A DEEP LEARNING APPROACH TO ESTIMATE REPLICATIVE LIFESPANS FROM YEAST CELL IMAGES

By

Justin Maurice Clark

A DEEP LEARNING APPROACH TO ESTIMATE REPLICATIVE LIFESPANS FROM YEAST CELL IMAGES

By

Justin Maurice Clark

A Thesis Submitted to the Faculty of the University of Tennessee at Chattanooga in Partial Fulfillment of the Requirements of the Degree of Master of Science: Computer Science

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

May 2019

ABSTRACT

The budding yeast Saccharomyces cerevisiae is an important model organism for cellular aging. A common metric for determining the lifespan of budding yeast cells is the replicative lifespan (RLS), how many times a mother cell divides in its lifetime. Traditionally, determining the RLS of yeast cells is a tedious manual process. To address this challenge, our long-term goal is to develop an automated RLS estimation process. Recently microfluidics-based methods have been developed, which generate time-series of images of individual cells. This work is focused on classifying these images into categories which can be used to estimate the RLS. We test three different deep learning models and found that all of the models have diverse and complementary errors, so we developed an ensemble of models that combine the best single models which led to high overall accuracy, precision and recall.

ACKNOWLEDGEMENTS

First, I would like to thank my thesis advisor Dr. Hong Qin for his assistance through the research process. Additionally, this work is funded in part by NSF award #1453078 (transferred to #1720215), NSF award #1761839, and internal funding of University of Tennessee at Chattanooga.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

One major goal in aging is to determine which genetic pathways play a key role in slowing down the aging process. The budding yeast *Saccharomyces cerevisiae* is a common model organism because of its short lifespan [1], which means that researchers can gain quick feedback on genomic alterations. There are a couple of ways to determine the lifespan of the budding yeast. The first is the chronological lifespan, i.e. how long does the cell live? The second is the replicative lifespan (RLS), i.e. how many times does the cell divide? In this work, we focus on the RLS.

To accurately estimate the RLS, a large sample size is needed. In [1], the authors devise a mechanism for high-throughput budding yeast cell analysis. Figure 1 from [1] describes the details of this system. Every ten minutes a picture is taken of an entire beacon, which usually consists of over 100 traps depending on how the image is cropped. The raw data is a time-series of these images of beacons. However, parsing the images into individual traps is necessary for the classification task. Each trap image is 60x60 pixels in the greyscale format, which means the pixels ranged in value from 0 to 255, inclusive.

In order to estimate RLS from a time-series of images, we need to classify each of these images into one of four categories: a trap with no cell (nC), a trap with a single mother cell (mC), a trap with a single mother and single daughter cell (mdC) and a trap with more than two cells (exC). This extra class is necessary because it can be difficult to determine where extra cells came from without having a true video feed of the cells. When a trap with more than two cells is encountered, we will throw it away if the necessary information cannot be inferred from earlier and later images. Figure 2 shows how these beacon images look as a whole and also how the different classes are determined from a single trap.

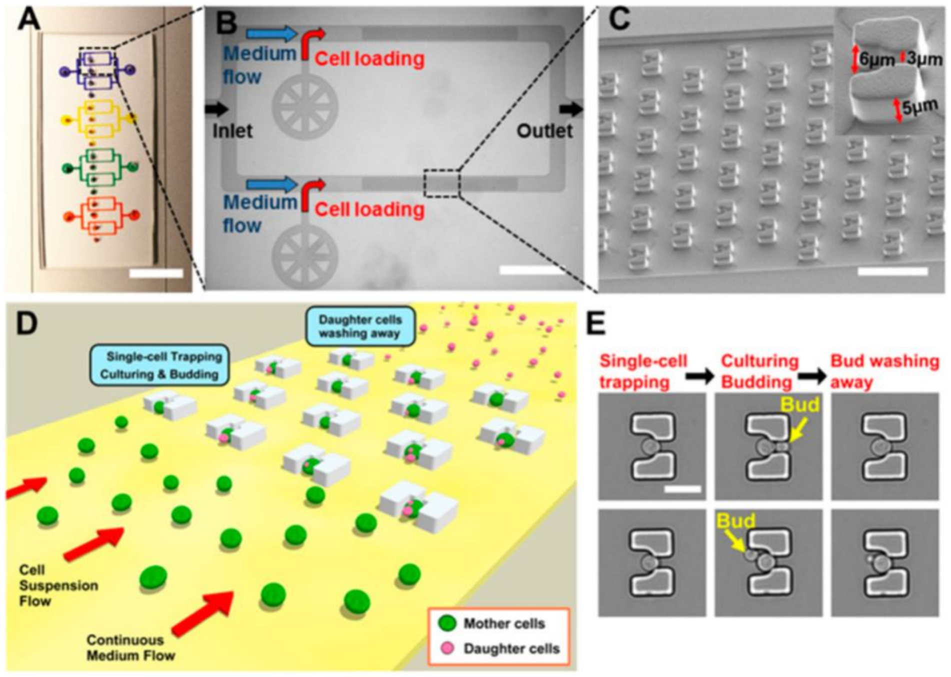

Figure 1 High-Throughput Yeast Cell System [1]: A) Scaled out view of the system containing four separate fluid channels. B) Magnified image of the fluid direction and cell loading locations. C) Microscopic image of the traps that will hold the yeast cells. D) Representation of how the traps will with mother and daughter cells. E) Microscopic images of individual traps and captured mother cells.

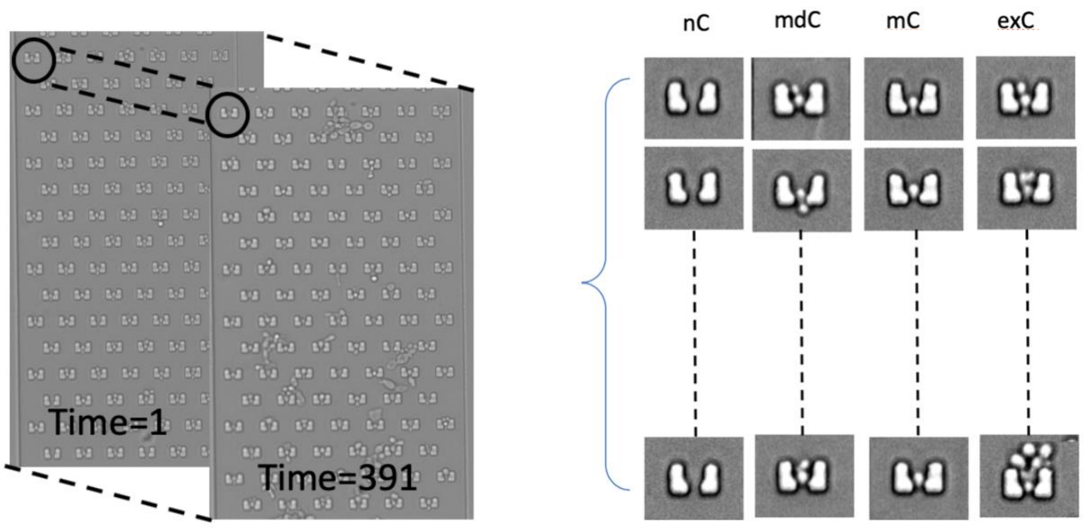

Figure 2 Raw and Processed Data Format [16]: The final images are 60x60 grayscale of individual traps with one of the four labels, nC, mC, mdC or exC.

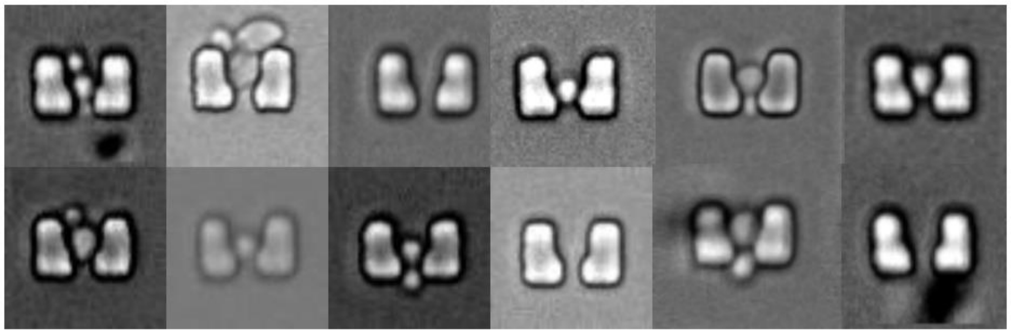

Figure 3 Variability of Dataset: Each individual trap image is highly variable. While traps and cells have a limited number of orientations, the contrast, brightness and image quality all add great complexity to the dataset. There are often shadows, depending on the lighting conditions of the experiment.

The dataset of individual trap images has the following key characteristics: it is relatively small with only a few thousand examples and the examples themselves are highly variable as shown in Figure 3. The purpose of this work is to develop methods to accurately classify images from this small and noisy dataset. In particular, deep learning image classification methods will be used.

In recent years, deep learning has been a popular method for image classification [28]. Many innovations have been driven by creating models that perform well on benchmark datasets such as MNIST, CIFAR10, CIFAR100 and Imagenet [2]. The basic idea of deep learning is to create or "learn" a function that can map a high-dimensional input space into some sort of output vector. In classification, the size of the output vector depends on the number of classes, while regression typically has a scalar output. This work will focus on classification. The function being learned is represented as a computational graph with edges that feed from one layer to the next. Determining the optimal weights (or parameters) of these edges are the primary objective of training deep neural networks as is done through an algorithm called backpropagation [21].

In image classification problems, the convolutional neural network (CNN) is the primary type of deep learning model employed. CNNs are particularly useful for image classification because they are designed for 2-dimensional (or higher) input tensors. In addition, the proximity of pixels in the input

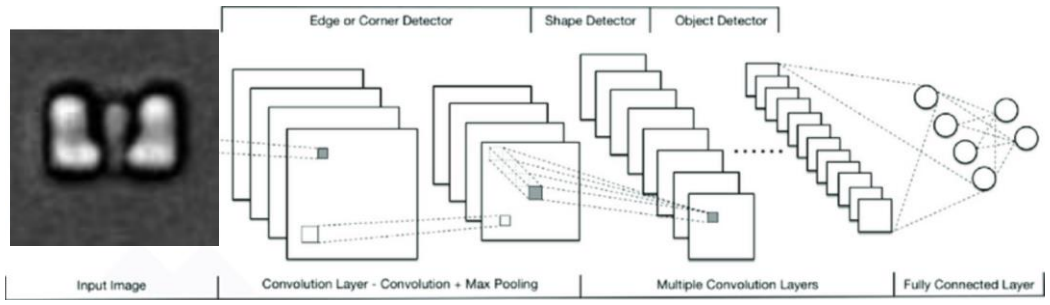

Figure 4 General Convolutional Neural Network Architecture [24]

images are taken into consideration, which helps CNNs learn how pixels should be oriented relative to each other. However, one of the major drawbacks to CNNs is that they require a large amount of training data due to the way the architecture is designed [5].

In Figure 4 above, a general CNN architecture shows the relationship between one layer and the next. In the early layers, convolutional kernels are taken from the original input image and fed to the next layer as an aggregation of all the kernels in a higher dimensional space. As the network gets deeper and deeper, there is an increasing level of abstraction from the original input image that can be learned. In other words, early layers detect simple features like edges and corners while later layers detect complex features like shapes and objects. The reason so much training data is necessary for CNNs is that the pooling layers necessary to convert a higher-dimensional tensor into a scalar can throw out valuable information. This makes CNNs relatively efficient to train (when compared to other image recognition models), but it also means that more data is necessary to offset the inefficient use of training data [5].

The key contributions of this work are as follows: First, labeling training images in a non-trivial way can lead to superior performance, particularly with our small and noisy dataset. Second, augmenting the training data in this problem was another highly effective strategy for improving classification performance. Finally, an ensemble of the top three models performed better than any of the three individual models because each of the three models had diverse and complementary errors, leading to a good “collaboration” between the models.

CHAPTER 2
LITERATURE REVIEW

In this chapter, we will review relevant literature to our image recognition task. First, Ghafari and Qin are working on an alternative to classifying individual trap images [16]. Instead, they are developing object detection and cell segmentation methods to track individual cells with deep learning architectures like YOLO [26] and a Pyramid-based fully convolutional neural network approach to cell segmentation [27]. With better cell tracking, RLS estimation could be much more accurate.

Given that our dataset is relatively small and is quite noisy, ideas and research related to learning on small, noisy datasets will be discussed. There are a few methods for handling smaller datasets, such as the microfluidic images of our problem: augmenting the existing data through basic affine transformations and noise masks [6,7], use concepts from machine teaching to optimally label training datasets [8], use best practices from deep learning architecture design to create the optimal architecture and balance model capacity and model simplicity [9] and finally, use a fundamentally different type of deep learning architecture that should theoretically be able to learn from smaller amounts of data than traditional CNN counterparts called CapsNet [5]. In this paper, we will discuss and test the merits of these methods as it relates to RLS estimation.

2.1 Data Augmentation

There are several ways to augment images from a training dataset, but the way augmentation is accomplished can have important consequences on model performance [23]. Affine transformations on the original images are a popular and simple augmentation method. For example, rotating, reflecting, shearing and scaling are all examples of affine transformations. Depending on the problem at hand, certain affine transformations may be more effective than others [7].

Another method for data augmentation is adding noise to the pixels of the original image. One way of accomplishing this type of augmentation is by creating a random matrix (mask) where each element is sampled from a Gaussian distribution where μ is zero and σ is large enough so the mask elements are meaningfully different from zero, but not so large that when this mask is added to the original image, the result is still noticeable as belonging to the same class as the original image. One more method worth covering is adjusting the brightness and/or contrast of the original image. In many image processing libraries, it is straightforward to increase and decrease the contrast and brightness [10].

2.2 Machine Teaching

Machine teaching is a concept in machine learning with the goal of optimizing the training dataset on which a model learns from. As is true in human learning, it is much easier for machine learning models to learn patterns in the training data if easier concepts (images, patterns, etc.) are seen first with more complex examples seen later [8].

Another way to approach this idea is to re-label training data. In other words, there may be no practical, real-world difference between two different types of images, but distinguishing between the two types of images might be easier for a machine learning algorithm to learn if one class is split into two. Consider an image recognition task where one of the classes of interest is "dog" but within the training examples we only have Pomeranians and wolves. It is theoretically easier for the machine learning algorithm to learn the class "dog" if we first split the Pomeranians and wolves into sub-classes and recombine those sub-classes into a single "dog" class afterwards. More details on the application of re-labeling in the context of estimating RLS of budding yeast cells will be given in chapters 3 and 4.

2.3 Best Practices in Deep Learning SimpleNet

There are no shortages of extremely complex and deep convolutional neural network architectures. Some popular examples are AlexNet [11], VGGNet [12], and GoogleNet [13]. Each of these networks have tens to hundreds of millions of parameters (neural network weights) to learn. With a small dataset this can lead to problems with generalization. One alternative to extremely complex architectures is to design a deep learning architecture with best practices in mind as described in [9], where a model called SimpleNet is proposed. The authors design SimpleNet to have enough capacity to perform well on many complex datasets, but not so complicated that it is unable to learn from smaller datasets.

SimpleNet is a convolutional neural network architecture with 13 layers and anywhere from 2-25 times fewer parameters than the existing state-of-the-art models. There are a few basic principles used to maximize the model capacity while minimizing the overall complexity (number of parameters to learn). First, the architecture gradually increases in size at each layer which allows better parameter utilization. Second, smart kernel sizing lets SimpleNet learn locality of features in each training image. For example, having 1x1 kernels early in the network tend to increase the level of abstraction, but any local correlations are lost. So, 2x2 and 3x3 kernels are used for pooling and convolutions, respectively, until the last two layers. Lastly, instead of designing SimpleNet layer by layer, the authors of [9] chose to think of the architecture in groups of layers where each group of layers is homogeneous and thus, can control overall network size as well as perform specific tasks well, such as object detection.

2.4 Capsule Networks

The last method to improve performance of a model on a relatively small training sample is to use an entirely different architecture altogether. In [5], the authors discuss a novel architecture called a capsule network. Capsule networks (or CapsNet) have recently proven themselves to be effective deep

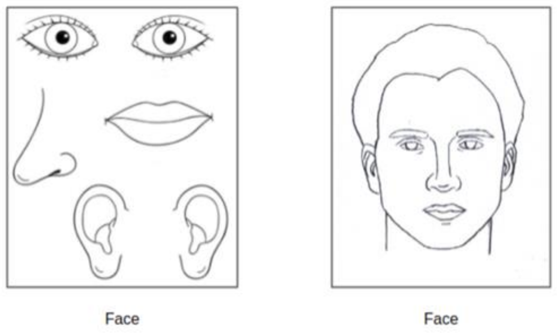

Figure 5 CNN Classification Mechanics: While CNNs are translationally invariant (shifting of an object does not affect output), they have a hard time learning how objects should be positioned relative to one another (i.e., CNNs don't care where objects appear in the image as long as they are there). Capsule Networks, on the other hand, can more easily learn where objects should be relative to one another.

learning architectures in many respects. In some applications, basic CapsNet architectures can outperform extremely sophisticated CNN architectures [5]. The basic idea of CapsNet is to replace the typical pooling layers of CNNs with a more sophisticated weight-routing mechanism. For example, max pooling layers take the most prominent value (most commonly, pixels) from a previous convolutional kernel (or filter) as input to the next layer. Unfortunately, noticing only the "loudest" value will ignore all the information of the other values in the convolutional kernel. Average pooling is one way to incorporate some information from all the values in the kernel, but again, information is lost. Information is always lost in pooling layers because a kernel with many values needs to output a single scalar value. This scalar output can be viewed as the likelihood that some feature from the previous layer is present.

On the other hand, capsule layers in a capsule network will output a vector for convolutional kernel inputs where the length of the vector represents how likely a feature from the previous layer is present and the values of the vector are an encoding of all the affine transformation of the kernel input. With a more data-efficient architecture (i.e. less information loss), fewer samples are required to create state-of-the-art models. For example, in [5], Hinton et. al. discovered that CapsNet can achieve near state-of-the-art performance on the MNIST dataset on 10% of the whole dataset. However, capsule networks also have downsides. For one, the time complexity is much greater in training capsule weights than in training the weights in pooling layers of CNNs. In general, capsule networks also tend to have more hyper-parameters than typical CNNs because they contain all the same hyper-parameters of CNNs, but with additional parameters which will be discussed later. Thus, the hyper-parameter search space can be quite large.

In the context of our small and noisy dataset, one of the most notable advantages of CapsNet learning the affine transformation of an image is the reduction in the size of training data necessary to create an accurate model. With a CNN, an extremely large number of training images are necessary to capture all of the variance possible within a single image. The architecture of CapsNet is very similar to the architecture of a general neural network. In Figure 6 from [14], we can see these differences. Rather than the traditional pooling layer (block), we see that CapsNet has a capsule layer block (which usually has at least one lower-level and one higher-level capsule layer) as well as a flattening layer that re-normalizes the capsule layer to be in terms that the traditional fully connected prediction layers can make sense of.

One of the trickiest parts of making CapsNet work is figuring out how to train weights. It is clear that a vector output would increase the number of trainable parameters greatly - but by how much exactly? In our image recognition task, we have 60x60 pixel input images. The SimpleNet architecture with 13 layers has a total of 5,490,821 parameters to learn while a simple (relatively speaking) CapsNet architecture with one convolution layer and one capsule layer block has to learn 19,498,768 parameters. Additionally, the weights in the capsule layers undergo a much more computationally intensive training process.

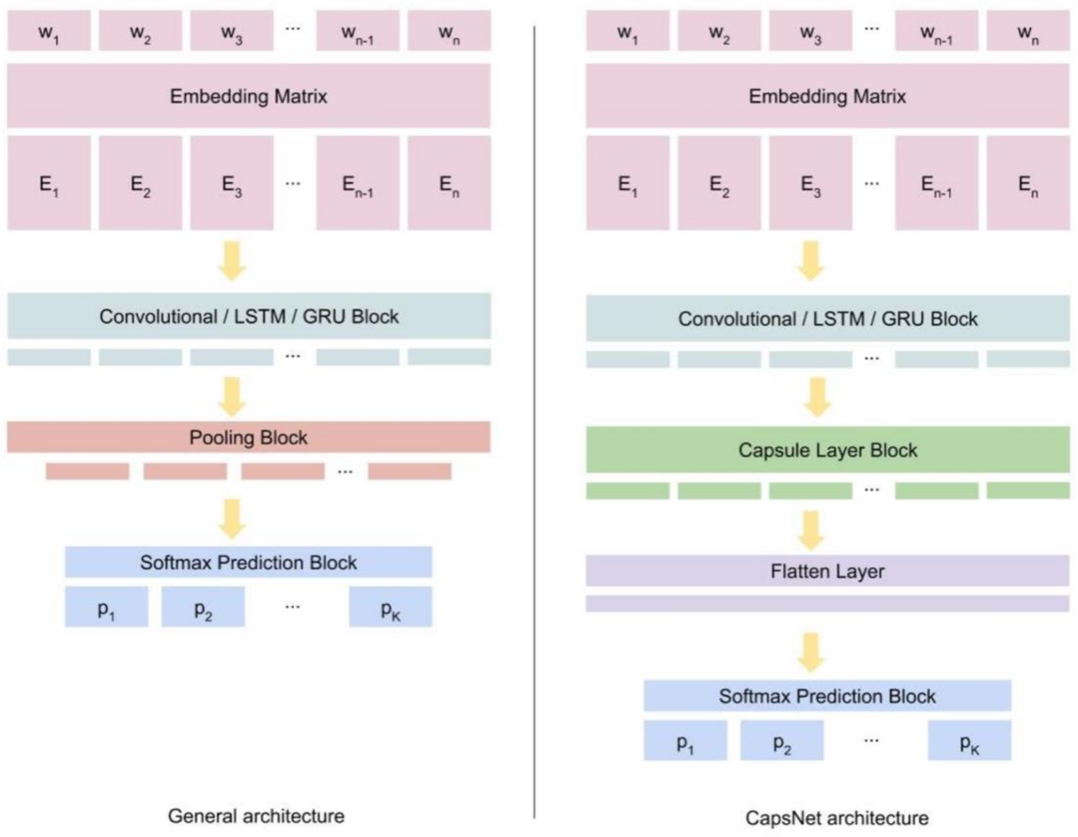

Figure 6 Comparing a General Architecture to a Capsule Network Architecture [14]

Now we will briefly discuss the training algorithm called “routing by agreement” [5]. If we think of pooling as a naive implementation of “routing”, it is easier to understand what the purpose of routing by agreement is. Basically, pooling looks at each element in a convolutional kernel and its output is a scalar that somehow represents the values of that kernel (e.g. maximum or average). With routing by agreement, the important concept is that there are low-level capsules and high-level capsules. This routing algorithm takes place between each low-level and high-level capsule pair to determine how each low- and high-level capsule pair should be related. A longer vector indicates a stronger relationship between each low- and high-level capsule and a shorter vector represents a weaker one. Note that low-level capsules refer to those that are earlier on in the architecture and high-level capsules refer to those that occur sequentially later. This iterative approach is a way of calculating the output of lower-level capsules while looking ahead at higher-level capsules.

2.5 Performance Metrics

To analyze the results of our models, there are a few key metrics we use. The first metric is *accuracy*, e.g. how many exC examples out of the whole sample did we classify correctly? Accuracy is given by the following equation: $Accuracy = \frac{true\ positives + true\ negatives}{total\ examples}$.

The next performance metric is *precision*. An example question that precision answers is, how many times was our prediction for the mC class correct out of all the times we predicted a mC example? The equation for precision is: $Precision = \frac{true\ positives}{true\ positives + false\ positives}$.

After precision, we are concerned with a metric called *recall*. One example of recall is, of all the times a true member of the mdC class appeared, how many times did we correctly classify it? The equation for recall: $Recall = \frac{true\ positives}{true\ positives + false\ negatives}$. Each of these three metrics has its own purpose and are oftentimes used together to determine the overall performance of a model [17].

In the context of RLS estimation and classifying images into one of the categories: nC, mC, mdC and exC (discussed in the introductory chapter), where we can think of each of these classes increasing in order from 1-4. It is important to note that some mistakes are more costly than others in terms of the effect on RLS estimation. It is better for our model to make mistakes in the direction of higher order, e.g. classifying a mdC image as an exC image is less costly than classifying an exC image as a mdC image. The reason for this is because it is easier for the RLS estimation algorithm to parse out the former mistake than the latter [16].

In summary, data augmentation, machine teaching concepts, an architecture based on deep learning best practices (SimpleNet), and a data-efficient neural network architecture (CapsNet) are all potential ways for handling our small and noisy dataset. Additionally, we have the key metrics to analyze each method. The following chapters will discuss how we tested these ideas and the results of each.

CHAPTER 3
METHODS

3.1 Deep Learning Architectures Tested

With the goal of classifying each trap image as accurately as possible, we needed to test different models. We tested a baseline convolutional neural network as well as the SimpleNet architecture and the CapsNet architecture, both of which were discussed in the previous chapters. The baseline CNN had the following architecture: two convolution layers, a pooling layer and two fully connected layers at the end for classification. Convolutional kernel sizes were 3x3 and pooling kernel sizes were 2x2. The activation function used is ReLU, as is normal in image classification and the dense layers have drop out probabilities of 0.25 and 0.5, respectively. This baseline model was chosen for its simplicity and similar variants are used in introductions to CNNs [28]. More information about each architecture tested is presented in Appendix A.

3.2 Data Augmentation

As discussed in Chapter 2, data augmentation is one method for achieving better generalization on small datasets. There were three augmentation methods employed in this problem, the first of which was to generate a random matrix where each element was sampled from a Gaussian distribution, $\mathcal{N}(\mu = 0, \sigma = 6)$. The standard deviation was chosen to create enough pixelated variance to possibly blur boundaries, but not enough to change the class of the images themselves. This matrix was then used as a mask and added to the original input matrix (image). The second data augmentation method was brightness adjustment where a single scalar was added to every element to create a uniform adjustment in brightness. These values ranged from (-16, +16), but the pixels were bounded to the range [0, 255]. Finally, the third was a reflection over the vertical axis. This was the most straightforward affine transformation because the traps themselves are symmetrical about the vertical axis.

3.3 Hardware and Hyper-Parameter Tuning

The models trained with 4 NVIDIA Tesla V100 GPUs. This was necessary particularly in performing a grid search on the CapsNet models which were quite sensitive to hyper-parameters, meaning that small changes in hyper-parameter combinations resulted in vastly different outcomes of the models. In order to optimize the hyper-parameters of the CapsNet architecture (the baseline CNN and SimpleNet were not sensitive to hyper-parameters), we decided to run a basic grid search on several hyper-parameters. These were: 1) the number of routing iterations, 2) learning rate, 3) batch size, 4) whether to add noise to training images, 5) the number of epochs in training. The hyper-parameter grid search options are shown in Table 1. Results of the grid search will be discussed in the CapsNet portion of Chapter 4.

3.4 Splitting the "Mother Cell" Class into Two Subclasses

As discussed in the introductory section, some concepts are easier for machine learning algorithms to learn if examples from the same class are different enough that they can be split into two or more separate classes for the purposes of training. Afterwards, these classes will be recombined into a single class for practical purposes. In Figure 7 below, the five classes are shown.

Table 1 Grid Search Options: A total of 108 combinations were tested.

Routing Iterations	**Learning Rate**	**Batch Size**	**Augment Data?**	**Epochs**
3	0.001	128	True	15
5	0.0005	256	False	20
7		512		25

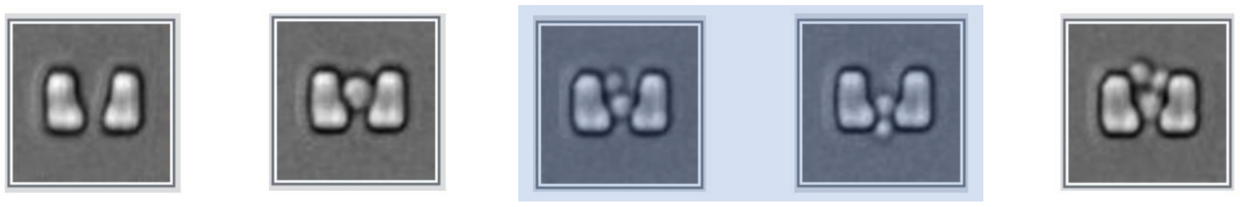

Figure 7 **Class Splitting**: Classes from left to right: No cell (nC), mother cell (mC), mother and daughter cell with the daughter cell on top (mduC), mother and daughter cell with the daughter cell below (mddC), and extra cells (exC).

Notice that the classes mduC and exC seem quite a bit more similar to each other than the mduC and mddC classes. This is because extra cells tend to accumulate over a mother cell. So, instead of making the machine learning algorithm learn the difference between mddC and mduC (since mddC is quite distinct), it can focus on the harder problem of learning the difference between mduC and exC since that is a more difficult difference to learn. Results for the four- and five-class dataset will be discussed in the Baseline CNN portion of Chapter 4.

3.5 Estimating RLS

After a model has been selected that accurately classifies each image into one of the four main categories, the final step is to estimate RLS based on those images. Ghafari and Qin [16] developed an algorithm that will take a series of trap images with classifications as input and will return a family tree and the estimated replicative lifespan of mother cells in each trap time-series. Of course, uncertainty arises when the extra cell (exC) class is encountered. Fortunately, the entire experimental sample of mother cells is not necessary for statistical significance.

CHAPTER 4

RESULTS

In this section, results of significant baseline CNN variations, SimpleNet variations and the CapsNet grid search will be presented. Additionally, we explore the performance of an ensemble of the best models from each of the three variations. Since the grayscale images can hide some cells that are sometimes indistinguishable from background noise to the human eye, the images in this section will be presented with colored contrast.

4.1 Early Problems

Early in the process of model selection and tuning, we discovered many training images that were misclassified. Some of the best models struggled to reach 60% test accuracy with the four-class dataset. By examining the misclassifications, we were able to quickly resolve this issue and relabel the training images.

4.2 Baseline Convolutional Neural Network

After re-labeling the training dataset, we noticed another odd performance issue: the baseline convolutional neural network architecture exhibited great test set instability and hovered around 70%-80% test set accuracy, as discussed in Figure 8 below. Based on these results (and results from CapsNet), we were motivated to split the mother and daughter cell class into the two classes discussed in the previous section, mother cell with the daughter cell on top and mother cell with the daughter cell below. The next step was to train a baseline model with the raw five-class training images without any sort of augmentation. Figure 9 shows the training progress of the best baseline model. These results pointed us towards the fact that we should do some form of data augmentation to see if performance could be significantly affected. The last iteration of the baseline CNN model was to try data augmentation *and* the five-class training dataset. The results of this are discussed in Figure 10, Figure 11, Table 2 and Table 3.

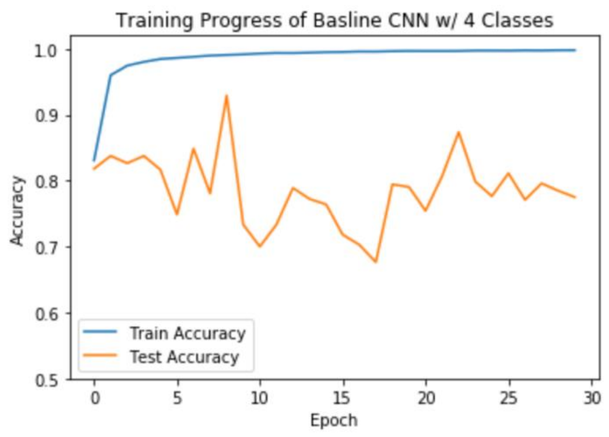

Figure 8 Training Progress of Four-Class Baseline CNN: This chart shows the progress (in particular, the train and test data accuracies) of a model being trained on four classes. While the training accuracy stabilized quickly, the test accuracy remained unstable. Even after using regularization methods such as dropout and augmenting the training data, this was the best performer.

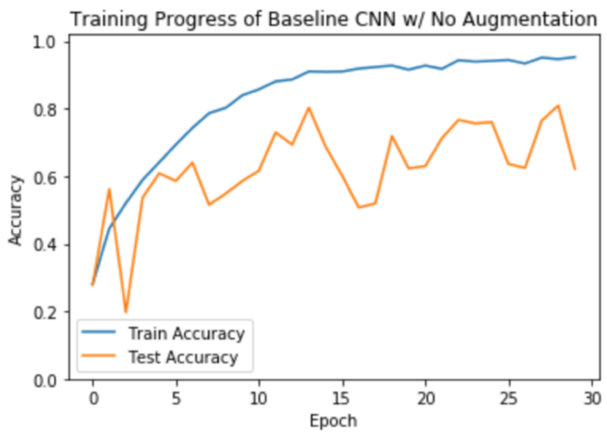

Figure 9 Training Progress of Five-Class Baseline CNN without Augmentation: The train accuracy slowly rises, as does the test accuracy. However, the best results of the test dataset barely reach 80% and are quite unstable.

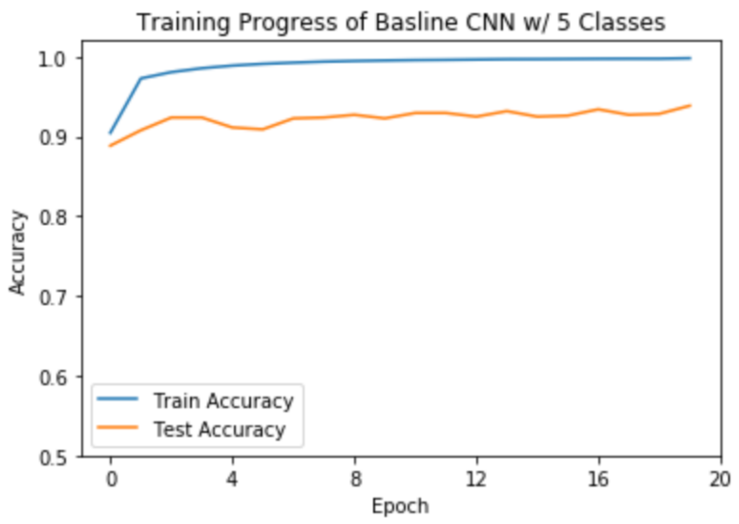

Figure 10 **Training Progress of Five-Class Baseline CNN with Augmentation:** After just 20 epochs, the model trained on a five-class, augmented dataset, was stable and performing quite well.

Table 2 **Baseline CNN Performance Metrics:** After recombining the mddC and mduC classes, the performance of the model increased further with metrics shown in this table. The precision for the exC class was relatively low, which means this model predicted the exC class often, but was not correct in many of those instances. The recall of the mdC class was also relatively low, which says that of the actual mdC images we encountered, we only correctly classified 85% correctly.

	Precision	**Recall**	**Count**
nC	1.00	1.00	80
mC	0.92	1.00	180
mdC	0.99	0.85	315
exC	0.85	0.99	176
Avg.	0.94	0.96	751
Weighted Avg.	0.94	0.94	751

Table 3 Baseline CNN Confusion Matrix: There are a couple of key things to note in the above table. First, of the predicted mother cell only images, there were 15 that were actually mother and daughter cells. This is a potentially costly mistake because the RLS calculation might miss some cell divisions. However, it is also likely that a daughter cell would be picked up further down the line in the series of images. The second thing to note is that there were only two of these problematic classifications when a true extra cell image is encountered. Conversely, there were 31 predicted cases of the extra cell class when it was actually an image of a mother and daughter cell. These predictions are not as problematic because the data can be thrown away.

	Predicted nC	**Predicted mC**	**Predicted mdC**	**Predicted exC**
True nC	80	0	0	0
True mC	0	180	0	0
True mdC	0	15	296	31
True exC	0	0	2	174

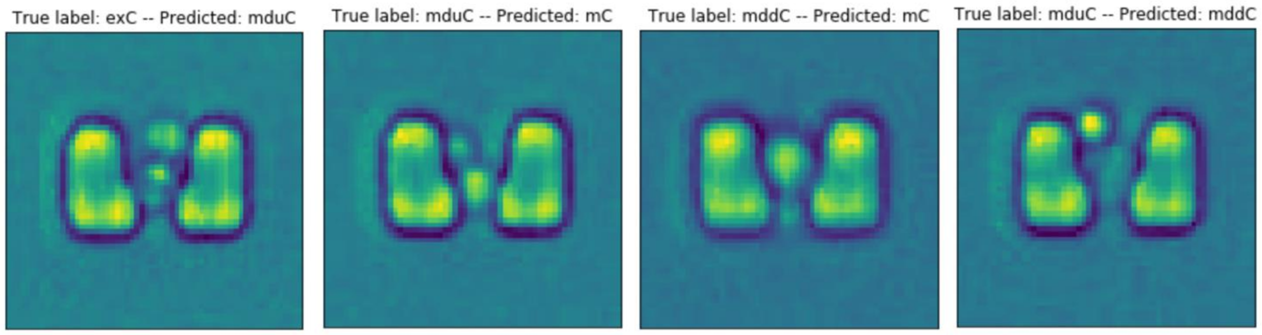
True label: exC -- Predicted: mduC
True label: mduC -- Predicted: mC
True label: mddC -- Predicted: mC
True label: mduC -- Predicted: mddC

Figure 11 Common Baseline CNN Misclassifications: There were two very common types of misclassifications. Image 1 (left) had small cells with very blurred boundaries. Fortunately, image 4 ends up not being a problem after recombining the mddC and mduC classes. Images 2 and 3 (middle) were a little more problematic because the baseline model did not recognize the daughter cells above or below the mother cells. Image 4 was an interesting case because the mother cell is almost entirely transparent. It is possible that the mother cell died, and the daughter cell had nowhere to go.

4.3 SimpleNet Architecture

After reviewing results from the baseline convolutional neural network, we decided to continue training our models on five classes rather than four. Beyond that, we wanted to see how SimpleNet performed on a training dataset with no augmentation. Figure 12 shows and discusses the training progress of this test. Note that the SimpleNet models are trained in four sets of 25-30 epochs as is recommended in [9].

It was immediately clear that we should try augmenting our dataset for the SimpleNet model. Results are shown and discussed in Figure 13, Figure 14, Table 4 and Table 5. In terms of precision, recall and accuracy, SimpleNet was our best performing model, but the next section will discuss why this isn't necessarily the best in terms of accurately calculating replicative lifespan. Fortunately, Section 3.5 will discuss how we can take the high accuracy of SimpleNet and combine it with our baseline CNN and CapsNet predictions to create a superior ensemble of models.

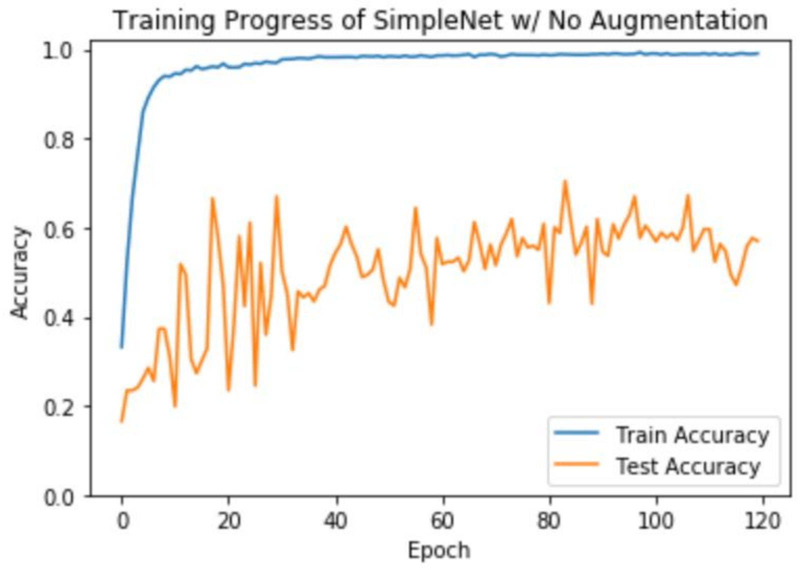

Figure 12 Training Progress of SimpleNet without Augmentation: There are a couple of noticeable problems with this training progress graph. First, we see that the training dataset is learned extremely quickly. Even though the traps and cells are in the same orientation, this is no surprise given that there are only a few thousand examples with fairly drasitc contrast and brightness differences. The second issue we see is the unstable and inaccurate performance of the model on our test dataset.

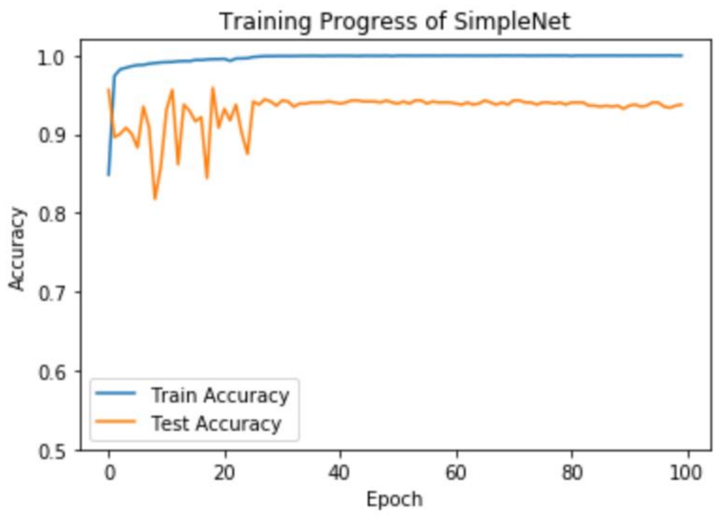

Figure 13 Training Progress of SimpleNet with Augmentation: After augmenting the dataset, there was a substantial improvement in the SimpleNet model. Even early on, the test dataset accuracy bounces from 80% to around 95%. Although it was initally unstable, the model stabilized right around epoch 30.

Table 4 SimpleNet Performance Metrics: Interestingly, the recall and precision of the mdC and exC classes for SimpleNet are almost the reverse of the baseline CNN counterparts. In other words, for every predicted exC instance, the model was right >99% of the time (high precision), but for all actual instances of the exC class encountered, the model only recognized it 85% of the time (low recall). For the mdC class, precision is low and recall is high.

	Precision	**Recall**	**Count**
nC	1.00	1.00	80
mC	0.97	1.00	180
mdC	0.92	0.98	315
exC	0.99	0.85	176
Avg.	0.97	0.96	751
Weighted Avg.	0.96	0.96	751

Table 5 SimpleNet Confusion Matrix: As was highlighted in Table 4, there were a full 26 times where SimpleNet predicted the mdC class when the instance was actually an exC class. Unfortunately, this is a more costly mistake than the incorrect classifications of the baseline CNN. This is because the RLS calculation algorithm might determine there has been a division when there was actually just a daughter cell that floated down from upstream. The 6 instances where the mC class was predicted, but it was actually the mdC class, is a more permissible mistake as discussed in Table 3.

	Predicted nC	**Predicted mC**	**Predicted mdC**	**Predicted exC**
True nC	80	0	0	0
True mC	0	180	0	0
True mdC	0	6	308	1
True exC	0	0	26	150

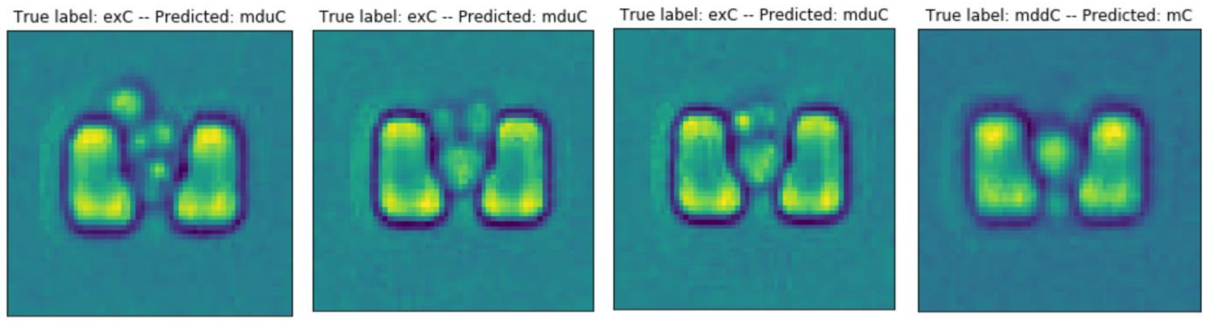
True label: exC -- Predicted: mduC
True label: exC -- Predicted: mduC
True label: exC -- Predicted: mduC
True label: mddC -- Predicted: mC

Figure 14 Common SimpleNet Misclassifications: Image 1 (left) shows an interesting case where it seems obvious that there are several cells clustered together. After further inspection, this image was classified with near 100% certainty. Although this is an uncommon instance, it still poses problems in RLS calculation. The mistake on image 2 is more understandable. There is certainly a mother cell with seemingly two daughter cells on top. However, the algorithm does not classify this in the exC class as it should be. Since one of these cells could actually be a true daughter cell, this image may not be as problematic. Image 3 is an example that is similar to image 2, but the boundary between the two cells on top of the bother cell are so thin that it is reasonable to think that it is a deformed single daughter cell to the untrained eye. Finally, image 4 shows a mistake that was common in the baseline CNN model. This is one of only six misclassifications of the sort and should not cause drastic issues in the RLS calculation.

4.4 CapsNet Architecture

Based on early results, the CapsNet architecture proved itself to be quite sensitive to hyper-parameters, unlike the baseline CNN and SimpleNet architectures. So, the best model was selected based on a grid search (discussed in the previous section). Table 6 shows the top 10 models from the 108 models tested. Interestingly, despite its promise, the capsule network architecture actually had the worst performance of all three models with the best performer only reaching 90.55% accuracy (on 5 classes). Results of the best CapsNet model after recombining the mddC and mduC classes are discussed in Figure 15, Table 7 and Table 8.

Table 6 **CapsNet Hyper-Parameter Grid Search:** There are a few of things to note about the top 10 models. Every single one of the best models had an augmented dataset. This is unsurprising, given the results of the baseline CNN model and SimpleNet, but it is worth pointing out the importance of this again. A batch size of 128 dominated the top 10 models with only 2 models being trained on batch sizes of 256 or 512. The number of routing iterations did not seem to play a major role, but the learning rate and number of epochs did seem to play some sort of role. However, those relationships are unclear.

Routing	Learning Rate	Batch Size	Augment Data?	Epochs	Accuracy
7	0.0005	512	True	20	0.8788
7	0.0005	128	True	25	0.8855
5	0.001	128	True	20	0.8895
3	0.0005	128	True	20	0.8961
7	0.0005	128	True	20	0.8961
3	0.001	128	True	15	0.8961
7	0.0005	265	True	20	0.8961
5	0.0005	128	True	20	0.9028
7	0.001	128	True	20	0.9028
5	0.0005	128	True	15	0.9055

Table 7 **CapsNet Performance Metrics:** Unsurprisingly, the model perfectly classified the nC class. It did struggle with the precision of the mC class, which is not a problem either of the other two models ran into. The recall of the mdC class was also relatively low. However, CapsNet did better than the other two models on the exC class. One possible reason for this is that the CapsNet architecture is good at learning where entities should be relative to one another. So, if there are multiple entities above the main compartment in a trap (where a mother cell would sit), CapsNet should recognize this and classify the example as being part of the exC class. This is in contrast with the SimpleNet architecture that struggled on such questions. Again, the overall accuracy, precision and recall were lower than the baseline CNN model or SimpleNet, but it is promising that what this model learns is *different.*

	Precision	Recall	Count
nC	1.00	1.00	80
mC	0.85	0.92	180
mdC	0.91	0.86	315
exC	0.91	0.94	176
Avg.	0.92	0.93	751
Weighted Avg.	0.91	0.91	751

Table 8 **CapsNet Confusion Matrix:** As seen in the performance metrics of Table 7, CapsNet struggled primarily with the mC and mdC classes. The 29 instances where the CapsNet classified a mdC image as a mC image are problematic in the RLS calculation as discussed in Table 3. However, there were only 11 instances of true exC images that were classified as mdC images. When compared to 26 of these misclassifications in the SimpleNet model, it can be seen as an improvement (though not as good as the baseline CNN).

	Predicted nC	**Predicted mC**	**Predicted mdC**	**Predicted exC**
True nC	80	0	0	0
True mC	0	165	15	0
True mdC	0	29	270	16
True exC	0	0	11	165

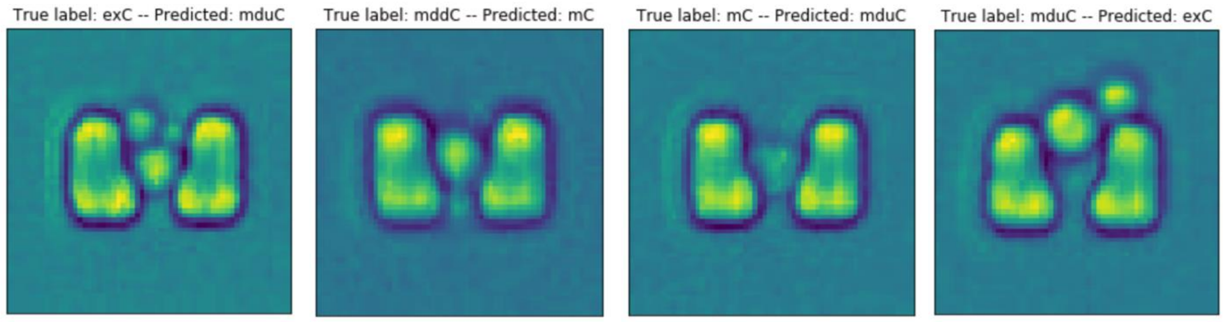
True label: exC -- Predicted: mduC
True label: mddC -- Predicted: mC
True label: mC -- Predicted: mduC
True label: mduC -- Predicted: exC

Figure 15 **Common CapsNet Misclassifications:** In image 1 (left), there is a small cell on the top right portion of the mother cell that seemed to be overlooked by the CapsNet model. One potential cause for this misclassification is that the two cells on top of the mother cell are quite different in size. Image 2 is one of the 29 problematic misclassifications that SimpleNet was good at detecting. Image 3 is an interesting example showing a transparent cell that is likely dead or senescent. This type of image is unlikely to happen often enough for the model to learn effectively. In image 4 (right), we see another interesting example. It looks as though a mother cell was too big for the trap and is reproducing daughters that flow over the outside edge of the trap. Since CapsNet learns how entities should be positioned relative to one another, when a cell is above the trap, it is reasonable to assume the model will predict such an example as being part of the exC class.

4.5 Ensemble of Models

Given the results of the best baseline CNN, SimpleNet and CapsNet models, there are a couple of different ways we could ensemble (i.e. combine) the models together to create a single aggregate model. The simplest method is to average the predictions of each model where each prediction is weighted equally. However, it turns out that weighing the predictions by overall model accuracy achieves slightly better performance. Thus, models in the ensembles presented are weighted by their overall test set accuracy. So, SimpleNet predictions had the highest weight, the baseline CNN was weighted slightly lower and CapsNet had the lowest prediction weights.

We will explore the results of every possible ensemble combination, which are shown in Figure 16. The classification matrices for each of the ensembles are shown in Tables 9-12 below. Performance metrics and common misclassifications of the best ensemble are shown in Table 13 and Figure 17.

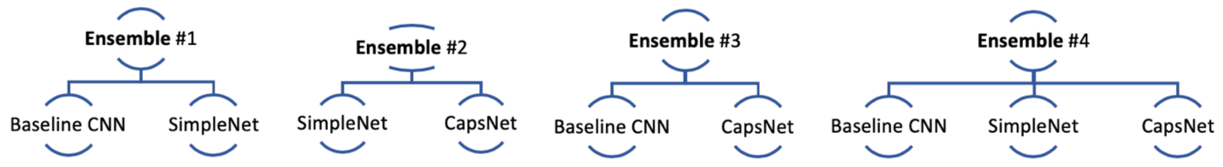

Figure 16 Ensemble Combinations

Table 9 Ensemble #1 Confusion Matrix: There were a total of 28 problematic classifications in this ensemble. Of those, there were 9 instances of a mdC image that were classified into the mC category and 19 instances where an exC image was classified as a mdC image.

	Predicted nC	**Predicted mC**	**Predicted mdC**	**Predicted exC**
True nC	80	0	0	0
True mC	0	180	0	0
True mdC	0	9	303	3
True exC	0	0	19	157

Table 10 Ensemble #2 Confusion Matrix: There were a total of 33 problematic misclassifications and 1 non-problematic misclassification. Although SimpleNet has the highest single-model accuracy, it tended to make mistakes in the wrong direction.

	Predicted nC	**Predicted mC**	**Predicted mdC**	**Predicted exC**
True nC	80	0	0	0
True mC	0	180	0	0
True mdC	0	6	308	1
True exC	0	0	27	149

Table 11 Ensemble #3 Confusion Matrix: There were a total of 17 problematic misclassifications and 28 non-problematic misclassifications. Of course, 28 misclassifications are still undesirable if it can be avoided. However, it seems that the second and third best single models (in terms of accuracy) work well together to create an ensemble of models that is very good from a practical standpoint.

	Predicted nC	Predicted mC	Predicted mdC	Predicted exC
True nC	80	0	0	0
True mC	0	180	0	0
True mdC	0	15	272	28
True exC	0	0	2	174

Table 12 Ensemble #4 Confusion Matrix: With only 16 problematic misclassifications and 11 non-problematic misclassifications, this ensemble is the best by far. Since each model performs well in a unique way, the aggregate of all three models is best from both a practical standpoint and from the perspective of the key performance metrics.

	Predicted nC	Predicted mC	Predicted mdC	Predicted exC
True nC	80	0	0	0
True mC	0	180	0	0
True mdC	0	12	292	11
True exC	0	0	4	172

Table 13 Ensemble #4 Performance Metrics: Given the diversity of each model, they made mistakes in different ways from one another. The CapsNet struggled to accurately predict the mC class but did quite well on the exC class, whereas the baseline CNN and SimpleNet models struggled with either the precision or recall of the exC class. Altogether, the weighted precision of this ensemble was 0.97 and the weighted recall was 0.96.

	Precision	Recall	Count
nC	1.00	1.00	80
mC	0.94	1.00	180
mdC	0.99	0.93	315
exC	0.94	0.98	176
Avg.	0.97	0.98	751
Weighted Avg.	0.97	0.96	751

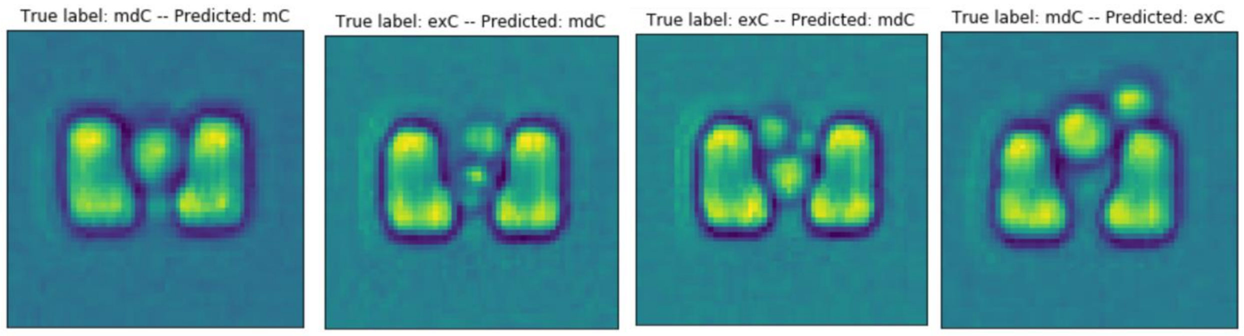
True label: mdC -- Predicted: mC
True label: exC -- Predicted: mdC
True label: exC -- Predicted: mdC
True label: mdC -- Predicted: exC

Figure 17 **Common Ensemble Misclassifications**: Each of the misclassifications of the ensemble are obviously misclassifications of at least one of the three models. Thus, in the previous three sections, we've seen each of the above examples.

Table 14 **Summary of the Best Models:** We can see that the ensemble was greater than its component models individually. A collaboration between the best models leads to a higher precision and accuracy than the best single model and ties SimpleNet on recall.

Model	**Precision**	**Recall**	**Accuracy**
Baseline CNN	0.94	0.94	0.938
SimpleNet	0.96	0.96	0.956
CapsNet	0.91	0.91	0.905
Ensemble #4	0.97	0.96	0.964

CHAPTER 5
DISCUSSION

Given the results presented in the previous section, we will now discuss the main takeaways. The first of the three major results was that splitting the mother and daughter cell class into two separate classes significantly helped the models learn on a small, noisy dataset. Second, training data augmentation is another highly effective strategy for dealing with the noisy trap images. Lastly, an ensemble with diverse models outperformed the single best model. After reviewing these results we will discuss future work related to the automatic estimation of RLS.

5.1 Splitting the mdC Class

The most important part of understanding why it was so effective to split the mC class into a mddC and mduC class is to first understand the nature of the images being classified. If we revisit Figure 7 from Chapter 3, we can reiterate the similarity between the extra cell class and the mother and daughter cell class, when the daughter cell is above the mother cell.

At the highest level, splitting the mdC class created a situation where the neural networks were able to more easily learn the differences of the mduC class and the exC class without having to learn that the mddC and mduC class are the same. Another concept that was not used in this work, but has interesting similarities is transfer learning [18]. Transfer learning is when a neural network starts with pre-trained weights, which can theoretically make it easier for models to learn weights. The machine teaching concept of splitting classes is similar to transfer learning in that both methods attempt to make it easy for the model to learn weights, however these approaches are tackled from different angles.

5.2 Data Augmentation

Many of the beacon images are quite uniform and predictable in terms of brightness, contrast and cell and trap shape. However, Figure 3 from Chapter 1 shows examples to illustrate the non-uniformity of the dataset. Unlike benchmark datasets such as MNIST [2], the microfluidic beacon images do not necessarily have the same brightness, trap color, background color, or even a uniform background. The result of all this noise makes the possible input space of the dataset massive. In addition, the small number of samples we have led to a sparse input space. The result is a sparse and high-dimensional input space.

Recall that machine learning aims to map some input space (represented by all possible inputs) onto an output space. When we augmented our data, we reduced the sparsity of our input space and the machine learning models were better able to learn the mapping between inputs and outputs.

5.3 The Effectiveness of an Ensemble of Models

The final major improvement upon our models (though not nearly as effective as machine teaching or augmenting our training data), was to ensemble our best performing models of each architecture, the baseline CNN, SimpleNet and CapsNet. In the previous section, we saw that the precision and recall for the mdC and exC classes were completely different between all the models. For example, SimpleNet had low recall and high precision for the exC class, but vice-versa for the baseline CNN. On the other hand, CapsNet had relatively high precision and recall for the same class but performed worse in areas the other two models had little trouble with (e.g. the mC class).

The most interesting result here is that CapsNet was not a great model individually, but when combined with either SimpleNet, the baseline CNN, or both, the results were always improved. This implies that the diversity of models plays an important role in ensemble methods. In fact, [19] supports this idea and the authors show that accuracy should not necessarily even be the top metric for determining which models are included in an ensemble. Rather, the authors show that optimizing for error diversity is key to improving overall accuracy.

5.4 Future Work

While correctly classifying images into one of the four discussed categories was the focus of this work, there are still improvements to be made. To approach the image pre-processing differently from data augmentation, we could focus on reducing the size of the possible input space by pre-processing the input images with an edge detection algorithm, so each image only had non-zero pixels on edges of the original images. Additionally, to help make the models learn weights more easily, we could focus on transfer learning [18] in addition to machine teaching. Lastly, we could improve the overall ensemble by adding more diversity to the set of models. For example, the sequential nature of the problem could lend itself nicely to an LSTM architecture [20]. Beyond the deep learning problem, an interface for biologists to quickly input full beacon images and receive the family trees of each cell and a probability distribution for the RLS of the entire experiment is needed for rapid genomic experiments on *Saccharomyces cerevisiae*.

REFERENCES

[1] Jo, M. C., Liu, W., Gu, L., Dang, W., & Qin, L. (2015). High-throughput analysis of yeast replicative aging using a microfluidic system. *Proceedings of the National Academy of Sciences*, *112*(30), 9364-9369.

[2] Datasets. (n.d.). Retrieved from http://deeplearning.net/datasets/

[3] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436.

[4] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[5] Sabour, S., Frosst, N., & Hinton, G. E. (2017). Dynamic routing between capsules. In *Advances in Neural Information Processing Systems* (pp. 3856-3866).

[6] McLaughlin, N., Del Rincon, J. M., & Miller, P. (2015, August). Data-augmentation for reducing dataset bias in person re-identification. In *Advanced Video and Signal Based Surveillance (AVSS), 2015 12th IEEE International Conference on* (pp. 1-6). IEEE.

[7] Perez, L., & Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*.

[8] Zhu, X. (2015, January). Machine Teaching: An Inverse Problem to Machine Learning and an Approach Toward Optimal Education. In *AAAI* (pp. 4083-4087).

[9] HasanPour, S. H., Rouhani, M., Fayyaz, M., & Sabokrou, M. (2016). Lets keep it simple, Using simple architectures to outperform deeper and more complex architectures. *arXiv preprint arXiv:1608.06037*.

[10] Changing the contrast and brightness of an image! (n.d.). Retrieved from https://docs.opencv.org/3.4.3/d3/dc1/tutorial_basic_linear_transform.html

[11] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

[12] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[13] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).

[14] Zafar. (2018). Beginner's Guide to Capsule Networks. (n.d.). Retrieved from https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-capsule-networks

[15] Parameter estimation using grid search with cross-validation. (n.d.). Retrieved from https://scikit-learn.org/stable/auto_examples/model_selection/plot_grid_search_digits.html#sphx-glr-auto-examples-model-selection-plot-grid-search-digits-py

[16] Ghafari, M., Qin, H., manuscript in preparation

[17] Buckland, M., & Gey, F. (1994). The relationship between recall and precision. *Journal of the American society for information science*, *45*(1), 12-19.

[18] Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, *22*(10), 1345-1359.

[19] Opitz, D. W., & Shavlik, J. W. (1996). Generating accurate and diverse members of a neural-network ensemble. In *Advances in neural information processing systems* (pp. 535-541).

[20] Gers, F. A., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with LSTM.

[21] Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network. In *Neural networks for perception* (pp. 65-93). Academic Press.

[22] Ran, F. A., Hsu, P. D., Wright, J., Agarwala, V., Scott, D. A., & Zhang, F. (2013). Genome engineering using the CRISPR-Cas9 system. *Nature protocols*, *8*(11), 2281.

[23] Van Dyk, D. A., & Meng, X. L. (2001). The art of data augmentation. *Journal of Computational and Graphical Statistics*, *10*(1), 1-50.

[24] Ma, Y., Xiang, Z., Du, Q., & Fan, W. (2018). Effects of user-provided photos on hotel review helpfulness: An analytical approach with deep leaning. *International Journal of Hospitality Management*, *71*, 120-131.

[25] Tan, K. (n.d.). Capsule Networks Explained. Retrieved from https://kndrck.co/posts/capsule_networks_explained/

[26] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).

[27] Zhao, T., & Yin, Z. (2018, September). Pyramid-Based Fully Convolutional Networks for Cell Segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention* (pp. 677-685). Springer, Cham.

[28] Advanced Convolutional Neural Networks | TensorFlow | TensorFlow. (n.d.). Retrieved from https://www.tensorflow.org/tutorials/images/deep_cnn

APPENDIX A

NEURAL NETWORK ARCHITECTURES BROKEN DOWN BY LAYER

Baseline Convolutional Neural Network Architecture Layer Breakdown

Layer Number	Output Size	Description
1	(58, 58, 60)	Convolution layer with 3x3 kernel size
2	(28, 28, 64)	Convolution layers with 3x3 kernel and 2x2 max pooling
3	(50176)	Flatten Layer with dropout probability of 50%
4	(128)	Densely connected layer with dropout probability of 25%
5	(5)	Densely connected output layer

SimpleNet Architecture Layer Breakdown

Layer Number	Output Size	Description
1	(60, 60, 64)	Convolution layers with 3x3 kernel and 25% dropout probability
2	(60, 60, 128)	Convolution layers with 3x3 kernel and 25% dropout probability
3	(60, 60, 128)	Convolution layers with 3x3 kernel and 25% dropout probability
4	(30, 30, 128)	Convolution layers with 3x3 kernel, 2x2 max pooling and 25% dropout probability
5	(30, 30, 128)	Convolution layers with 3x3 kernel and 25% dropout probability
6	(30, 30, 128)	Convolution layers with 3x3 kernel and 25% dropout probability
7	(15, 15, 256)	Convolution layers with 3x3 kernel, 2x2 max pooling and 25% dropout probability
8	(15, 15, 256)	Convolution layers with 3x3 kernel and 25% dropout probability
9	(15, 15, 256)	Convolution layers with 3x3 kernel and 25% dropout probability
10	(7, 7, 512)	Convolution layers with 3x3 kernel, 2x2 max pooling and 25% dropout probability
11	(7, 7, 2048)	Convolution layers with 3x3 kernel and 25% dropout probability
12	(3, 3, 256)	Convolution layers with 3x3 kernel, 3x3 max pooling and 25% dropout probability
13	(256)	Convolution layers with 1x1 kernel and flattened
14	(5)	Densely connected output layer

CapsNet Architecture Layer Breakdown

Layer Number	Output Size	Description
1	(52, 52, 256)	Convolution layer with 2x2 kernel size
2	(15488, 8)	Low-level capsule, convolutional layer with 3x3 kernel and where routing algorithm takes place
3	(5, 16)	High-level capsule after squashing output vectors
4	(5)	Densely connected output layer

VITA

Justin Clark was born in El Paso, Texas to the parents of Charles and Cheryl Clark. He is the first of two children, a younger brother. After moving to Wisconsin at the age of two, his childhood was spent in Combined Locks, Wisconsin. He attended Janssen Elementary School in Combined Locks and 6th Grade at JRG in Kimberly, Wisconsin. In 7th grade, Justin moved to Chattanooga, Tennessee and attended Ooltewah Middle School. For high school, he attended Ooltewah High School for his freshman year and East Hamilton High School for the remainder of high school. After graduation, he attended the University of Tennessee, Knoxville for undergraduate studies and majored in Business Analytics. After graduating from UTK, he began his master's degree in computer science at the University of Tennessee, Chattanooga. Justin will work as a data scientist and software engineer after graduation at a company he co-founded.