Fall 2016

USING CONVOLUTIONAL NEURAL NETWORKS FOR FINE GRAINED IMAGECLASSIFICATION OF ACUTE LYMPHOBLASTIC LEUKEMIA

Richard K. Sipes
Eastern Washington University

USING CONVOLUTIONAL NEURAL NETWORKS FOR FINE GRAINED IMAGE CLASSIFICATION OF ACUTE LYMPHOBLASTIC LEUKEMIA

A Thesis

Presented To

Eastern Washington University

Cheney, Washington

In Partial Fulfillment of the Requirements

for the Degree

Master of Science in Computer Science

By

Richard K. Sipes

Fall 2016

THESIS OF Richard K. Sipes APPROVED BY

Dr. Dan Li, Graduate Student Committee — DATE 12/07/2016

Dr. Yun Tian, Graduate Student Committee — DATE 12/07/2016

Dr. Doris Munson, Graduate Student Committee — DATE ~~Dec~~ Dec 7, 2016

ABSTRACT

USING CONVOLUTIONAL NEURAL NETWORKS FOR FINE GRAINED IMAGE CLASSIFICATION OF ACUTE LYMPHOBLASTIC LEUKEMIA

by

Richard K. Sipes

Fall 2016

Acute lymphoblastic leukemia (ALL) is a cancer of bone marrow stems cells that results in the overproduction of lymphoblasts. ALL is diagnosed through a series of tests which includes the minimally invasive microscopic examination of a stained peripheral blood smear. During examination, lymphocytes and other white blood cells (WBCs) are distinguished from abnormal lymphoblasts through fine-grained distinctions in morphology. Manual microscopy is a slow process with variable accuracy that depends on the laboratorian's skill level. Thus automating microscopy is a goal in cell biology. Current methods involve hand-selecting features from cell images for input to a variety of standard machine learning classifiers. Underrepresented in WBC classification, yet successful in practice, is the convolutional neural network (CNN) that learns features from whole image input. Recently, CNNs are contending with humans in large scale and fine-grained image classification of common objects. In light of their effectiveness, CNNs should be a consideration in cell biology. This work compares the performance of a CNN with standard classifiers to determine the validity of using whole cell images rather than hand-selected features for ALL classification.

ACKNOWLEDGMENTS

I would like to thank Dr. Dan Li for her encouragement and feedback throughout the course of writing this thesis as well as Stu Steiner for encouraging me to pursue a master's degree.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Acute Lymphoblastic Leukemia

Acute lymphoblastic leukemia (ALL) is the most common childhood cancer. ALL occurs when bone marrow stem cells develop defects in their DNA that allow them to overproduce. As a result, immature stem cells called lymphoblasts flood the body leading to a long list of aspecific symptoms: bruising, bleeding from gums or nose, infections, bone pain, fever, swollen lymph nodes, shortness of breath, and weakness [23].

Although ALL is treatable, early detection is critical for survival. Doctors require laboratory confirmation through a variety of tests. Some tests are invasive and require a bone marrow biopsy or lumbar puncture. Other tests are minimally invasive and require a single peripheral blood sample. The complete blood count (CBC) is an example of a minimally invasive test. During this test, a blood sample passes through a hematology analyzer that yields quantitative results. Abnormal findings require further investigation through microscopic examination. During the examination, a laboratorian counts the types of cells they encounter and notes qualitative findings like cell morphology.

Normal blood components include thrombocytes, erythrocytes (red blood cells, RBCs), and luekocytes (white blood cells, WBCs). To differentiate them under a microscope, a stain is applied to a peripheral blood smear. Thrombocytes appear as small bluish-purple fragments. RBCs appear as greyish-pink biconcave disks in greater numbers than other components. WBCs contain a dark blue-purple staining nucleus and are subclassed into five types: neutrophil, lymphocyte, eosinophil, basophil, and monocyte. WBC subclassification requires assessment of several morphological char-

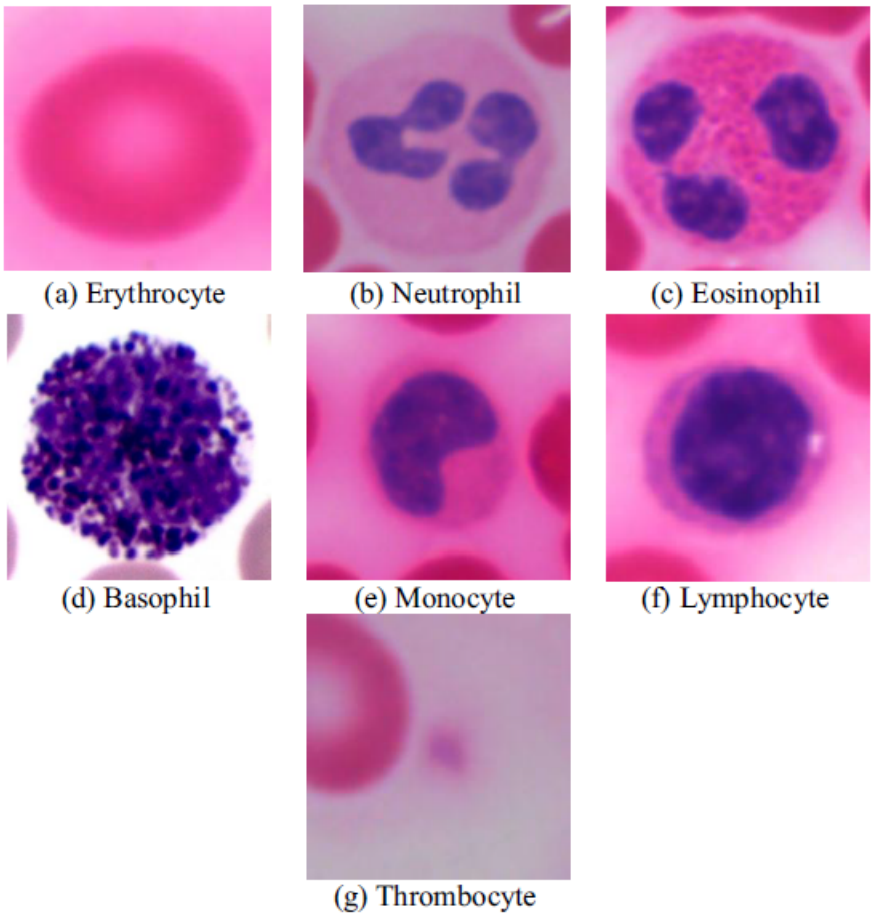


Figure 1.1: Cells commonly found in a stained peripheral blood smear [37].

acteristics including: cell size, cytoplasm color, presence of blue or red staining granules, number of nuclear lobes, cytoplasm to nucleus ratio, and presence of subcellular components like vacuoles and nucleoli [37].

In the case of ALL, lymphoblasts are present in the microscopic examination. Lymphoblasts have a different morphology than normal lymphocytes, but can also vary in appearance amongst themselves. Figures 1.2 and 1.3 show the morphology of normal lymphocytes and three lymphoblast subtypes respectively. A normal lymphocyte has a round nucleus that stains blue-purple and is roughly the same size as a RBC. The nucleus lacks nucleoli, is dense with closed chromatin, and has smooth boundaries. The cytoplasm stains light blue and is scanty, but may be abundant depending on the lymphocyte's reactivity. In contrast, a lymphoblast may have a larger nucleus that stains sparse red-purple. The nucleus may contain distinct nucleoli, have open chromatin, and be indented with rough boundaries. The cytoplasm may stain deep blue but is otherwise scanty. Even with these descriptions, there is considerable variation between ALL subtypes [20].

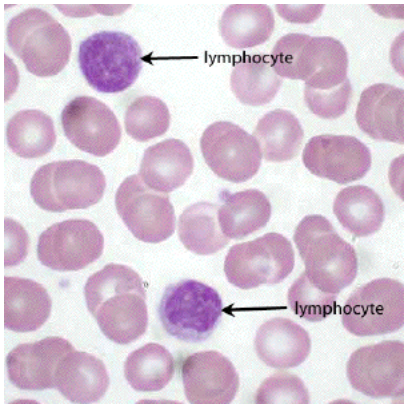


Figure 1.2: A selection of normal lymphocytes in a peripheral blood smear.

Manually distinguishing normal lymphocytes from abnormal lymphoblasts in an objective and consistent manner is difficult. The accuracy of results varies with the observer's skill level and diligence, as well as the quality of the blood sample. Furthermore, manual microscopy is a slow process that takes several minutes to complete. To improve the speed and accuracy of this process, automating microscopy is a goal in cell biology.

Machine Learning

In order to automate microscopic examination, the problem must be broken down into components that can be represented in a computer program. Microscopic images are readily captured and saved as digital files. A computer program must then take these images, which represent structured data, and interpret their meaning. In order to do so, the program must find and identify patterns in the data–preferably in an efficient manner. Using a computer program (algorithm) to find patterns in structured data is referred to as machine learning [4].

There are two broad categories of machine learning: unsupervised and supervised [4, 33]. Unsupervised learning discovers an inherent structure in the data without guidance or user interaction. In contrast, supervised learning develops a predictive model through guidance from a labeled dataset. The model is developed in two stages. During the learning stage, a computer algorithm infers a predictive model from the labeled examples. During the testing stage, the model is evaluated on its ability to complete the task.

One could hand program a predictive model by using a series of if-then rules. For example, one rule for finding a lymphoblast is "if the cell is a lymphocyte, then check if it has nucleoli." However, programming explicit rules for every possible combination of variables in a complex task like microscopy is difficult; the code would be brittle to change and difficult to test. Fortunately,

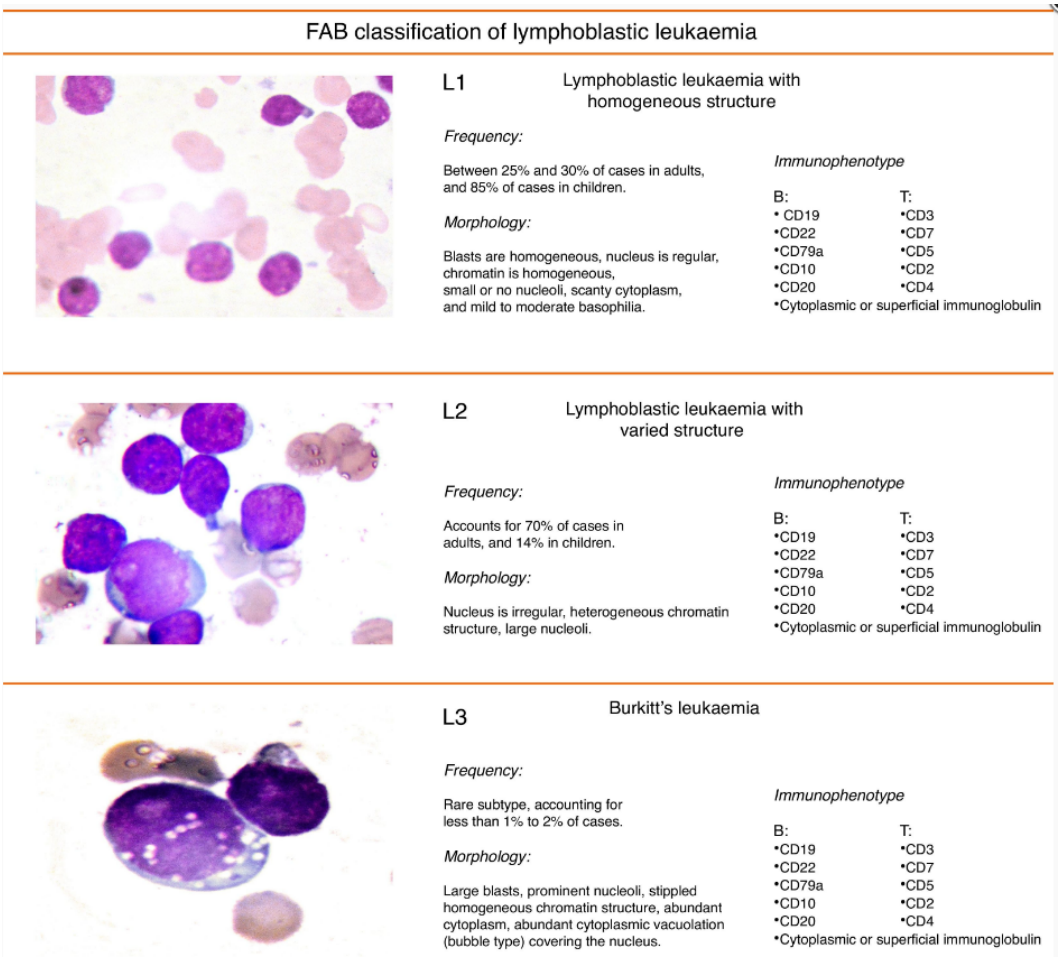


Figure 1.3: French-American-British classification of lymphoblastic leukemia for subtyping ALL.

this is unnecessary because supervised machine learning infers a predictive model from a dataset in the absence of strict programming rules [33]. This means that the machine learning algorithm can handle data with many variables because programming rules for every combination of variables is unnecessary. The number of variables is also referred to as the dimensionality. During the course of supervised learning, the algorithm feeds each example to the model in the form of an input vector of features, and the model returns a prediction in the form of an output vector. The algorithm then compares the output with the example's label. The label is also known as the ground truth. If the prediction is off from the ground truth, the algorithm tunes the model's parameters to make better predictions.

Supervised machine learning algorithms have three components: representation, evaluation, and optimization [5]. The representation is the class of algorithm or model that is programmed into the computer. Examples of representations include K-nearest neighbor, support vector machine, and neural network. The evaluation is the measure of the model's performance. Examples include accuracy, precision and recall, and squared error. The optimization is the strategy for tuning the model to get better performance. Optimizations include greedy search and gradient descent [5].

Supervised machine learning is widely applied to classification problems [4, 5]. In classification, the predictive model (classifier) maps an input vector to a single discrete value. An example is recognizing hand-written digits; the classifier's input is an image of a digit and its output is a label from 0-9. While learning, the classifier compares its output to the image's ground truth label and adjusts parameters when there are discrepancies. Once the classifier achieves an acceptable level of accuracy, its performance is measured on a test set. The test set is disjoint from the training set and representative of the larger set of images the classifier will incur in the wild. By keeping distinct training and test sets, we get a better idea of how the classifier handles unseen data (generalizes) [2, 5].

Machine Learning in Cell Biology

Sommer and Gerlich wrote an introduction to applied machine learning in cell biology [33]. In this domain, the goal is to classify cells from microscopic images. The previously mentioned representations, evaluations, and optimizations still apply. Classifying cell images poses additional challenges for which a data processing pipeline is proposed in Figure 1.4. The pipeline includes data preprocessing, object detection, feature extraction, training, and classification. Here the first three steps are only briefly covered because they do not constitute machine learning. Training and classification

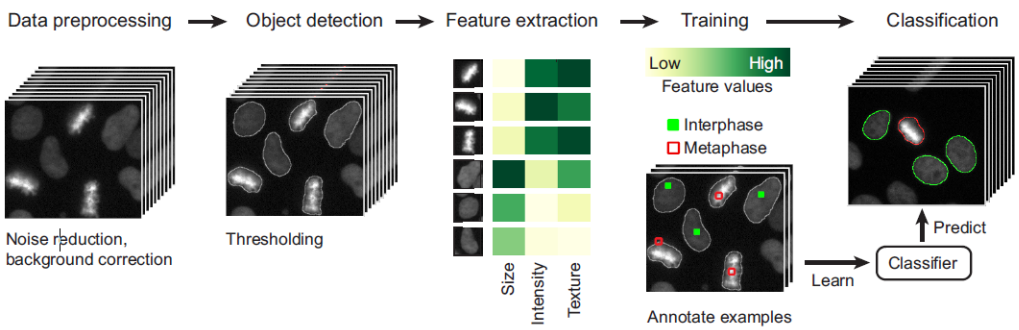


Figure 1.4: Image processing pipeline [33].

constitute machine learning and are covered in depth later. Explanations follow where appropriate.

Data preprocessing

Data preprocessing involves translating raw data into a form suitable for the model [4]. It is also a strategy for improving model performance by enhancing the signal of interest over background noise. An array of general preprocessing steps exist. They include data normalization, feature scaling, dimensionality reduction, and mean subtraction [12]. Other strategies are specific to the problem being addressed. In microscopic image analysis, this may include correcting for uneven stage illumination and smoothing filters for reducing microscopy artifacts [3]. Quality control is another form of data preprocessing. Examples include removing outliers or samples with missing values [4].

Object detection

Depending on the application, isolating individual cells may be necessary. Intensity thresholding and contour detection are two strategies for segmenting cells. Other approaches are specific to cell phenotypes and rely on fluorescent markers or separate machine learning algorithms called pixel classifiers [32].

Feature Extraction

After isolating cells, discriminatory features are extracted for input into the learning algorithm. Such features may include textures and contours. Deciding which features to include has great impact on the model's performance; including most or all features is not recommended because it increases the model's complexity, inhibits learning, and extends computational time [2, 33]. Feature selection may constitute a trial-and-error process using subsets of training data for cross validation.

Classification Models

Most classifiers share a common discriminative approach: during learning, a division is drawn through the dataset and used for distinguishing classes. This division is referred to as a decision boundary. The classifier may use a linear model to draw the decision boundary if the feature space is two-dimensional and contains classes divisible by a straight line. For more complicated two-dimensional class distributions, or if the feature space is three-dimensional, a linear model may still apply. In this case, the decision boundaries are curves or hyperplanes respectively. To draw a complex decision boundary, higher-order features are generated from the ones provided. Features can be squared, cubed, or multiplied in combinations to achieve this. In so doing, nonlinear terms are introduced to the model, but the model's linearity is determined by its parameters, which remain unchanged.

Despite this added flexibility, linear models are not ideal for classification; a linear model's continuous output is undefined for a classification problem's discrete values. Adding thresholding rules resolves some inconsistency, but these rules break with training data that is spread out in the feature space. In image classification, every pixel in the image may serve as an input feature to the model. This means for a small image with a height and width of 20 pixels, the input has 400 dimensions. Larger pictures have dimensions in the thousands. To address the complexity of this higher dimensional problem, non-linear models are used.

Sommer and Gerlich consider state-of-the art models capable of non-linear classification of cell images in high-throughput cell biology and bioimage informatics [33]. These models include support vector machine [10], adaptive boosting [6], and random forest [32]. K-nearest neighbor (KNN) and neural network (NN) are also models applied to WBC classification[18, 29, 35, 38]. Knowing which model to use is described as a "black art" learned from experience or trial-and-error rather than textbooks [5]. To better understand the models that appear later in this work, brief descriptions for KNN, SVM, and NN follow.

K-Nearest Neighbor (KNN)

Of the image classifiers, KNN is the simplest to understand and implement. This is due to the fact that a KNN does no training. Instead, a test image is compared to all example images in the labeled dataset. The images that are most similar to the test image serve as a simple majority vote towards the test image's predicted label. To compare the test image to an example image, each is unrolled into a flat vector of pixel values, I_t and I_e respectively. Then the difference $d(I_t, I_e)$ between the

images is calculated pixel-by-pixel and summed:

$$d(I_t, I_e) = \sum_p |I_t^p - I_e^p|$$

The k example images of minimum difference from the test image are the test image's nearest neighbors. KNN is most suitable for low-dimensional problems. In image classification, KNN performance serves as a baseline for other classifiers.

Support Vector Machine (SVM)

SVMs are widely used in academia and industry because they have clean implementations suitable for learning complex, non-linear decision boundaries. Their clean reputation is due to the fact that an SVM uses a nonlinear mapping function that transforms input data to a higher-dimensional feature space in a computationally efficient manner. In this feature space, the SVM draws a decision boundary of maximum margin. SVMs have parameters that are learned during training. SVM implementations are readily available in scientific software packages.

Neural Network (NN)

NNs are another non-linear model loosely analogous to biological neurons. An NN consists of three types of layers: an input layer for receiving data, one or more hidden layers for transforming the data, and an output layer for delivering the classification. Each layer is composed of neurons (nodes) that are fully connected with the preceding layer. Each node in a hidden layer contains a set of learnable parameters. These parameters are used to perform a dot product on the input it receives from the previous layer. The dot product is followed by a non-linear function at the discretion of the model's architect. The output layer receives the transformed data and produces classification scores for predicting the label.

The NN represents a differentiable function, which is significant for model training. During training, the NN's classification error is measured. By taking this error and calculating the partial derivative of each node in the NN from the output layer through the hidden layers, the model can adjust the parameters accordingly to reduce future errors. This process of calculating derivatives from the output layer through the hidden layers is referred to as backpropagation. Like SVMs, NN implementations are available in software packages, but are also developed according to the architect's needs.

Chapter 2

Related Work

As the data processing pipeline suggests, classifying cells is a series of problems leading up to the training and testing of a machine learning model. Prior works on WBC segmentation and classification used a variety of approaches with performance rates approaching that of human experts.

Segmentation

WBC segmentation involves separating the cell from its background, often through identification of the cell's cytoplasm and nucleus [27]. This is readily achieved through image processing functions available in math software. Converting the image to a different color space, contrast stretching, thresholding, clusterization, watershedding, and morphological filtering are some steps mentioned the in literature [19, 24, 26, 30, 31, 35]. These steps may produce a binary image of white WBC components for masking the original color image [19, 25, 31].

In multiple works, WBC segmentation exploited morphological observations from gray-scale microscopic images[24, 30]. Since WBCs stain darker than other blood components, contrast stretching was performed to enhance their nuclei. Then a morphological filter was derived by averaging the WBC diameters. Applying this morphological filter further enhanced WBC nuclei while reducing smaller blood components [24]. These steps produced sub-images of fixed dimension containing centrally located WBCs with high accuracy [30]. Putzu et al. improved on this strategy by inserting additional color-space conversion and thresholding steps. In addition, grouped WBCs were separated through watershed segmentation yielding 92% accuracy [25]. Scotti described "robust" methods for segmenting cells using L*a*b color space and fuzzy k-means clusterization to also report an accuracy of 92% [31].

In contrast to these works, Su et al. performed PCA on WBC pixels in the HSI color space to derive ellipsoidal equations. These equations were then used to discern which pixels belonged to WBCs. Morphological operators were applied to the resulting images to remove noise and fill holes. This approach yielded segmentation sensitivity and specificity rates above 97% for most cell types [35].

Feature Selection

Once a WBC is segmented from its background, distinctive features are extracted and fed to the machine learning model. As previously mentioned, feature selection is an important factor in model performance. Cross validation is often performed to find which combinations of features yield the best results. WBC features typically come from the cell's nucleus, and include geometric, textural, and color properties [35]. Geometric features include the cell's length, area, and diameter. Color features include color distribution and histograms. Textural features include contrast, entropy, and homogeneity. Many more features can be measured or computed using the ones listed here.

The type and number of features extracted varies in the literature. Rezatofighi et al. used 10 textural features to classify four types of WBCs [29]. Piuri and Scotti extracted 23 features, of which most were geometric, in their effort to classify five types of WBCs [24]. Su et al. selected a combination of 20 geometric, color, and texture features of which most were texture for WBC classification [35]. Mohapatra et al. selected 44 features in roughly equal proportions amongst the same geometric, color, and texture categories for binary classification of ALL [20].

These works indicate no standard exists for feature selection; some authors use more features for fewer classifications while others extract a single category of features to perform more classifications. It is left to the experimenter to identify which features yield the best results for their models.

Model Selection

As with feature selection, the literature supports varying approaches to model selection. Some reports optimize a single type of model. Theera-Umpon and Gader trained neural networks for counting and classifying WBCs with accuracies in the low 80% [38]. Kazemi et al. used SVMs to classify acute myelogenous leukemia with 96% accuracy [13].

Other reports test and evaluate several types of models which, amongst others, may include KNN, SVM, and NN. Piuri and Scotti trained KNN and NN models for classifying WBCs. The

best model was a feed-forward NN (FF-NN) with 92% accuracy [24]. Rezatofighi et al. compared the performance of an NN and SVM. The SVM yielded the best overall accuracy of 96% [29]. In contrast, Su et al. also trained NN and SVM models. The NN had the best overall accuracy of 99% [35].

Automating ALL Classification

Few studies use microscopic images for classifying blood disorders [27]. Of the authors previously mentioned, Scotti [30] and Mohapatra [20] proposed systems for automatic ALL classification.

Scotti selected lymphocytes from gray-scale images using a five-step process that included canny-based filters and morphological operators. From the resulting binary image, cytoplasm and nucleus features were selected using threshold segmentation. The selection process produced six sub-images from which 21 geometric and 2 color features were extracted through measurement and computation. KNN, linear Bayes Normal, and FF-NN classifiers were tested. Of these classifiers, the FF-NN yielded the best performance with a mean error of 0.0133 [30].

Mohapatra et al. selected lymphocytes by converting the image color space from RGB to L*a*b*. The a* and b* components were then fed to a shadowed C-means clustering algorithm that distinguished each image pixel into background, cytoplasm, and nucleus regions. Cytoplasm and nucleus features were extracted through measurement and computation. In all, 44 features were extracted: 17 geometric, 15 texture, and 12 color. The 44 features were further narrowed using an independent-sample "t" test, which found that 32 were statistically significant. Naive Bayesian, KNN, NN, and SVM classifiers were tested. In addition, an ensemble of classifiers (EOC) consisting of a KNN, NN, and SVM were tested using simple majority voting. It was reported that the EOC outperformed the individual ALL classifiers with an average accuracy of 94.73% [20].

While the literature reports gains in automating WBC microscopy, their hand-crafted, domain-specific approaches to segmentation and feature extraction are time-consuming and do not generalize to broader microscopic classification problems. Furthermore, there are no standards for segmentation, feature extraction, and classification. Within each step, approaches could be mixed and matched with any number or type of classifier, resulting in a never-ending source of work.

Underrepresented in cell biology is a classifier designed specifically for image classification that provides some focus and refinement to research efforts. As previously mentioned, Sommer and Gerlich list state-of-the-art classifiers for cell biology known for decades (SVM, adaptive boosting, and random forest) [33]. However, outside of cell biology exists an actively researched classifier

specialized for images. Recent work reports significant improvements in image classification over a range of scale. This specialized classifier is called the convolutional neural network (CNN).

Convolutional Neural Networks

CNNs are described as an NN with at least one convolutional layer [7]. As such, a CNN shares a number of similarities with an NN. Both models are structured into layers that receive input, transform the data, and deliver a classification. Like an NN, a CNN has learnable parameters spread over a number of layers. Furthermore, a CNN is a differentiable function that undergoes backpropogation during learning to tune the model's parameters.

A key difference is that the CNN is designed to treat an image as a three-dimensional volume rather than a one-dimensional feature vector; an image has height and width, but it also has depth when considering each component of its color space. For example, images in the RGB color space have a depth of three–one for each color channel. Treating an image as a volume allows the CNN to learn features that are related spatially. A CNN also has several specialized layers that transform the image's volume in a variety of ways. The namesake convolutional layer performs much of the computation that goes into classifying an image. Within a convolutional layer is a series of filters that slide, or convolve, over an image volume. If the CNN is well trained, these filters identify shapes, textures, colors, and other features in the image. This is one benefit of a CNN; important features in the image dataset are learned by the model [12].

CNNs have success in practical applications of large scale [7]. The earliest of these successes was reported in 1998 by Lecun et al. for handwriting recognition [17]. Most recently, the availability of enormous datasets on the Internet and open source movement in machine learning technologies have propelled CNNs forward in large-scale visual recognition challenges. In these challenges, researchers develop CNNs for classifying images into $10^2 - 10^3$ classes using benchmark datasets ranging in sizes of $10^3 - 10^5$ images. The images themselves typically represent common objects and animals. The most successful model architectures are $10^1 - 10^2$ layers deep and train for days or weeks. In 2014, a CNN developed by Google trained with the ILSVRC dataset to achieve a top-five error of 6.67% [36].

Along with common object datasets, fine-grained benchmark datasets exist. Fine-grained refers to the task of distinguishing classes that are very similar, such as dog and bird species [14, 39]. Fine-grained datasets are typically one to two orders of magnitude smaller than common object datasets because they require expert labeling that is expensive to obtain [15]. Such small datasets

are insufficient for training deep CNNs without overfitting [40]. To combat this, one approach uses a deep CNN pre-trained on a common object dataset. This type of off-the-shelf transfer learning had astounding results on fine-grained image classification (FGIC) of birds and flowers, beating highly-tuned state-of-the-art classifiers [28]. In another example, an off-the-shelf CNN was further trained on a noisy dataset consisting of publicly-available online image search results. Despite the absence of expertly labeled data, the CNN obtained state-of-the-art accuracies for bird and dog speciation [15]. As of this writing, one report on WBC classification uses an off-the-shelf CNN. Using low resolution gray-scale images and the model reported by Lecun et al., the CNN out performed SVMs for five types of WBCs [9].

In light of growing evidence, CNNs should be a consideration for all image based classification. Even when the data set is small and noisy, and the class distinctions fine-grained, CNNs outperform specialized state-of-the art models. ALL classification could be viewed as FGIC because of the similarities they share with lymphocytes and the relative scarcity of cytological images. This work examines the utility of using a CNN for FGIC of ALL.

Chapter 3

Background: Classification Models

The three supervised machine learning components defined in the introduction (representation, evaluation, and optimization) are now formalized with mathematical definitions as they apply to classification. Synonymous to the representation is a **score function** that takes data as input and maps it to a class score. An evaluation is a **loss function** that takes the class score and compares it to a ground truth label. The loss function returns a measure of error called the loss. The goal of classifier training (optimization or learning) is to learn a score function that minimizes loss.

The score function's input is the training data set $x_i \in R^D$ where x_i is training example i of dimension D. For image classification, the image is typically unrolled into a long, flat feature vector. The vector's dimensionality is the number of pixels in the image. Each example image has a corresponding label $y_i \in 1 \dots K$ where K is the number of class labels. The score function that maps an image to a class score is thus:

$$f : R^D \mapsto R^K.$$

Karpathy provides an in-depth guide to image classifiers along with current best practices in this rapidly evolving field [12]. Goodfellow et al. and Nielsen provide formal model representations and proofs [7, 22]. What follows are details of the models and methods used in this work.

Linear Model

A linear model serves as an entry point for understanding other classifiers. A simple linear model has the score function:

$$f(x_i, W, b) = Wx_i + b$$

W and b are the model's learnable parameters. During training, these parameters are tuned to make an optimized score function. W is a matrix of weights with K rows and D columns. Each row contains D parameters for producing a weighted sum for one of K classes. b is a vector of bias terms with dimension D. Bias terms allow the model to learn decision boundaries that may not cross the origin.

Figure 3.1 is a cartoon example of a linear model that receives an image and returns three class scores. More specifically, the input is an image flattened into a feature vector of raw pixel values and the output is the weighted sums for three classes. The scores reflect the current model's knowledge. Since the model received a cat image but returned the highest score for a dog, it needs training.

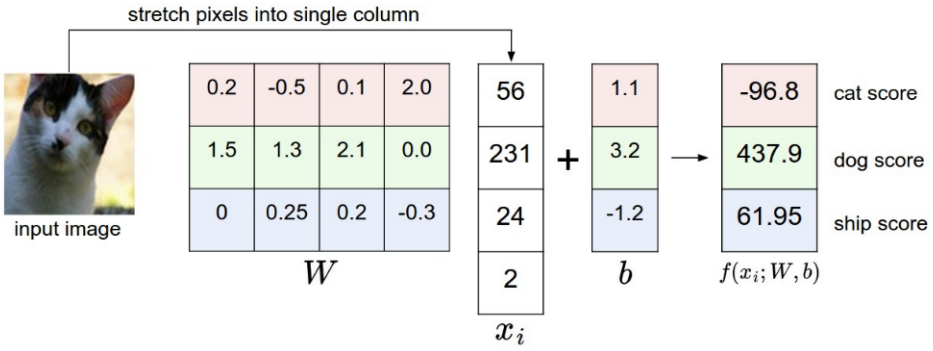


Figure 3.1: Graphic of a linear model mapping an image to three class scores [12].

In practice, the linear model is simplified by combining the bias terms with the weights to form a single matrix W. To facilitate this, the feature vector x_i is extended by one dimension containing the constant 1. Thus the linear model simplifies to matrix multiplication (dot product):

$$f(x_i, W) = Wx_i$$

Figure 3.2 visualizes this simplification.

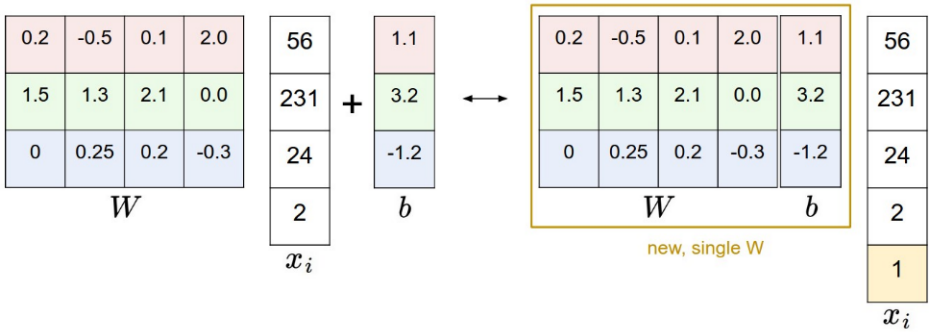


Figure 3.2: The simplified linear model after combining bias terms and weights into W while extending x_i by one dimension [12].

SVM Loss Function

During training, the loss function calculates the model's error. Another way to think of error is how well the model's predictions match the labels of the training examples. One type of error borrows from an SVM's decision boundary that separates classes by a specified margin. It is called the **Multiclass Support Vector Machine loss** (SVM loss):

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

L_i is the loss associated with classifying example i. It is the sum of errors for class scores s_j not associated with the correct label y_i that differ from the correct class score s_{y_i} within a fixed margin Δ. In other words, the SVM loss accumulates when class scores fall within a margin of the correct class score. If the scores fall below the margin, the loss is thresholded at zero. By minimizing this loss, the model ensures that the score for the correct class is marginally higher than the other class scores. Figure 3.3 visualizes this concept.

Figure 3.3: Visualizing the SVM loss margin. Class scores that fall within the red delta region contribute to the loss. Class scores that fall below the delta contribute zero loss. During training, the loss is minimized so that the correct class score is marginally higher than all other class scores [12].

For a linear model, the SVM loss function is defined:

$$L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

where w_j is the j-th row of W containing weights for a particular class. Taking the dot product between w_j and feature vector x_i returns the class score s_j. The standard value for Δ is 1.

Cross Entropy Loss Function

Another way to calculate a model's error is through the **cross-entropy loss**:

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

Whereas the class scores in the SVM loss function were arbitrarily valued, the class scores in the

cross-entropy loss are normalized log probabilities. $f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$ is called the softmax function. It converts class scores for a single example into probabilities that sum to one. Figure 3.4 compares the cross-entropy loss and SVM loss functions.

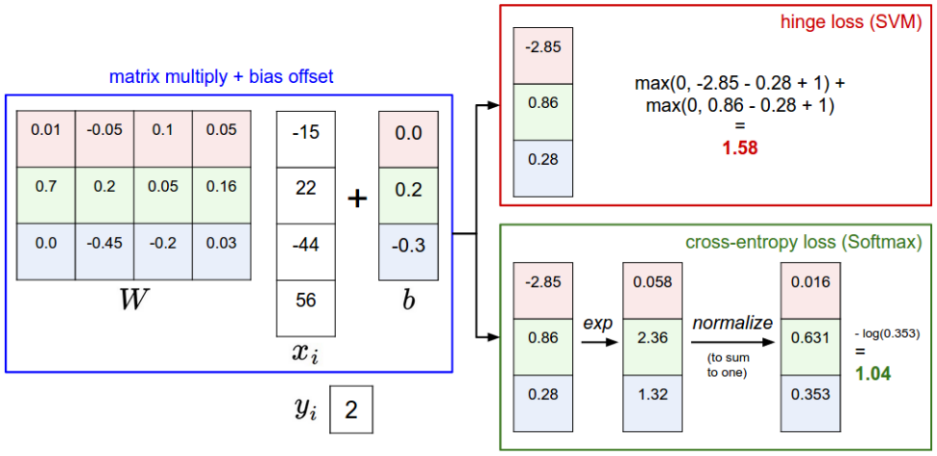


Figure 3.4: Comparison of SVM and cross entropy loss functions. For the given example x_i, the correct class label y_i is 2. The parameters that calculate the class 2 score are shaded blue [12].

In most scenarios, the cross-entropy loss and SVM loss are comparable. Some architects prefer the cross-entropy's probabilistic interpretation, while others prefer the SVM for the simplicity that comes with zero-thresholding. In this work, the cross-entropy loss was selected.

Classifier Loss Function

The SVM and cross-entropy loss functions calculate the loss for a single training example. The average loss of the training examples is the loss for the model. In addition, a regularization loss is introduced. Regularization is a strategy for preventing the model from overfitting the data; it penalizes the model for learning parameters of large magnitude. Thus the loss function for the model is:

$$L = \frac{1}{N} \sum_i L_i + \lambda R(W)$$

where $\frac{1}{N} \sum_i L_i$ is the model's data loss averaged over all N training examples and $\lambda R(W)$ is the regularization loss. λ is a value controlling the regularization strength. Specifically, an L2 norm regularization sums the squared elements of the parameters in W:

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

Figure 3.5 depicts how information flows through a generic classifier to the loss function.

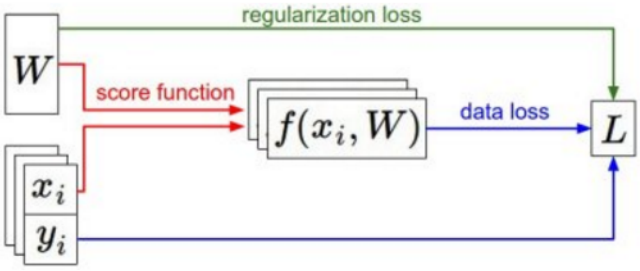


Figure 3.5: The flow of information in a classifier [12].

Optimization

The loss represents the classifier's current state of knowledge. A classifier improves its state of knowledge through optimization. More specifically, optimization is the act of finding weight parameters W that minimize the loss function.

A visualization helps with understanding optimization. Imagine the loss function as a topographical map in a three-dimensional feature space (Figure 3.6). The x- and y-axis are model parameters W and the z-axis is the elevation (loss). High elevations result from bad combinations of parameters. Low elevations result from good combinations of parameters. Since a good combination of parameters is unknown at the start of optimization, they are randomly initialized. This is equivalent to choosing a random start position on the map. During optimization, small steps are taken down the terrain to reach lower elevations. The act of taking a step is equivalent to adjusting the parameters W. Steps vary in direction, but by maintaining an awareness of the local gradient at any particular location, the overall direction is downward towards the minimum elevation.

The type of optimization described above is known as **gradient descent**. It is an iterative process in which the model parameters W are improved in a loop. By calculating the gradient of the loss function for any given set of parameters, the parameters can be updated to reduce loss. Note that the example contains two parameters while image classification models require many more parameters. Fortunately, gradient descent generalizes to models of arbitrary dimensionality. Gradient descent does not guarantee that the global minimum is reached, but in practice reaching a local minimum can yield good results. The gradient can be calculated numerically with finite difference approximation or analytically with calculus. With calculus, the gradient is found by taking the derivative of the loss function. This approach is faster to compute and thus preferred for gradient descent.

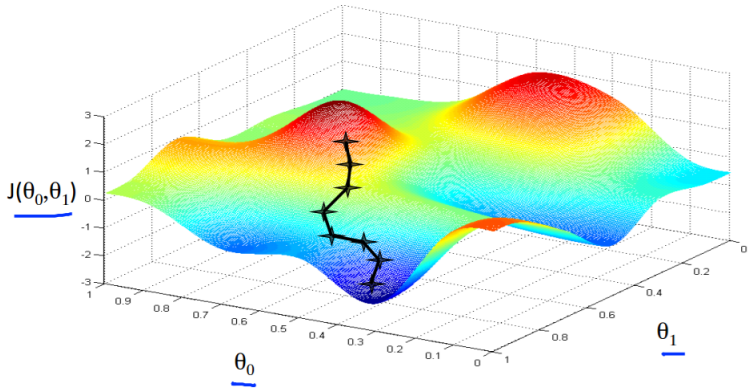


Figure 3.6: Toy illustration of gradient descent. The model consists of two learnable parameters θ_0 and θ_1 along the x- and y-axis. The loss function $J(\theta_0, \theta_1)$ is along the z-axis. The descent begins with random placement on the map and ends at the low elevation through a series of small steps [21].

Neural Network

Although useful for introducing concepts, a linear model is not suitable for image classification because of the problem's high dimensionality. At this point, the discussion turns to nonlinear classifiers. The NN is a nonlinear classifier found in image classification literature. The details of its score function, loss function, and gradient descent are covered here.

An NN has a layered structure that includes an input layer, one or more hidden layers, and an output layer. NNs are often described by the number of hidden layers; a network with one hidden layer is referred to as a one-layer NN. Figure 3.7 depicts one-layer and two-layer NNs. The networks are layered acyclic graphs where nodes are fully connected with preceding layers.

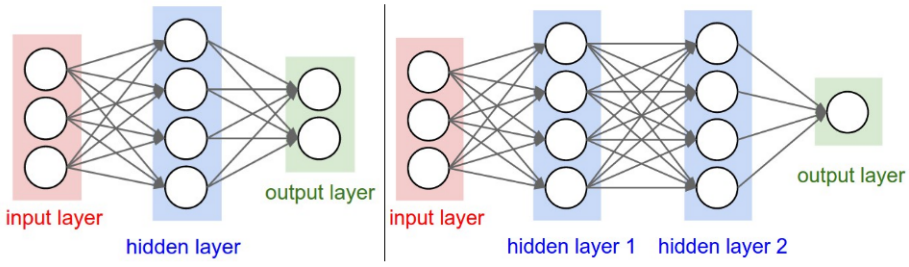


Figure 3.7: NN with one hidden layer (left) and NN with multiple hidden layers (right) [12].

Succinctly, an NN's score function is "a sequence of linear mappings with interwoven non-linearities" [12]. Data enters the model through the input layer. Each node in the input layer corresponds to one dimension in the feature vector. The input layer then passes these features to the first hidden layer. Each node in a hidden layer contains an activation function. The activa-

tion function takes a vector of inputs from the preceding layer and with its parameters calculates a weighted sum. The weighted sum constitutes the linear mapping previously described in the linear model. Next the weighted sum passes through a non-linear function and the result is propagated to the next layer. Each node in the output layer is a class score for a single example. Figure 3.8 shows details of a hidden layer node (activation unit).

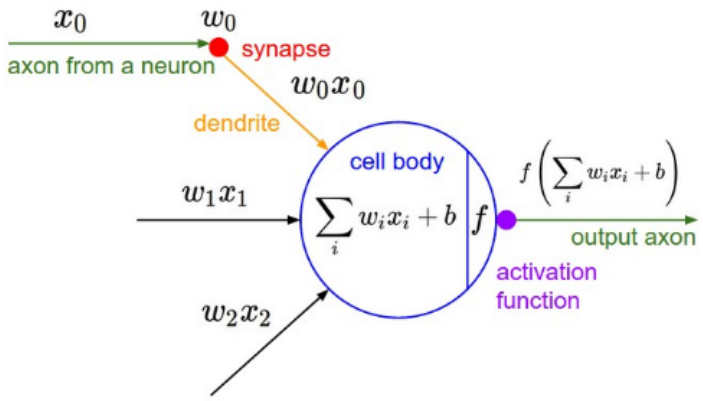


Figure 3.8: Close-up of single node in the hidden layer of an NN. The node receives a vector $[x_0, x_1, x_2]$ and calculates a weighted sum using its parameters w_0, w_1, w_2 and b. The weighted sum is then input to a non-linear function $f(\sum_i w_ix_i+b)$. Output is directed to the next layer. Analogous terms for a biological neuron are included [12].

Without the non-linear activation functions, the NN would be an extended linear classifier. The non-linear functions allow the NN to learn more complex decision boundaries and are described as giving the model "wiggle" [12]. Possible non-linear functions include sigmoid, tanh, and rectified linear unit (ReLU). In practice, ReLU is found to have several benefits, including its simplicity and acceleration of gradient descent. The ReLU function takes the form $f(x) = \max(0, x)$, where x is the weighted sum of the inputs. It thresholds negative values at zero, ensuring that model training maximizes correct class scores rather than minimizing incorrect class scores.

Neural Network Loss and Optimization

Nodes learn to recognize features in the input by tuning their parameters through gradient descent. First, an example image is sent through the NN in a process known as **forward propagation**. If a node's activation function learns to recognize some linear region of the input, a signal is propagated to the next layer. Otherwise, a zero is propagated (if using ReLU activation function). The process repeats for each hidden layer until class scores are computed in the output layer. Finally, the class scores pass from the output layer to a loss function (e.g. SVM or cross-entropy) and the error is

calculated.

Taking the derivative of the loss function approximates the gradient. This gradient is propagated backwards through the NN to the first hidden layer. Along the way, the partial derivative with respect to a given node's activation function is calculated. This partial derivative appropriates the node's output to the model's loss. In other words, each node learns what effect it has on the output of the model. The process of sending the gradient from the end of the NN to the first hidden layer and calculating partial derivatives along the way is called **backpropagation**. The specifics of backpropagation involve the calculus chain rule and are left out of this discussion. The intuition of backpropagation is that each node learns how much effect it has on the model's output. Once the node learns this, it can increase or decrease its parameters to help minimize the model's loss.

The gradient for a particular node's activation function can be more formally expressed:

$$\delta_j^{(l)} = \text{ gradient of activation unit } a_j^{(l)} \text{(unit j in layer l)}$$

$$\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} L_i \text{ for (j >= 0)}$$

where $\frac{\partial}{\partial z_j^{(l)}}$ is the partial derivative, $z_j^{(l)}$ is the node's weighted sum of inputs, and L_i is the loss for example i. Changing the z values will change the score function's output, and thus the loss function's output. z values are changed by adjusting the node's weight parameters $w_j^{(l)}$.

The parameter update for any given weight inside an activation unit takes the form

$$w_{ji}^{(l)} := w_{ji}^{(l)} - \alpha \frac{\partial}{\partial w_{ji}^{(l)}} L(W)$$

where $w_{ji}^{(l)}$ is the i^{th} weight of the j^{th} node in network layer (l), and $\frac{\partial}{\partial w_{ji}^{(l)}} L(W)$ is the partial derivative of the loss function with respect to the model's weights W. The update occurs in the negative direction of the gradient proportional to the learning rate α.

Each round of parameter updates requires passing a batch of examples through the NN, averaging their loss, and accumulating gradients via backpropogation. These steps constitute one iteration of **stochastic gradient descent** (SGD). In practice, SGD occurs over many iterations. The model's progress is observed by plotting loss over iteration number. This plot is known as a **learning curve**. Learning curves are useful in determining if a model is learning correctly. If the learning curve's trajectory is unsatisfactory, model implementation details are reconsidered. Another learning curve is the model's accuracy (correct classification percentage) over time. During training, plotting the

accuracies for both training and validation sets can help determine if bias or variance are present.

Neural Network Architecture

The NN models depicted in Figure 3.7 are two simple architectures. The number of nodes in the input layer equals the dimensionality of the input feature vector, and the number of nodes in the output layer equals the number of class labels. For many scenarios, the number of nodes in a hidden layer is comparable to or slightly more than the input layer.

However, if an NN receives whole images as input, the number of hidden layer nodes is far less than the input layer because of computer memory limitations. For example, an NN performing classification on medium-sized color images of dimension 224x224x3 (224 pixel width, 224 pixel height, 3 color channels) would have an input layer containing 224*224*3 = 150528 nodes. Since each node in a hidden layer is fully connected to the previous layer, the first hidden layer nodes would each contain 150,528 weights. If the hidden layer nodes equaled the number of input layer nodes, the number of learnable weights in the first hidden layer would be $150,528^2$. This huge number of parameters would consume a computer's memory. To resolve this, the number of hidden layer nodes is limited to several orders of magnitude below the input layer's.

The number of hidden layer nodes is a model **hyperparameter**. A hyperparameter is a feature of the model that cannot be learned by the model itself and is instead selected by the architect. Another hyperparameter is the number of hidden layers. Generally, an NN begins with one hidden layer and more are added if the model is underperforming. Selecting hyperparameters is done through a process called **validation** (Figure 3.9). First, training data is split into several folds. Second, NN architectures with varying hyperparameters are trained on the data folds. Third, each architecture is tested on a reserved data fold called the validation fold. The validation fold acts as psuedo test data. Finally, the architecture with the best performance on the validation fold is selected to run on the test data. Only the performance on the test data is reported. By following these steps, we obtain a better idea of how the model performs on unseen data.

Neural Network Implementation Notes

In addition to selecting a network architecture, preprocessing image data and initializing weights are important considerations for model performance.

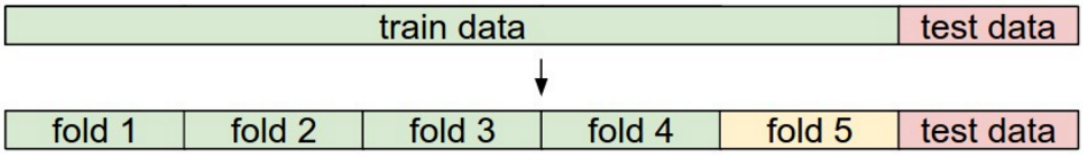


Figure 3.9: Validation is performed for hyperparameter selection. It involves splitting training data into folds (1-5), training models with varying hyperparameters on the training folds (1-4), and testing them on the validation fold (5). The best performing model is selected for evaluation on the test data [12].

Image Preprocessing

As stated in the introduction, data preprocessing improves model performance. With image classification, mean image subtraction is often performed: the average of each pixel value in the training set is subtracted from every image pixel input to the model. This has the effect of zero-centering the data, which gives the model flexibility during gradient descent (Figure 3.10).

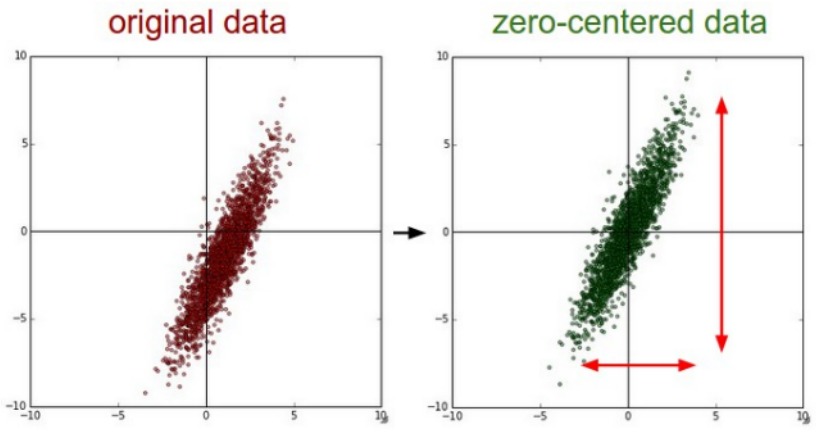


Figure 3.10: Distribution of a two-dimensional toy data set (left). Mean subtraction zero-centers the data (right) [12].

Weight Initialization

Before optimization begins, model parameters need to be randomly initialized. One method is to initialize each node's parameters with small random numbers. This is called "symmetry breaking" because it forces the node's activation functions to produce unique output from the very beginning. These unique outputs ultimately lead to unique gradient updates, which allows the nodes to learn distinct features. If instead every node's parameters were initialized to zero, every node would produce the same output and contribute equally to the gradient. This would cause every node to learn the same feature and the overall model would learn nothing.

Convolutional Neural Network

CNNs are similar to NNs in that they have layers consisting of nodes, use loss functions to measure their knowledge state, and learn parameters through gradient descent. A key difference is that a CNN is designed for images and treats them as a three-dimensional volume. In addition, CNN nodes are themselves arranged into three-dimensional volumes (Figure 3.11).

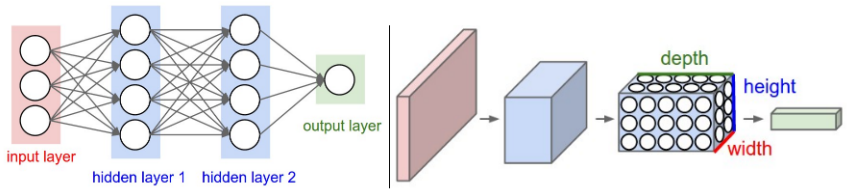


Figure 3.11: Neural network (left) and convolutional neural network (right) with equivalent layers color-coded [12].

Recall that NN hidden layer nodes are fully connected to the previous layer, which greatly increases the number of learnable weights and ultimately affects the architecture. In contrast, a CNN has layers of nodes that are no longer fully connected to previous layers. Instead, they have a localized focus on the input volume (Figure 3.12). By arranging nodes into three-dimensional volumes with a localized focus, a CNN can limit the number of parameters needed to perform classification.

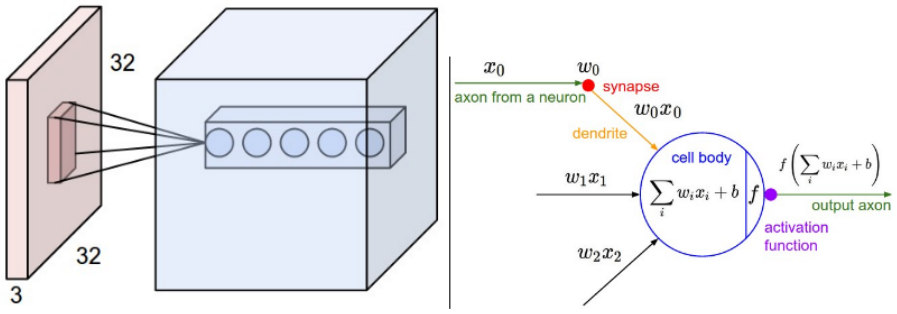


Figure 3.12: CNN nodes have a localized focus on the input but their internal functions are similar to NN nodes. The red volume is an input image and the blue volume is a network layer. The network layer is a 3D volume of nodes. The five aligned nodes have the same localized focus [12].

CNNs have a modular structure with a variety of layers. The **input layer** delivers the image with its original height and width as well as depth equal to the number of color channels. This is in contrast to an NN input layer that delivers the image as a flattened feature vector. The **convolutional** (CONV) **layer** is a three-dimensional layer with localized nodes. Nodes in this layer calculate the weighted sum of a small region of the input volume. The **ReLU** (RELU) **layer**

performs the same $max(0, x)$ operation as in the NN, providing a means for learning non-linear decision boundaries. A **pooling** (POOL) **layer** downsizes its input volume along spatial (height and width) dimensions, serving as another means for limiting the number of learnable parameters in the model. Near the end of the CNN are **fully connected layers** (FC), which are the same as a NN's fully connected hidden layers. By placing FC layers at the end of the network after POOL layers have downsized the input volume, the model can learn robust features without a burdensome amount of parameters. The **output layer** delivers a vector of class scores.

A CNNs learnable parameters are located in the CONV and FC layers and are tuned through gradient descent. The POOL and RELU layers do not have parameters because they perform fixed functions. CNNs undergo validation to select comparatively more hyperparameters than an NN. The dimensions of a CONV layer, the amount of downsizing in a POOL layer, and the number of FC layer nodes are just a few of the hyperparameters.

Convolutional Layer

CONV layer nodes have local connectivity on the input volume. As such, they have a spatial window known as a **receptive field**. The depth of the receptive field always equals the depth of the input volume, but the height and width are hyperparameters. Another feature of the CONV layer is that nodes at the same depth all share the same parameters. This groups the nodes into a "depth slice," which is also referred to as a filter. By sharing parameters in this way, the CNN greatly reduces the number of parameters stored in memory.

When an input volume reaches a CONV layer, the filters scan over the input, or convolve. As the filters convolve, they calculate dot products between their parameters and the image's pixel values. The movement of the filters is determined by their receptive field size and the length of their strides. Necessarily, the filters must convolve over the image evenly so as not to miss any detectable feature. One method for ensuring this is **zero-padding.** This involves adding a layer of pixels with zero values around the image.

The CONV layer transforms the input volume into an output volume. The output volume's spatial size is determined by the number of spatial strides the filters make over the input volume. The output volume's depth is determined by the number of filters. Figure 3.13 shows the transformation of an input volume into an output volume using two filters.

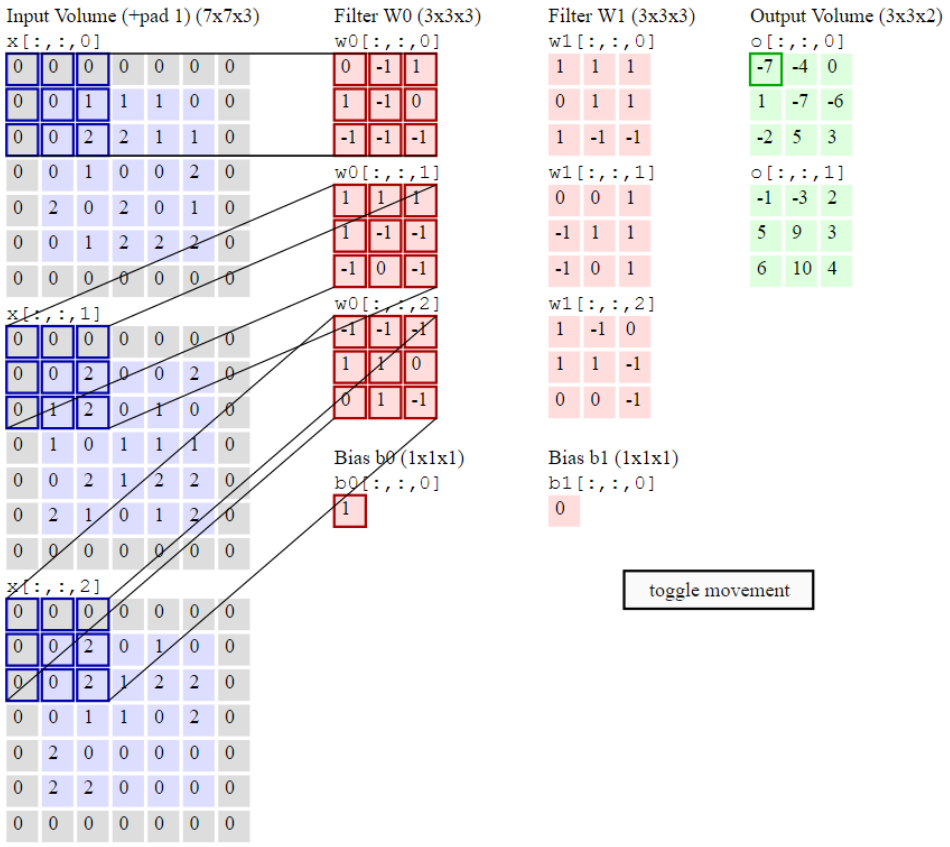


Figure 3.13: Detailed view of convolutional layer filters (red) performing dot products on the input volume (blue) to produce the output volume (green). The layer volumes were flattened depth-wise to better visualize the calculations. Specifically, the first filter, W0, is performing a dot product on the upper-left corner of the input volume along its entire depth. The input volume is zero-padded, allowing the filters to evenly convolve over the input [12].

Pooling Layer

Like CONV layers, POOL layers also transform their input volumes into smaller output volumes. In this way, pooling helps further reduce the number of learnable parameters in the model. However, POOL layers also have a destructive quality to them, so they always follow CONV and FC layers and never precede them. A POOL layer has a receptive field size that makes strides over the input volume similar to a CONV layer filter. At each stride, the POOL layer examines the pixels in its receptive field, and downsamples them by taking only the max value (Figure 3.14).

Convolutional Neural Network Architecture

Like NNs, CNNs have variable architectures. A general scheme is:

INPUT -> [CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC

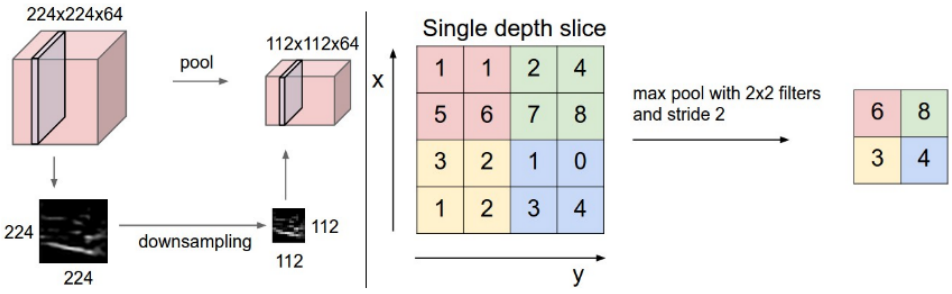


Figure 3.14: A pooling layer reduces the input volume size by downsampling pixels [12].

where the INPUT layer is followed by M repeating units of CONV -> RELU -> POOL layers and K repeating units of FC -> RELU layers. In addition, N repeating units of CONV -> RELU layers may precede an optional POOL layer. Architectures vary dramatically depending on the capabilities of the hardware, the size of the data set, and scale of the classification problem. A general heuristic is more layers are better but with the risk of overfitting the training data.

Convolutional Neural Network Loss and Optimization

A CNN has the same options as an NN when selecting a loss function; SVM and cross-entropy loss functions are comparable choices. In this work, the cross-entropy loss was selected.

Like an NN, a CNN represents a single differentiable function for which gradient descent is performed during optimization. A key difference is that nodes in a depth slice of a CONV layer share parameters. As such, they compute and combine their individual gradients to update the shared parameters. In addition, CNNs that contain POOL layers must keep track of which pixels they downsample in order to backpropagate the gradient. This is done by caching a matrix of switches.

Chapter 4

Methods

This chapter describes the image dataset and the actions performed for binary classification of abnormal lymphoblasts and normal WBCs using supervised machine learning models.

Dataset

The dataset used in this work is a public image database assembled by Labati et al. called ALL-IDB [16]. ALL-IDB is designed specifically for ALL classification and contains hundreds of white blood cell images. To obtain these images, peripheral blood samples were collected by researchers at the M. Tettamanti Research Center for Childhood Leukemia and Hematological Diseases in Monza, Italy. Microscopy slides were made following the Wright staining procedure. The slides were viewed with a bright field illuminated microscope at a resolution of 300-500μm. From the microscope, 109 color JPGs with a resolution of 2592x1944 pixels were captured using an attached Canon PowerShot G5 camera. From these images, 260 sub-images centered on individual WBCs were cropped to a resolution of 257x257 and saved as TIFs. The sub-images were expertly labeled as either normal WBCs (Y=0) or abnormal lymphoblasts (Y=1) in equal ratio. All Y=0 images came from healthy individuals and Y=1 images came from ALL patients. The images contain off-center RBCs and have non-uniform background illumination.

The ALL-IDB dataset was supplemented with images from CellaVision Proficiency Software for hematology laboratory training. 128 sub-images containing normal WBCs (Y=0) were cropped to a resolution of 257x257, converted to the RGB color space, and saved as PNGs (Table 4.1).

Table 4.1: Dataset Characteristics

	ALL-IDB	CellaVision	Total
Images:	260	128	388
Lymphoblasts:	130	0	130
Resolution:	257x257	257x257	

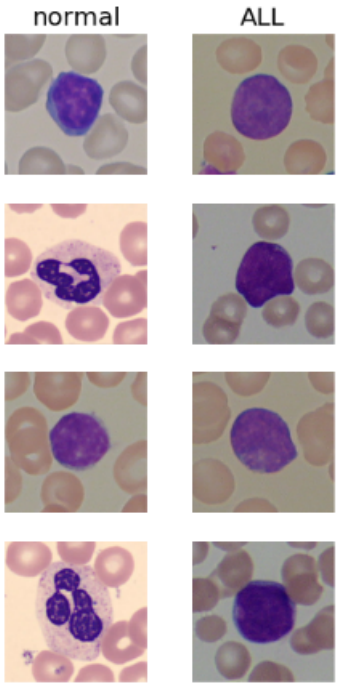


Figure 4.1: A random selection of images centered on normal WBCs (left) and abnormal lymphoblasts (right). The images also contain noise in the form of RBCs and non-uniform background illumination.

Data Preprocessing

Mean cell image subtraction is a common preprocessing technique to improve model training. The mean cell image was calculated from the training set and subtracted from every image in both training and test sets (Figure 4.2). The images use the RGB color space with pixel values falling in a range of 0-255. This small range eliminated the need for data normalization.

Feature Extraction

Typical feature extraction methods found in cell biology were bypassed in order to assess how well models learn features from whole color images and their raw pixel values.

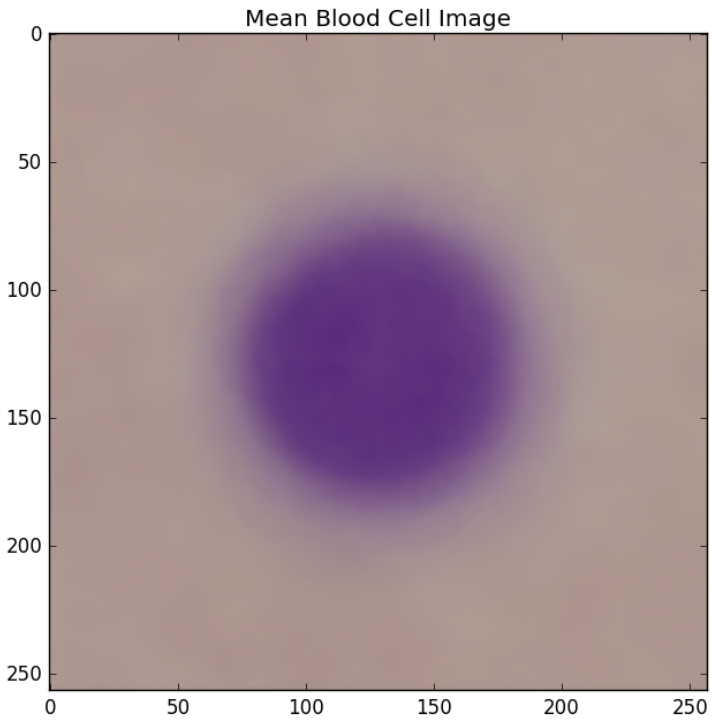


Figure 4.2: The mean cell image was calculated from the training data and subtracted from every image prior to classification.

Classifiers

Standard supervised machine learning models were selected for baseline comparisons. These included KNN and NN models. In addition, a CNN was developed to examine its ability to learn cell features for classification.

The ALL-IDB and CellaVision datasets were combined for model training and testing. While combining the datasets presumably enhanced the model's ability to generalize, it also resulted in a 1:3 ratio of lymphoblasts to normal WBCs. Imbalanced class ratios are a reality for fine-grained image classification in cell biology since by definition normal cells are more abundant. For each round of training, the data was randomly shuffled and split 60:20:20 into training, validation, and test sets respectively.

Model parameters were tuned using the training data and hyperparameters selected by evaluating performance on the validation data. Model training was monitored through learning curve

assessment. Once suitable hyperparameters were selected, models were trained and performance evaluated on the test data six times.

K-Nearest Neighbors

A KNN classifier was implemented using the euclidean distance between two vectors:

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

Cross validation was performed to determine k neighbors (Figure 4.3). Training data was split into five folds. For values of k=1...30, classifier accuracy was measured five times using each fold once for validation and the remaining folds for training. The highest average accuracy was observed for k=15.

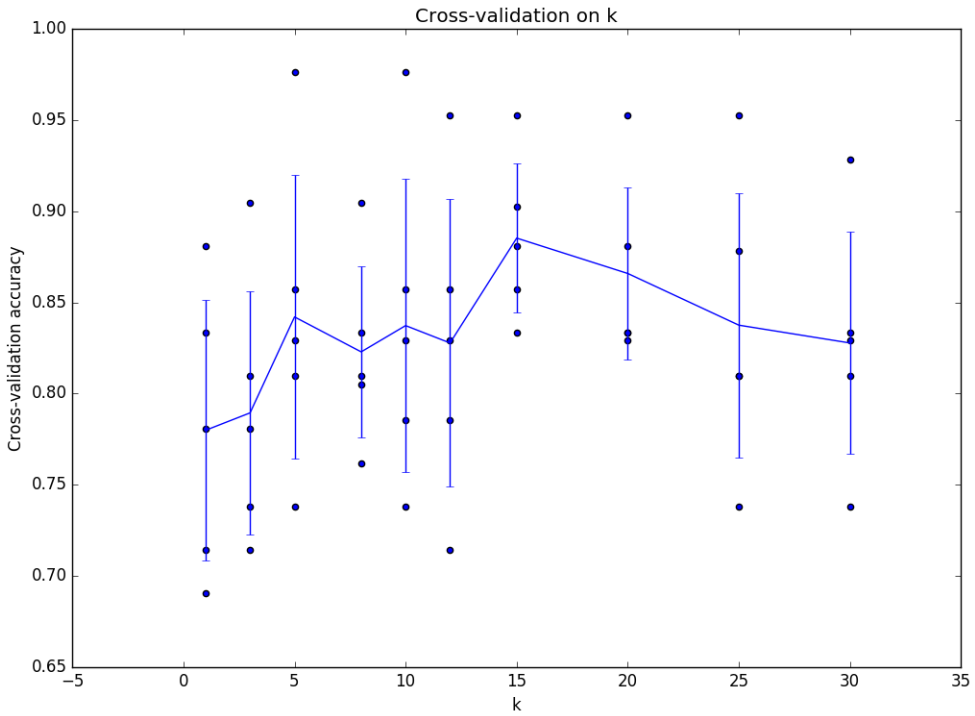


Figure 4.3: Cross-validation for determining k=15 neighbors.

Neural Networks

Two NNs were developed using ReLU for the nonlinearity function and L2 regularization. Random search validation [1] was performed for learning rate α and regularization strength λ hyperparameter selection. Cross-entropy and SGD were selected for the loss and optimization respectively.

NN-1 architecture consisted of two fully-connected hidden layers of 50 nodes each:

NN-1: [INPUT]->[FC]->[RELU]->[FC]->[LOSS]

NN-2 extended the NN-1 architecture with batch-normalization and dropout layers:

NN-2: [INPUT]->[FC]->[BNORM]->[RELU]->[DROP]->[FC]->[LOSS]

Batch normalization addresses internal covariate shift by normalizing layer inputs [11]. Dropout is a regularization strategy that randomly removes nodes from the network with a fixed probability during every forward propagation. It is the equivalent of training an ensemble of NNs within one NN. Dropout makes the NN more robust to unfortunate random weight initialization and enhances learning [34].

Convolutional Neural Network

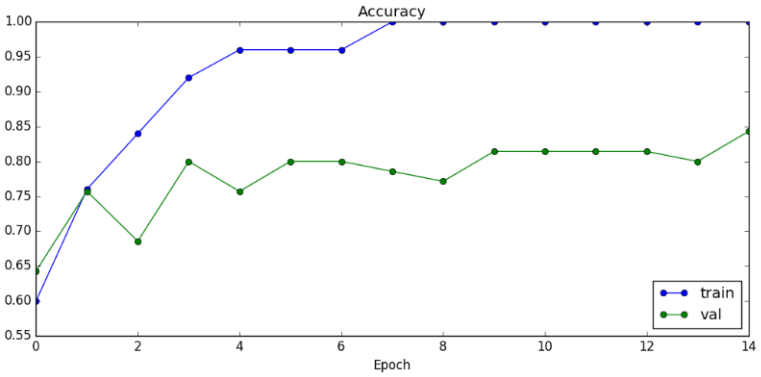
The two-layer NN-1 architecture was converted to a three-layer CNN by inserting convolutional, ReLU, and pooling layers:

CNN: [INPUT]->[CONV]->[RELU]->[POOL]->[FC]->[RELU]->[FC]->[LOSS]

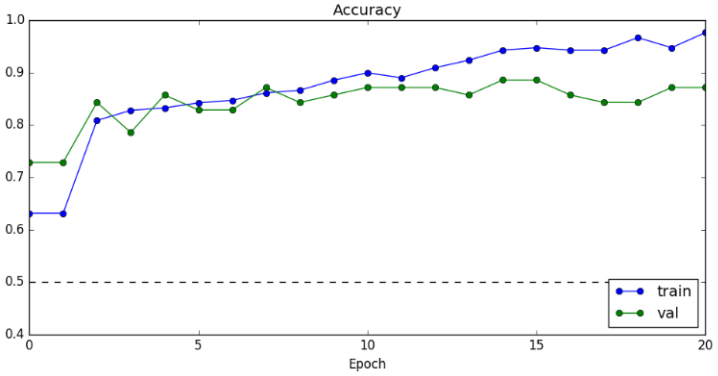
The CONV layer had a volume of 7x7x16 (16 filters sized 7x7 with stride 1). The POOL layer contained a 2x2 max pooling filter. The FC layers each contained 50 nodes. Images entering the CNN were spatially resized to 224x224x3 pixels and zero-padded to facilitate volume transformation in the CONV and POOL layers.

Training

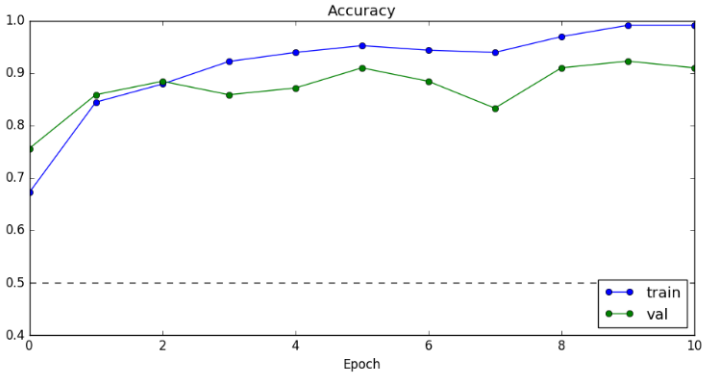
NN-1, NN-2, and CNN were trained using stochastic gradient descent with a batch size of 25 training images. The quality of training was determined through learning curves that plotted training and validation accuracies over rounds of gradient descent. Curves that leveled off (or reach 100%) indicated the model reached optimal training for the given hyperparameters and training data. A validation accuracy that approached the training accuracy suggested the absence of overfitting. (Figure 4.4).



(a) NN-1



(b) NN-2



(c) CNN

Figure 4.4: Examples of learning curves from the models trained with stochastic gradient descent.

Performance Analysis

In binary classification, examples are labeled as positive or negative. A confusion matrix counts the agreements and disagreements between a classifier's predictions and a dataset's labels (Table 4.2). This leads to four defined outcomes:

True Positive (TP) : Positive prediction matches positive label

True Negative (TN) : Negative prediction matches negative label

False Positive (FP) : Positive prediction contradicts negative label

False Negative (FN) : Negative prediction contradicts positive label

Table 4.2: Confusion Matrix

		Actual class	
		Positive	Negative
Predicted class	Positive	True positive (TP)	False positive (FP)
	Negative	False negative (FN)	True negative (TN)

The four outcomes in the confusion matrix are used to evaluate model performance. Accuracy is the fraction of matching outcomes out of all outcomes. When a dataset contains comparatively fewer positive examples than negative examples, precision, recall, and F1-score describe model performance better than accuracy. Precision measures the model's exactness in predicting positives. It answers the question: "Of all predicted positives, what fraction actually are positive?" Recall measures the model's completeness in predicting positive results. It answers the question: "Of all examples, what fraction were correctly predicted positive?" The F1-score combines precision and recall into one value for easier comparisons between models. It does so through the harmonic mean of precision and recall. Below are the mathematical definitions for these performance measures:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \times 100\%$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 - score = \frac{2TP}{2TP + FP + FN}$$

For any of these measures, higher scores indicate better performance. Accuracies of 100% indicate perfect predictive performance. Precision, recall, and F1-scores of 1.0 indicate the same. For the classification task in this work, a positive label was associated with a lymphoblast and a negative label was associated with a normal WBC. The combined datasets contained fewer positive examples than negative examples, so precision, recall, and F1-score are reported along with accuracy.

Chapter 5

Results & Discussion

ALL is indicated during microscopic examination of a peripheral blood smear by the presence of abnormal lymphoblasts. Lymphoblasts are distinguished from normal WBCs and in particular lymphocytes through fine-grained distinctions in their morphology. Thus the examination is a binary classification problem in which lymphoblasts are positive and WBCs are negative.

Prior works focus on segmentation and feature extraction of WBC characteristics. These features are hand-selected for input to standard supervised machine learning models. Underrepresented in ALL classification is the idea of fine-grained image classification (FGIC) and the CNN model that assumes image input. The CNN is capable of learning features from whole color images without hand-selection. This report sought to reframe ALL classification as a FGIC problem and test the validity of using whole images with a CNN in the absence of hand-selected features.

The dataset consisted of cell-centered images expertly labeled as normal WBCs or lymphoblasts. The images contained noise in the form of surrounding RBCs and uneven background illumination. In all, 388 images containing 130 lymphoblasts were split for training, validation, and testing. Three standard classifiers and one CNN were implemented using NumPy and Scikit-learn packages on a computer with an Intel Core i7 2.4GHz CPU, 8GB RAM, and Window 8.1 operating system. Each model's performance was averaged over six rounds of training/testing. Due to the skewed dataset, precision, recall, and F1-scores were reported (Table 5.1) in addition to accuracy (Figure 5.1).

Table 5.1: Classifier Precision, Recall, and F1-scores

	Precision	**Recall**	**F1-Score**
KNN	0.85	0.61	0.71
NN-1	0.83	0.87	0.80
NN-2	0.78	0.87	0.82
CNN	0.88	0.90	0.89

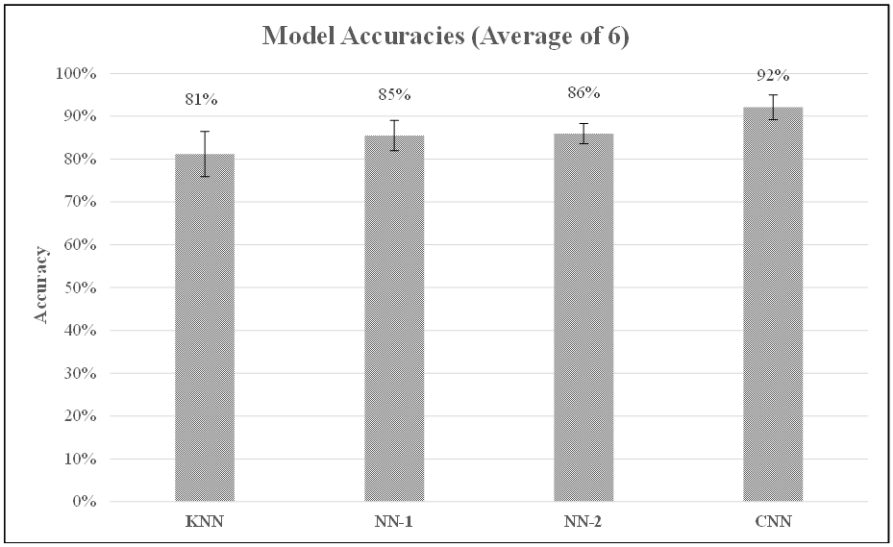


Figure 5.1: Average performance (± one std dev) of the four models.

The KNN had the lowest and most variable accuracy (81±5%). This is attributed to the model retaining the training set for measuring nearest neighbors. This made it vulnerable to data segmentation where particularly noisy images or cell populations are unequally distributed in training or test sets. Despite this, the KNN had decent performance for the classification task, making an argument for the predictive value of raw pixels from noisy cell-centered images. NN-1 and NN-2 accuracies were closely matched due to their similar architectures (85±3%; 86±2%). NN-2 had slightly better performance and less variation due to enhanced regularization from dropout and better weight initialization from batch-normalization. Inserting additional fully-connected layers did not improve performance. With only modest gains following model enhancements, the models reached a limit in whole image classification. The CNN had the best accuracy of the models tested (92±3%). The improved accuracy is the result of a single convolutional layer. Anatomic pathology error, which includes cytology, is reported to have a mean error rate of 1-5%, although wide variability is reported [8]. This indicates that the CNN is close to reaching clinical laboratory performance rates. The precision, recall, and F1-scores corroborate the accuracies. The F1-scores show clear benefits in using NNs over the KNN, and further benefit in using a CNN.

The CNN accuracy approaches or matches previous works that used hand-selected features and standard classifiers. This suggests using whole cell images in combination with a CNN holds promise for FGIC of ALL. The CNN needs improvement in light of the learned noisy filters (Figure 5.2). Extending training time is a possible solution, although the learning curve shows flat performance gains. Increasing regularization strength may provide additional refinement. Above all, a larger

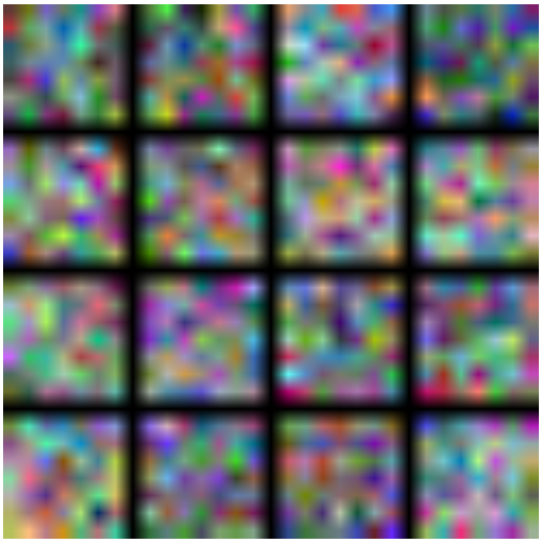


Figure 5.2: The sixteen filters learned by the CNN had considerable noise.

dataset and deeper network would provide conclusive results.

Conclusion

This work shows that using raw pixel values from noisy cell-centered images is a viable strategy for classifying ALL. Trading hand-selected feature extraction and specialized models in favor of a CNN with learnable filters may help standardize and accelerate future research efforts. In addition, the CNN generalizes to broader cell classification tasks because it makes the explicit assumption of image input. The models tested in this report are small in comparison to other published works. Nonetheless, they hold promise for FGIC of WBCs. Future work should focus on reducing filter noise and developing deeper models to accommodate larger scale classification tasks in cell biology. Using an off-the-shelf CNN and noisy image search results was explored in other classification domains. This approach motivates future work.

Bibliography

[1] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, 2012.

[2] C. M. Bishop, *Pattern Recognition and Machine Learning*, M. Jordan, J. Kleinberg, and B. SchoÂšlkopf, Eds. Springer, 2006.

[3] W. Buchser, M. Collins, T. Garyantes, R. Guha, S. Haney, V. Lemmon, Z. Li, and O. J. Trask, *Assay Development Guidelines for Image-Based High Content Screening, High Content Analysis and High Content Imaging*, Eli Lilly & Company and the National Center for Advancing Translational Sciences, October 2012.

[4] D. de Ridder, J. de Ridder, and M. J. T. Reinders, "Pattern recognition in bioinformatics," *Briefings in Bioinformatics*, vol. 14, no. 5, pp. 633 – 647, 2013.

[5] P. Domingos, "A few useful things to know about machine learning," *Communications in ACM*, vol. 55, pp. 78–87, 10 2012.

[6] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: A statistical view of boosting," *The Annals of Statistics*, vol. 28, no. 2, pp. 337 – 407, 2000.

[7] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning," book in preparation for MIT Press.

[8] D. Grzybicki, B. Turcsanyi, M. Becich, D. Gupta, J. Gilbertson, and S. Raab, "Database construction for improving patient safety by examining pathology errors," *American Society for Clinical Pathology*, 2005.

[9] M. Habibzadeh, A. Krzyzak, and T. Fevens, "White blood cell differential counts using convolutional neural networks for low resolution images," in *International Conference on Artificial Intelligence and Soft Computing*, 2013.

[10] M. Held, M. H. A. Schmitz, B. Fischer, T. Walter, B. Neumann, M. H. Olma, M. Peter, J. Ellenberg4, and D. W. Gerlich, "Cellcognition: time-resolved phenotype annotation in high-throughput live cell imaging," *Nature Methods*, vol. 7, no. 9, pp. 747 – 756, September 2010.

[11] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *Google*, 2015.

[12] A. Karpathy. (2015) Convolutional neural networks for visual recognition. Github.io. Stanford. [Online]. Available: http://cs231n.github.io/

[13] F. Kazemi, T. A. Najafabadi, and B. N. Araabi, "Automatic recognition of acute myelogenous leukemia in blood microscopic images using k-means clustering and support vector machine," *Journal of Medical Signals & Sensors*, 2016.

[14] A. Khosla, N. Jayadevaprakash, B. Yao, and L. Fei-Fei, "Novel dataset for fine-grained image categorization," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[15] J. Krause, B. Sapp, A. Howard, H. Zhou, A. Toshev, T. Duerig, J. Philbin, and F.-F. Li, "The unreasonable effectiveness of noisy data for fine-grained recognition," *CoRR*, 2015.

[16] R. D. Labati, V. Piuri, and F. Scotti, "All-idb: the acute lymphoblastic leukemia image database for image processing," in *International Conference on Image Processing*, September 2011.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, 1998.

[18] X. Long, W. L. Cleveland, and Y. L. Yao, "A new preprocessing approach for cell recognition," *IEEE Transactions on Information Technology in Biomedicine*, vol. 9, no. 3, pp. 407 – 412, October 2005.

[19] M. Mohamed and B. Far, "An enhanced threshold based technique for white blood cells nuclei automatic segmentation," in *IEEE 14th International Conference on e-Health Networking, Applications and Services*, 2012.

[20] S. Mohapatra, D. Patra, and S. Satpathy, "An ensemble classifier system for early diagnosis of acute lymphoblastic leukemia in blood microscopic images," *Neural Comput & Applic*, 2014.

[21] A. Ng. (2016) Machine learning. Coursera. Stanford University.

[22] M. A. Nielsen, *Neural Networks and Deep Learning*, M. A. Nielsen, Ed. Determination Press, 2015.

[23] PDQ. (2016) Pdq childhood acute lymphoblastic leukemia treatment. National Cancer Institute. [Online]. Available: http://www.cancer.gov/types/leukemia/patient/child-all-treatment-pdq

[24] V. Piuri and F. Scotti, "Morphological classification of blood leucocytes by microscope images," *Computationvl Intelligence for Measurement Systems and Applications*, 2004.

[25] L. Putzu and C. D. Ruberto, "White blood cells identification and counting from microscopic blood image," *International Journal of Medical, Health, Biomedical, Bioengineering and Pharmaceutical Engineering*, vol. 7, no. 1, 2013.

[26] H. Ramoser, V. Laurain, H. Bischof, and R. Ecker, "Leukocyte segmentation and classification in blood-smear images," *Engineering in Medicine and Biology 27th Annual Conference*, 2005.

[27] J. Rawat, H. Bhadauria, A. Singh, and J. Virmani, "Review of leukocyte classification techniques for microscopic blood images," *International Conference on Computing for Sustainable Global Development*, 2015.

[28] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, June 2014.

[29] S. H. Rezatofighi, H. Soltanian-Zadeh, and K. Khaksari, "Automatic recognition of five types of white blood cells in peripheral blood," in *ICIAR 2010*, June 2010.

[30] F. Scotti, "Automatic morphological analysis for acute leukemia identification in peripheral blood microscope images," *Computational Intelligence for Measurement Systems and Applications*, 2005.

[31] ——, "Robust segmentation and measurements techniques of white cells in blood microscope images," in *Instrumentation and Measurement Technology Conference*, 2006.

[32] C. Sommer, C. Straehle, U. Kothe, and F. A. Hamprecht, "Ilastik: Interactive learning and segmentation toolkit," *IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pp. 230–233, 2011.

[33] C. Sommer and D. W. Gerlich, "Machine learning in cell biology - teaching computers to recognize phenotypes," *Journal of Cell Science*, vol. 126, no. 24, November 2013.

[34] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, 2014.

[35] M.-C. Su, C.-Y. Cheng, and P.-C. Wang, "A neural-network-based approach to white blood cell classification," *The Scientific World Journal*, January 2014.

[36] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *Computing Research Repository*, 2014.

[37] W. Tai, R. Hu, H. Hsiao, R. Chen, and J. Tsai, "Blood cell image classification based on hierarchical svm," *IEEE International Symposium on Multimedia*, 2011.

[38] N. Theera-Umpon and P. Gader, "System-level training of neural networks for counting white blood cells," *IEEE Transactions on Systems, Man, and Cybernetics*, 2002.

[39] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The caltech-ucsd birds-200-2011 dataset," California Institute of Technology, Tech. Rep., 2011.

[40] S. Xie, T. Yang, X. Wang, and Y. Lin, "Hyper-class augmented and regularized deep learning for fine-grained image classification," *Computer Vision and Pattern Recognition*, 2015.

VITA

Author:

Richard K. Sipes

Place of Birth:

Knobnoster, Missouri

Schools Attended:

Eastern Washington University

Providence Sacred Heart Medical Laboratory Science Program

Degrees Awarded:

Bachelor of Science in Biochemistry, 2010, Eastern Washington University

Medical Laboratory Scientist Certification, 2012, ASCP

Masters of Science in Computer Science, 2016, Eastern Washington University

Honors and Awards:

Graduate Assistantship Sept. 2014 - June 2016

Awarded 2010 Outstanding Chemistry Graduate

First Author: "Evidence that aberrant protein metabolism contributes to chemoresistance in multiple myeloma cells," Oncology Reports, e-published March 2012