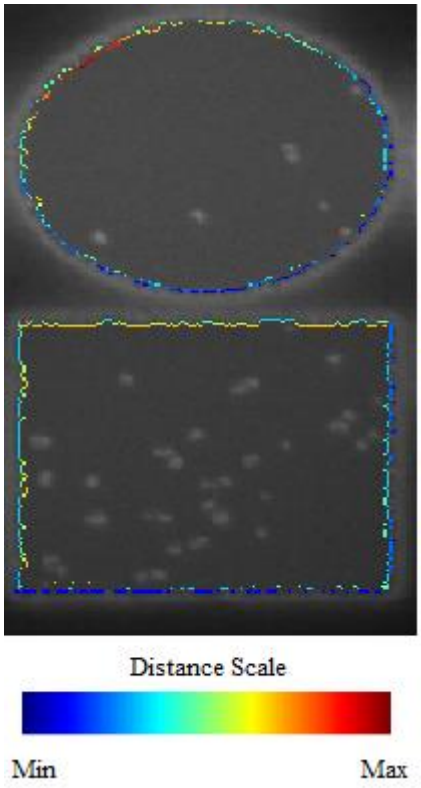

Fig. 8. Visualization of distance between contour from STL slice and the neural network detected inner edge.

IV. Porosity Detection

The second portion of this project is porosity detection. The regular approach for porosity detection is to first find the region of interest using the STL slice as a mask in order to isolate the pixels within the object. The next step is to perform statistical analysis to segment the pores from non-pores.

A. Convolutional Neural Network Method

The new porosity detection method utilizes convolution neural networks. This approach was inspired by the use of neural networks in segmenting membranes by Ciresan et al [3]. The initial step is the same as the regular method, isolate the pixels of interest by using a mask of the STL slice. Next for each pixel *p* within the region of interest the convolutional neural network classifies *p* as either pore or non-pore. The input to the network is the 17x17 window that is centered on *p*. The output is the probably that *p* is a pore. The porosity detection process is shown in Figure 9.

B. Architecture

A convolutional neural network has multiple types of layers that extract features within an image then classifies the features. The three layer types commonly used are convolutional, pooling, and fully connected layers [4]. The convolutional layers consist of kernels that move across a 2D input and generate a 2D activation map. The pooling layers reduce the output from the convolutional layers. The type of pooling layer used in this

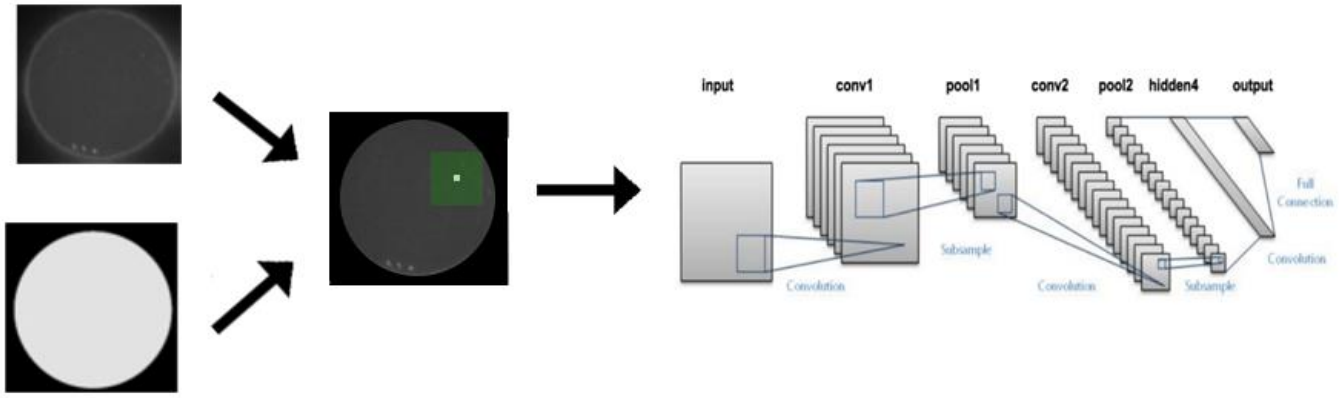

Fig. 9. Porosity detection process using a convolutional neural network for a single pixel. The green square in the second step represents the 17 x 17 window centered around the selected pixel. The window is the input for the convolutional neural network and the probability of porosity is the output. The neural network architecture shown is only used as an example and is not the architecture used in this project.

project is max-pooling with kernel sizes of 2x2. The pooling filters moves across the input and keeps the max value in a 2x2 area and discards the other three values. The final type of layer used is a fully-connected layer. These layers operate the same way as layers in normal feed-forward neural networks. The architecture used is shown in Table I.

Table I. *Convolutional Neural Network Architecture*

Layer	Type	Maps and Neurons	Kernel Size
0	Input	1 x 17 x 17	
1	Convolutional	16 x 17 x 17	4 x 4
2	Max Pool	16 x 9 x 9	2 x 2
3	Convolutional	16 x 6 x 6	4 x 4
4	Max Pool	16 x 3 x 3	2 x 2
5	Convolutional	16 x 2 x 2	2 x 2
6	Fully Connected	100 neurons	1 x 1
7	Output	2 neurons	

C. Training

The results of the regular method are used as groud truth. Training and testing examples are selected by saving the 17x17 window surrounding eligible pixels as an image with its corresponding label in the levelDB format Caffe supports by defualt. The total set of pixels is divided into a training set and a testing set. The examples are split 75% for training and 25% for testing. Over 99% of eligible pixels are classified as pores so the training set needed to be augmented with pore examples. Each positive example is duplicated then rotated either 90, 180, or 270 degrees and added to the training set. Additionally, non-porosity examples are randomly discarded in order to make the training set have approximately 50% of each type. The training phase in Caffe is configured to process the testing set after each epoch and terminate when the testing set has stopped improving five consecutative times. The training phase takes much longer than the edge detection because the neural network is much more complex and there are many more examples in the training set. The total training process takes approximately 50 minutes until convergence.

D. Results

The neural network porosity detection was tested over an entire stack of layers. By trial and error a probability threshold of 95% was selected that determines if a pixel is accepted as a pore or not. This means only pixels with an output higher than .95 from the neural network is considered a pore. Overall the convolutional neural network was able to detect the porosity comparable to the traditional method but had with several issues. The neural network detected pixels around the pores as pores as well and also generated false positives near the edge. Results from a subsection of a layer are shown in Figure 10, the near-IR image is shown on the left and the porosity detected is marked on the right with white.

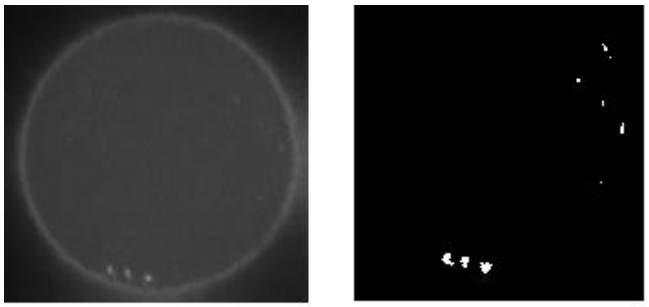

Fig 10. Near-IR image and results from the porosity detection with a 95% threshold.

V. Future Work

The initial results are promising however there are still many improvements that could be made. For the edge detection portion, speed could be improved by processing the profiles from all the layers at the same time compared to an individual layer at a time. A planned improvement for the porosity detection is to use the edge detection to create a more accurate mask. This should allow for better detection of porosity especially near the edges. Both portions of the project could also see considerable accuracy improvements with better ground truth data.

VI. Acknowledgements

I would like to acknowledge and thank Dr. Michael Berry, my advisor at the University of Tennessee. I would also like to acknowledge and thank Dr. Vincent Paquit, my mentor at Oak Ridge National Laboratory (ORNL). My assistantship at ORNL was supported by the Graduate Research Assistantship for Master Students (GRAMS) program through University of Tennessee's Center of Intelligent Systems and Machine Learning (CISML).

References

[1] Y Jia, E Shelhamer, J Donahue, S Karayev, J Long, R Girshick, S Guadarrama, and T Darrell. Caffe: Convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093, 2014

[2] J. A. Nelder and R. Mead, A simplex method for function minimization, Computer Journal 7 (1965), 308–313

[3] Dan Claudiu Ciresan, Alessandro Giusti, Luca Maria Gambardella, and Jurgen Schmidhuber, "Deep neural networks segment neuronal membranes in electron microscopy images," in Neural Information Processing Systems, 2012

[4] D. Scherer, A. Muller, and S. Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In International Conference on Artificial Neural Networks, 2010.