Nuclei Localization and Classification in Post-treated Breast Surgical Specimens

by

Rene Bidart

A research paper
presented to the University of Waterloo
in fulfillment of the
requirement for the degree of
Master of Mathematics
in
Statistics

Waterloo, Ontario, Canada, 2007

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Finding an classifying the normal epithelial, malignant epithelial and lymphocyte cells in breast cancer tissue samples is useful for patient management, but is time consuming and suffers from inter-observer variability when done manually. In this paper we attempt using U-Net, regression based methods, and classifier based methods for this problem. We find a using classifier based method with fully convolutional neural networks (CNN) for localization and classification is both efficient and accurate. To find the nuclei in the slide (localization), this CNN is applied over the entire slide, generating 4 heatmaps corresponding to the probability of a pixel being the center of a lymphocyte, normal, malignant or nuclei, or non-nuclei. Non-maximum suppression is used to extract nuclei location predictions from this heatmap, and then the highest probability is taken as its classification. The final classification accuracy was 94.6%, surpassing previous machine learning methods on this dataset.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

There are a variety of methods used to treat breast cancer, including surgery, radiation and chemotherapy. Neoadjuvant treatment is used for certain high risk, large, or locally advanced patients to attempt to reduce the size of the tumor before surgery. It is useful to estimate how effective this treatment was, and the extent of the tumor burden.[24]

Currently the standard way to do this is for pathologists to examine tissue samples and estimate the cellularity fraction of cancer.[18] The issue is that this is time consuming and suffers from inter-observer variability.[13] We propose to assist the pathologist by localizing and classifying the nuclei in a given slide, which we hope can be useful to more quickly and accurately assess tumor burden.

Deep learning using convolutional neural networks (CNN) has gained popularity for a variety of tasks since the 2012 imagenet competition, where they demonstrated previously unseen effectiveness for image recognition [9]. Recently CNNs have also performed well on medical specific tasks, including reaching human level accuracy for segmenting cancer lesions on full slide images[11]. Because of these related successes, we think that deep learning can be useful for this problem as well.

Chapter 2

Background

2.1 Convolutional Neural Networks

Before we can look into image segmentation and localization we will look at the architecture that underlies all of these methods, the convolutional neural network (CNN). There is a brief review of why convolution is so useful for image data, followed by a review of the most prominent model architectures used for this problem, to inform the architectures that will be used for our problem. For each network we will discuss the aspects that have proven to be useful, omitting details that tend to be practically irrelevant.

2.1.1 Convolution

In a traditional feed forward neural network every input variable connected to every variable in the hidden layer. In image recgonition, this corresponds to there being a weight that must be learned for each pixel in the input image, times the number of hidden nodes. With high dimensional input, this will become too computationally expensive, and prone to overfitting.

The solution to this is to only connect each neuron in the hidden layer to a small spatially connected section of the input image, instead of being connected to the full input. This takes advantage of the fact that in images there are features that are translation invariant, meaning that the same features will be computed for different spatial locations in the image.

In a convolution layer, a filter is learned that is applied across the entire input image. This filter is of size $(height \cdot width \cdot input\_features)$ For example, in the first layer of a CNN there could be a filter of size $3x3x3$ applied to the input image. The last dimension of the filter is determined by the number of features in the input, while the height and width are user specified.

Many of these filters are learned for each layer, each extracting a different feature map from the input image. Each convolution layer is a feature extractor, and by stacking more and more layers together we can extract higher and higher level features from the image.

We will now review what architecture choices have given the best results for CNNs, highlighting what is relevant for out problem. This is done in a chronological order.

2.1.2 LeNet

The first successful neural network was the LeNet [10], which was applied to digit recognition. As shown in figure 2.1, this is a simple architecture consisting of two fully connected layers with filter size of 5x5, each followed by max pooling layers. Finally there are three fully connected layers. The intuition is that the convolution layers extract features from the image, which are then downsampled by the max pooling layers to reduce the dimension to a reasonable size. The final fully connected layers use these features to generate class predictions.

Max pooling is an operation that downsamples the feature map, reducing the spatial size of the input, reducing the parameters needed in subsequent layers. This generally uses a maximum filter of size 2x2, applied with a stride of 2. This downsamples the spatial dimension by a factor of 2, but leaves the number of feature maps unchanged.

2.1.3 Alex Net

Between the 1990s and 2012 neural networks received little attention, and of other computer vision methods relying on traditional machine learning methods were used for classification.

It is not that there was no work going on in the field. In [27], they introduced the first CNN that ran on GPU, and during this time there were plenty of architecture and training methods, but this did not receive much attention until 2012, when AlexNet was created.

AlexNet beat competing architectures in the imagenet competition with accuracy of 15.4%, compared to the next best at 26.2% restoring interest in using neural networks

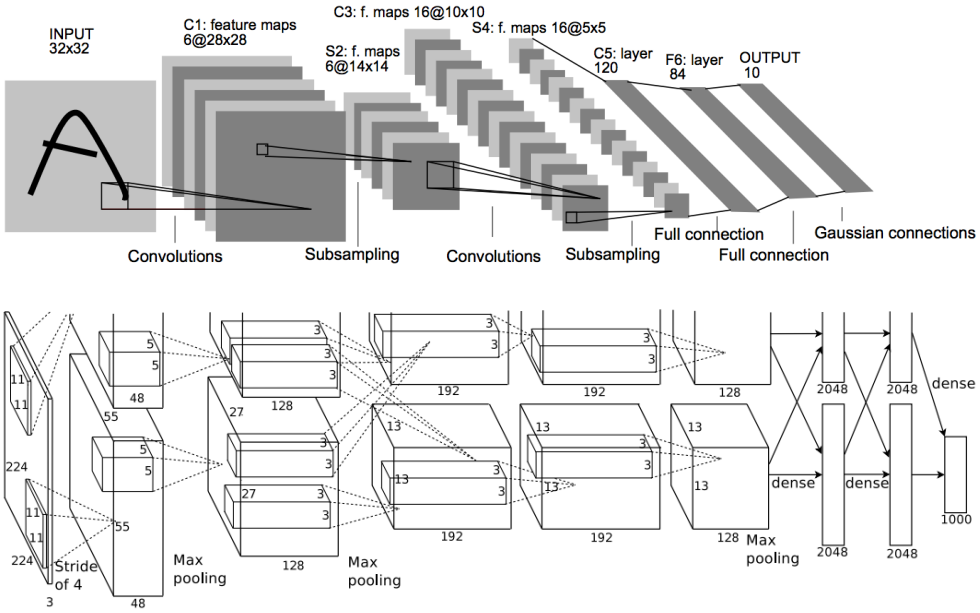


Figure 2.1: *(a) The LeNet architecture, consisting of two convolution layers followed by three fully connected layers. [10] (b) AlexNet, which has a two path architecture to allow the model to fit on two gpus*

for computer vision. This network is significantly larger than the LeNet, consisting of 5 convolution layers of size 11x11, 5x5 and 3x3 followed by 3 fully connected layers

Previous networks had suffered from three problems, which the Alexnet had addressed:

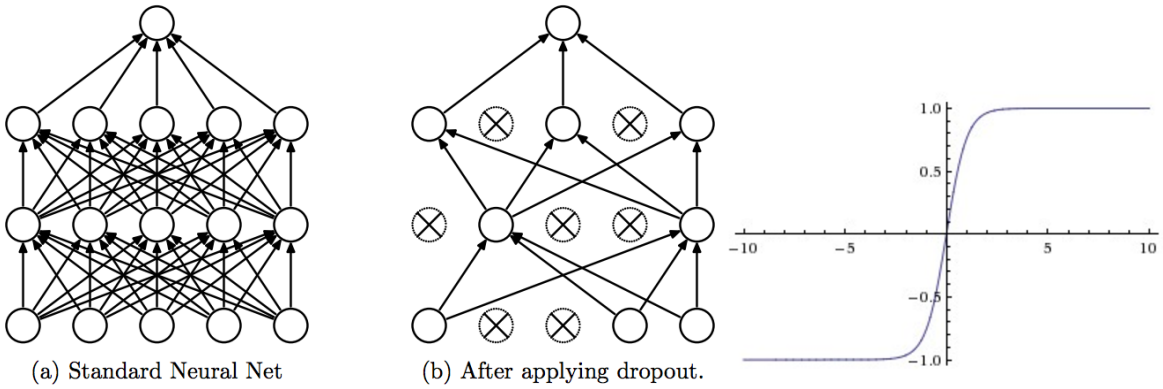
- **Compute Power** - A large network is necessary to have the complexity to do image recognition, but it extremely computationally expensive to train this on CPU. In AlexNet this problem is overcome by training the network on GPU. In order to train a large enough network, it had to be split into two parts for it to be trained on two GPUs. This is why the network has an unusual architecture (figure 2.1)

- **Vanishing Gradient** - Traditional activation functions like tanh or sigmoid have an issue where neurons with either very high or very low input become saturated, and reduce the gradient close to 0. As shown in figure 2.2 (b), when the x-value is significantly far from 0, the slope will be close to 0. This means that these types of activation functions need very careful weight initialization, and tend to cause this vanishing gradient problem as multiple layers of these functions are stacked together. To overcome this the ReLU activation function is used $\max(0, x)$ In practice they found it converged much faster to use ReLU than tanh. The ReLU is not without problems though. When the value is below 10 the ReLU activation function will also kill the gradient. This can also result in "dead neurons", meaning that you can learn neurons that will never activate, because the input value will always be below 0 [8].

- **Regularization** - This architecture has 60 million parameters, and so overfitting can be a problem. **Data augmentation** is used to help prevent overfitting. They apply random horizontal reflections and translations to the images, artificially increasing the training set size while preserving the labels of each example. They also add random noise to the RGB channels. Another regularization method used is dropout.

  Dropout[20] consists of randomly setting the output of certain nodes to 0 during the training procedure. These nodes are removed from both the forward pass and backpropagation. Intuitively we can think that the neurons are being forced to learn a redundant representation, with them being able to perform classification even if half are missing. It can also be thought of as training a large ensemble of models, each of which are sub networks of the full network.

  At test time, ideally we would run many forward passes with random dropout combinations, and average the results. Unfortunately this is too computationally expensive, so instead we can approximate this by turning on all the connections, and rescaling



(a) Standard Neural Net (b) After applying dropout.

Figure 2.2: *(a) An example of a simple two layer network, and the same network after dropout has been applied. [20] (b)The tanh non-linearity, showing how the derivative goes to 0 as x values become large or small*

the weights by the amount of dropout used. Then a single forward pass can give the prediction.

A higher amount of dropout is used where there is more redundant weights, for example fully connected layers will use a higher amount than convolution layers. fig 2.2(a)

2.1.4 VGG-16

In this network they use the standard idea of stacking convolution layers, followed by fully connected layers, with max pool for downsampling. The big difference between previous networks and this one was that it used a deeper architecture, with 16 layers. This was made possible by using smaller convolution kernels. In past networks they would tend to mix kernels of different sizes, but in VGG they used a simple architecture consisting only of 3x3 convolutions. The logic behind this was that using multiple smaller kernels will allow each kernel to have the same receptive field as a larger kernel, but while using fewer parameters.

We can compare a 5x5 convolution kernel to two layers 3x3 kernels. In the 5x5 kernel $25C^2$ weights will need to be stored, whereas with two layers of 3x3 convolutions vs $9C^2 + 9C^2 = 18C^2$, where c is the number of input channels. They will both have the same receptive field, although the stacked 3x3 layers will use two non-linearities rather than one.

2.1.5 GoogleLeNet

The VGG network was the limit of blindly stacking more convolution and fully connected layers. It It was effective, but results in large and difficult to train networks. In the GoogleLeNet (Inception V1), a different approach is taken, resulting in a smaller and better performing network. This paper introduced three main ideas:

- **Inception module** - The inception module is based on the idea that a different sized convolution filters can be useful at any given layer of a network. They implement this by simultaneously applying filters of size 1x1, 3x3, 5x5 and max pooling, and then concatenating the output. (figure 2.3 (a)) By doing this the network is allowed to learn which sized filter is the most useful, and can combine information form different spatial resolutions.

- **1x1 convolutions** - The issue with the inception module is that the number of features can become large quickly when you are concatenating the output from these different convolutions, and so it is prohibitively expensive to apply 3x3 and 5x5 convolutions. Their insight was to use a 1x1 convolution layer to downsample the input to a smaller number of features. (figure 2.3 (b))

- **Average pooling** - Another important insight was realizing that the final fully connected layers in a CNN are not necessary. These can contain a disproportionate number of the weights in a CNN, and so are a inefficient part of the network. For example, in VGG-16 the first fully connected layer has 100 million out of the total 140 million network parameters.
  To use global average pooling instead of the fully connected layers, the final convolution layer must have the same number of output feature maps as the number of classes in the dataset. Average pooling is done on each of the spatial layers to give a probability for each class. Intuitively, in the final convolution layer the network is forced to learn some kind of probability map for each class, and then we output the overall probability by taking the average of this.

Using all these methods results in a network with only 4 million parameters, much fewer than the 140 million parameters used in VGG. This network was the 2014 imagenet winner.

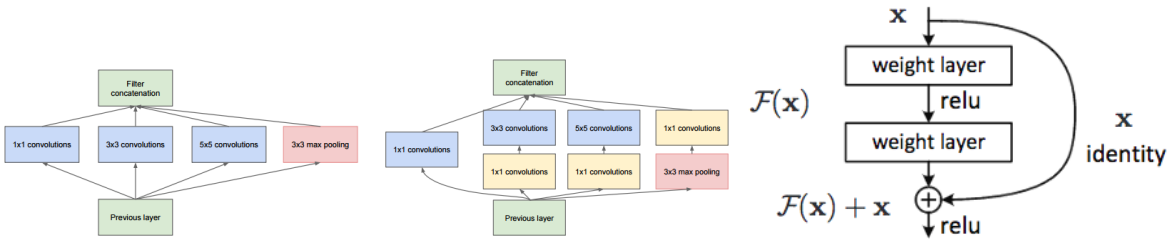


Figure 2.3: *(a) The inception module, showing the different filters being applied and then the output being concatenated, as well as a version (b) using 1x1 convolutions to do compress the inputs to the convolutions. (c) Example of a residual connection [6]*

2.1.6 ResNet

Eventually, when too many layers are stacked the network is unable to learn. This is not due to overfitting, as even the training set error increases. This happens even when layer normalization is used to prevent the exploding gradient problem.

To train very deep networks residual connections are added between convolution layers. A residual connection is an element wise addition of previous layer output to the current layer. If we think of the desired mapping as $\mathcal{H}(x)$ the we can learn $\mathcal{F}(x) := \mathcal{H}(x) - x$ instead. Basically the network will learn a mapping that will undo the element wise addition, from the optimal function. Their intuition is that it is easier to optimize the residual mapping than to optimize the original mapping. Also, this network has the benefit of being able to learn to skip convolution layer entirely, by forcing the weights to go to 0.

When the residual mapping goes to a layer with lower spatial dimension, the residual connection is applied to the input of stride two. If there is a larger number of features, the residual connection will learn a projection to the higher number of features, using 1x1 convolution. They experimented with using networks of depth of up to 1202, which they were able to optimize without difficulty, but the 152 layer network performed best, and won imagenet 2015.

There were other attempts at creating types of skip connections, for example highway networks, where a gating function is learned on the residual connections. A parameter is learned, which can close the gate and turn the network into one without skip connections. These are not as effective as the residual connections because they do not show increased accuracy as the number of layers goes over 100[21].

2.1.7 Inception V3

This model is an improvement over the original GoogleLeNet paper, and incorporates more recent architecture improvements:

- **Factorizing convolutions** In this paper they took the factorization of convolutions in VGG one step further. They factored their 5x5 convolutions into 3x3, but also realized that it is possible to factor the 3x3 convolutions into a 1x3 convolution followed by a 3x1 convolution. Using this factorization of the 3x3 convolution reduces the number of parameters from $9C^2$ to $3C^2 + 3C^2 = 6C^2$, while keeping the same input field. While this type of factorization of convolution kernels preserve the input field, this is actually a totally different network, with an extra layer of non-linearity. This being said, in practice this factorization tends to work well when applied on medium sized feature maps (12x12 - 20x20), but not on the raw data or very early layers.

- **Batch Normalization**[7] - During training, the distribution of each of the layers inputs change because of the parameters of the previous layers changing. Batch Normalization normalizes the layer inputs, per each batch in training. Batch normalization allows for faster learning rates, allows for the use of saturating non-linearities, and makes networks more robust to bad weight in initializations. At each step in mini-batch gradient descent, each layer has a d-dimensional input $x = (x^(1), ..., x^(d))$, and each dimension is normalized independently to:

$$\hat{x^{(k)}} = \frac{x^{(k)} = E[x^(k)]}{\sqrt{Var[x^{(k)}]}} \tag{2.1}$$

  The expectation and variance are computed for each batch during training, but at test time they are computed over the entire training set. By normalizing the inputs this can limit what the layer can represent, so they also introduce learned parameters into the normalization $\gamma^{(k)}, \beta^{(k)}$ to scale and shift it:

$$y^{(k)} = \hat{x^{(k)}}\gamma^{(k)} + \beta^{(k)} \tag{2.2}$$

  This means that the normalization can learn the identity transform if this is more useful, but adds two extra parameters to be learned after every activation.

It has also been shown that combining inception modules with residual connections (Inception V4)[22] can be useful.

2.1.8 Overview

Based on the results seen in these papers, we can see some general principles for designing neural network architectures:

- **ReLU** - ReLU is a standard choice to fight the vanishing gradient problem.
- **Dropout** - Its usefulness seems to be dependent on the number of redundant weights. For fully connected networks a high dropout amount (about .5), should be used, while for smaller, factored convolution networks this should be much lower.
- **Factored Convolutions** - Factor convolutions into 1x3, 3x1 for all medium layers, for early layers it is better to use convolutions that are not fully factored, such as 3x3.
- **Batch Normalization** - Generally useful, but sometimes can be unnecessary or hurt some complex architectures.
- **Inception Module** - These are useful, and most helpful when applied in the middle of a CNN.
- **Residual Connections** - For deeper networks these are useful, but because of the simple nature of our problem, our network was not so deep, so these are not used.

2.2 Related Work

2.2.1 Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images

In this paper, [1] the goal was to segment neuronal structures, on 50-nanometer (512x512 pixel) greyscale slices. This is achieved by training a binary classifier to predict the label of each individual pixel. They used a standard 11 layer architecture with convolution, max pool, and fully connected layers, treating this as normal classification problem.

The training data was fed into the classifier as a subsection of size 95x95 pixels, centered at a selected pixel. To generate the training set, they used all of the neuron pixels, as well as a random sample of the same size of non-neuron pixels. Not all non-neuron pixels are

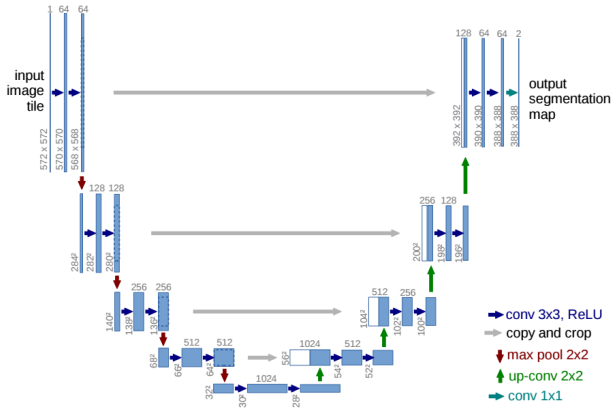


Figure 2.4: *The U-Net architecture[16] (from U-Net paper). Each blue box shows the a multi-channel feature map, with the number of channels at the top of each box, and the spatial size on the side. White boxes show copied feature maps.*

included because the dataset is heavily biased towards this class. Augmentation of random 90 degree rotations, and mirroring is used.

At test time the classifier is applied to all pixels in the image, generating a map of neuron probability. This map was then spatially smoothed by a 2-pixel-radius median filter, and thresholded to obtain the pixel classifications. The downside to this method is that it must be applied to all pixels in the image, and this is extremely computationally expensive. Future work uses fully convolutional architectures to eliminate this redundant computation.

2.2.2 U-Net

The U-Net [16] is a network designed for the binary segmentation of neuronal structures and other tissue. It takes as input an image, and outputs another image with the classification of each pixel.

The architecture is shown in fig 2.4. They use a fully convolutional design, meaning that no fully connected layers are used. This allows the network to process images of any resolution. This network is consists of a number of a number of convolution layers which transform the image to a deep but spatially small representation using 3x3 convolutions and max pooling. This is transformed back into an image by using 3x3 convolutions, and an 2x2 up-convolution layers. This is similar to an autoencoder, but it also has skip

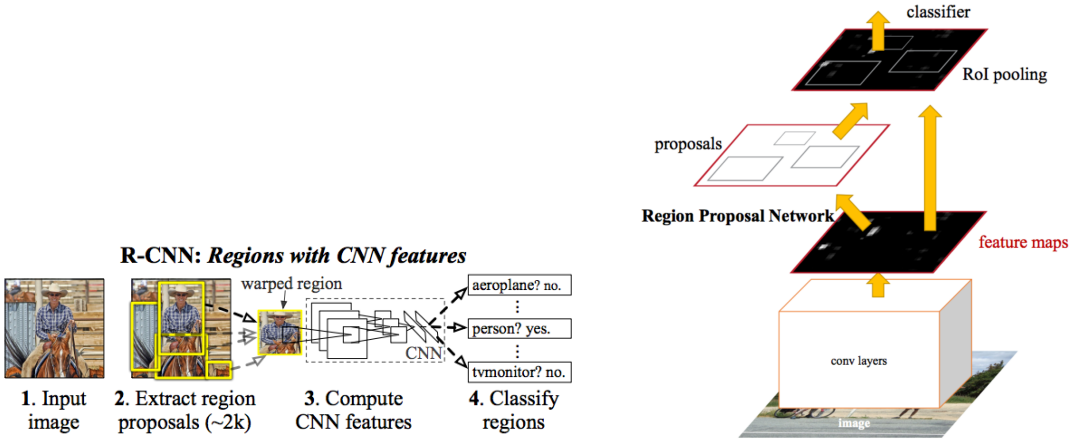


Figure 2.5: *(a)The RCNN architecture. (b) Faster RCNN.*

connections from the encoder layers to the decoder layers of the same size.

This network is trained using a pixel wise cross-entropy loss, with a large amount of data augmentation. They found using rotations, translations, and especially random elastic deformations was useful to make up for their small training set size.

2.2.3 Faster RCNN

Faster RCNN[15] is an improvement on the RCNN[5] and FAST-RCNN[4] models. In the RCNN family of models, they perform image segmentation by outputting bounding boxes for all objects in an image, along with their classification. They do this in two steps, first generating region proposals, and then classifying these regions

RCNN used a classical computer vision algorithm, selective search [25] to output a set of possible bounding boxes for each image. The details of this are beyond the scope of this paper, but the idea is that the algorithm will cluster adjacent pixels based on similarity, and then group these together as possible objects. This can output many more bounding boxes than there are true objects.

The proposal regions are then fed through a pre-trained CNN similar to AlexNet, and the extracted features are fed to an SVM for classification. After this, linear regression is used to improve the bounding box coordinates. This is shown in figure 2.5(a).

FAST RCNN simplifies this process by changing the SVM classifier and the linear

bounding box regression to use fully connected layers, both computed directly from the extracted CNN features. They also realized that there was there was a lot of duplicated computation from computing the features for each proposal region should not be computed multiple times, so they reuse the computed features for each proposal region.

Faster RCNN again improves on this design by using the CNN to generate the region proposals as well as do the classification.

They train a region proposal network, which is applied across the image, and at each location outputs k potential bounding boxes, as well as a score for how likely this is to be correct. The second half of the network (classification) is similar to Fast RCNN. This is shown in figure 2.5 (b)

2.2.4 Countception

In[2], they do object counting and localization, using a regression based method. They use an inception style network taking an input of 32x32 pixels, and the network outputs a count of all the cells within that window. For this problem they worked with only one class of objects.

For classification and localization, this network is applied over the entire image. They found that doing redundant counting was most effective, and so the network is applied to the input image with a stride of 1. This is then rescaled by a factor of $(\frac{receptive field height}{stride})^2$ This will create a count map, which can be integrated over to give the total number of cells.

A drawback of this method is that it does not give precise localizations of each cell, and so the output is less interpretable than a method that gives exact localizations. This network is also able to be run in a fully convolutional way, so prediction can be run efficiently.

Chapter 3

Data

The data consists of breast tissue samples from 64 patients post neoadjuvant treatment, and was scanned and prepared at the Department of Anatomic Pathology at Sunnybrook hospital, Toronto. These are H&E stained slides taken at 20x magnification. We have subsections of the full slides of size about 500x500 pixels. These subsections were selected so to contain a mixture of the three classes, and so can't be considered representative of the full slide.

Each of the slides are labeled by an expert pathologist, using Sedeen Viewer (Pathcore, Toronto, Canada), with a point placed at the center of each cell nuclei, and labeled as either Lymphocyte, Normal Epithelial or Malignant Epithelial. These labels came in the format of a separate xml file, with a list of the points and their corresponding class. In figure 3.1(a) we see an example of this, with the cell centers indicated as dots colored according to their class.

We will refer to three different levels of data, as explained below:

1. **Slides** - Raw 20x magnification H&E stained slides. These are very high resolution image, and we did not have access to this for the project, and so these will not be discussed further.

2. **Patches** - Annotated subsections of the full slides, generally about 500x500 pixels. We have 154 of these with the nuclei labeled, of which 148 had acceptable labels (figure 3.1(a))

3. **Tiles** - Extracted nuclei from the image patches. There is one tile corresponding to each nuclei, as well as some corresponding to non-nuclei. We experimented with

taking different sized regions around the nuclei, from 32 pixels to 256. The are not exactly one nuclei, they are selected based on the nuclei at the center of the image. A larger sized tile, like 256x256 may contain multiple nuclei, but will always be centered on the one of interest. Each tile will have a class of malignant, normal, lymphocyte or non-nuclei. The input image is zero padded to ensure that the same sized image will be produced for all cells, even if the nuclei center is close to the edge of the slide. (figure 3.1(b))

While 148 patches is a small number of samples, in total there were over 27000 nuclei annotated by the doctor, which is enough to train deep learning models without using transfer learning. This being said, the nuclei in each of the 148 patches are highly correlated, so this is different than a dataset of 27000 randomly sampled nuclei. There also a few issues with the data:

1. **Unlabeled nuclei** - Not all of the nuclei the patches are labeled. If the doctor was unable to classify a cell, its location is not included in the labels. This means that any attempt at localization will give many false positives identified.

2. **Mislabeled data** - Of the 154 images provided, 6 had problems where there were hundreds of incorrect nuclei identified. Because of the large number of nuclei in our dataset these could simply be excluded from our data.

3. **Unrepresentative patches** - The patches were not selected randomly from the full slide images. The pathologist selected these subsections so that there would be a mixture of the three classes of nuclei. Any models trained on this dataset could have worse performance if applied on image patches that were randomly selected, because they may have a different class distribution of nuclei.

The data is divided into 3 sets of of size 60% (train), 25% (test) 15% (validation). These sets were used in the usual way, with the models trained on the training set, and validation set performance used for hyperparameter tuning. The test set is used to evaluate the performance of the final models.

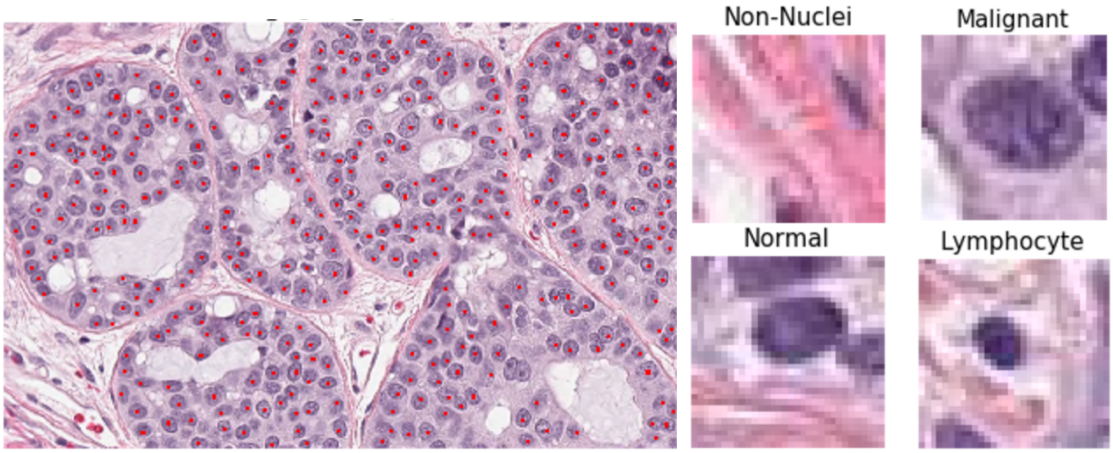


Figure 3.1: *(a) Subsection of H&E stained slide, taken at 20x magnification, (image patch) showing the malignant nuclei labeled in red (b) Examples of segmented nuclei (tiles) from the image patch. This is with a 32x32 pixel (*16 µm *x* 16 µm*) region*

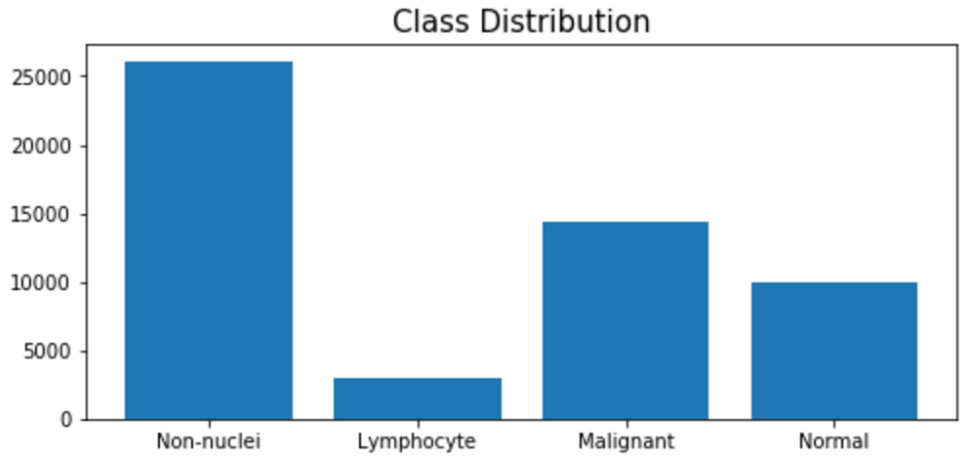


Figure 3.2: *(a) Distributions of the labeled nuclei, showing how malignant nuclei are over represented. The non-nuclei class is created by randomly sampling non-nuclei tiles, and has the same number as the three nuclei classes combined.*

Chapter 4

Methodology

This problem is different from any of the problems seen in the literature, so it can be difficult to directly apply these methods. We decided on three methods based on: **Regression**, **U-Net**, and **Classifier**, because they have been successful with related problems, as shown in the literature review.

The RCNN family of approaches are omitted entirely from this study. We felt that there was too large a discrepancy between the use case of these approaches, and the problem we have. This approach is used for object detection in the case when there are relatively few large objects in a scene, and we have images with bounding box labels. Although this is a popular approach, in our problem none of these conditions are met, and so it is unlikely to be useful.

We found that the classifier based method performed best, and the other two methods were unable to generate decent results on this problem. Below we will expand these methods, particularly how they were modified to suit this particular problem:

4.1 Classifier Based Method

4.1.1 Outline

CNNs have been shown to be very effective in image recognition, and so for this method we will take advantage of this by using a classifier to perform localization. The outline of our approach is:

1. **Tile level classifier** - Train a classifier on the tiles to detect if an image is centered on a normal, malignant or lymphocyte nuclei, or else not on any nuclei.

2. **Heatmaps** - Apply the classifier to all the pixels in the image, outputting a 4 dimensional heatmap of the probability of each nuclei class.

3. **Non-maximum suppression** - Use non-maximum suppression to convert these heatmaps into a set of estimated cell locations.

4.1.2 Tile level classifier

We first ignored the localization problem, looking only at classification of the nuclei, because this is a straightforward problem for deep learning. To look at cell classification alone, we created a data set consisting of nuclei of each of the three classes, and non-nuclei. Tiles of non-cell tissues are created by randomly selecting points that were at least 5 µm (10 pixels) from any other cell. This will enable the trained classifier to do localization as well. We selected 183 non-nuclei tiles from each slide, so there were 27,000 nuclei and non-nuclei examples. The class distribution was skewed towards malignant and normal cells, as shown in figure 3.2

Architecture

This classifier has a different goal from the standard architectures designed for imagenet because we are aiming to predict the class of the nuclei at the center of the image, rather than giving a class for the entire image. We need to design a classifier that will only detect the center of the nuclei, and not respond when it is on the edge of one. This is especially important when using large tile sizes, because there will be many more nuclei in each slide, but we still must only classify the one in the center.

It common to use max pooling layers in CNNs, but we did not, because max pooling introduces translation invariance. There are nuclei that are close together, and so any significant translation invariance could prevent the models from identifying the center correctly. These observations was backed up by our empirical results, which found that accuracy with max pool was significantly lower. We also designed the non-nucleus tiles so that the classifier would explicitly learn to make its classification based only on the center of the nucleus. Non-nucleus tiles were considered as those that didn't contain a nucleus in the center of the tile, rather than not containing a nucleus anywhere in the tile.

To choose the best architecture for this problem gird search is used, varying depth and number of filters and nodes in each layer. We also investigated using the inception module [23], and we found this to be useful. We settled on using a network with two 3x3 convolutions, followed by two inception modules. Some other standard architecture choices are included such as **Batchnorm**, **Dropout** and **ReLU**, as explained in the background section. Instead of using fully connected layers, we designed this network to be fully convolutional (figure 4.1).

For the final layer, instead of fully connected layers we use a convolution kernel that spans the full width of the feature map, applied with no zero padding. The output from this will be four convolutional feature maps, with spatial dimension of 1x1. These correspond to the class probabilities. Because this network is fully convolutional, it can accept varying sized input, with this corresponding to an increased output spatial dimension. The main benefit to this design is that it allows for fast predictions. With a network using fully connected layers, it can only accept a fixed sized input, and so this network would have to be iteratively applied to every pixel in the input image. This is extremely ineffecient because the convolutional layers will compute each input pixel many times, which is entirely duplicated computation.

Fully convolutional architectures are chosen over networks containing fully connected layers, because this allows for fast inference at test time. The architecture will have a set of convolutional layers, as in any normal CNN, but instead of a fully connected layer at the end, we use a convolution with a kernel size that is as wide as the feature map with no padding. This is actually equivalent to a fully connected layer, but it has the benefit of accepting variable input sizes. When the network is tested on a larger image than was trained on, it will output multiple probabilities corresponding to the spatial locations of the image.

Data augmentation

Data augmentation is used to artificially increase the size of the training set by randomly applying transformations to each batch of data before it is fed into the network. If reasonable transformations are selected, this should help to reduce overfitting. Our data augmentation process was quite standard, but with a few adjustments to prevent translation invariance:

1. **Rotations and flips** - Our data does not come in any particular orientation, and so the classification of a cell should be invariant to any rotation or flip that is applied.

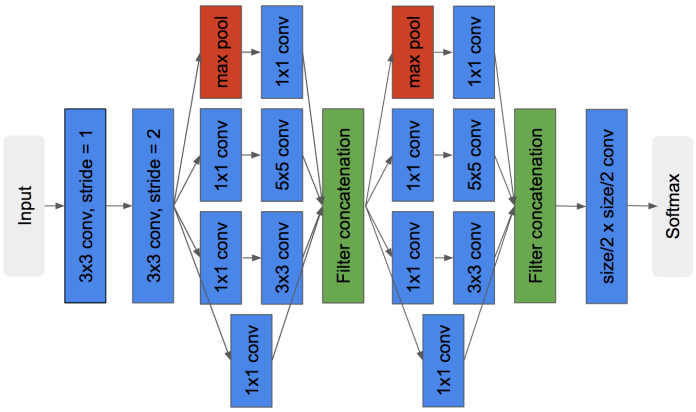


Figure 4.1: *The architecture of our best performing CNN. This is based off of the inception architecture [23], but with fewer layers and using a large convolution kernel instead of global average pooling at the last layer.*

For this reason we chose to augment the training data using random rotations and flips.

2. **Translation** - We will also randomly translate images by a small amount. The idea is that the doctor's annotation is probably not always in the exact center of the nucleus, and so we can augment the data by slightly moving the cell off center in the image. The key with this is to put the level of translation only as big as the variance in doctor annotation. Any larger and the classifier will not properly learn to focus on the nuclei in the center of the image.

Optimization is done using the Adam optimizer with a learning rate of .0001. The models generally took about 75 epochs for the model to converge. All models were implemented using Keras [3] with the tensorflow backend.

The best performing CNN architecture used 2 convolution layers, followed by two inception modules [23]. This was designed in a fully convolutional way to allow for fast prediction.

4.1.3 Heatmaps

We can apply the tile classifier to any pixel in the input image to give the probability of a nucleus being centered at that location, as well as the class probabilities of that nucleus.

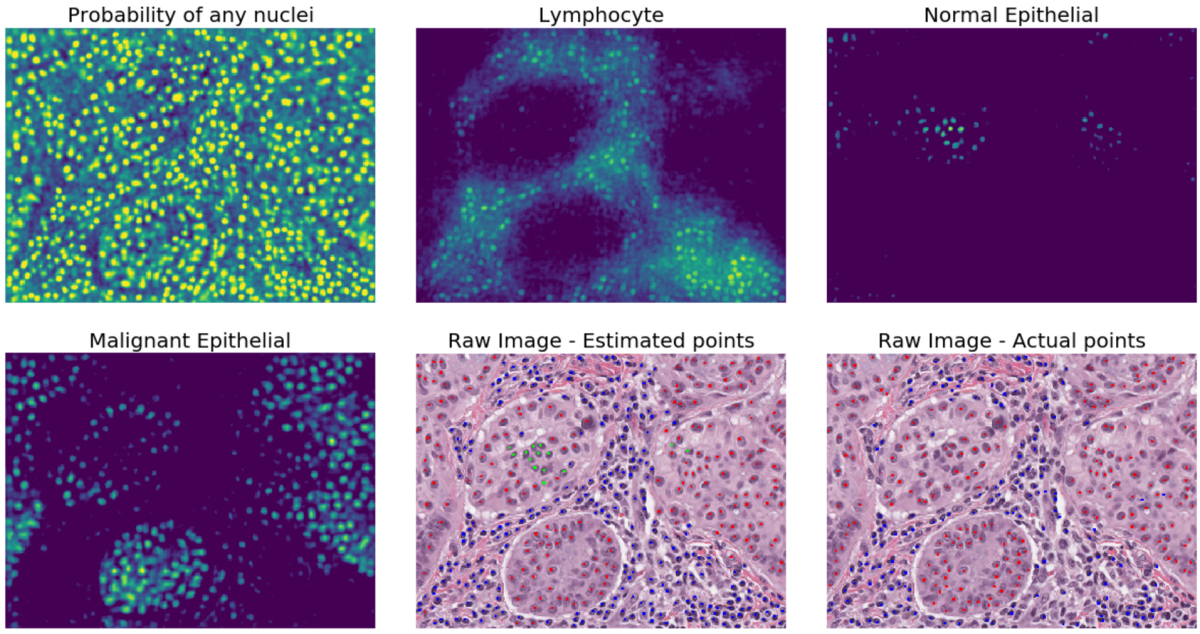


Figure 4.2: *Example heatmaps produced by the classifier, and the predictions on the raw image. Lymphocyte, normal and malignant epithelial nuclei are shown as blue, green and red dots respectively.*

By applying the classifier across the entire image, we will output 4 heatmaps corresponding to each of the cell probabilities as well as the probability of no cell. By using the fully convolutional design, we are able to output the heatmap in one pass of the CNN, by using the full patch as its input. This is shown in figure 4.2.

This is the reason that the non-cell data was added as a class for the training the classifier. Now we can explicitly see the probability of their being a cell at any given location the classifier is applied by looking at (1-probability of no cell). This should produce heatmaps that indicate probability of a given pixel being the center of a nuceli.

4.1.4 Non-Maximum Suppression

We need a method to convert these four heatmaps into a set of cell locations and classes. To do this we use non-maximum suppression. This relies on the observation that the points on the cell probability heatmap of the highest probability are the most likely to be the center of the cells. For non-maximum suppression we iterate the following procedure:

while(max probability > cutoff):

1. find maximum pixel value
2. set all pixels within radius r of maximum to 0

In this procedure we iteratively find the pixel with highest cell probability, and then set all points withing a radius r of this pixel to 0. This is repeated until there is no more pixels above a certain threshold, $cutoff$. At each step, the classification of that point is selected as the nucleus class with highest probability from the heatmap.

4.1.5 Choosing the radius

Our aim is to find the center of the nuclei, and then set all other points inside the cell to 0, to ensure that we only give one location per nuclei. If all nuclei were uniform size, this would be easy to implement. We would find the radius of each nuclei, and then we would be quite sure non max suppression would work provided the heat map had given good predictions for the nuclei center. The problem is that we know nuclei are different sizes, with the malignant nuclei being especially variable.

When picking the radius we are balancing two opposing objectives, we want the radius to be wide enough so we don't generate more than one prediction per nuclei, but we also need it to be small enough that we will not suppress a neighboring cell's nucleus. To find the optimal value, we looked at the distribution of the distance from a nuclei to its closest neighbor, shown in figure 4.3.

We see that almost all cells are more than 8 pixels apart. This gives an indication of how wide to set the radius, because if we make it significantly wider than 8 we will start suppressing the cells neighbors. By plotting the trade of between true positives and false positives according to the cutoff (figure 4.4(a), we can empirically we see that the best radius is either 5 µm(10 pixels) or 6 µm(12 pixels). We will choose 10 for the further experiments.

4.1.6 Variable Sized Radius

We also investigated using different radius dependent on the predicted class of the nuclei, because the nuclei have different sizes. The idea was that you should use a larger radius

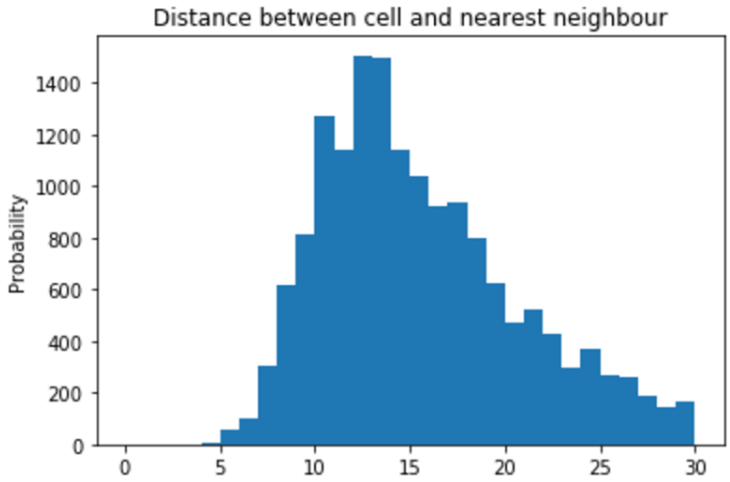


Figure 4.3: *Training set distribution of distance from all nuclei to their nearest neighbor*

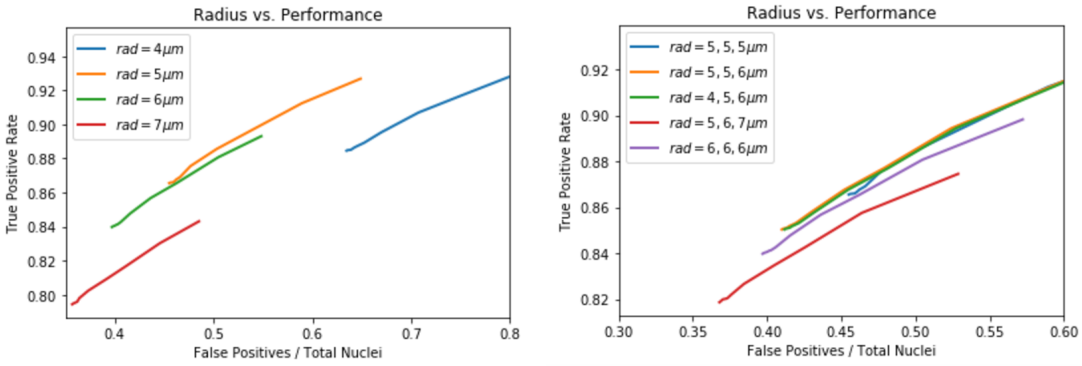


Figure 4.4: *(a) True positives vs. false positives trade off while varying the radius.* 5 µm *or* 6 µm *showed the best result. (b) Results from using a variable sized radius. The legend shows the radius used for lymphocyte, normal and malignant nuclei*

for the larger malignant cells, and a smaller one for the lymphocyte cells. As shown in figure 4.4 (b), this variable radius does not seem to give better results.

4.2 U-Net Based Methods

The U-Net model was modified to be used on this problem, with the main changes being using a smaller network than the original paper, and having to construct artificial pixel labeled data. We did not find this approach was effective.

4.2.1 Data

The U-Net model is used for cases where there is pixel level labeled data. Our dataset only has point annotations, and so there is no direct way to apply this model. To deal with this issue we create fake pixel level labels for each class of nuclei. This is done by assigning all pixels of distance less than 10 from the pathologists annotation as being a nucleus of that class. This is clearly a false assumption, as nuclei of different classes are different sizes, and we should not assume they are round.

This creates a set of 4 images from each patch. They correspond to to a different classes of nuclei, and and the no-nuclei class.

4.2.2 Architecture

This network is based closely off of the architecture used in the U-Net paper, figure 2.4, but we used a smaller architecture. The original U-Net paper uses 4 max pooling layers and 23 convolution layers for down sampling, while here only two max pool layers, and 15 convolution layers are used. We also added some standard methods to help improve the performance, such as batch normalization and dropout of 10%.

Data augmentation

As in the classification model, data augmentation is used to artificially increase the size of the training set. The difference here is that here translation invariance is not an issue, and so more data augmentation can be used.

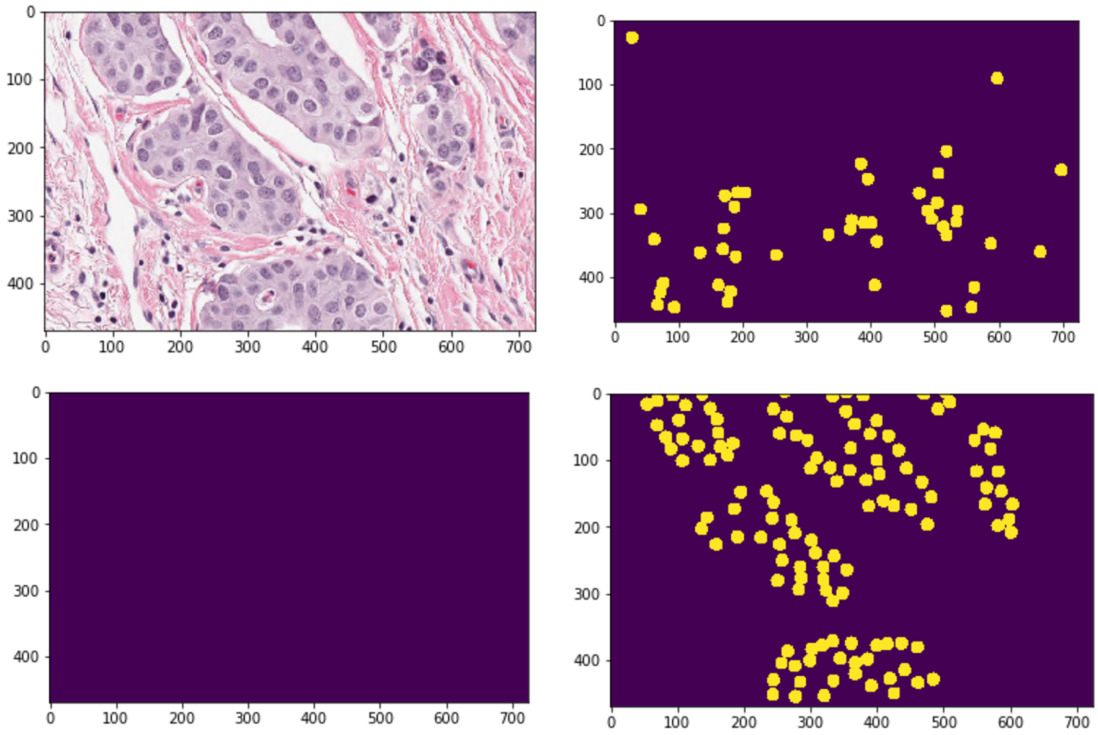


Figure 4.5: *Examples of an patch, and the corresponding labels generated for the u-net model. The top right shows the lymphocyte nuclei, while the bottom right shows the malignant nuclei. There are no labeled normal nuclei in this slide*

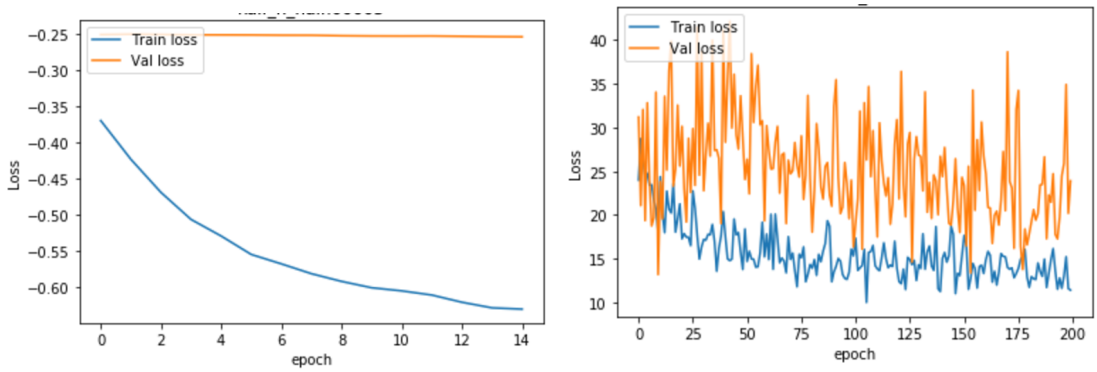


Figure 4.6: *(a) The U-net model badly overfitting the training set. (b) The regression model overfitting*

1. **Rotations and flips** - Our data should be invariant to rotation and flips, so both of these are used for augmentation.
2. **Crops** - We randomly will select a subsection of the patch of size 320x368 to be fed into the classier.

Optimization is done using the Adam optimizer with a learning rate of .0001. We found these model had a strong tendency to badly overfit the training set. (figure 4.6)

4.3 Regression Based Methods

The regression based methods are based off the idea from the Countception[26] paper. We treat this as a regression problem, where given an image, our goal is to output the number of cells of each class contained in it.

In this paper they focused on the case where there was only one class of cell, and the cells were labeled with point annotations. Our data also has point annotations, and so we can use a similar approach, except instead of the network outputting one value, we will create a network outputting 3 values for each nuclei class.

4.3.1 Data

The input data for this model is a random subsection of the image patch, and the labels are the counts of each cell type contained within this section. The number of nuclei within

this subsection is determined by the number of annotations contained within it. A nucleus will be counted even if it is not fully included in the section.

4.3.2 Architecture

We tested architectures using 3-6 convolution layers, and 2 fully connected layers, using max-pool for downsampling. This is an architecture similar to those used for classification, with the change being that in final output layer the softmax is removed, and so each of the three nodes are used for regression, corresponding to the counts for each of the nuclei classes.

Data augmentation

As with the other models, in this case our data is not given in any particular orientation, and so should be invariant to rotation and flips. To ensure this happens and to augment the dataset we use random rotation and flips for augmentation.

Similarly to the other models, this was trained using the Adam optimizer, but with a learning rate of 0.00005.

Chapter 5

Results & Discussion

Of the three models tested, we found that the classifier based method was the only one to give acceptable results. Here we will review the results, and discuss some possible areas for improvement:

5.1 Results

Both the U-Net and the Regression based models performed quite poorly, with both of them badly overfitting the training set. This is apparent from the graphs in figure 4.6. Because there was so much difficulty to make these models learn anything, the performance was not investigated any further. The classifier based method had much better performance, so we will look into it's performance in detail.

For the classifier based method, the localization performance is difficult to evaluate, because there are many nuclei in the dataset that are not labeled. We are not sure on the number of unidentified nuclei, and so we can't be sure what a reasonable false positive rate is. In the test set we identified 86.2% of nuclei in the test set, with a 46.5% false positive rate.

Looking at the overall accuracy results (table 5.1) shows how using a larger receptive field improves classification performance. This happens up to about 192x192, after which we did not see an improvement in performance. This confirms our intuition that global information is useful for this classification problem. The nucleus should be fully contained in even the smallest receptive field (32x32), but performing classification using this information alone is not enough.

Table 5.1: Classification Accuracy vs. Receptive Field Size

Model	**Test Accuracy**
32x32 Inception	82.7%
64x64 Inception	93.7%
128x128 Inception	93.9%
192x192 Inception	94.6%

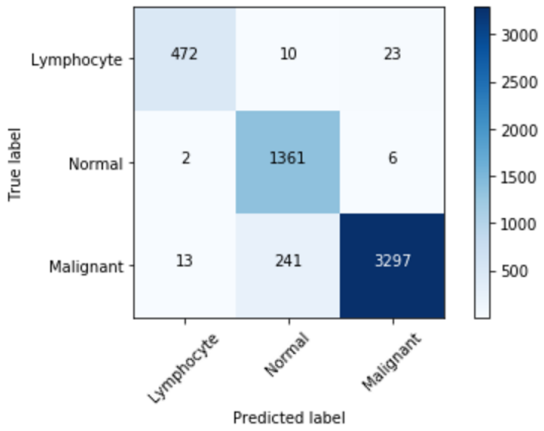


Figure 5.1: *The classification accuracy of the 192x192 inception model. The biggest issue is misclassifying malignant nuclei as normal.*

A more detailed comparison is also shown in figure 5.1, where we can see how the main issue is that some malignant cells are being misclassified as normal. The classification accuracy is computed using only the nuclei that were identified from non-maximum suppression.

5.2 Discussion

The U-Net model or regression models performed poorly, but this does not necessarily mean these models can not be useful for this problem. Probably the issue is with the way in which our data was labeled, which had with many unlabeled nuclei.

Because our data had many unlabeled nuclei, the regression based model will constantly be trying to learn based of of incorrect counts. The problems with the U-Net model are

Table 5.2: Test Set Classification Results

Class	**Accuracy**	**Sensitivity**	**Specificity**
Lymphocyte	99.1%	93.4%	99.6%
Benign Epithelial	95.2%	99.4%	93.8%
Malignant Epithelial	94.8%	92.8%	98.5%

even greater, because not only does it have the issue of missed nuclei, there is also the problem of incorrect pixel level labels. Given these concerns it is not surprising this model had difficulty learning.

We had attempted different methods to improve the classification accuracy, such as KNN, or applying a CNN to the generated heatmap, but these did not yield significant improvements. The best way to increase accuracy was by increasing the input size of the classifier, as can be seen from out results. This makes sense because in histology data there is more relevant information for classification than can be found by looking at the cell nuclei alone. Nearby tissue, or even the global layout of cells and tissue contains relevant information for both classifying and interpreting the slide.

5.3 Future Work

It is possible to further speed up the classifier using a smaller network to produce heatmaps for localization of the cells, and then applying a classifier using a larger input area to update the classification for the uncertain nuclei. The idea behind this is that local information is sufficient for localization of a cell, but we need more global information for classification. Because of this a good CNN for classification will be more computationally expensive than one useful only for localization, and we can save computation by only applying the expensive classifier to locations we know are nuclei.

For the localization problem, this paper focused on generating a heatmap that had high probability only at the center of the nucleus, enabling us to use a simple localization process like non-maximum suppression. Traditional segmentation methods could be used to augment this, as we didn't attempt made to separate cells based on their boundaries.

It would be more in interesting to try U-Net on properly pixel labeled data, or to try

the regression method on data where all the nuclei are labeled. The benefit of the classifier based method was that it did a reasonably good job of separating the localization and and classification tasks, so it was able to show decent results even with the mislabeled data. Better labeled data would also may allow us to better evaluate using variable radius for non-maximum suppression. This idea seems intuitively appealing, and we are hopeful it could be be shown to be useful.

References

[1] Dan Ciresan, Alessandro Giusti, Luca M. Gambardella, and Juergen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2843–2851. Curran Associates, Inc., 2012.

[2] Joseph Paul Cohen, Henry Z. Lo, and Yoshua Bengio. Count-ception: Counting by fully convolutional redundant counting. *CoRR*, abs/1703.08710, 2017.

[3] Keras Developers. Keras: Deep learning library for theano and tensorflow. https://keras.io/, 2016.

[4] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[5] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[8] Andrej Karpathy. cs231 lecture notes, 2016.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

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

[11] Yun Liu, Krishna Gadepalli, Mohammad Norouzi, George E Dahl, Timo Kohlberger, Aleksey Boyko, Subhashini Venugopalan, Aleksei Timofeev, Philip Q Nelson, Greg S Corrado, et al. Detecting cancer metastases on gigapixel pathology images. *arXiv preprint arXiv:1703.02442*, 2017.

[12] Mohammad Peikari, Sharon Salama, Sherineand Nofech-Mozes, and Anne Martel. Automatic cellularity assessment from post-treated breast surgical specimens. *preprint*, 2017.

[13] Florentia Peintinger, Bruno Sinn, Christos Hatzis, Constance Albarracin, Erinn Downs-Kelly, Jerzy Morkowski, Rebekah Gould, and W Fraser Symmans. Reproducibility of residual cancer burden for prognostic assessment of breast cancer after neoadjuvant chemotherapy. *Modern Pathology*, 28(7):913–920, 2015.

[14] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[16] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[17] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[18] Sunati Sahoo and Susan C. Lester. Pathology of breast carcinomas after neoadjuvant chemotherapy: An overview with recommendations on specimen processing and reporting. *Archives of Pathology & Laboratory Medicine*, 133(4):633–642, 2009. PMID: 19391665.

[19] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1605.06211, 2016.

[20] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[21] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.

[22] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.

[23] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *arXiv preprint arXiv:1512.00567*, 2015.

[24] A. M. Thompson and S. L. Moulder-Thompson. Neoadjuvant treatment of breast cancer. *Annals of Oncology*, 23(0):231, 2012.

[25] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[26] Weidi Xie, J. Alison Noble, and Andrew Zisserman. Microscopy cell counting and detection with fully convolutional regression networks. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 0(0):1–10, 0.

[27] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.