Office of Graduate Studies

Dissertation / Thesis Approval Form

This form is for use by all doctoral and master's students with a dissertation/thesis requirement. Please print clearly as the library will bind a copy of this form with each copy of the dissertation/thesis. All doctoral dissertations must conform to university format requirements, which is the responsibility of the student and supervising professor. Students should obtain a copy of the Thesis Manual located on the library website.

Dissertation/Thesis Title:	Comparison of NCD and CNN based mitotic classification of neural stem cells in phase contrast microscopy
Author:	Michael Marino

This dissertation/thesis is hereby accepted and approved.

Signatures:

Examining Committee

Chair	____________________
Members	____________________
	____________________
	____________________
	____________________
	____________________
Academic Advisor	____________________
Department Head	____________________

Comparison of NCD and CNN based mitotic classification of neural stem cells in phase contrast microscopy

A Thesis

Submitted to the Faculty

of

Drexel University

by

Michael Marino

in partial fulfillment of the

requirements for the degree

of

Masters of Science in Electrical Engineering

September 2016

Acknowledgments

I would like to thank first and foremost Dr. Andrew Cohen for advising me through this process and offering his patience, support, and unwavering optimism from day one, as well as all of my colleagues at the computational biology lab for their support and insight - particularly Mark Winter and Eric Wait for their steady flow of ideas and explanations as well as Angeline Aguinaldo for supplying some very useful code in developing the LEVER interface. I would also like to thank Dr. James Shackleford for generously giving me access to his resources in my exploration of deep learning and always fueling my curiosity with his own. With the heavy lifting on the deep learning side I also owe a huge thanks to Dr. Matthew Stamm and Belhassen Bayar for freely sharing their insight into convolutional neural networks whenever and wherever I happened to chase them down. And lastly I would like to thank my committee, particularly the one not mentioned as of yet, Dr. Timothy Kurzweg, who was, in fact, the first professor at Drexel that I had the privilege of hearing a lecture from and has been steadfastly generous with his time and knowledge ever since.

Table of Contents

List of Tables

List of Figures

Abstract

Comparison of NCD and CNN based mitotic classification of neural stem cells in phase contrast microscopy

Michael Marino
Andrew R. Cohen, Ph.D.

Automated mitosis detection is a major difficulty in segmentation and tracking algorithms. This thesis explores the implementation of an automated mitotic detection algorithm for phase-contrast data into a segmentation and tracking platform called LEVER^2. We implement two separate classification algorithms - one based on an estimate of the pairwise normalized compression distance (NCD) and one deep learning implementation using convolutional neural networks (CNN) - in combination with local statistics in order to limit the search region for mitotic events. The CNN provided a significantly higher detection rate while the NCD provided a significantly lower false positive rate. The NCD classifier generated an overall detection rate of 0.736 and an overall false positive rate of 0.0083, while the CNN approach generated an overall detection rate of 0.880 and a false positive rate of 0.0567. However, it was also determined that using a bi-modal classifier incorporating the peak normalized cross-correlation of the image with the $t-1$ frame the CNN implementation could outperform the NCD classifier - achieving a false alarm rate of 0.0079 with a detection rate of 0.809 by thresholding out cell images that were above a threshold peak cross-correlation.

Chapter 1: Introduction

Segmentation and tracking algorithms for microbiology data have become one of the most prominent applications of computer vision and image processing to date. The ability to robustly evaluate the behavior and form of cells in a culture using automated algorithms has the potential to save biologists countless man-hours of manual data-processing. One of the most difficult problems in accurate tracking today is the problem of accurate mitosis detection. Without accurate mitosis detection, modeling track division relies entirely on the accuracy of the segmentation from frame to frame. This paper explores two different methods for mitotic detection - one using compression based similarity in the form of a wavelet-based estimation of the normalized compression distance (NCD)[3] and another based on deep learning - specifically convolutional neural networks[4].

1.1 Background & Theory

1.1.1 LEVER

LEVER is a tool for lineage editing and validation in neural stem cells (NSCs)[5]. There is a ubiquitous demand in the field of microbiology for more sophisticated tools to evaluate time-lapse microscopy data. LEVER is designed to quantify the lineage and patterns of the behavior of neural stem cell development. The software package contains two integarted modules - one that performs automated segmentation and tracking and another that provides and interactive software application that allows a user to inspect and edit the automated results provided by the first module[2]. The interactive software package allows a user to provide feedback regarding the accuracy of cell splits which are then utilized to re-learn the behavior of the cell lineages to a significantly more accurate level. The segmentation and tracking uses the algorithm in[6 7] with a resegmentation step that utilize the user-provided feedback as described in[8]. Steps to download and install LEVER can be found in[2].

1.1.2 Computability, Kolmogorov Complexity, and Distance Measures

The classification algorithm used to classify each hull as mitotic or non-mitotic utilizes concepts in Algorithmic Information Theory developed in this section. The central concepts of computability and distance, specifically information distance adapted to a compression distance, in combination with the concept of Kolmogorov Complexity, are the theoretical basis for the classifier.

The notion of computability has been an ongoing topic of conversation for at least decades. Let us first define a Turing Machine as a machine consisting of a finite program called the *finite control* which is capable of manipulating a linear list of *cells* (generally visualized as a one-dimensional tape consiting of horizontally conjoined sqaures), and one access pointer referred to as the *head*. The machine is capable of only the most basic functions. It is capable of reading the contents of the square it is currently pointing to, overwriting that square, or moving left or right[3]. Considering the symbol on each square of tape to be either a 0 or a 1 or a blank, this theoretical machine can be considered the most simple form of what we would consider a computational device. The device is constructed as to behave according to a list of rules based on the state of the machine and the current input. Thus, the machine is a mapping from a set of states and inputs to a set of states and outputs. The significance of such a machine is that it is arguably the most basic abstraction possible that can still perform any computational task that a modern computer is capable of, albeit at a significantly lower speed. It is the idea of this most basic computational device that lies at the heart of the notion of Kolmogorov Complexity. The Kolmogorov Complexity of a string is defined as the shortest description length program that, given as an input to a Universal Turing Machine, would produce that string as an output[3].

Information Distance & Normalized Information Distance

The notion of Kolmogorov Complexity gives an objective evaluation of information content of a particular string. A natural progression in this thinking is to derive a related distance measure that can measure the distance between two strings with respect to their information content, or, in other words, a similarity measure of the strings based on their information content[3]. One universal information distance between two strings is the amount of information necessary to transform from one to the other, or, stated another way, given a string x, what amount of information is required to generate a string y, or vice-versa[3]. Similar to the definition of Kolmogorov Complexity, the minimal information distance between x and y could be considered as the length of the shortest program for a universal Turing machine to transform x into y or transform y into x. This measure is equal to the maximum of the conditional Kolmogorov Complexity, $K(y|x)$ up to an additive logarithmic term[3]. This value in and of itself is not suitable as an information distance because of particular cases such as the empty string, ϵ, where $K(\epsilon|x)$ is small regardless of x. So a more suitable choice for the information distance from x to y is the value $K(y|x) + K(x|y)$[3]. Alternatively, we could avoid such a problem by defining the maximum information distance as the value[3]

$$E_0(x, y) = max\{K(x|y), K(y|x)\} \tag{1.1}$$

Further, another issue with the information distance defined by (1.1) is that it is a function of the length of the strings involved. Thus, if we are more interested in a relative distance rather than an absolute one we define the *normalized information distance* (NID) to be[3]

$$e(x, y) = \frac{max\{K(x|y), K(y|x)\}}{max\{K(x), K(y)\}} \tag{1.2}$$

Normalized Compression Distance

Since Kolmogorov Complexity is theoretically uncomputable[3], in practical applications an approximation of the NID must be used. On such approximation utilizes a real-world compressor to replace the notion of Kolmogorov complexity. Defining xy as the concatenation of the string x with the string y and denoting the compressed size of the string x as $Z(x)$ we define the *normalized compression distance* as[3]

$$e_z(x, y) = \frac{Z(xy) - min\{Z(x), Z(y)\}}{max\{Z(x), Z(y)\}} \tag{1.3}$$

1.1.3 The Discrete Wavelet Transform & Image Compression

Image compression algorithms are a pressing field of study for researchers today. An overview of such a topic is beyond the scope of this thesis, however, the particular compression algorithm used for the implementation will be discussed. A wavelet is a signal which can be visualized as an oscillatory pulse of limited duration that integrates to zero[9]. A general form for 2D transformations of an image $f(x, y)$ can be written in terms of the transform domain variables u and v as[10]

$$T(u, v) = \sum_{x,y} f(x, y) g_{u,v}(x, y) \tag{1.4}$$

It is significant to note that in the case that the kernel function $h_{u,v}(x, y)$ is separable such that

$$h_{u,v}(x, y) = h_u(x) h_v(y) \tag{1.5}$$

the computation can be implemented for the 2-D transform by performing row-column or column-row passes of the 1-D transform[10].

The concept of wavelets is designed to overcome the zero-time resolution of the Fourier transform defined as

$$\mathfrak{F}\{\omega\} = \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt \tag{1.6}$$

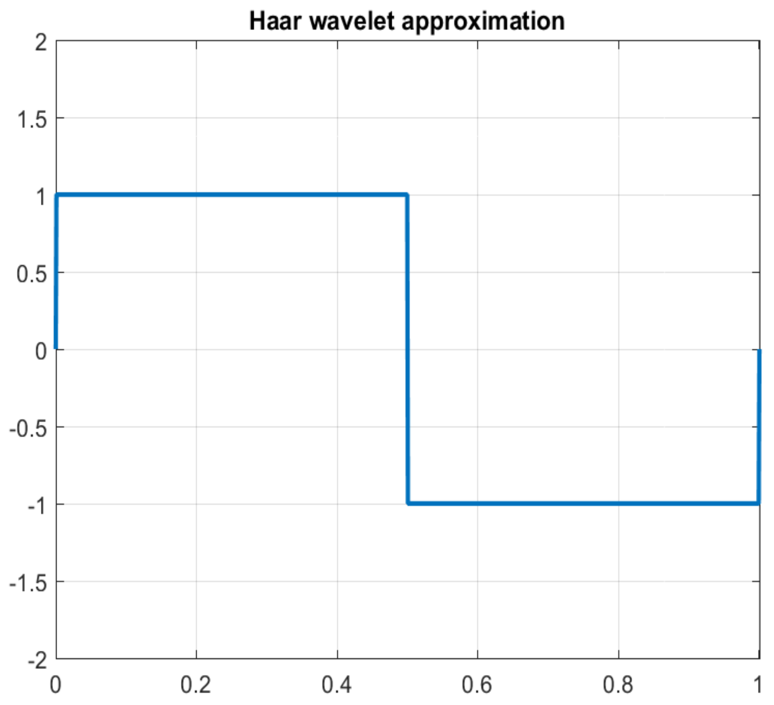
where j is the $\sqrt{-1}$, $f(t)$ is the signal and ω is the frequency variable in rad/s. In this equation the time variable, t, is integrated out which gives full frequency resolution and zero time resolution, implying that frequency information for the overall signal is known with full resolution while no information regarding the location in time where the frequency exists is left. For a signal with constant frequency this is perfect. However, if the frequency composition varies in time (or in space in the case of an image) there is significant information lost. Wavelets are one way to remedy this. In continuous time, the continuous wavelet transform (CWT) is defined as[11]

$$\gamma(s, \tau) = \int f(t) \psi^*_{s,\tau}(t) dt \tag{1.7}$$

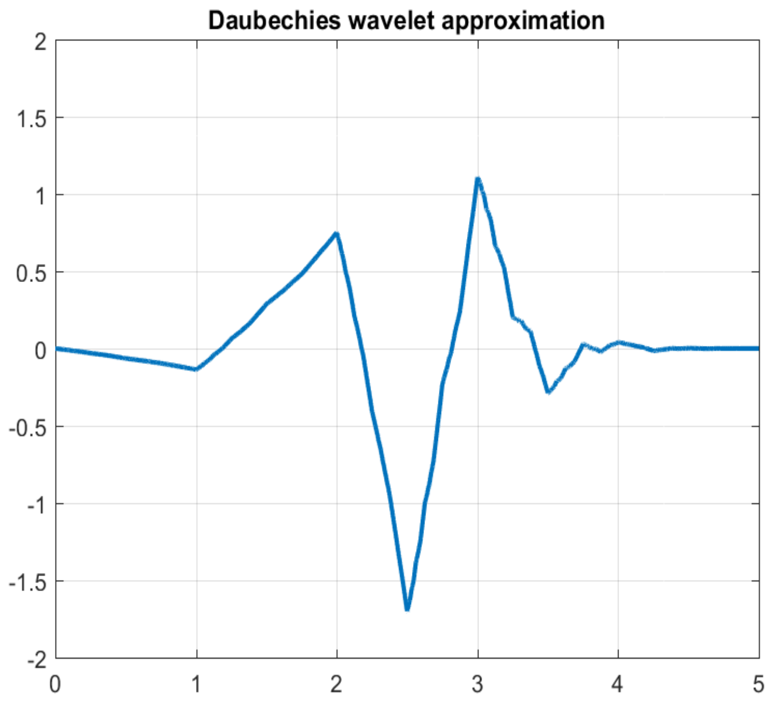
where $\psi(t)$ acts as the basis function for the signal decomposition and is referred to as the mother wavelet, * refers to complex conjugation, s is referred to as the scale factor (as what is being evaluated is not a pure frequency as in the case of the Fourier transform), and τ is the translation factor which act on the mother wavelet according to[11]

$$\psi_{s,\tau}(t) = \frac{1}{\sqrt{s}}\psi\left(\frac{t-\tau}{s}\right) \tag{1.8}$$

thus, as the names imply, scaling and shifting the original mother wavelet. Some examples of mother wavelet function approximations can be seen below in figure 1.1.

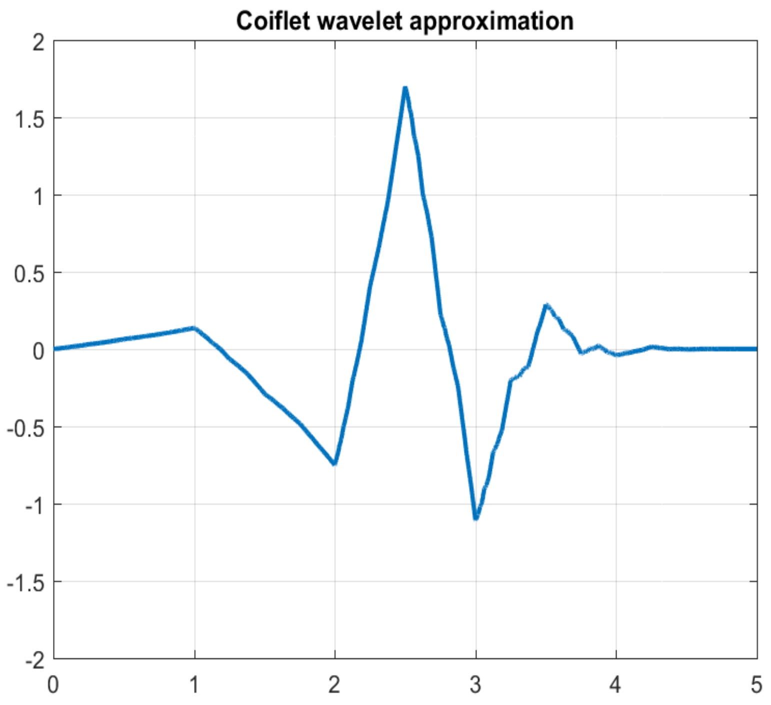


(a) Haar Wavelet

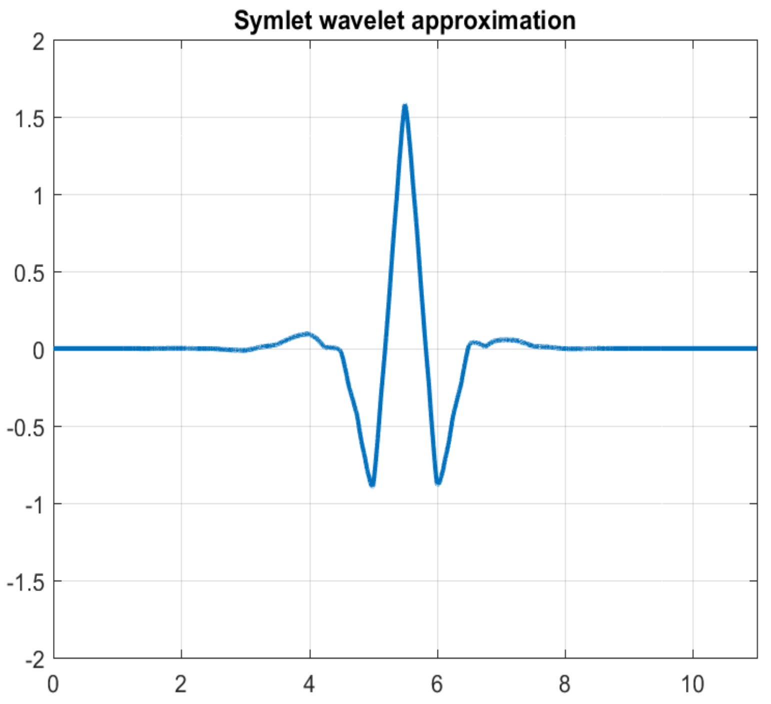


(b) Daubechie Wavelet

Figure 1.1: Examples of common mother wavelet function approximations used in computing the discrete wavelet transform (DWT)

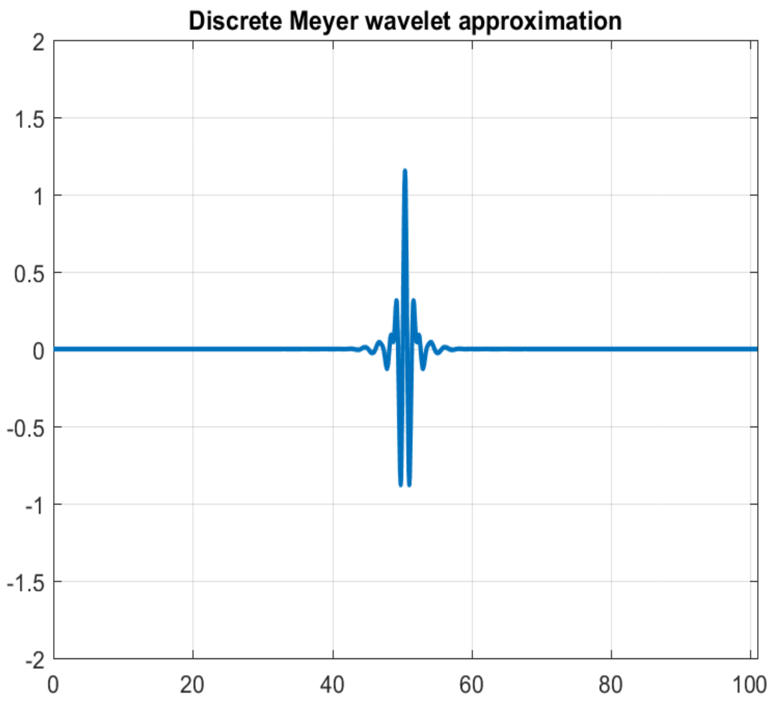


(c) example1C



(d) example1C

Figure 1.1: (cont'd) Examples of common mother wavelet function approximations used in computing the discrete wavelet transform (DWT)



(e) Wavelet Examples

Figure 1.1: (cont'd) Examples of common mother wavelet function approximations used in computing the discrete wavelet transform (DWT)

The properties that a mother wavelet function must exhibit are specified as[11]

$$\int \frac{|\Psi(\omega)|^2}{|\omega|} d\omega < \infty \tag{1.9}$$

where $\Psi(\omega)$ refers to the Fourier transform of $\psi(t)$ and (1.9) is referred to as the *admissibility condition*, and also implies that

$$|\Psi(\omega)^2|\Big|_{\omega=0} = 0 \tag{1.10}$$

and that

$$\int \psi(t)dt = 0 \tag{1.11}$$

or the zero-mean condition[11]. For the discrete case, the wavelet basis functions are defined according to

$$\psi_{j,k}(t) = \frac{1}{\sqrt{s_0^j}} \left(\frac{t - k\tau_0 s_0^j}{s_0^j} \right) \tag{1.12}$$

where j and k are integers, s_0 is the dilation step, and τ_0 is the discrete translation factor[11].

The result of wavelet decomposition is a series of wavelet coefficients that represent the amount of energy contained in a particular basis function in a particular temporal or spatial area. Thus, wavelets can be used as low-pass, high-pass, or band-pass filters by only using a subset of coefficients. Typical implementations of wavelet approximations consist of performing high-pass and low-pass analysis of the image in the horizontal, vertical, and diagonal directions separately. Further, wavelet analysis has proven valuable in image compression as it has been shown that relatively few large coefficients will capture the majority of the information contained in the image, so that most coefficients can be set to zero and the image representation in the wavelet domain can be implemented using sparse matrices[12]. The sparsity of the wavelet coefficients is thus a measure of the compressibility of the image. An efficient algorithm for image compression using the wavelet coefficients was also proposed in[9].

1.1.4 Spectral Clustering

Spectral clustering as defined in[13] is a method of using the eigenvalues and eigenvectors of a distance matrix in order to cluster data that would not lend itself well to clustering in its raw form. The generalized algorithm can be summarized in the steps -

1. Generate an affinity matrix $A \in \mathbb{R}^{nxn}$ based on some distance or similarity metric
2. Generate a diagonal matrix $D \in \mathbb{R}^{nxn}$ whose $(i,i)^{th}$ entry is the row sum of the matrix A.
3. Define $L = D^{-1/2}AD^{-1/2}$
4. Perform an eigendecomposition on L and find $\{x_1, x_2, ...x_k\}$ the k orthogonal eigenvectors corresponding the the k distinct largest eigenvalues, where k is the number of clusters the data is to be divided into and form the matrix $X = [x_1x_2...x_k] \in \mathbb{R}^{nxk}$
5. Perform clustering on the resulting X matrix treating each row in X as a point in $\mathbb{R}^k$

6. Assign raw data point i to the cluster j which row i in X was assigned to

In summary, the algorithm performs a spectral decomposition of a distance matrix and uses clustered row vectors of a matrix composed of the k principal eigenvectors in order to cluster the raw data.

1.1.5 Convolutional Neural Networks

Traditional neural networks represent a family of well-known algorithms used in machine learning that extend the field of linear models for regression and classification of the form[14]

$$y(\mathbf{x}, \mathbf{y}) = f\left(\sum_{j=1}^{M} w_j \phi_j(x)\right) \tag{1.13}$$

where $f(\cdot)$ is, in the case of classification, a nonlinear activation function, or is the identity function in the case of regression, w_j represents the j^{th} weight component and $\phi_j(x)$ represents the j^{th} basis function in the model. Neural networks extend this paradigm to a network where the $\phi_j(x)$ depend on a set of parameters and these parameters as well as the weights, w_j, are adjusted according to an optimization function during training[14]. This type of network can be composed of several hidden layers each with separate parameters and weights. Figure 1.2 below shows a diagram of a basic neural network framework.

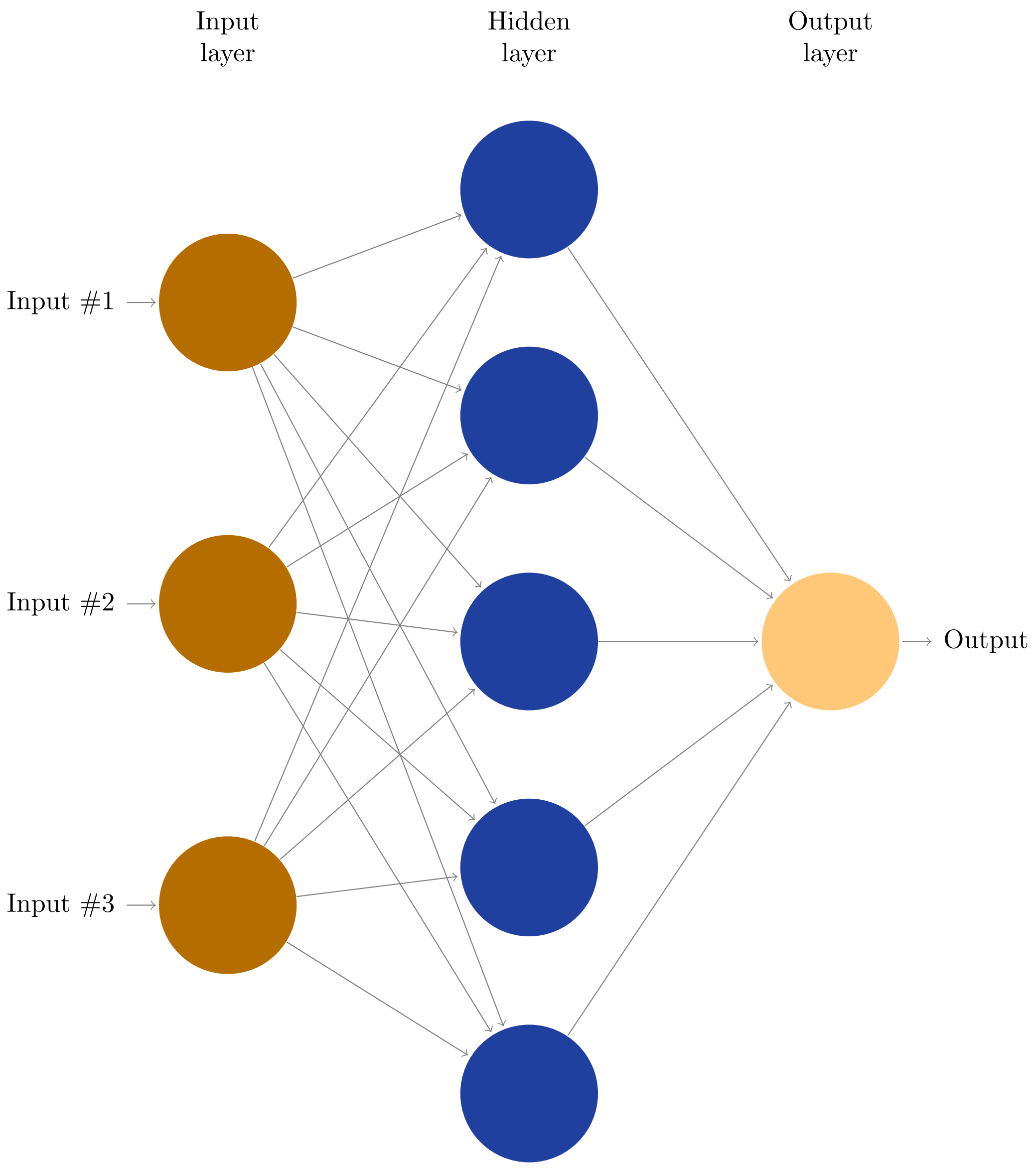


Figure 1.2: Example of basic neural network architecture[1]. This network shows three inputs followed by one hidden layer composed of five neurons and then one output

In the framework in Figure 1.2 there are three input nodes followed by one hidden layer composed of four nodes followed by a single output node. This model performs well in some applications (and is also a component of many typical convolutional neural network architectures as will be discussed later) but is of limited value in computer vision problems due to its treatment of all inputs as independent entities and thus eliminating all spatial information.

In order to exploit spatial relationships between pixels in computer vision tasks, rather than simply learning weights to be multiplied by each input pixel, a convolutional layer is designed to learn kernels which are then convolved with the input images resulting in feature maps. A typical convolutional network is composed of convolutional layers as well as multiple additional layers which can be generalized as -

- convolutional layer
- non-linear activation layer
- sub-sampling layer
- cross-channel normalization layer
- fully-connected layer

Convolutional Layer

A convolutional layer is composed of sets of units organized into planes which are referred to as *feature maps*[14]. Each feature map is limited to having the same weights for each sub-region. For instance, a layer may consist of 128 26x26 feature maps with a 5x5 kernel of weights associated with each of the 128 channels. Training this layer would constitute learning 128x5x5 = 3200 weights. The feature maps from each layer are generally then subject to some sort of activation layer as well as either a sub-sampling layer, cross-channel normalization, or both, before being fed into another convolutional layer or a fully connected layer.

Non-linear Activation Layer

The non-linear activation layer parallels the function $f(\cdot)$ in equation (1.13). Activation functions are used in traditional neural networks to model the output of a layer in an approximately discrete fashion, which is the ultimate goal of classification. Three typical activation functions are the functions

$$f(x) = tanh(x) \tag{1.14}$$

and

$$f(x) = (1 + e^{-x})^{-1} \tag{1.15}$$

which is more generally known as the logistic sigmoid function and will be discussed further with the softmax layer. With respect to the first derivative, both of these functions approximate the unit step function defined as

$$f(x) = \begin{cases} 0 & : x < 0 \\ 1 & : x \geq 0 \end{cases} \tag{1.16}$$

the derivative of which is zero everywhere with the exception of at $x = 0$ where it is infinite. This type of activation function lends itself well to classification tasks. However, the downside is that neuron saturation leads to longer training times. Since the derivative goes to zero as the input moves away from zero, and training algorithms are based on the first derivative, or similarly the gradient in the multi-dimensional case, if a neuron's output moves away from zero it will no longer have a significant impact on the update function. This phenomenon is referred to as the neuron saturating. Alternatively, a non-saturating activation function called Rectified Linear Units (ReLUs) as defined in[15] as

$$f(x) = \begin{cases} 0 & : x < 0 \\ x & : x \geq 0 \end{cases} \tag{1.17}$$

has been shown to decrease learning time significantly in some applications[16].

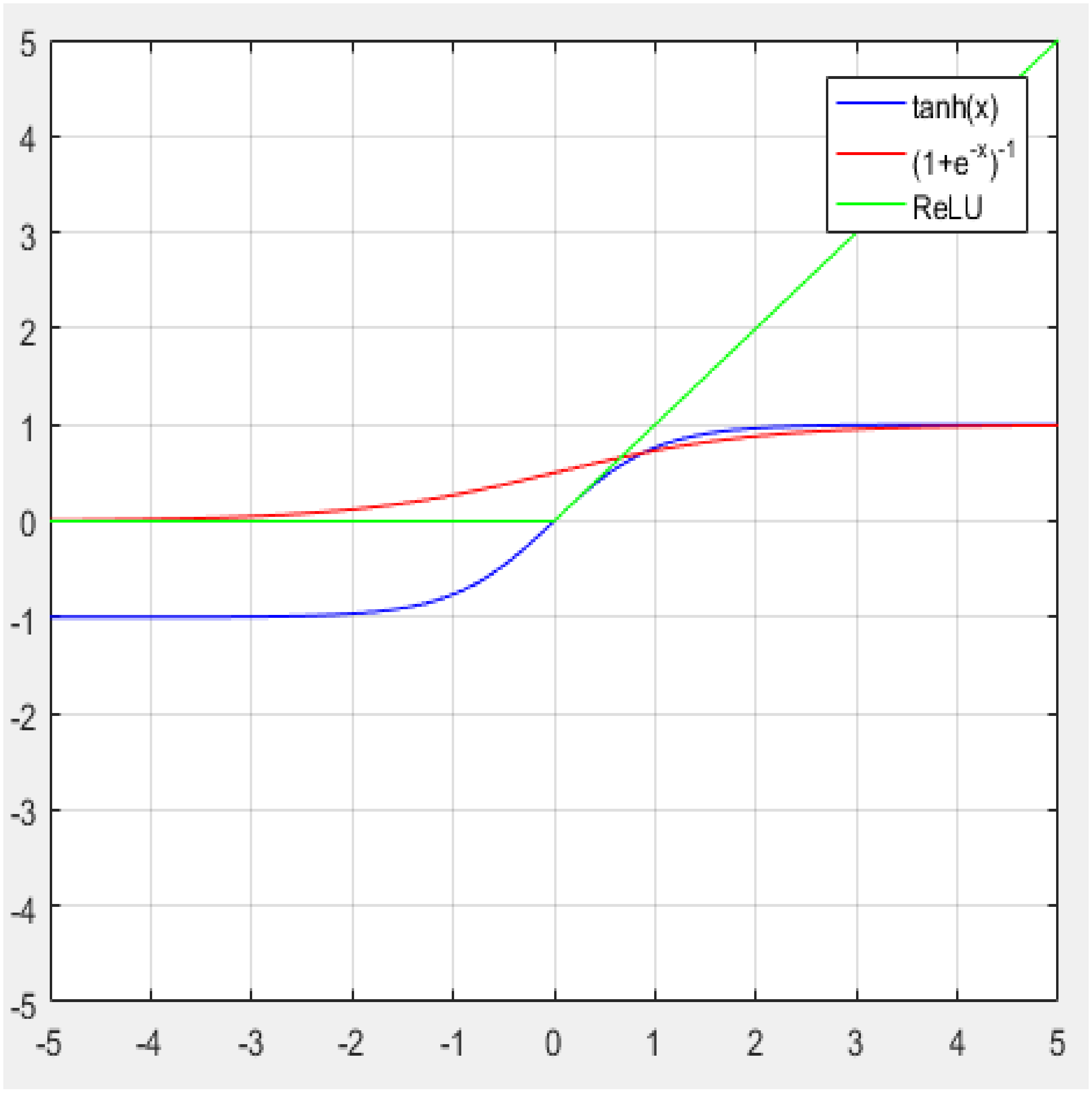


Figure 1.3: Plot of activation functions in equations (1.14), (1.15), and (1.17). Note how both $tanh(x)$ and $(1+e^{-x})^{-1}$ flatten as they move away from $x = 0$ while the ReLU function has a constant slope of one from $x > 0$ to $x = \infty$. The fact that the derivative does not go to zero as x moves away from zero is what decreases training time by preventing saturation.

Sub-sampling Layer

The subsampling layer effectively decreases the effect of noise on the output. Convolutional neural networks are designed to take as input images with minimal pre-processing and so noisy images can present a problem in training. Additionally, sub-sampling decreases the number of pixels composing each image. As the end result of the convolutional layers is intended to be input into a fully connected layer, the smaller size is advantageous at this point. Common sub-sampling layers include max-pooling, min-pooling, and average-pooling. At this layer, a window-size is chosen and, depending on the type of pooling, either the maximum value, minimum value, or average value from that window will be taken as the output.

Cross-channel Normalization

Cross-channel normalization, also referred to as local response normalization, is a means of introducing a lateral inhibition between adjacent channels which creates competition for large activities amongst neighboring kernels[16]. The output of the cross-channel normalization layer, $b^i_{x,y}$, is defined according to the equation[16]

$$b^i_{x,y} = \frac{a^i_{x,y}}{\left(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} (a^j_{x,y})^2\right)^{\beta}} \tag{1.18}$$

where $a^i_{x,y}$ is the non-normalized activity at position (x, y) in feature map i, and $k, n, \alpha, and \beta$ are parameters that can be manually determined or potentially optimized using a validation set. The values used in[16] were $k = 2, n = 5, \alpha = 10^{-4}$, and $\beta = 0.75$. These values were used in the present implementation with the exception of n which MATLAB does not allow for deviation from. Incorporating the cross-channel normalization layer generated an error-reduction of 1.7% in the validation set.

Fully connected layer & softmax

The final layer before classification, the fully connected layer, is a traditional neural network as seen in figure 1.2 treating all pixel values from the previous layer as individual inputs. This layer is then followed by a softmax layer, the purpose of which is to generate a probabilistic model for the output.

The softmax is a generalization of the logistic regression model which defines the posterior probability of a class C_1 according to[14]

$$p(C_1|\boldsymbol{\phi}) = y(\boldsymbol{\phi}) = \sigma(\mathbf{w}^T\boldsymbol{\phi}) \tag{1.19}$$

where $\sigma(\cdot)$ is the sigmoid function defined in (1.15). This probabilistic model has the advantage over the Gaussian model of requiring only M adjustable parameters for an M dimensional space rather than $2M$ parameters representing the mean values and $M(M+1)/2$ parameters representing the covariance matrix. Thus, for many-dimensional models such as neural networks it is preferred[14]. Since the support of the sigmoid function is between zero and one it can be interpreted as a probability within the context of a two class classification problem. However, convolutional neural networks are generalized to mutli-class problems, and so the logistic regression model is not sufficient to generate a probabilistic model for the output classification. Thus, this model was generalized to the normalized exponential, also known as the softmax function, defined as

$$p(C_k|\mathbf{x}) = \frac{exp(\mathbf{x}^T\boldsymbol{\phi_k})}{\sum_j exp(\mathbf{x}^T\boldsymbol{\phi_j})} \tag{1.20}$$

where $p(C_k|\mathbf{x})$ is the posterior probability for class k, $\mathbf{x}$ is the observation vector of the datapoint being classified, and $\boldsymbol{\phi_k}$ is the weight vector for the neuron in the final fully connected layer that is designated to class k. Thus, the final classification is based on a probabilistic model defined using the softmax equation in (1.20). The default behavior of a network is to set the output to the class with the maximum probability. However, alternatively the model could be re-defined to any threshold based on the effective cost of misclassification vs. false alarm by, for instance, using the ROC in the two-class case.

Stochastic Gradient Descent and Backpropogation

The learning parameters $\mathbf{w}$ are updated according to stochastic gradient decent according to the rule[14]

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \Delta E_n \tag{1.21}$$

where $\mathbf{w}^{(\tau)}$ denotes the weight parameters after iteration τ, η is the learning rate parameter which determines the rate of descent, and ΔE_n denotes the gradient of the weight parameters with respect to the parameters for the n^{th} mini-batch of the dataset[14]. *Stochastic gradient descent* differs from standard gradient descent, which is defined similarly to (1.21) but where the entire dataset is considered in each update, thus ensuring a global optimal solution, whereas stochastic gradient descent uses subsets of the data for each update. This approach is used when the datasets are sufficiently large to make training with standard gradient descent unattractive in terms of computation time.

In a network consisting of several to many layers with parameter counts on the order of 10^3 to 10^6, calculating ΔE_n defined here to be the gradient with respect to these parameters is a computationally intensive task even for a small subset of the data. These update computations are defined using the chain rule from calculus which states that, given a function f that is itself a function of of two other functions, u and v, parameterized by t,

$$f\big(u(t), v(t)\big) = f(u, v) \tag{1.22}$$

then

$$\frac{df}{dt} = \frac{\partial f}{\partial u}\frac{du}{dt} + \frac{\partial f}{\partial v}\frac{dv}{dt} \tag{1.23}$$

This basic relationship can be extended to any number of input variables. The multi-input generalization will yield a gradient composed of partial derivatives with respect to each input parameter, rather than a derivative. Defining $\mathbf{x_n}$ as the output to the module or layer defined by the weights $\mathbf{w_n}$, and if the partial derivative E^p with respect to the outputs $\mathbf{x_n}$ are known, then the partal derivatives with respect to $\mathbf{w_n}$ and $\mathbf{x_{n-1}}$ can be calculated according to the recurrence relation[17]

$$\frac{\partial \mathbf{E^p}}{\partial \mathbf{w_n}} = \frac{\partial \mathbf{J}}{\partial \mathbf{w}}(\mathbf{w_n}, \mathbf{x_{n-1}})\frac{\partial \mathbf{E^p}}{\partial \mathbf{x_n}} \tag{1.24}$$

$$\frac{\partial \mathbf{E^p}}{\partial \mathbf{x_{n-1}}} = \frac{\partial \mathbf{J}}{\partial \mathbf{x}}(\mathbf{w_n}, \mathbf{x_{n-1}})\frac{\partial \mathbf{E^p}}{\partial \mathbf{x_n}} \tag{1.25}$$

where $\mathbf{J}$ denotes the Jacobian matrix whose ij^{th} element $\mathbf{J}_{ij}$ is defined with respect to the j^{th} output y_j and the i^{th} input x_i as[14]

$$\mathbf{J}_{ij} = \frac{\partial y_j}{\partial x_i} \tag{1.26}$$

Using the relationships defined in (1.24) and (1.25), the gradient of the weights $\mathbf{w}$ with respect to the inputs $\mathbf{x_0}$ can be computed recursively beginning from layer N down to layer 1, which is the technique referred to as backpropogation[17].

Chapter 2: Related Work

Many works have attempted to solve the problem of mitosis detection since the advent of segmenting and tracking computational algorithms. In[18] Huh *et al.* propose a three-step approach in which candidate patch sequences are first identified, followed by feature extraction and finally temporal localization of birth events by estimating a probabilistic model based on the features. The approach for determining candidate patches consists of locating pixel areas exhibiting above-average brightness which could be considered similar to our approach of using the ReLu L1 of the difference image in order to filter out false positives. In our approach, however, it was found that the 2D normalized auto-correlation performed more effectively and was less correlated with the latter steps taken in detection. In[19], Padfield *et al.* locate mitotic events by linking non-mitotic images through a distance metric and the fast marching method in order to connect parent cells with children cells to find mitotic events. This approach relies on effective tracking as a prerequisite for effective results, an assumption not made in the current work. In[20], Yang *et al.* utilize a brightness threshold to extract ROIs, followed by smoothing the ROI region and computing local properties and then searching for relative changes in the local properties in order to classify mitosis. In[21], Liu *et al.* also use relative bright regions in order to identify mitotic candidates, followed by a *hidden conditional random field* approach which is similar to hidden Markov models but without the assumption of independent observations given the values of hidden variables[21]. The only example of utilizing deep learning for mitosis detection was found in[22] where Wang *et al.* use CNNs to classify mitosis in breast cancer histology images. However, their detection rate was relatively limited at 0.65.

The basis for the theory of the NCD classifier was developed in[3]. The efficacy of clustering by compression was demonstrated in[23] in which Cilibrasi *et al.* tested compression-based similarity in various applications including genomics, literature, music, and character recogniton. This work was extended in[24] in which Cohen *et al.* utilized the NCD of multisets for retinal progenitor cell fate prediction as well as in successfully labeling a mutant huntington protein population of mice that the pairwise NCD was unable to effectively label. The pairwise NCD was also utilized by Joshi *et al.* in[25] in order to successfully separate mitotic from non-mitotic frames in phase contrast microscopy date. This algorithm is the NCD pairwise implementation used in the present work. The CNN architecture was designed using the concepts developed in[16] with theoretical foundations also found in[14][4][17].

Chapter 3: Contributions

- Implemented NCD and CNN based mitotic detection algorithms for phase contrast microscopy neural stem cell image sequences
- Tested classification schemes on dataset including 519,097 segmentations with 1,549 mitotic events for an overall detection rate of 0.88 and an overall false positive rate of 0.0567 for the CNN and an overall detection rate of 0.736 and false positive rate of 0.0083 for the NCD
- Evaluated a frame-to-frame increase in brightness as well as a peak normalized autocorrelation as filtering mechanisms for the classifier, finding that the normalized autocorrelation works well to filter false positives in the CNN case but has little effect in the NCD case
- Incorporated NCD classifier into LEVER platform in order to test impact on lineaging algorithm, generating six lineage trees for an overall recall rate of 0.394 and an overall precision rate of 0.500

Chapter 4: Implementations

4.1 CNN Architecture

Many slightly varying CNN architectures were tested ranging from one convolutional layer to four, with the best overall error rates on the validation testing set used to classify the entire data set. The optimal architecture can be seen in figure 4.1 below.

The architecture consists of three convolutional layers - the first layer is a 5x5 convolutional window with 4 channels, the second is a 3x3 window with 8 channels, and the final is a 3x3 window with 12 channels, with a 2x2 max-pooling layer after the second two convolutions. These layers are then followed by two fully connected layers - one hidden layer consisting of 256 neurons and one output connected layer consisting of two neurons, which are then fed into the softmax classification function. Each layer with the exception of the final output layer is also followed by ReLu activation layer. Without the ReLu layer the testing accuracy did not reach above 55%.

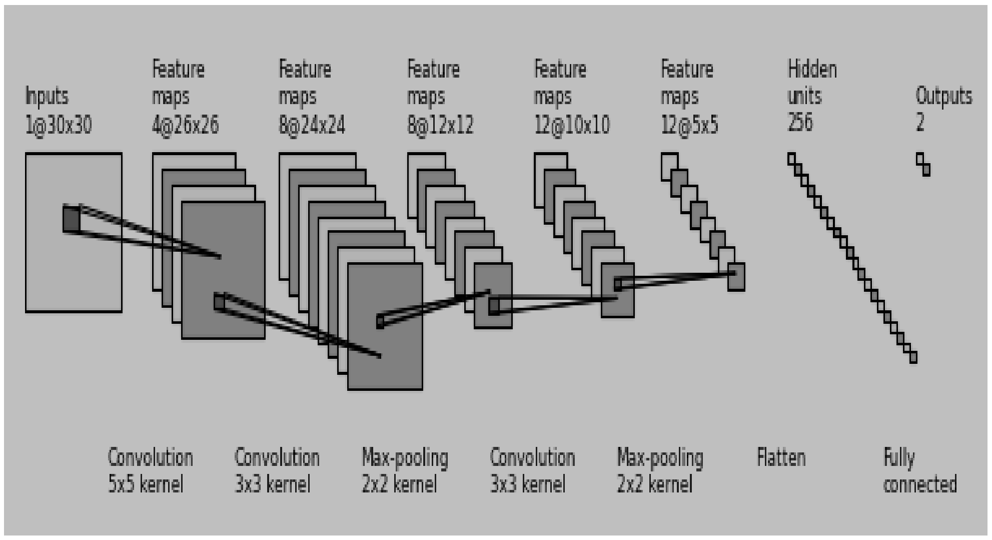
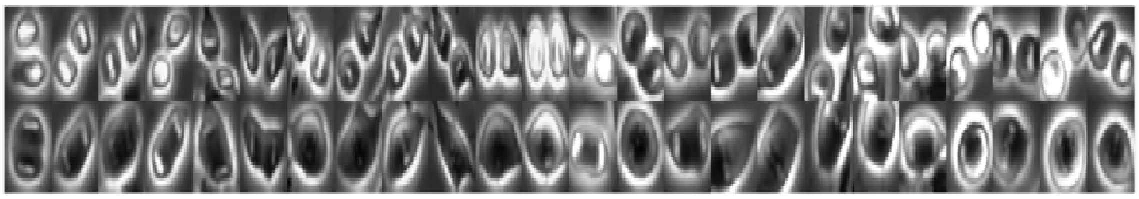


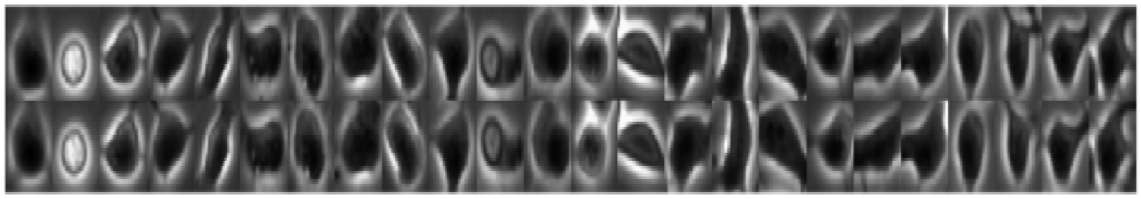
Figure 4.1: CNN architecture chosen as optimal model - The input at the left is a grayscale image, which is input to a four-channel convolutional layer with a 5x5 kernel size, followed by an eight-channel convolutional layer with a 3x3 kernel size, followed by a max-pooling layer with a 2x2 kernel and a stride of two which effectively decreases the area of each feature map by a factor of four, followed by a twelve-channel convolutional layer with a 3x3 kernel size, and another max-pooling layer identical to the previous one. The output of the final max pooling will be twelve 5x5 feature maps which are then flattened and passed into a fully-connected layer with 256 neurons, and finally the last fully-connected layer consisting of two nerons - one for each class. The output of these last two neurons are fed into the softmax classification layer.

4.2 NCD Implementation

The NCD implementation was based on that in[25] which uses the L1 norm of the discrete wavelet approximation of a difference image as the approximate compressed size of the image. The algorithm is composed of three steps. First, the a 30x30 image consisting of the hull to be classified is extracted from the larger image, as well as the same spatial region extracted for the $t + 1$ time frame. The low-pass discrete wavelet approximation based on the symlet shown in figure 1.1d of both images are then taken, and the resulting t cell approximation is subtacted from that in the $t + 1$ frame. The "training set" used in this case consisted of 24 mitotic cases and 24 non-mitotic cases chosen for the distinctive patterns exhibited in the sets. The normalized training sets can be seen below in figure 4.2 as well as the normalized low-pass DWT approximations and their difference images (bottom row) in figure 4.3.

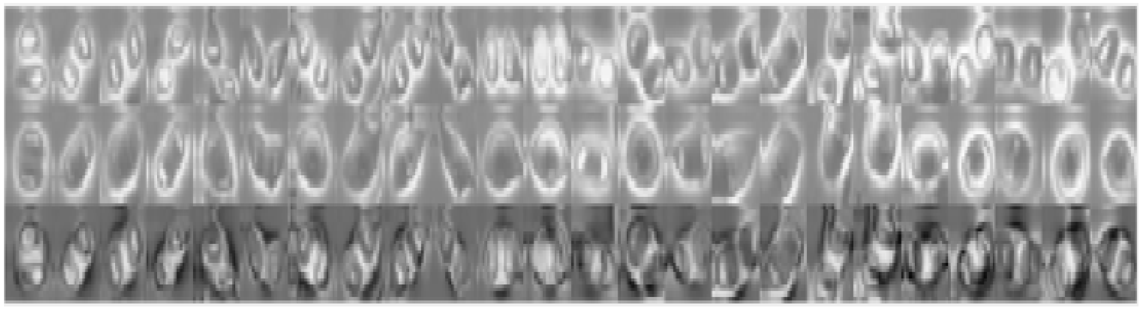


(a)

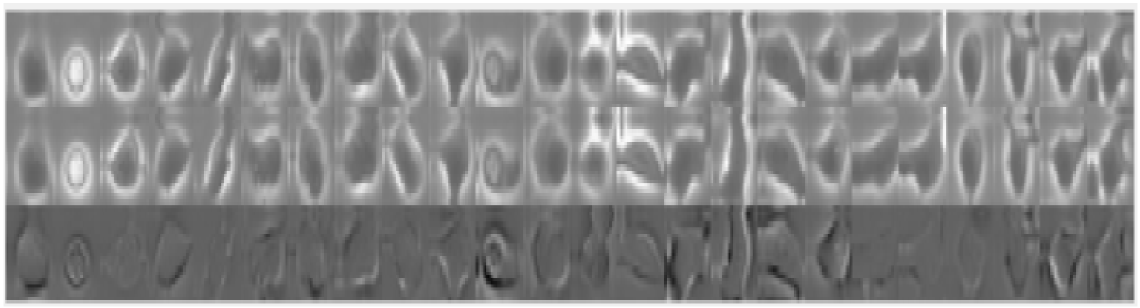


(b)

Figure 4.2: Training sets for NCD implementation - top row is the *t+1* frame and bottom is the *t* frame. The difference image of the discrete wavelet approximations of these images were calculated. The L1 norm of each wavelet approximation was then taken as the compressed size of the image which was used in equation (4.1)



(a) mitotic training data



(b) non-mitotic training data

Figure 4.3: Normalized DWT approximations for training sets for NCD implementation - top row is the *t+1* frame and second is the *t* frame. Bottom row represents the difference image. The L1 norm of the difference image is what is used as the approximation of the compressed size of the difference image.

It can be seen that the difference images in figure 4.3b which represent the non-mitotic data are significantly more uniform while those seen in 4.3a show a bright pattern towards the center of the image. Thus, we utilize the L1 norm of the difference image as an approximation of the L0 norm, which is a measure of sparsity and thus compressibility[12], replacing the compressed size of the image, $Z(\cdot)$, in equation (1.3) with the L1 norm of the difference image. A distance matrix is formed using the NCD equation in (1.3) where x and y represent the two 30x30 images, thus in this case $Z(x)$ would be the L1 norm of the DWT approximation of the difference image and $Z(xy)$ represents the L1 norm of the dwt approximation of image x concatenated with image y. For each cell image to be classified, the affinity matrix was then generated according to

$$A_{ij} = \frac{Z(x_i, x_j) - min\{Z(x_i), Z(y_j)\}}{max\{Z(x_i), Z(y_j)\}} = \frac{\sum_{n,m} x_{ij_{nm}} - min\left\{\sum_{n,m} x_{i_{nm}}, \sum x_{j_{nm}}\right\}}{max\left\{\sum_{n,m} x_{i_{nm}}, \sum x_{j_{nm}}\right\}} \tag{4.1}$$

where x_{ij} denotes the concatenation of image x_i with image x_j and $x_{i_{nm}}$ refers to the pixel belonging to the n^{th} row and m^{th} column in image x_i. Here the first 48 images represent the training data in figure 4.2 with the image to be classified appended as the 49^{th} entry. This affinity matrix A was then used as the affinity matrix in the spectral clustering algorithm defined in 1.1.4 with the final matrix X clustered using k-means clustering. The image to be classified was classified as mitotic if the resulting clustering paired it with the mitotic data and non-mitotic if it was paired with the non-mitotic data.

4.3 LEVER Implementation

The mitotic classification results were implemented into LEVER tracking and segmentation using the NCD classifier. This classifier was chosen due to its significantly better performance with regard to false positives. The goal is to improve LEVER tracking accuracy using mitototic information. Each accurately classified mitosis represents a possible increase in accuracy. However, each false positive represents a decrease in accuracy. Thus, a conservative classifier is best for this application.

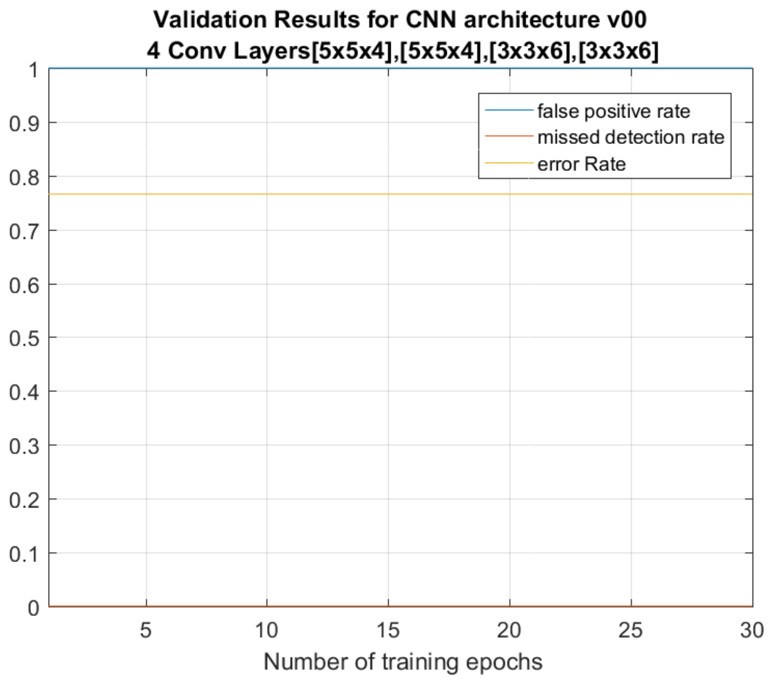
The classification is performed after the original segmentation and tracking is complete. Each hull is classified with the result fed into a resegmentation algorithm. It is assumed that all cells are accurately segmented in the $t = 1$ frame and thus the tracks to be preserved are defined as all those present in this frame. First easily detected redundant positives are removed by evaluating the temporal/spatial proximity of the mitotic events to one another. Next, the movie is then processed frame by frame, preserving only the desired tracks from the initial frame until a mitotic event is reached. Once a mitotic event is reached, the tracking graph attempts to associate it with one of the preserved families. If no match is made, the event is ignored. If a match is made, the hull is split using a Gaussian mixture model and a mitotic event is added considering the newly split hull as the daughter cells and its $t - 1$ pairing as the parent. This approach was chosen in order to assume as little as possible regarding the accuracy of the pre-existing segmentation, since the motivation for the classifier is to resolve pre-existing errors rather than to build on an already perfectly segmented movie. Once a mitotic event was created, a new track was added to those to be preserved and the resegmentation continued until another mitotic event was detected.

Chapter 5: Experiments & Results

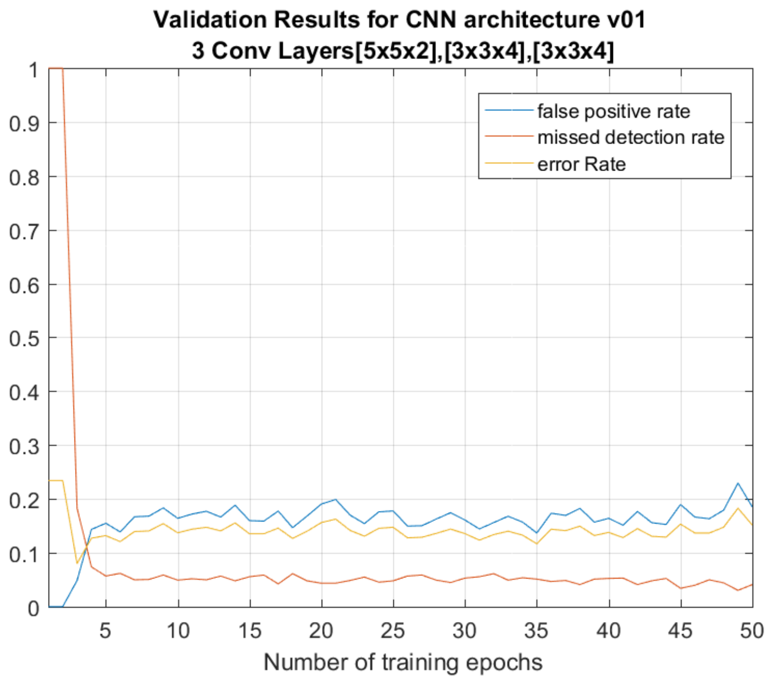
5.1 CNN Results

The data being used is a set of phase contrast images of neural progenitor stem cells taken from mice brains from three separate experiments. Subsets of the data taken on 02/09/14 and 10/03/12 were used as the training set and a subset of the data taken on 09/26/12, which was the most focused on for classification of the NCD, was used as the validation set. This entire data set was also used as the further testing set once a model had been chosen.

The training set consisted of all 1,549 mitotic events across the two training sets as well as 7,613 non-mitotic events. In order to increase the size of the mitotic training set, each image rotated by 90 180 and 270 resulting in 4x the amount of training data, or 6,196 mitotic images. This is justified since our model should ideally be rotationally invariant. Based on other implementations, it appears the general practice is to use an equal amount of training data for each class. When this practice was implemented in this case, however, the training models did not tend to converge to a model of accuracy beyond 55%. As such, the updated training set was updated to the current one. Each model was then validated on a subset of the 09/26/12 dataset consisting of all 1,583 mitotic events in the data set as well as 5,175 non-mitotic events. The latter value was chosen simply as a convenient amount of data that balanced scope and testing time, which was initially a factor. Approximately 30-40 different architectures were tested varying between zero and four convolutional layers, with that shown in figure 4.1 performing optimally with a validation accuracy of 92.69% - a false positive rate of 5.87% and a detection rate of 88.0%. The resulting validation accuracy vs training epoch for the optimal architecture as well as several other varying architectures is shown below in figure 5.1. Note that the model in figures 5.11a, trivially classified all data as either mitotic or non-mitotic through all training epochs, while the models in figures 5.11b, 5.1c, and 5.1d all reached validation accuracy rates between 90% and 100%.

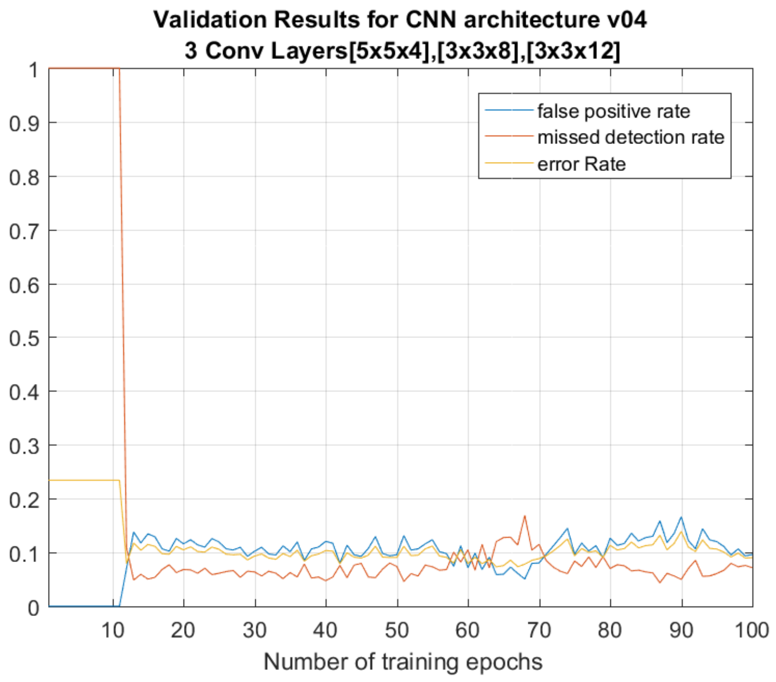


(a)

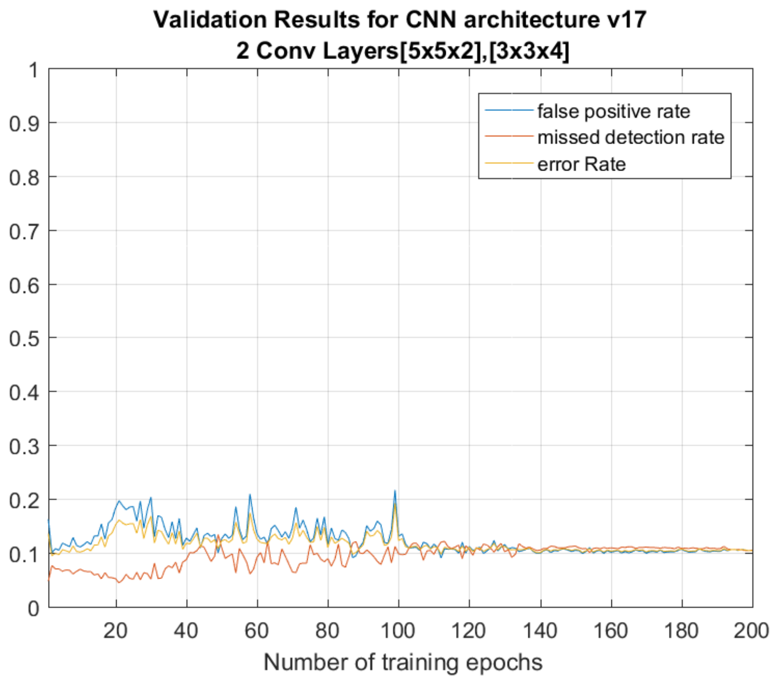


(b)

Figure 5.1: Validation error vs training epoch for several architectures. The dimensions of the convolutional layers are shown above the graph in the form ixjxk where the first two dimensions represent the kernel size and the third dimension gives the number of channels in that layer.



(c)



(d)

Figure 5.1: (cont'd) Validation error vs training epoch for several architectures. The dimensions of the convolutional layers are shown above the graph in the form ixjxk where the first two dimensions represent the kernel size and the third dimension gives the number of channels in that layer.

Compared to architectures in the general vicinity of that in figure 4.1, this particular architecture appears to be somewhat of a sweet-spot. A variety of architectures were tested from one convolutional layer up to four with varying channel depth and varying window size ranging from 2x2 to 5x5. In general, a variety of similar architectures to this one would achieve a validation accuracy from 85-90%, including one with only two convolutional layers rather than three which represents a significant drop in the number of parameters to be trained. However, no other architecture would consistently reach as low as the 7-8% error rate generated by this architecture. Particularly, this architecture achieved the lowest false positive rates that were achieved without sacrificing a completely unacceptable amount of detection accuracy. It is noteworthy, also, that no architecture tested with four convolutional layers achieved a training accuracy above 65%.

The optimal architecture shown in figure 4.1 was further tested on the entire testing data set which included a total of 519,097 images - 1583 mitotic and 517,514 non-mitotic (this should explain why my model leaned in the direction of low false positive rate). However, it was noticed in the data analysis process that a significant amount of the "false positives" came from images that were within three frames of a mitotic frame. Specifically, there were instances where the mitotic frame as well as two or three frames ahead of it were considered mitotic. It does not seem useful to consider this the same as a true false positive as it may actually be useful for the model and so these neighboring positive results were filtered out, but with negligible effects on the actual error rates since the non-mitotic frames are so much more numerous than the mitotic ones (just consider that given the number of mitotic vs non-mitotic frames, if I were to just simply call every cell classify non-mitotic I could claim 99.7% accuracy), it still seemed worth mentioning for the sake of rigour. Below are the resulting ROC curves with the any cell image that was within three frames of splitting removed. With the threshold set at 0.5, the detection rate was 88.0%, as stated above, and the false positive rate on all 517,514 non-mitotic cell images was 5.67%.

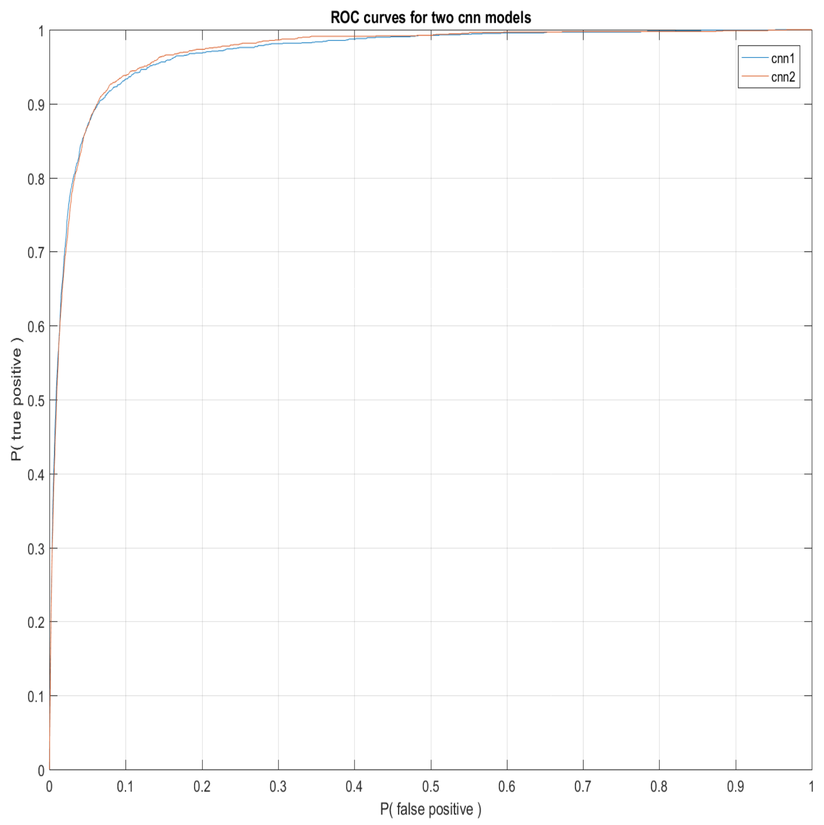


Figure 5.2: Two CNN models from chosen architecture (figure 4.1) - this model consisted of three convolutional layers - $[5x5x4], [3x3x8], [3x3x12]$. The architecture for both is identical but the weights will differ as the models result from different training epochs. CNN2 can be seen to achieve a slightly improved ROC curve from a detection rate of 0.9 and above but CNN1 performs slightly better at lower detection rates between 0.6 and 0.9.

5.2 NCD Results

The NCD implementation as described in 4.2 was also tested on the 519,097 cell images from the 09/26/12 data set. The algorithm performed significantly better with regard to false positives with a rate of only 0.0083 or 0.98%. The detection rate, however, was also significantly lower, successfully classifying 1140 out of the 1549 mitotic events for a detection rate of 73%. The results are tabulated below in table 5.1. Some examples of the cells that were classified are also shown below in figures 5.5 through 5.6. The results for all mitotic images are also visualized in the appendix.

Table 5.1: Results of classification testing on 09/26/12 dataset - this dataset consisted of 519,097 cell hulls, 1,549 of which were true mitotic events

Implementation	Missed Detection Rate	False Positive Rate	Precision
CNN	.097	.0567	.045
NCD	.264	.0083	.187

Confusion matrices for the CNN and NCD classifiers can be seen below in figures 5.3 and 5.4, respectively. Note that even at a false positive rate below 1% in the case of the NCD, the ratio of false positives to true positives is nearly 5:1 due to the significantly greater class probability for the non-mitotic class. The ratio of true non-mitotic hulls to true mitotic hulls is approximately 327:1, as can be seen in the last column of both confusion matrices.

		classification outcome		
		mitotic	**non-mitotic**	**total**
true class	**mitotic**	1,393	190	1,583
	non − mitotic	29,345	488,169	517,514
	total	30,738	488,359	

Figure 5.3: Confusion matrix for CNN classifier - note that the accurate classifications are along the main diagonal while the false classifications are along the opposing diagonal.

		classification outcome		
		mitotic	**non-mitotic**	**total**
true class	**mitotic**	1,174	409	1,583
	non − mitotic	5,094	512,420	517,514
	total	6,268	512,829	

Figure 5.4: Confusion matrix for NCD classifier - note that the accurate classifications are along the main diagonal while the false classifications are along the opposing diagonal.

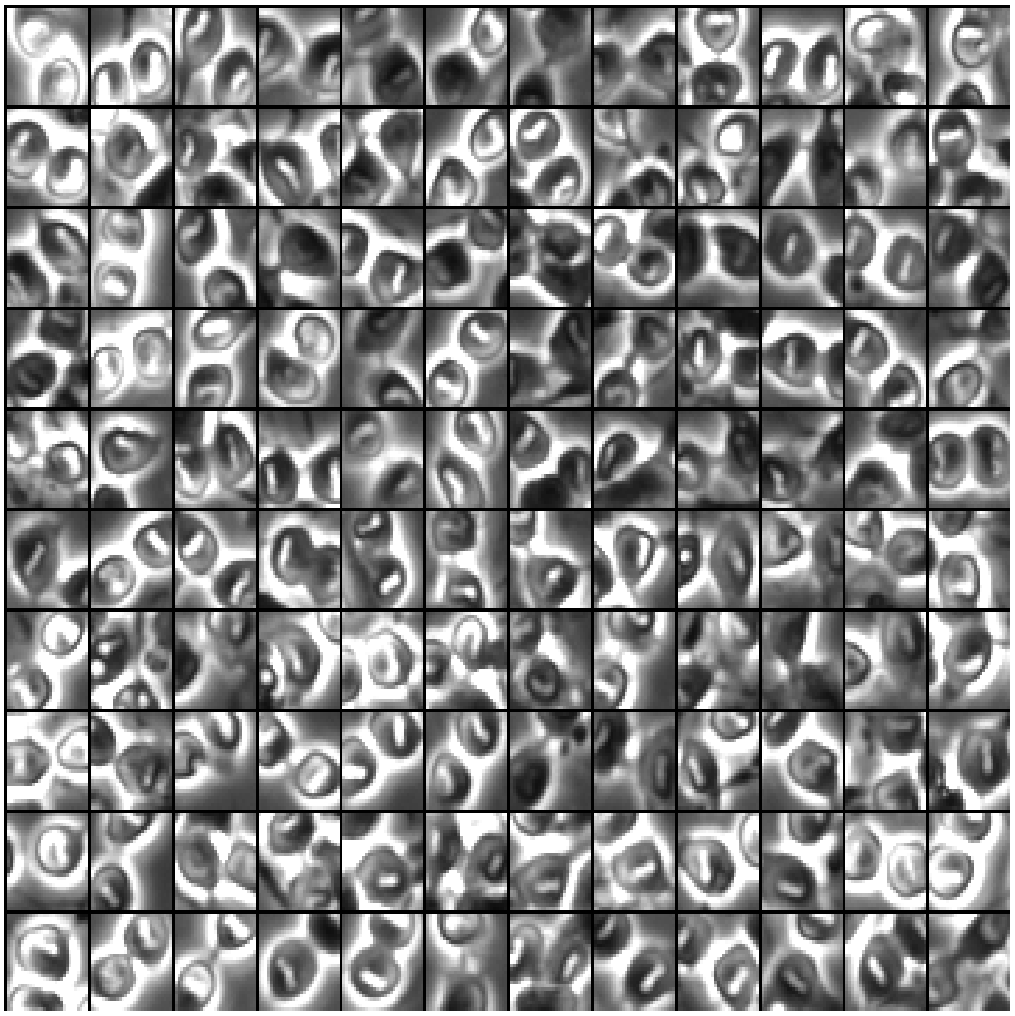


Figure 5.5: A subset of the mitotic events detected by both algorithms

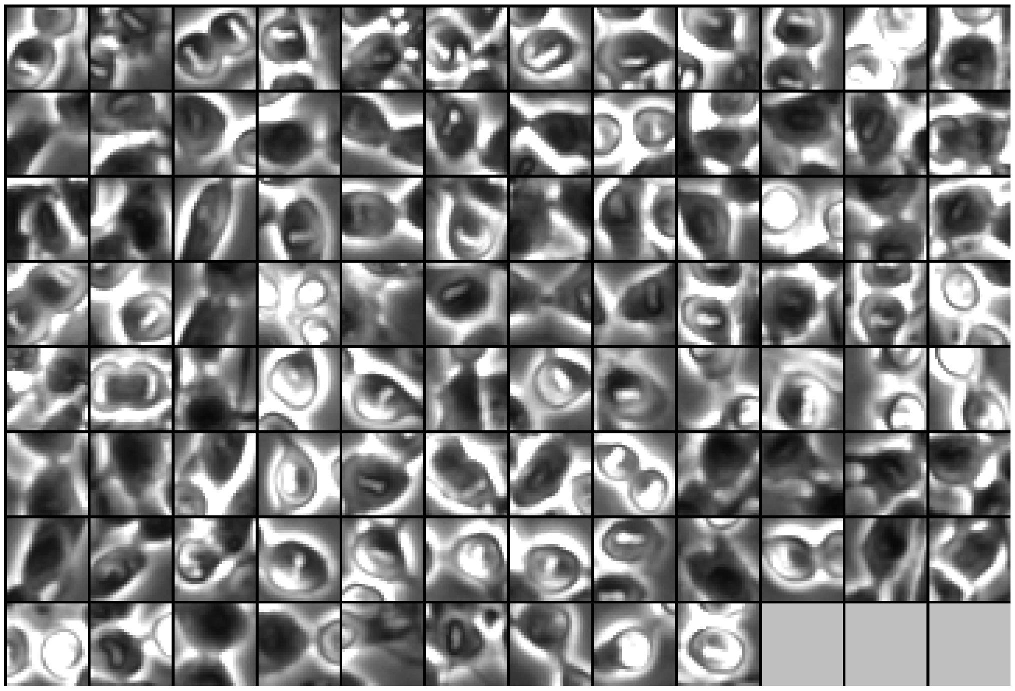


Figure 5.6: Mitotic events missed by just the CNN algorithm

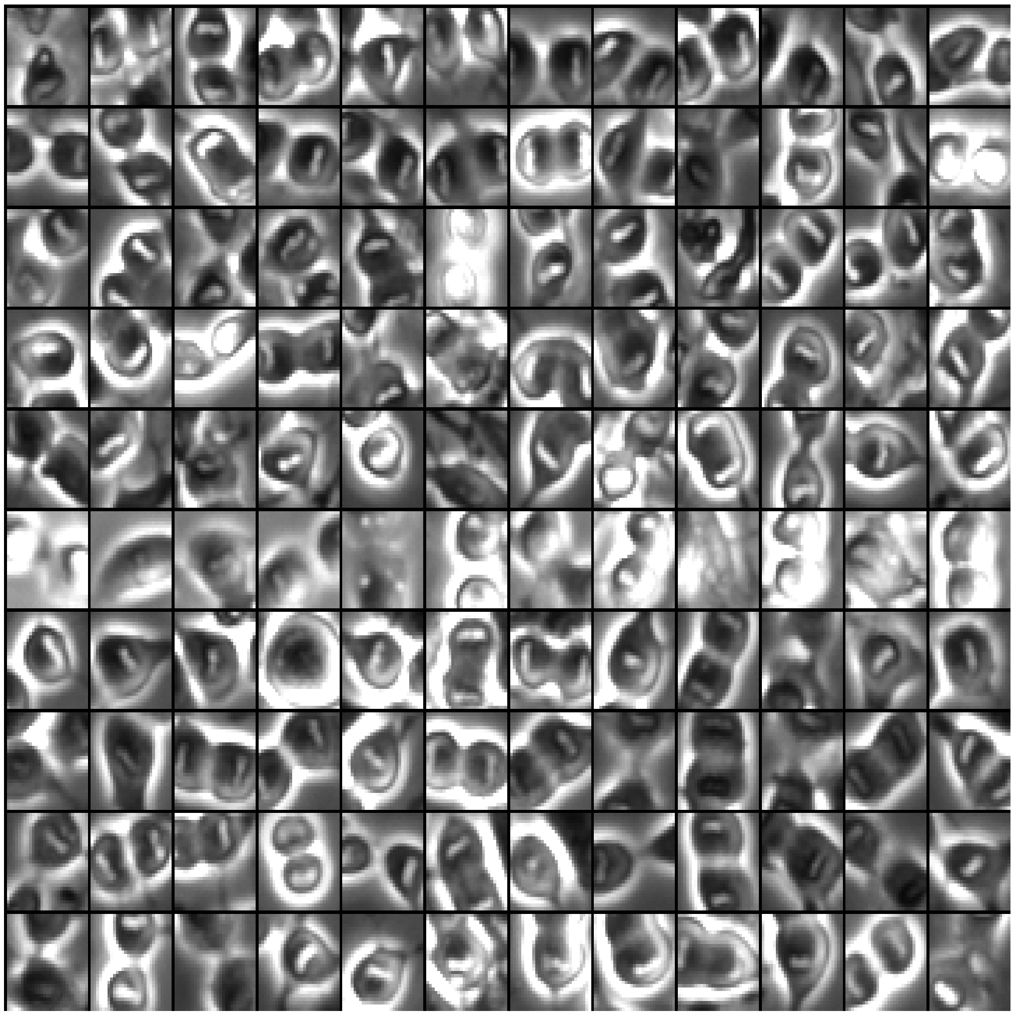


Figure 5.7: Sample of mitotic events missed by just the NCD algorithm

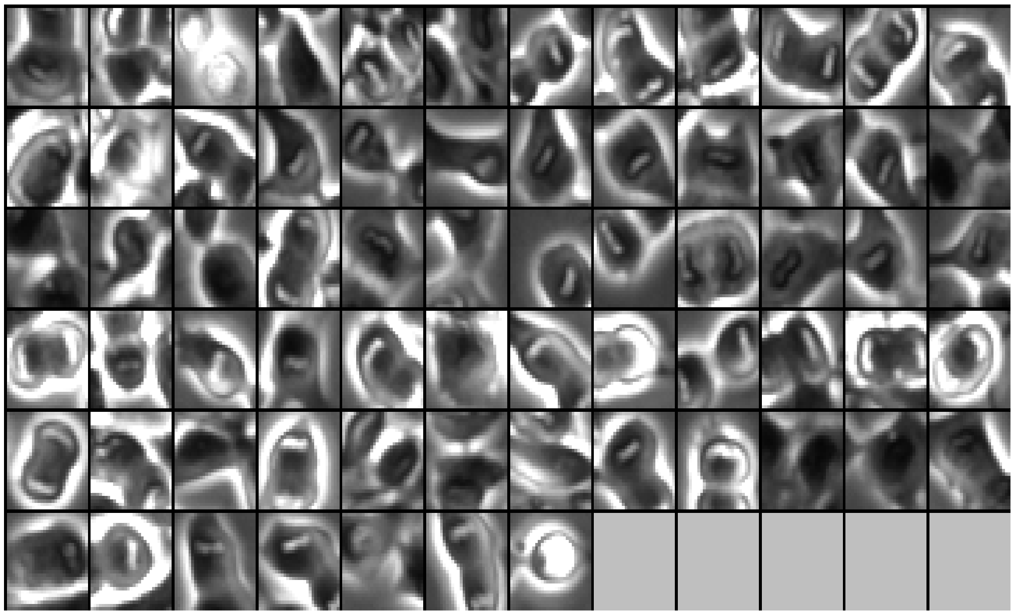


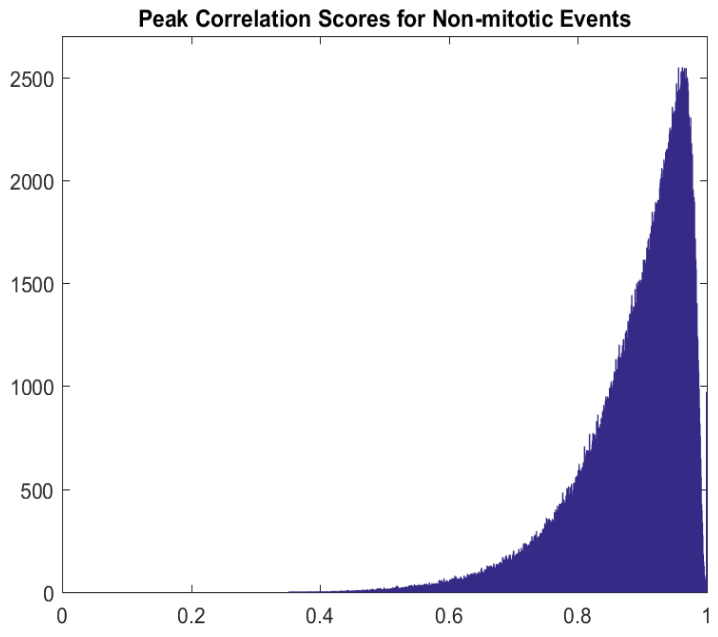
Figure 5.8: All mitotic events missed by both algorithms

5.3 Filtering Methods

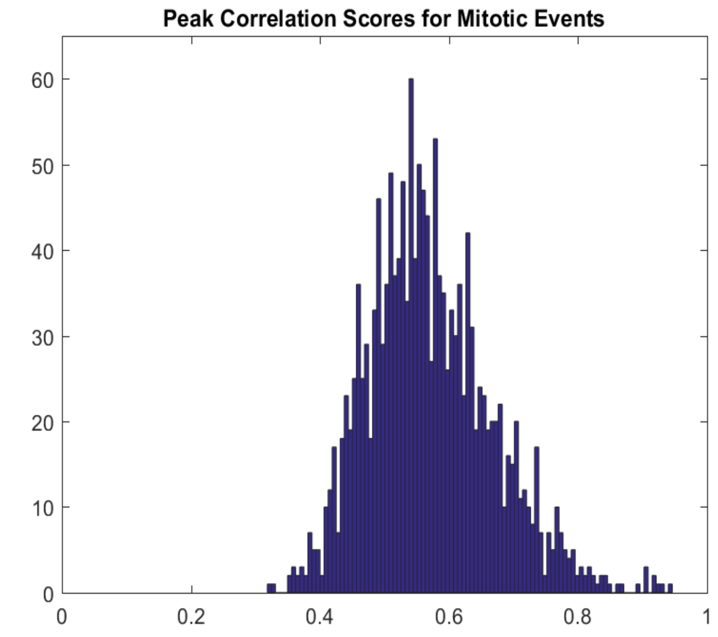
In order to deal with the accuracy issues posed by both algorithms, filtering methods were that could be used in order to optimize the results were explored. The two statistics that appeared to separate mitotic events from non-mitotic events to a certain extent were the L1 of the difference image put through the equivalent of the ReLu function defined in (1.17) which filters out any decrease in brightness from the t frame to the $t+1$ frame, and the peak correlation determined by registering the 30x30 extracted image from the t frame with the respective 90x90 subimage in the $t+1$ frame. The image was registered using the normalized 2D cross-correlation defined as[10]

$$\gamma(x,y) = \frac{\sum_{s,t}\left(w(s,t)-\bar{w}\right)\left(f(x+s,y+t)-\bar{f}_{xy}\right)}{\sqrt{\sum_{s,t}\left(w(s,t)-\bar{w}\right)^2 \sum_{s,t}\left(f(x+s,y+t)-\bar{f}_{xy}\right)^2}} \tag{5.1}$$

where, in this case, the template image w is the 30x30 subimage of the cell, $\bar{w}$ is the average of this region, f is the larger 90x90 subimage and $\bar{f}_{xy}$ is the average of the image f in the overlapping region. The value of γ will vary from -1 to 1 where a high $|\gamma|$ indicates a good match between the template and the image[10]. Below in figures 5.9 through 5.10 are the histograms for the Relu L1 of the difference images for the test data set as well as those for the peak normalized cross-correlation from the centroid of the cell, as well as the ROC curves if each were used as the only classifier in figure 5.11.

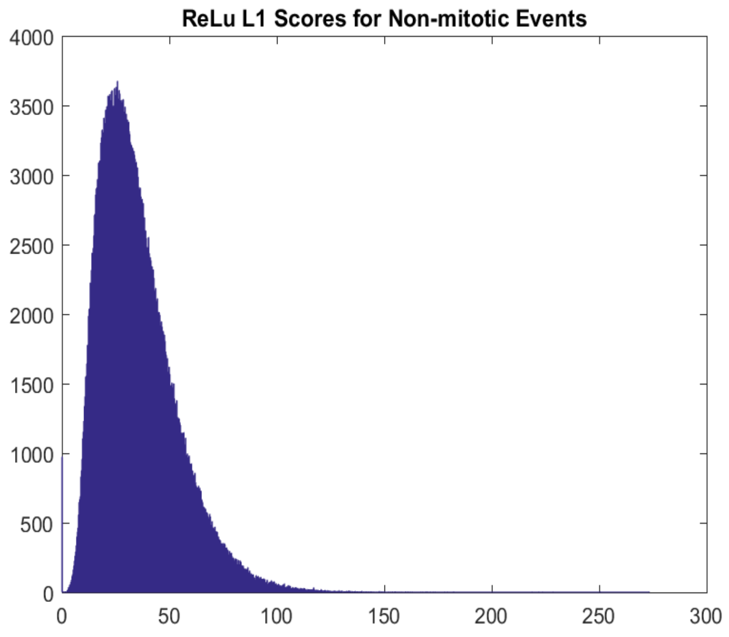


(a)

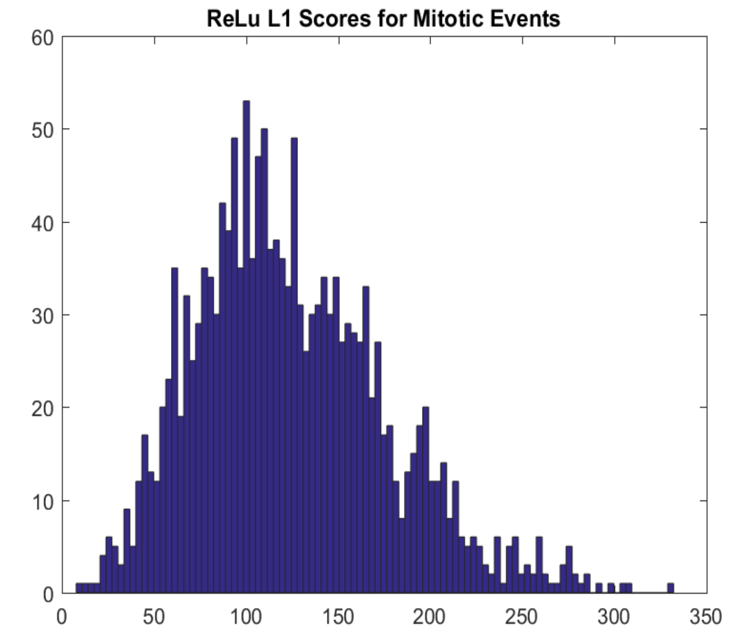


(b)

Figure 5.9: Histograms for the peak cross-correlation of non-mitotic (a) and mitotic events (b) - the histogram in curve (a) shows a relatively clean frequency distribution while that in (b) is a bit more course due to the scarcity of mitotic data.

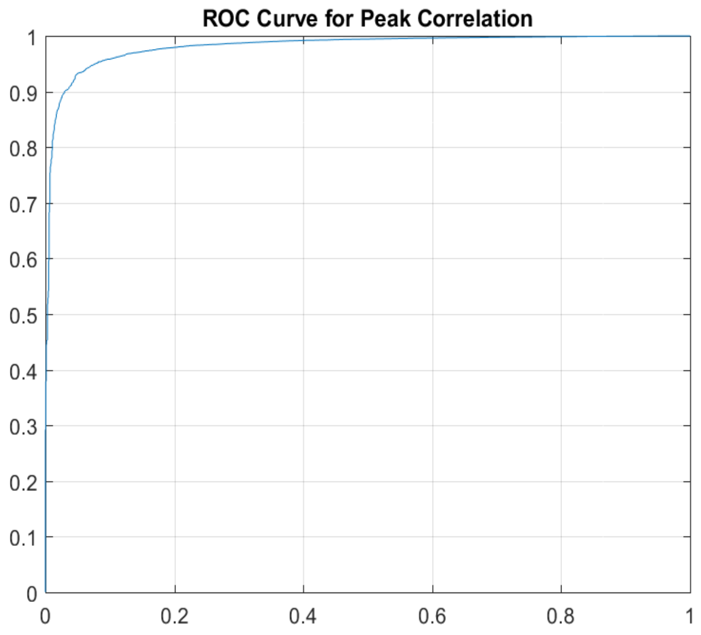


(a)

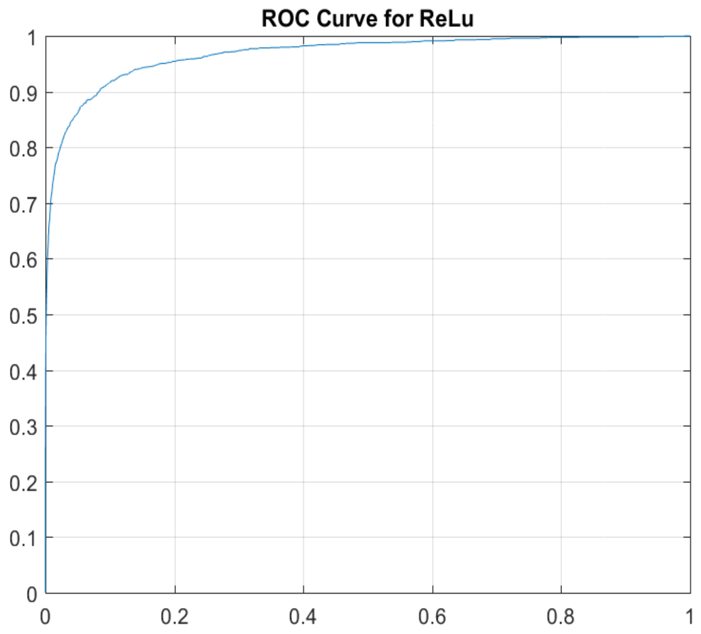


(b)

Figure 5.10: Histograms for the ReLu of the difference images for non-mitotic (a) and non-mitotic events (b) - the histogram in curve (a) shows a relatively clean frequency distribution while that in (b) is a bit more course due to the scarcity of mitotic data.



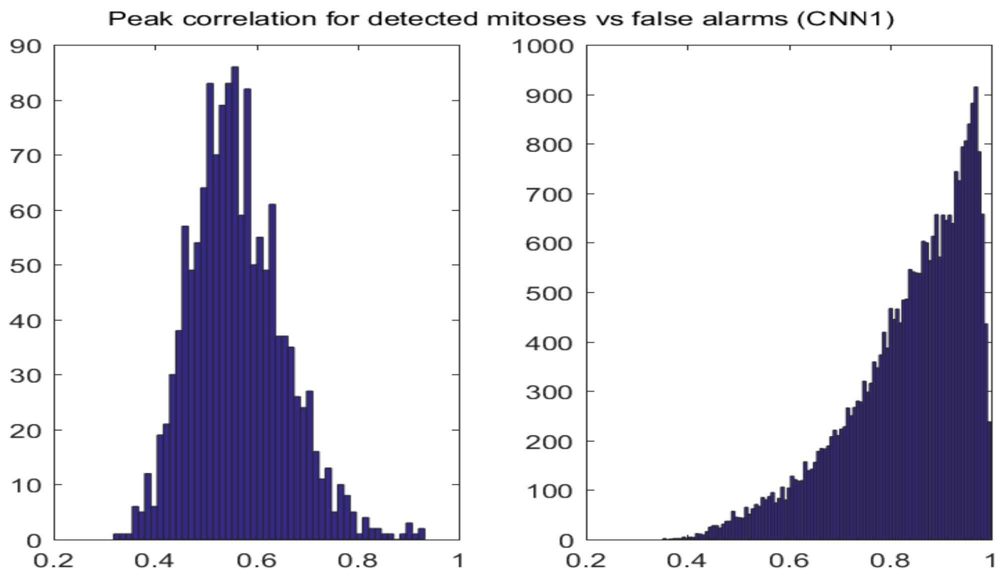
(a)



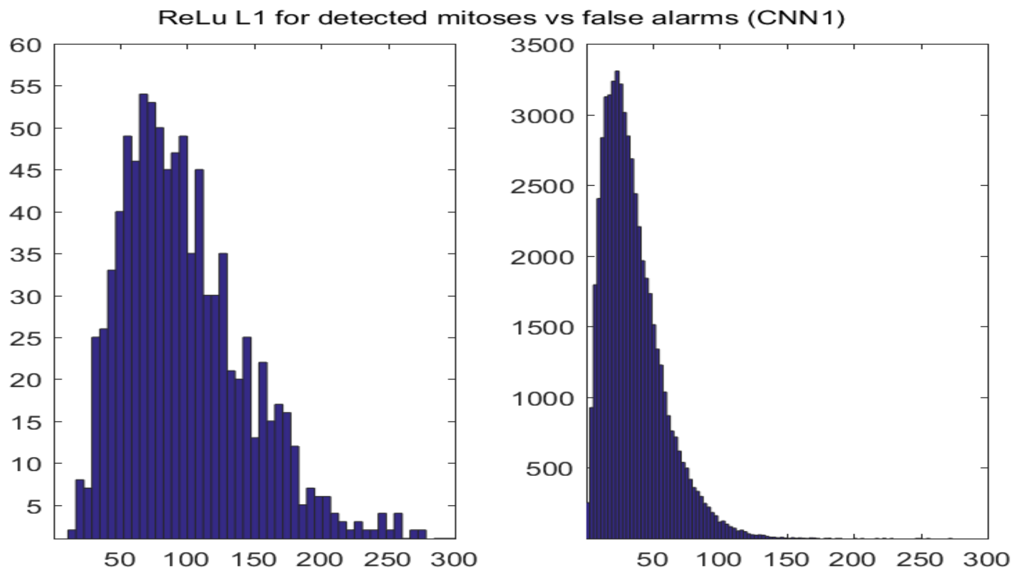
(b)

Figure 5.11: ROC curves for using the peak correlation (a) and ReLu L1 difference (b) as classifiers - both show an ROC curve considerably above the diagonal implying they both contain useful information regarding the locations of mitotic events.

In attempting to use either statistic as a filter out false positives, the question is whether or not the statistics introduce new information and are thus independent of the classifiers already in use. The figures below show the histograms for peak correlation and the ReLu L1 difference of the detected mitotic events vs the false alarms for both CNN classifiers in figure 5.2 as well as the NCD implementation.

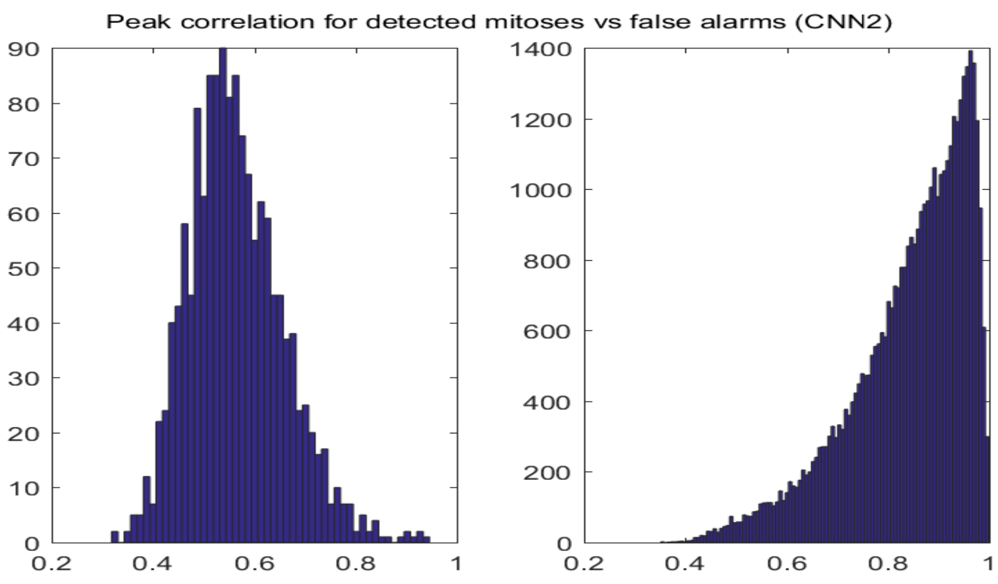


(a)

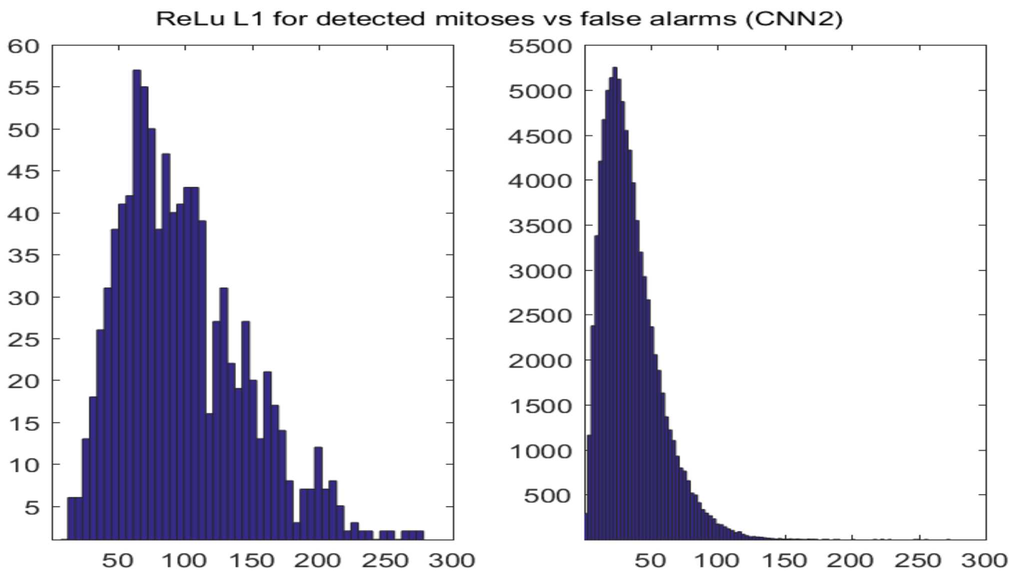


(b)

Figure 5.12: Histograms for peak cross-correlation (top) and ReLu L1 difference (bottom) for first CNN classifier for successful detections and false positives - (a) shows a significant different distribution for the peak correlations of the false alarms vs that of the true positives. This may make this useful to include in a CNN classifier in order to lower the false positive rate.

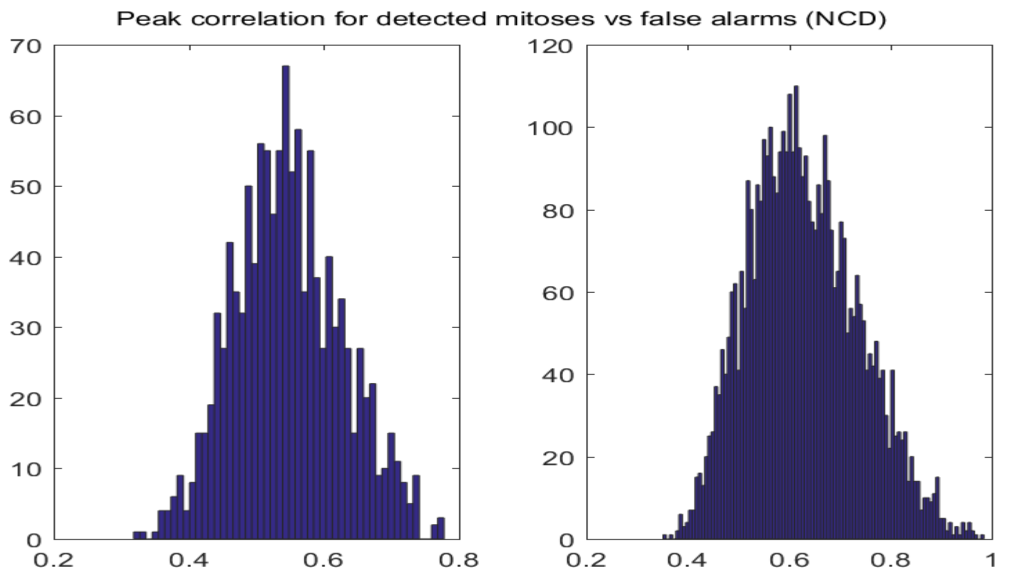


(a)

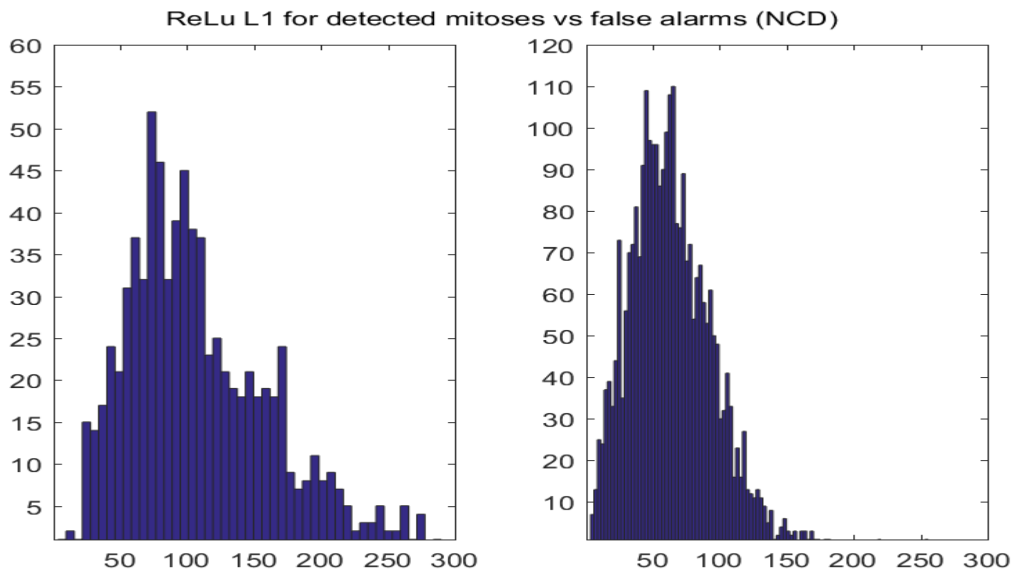


(b)

Figure 5.13: Histograms for peak cross-correlation (top) and ReLu L1 difference (bottom) for second CNN classifier for successful detections and false positives - (a) shows a significant different distribution for the peak correlations of the false alarms vs that of the true positives. This may make this useful to include in a CNN classifier in order to lower the false positive rate.



(a)



(b)

Figure 5.14: Histograms for peak cross-correlation (top) and ReLu L1 difference (bottom) for NCD classifier for successful detections (left) and false positives (right). The differences in the distributions in is not nearly as significant as the difference in figure 5.10a and 5.10b which may imply a correlation between the false alarms and these statistics. This would imply redundant information if both were included into the classification scheme and thus not much gain for the added parameter.

It can be seen from figures 5.12 and 5.13 that the distribution of the peak correlation values is significantly different from the peak correlation values for true mitotic events. Additionally, the distribution of the ReLu L1 difference is shifted slightly to the right for true mitotic events but not quire as dramatically. In the case of the NCD implementation (figure 5.14) the peak correlation of the false alarms appears to be correlated with that of the true mitotic events and so this statistic would not be as useful as a filter in this case. Tabulated results from using the peak cross-correlation as a filtering mechanism for both the NCD and CNN classifiers are shown below in table 5.2. Additionally, the resulting detection and error rates are plotted below in figures 5.15 to 5.17. It is significant that whereas this filtering mechanism does not improve the performance of the NCD classifier, it can be seen to significantly improve that of the CNN classifier such that the approach outperforms the NCD implementation for a range of thresholds.

Table 5.2: Corresponding detection and false positive rates for both the CNN and NCD when false positives are filtered using the peak cross-correlation as a secondary classifier. The threshold was set such that any hull with a peak cross-correlation above that value was set to non-mitotic without consideration of the CNN or NCD classification. Note that at a threshold of 0.7 the CNN outperforms the NCD at any threshold level.

	CNN		NCD	
Threshold	Detection	False Positive	Detection	False Positive
1	.9028	.0544	**.7416**	**.0084**
.9	.8762	.0336	.7416	.0083
.8	.8686	.0173	.7416	.0078
.7	**.8092**	**.0079**	.7113	.0062
.6	.6001	.0030	.5540	.0036
.5	.2243	.0008	.2129	.0009

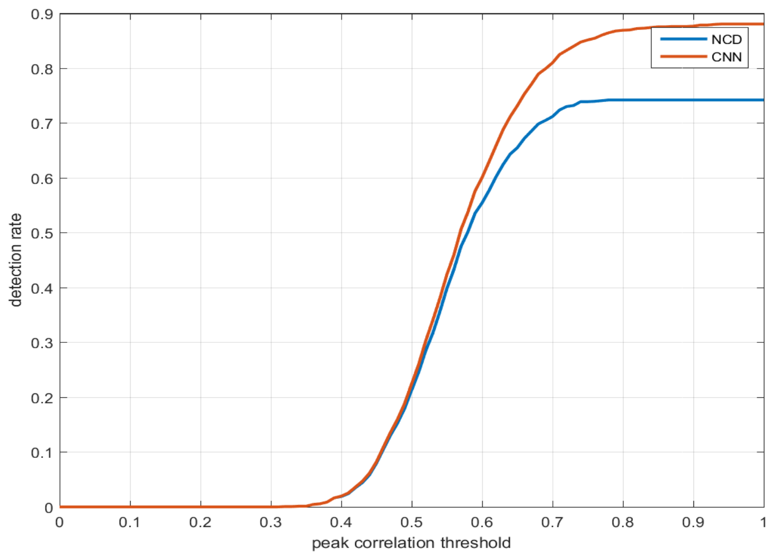


Figure 5.15: Resulting detection rates when varying the peak correlation threshold from 1 to 0. Any cell image with a peak cross-correlation above the threshold was considered non-mitotic without concern for NCD or CNN results. Both detection rates begin to drop off around 0.7.

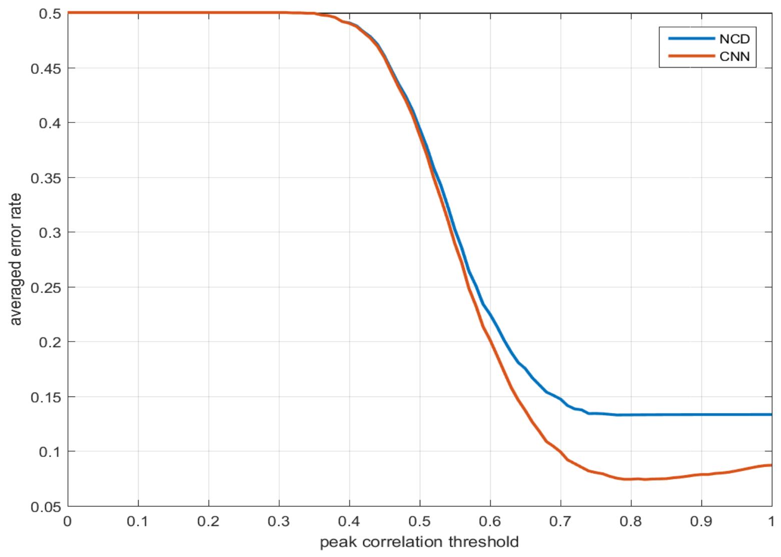


Figure 5.17: Overall error rate taken by averaging those in figures 5.15 and 5.16 above. Note that the CNN performance approaches minimum as the threshold approaches 0.8, while the NCD performance is only affected negligibly until the accuracy drop-off point.

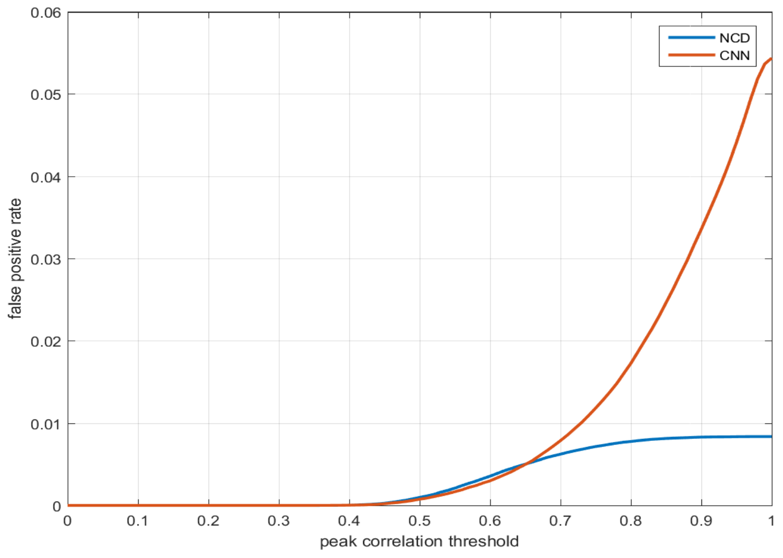


Figure 5.16: Resulting false alarm rates when varying the peak correlation threshold from 1 to 0. The NCD false alarm rate is only affected negligibly up until a threshold of 0.7, which is also where the detection rate begins to drop off as can be seen above in figure 5.15.

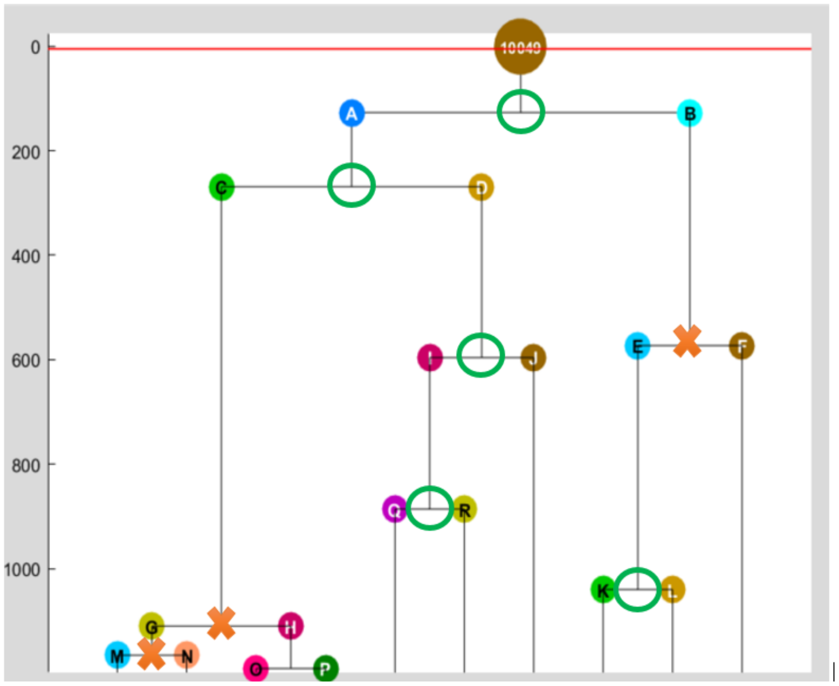
5.4 Lineage Generation in LEVER

Below in figure 5.18 are examples of lineage trees resulting from implementation in LEVER. Recall rates are included for reference, however the precision rates are the main focus of this section. Lower recall rates than the overall rate are due to detected mitotic events being associated with a different cell family. However, these results are intended to demonstrate the impact of the precision rate on lineage accuracy gains from the algorithm. Each branching in the graph represents a mitotic event being detected. Branching marked with an 'X' represent false positives while those not marked represent true detections. The resulting precision rates can be seen below in Table 5.3.

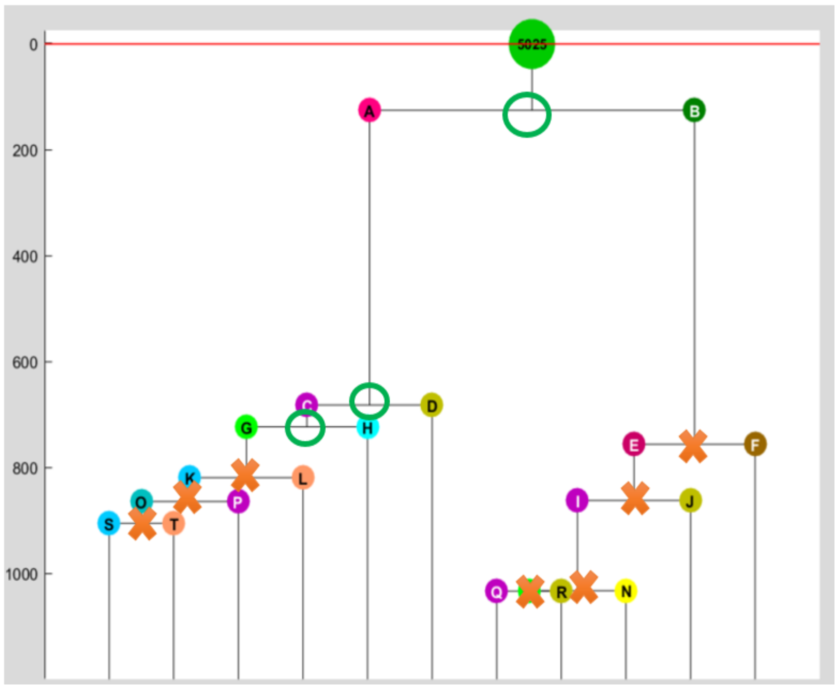
Table 5.3: Resulting precision rates for the specific lineages generated automatically in figures 5.18a through 5.18f. Note that since the ultimate goal of the algorithm is an increase in accury in the lineages above that generated without the mitotic classifier, a precision rate above 0.500 represents an increase in accuracy while a precision rate below 0.500 represents a decrease in accuracy after using the classifier.

Figure	Precision	Recall
5.18a	.625	.270
5.18b	.300	.375
5.18c	.500	.300
5.18d	1.00	.400
5.18e	.800	.800
5.18f	.583	1.00
overall	.500	.394

The precision rate is focused on here because of the particular application. Since the goal is to utilize the algorithm in order to improve the current tracking accuracy, each true positive that is accurately tracked could be considered a gain of 1, while each false positive that is incorporated into the tracking could be considered a loss of one. Thus, only results with precision ratings above 0.5 would be considered to have a net benefit.

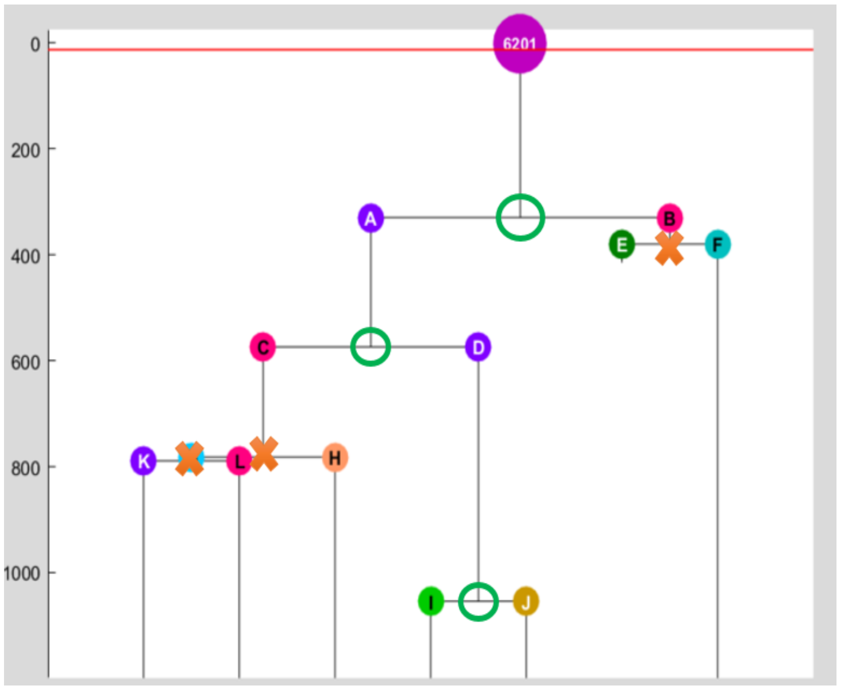


(a) precision - 0.625, recall - 0.270

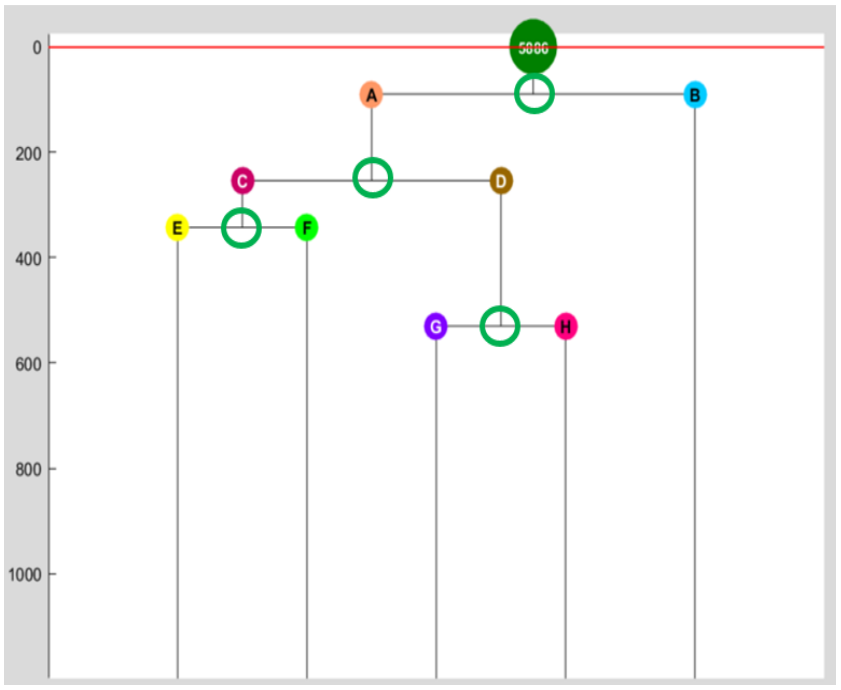


(b) precision - 0.300, recall - 0.375

Figure 5.18: Lineage tree examples using the NCD mitotic classification - the orange x's indicate a false positive included into the lineage tree.

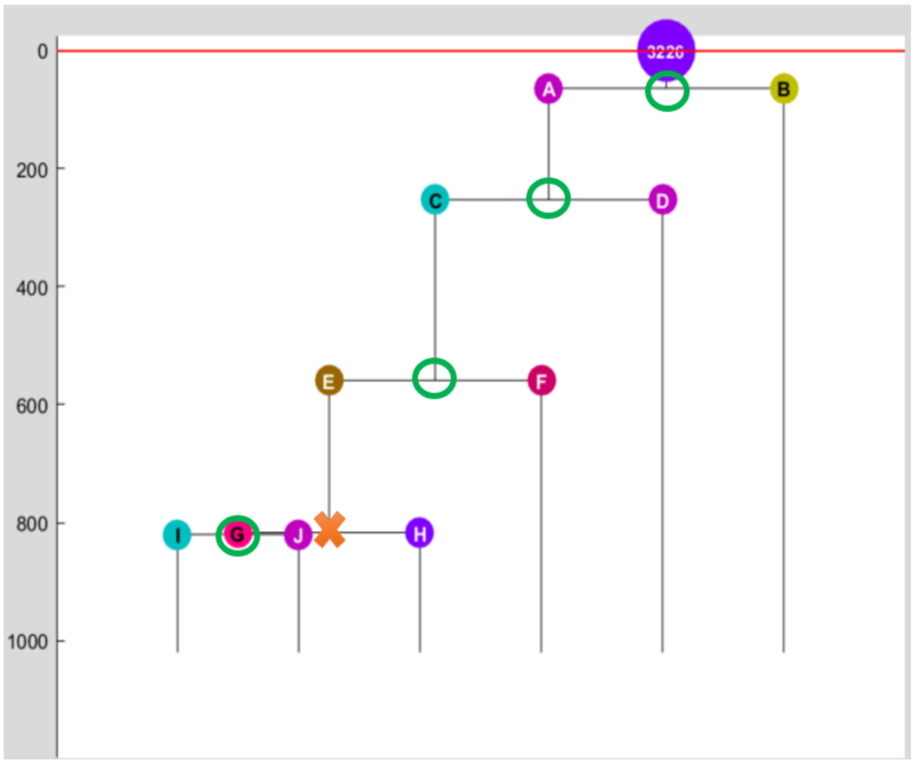


(c) precision - 0.500, recall - 0.300

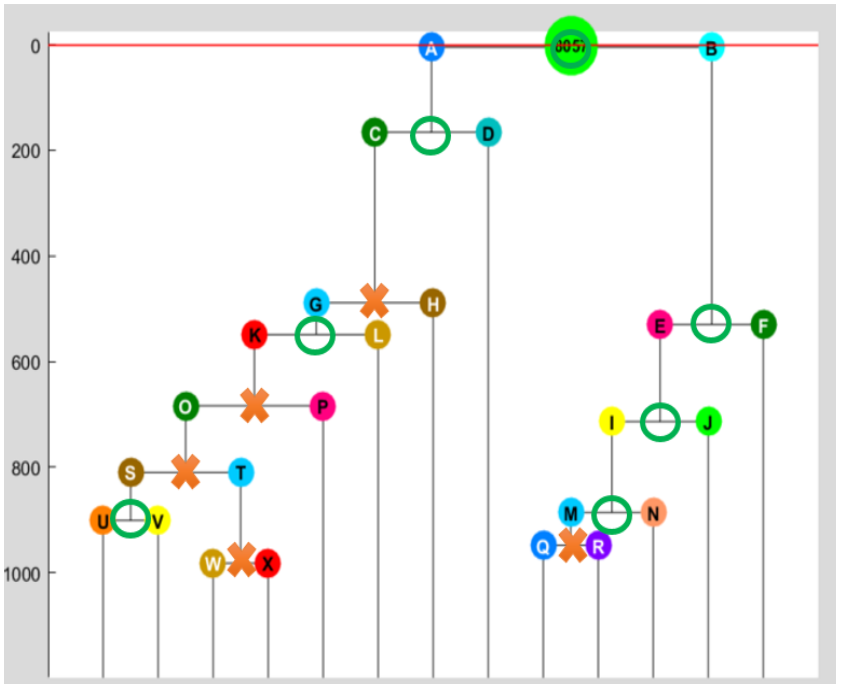


(d) precision - 1.00, recall - 0.400

Figure 5.18: (cont'd) Lineage tree examples using the NCD mitotic classification - the orange x's indicate a false positive included into the lineage tree.



(e) precision - 0.800, recall - 0.800



(f) precision - 0.583, recall - 1.00

Figure 5.18: (cont'd) Lineage tree examples using the NCD mitotic classification - the orange x's indicate a false positive included into the lineage tree.

Chapter 6: Discussion

With respect to overall accuracy on the entire test data set the NCD is far more accurate without adding the filtering modality. However, this is due to the majority of the test images consisting of non-mitotic data and so the overall accuracy rate is approximately one minus the false positive rate. If each type in the test dataset were equally probable, the CNN classifier would perform at 91.2% while the NCD classifier would perform at 86.2%, making it likely that if the mitotic candidates were narrowed down sufficiently the CNN would be optimal. Additionally, once the peak normalized cross-correlation has been introduced as a filtering mechanism for the classifier, however, the accuracy of the CNN can be modified to outperform the NCD as was shown in table 5.2 and figures 5.15 through 5.17.

In terms of computation time the CNN is significantly better once training time is complete. An accurate CNN model can be trained in one to two hours, after which classification time is negligible whereas the NCD classifier takes approximately 10 seconds per hull to generate. In a dataset consisting of 1,000,000 hulls this would amount to approximately 10,000,000 seconds which amounts to 115 days without a parallel implementation, or 3.62 days in the case of a 32-core parallel architecture as was used in this case. Thus, in terms of computational efficiency a CNN classifier has significant advantage.

In terms of optimization, there are several considerations. One significant flaw in the implementation of the NCD is that an effective model using a lossless compressor such as bzip was not successfully implemented. An implementation using an entropy encoder would likely approximate the true NCD more accurately. This could lead to better results and also perhaps be implemented on a single frame image rather than a difference image, removing the dependency on cell movement from frame to frame. However, a lossless compression implementation would increase the already demanding computation time significantly.

With regard to the CNN classifier, there are also several places to look in terms of optimization. The architecture in[16] did not use a single classification but rather a voting strategy with an ensemble of classifications. This approach may make the model more robust with respect to false positives. Another obvious flaw in this implementation was the low amount of training data available for the mitotic events. Successful convolutional networks are often trained on training sets in the range of 10^6 which is three orders of magnitude above what was available for the present experiment. Further, the method of acquiring training data was by using previously labeled cell images extracted based on the centroid of the frame previously. Thus, even the process of acquiring the training data was dependent on the efficacy of the tracking and segmentation. A manually selected training set may have been more accurate with regards to spatial orientation of the event, as well as introducing the possibility of ignoring mitotic events that are not representative of the visual pattern that we are interested in detecting. For instance, if the mitosis is not even apparent to the human eye it may be unreasonable to expect any contemporary classifier to detect it, at least until above-human accuracy has already been demonstrated for such applications.

One major place to look at in terms of optimization is in the way the classifier is implemented into LEVER. Utilizing the normalized cross-correlation as a filtering mechanism is just one place to look as far as making the final decision as to whether or not a mitosis exists. Utilizing additional information based on the tracking and segmentation information *before* it is updated could also provide some useful gains. Rather than simply using the mitotic information as a means of correcting errors, a more prudent approach may be to combine all of the information together in the decision-making process.

Chapter 7: Conclusions

For this particular application and at the level of accuracy presently achieved it appears the NCD is optimal as a single-modality classifier. The CNN implementation, on the other hand, appears to be more improvable by combining it with other classification modalities such as the peak cross-correlation filtering mechanism used in this work. However, the present accuracy of either algorithm does not appear to warrant a significant contribution to lineage accuracies without further optimization. The significant impact of false positives on the results showed clearly why other approaches such as that shown in[19] [20] [21] develop a robust candidate search phase before implementing a classifier. Although the statistical filter implemented did improve results, a more sophisticated algorithm would likely generate a significant increase in precision. Increasing recall, however, may be a more difficult problem. Several CNN implementations reached detection rates as high as 95% but with the parallel increase in false positives that made the model completely ineffective for the task of increasing tracking accuracy. If an extremely reliable approach to candidate search was used to limit the testing regions, significantly higher detection rates may be able to be achieved by optimizing the CNN model for the narrowed-down search region.

Bibliography

[1] Sergios Theodoridis, Konstantinos Koutroumbas, et al. Pattern recognition., 1999.

[2] Mark Winter, Eric Wait, Badrinath Roysam, Susan K Goderie, Rania Ahmed Naguib Ali, Erzsebet Kokovay, Sally Temple, and Andrew R Cohen. Vertebrate neural stem cell segmentation, tracking and lineaging with validation and editing. *Nature Protocols*, 6(12):1942–1952, 2011.

[3] Ming Li and Paul Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer Science & Business Media, 2009.

[4] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[5] Drexel Computational Biology Lab. Computational image sequence analysis. `http://bioimage.coe.drexel.edu/info/`, 2016.

[6] Mark R Winter, Cheng Fang, Gary Banker, Badrinath Roysam, and Andrew R Cohen. Axonal transport analysis using multitemporal association tracking. *International journal of computational biology and drug design*, 5(1):35–48, 2012.

[7] Mark Winter, Eric Wait, Badrinath Roysam, Susan K Goderie, Rania Ahmed Naguib Ali, Erzsebet Kokovay, Sally Temple, and Andrew R Cohen. Vertebrate neural stem cell segmentation, tracking and lineaging with validation and editing. *Nature Protocols*, 6(12):1942–1952, 2011.

[8] Mark R Winter, Mo Liu, David Monteleone, Justin Melunis, Uri Hershberg, Susan K Goderie, Sally Temple, and Andrew R Cohen. Computational image analysis reveals intrinsic multigenerational differences between anterior and posterior cerebral cortex neural progenitor cells. *Stem cell reports*, 5(4):609–620, 2015.

[9] M Mozammel Hoque Chowdhury and Amina Khatun. Image compression using discrete wavelet transform. *IJCSI International Journal of Computer Science Issues*, 9(4):327–330, 2012.

[10] Rafael C Gonzalez, Richard E Woods, and Steven L Eddins. *Digital image processing using Matlab 2nd edition*. Gatesmark Publishing, 2009.

[11] Clemens Valens. A really friendly guide to wavelets. *ed. Clemens Valens*, 1999.

[12] Emmanuel J Candès and Michael B Wakin. An introduction to compressive sampling. *IEEE signal processing magazine*, 25(2):21–30, 2008.

[13] Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

[14] C Bishop. Pattern recognition and machine learning (information science and statistics), 1st edn. 2006. corr. 2nd printing edn, 2007.

[15] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[17] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[18] Seungil Huh, Ryoma Bise, Mei Chen, Takeo Kanade, et al. Automated mitosis detection of stem cell populations in phase-contrast microscopy images. *IEEE transactions on medical imaging*, 30(3):586–596, 2011.

[19] Dirk Padfield, Jens Rittscher, Nick Thomas, and Badrinath Roysam. Spatio-temporal cell cycle phase analysis using level sets and fast marching methods. *Medical image analysis*, 13(1): 143–155, 2009.

[20] Fuxing Yang, Michael A Mackey, Fiorenza Ianzini, Greg Gallardo, and Milan Sonka. Cell segmentation, tracking, and mitosis detection using temporal context. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 302–309. Springer, 2005.

[21] AA Liu, K Li, and T Kanade. Mitosis sequence detection using hidden conditional random fields. In *2010 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pages 580–583. IEEE, 2010.

[22] Dan C Cireşan, Alessandro Giusti, Luca M Gambardella, and Jürgen Schmidhuber. Mitosis detection in breast cancer histology images with deep neural networks. In *International Conference on Medical Image Computing and Computer-assisted Intervention*, pages 411–418. Springer, 2013.

[23] Rudi Cilibrasi and Paul MB Vitányi. Clustering by compression. *IEEE Transactions on Information theory*, 51(4):1523–1545, 2005.

[24] Andrew R Cohen and Paul MB Vitányi. Normalized compression distance of multisets with applications. *IEEE transactions on pattern analysis and machine intelligence*, 37(8):1602–1614, 2015.

[25] Joshi et al. Measuring visual structure in pahse and fluorescence microscopy using image compression.