

Red Blood Cell Classification using Convolutional Networks

Exploring Machine Learning Applications in Pathology

Devansh Tandon

Advisors: Dana Angluin, Eben Olson, Richard Torres

December 2016

Abstract

This project investigates the application of machine learning techniques to problems in pathology. Using a dataset compiled by pathologists at the Yale School of Medicine, I develop a model to classify red blood cells into different groups of pathological significance. A number of convolutional network architectures are explored and tested on the original dataset of 750 images. Using a deep convolutional neural network, I am able to achieve up to 97% accuracy, a significant improvement over previous approaches.

1 Introduction

Pathologists are tasked with providing definitive disease diagnoses to guide patient treatment and management decisions. The primary method used by pathologists to make these diagnoses continues to be manual review of slides under a microscope. This qualitative visual analysis of microscopic images has many limitations: lack of standardization, diagnostic errors, and significant

cognitive load on a pathologist evaluating thousands of cells across hundreds of slides in a typical workday.

Manual pathological review is very time consuming and tedious, and there is great interest in developing image analysis systems that can assist pathologists by automating routine tasks and performing speculative segmentation and classification of images.

This project is an attempt to apply a new approach to the image classification problem in the context of pathology – using deep convolutional networks to classify red blood cells. The larger goal is to develop a system that can speculatively classify cells into categories with pathological significance to assist pathologists in their daily workflow.

If the algorithm is able to identify the correct structures and classify them accurately, it would be possible for the computer to make a diagnosis or compute some preliminary diagnostic score. Such a metric would be very fast, have a large baseline of objectivity since it can thoroughly examine all the available data, and be easily scalable. This would be a powerful addition to current clinical practice.

2 Background

2.1 Pathology

Pathology is the practice of determining the cause of a disease, especially by examining samples from the body. A common tool in anatomic pathology is histology, the study of the microscopic anatomy of cells and tissue. Histology slides are increasingly being digitized to produce high-resolution images. These digital slides enable the possibility of applying image analysis techniques to digital pathology for detection, segmentation and classification of cells. Algorithmic approaches have already shown benefits in some contexts by reducing the tedious process of quantifying information in the slides, and also by providing a diagnosis that can inform the decisions of pathologists [1].

Typically, a pathologist would look at a slide through a microscope to search for areas or structures of interest to analyze. It is tedious and time consuming for a pathologist to manually search through hundreds of slides for this information. A computer would be much more efficient at scanning through thousands of digital slides to identify and mark areas of interest to augment a pathologist's practice.

Most of the current literature in digital pathology approaches image classification tasks using conventional machine learning techniques [1]. A typical process would involve manual feature engineering by running image processing code on input images to extract features such as area of cells, mean and variance of distance between nucleus and cell walls, etc. These features are then used to train classification models using regressions, support vector machines and neural networks.

This project is focused on the classification of red blood cells. Much diagnostic information can be extracted from the shape and size of the red blood cells, the number of the red blood cells in a particular sample of the blood, and the ratio between the area containing oxygen and the whole area of each cell [5]. Analyzing the hematological images manually is tiresome and time consuming. It also suffers from inter and intra observer variability. Hence, automation of this task will be helpful in identifying diseases related to the red blood cells accurately, and saving the precious time of hematologists.

One issue with medical computer vision applications is the lack of training data. While there are many publicly available image datasets (e.g. IMAGENET, CIFAR), medical image datasets are generally smaller and much rarer. This makes over-fitting a real concern for machine learning models in pathology, since it is difficult to build a robust model using a small dataset. The lack of a large, standardized, public dataset for researchers to work with is one of the biggest challenges in using deep learning techniques for applications in pathology. This project is based on a new dataset being developed by a lab at the Yale School of Medicine.

2.2 Deep Learning

Arthur Samuel described machine learning as a field that “gives computers the ability to learn without being explicitly programmed” (1959). Machine learning algorithms can generalize from examples, and extract patterns from raw data. This is often a very attractive alternative to manually constructing programs, and in the last decade the use of machine learning has spread rapidly throughout computer science and beyond. Pedro Domingos summarizes machine learning as ‘Learning = Representation + Evaluation + Optimization’ [2].

The performance of these simple machine learning algorithms depends heavily on the representation of the data they are given. Many artificial intelligence tasks can be solved by designing the right set of features to extract for that task, then providing these features to a simple machine learning algorithm. However, for many tasks, it is difficult to know what features should be extracted. for example, suppose that we would like to write a program to detect birds in photographs. We know that birds have wings and feathers, so we might like to use the presence of a wing or feathers as a feature. Unfortunately, it is difficult to describe exactly what a wing or feathers look like in terms of pixel values. Extracting the right features to train on, or finding the right representation of the data is a big challenge in developing a machine learning algorithm.

Conventional machine-learning techniques are limited in their ability to process natural data in their raw form. for decades, constructing a pattern-recognition or machine-learning system required careful engineering and considerable domain expertise to design a feature extractor that transformed the raw data (such as the pixel values of an image) into a suitable internal representation or feature vector from which the learning system, often a classifier, could detect or classify patterns in the input.

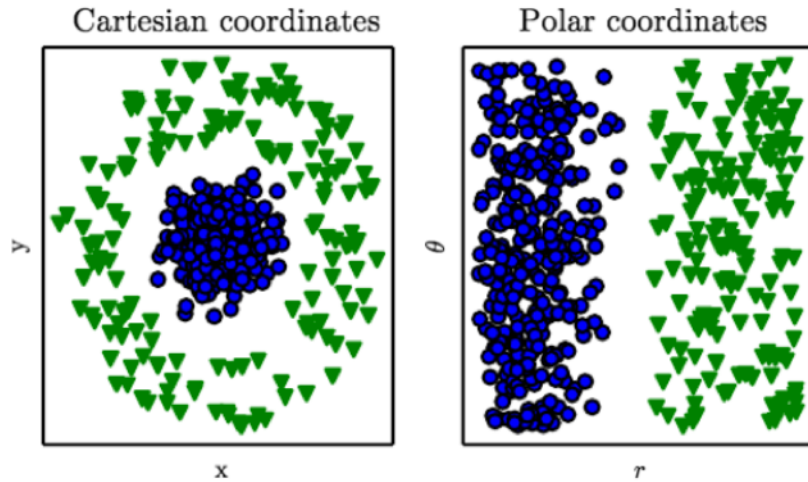


Figure 1: Example of different representations: suppose we want to separate two categories of data by drawing a line between them in a scatterplot. In the plot on the left, we represent some data using Cartesian coordinates, and the task is impossible. In the plot on the right, we represent the data with polar coordinates and the task becomes simple to solve with a vertical line. [from Goodfellow et al, 2016] [3]

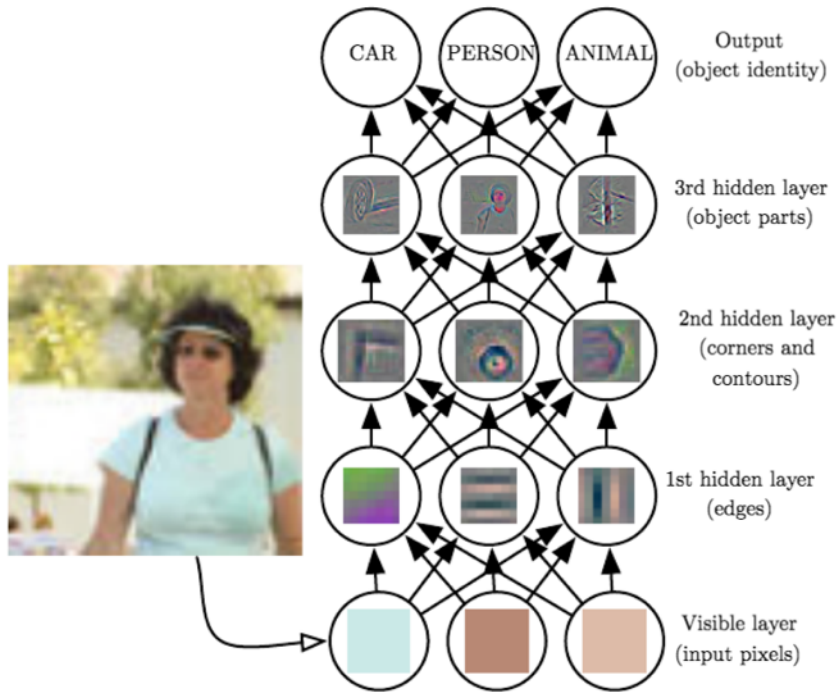


Figure 2: Illustration of a deep learning model. It is difficult for a computer to understand the meaning of raw sensory input data, such as an image represented as a collection of pixel values. Learning or evaluating the mapping from pixel values to the object seems insurmountable if tackled directly. Deep learning resolves this problem by breaking the desired complicated mapping into a series of nested simple mappings, each described by a different layer of the model. A series of hidden layers extract increasingly abstract features from the image. The model must determine which concepts are useful for explaining the relationships in the observed data, and represent this using the hidden layers. Given the pixels, the first layer can easily identify edges, by comparing the brightness of neighboring pixels. Given the description of the edges, the second hidden layer can easily search for corners and extended contours, which are recognizable as collections of edges. Given the description of the image in terms of corners and contours, the third hidden layer can detect entire parts of specific objects, by finding specific collections of contours and corners. finally, this description of the image in terms of the object parts it contains can be used to recognize the objects present in the image. [from Goodfellow et al, 2016] [3]

This problem can be solved by using machine learning to not only discover the mapping from a representation of data to the output, but also to discover the best representation itself – called representation learning. Representation learning is a set of methods that allows a machine to be fed raw data and to automatically discover the representations needed for detection or classification. Deep learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level. With the composition of enough such transformations, very complex functions can be learned.

2.3 Convolutional Neural Networks

In the last few years, deep neural networks have led to breakthrough results on a variety of pattern recognition problems, such as computer vision and voice recognition [6]. One of the essential components leading to these results has been a special kind of neural network called a convolutional neural network. At their most basic, convolutional neural networks can be thought of as a kind of neural network that uses many identical copies of the same neuron. This allows the network to have lots of neurons and express computationally large models while keeping the number of actual parameters – the values describing how neurons behave – that need to be learned fairly small.

A typical block of a convolutional network consists of three stages. In the first stage, the layer performs several convolutions to produce a set of linear activations. In the second stage, each linear activation is run through a nonlinear activation function. In the third stage, we use a pooling function to modify the output of the layer further. Convolution is an extremely efficient way of describing transformations that apply the same transformation of a small, local region across the entire input.

Convolutional Neural Networks leverage four important ideas: sparse interactions, parameter sharing, equivariant representations, and pooling. They use convolution in place of general matrix multiplication in their layers.

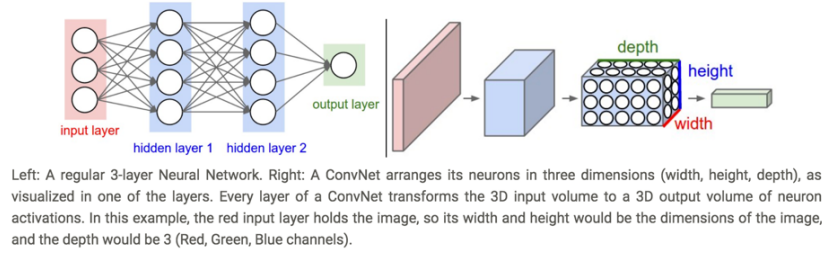


Figure 3: from Stanford CS231 Course Notes [4]

Convolution Operation

In the general form, a convolution is a mathematical operation on two functions. It is an integral that expresses the amount of overlap of one function g as it is shifted over another function f . It is defined as:

$$s(t) = (f * g)(t) = \int f(\tau)g(t - \tau)d\tau$$

In machine learning, the input is usually a multidimensional array of data and the kernel is usually a multidimensional array of parameters that are learned by the algorithm. An example of a convolution operation: if we use a two-dimensional image I as our input, and a two-dimensional kernel K , we get:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n).$$

Sparse Interactions

In traditional neural networks, there is a matrix of parameters, with each parameter describing the interaction between each input unit and each output unit. Thus, every output unit interacts with every input unit (the network is fully connected). However, by making the kernel much smaller than the input, convolutional networks achieve sparse interactions. For example, when processing an image, the input image might have thousands of pixels, but we can detect small, meaningful features such as edges with kernels that occupy only tens of pixels. This means that we need to store fewer parameters, which both reduces

the memory requirements of the model and improves its statistical efficiency. It also means that computing the output requires fewer operations.

These improvements in efficiency are usually quite large. If there are m inputs and n outputs, then matrix multiplication requires $m * n$ parameters and the algorithms used in practice have $O(m * n)$ runtime (per image). If we limit the number of connections each output may have to k , then the sparsely connected approach requires only $k * n$ parameters and $O(k * n)$ runtime. It is possible to obtain good performance on machine learning tasks while keeping k several orders of magnitude smaller than m .

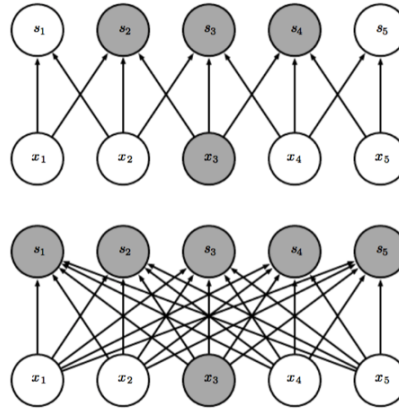


Figure 4: (Top) When s is formed by convolution with a kernel of width 3, only three outputs are affected by x_3 . (Bottom) When s is formed by matrix multiplication, connectivity is no longer sparse, so all of the outputs are affected by x_3 . [from Goodfellow et al, 2016] [3]

Parameter Sharing

Parameter sharing is the idea of using the same parameter for more than one function in a model. In traditional neural networks, each element of the weight matrix is used only once when calculating the output of the layer – it is multiplied by the input unit and not reused. In a convolutional neural network, each member of the kernel is used at every position of the input. This means that rather than learning a separate set of parameters for every location in the

input image, convolutional networks only learn one set. Convolution is thus much more efficient than matrix multiplication in both memory requirements and statistical efficiency (since there are fewer parameters to be trained).

Equivariance

To say a function is equivariant means that if the input changes, the output changes in the same way. Specifically, a function $f(x)$ is equivariant to a function $g(x)$ if $f(g(x)) = g(f(x))$. In the case of convolution, if we let g be any function that translates the input, i.e., shifts it, then the convolution function is equivariant to g . for example, let I be a function giving image brightness at integer coordinates. Let g be a function mapping one image function to another image function, such that $I = g(I)$ is the image function with $I(x, y) = I(x - 1, y)$. This shifts every pixel of I one unit to the right. If we apply this transformation to I , then apply convolution, the result will be the same as if we applied convolution to I , then applied the transformation g to the output.

The key idea here is that convolution creates a 2D map of where particular features appear in the input. If we move the object in the input, its representation will move the same amount in the output. for example, when processing images, it is useful to detect edges in the first layer of a convolutional network. The same edges appear more or less everywhere in the image, so it is practical to share parameters across the entire image.

Pooling

A pooling function replaces the output of the network at a particular location with a summary statistic of nearby outputs. for example, the max pooling operation outputs the maximum output within a rectangular neighborhood. Pooling helps the representations learned to become invariant to translations in the input. Invariance to local translation can be a very useful property if we care more about whether some feature is present than exactly where it is. Pooling over spatial regions produces invariance to translation, but if we pool over the outputs of separately parametrized convolutions, the features can also

learn which transformations to become invariant to.

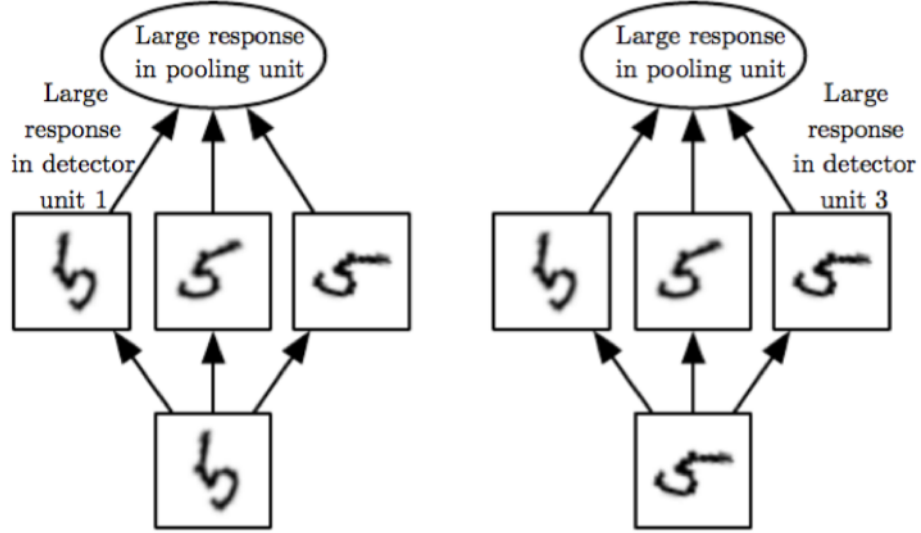


Figure 5: Example of learned invariances: A pooling unit that pools over multiple features that are learned with separate parameters can learn to be invariant to transformations of the input. Here we show how a set of three learned filters and a max pooling unit can learn to become invariant to rotation. All three filters are intended to detect a hand-written 5. Each filter attempts to match a slightly different orientation of the 5. When a 5 appears in the input, the corresponding filter will match it and cause a large activation in a detector unit. The max pooling unit then has a large activation regardless of which detector unit was activated. We show here how the network processes two different inputs, resulting in two different detector units being activated. The effect on the pooling unit is roughly the same either way. [from Goodfellow et al, 2016] [3]

3 Data

This project is based on a dataset collected by the flow Cytometry Lab in the Yale School of Medicine, run by Dr. Richard Torres. The dataset consists of 750 images of red blood cells that have been labeled by pathologists working

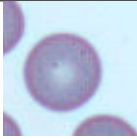
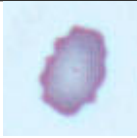

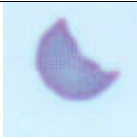

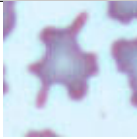
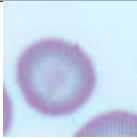
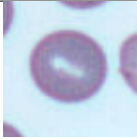
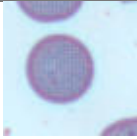
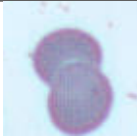
the Yale School of Medicine and the Yale-New Haven Hospital.

Pathologists look at slides of red blood cells (RBC) for a number of clinical signs to inform their diagnosis as a part of their daily practice. Dr. Torres' lab developed a tool that allows pathologists to easily label this RBC data – by right clicking on a cell and selecting a group when looking at digital images, they can easily mark it as belonging to one of 10 groups of clinical significance. The tool then crops a region of the image (around the cell that the pathologist clicked on) and labels it appropriately.

This image data was run through a number of preparation steps before being fed into the model. First, the images were cropped to a uniform (70 x 70) pixel size for each image, with the cell in question being centered to the extent possible. Due to the small size of the dataset (about 750 images), there were a number of augmentation steps to try to increase the size of the dataset to be able to train a better model. These included a number of image transformations – spatial transformations of rotations, flips, normalization, and centering of images. After augmenting, the size of the dataset could be increased to about 4000 images.

```
Dataset stats:
Loading September_1_total_non_overlap
0 - NORMAL: 203
1 - Echinocyte: 63
2 - Dacrocyte: 18
3 - Schistocyte: 152
4 - Elliptocyte: 18
5 - Acanthocyte: 33
6 - Target cell: 145
7 - Stomatocyte: 22
8 - Spherocyte: 48
9 - Overlap: 46
```

Figure 6: Summary statistics for the original dataset.

Class	Example Picture	Number
Normal		203
Echinocyte		63
Dacrocyte		18
Schistocyte		152
Elliptocyte		18
Acanthocyte		33
Target Cell		145
Stomatocyte		22
Spherocyte		48
Overlap		46

4 Methodology

After collecting the augmented dataset, a number of convolutional neural networks were trained on the data. All the experiments were conducted after splitting the data set into a training group (80%) and a test set (20%). The training group was further split into a training set (80%) and a validation (20%) set.

The original dataset had a wide disparity in the number of examples between classes – for example, 203 examples for class Normal and only 18 for class Dacrococyte. To address this, the validation set was constructed using stratified sampling – so that the smaller classes would also be accurately represented in the validation set. This significantly improved the accuracy of results and helped combat over-fitting.

Most architectures were composed of a combination of convolutional layers with a rectified linear unit (ReLU) activation function and pooling layers (sometimes organized into blocks), dropout layers, and a final densely connected layer with a softmax activation function for the final classification output. Most models used a categorical cross entropy loss function, a stochastic gradient descent optimizer, and categorical accuracy as the primary metric.

A number of open source libraries were used to implement and train the convolutional architectures, including Keras, Tensorflow, Theano and Lasagne. Code implementing the final architecture is attached in Theano+Lasagne and Theano+Keras.

5 Results

The convolutional neural network was successful at the Red Blood Cell classification task, and was able to outperform other approaches. The model was able to achieve 97.59% accuracy. The final architecture, confusion matrix and summary metrics are presented below.

from the confusion matrix, we can see that most of the errors made by the

model are concentrated in the categories with few examples. This is promising, because we would expect the model to perform worse on categories where it has less examples to train and learn from. Given more data, it is likely that the architecture will be able to learn a better representation, and get better at differentiating between these categories.

Summary classification metrics:				
	precision	recall	f1-score	support
0	0.99	1.00	0.99	203
1	0.98	0.98	0.98	63
2	0.93	0.72	0.81	18
3	0.95	0.96	0.96	152
4	1.00	0.94	0.97	18
5	0.89	0.94	0.91	33
6	1.00	1.00	1.00	145
7	0.95	0.95	0.95	22
8	0.98	0.96	0.97	48
9	1.00	1.00	1.00	46
avg / total	0.98	0.98	0.98	748

Figure 7: Classification metrics using final architecture.

Accuracy: 0.975935828877

Confusion matrix:

```
[[203  0  0  0  0  0  0  0  0  0]
 [  1 62  0  0  0  0  0  0  0  0]
 [  1  0 13  3  0  0  0  0  1  0]
 [  0  1  1 146  0  4  0  0  0  0]
 [  0  0  0  1 17  0  0  0  0  0]
 [  0  0  0  2  0 31  0  0  0  0]
 [  0  0  0  0  0  0 145  0  0  0]
 [  1  0  0  0  0  0  0 21  0  0]
 [  0  0  0  1  0  0  0  1 46  0]
 [  0  0  0  0  0  0  0  0  0 46]]
```

Figure 8: Confusion matrix.

```

Layer 0 type: <class 'lasagne.layers.input.InputLayer'> output: (32, 3, 70, 70)
Layer 1 type: <class 'lasagne.layers.dnn.Conv2DDNNLayer'> output: (32, 32, 68, 68)
Layer 2 type: <class 'lasagne.layers.dnn.Conv2DDNNLayer'> output: (32, 32, 66, 66)
Layer 3 type: <class 'lasagne.layers.dnn.Conv2DDNNLayer'> output: (32, 32, 64, 64)
Layer 4 type: <class 'lasagne.layers.dnn.MaxPool2DDNNLayer'> output: (32, 32, 31, 31)
Layer 5 type: <class 'lasagne.layers.noise.DropoutLayer'> output: (32, 32, 31, 31)
Layer 6 type: <class 'lasagne.layers.dnn.Conv2DDNNLayer'> output: (32, 64, 29, 29)
Layer 7 type: <class 'lasagne.layers.dnn.Conv2DDNNLayer'> output: (32, 64, 27, 27)
Layer 8 type: <class 'lasagne.layers.dnn.Conv2DDNNLayer'> output: (32, 64, 25, 25)
Layer 9 type: <class 'lasagne.layers.dnn.MaxPool2DDNNLayer'> output: (32, 64, 12, 12)
Layer 10 type: <class 'lasagne.layers.noise.DropoutLayer'> output: (32, 64, 12, 12)
Layer 11 type: <class 'lasagne.layers.dnn.Conv2DDNNLayer'> output: (32, 128, 10, 10)
Layer 12 type: <class 'lasagne.layers.dnn.Conv2DDNNLayer'> output: (32, 128, 8, 8)
Layer 13 type: <class 'lasagne.layers.dnn.Conv2DDNNLayer'> output: (32, 128, 6, 6)
Layer 14 type: <class 'lasagne.layers.pool.GlobalPoolLayer'> output: (32, 128)
Layer 15 type: <class 'lasagne.layers.noise.DropoutLayer'> output: (32, 128)
Layer 16 type: <class 'lasagne.layers.dense.DenseLayer'> output: (32, 10)

```

Figure 9: Layers of architecture.

Layer	Specification
Input Image	shape = (3 x 70 x 70)
2D Convolution	num nodes = 32, filter = (3 x 3), activation = ReLU
2D Convolution	num nodes = 32, filter = (3 x 3), activation = ReLU
2D Convolution	num nodes = 32, filter = (3 x 3), activation = ReLU
Max Pooling	pool size = (3 x 3), strides = (2 x 2)
Dropout	0.25
2D Convolution	num nodes = 64, filter = (3 x 3), activation = ReLU
2D Convolution	num nodes = 64, filter = (3 x 3), activation = ReLU
2D Convolution	num nodes = 64, filter = (3 x 3), activation = ReLU
Max Pooling	pool size = (3 x 3), strides = (2 x 2)
Dropout	0.25
2D Convolution	num nodes = 128, filter = (3 x 3), activation = ReLU
2D Convolution	num nodes = 128, filter = (3 x 3), activation = ReLU
2D Convolution	num nodes = 128, filter = (3 x 3), activation = ReLU
Max Pooling	pool size = (3 x 3), strides = (2 x 2)
Dropout	0.5
Dense Layer	10 dimensions, activation = softmax
Optimizer	Stochastic Gradient Descent
Loss function	Categorical Cross Entropy

5.1 Results in Perspective

It is tough to show a valid comparison of these results and the classification accuracy since the data the model is trained on is a proprietary dataset being collected by the Yale School of Medicine. There are no previous results to benchmark against, since this project aimed to explore whether convolutional networks would be successful in image classification in the context of pathology – to apply recent breakthroughs in deep learning to a new problem. The only comparable work on this dataset has been previous work in the lab.

Previous approaches to similar classification tasks have relied on feature engineering [1]. A number of features can be manually extracted from the images to describe the cells in question such as: area, perimeter, circularity, dispersion, variance and specifically designed features. Many papers use the mean and variance of the distance between the cell wall and the nucleus as a feature. These features are then used to train a SVM or a neural network that then classifies the image.

In ‘feature Extraction and Classification of Blood Cells Using Artificial Neural Network’ [8], a simple back propagation neural network is trained on features that are extracted from the data after image processing, to classify cells as normal or abnormal. They are able to achieve 80% and 66.6% accuracy for normal and abnormal respectively.

Lee and Chen also use features extracted from images of red blood cells to classify them in ‘Cell morphology based classification for red cells in blood smear images’ [7]. They define shape features such as cell circularity, medial axis ratio, and cell deformation ratio, and use image processing code to extract them. They also define another class of texture features such as the mean and variation of cell intensity and the smoothness of the cell. The gains in accuracy in these approaches come from feature engineering – it takes a lot of experimentation and domain knowledge to find the features that will best represent the image data. Once features are extracted, a classical neural network is used to train a classifier based on these features. The best accuracy achieved by this approach is 91% on a dataset of 200 images.

Red blood cell disease classification.

Disease type (total images used)	All features	Classification methods			
		mRMR dual layer [15]	RCFS [24]	NMIFS [4]	Hybrid NN classifier
Burr cell (48)	25	42	38	39	41
Sickle cell (64)	43	54	50	52	58
Horn cell (44)	24	35	39	38	41
Elliptocyte cell (44)	27	38	38	40	42
Accuracy	40%	84.5%	84%	84.5%	91%

The results highlighted in bold indicate the best results achieved in the separate experiments.

Figure 10: Classification metrics from ‘Cell morphology based classification for red cells in blood smear images’ [7].

The convolutional network approach does not require any feature engineering or domain specific knowledge, since the lower layers of the network learn their own representation of the data, which later layers build on. Previous approaches used in the lab on the same dataset involved training an SVM and neural network model on features extracted from the image, and were able to achieve 70% accuracy.

The high accuracy of this model demonstrates that deep convolutional networks can be successfully applied to classification tasks in pathology. This is a promising result, especially since as more data is collected, the accuracy could be improved further using the same architecture.

5.2 Alternative Architectures Explored

Simple Convolution

The first few architectures explored involved 1-2 simple convolutional layers, followed by pooling and the final dense softmax layer for classification. These simple architectures were able to achieve about 30% accuracy, and since there

were fewer free parameters they would train very quickly. These models would get caught in a local maximum – categorizing all images as belonging to the biggest class.

Convolution Block

The next architectures involved creating a block of layers – 3 convolutional layers, followed by a pooling layer, and a dropout layer. I experimented with different numbers of nodes in each block. Adding dropout layers between blocks was essential to getting the model to explore beyond the local valleys. This architecture was able to get up to 70% accuracy.

Due to the small size of the original dataset, when training a deep convolutional network, it is likely that there would be enough free parameters to completely describe the dataset – this makes overfitting a real concern. One way to mitigate this problem is to introduce dropout layers. The key idea here is to randomly drop units (along with their connections) from the neural network during training. This significantly reduces overfitting and improves the performance of the network.

VGG Net Inspired

The final architecture was a number of blocks chained together, inspired by the VGG Net architecture developed at the Visual Geometry Group at Oxford [9]. This model was developed for the ImageNet challenge and is a very deep convolutional network made of 16 and 19 layers.

The images in the RBC dataset were composed of relatively simple features (in comparison to ImageNet), so the deeper layers that represent higher level features were not necessary. Adding further blocks did not increase accuracy in classification, but took much longer to train due to the increase in number of parameters and led to overfitting.

6 Conclusion

Historically, approaches to histopathological image analysis in digital pathology have focused primarily on low level image analysis tasks (e.g., color normalization, nuclear segmentation, and feature extraction), followed by the construction of classification models using classical machine learning methods, including: regression, support vector machines, and random forests.

Since 2012, deep learning-based approaches have consistently shown best-in-class performance in major computer vision competitions, such as the ImageNet challenge [6]. This project was an attempt to apply a deep convolutional network to a classification problem in pathology. In a deep learning based approach there is no need for manual feature extraction. Instead, deep learning algorithms take as input only the images and their labels and learn a complex set of model parameters, building on internal representations of the image, with supervision coming only from the image labels.

The model trained on a dataset collected by Dr. Torres' lab is able to achieve a high classification accuracy of 97.59%, even with limited training data. Since the errors made by the model are concentrated in the classes with very few samples to train on, it is likely that the architecture has the potential to attain even better results given better training data. This is a promising result, since the convolution network is able to outperform traditional approaches in digital pathology and has the potential to be a useful aid in the daily practice of a pathologist.

References

- [1] Rohit Bhargava and Anant Madabhushi. Emerging themes in image informatics and molecular analysis for digital pathology. *Annual Review of Biomedical Engineering*, 18(1):387–412, Nov 2016.
- [2] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78, Jan 2012.
- [3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.
- [4] Andrej Karpathy. Cs231n convolutional neural networks for visual recognition.
- [5] Kyungsu Kim, Jeonghee Jeon, Wankyoo Choi, Pankoo Kim, and Yo-Sung Ho. Automatic cell classification in human’s peripheral blood images based on morphological image processing. *AI 2001: Advances in Artificial Intelligence Lecture Notes in Computer Science*, page 225–236, 2001.
- [6] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- [7] Howard Lee and Yi-Ping Phoebe Chen. Cell morphology based classification for red cells in blood smear images. *Pattern Recognition Letters*, 49:155–161, 2014.
- [8] Samira. Feature extraction and classification of blood cells using artificial neural network. *American Journal of Applied Sciences*, 9(5):615–619, Jan 2012.
- [9] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.