CELL NUCLEI SEGMENTATION

USING DEEP LEARNING TECHNIQUES

by

Rajendra K. C. Khatri

APPROVED BY SUPERVISORY COMMITTEE:

Yan Cao, Chair

Viswanath Ramakrishna

Mieczyslaw K. Dabkwoski

Yifei Lou

I want to dedicate this dissertation to my sons Pragyan and Bigyan.

CELL NUCLEI SEGMENTATION USING DEEP LEARNING TECHNIQUES

by

RAJENDRA K. C. KHATRI, MS

DISSERTATION

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY IN

MATHEMATICS

THE UNIVERSITY OF TEXAS AT DALLAS

August 2021

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor Dr. Yan Cao for her continuous support and guidance throughout my PhD journey. She has been very supportive and motivated me from the very beginning. I would also like to thank Dr. Viswanath Ramakrishna, Dr. Mieczyslaw K. Dabkowski, and Dr. Yifei Lou for their continuous guidance as committee members. I had opportunities to learn from all of my committee members. My profound appreciation and gratitude to my parents who have been my inspiration always. Special thanks to my wife for her caring and emotional support throughout the years.Last but not least, I would like to thank all my teachers and all my friends and relatives who have, directly and indirectly, helped me achieve this goal.

July 2021

CELL NUCLEI SEGMENTATION

USING DEEP LEARNING TECHNIQUES

Rajendra K. C. Khatri, PhD
The University of Texas at Dallas, 2021

Supervising Professor: Yan Cao, Chair

Pathological examination usually involves manual inspection of hematoxylin and eosin (H&E)-stained images, which is labor-intensive, prone to significant variations, and lacking reproducibility. One of the fundamental tasks to automate this process is to find all the cell nuclei in the H&E-stained images for further analysis. We attempt this problem using deep learning techniques. First, we introduce a semantic pixel-wise segmentation technique using dilated convolutions. We show that dilated convolutions are superior in extracting information from textured images. H&E-stained images are highly textured, which makes dilated convolutions an ideal technique to apply. Our dilated convolutional network (DCN) is constructed based on SegNet, a deep convolutional encoder-decoder architecture. Dilated convolution layers with increased dilation factors are used in the encoder to preserve image resolution. Dilated convolution layers with decreased dilation factors are used in the decoder to reduce gridding artifacts. Our DCN network was tested on synthetic data sets and a publicly available data set of H&E-stained images. We achieve better segmentation results than state-of-the-art. To further separate the instance of each cell nuclei, we adapt our DCN with a single shot multibox detector (SSD) and achieve promising results. Our methods are computationally efficient and can be run on a personal laptop computer. This work is the first step to-

wards using mathematical models to generate diagnostic inferences and providing clinically actionable knowledge to physicians and patients.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION [1]

When diagnosing and treating patients with cancer, the cellular distribution of tumors is of great interest. Such an analysis can be accomplished using hematoxylin and eosin (H&E)-stained images [7] of cellular tissue extracted via biopsy. Currently, the pathological examination has been performed manually by visual inspection of these images. However, this process is labor-intensive, prone to large variations, and lacking reproducibility in the diagnosis of a tumor. As a result, it is desired to design automated processes to study different aspects of cellular distribution. In particular, image segmentation (classifying each pixel as cell nuclei or background) is a crucial first step in enabling the visual study of the disease.

We aim to develop an automatic workflow to detect cell nuclei in cancerous tumors portrayed in digital renderings of the H&E-stained images. Our goal is to create a binary label for a given H&E-stained image, describing which pixels are part of cell nuclei (denoted "object" pixels) and which are not (denoted "background" pixels). Convolutional Neural Network (CNN) [30, 28, 51] has been recently demonstrated to be particularly effective in image recognition and classification. Advanced CNN methods such as the Fully Convolutional Neural Network (FCN) [35], U-Net [48] and SegNet [1], which are trained from end-to-end and pixel-to-pixel, have been very successful in semantic segmentation. At the same time, there are also developments on efficient implementations of traditional variational image segmentation models [33, 21].

Both global image context and spatial accuracy are crucial in cell nuclei segmentation. In CNNs, multi-scale contextual information is acquired by successive pooling, and subsampling layers which reduce the image resolution until a global prediction is obtained [28, 51]. To

[1] This chapter includes verbatim excerpts from
K C Khatri,R.; Caseria, B.; Lou, Y.; Xiao, G. AND Cao, Y. "Automatic Extraction of Cell Nuclei Using Dilated Convolutional Network", Inverse PROBLEMS AND Imaging, Vol. 15, No. 1, 27-40, 2021.

recover the lost resolution, repeated deconvolutions or unpooling are usually performed to obtain the full-resolution dense prediction [42, 35, 48, 1]. Dilated Convolutions [60, 4] is a promising technique to obtain large context information without the loss of resolution. We design a dilated convolutional neural network for cell nuclei segmentation based on the architecture of SegNet. To further separate the instance of each cell nuclei, we adapt our DCN with a single shot multibox detector (SSD) to detect each individual cell nucleus.

Our contributions are as follows. First, we study the pros and cons of dilated convolutions and suggest using them in applications to improve performance. We also propose a method to implement dilated convolutions with dilation factor 2^n through efficient conventional convolutions. Second, we design a dilated convolutional neural network DCN for cell nuclei segmentation, achieving better results than state-of-the-art. Third, based on DCN and the SSD framework, We also create a neural network DCNSSD for individual cell detection. Lastly, the proposed workflow is efficient enough to run on a personal laptop. This work is the first step towards using mathematical models to generate diagnostic inferences and providing clinically actionable knowledge to physicians and patients.

The rest of the thesis is organized as follows. We review the convolutional neural network in Chapter 2. We then review semantic segmentation and detail the proposed approach in Chapter 3. Data preparation and preprocessing such as color normalization, data augmentation is discussed in Chapter 4. The experiments on synthetic data and real H&E-stained images are conducted in Chapter 5. Instance segmentation of cell nuclei is discussed in Chapter 6. Finally, conclusions are given in Chapter 7.

CHAPTER 2

CONVOLUTIONAL NEURAL NETWORK

2.1 Convolution Operation

A convolutional neural network or CNN is a special kind of neural network that involves the convolution operation in at least one layer. CNN can be applied to many types of data, such as image data [30, 28, 51, 55] and time-series data [13]. CNN has been significantly helpful in many application which involves huge input data such as any computer vision task, mainly because of its sparse interaction between the layers, parameter sharing, and equivariant representation properties[10].

A convolution [54, 10] is defined as follows:

$$(x * w)(t) = \int_{-\infty}^{+\infty} x(a)w(t-a)da, \tag{2.1}$$

where x is the input signal and w is the filter. A 2-dimensional discrete convolution can be defined as following:

$$y(m,n) = x * w(m,n) = \sum_{i=1}^{M}\sum_{j=1}^{N} x(m-i, n-j)w(i,j), \tag{2.2}$$

where $y(m,n)$ is the output of the convolution, $x(m,n)$ is the input, $w(i,j)$ is the filter with size $M \times N$. A slightly different equation than convolution known as cross-correlation is defined as:

$$y(m,n) = x * w(m,n) = \sum_{i=1}^{M}\sum_{j=1}^{N} x(m+i, n+j)w(i,j). \tag{2.3}$$

In the convolution operation, we flip the rows and columns of the kernels, multiply the input signal with the kernel element-wise and then sum it up to get the value. In contrast, cross-correlation computation is more straightforward as we don't need to flip the rows and columns of the kernel. In deep learning, people use the term convolution for cross-correlation.

Assume we have an input image x and a kernel w of size $k \times k$. To do convolution, we place $k \times k$ windows over the image starting from the top left corner, do element-wise multiplication at overlapped pixels and sum the total. This will gives the output at the center pixel of the overlapped area. We slide the window first on the right and later slide it down. The window has to overlap the region of the input image. Hence the output image will be smaller than the input image if there is no padding of the input image. The example on Figure 2.1 shows one convolution operation:

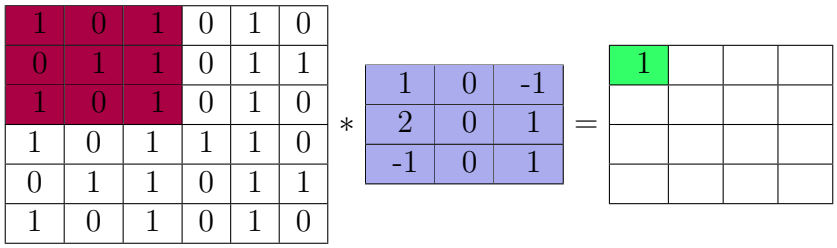

Figure 2.1: Convolution of an input image of size 6×6 with a filter of size 3×3 with stride one and no padding results in the matrix of size 4×4 on the right.

In general, for an input image of size $m \times n$ convolving with kernel of size $k \times k$ has an ouput dimension :

$$(m - k + 1) \times (n - k + 1) \tag{2.4}$$

2.1.1 Filter

The output of the convolution is also called the feature map. The 3×3 matrix in 2.1 is a kernel, also known as filter. Filters in convolution operation extract the features from the input image, such as edges, patterns, etc. They can also be used for smoothing, sharpening, enhancing, or blurring an input image. Let us have a look at some examples of the filters:

- Vertical edge detector:

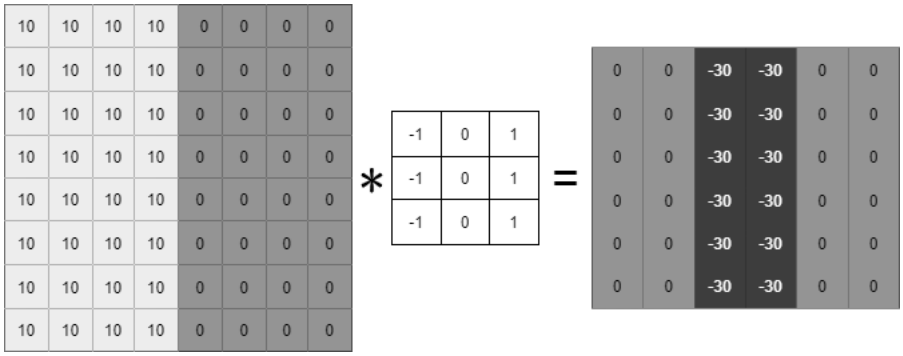

Figure 2.2: Example of vertical edge detection. The big matrix on the left shows the input image, the small matrix on the left shows the filter for vertical edge detection. The matrix on the right shows the output edge detection.

- Horizontal edge detector:

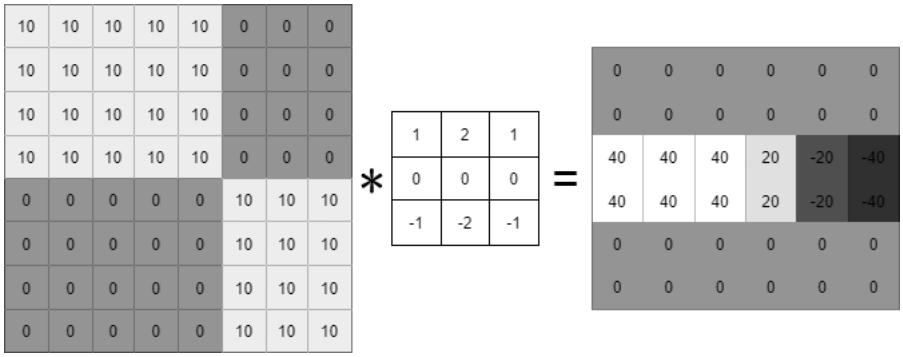

Figure 2.3: Example of horizontal edge detection. The big matrix on the left shows the input image, the small matrix on the left shows the filter for horizontal edge detection. The matrix on the right shows the output edge detection.

2.1.2 Strides and Padding

Strides and padding are the hyperparameters. Padding means adding extra pixels around the input image. In the convolution example above, a 6×6 input image convolving with a 3×3 filter results in a 4×4 image. While doing so, we lose the spatial dimension of the image. In addition, the filter overlaps with the border pixel only once, whereas it overlaps the middle pixels multiple times. So it may not quite nicely extract the features from the

border though there may be lots of information in the border. To avoid such drawbacks, we have to add extra pixels around the image, known as padding.

If we have an input image dimension of $m \times n$ and a kernel of size $k \times k$ and the padding is p (adding p pixels around the border of an image) then the size of the input image will become $(m + 2p) \times (n + 2p)$. Eventually, the output size will be :

$$(m + 2p - k + 1) \times (n + 2p - k + 1) \tag{2.5}$$

For the output size to be equal to the input size, we must have:

$$m + 2p - k + 1 = m \Longrightarrow p = \frac{k-1}{2}. \tag{2.6}$$

For the example above with an input image of size 6×6 and a kernel of size 3×3, to make the output image the same size as the input image, we need $p = \frac{k-1}{2} = \frac{3-1}{2} = 1$ i.e. 1 pixel padding around the boarder of the input image. Note that the padding parameter $p = 0$ means no padding. There are two usual strategies for the padding, one is valid padding which means no padding and another is "same" padding in which output size should be same as input.

Figure 2.4 shows an example of padding with one unit of zeros around the image boarder: Strides is another hyperparameter in convolution operation which decides the shift amount of the sliding filter. If the filter shifts one pixel at a time during the convolution operation, then the stride is 1 unit. If we choose stride 2 in the 2.1, the output size will be 2×2, compared to a 4×4 with stride 1. Stride 2 is often used in pooling operations.

Consider an input image of size $m \times n$, a kernel of size $k \times k$, padding $= p$ and stride $= s$, then the output will have the dimension:

$$\left\lfloor \frac{m + 2p - k}{s} + 1 \right\rfloor \times \left\lfloor \frac{n + 2p - k}{s} + 1 \right\rfloor \tag{2.7}$$

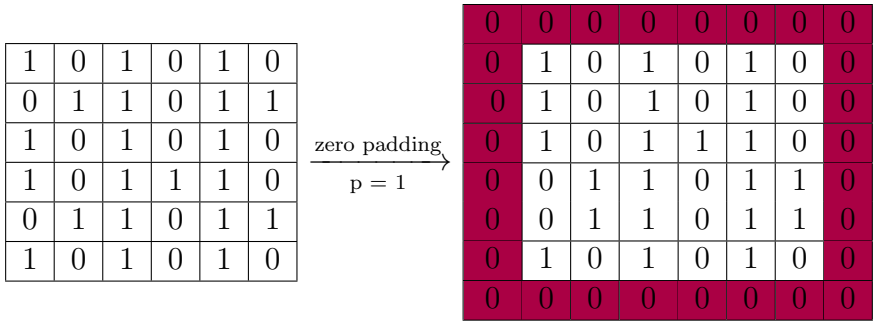

Figure 2.4: An example of padding with one unit of zeros around the image border.

2.1.3 Types of padding

Padding is a useful tool in CNNs, usually to get the output of the same size as the input. There are various types of padding described below[54]:

- Zero padding

  Adding zeros around the edges of the image is called zero paddings. Adding one layer padding around the image increases the height and width of the image by two. By default, whenever we say padding, it is zero padding. See Figure 2.4 for an example.

- Constant padding

  Instead of padding by zeros, constant padding is done by padding user defined constant value.

- Reflection padding

  Padding is done by reflecting or mirroring the pixel values directly in the opposite direction of the edges. See Figure 2.5 for an example.

- Symmetric padding

  Symmetric padding looks similar to reflection padding. Rather than reflecting like a mirror, you first take a copy, and then mirror it. See Figure 2.6 for an example.

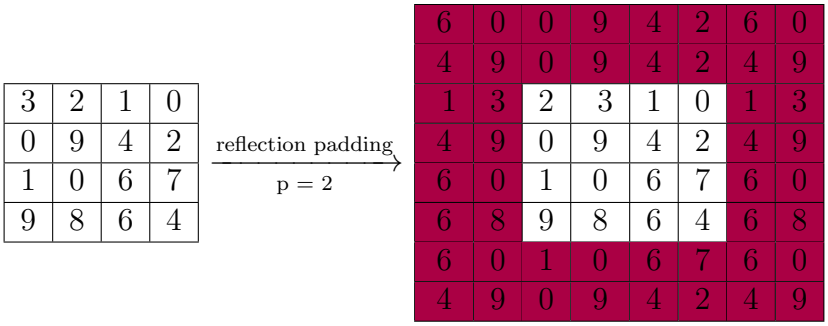

Figure 2.5: An example of reflection padding of an image.

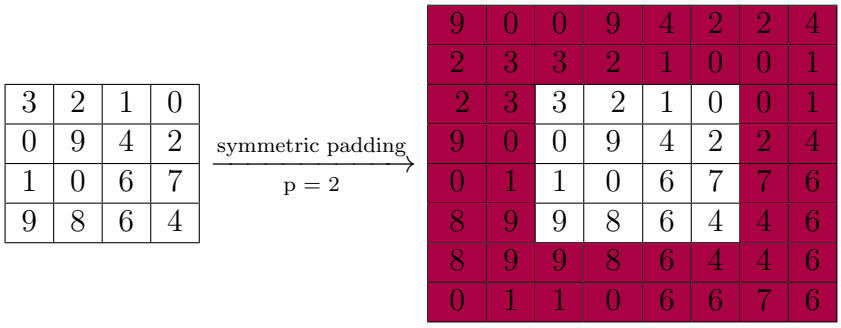

Figure 2.6: An example of symmetric padding of an image.

2.2 Pooling Layer

The pooling layer downsamples the input image by keeping the only significant pixels of an image. Thus, it helps to speed up the computation, and feature detection becomes more robust. The pooling layer has no parameter to learn. Instead, we choose the size of the filter and the stride. The typical choice of filter size is 2×2 with stride 2, which has the effect of shrinking the representation by a factor of 2.

- Max Pooling

  Max pooling[61, 40] is done by extracting the maximum pixel value of the corresponding pooling windows. The following diagram illustrate the max pooling operation in the 4×4 input image with 2×2 pooling and stride of two. See Figure 2.7 for an example.

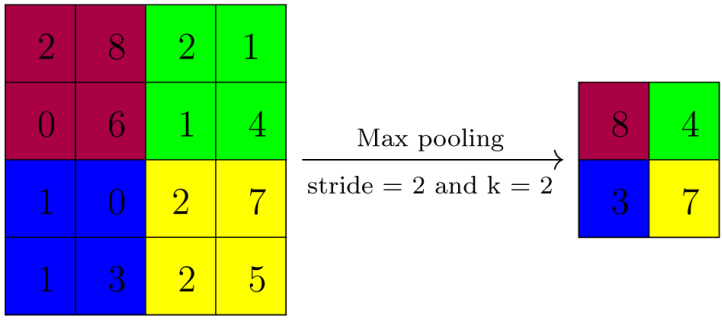

Figure 2.7: An example of max pooling operation.

- Average Pooling

  Average pooling is done by extracting the average pixel values of the corresponding pooling windows. The following diagram illustrate the average pooling operation in the 4×4 input image with 2×2 pooling and stride of two. This operation halved the input image to 2×2. See Figure 2.8 for an example.

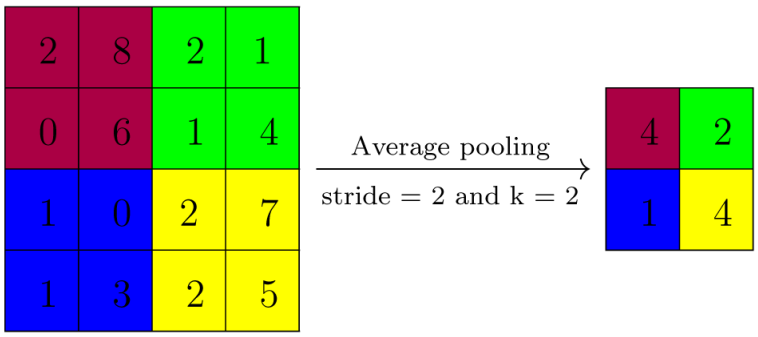

Figure 2.8: An example of average pooling operation.

2.3 Convolution in 3D

We discussed the convolution for the $2D$ matrix. Now let us consider the $3D$ volume (or 3D Tensor). The third dimension is the depth of the input volume, or we can call it the number of channels. RGB image has three channels: red, green, and blue. For the convolution, the filter must have the same number of the channel as the input. We do the convolution

similar as in $2D$ convolution in each channel with the corresponding channel in the kernel and sum them up to get the number. We apply multiple kernels to detect the different features from an image. While doing so, we keep on stacking the feature maps obtained to get an output volume. So the output feature map will have the depth equal to the numbers of filters applied. In general, for an input image of size $M \times N \times C$,(C is the number of channels),and a kernel of size $k \times k \times D$ with padding $=p$ and stride $= s$, the output will have the dimension:

$$\left\lfloor \frac{M + 2p - k}{s} + 1 \right\rfloor \times \left\lfloor \frac{N + 2p - k}{s} + 1 \right\rfloor \times \text{number of filters}$$

Figure 2.9 shows the convolution operation on input image of size $m \times n \times c_1$,(c_1 is 3 in this case) and total four filters are used. We observe that the number of channels of the input image and the filter is the same, c_1. The output of convolution with each of these filters results in a matrix of size $m_1 \times n_1$. Then finally we stacked them up to get the final output volume of size $m_1 \times n_1 \times 4$.

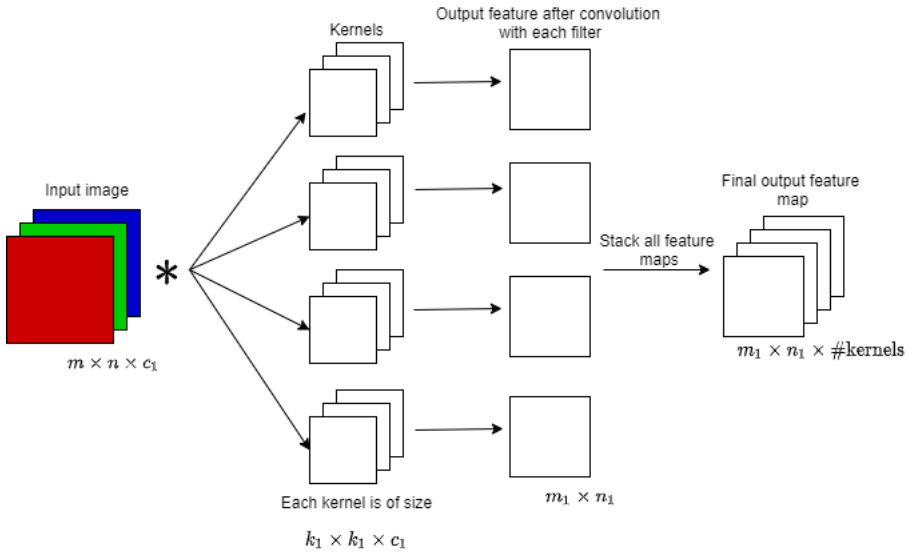

Figure 2.9: 3D convolution operation with multiple filters

2.4 Dense Layer

For the classification problem using CNN, the convolutional layers are followed by a dense layer or fully connected (FC) layer. Each neuron of the FC layer is connected to each neuron of the previous layer. First, we flatten the output volume obtained from the convolution, and then we define the FC layer. FC layers are followed by Sigmoid function for the binary classification and by Softmax for multiclass classification. Notice that for the image segmentation task, we do not need the FC layer. However, it is essential to know about this layer as we sometimes use a pre-trained network used for classification also for the segmentation task. The idea is that we remove the FC layer from the net and add an extra layer to get the output as an image again. More about this will be discussed in later sections.

2.5 Activation Functions:

Deep neural networks are designed to capture complex features from the input data. Activation functions are able to capture nonlinearity from the data, so it is the backbone of a deep neural network. A deep network without activation functions is equivalent to just another shallow network. In CNNs, we apply activation functions to the outcome of convolution operation added with bias term. There are several commonly used activation functions[49, 2]. Some of them are described below:

Sigmoid

The sigmoid function is given by the following equation:

$$\sigma(x) = \frac{1}{1 + e^x}. \tag{2.8}$$

In general artificial neural network (ANN), input for this function is the sum of weighted inputs and bias. For the CNN, sigmoid is applied to the sum of output of the convolution

operation and bias. The output of sigmoid is between 0 and 1, which can be interpreted as a probability whether certain input belongs to the given class or not. So, it is usually used in the last layer for the binary classification. We usually avoid using sigmoid in the hidden layer of neural network, because of the problem of vanishing gradient. The derivative of the sigmoid function is given by:

$$\sigma'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = \sigma(x)(1-\sigma(x)). \tag{2.9}$$

We can see in Figure 2.10 that the maximum derivative is attained at $x = 0$ and the derivative is approaching towards zero on both sides. So using the sigmoid in the hidden layer results in the gradient approaching to zero in long chain rule computation.

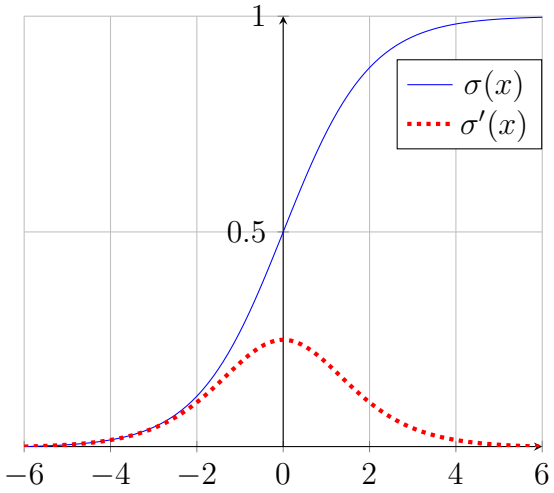

Figure 2.10: Sigmoid and its derivative function.

Hyperbolic Tangent

A slightly different function than the sigmoid is hyperbolic tangent which outcomes values between -1 and 1. The $\tan h$ function and the corresponding derivative function is given by the following equations:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.10}$$

$$\tanh'(x) = 1 - f(x)^2 \tag{2.11}$$

Figure 2.11 show the hyperbolic tangent function and its derivative function.

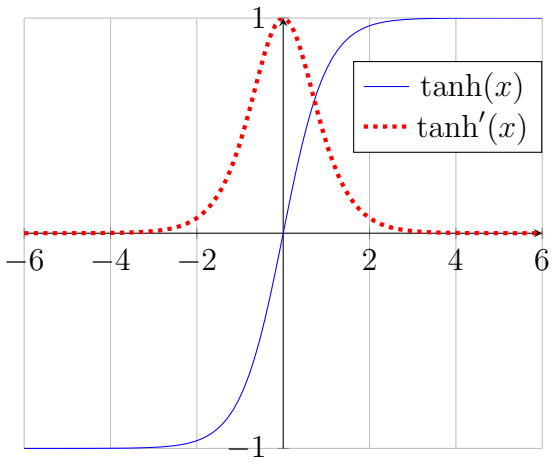

Figure 2.11: Hyperbolic tangent and its derivative functions.

Hyperbolic tangent is also affected by the vanishing gradient problem.

Rectified Linear Unit (ReLU)

Another activation function, rectified linear unit (ReLU) [9], is defined by the following equation.

$$f(x) = \max(0, x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{2.12}$$

We observe that the gradient of the ReLU function is 1 when the input is greater than 0 and 0 otherwise.

$$f'(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{2.13}$$

So multiplying a chain of gradients of ReLU in the backpropagation process obtain either 1 or 0. There is no vanishing of the slope. The gradients pass to the very early layer either

as they are or become precisely 0 in the process. Therefore, it is a widely used activation function in the deep neural network. Figure 2.12 show the ReLU function and its derivative function.

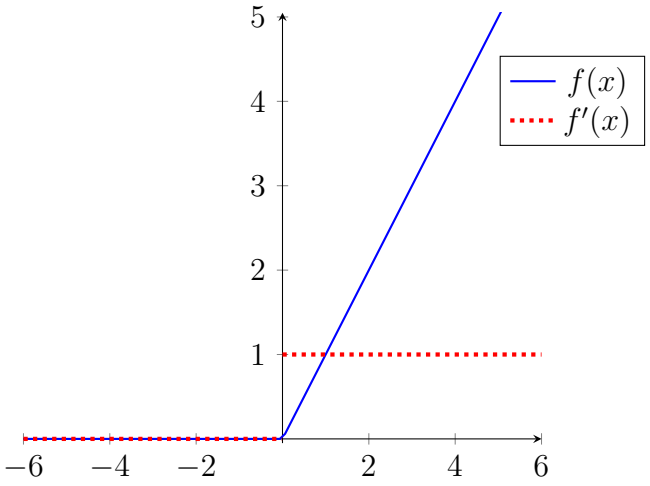

Figure 2.12: Rectified Linear Unit and its derivative functions.

There are some other variants of ReLU. For example, Leaky ReLU, which is defined as:

$$f(x) = \max(0.1 * x, x) = \begin{cases} x & \text{if } x \geq 0 \\ 0.1x & \text{if } x < 0 \end{cases} \tag{2.14}$$

Figure 2.13 shows the Leaky ReLU function.

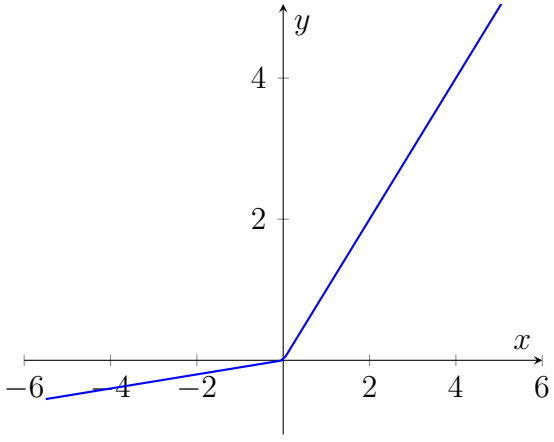

Figure 2.13: Leaky ReLU function.

SoftMax

Sofmax activation is defined by the following equation:

$$f(X)_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \text{ for } i = 1, 2, ..., k, \tag{2.15}$$

where $X = (x_1, x_2, ..., x_k)$ is a k-dimensional vector and k is the number of classes. Softmax activation is often used in multiclass classification problems. It gives the probability of that the input value belongs to certain class.

2.6 ConvNet

Now we have defined all the building blocks to build a convolutional neural network (ConvNet). A typical ConvNet is shown in Figure 2.14.

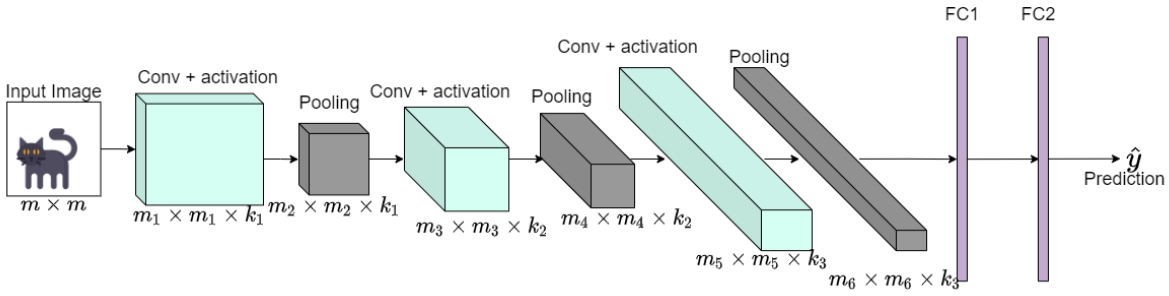

Figure 2.14: A CNN architecture for classification.

Earlier layers of the neural networks extracts basic features of an images such as line and edges. As networks grows deeper it detects complex features. A typical ConvNet follows a common pattern of decreasing the spatial dimension by the use of pooling layer while increasing the depth of the feature map by increasing the number of the filters in each step.

2.7 CNN Training

Figure 2.14 shows the general outlook of CNN for classification. Input images are supplied through several convolutional, activation, and pooling layers, and finally the output $\hat{y}$ is

obtained. We compare the predicted label $\hat{y}$ with the actual label y and calculate the error. Sum of the error of all the data gives the total error, sometimes we call it a cost function or loss function. Our goal is to minimize the loss, so that the predicted value is close to actual value. We use optimization algorithms to minimize the loss function. Loss function can be chosen based on the requirements of different models and the types of datasets. Some of the loss functions are discussed below:

Loss Functions

Let y_i, $\hat{y}_i$, and N denotes the actual value, predicted value, and the number of observations. Some of the loss functions are discussed below:

- Mean square error (MSE)

  Mean squared error is usually used for regression. it is defined as follows:

  $$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2.$$

- Mean absolute error (MAE)

  Mean absolute error is also usually used for regression. It is defined as follows:

  $$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

  Mean absolute error is less sensitive to outliers in data than the mean squared error.

- Cross-entropy loss (CEL)

  Cross-entropy loss [18] for multiclass classification is defined as follows:

  $$\text{CEL} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{c}(y_{ij}\log p_{ij})$$

  where y_{ij} is the true label, and p_{ij} is the predicted probability for ith observation belongs to class j, and c is the number of classes. Note that for each fixed i, only one of the y_{ij} is 1, the rest are zeros.

- Huber loss (HL)

  Huber loss [16] is defined as follows:

$$\mathrm{HL} = \begin{cases} \frac{1}{2N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 & \text{if } |y_i - \hat{y}_i| \leq \delta \\ \frac{1}{N}\sum_{i=1}^{N}(\delta|y_i - \hat{y}_i| - \frac{\delta^2}{2}) & \text{if } |y_i - \hat{y}_i| > \delta \end{cases}$$

  where δ is a constant. Huber loss is usually used for regression. It's less sensitive to outliers in data than the the mean squared error.

- Focal loss

  The Focal loss function is able to deal with class imbalance problems. There are two kinds of imbalances. First one is the class imbalances that is the number of background and foreground examples have high difference. Another is the imbalances within multiple foreground values, i.e., the imbalances within the multiple classes. Tsung-Yi Lin et. al.[32] defined focal loss as following: Consider cross-entropy for binary classification:

$$CE(y,p) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{otherwise} \end{cases} \tag{2.16}$$

  Define, p_t as:

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise} \end{cases}$$

  Thus we can rewrite $CE(y,p)$ as:

$$CE(y,p) = CE(p_t) = -\log(p_t)$$

  A little modification to above equation is

$$CE(p_t) = -\alpha_t \log(p_t) \tag{2.17}$$

where

$$\alpha_t = \begin{cases} \alpha, & \text{if } y = 1 \\ 1 - \alpha & \text{otherwise} \end{cases}$$

and $\alpha \in [0, 1]$

This is the balanced cross-entropy where α factor is able to manage the imbalances of the positive/ negative examples. Still, it does not distinguish between easy/hard samples [32] (Easy examples are the examples that are correctly classified, and hard examples are the examples that are misclassified). So we need another modification in the model by degrading the weight for easy examples and focus on training hard negatives. Hence the focal loss is defined as follows:

$$FL(p_t) = -\alpha_t (1 - p_t)^{\gamma} \log(p_t) \tag{2.18}$$

2.7.1 Backpropagation

During the training process of the CNN, for each training data, the error is evaluated using predicted values and the true values. The Sum of error of all input data gives the total cost of the model. We discuss some of the cost functions in Section 2.7. The objective is to minimize the cost by repeatedly updating the networks' parameters (weights and biases) until the optimal set of possible parameters is obtained. The training process utilizes a gradient, the rate at which cost changes with respect to each parameter. We use gradient descent or some variants of gradient descent for the cost minimization problem. The gradient provides a direction to move in the error surface by adjusting the parameters iteratively. This strategy is called backpropagation [2], where the error is propagated backward from the output layer up to the input layer. One of the difficulties of this procedure is that parameters can be anywhere in the networks. Finding a gradient involves calculations of partial derivatives with respect to all the parameters. This process sometimes needs a long chain rule, especially

for the parameters in earlier layers of the networks. As a result, gradients could ultimately vanish or decay back through the networks, known as the vanishing gradient problem.

2.7.2 Optimizer

- Gradient Descent

  The objective is to find the parameters W of the model that minimizes the total loss function or the cost function L. Gradient descent[2, 41] is an iterative method to find the minimum value by updating the parameters as follows:

  $$W_t = W_{t-1} - \eta \nabla L(W_{t-1}). \tag{2.19}$$

  where W is a vector represents all the network parameters, L is the loss function, and η is the learning rate or step size, which is a hyperparameter and can be adjusted as we need. But we need to keep in mind that if the learning rate is too large, the method may converge very slow or may not even converge and if is too small, the iteration process becomes very slow. Even with appropriate fixed η, gradient descent can converge because near the optimal point slopes becomes smaller. There are different version of gradient descent. Sometime we call gradient descent, a batch gradient descent also. The drawback of batch gradient descent is that it can be very slow if we have large data because it uses all the data in each iteration.

- Stochastic Gradient Descent

  We randomly select a single sample in each iteration and the gradient will be calculated for that specific sample only.

- Mini Batch Gradient Descent

  We divide the dataset in mini-batches and apply the gradient descent in each mini

batches. So this is the combination of batch gradient descent and the stochastic gradient descent. If the size of each mini batches is 1 , then it is the same as stochastic gradient and also batch gradient descent is the special case of mini batch gradient descent when the size of mini batch is the total number of training example in the dataset. Mini batch gradient descent method is often referred as stochastic gradient descent method nowadays in deep learning.

- Stochastic Gradient Descent with Momentum (SGDM)

  SGDM [44] is a method which adds the momentum term to the gradient to help accelerate gradient in the right directions. This often leads to faster converging of the algorithm. The update rule is as follows:

$$W_{t+1} = W_t - \eta \nabla L(W_t) + \beta(W_t - W_{t-1}) \tag{2.20}$$

  where W is a vector represents all the network parameters, L is the loss function, η is the learning rate, and β is a hyper-parameter in $[0, 1]$. $\beta = 0.9$ is often used in applications.

- Adaptive Moment Estimation (Adam)

  Adam [26] is an extension of the stochastic gradient descent algorithm, which is based on adaptive estimates of lower-order moments. The update rule is as follows:

$$\begin{aligned}
W_t &= W_{t-1} - \eta . \frac{\hat{v_t}}{\sqrt{\hat{s_t}} + \epsilon} \\
\hat{v_t} &= \frac{v_t}{1 - \beta_1^t} \\
\hat{s_t} &= \frac{s_t}{1 - \beta_2^t} \\
v_t &= \beta_1 v_{t-1} + (1 - \beta_1) \frac{\partial L}{\partial W_t} \\
s_t &= \beta_2 s_{t-1} + (1 - \beta_2) \left[\frac{\partial L}{\partial W_t} \right]^2
\end{aligned}$$

where W is a vector represents all the network parameters, L is the loss function, η is the learning rate, and β_1, β_2 are hyper-parameters in $[0, 1)$ which control the exponential decay rates of the moving averages v_t. ϵ is a very small constant for numerical stability.

2.7.3 Regularization

One way to avoid over-fitting the model is to add regularization term in the cost function. Let $f(x;\theta)$ be the cost function, then

$$f(x;\theta) = f^{model}(x;\theta) + \lambda\Omega(\theta) \tag{2.21}$$

where $f^{model}(x;\theta)$ is the cost function of the model, $\Omega(\theta)$ is the regularization function, which is a function of parameters of the model and λ is the regularization parameter. Our objective is to minimize the function $f(x;\theta)$. The regularization function $\Omega(\theta)$ helps to make the model less complex by penalizing some parameters. The hyperparameter λ controls how much of an effect the regularization has on the objective function. In general, we want the regularization term $\lambda\Omega(\theta)$ much smaller than the cost function of the model $f^{model}(x;\theta)$. If λ is very big, it leads that all parameters close to zero and hence resulting in underfitting the model. It is the case when we have high bias and low variance. On the other hand if λ is too small or close to zero, it leads to overfitting of the data.

- L_2 regularization(Ridge regression)

$$\Omega(\theta) = \|\theta\|_2^2 = \theta^T\theta \tag{2.22}$$

  L_2 regularization[14] is also called the weight decay because of the fact that it causes the weight decrease in each iteration.

- L_1 regularization(LASSO)

$$\Omega(\theta) = \sum_{i=1}^{n} |\theta_i| \tag{2.23}$$

L_1 regularization[57] is also called least absolute shrinkage and selection operator (lasso) regression. It has the effect of making parameters to zeros and hence the model will become sparse. Hence L_1 regularization has the effect of feature selection.

- Dropout

  Unlike L_1 and L_2 regularization which regularize the model by modifying the cost function, dropout[52] regularizes the model by modifying the network itself. In the dropout layer we randomly drop neurons from the network in each iteration.

- Data Augmentation

  Data augmentation is a regularization technique to create artificial training data. It is one of the many popular methods to prevent over-fitting the model, and generally, we get better performance after data augmentation. There are various ways to generate additional data. For example, we can do a reflection of the image about the x-axis or y-axis. Similarly, we can do a translation with a small amount or a rotation from a different angle. We can also do shearing if it is reasonable.

2.7.4 Batch Normalization

Batch normalization[17] is the process of scaling and normalizing the data. For example, for a given dataset X, we can make the mean and standard deviation being 0 and 1 respectively by

$$X^{normalized} := \frac{X - \mu}{\sigma} \tag{2.24}$$

Batch normalization usually leads the optimization algorithm runs much faster and converges in fewer iterations.

CHAPTER 3

SEMANTIC SEGMENTATION OF CELL NUCLEI USING DEEP LEARNING [1]

3.1 Definition

Semantic segmentation is the task of assigning each pixel in an image to a given class label. So if we input an image in the network, it outputs an image of the same size with the intensity of each pixel being the label. If there are two objects of the same category, the same label will be assigned to them, hence semantic segmentation will not distinguish the two objects. To distinguish between separate objects of the same class, instance segmentation models are needed which will be discussed in Chapter 6. Semantic segmentation has been used in enormous applications, including geosensing for land usage[31, 43], autonomous driving[3, 6], facial segmentation for gender, age, and expression estimation[25], and medical image analysis[11, 20, 23, 56, 36, 59, 58, 22]. Most of the semantic segmentation models have an encoder-decoder architecture. Figure 3.1 shows the general flow of semantic segmentation architecture.

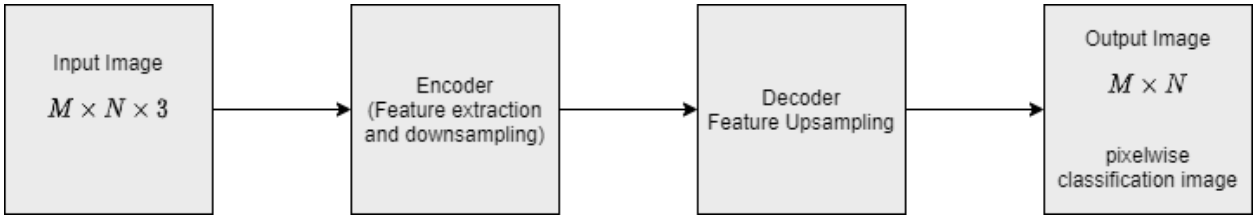

Figure 3.1: High level semantic segmentation architecture

[1]This chapter includes verbatim excerpts from
K C Khatri,R.; Caseria, B.; Lou, Y.; Xiao, G. and Cao, Y. "Automatic Extraction of Cell Nuclei Using Dilated Convolutional Network", Inverse PROBLEMS AND Imaging, Vol. 15, No. 1, 27-40, 2021.

3.1.1 Encoder

The encoder for the semantic segmentation can be structured in the same way as for the classification neural network, where we have a bunch of blocks of convolution layers followed by the pooling layers, but we do not include the dense (fully connected) layers. Dense layers are removed because they make it hard to manage different input sizes. So the output of the encoder is a spatial map instead of classification scores. We call it a heatmap or the feature map. The obtained heatmap will be the input for the following decoder part. Figure 3.2 illustrates the encoder for semantic segmentation network, which is obtained by removing the fully connected layer from the CNN for classification.

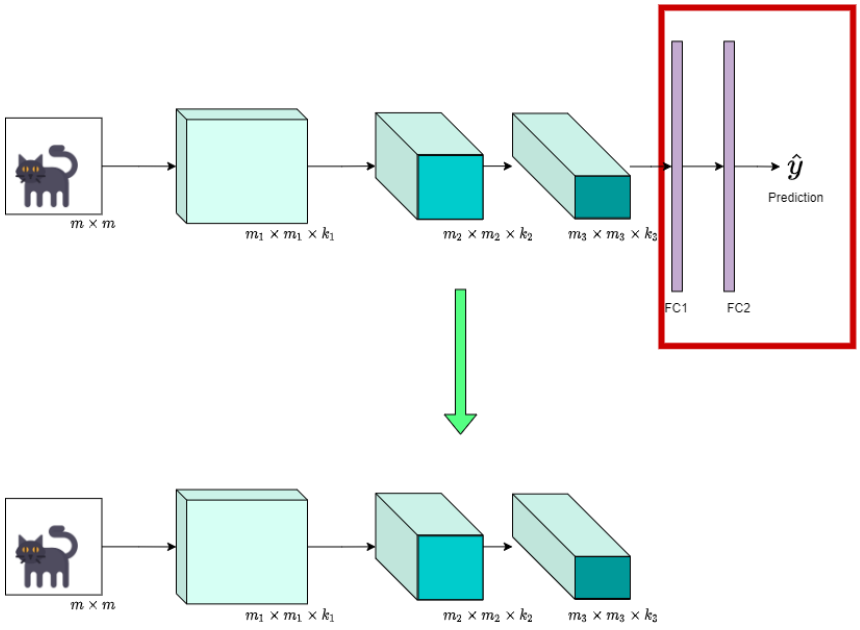

Figure 3.2: Encoder network for semantic segmentation

The encoder or the feature extraction part of the network is usually based on the convolution neural networks for classification. VGG-16[51], ResNet[12], AlexNet[28], MobileNet[15] are some of the popular architectures for classification. Figure 3.3 illustrate the architecture of the popular ConvNet VGG-16:

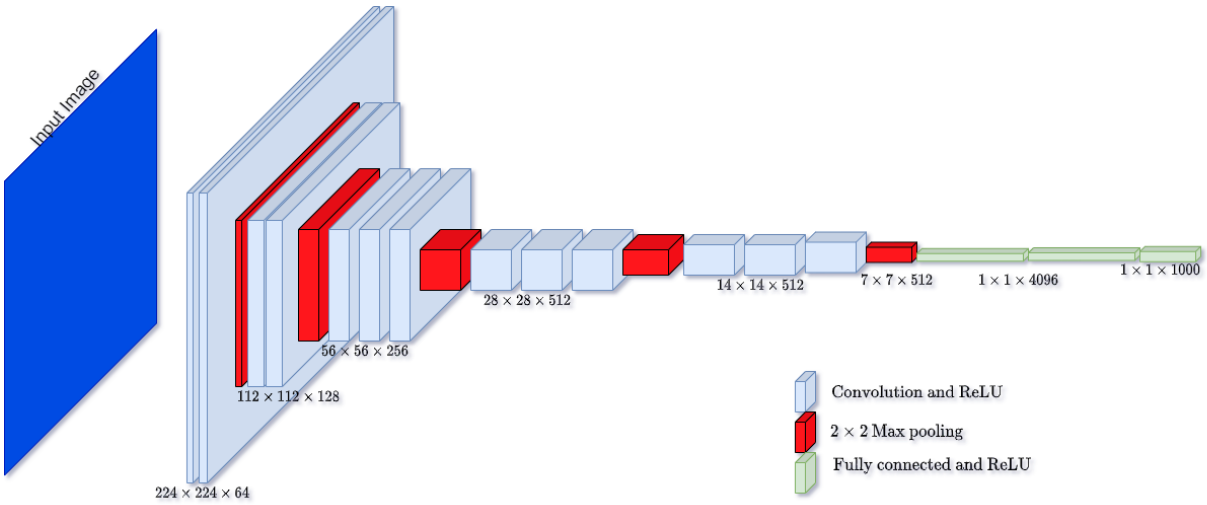

Figure 3.3: VGG-16 architecture for image classification

We can use these architectures as the base of the encoder and of course we can also design our own encoder architecture.

3.1.2 Decoder

Usually, the spatial dimension of the image getting smaller as it goes deeper and deeper in the encoder part of the network. The feature map or the output of the encoder network losses lots of spatial information during the convolution and pooling operations. To get a pixel-wise classification, we need to modify the network architecture gradually to obtain the spatial dimension as the original input. This task is done by the decoder part of the network. The decoder part of the network usually increases the height and the weight of the feature map gradually by upsampling. There are various ways to upsample the input in a network layer.

Deconvolution (Transposed Convolution)

We use deconvolution to project the feature maps to a higher dimensional space. By name, it should mean inverse convolution, which is not precisely correct in the context of semantic

segmentation, so some people like to use a different name like transposed convolution or inverse strided convolution.

Suppose a 3×3 filter convolves with an input image of 4×4 with stride two and padding 1; the resulting image will be downsampled by a factor of 2. i.e. it will have size of 2×2. Now, if we want to project the 2×2 matrix to 4×4 dimensional matrix, we will do convolution with 2×2 kernel with stride of $\frac{1}{2}$. So the output will be upsampled two times. Deconvolution forward pass is the same as the normal convolution backward pass. This idea is implemented in the popular semantic segmentation network FCN (Fully Convolutional Network [35]).

There is a reason for calling so-called deconvolution a transposed convolution. Let us consider the following convolution shown in Figure 3.4

3	2	1	0
0	9	4	2
1	0	6	7
9	8	6	4

$*$

1	0	-1
2	0	1
-1	0	1

$=$

11	29
1	10

Figure 3.4: An example of convolution operation.

We can express this operation as matrix multiplication of two matrices, where the 3×3 kernel matrix is represented by the matrix on the left and the input matrix is flattened to get the 16×1 matrix in the right as shown in Figure 3.5. The final result is reshaped to get a desired 2×2 matrix.

$$\begin{bmatrix} 1 & 0 & -1 & 0 & 2 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 2 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 2 & 0 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 2 & 0 & 1 & 0 & -1 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 3 \\ 2 \\ 1 \\ 0 \\ 0 \\ 9 \\ 4 \\ 2 \\ 1 \\ 0 \\ 6 \\ 7 \\ 9 \\ 8 \\ 6 \\ 4 \end{bmatrix} = \begin{bmatrix} 11 \\ 29 \\ 1 \\ 10 \end{bmatrix}$$

Figure 3.5: An equivalent representation of the convolution in example 3.4

For the corresponding transposed convolution in the decoder part, we need a 16×4 convolution matrix (transpose of 4×16 matrix) and multiply it with the 2×2 input matrix (the output of convolution) which is resized to a 4×1 matrix.

Let us reshape the kernel and the input image as shown in Figure 3.6 [5].

$$\text{Kernel, } K = \begin{bmatrix} k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 \\ k_7 & k_8 & k_9 \end{bmatrix} \longrightarrow \begin{bmatrix} k_1 & k_2 & k_3 & 0 & k_4 & k_5 & k_6 & 0 & k_7 & k_8 & k_9 & 0 & 0 & 0 & 0 & 0 \\ 0 & k_1 & k_2 & k_3 & 0 & k_4 & k_5 & k_6 & 0 & k_7 & k_8 & k_9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & k_1 & k_2 & k_3 & 0 & k_4 & k_5 & k_6 & 0 & k_7 & k_8 & k_9 & 0 \\ 0 & 0 & 0 & 0 & 0 & k_1 & k_2 & k_3 & 0 & k_4 & k_5 & k_6 & 0 & k_7 & k_8 & k_9 \end{bmatrix}$$

$$\text{Input Image, } I = \begin{bmatrix} i_1 & i_2 \\ i_3 & i_4 \end{bmatrix} \longrightarrow \begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \end{bmatrix}$$

Figure 3.6: An example on how to reshape the kernel and input image for transposed convolution.

Then the transposed convolution of the input matrix, I with kernel K is given by the following matrix multiplication:

$$\text{Output } O = K^T \times I$$

where K^T is transpose of matrix K, thus is of size 16×4 and the resulting matrix will be of size 16×1 which we will rewrite it to get upsampled 4×4 matrix. Alternatively, the transpose convolution can be shown as the following:

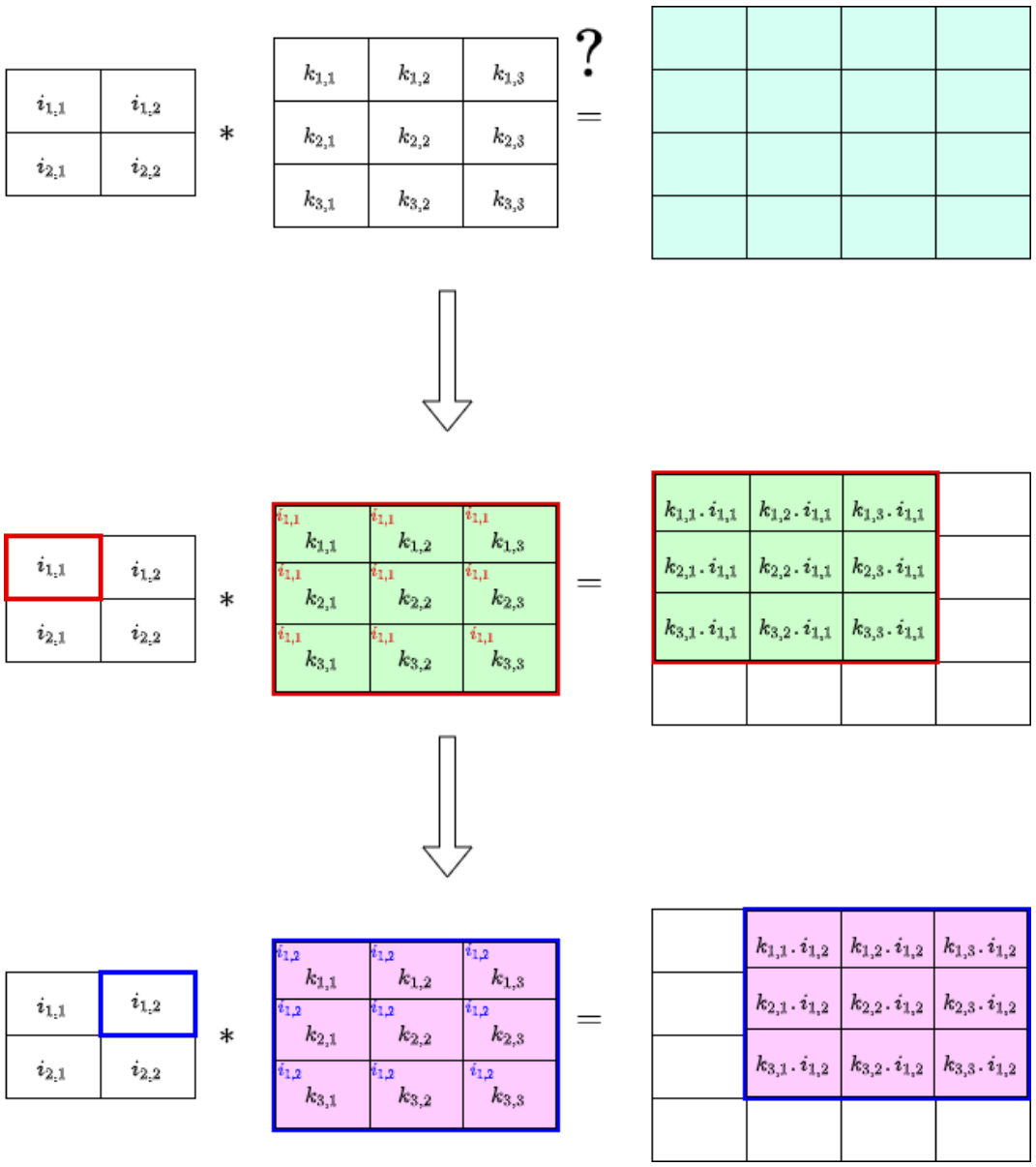

Figure 3.7: Transpose Convolution of 2×2 input with 3×3 filter, 1 stride and no padding

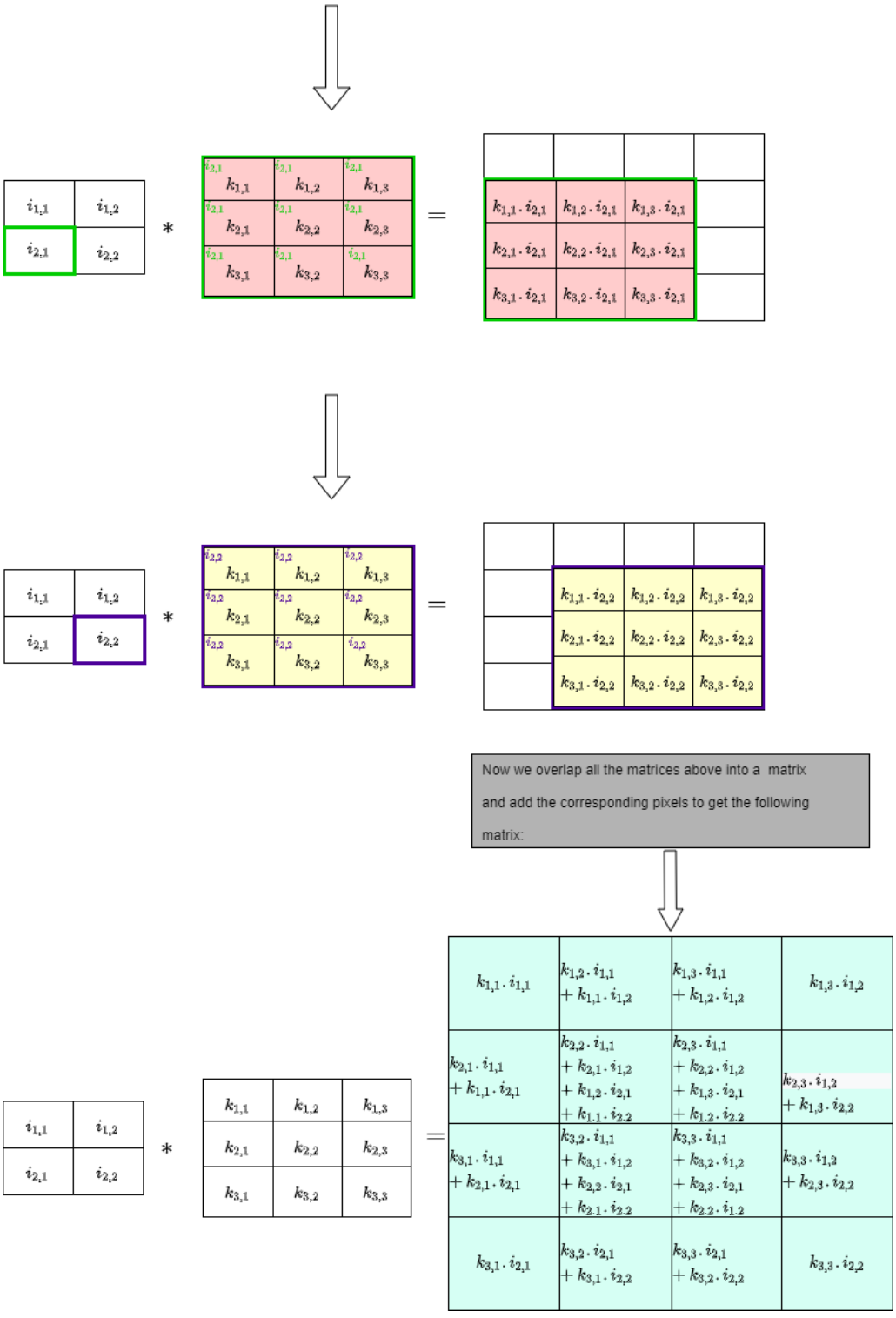

Figure 3.7, continued.

The transpose convolution with one unit stride and padding is shown in Figure 3.8.

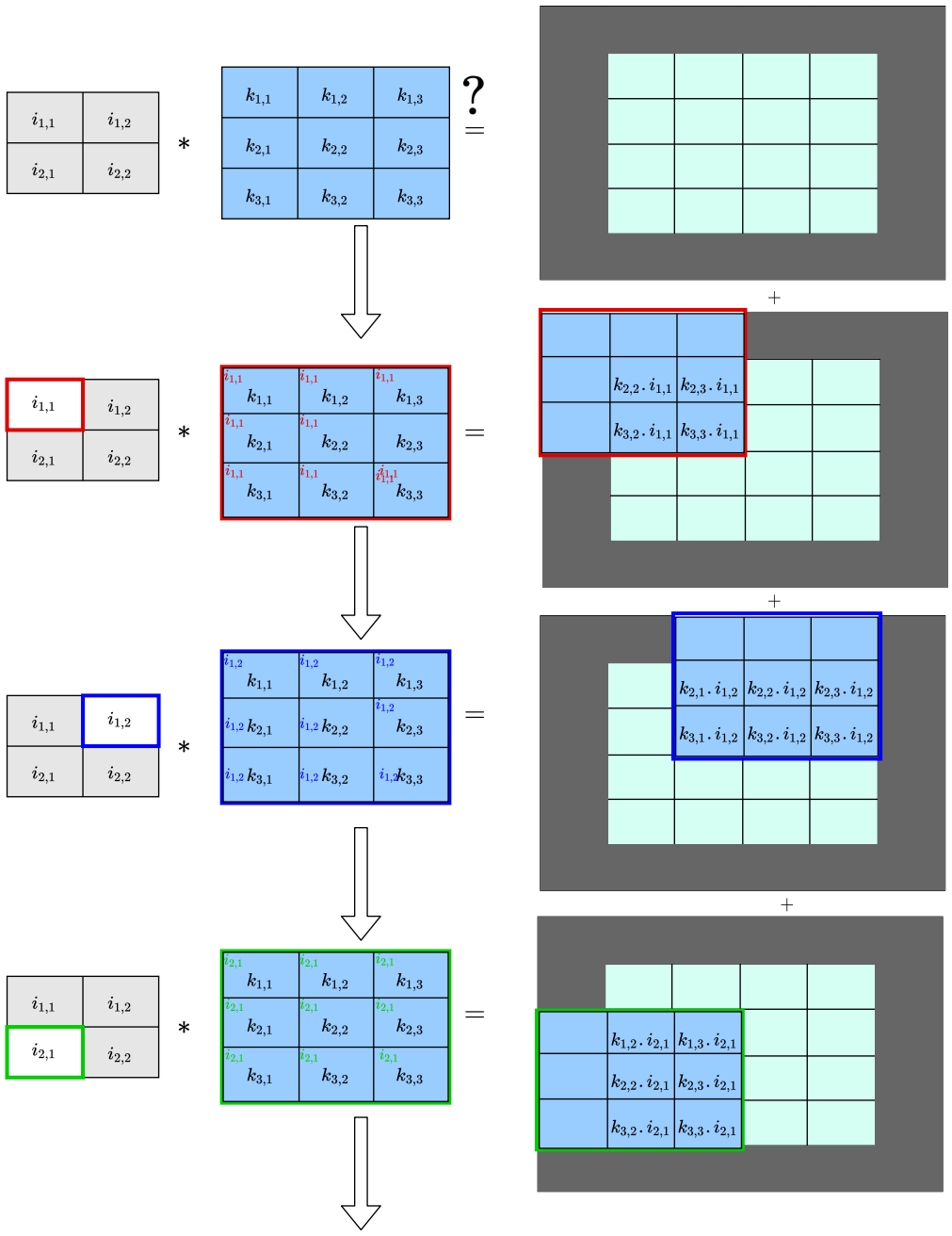

Figure 3.8: Transpose Convolution of 2×2 input with 3×3 filter, stride 2 and 1 zero padding

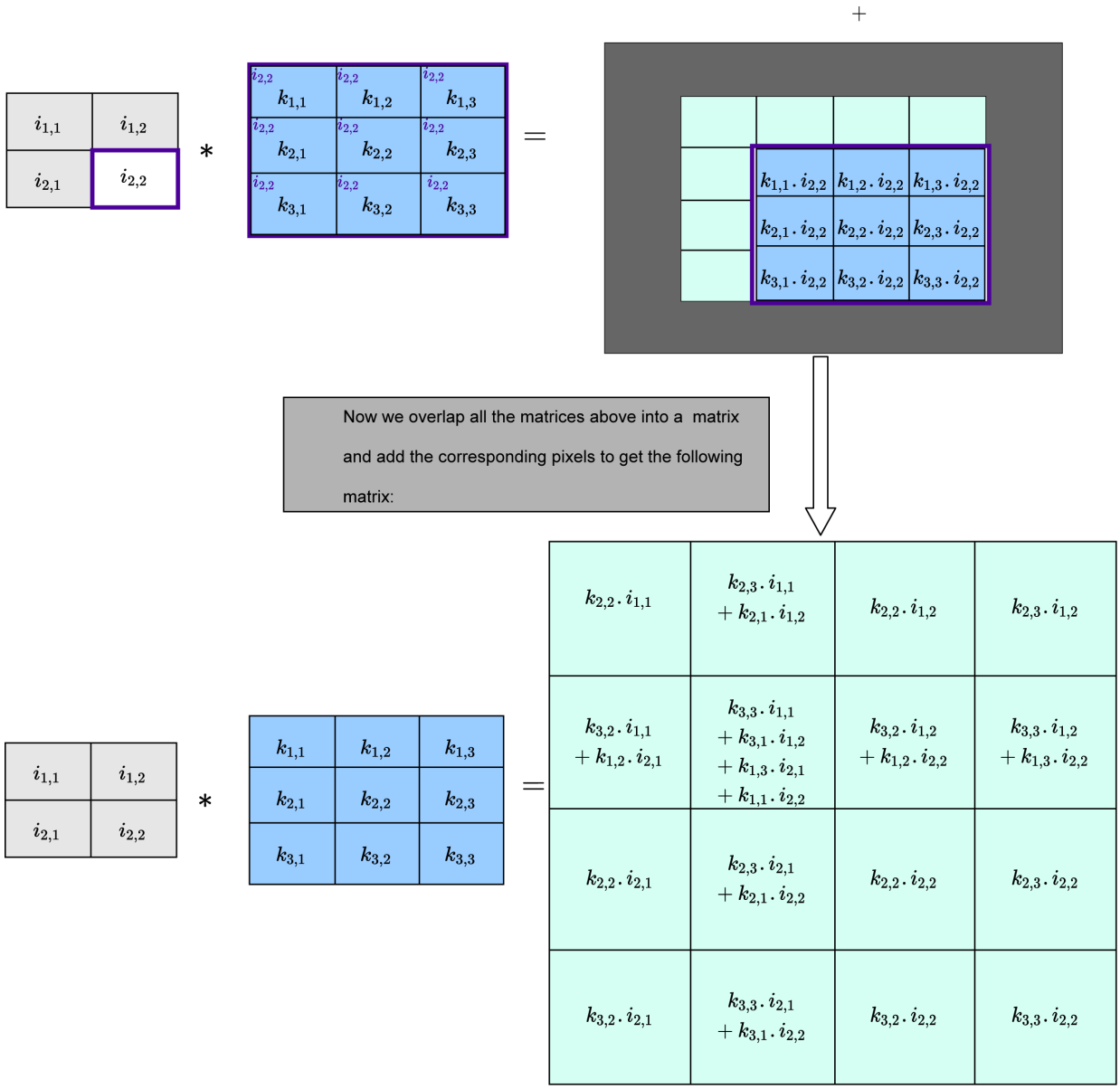

Figure 3.8, continued.

The transpose convolution is usually applied to the output image of the corresponding convolution and pooling operations (for downsampling) in the encoder part. In order to get an output with the same size as the original image before the downsampling operation, we need to choose a kernel of specified size, stride, and padding.

Scaling

Upsampling can also be done by simple scaling algorithms such as nearest neighbor interpolation and bilinear interpolation [54]. Figure 3.9 shows how we can upsample a 2×2 matrix by factor of two to get the resulting matrix of size 4×4 using nearest neighbor interpolation.

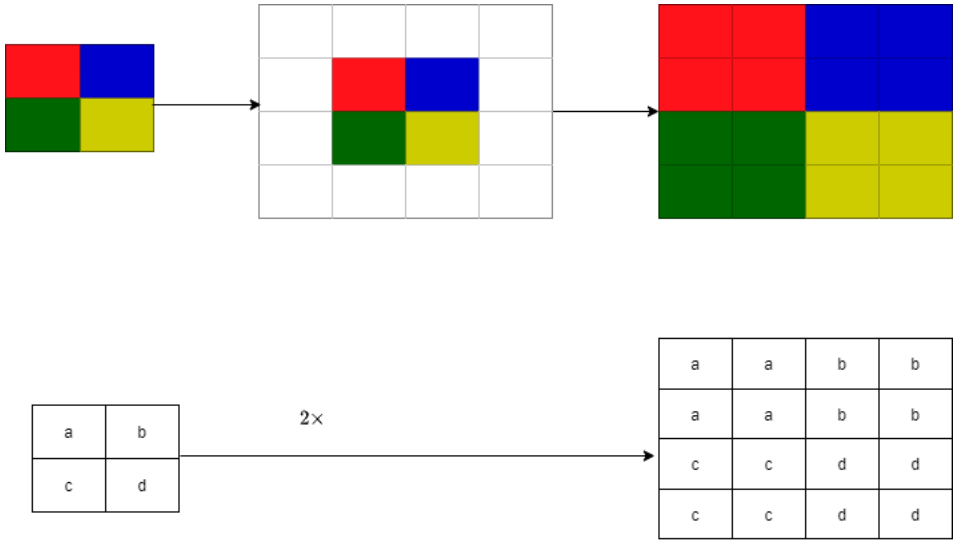

Figure 3.9: An illustration of upsampling by nearest neighbor interpolation.

Figure 3.10 illustrates the bilinear interpolation technique to upsample the 2×2 input into a 4×4 output,

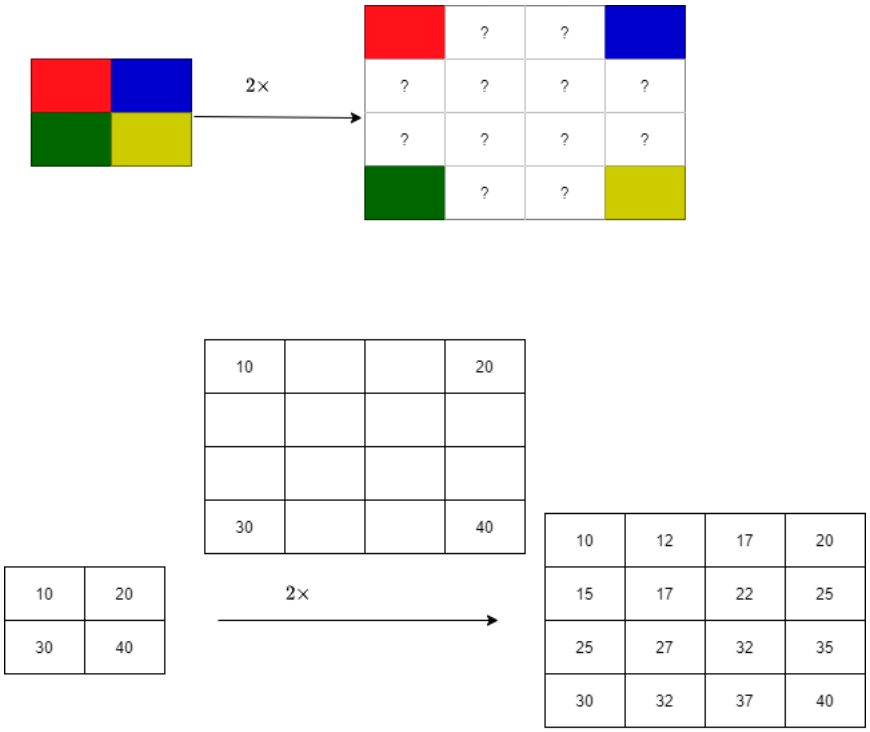

Figure 3.10: An illustration of upsampling by bilinear interpolation.

Unpooling

Another popular method for upsampling is the method called unpooling [42]. Unpooling is the opposite operation of pooling. Pooling is used to downsample the network, while unpooling is used to upsample it. The position of the pixel in the input image in pooling layer is remembered. In the corresponding upsampling part, the input pixel will be positioned to the corresponding position while doing max pooling in encoder and all other position will be filled by either the neighboring pixel or zeros. Figure **??** shows the max pooling and max unpooling operations.

3.2 Some examples of semantic segmentation networks

In this section, we will discuss some of the popular semantic segmentation architectures:

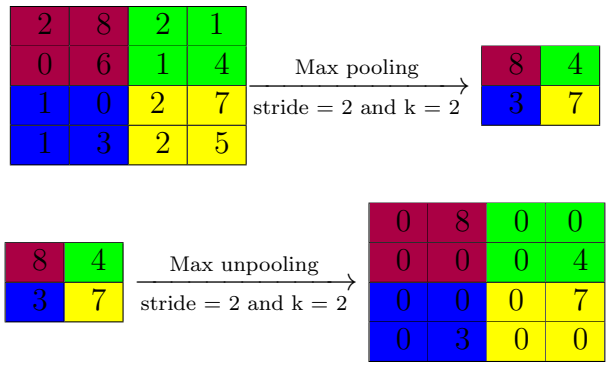

Figure 3.11: Max pooling and max unpooling operations.

Fully Convolutional Network (FCN)

Traditional convolutional networks have fully connected layers which make hard to manage different input sizes. Fully convolutional networks proposed by Long et al. [35] replaced the fully connected layers with convolutional ones to output spatial maps instead of classification scores. Then those maps are upsampled using deconvolutions to produce dense predictions. FCN shows how to train CNNs end-to-end to make dense predictions for semantic segmentation.

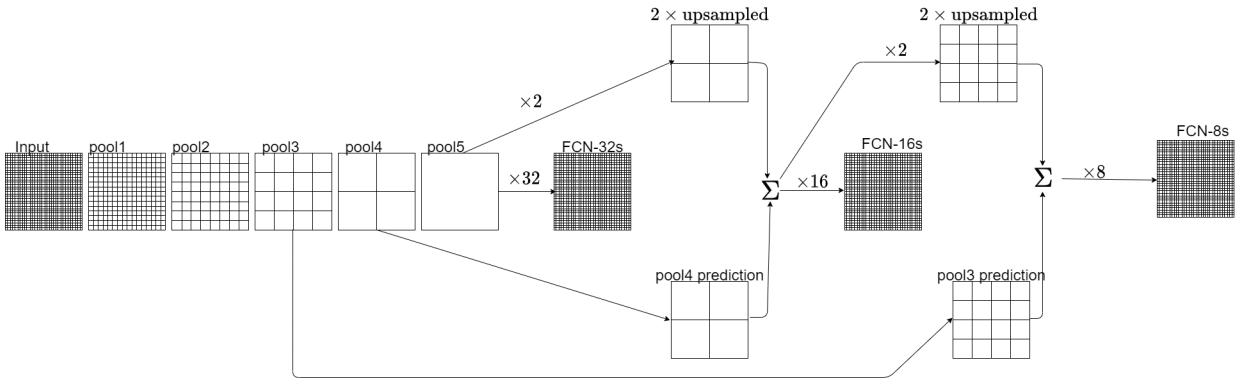

Figure 3.12: FCN upsampling techniques as suggested in [35]

The encoder of the FCN is the taken from VGG-16 which is shown in Figure 3.3. Final three fully connected layers are removed from the VGG-16. So the output of the encoder is given by the fifth pooling layer. Each pooling layer in the network downsamples the input

by the factor of 2. So, the final output is downsampled by the factor of 32. The paper [35] has 3 models of FCN namely FCN-32s, FCN-16s, and FCN-8s. We can see in the figure 3.12, output of pool5 is upsampled 32 times to get the original spatial dimension image. For FCN-16s, first pool5 is upsampled by factor of 2, concatenate it with the pool4 output, then upsampled it 16 times to get the final output. Similarly, for FCN-8s, concatenated result of 2 times upsampled pool5 and pool4 prediction is upsampled by factor of 2. After that we concatenate this with pool3 prediction and then outcome of this is upsampled by factor of 8.

FCN is the first neural network proposed for segmentic segmentation. However, the architecture of FCN has a large number of trainable parameters in the encoder network, which makes it hard to train and memory intensive when it comes to testing time.

U-Net

Another popular network for semantic segmentation is U-Net proposed by Olaf Ronneberger et al [48]. Similar to FCN, the fully connected layers are replaced by the convolutional layers for upsampling. The architecture of U-Net is symmetric; that is, every downsampling layer has a corresponding upsampling layer. Moreover, U-Net has another operation called skip connection, where every corresponding feature map in the encoder is copied and concatenated to the upsampled feature map in the decoder part. One of the effects of downsampling the image is that it may lose lots of very detailed, fine-grained spatial information while getting very high-level contextual details. So, the skip connection allows the neural network to take the very high resolution, low-level feature information to pass directly to the corresponding upsampled layer. And so this way, the upsampled layer has both the high-level spatial information with lower resolution as well as low-level more detailed information. Figure 3.13 illustrate the architecture of U-net. U-net was first proposed specially for biomedical image segmentation. However, it has been widely used in many other applications.

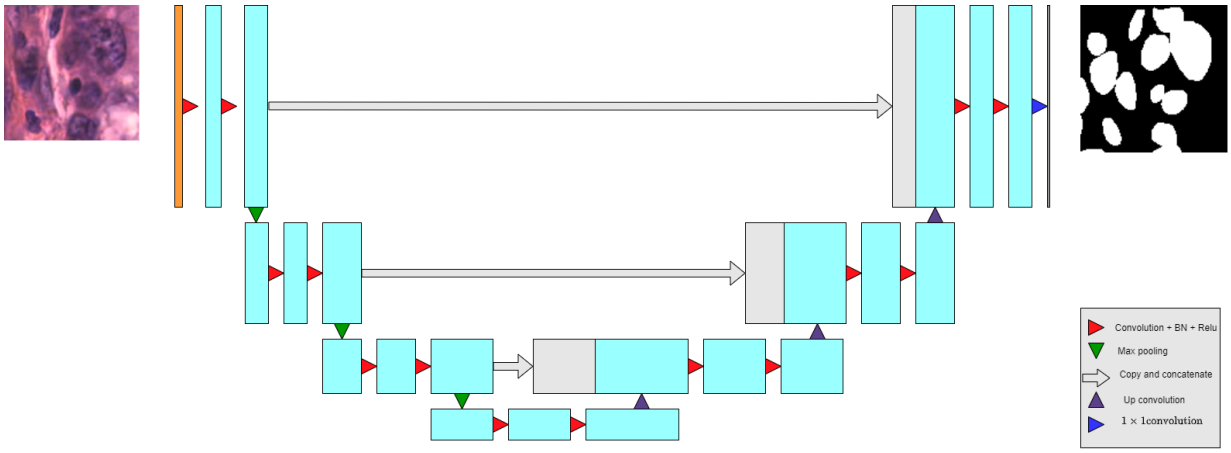

Figure 3.13: U-Net architecture. This figure is inspired from the original paper [48].

SegNet

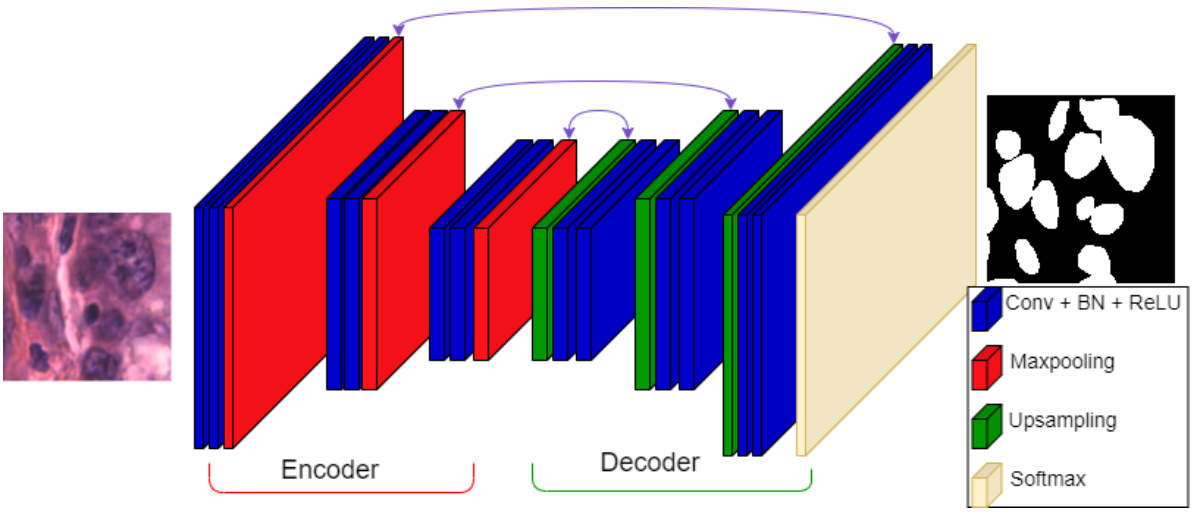

Figure 3.14: SegNet architecture. Figure is inspired from the original paper [1]

To overcome the problems caused by deconvolution, Noh et al. [42] proposed a semantic segmentation algorithm (DeconvNet) by learning a deep deconvolution network. Independently, Badrinarayanan et al. [1] proposed SegNet, a deep convolutional encoder-decoder architecture for semantic pixel-wise segmentation. The SegNet also has the symmetric architecture of the encoder and its corresponding decoder, followed by a final pixel-wise classification

layer. A mix of convolutional and pooling layers is used in the encoder network to combine and summarize the data obtained from the RGB coordinates. A mix of upsampling and convolutions is performed in the decoder network to map the low-resolution encoder feature maps to the same resolution feature maps as the input for pixel-wise classification. The max pooling indices at the corresponding encoder layer are recalled to perform the unpooling operation for upsampling. We will get sparse feature maps after the upsampling, which is followed by the convolutions with trainable filters. This upsampling strategy improves boundary delineation and reduces the number of parameters. The last decoder is connected to a softmax classifier to obtain a pixel-wise classification map. The softmax output has the number of channels equal to the number of class labels, and each channel represents the probabilities that the pixels belong to the corresponding class. The final predicted segmentation map corresponds to the class with maximum probability at each pixel. SegNet is efficient both in terms of memory and computational time during inference, so it is handy for practical applications.

3.3 Our Proposed Model: Dilated Convolutional Network (DCN)

Our proposed network architecture is shown in the Figure 3.15.

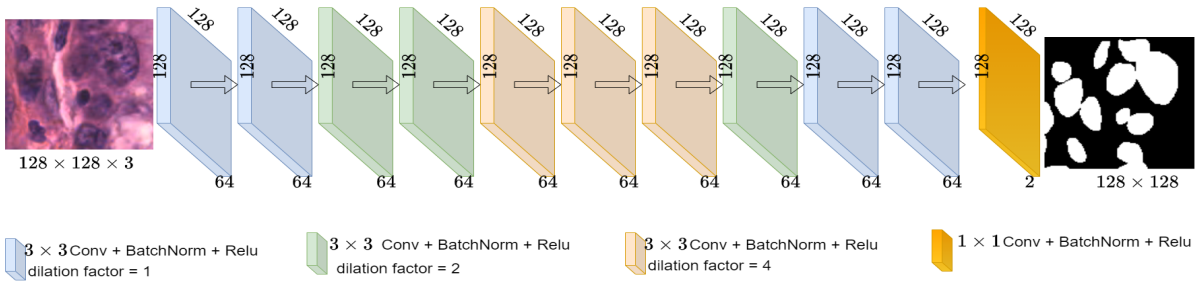

Figure 3.15: Dilated ConvNet Architecture

3.3.1 Dilated(Atros) Convolution:

One of the critical components of our design is the dilated convolutional layers [60]. A 2-dimensional dilated convolution can be defined as following:

$$y(m,n) = x * w(m,n) = \sum_{i=1}^{M}\sum_{j=1}^{N} x(m + r.i, n + r.j)w(i,j) \tag{3.1}$$

where $y(m,n)$ is the output of the convolution, $x(m,n)$ is the input, $w(i,j)$ is the filter with size $M \times N$ And r is a dilation factor. $r = 1$ will give us the usual convolution. For the input size of $m \times n$, kernel size k, padding p, stride s, and dilation factor r, we have the following output size:

$$\left\lfloor \frac{m + 2p - k - (k-1)(r-1)}{s} + 1 \right\rfloor \times \left\lfloor \frac{n + 2p - k - (k-1)(r-1)}{s} + 1 \right\rfloor \tag{3.2}$$

Figure 3.16 show convolution kernels with different dilation factors.

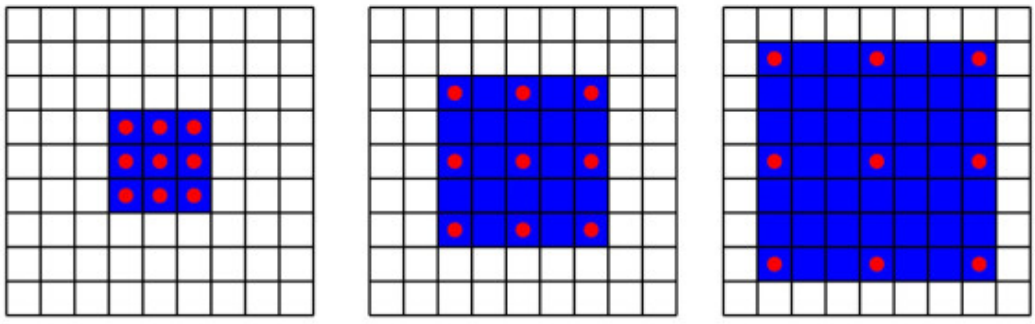

Figure 3.16: 3×3 convolution kernels with different dilation factors 1, 2 and 3 respectively. Red dots indicate nonzero values

In dilated convolutions, we expand the alignment of the kernel weights by the dilation factor. The weights are placed more sparse around a pixel by increasing this factor, which enlarges the kernel size. The receptive field or the region in the input space that contributes to feature extraction can be efficiently expanded while maintaining the resolution

by monotonously increasing the dilation factors through layers. Therefore the global information can be exchanged between layers without the loss of input resolution. When the dilation factor is greater than 1, the neighboring pixels in the output have non-overlapping receptive fields. So, the use of dilated convolutions can cause gridding artifacts. We can remove this by using dilated convolution layers with decreased dilation factors.

For a given image, we propose a semantic pixel-wise segmentation technique using dilated convolutions. The architecture of our network is based on SegNet, a deep convolutional encoder-decoder architecture. All the max pooling layers in the SegNet are removed for the encoder, and dilated convolution layers replace the convolutional layers with increased dilation factors to preserve image resolution. All max un pooling layers are removed for the decoder, and dilated convolution layers replace the convolutional layers with decreased dilation factors to remove gridding artifacts. We choose all dilation factors as 2^n to balance the computation load and the size of the receptive fields. The whole process is like enlarging the view gradually and then gradually focusing on the point of interest. We remove all max pooling layers because the networks' performance with and without max pooling layers are similar, yet there is more computational work with max pooling layers. In histopathology images, the percentage of regions occupied by cell nuclei is much less than the background regions. For the training data set we use, the percentage of cell nuclei regions is 23%. To balance the classes, we used inverse frequency weighting in the classification layer [19]. Table 3.1 shows the detailed network architecture of a SegNet and our dilated convolutional network (DCN).

Our DCN has 34 layers; each convolution layer (except the first and last one) has $3 \times 3 \times 64 \times 64$ weights and 64 bias values. Each batch normalization layer has 128 learnable parameters. Hence the total number of learnable parameters is 335,554. The corresponding SegNet (SegNet with encoding depth 3) has 45 layers, 373,638 learnable parameters. We consider U-Net with 64 output channels for the first encoder. U-Net has an architecture that

the number of output channels for each convolutional layer in the encoder is doubled as we advance. The number of output channels for each convolutional layer in the decoder is halved. U-Net with encoding depth 3 (U-Net3) has 46 layers, 7,697,410 learnable parameters, U-Net with encoding depth 4 (U-Net4) has 58 layers, 31,031,810 learnable parameters. Compared to U-Net, our DCN, which has much less learnable parameters (335,554 vs. 7,697,410 for U-Net3), is less likely to overfit the data. Another advantage of our DCN is the efficiency during inference. Our DCN is trained with image patches of fixed size; however, the trained DCN can be applied to images of any size, making the testing easy and efficient. U-Net has extra links between further-away layers; hence the testing image must have the same size as the training images. Our DCN keeps the original size of the input image at each layer; hence, the memory usage and computational load are higher than SegNet.

The run time of dilated convolution layers in the Matlab deep learning toolbox is much longer than the run time of the conventional convolution layers. To overcome this drawback, we design a method to convert dilated convolutions to conventional convolutions. This method works for all dilated convolutions with a dilation factor of 2^n. Assume matrix A is the input matrix, we can reorder the entries of A to get a matrix B, so that in B, all odd rows of A are before the even rows of A, and all odd columns of A are before the even columns of A. We call this procedure "matrix splitting" and the reverse operation "matrix merging". After matrix splitting, a dilated convolution on A with a dilation factor 2 is equivalent to a conventional convolution on B. Figure 3.17 shows matrix A and its reordering B. It also shows a dilated convolution on A and its corresponding convolution on B. The third column of Table 3.1 shows how to use this observation to implement our DCN to train it efficiently. However, the testing time is longer; half of the test time is for matrix splitting and merging. Thus it shows there is room for improvement for implementing dilated convolutions in neural network models.

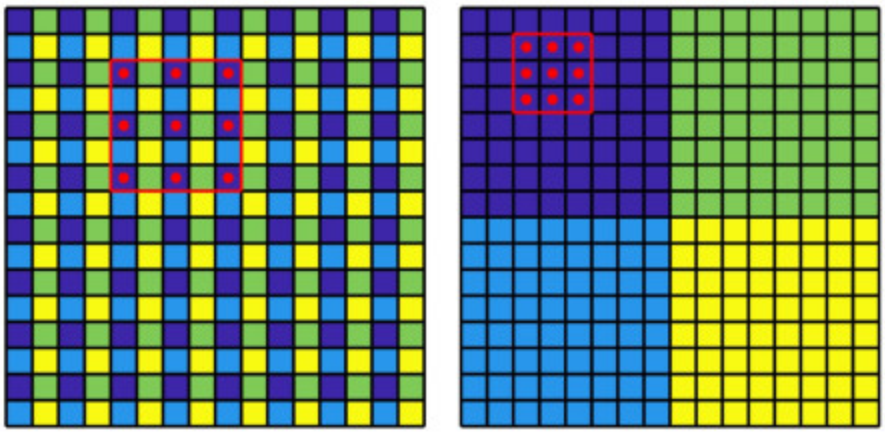

Figure 3.17: A matrix A and its reordering B. It also shows a dilated convolution on A and its corresponding convolution on B.

3.4 Dealing with Class Imbalance Problem

Since our dataset has images of which most of the pixels are related to the background, the result will be unreliable if we do training in this situation. We may get high accuracy even if it classifies most cell nuclei pixels as the background pixel. So we need to think about what we can do to overcome it. One of the ways is to consider different evaluation metrics rather than only assessing the global accuracy. Besides, we can add class weight to the loss function.

- Weighted cross-entropy loss

  Consider the binary cross-entropy loss :

$$Loss = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log p_i + (1 - y_i) \log (1 - p_i)] \tag{3.3}$$

  where y_i is the binary ground truth label, taking values 0 or 1, p_i is the predicted probability that the ith pixel belongs to the positive class or class 1, and N is the total number of pixels. In this loss function, every pixel has the same weight. Most of the values for this loss function is contributed by the background pixels or negative class

pixels as there are much more background pixels than the nuclei pixels, so the accuracy result based on this loss function may be biased. To overcome this we add weight to each pixel so that each class contributes to the loss function equally. We impose high weight for the class with less number of pixels and low weight for the class with more number of pixels. Here in our case we apply high weight for the positive class and low weight for the negative class. The weighted binary cross-entropy loss is defined as follows:

$$Weighted\ Loss = -\frac{1}{N}\sum_{i=1}^{N}[w_1 y_i \log p_i + w_0(1-y_i)\log(1-p_i)], \tag{3.4}$$

where

$$w_1 = \frac{\text{number of pixels in class 0}}{\text{total number of pixels}}, \tag{3.5}$$

$$w_0 = \frac{\text{number of pixels in class 1}}{\text{total number of pixels}}. \tag{3.6}$$

The weights defined above make both classes contribute the same to the loss function and hence will be useful in handling the class imbalance problem.

- Soft dice loss

  Another loss function that works well in the presence of imbalanced data is the soft dice loss [53], which is defined as follows:

$$L = 1 - \frac{2\sum y_{true} y_{pred}}{\sum y_{true}^2 + \sum y_{pred}^2} \tag{3.7}$$

where y_{true} and y_{pred} is the ground truth and the prediction respectively and the sum is over all the pixels. The second term in this equation measures the overlap between the ground truth and the prediction. We want to minimize the loss function. The higher the loss, the lower the number of overlap in the pixels and vice versa. The

fraction $\frac{2\sum y_{true}y_{pred}}{\sum y_{true}^2+\sum y_{pred}^2}$ demands that when the ground truth pixel is 1, the predicted pixel should also be close to 1. Similarly, when the ground truth class is 0, the predicted class should also be close to 0. Otherwise we will get a small number for this ratio and hence high loss.

The use of weighted cross-entropy loss results in the best performances in our experiments.

Table 3.1: Comparison of the network architectures of the SegNet and our Dilated Convolutional network, where "Conv" means "Convolutions" and D is the dilation factor. Third column shows how to use matrix splitting and merging procedures to implement dilated convolutions through efficient conventional convolutions.

	SegNet	Our DCN	Our DCN (for efficient training)
	128x128x3 Input (or 64x64 Input)	128x128x3 Input (or 64x64 Input)	128x128x3 Input (or 64x64) Input
Encoder	64 3x3 Conv Normalization & ReLU ReLU 64 3x3 Conv Normalization & ReLU Max Pooling 64 3x3 Conv Normalization & ReLU 64 3x3 Conv Normalization & ReLU Max Pooling 64 3x3 Conv Batch Normalization ReLU 64 3x3 Conv Batch Normalization ReLU Max Pooling	64 3x3 Conv, D=1 Normalization & ReLU ReLU 64 3x3 Conv, D=1 Normalization & ReLU 64 3x3 Conv, D=2 Normalization & ReLU 64 3x3 Conv, D=2 Normalization & ReLU 64 3x3 Conv, D=4 Batch Normalization ReLU 64 3x3 Conv, D=4 Batch Normalization ReLU	64 3x3 Conv Normalization & ReLU ReLU 64 3x3 Conv Normalization & ReLU Matrix Splitting 64 3x3 Conv Normalization & ReLU 64 3x3 Conv Normalization & ReLU Matrix Splitting 64 3x3 Conv Batch Normalization ReLU 64 3x3 Conv Batch Normalization ReLU
Decoder	Max Unpooling 64 3x3 Conv Normalization & ReLU 64 3x3 Conv Normalization & ReLU Max Unpooling 64 3x3 Conv Normalization & ReLU 64 3x3 Conv Normalization & ReLU Max Unpooling 64 3x3 Conv Normalization & ReLU 2 3x3 Conv Normalization & ReLU	64 3x3 Conv, D=4 Normalization & ReLU 64 3x3 Conv, D=2 Normalization & ReLU 64 3x3 Conv, D=1 Normalization & ReLU 64 3x3 Conv, D=1 Normalization & ReLU 2 1x1 Conv	64 3x3 Conv Normalization & ReLU Matrix Merging 64 3x3 Conv Normalization & ReLU Matrix Merging 64 3x3 Conv Normalization & ReLU 64 3x3 Conv Normalization & ReLU 2 1x1 Conv
	Softmax Pixel Classification	Softmax Pixel Classification	Softmax Pixel Classification

CHAPTER 4

DATA PREPARATION AND PREPROCESSING [1]

4.1 Dataset

4.1.1 Synthetic Data Sets

We generate two data sets of random triangles. Each contains 200 training images and 100 test images of size 64×64. Each triangle in the first data set has an uniform foreground and an uniform background. Each triangle in the second data set has a textured foreground and a textured background. Figure 4.1 shows some sample training images in the first data set. Figure 4.2 shows some sample training images in the second data set.

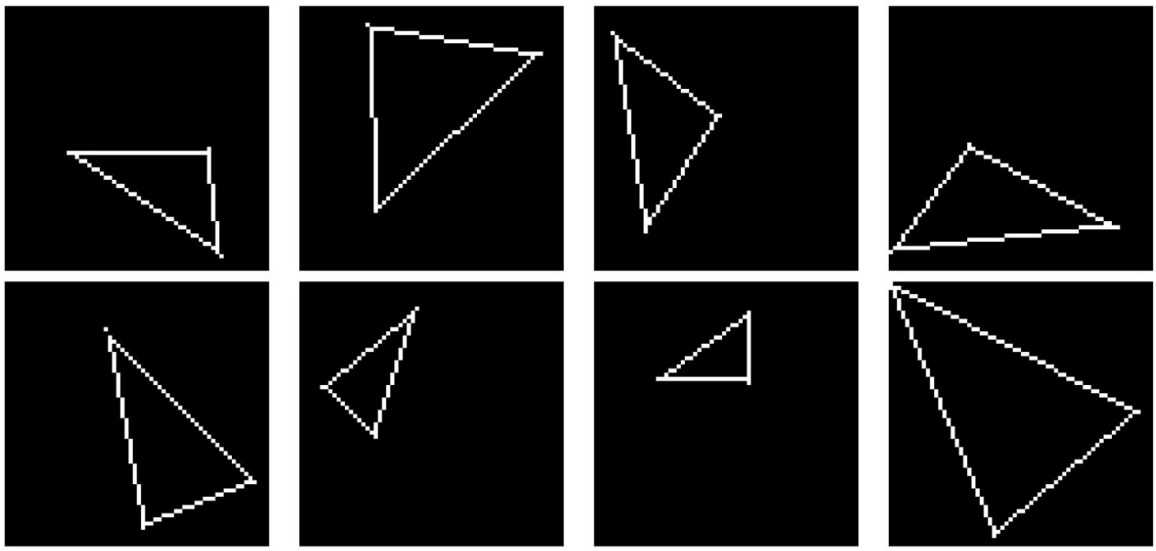

Figure 4.1: Sample training images in the triangle data set with a uniform foreground and a uniform background.

[1]This chapter includes verbatim excerpts from
K C Khatri,R.; Caseria, B.; Lou, Y.; Xiao, G. and Cao, Y. "Automatic Extraction of Cell Nuclei Using Dilated Convolutional Network", Inverse PROBLEMS AND Imaging, Vol. 15, No. 1, 27-40, 2021.

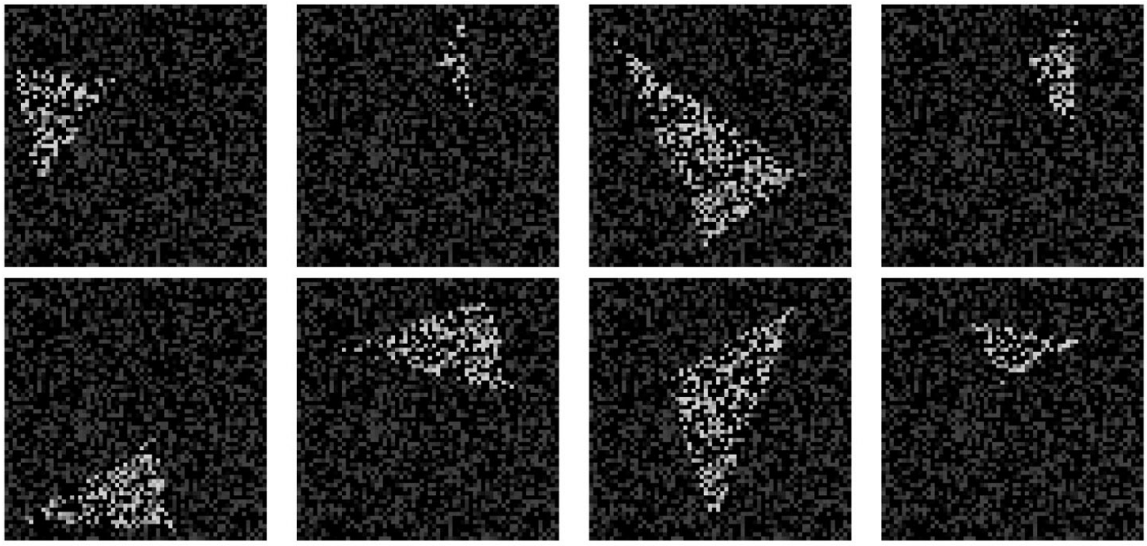

Figure 4.2: Sample training images in the triangle data set with a textured foreground and a textured background.

4.1.2 H & E-stained Pathology Image Data Sets

We also test the proposed method on a publicly available dataset of real H & E-stained pathology images [29]. The dataset consists of pathologically labeled images with annotated boundaries of each cell. The data set was generated using the H & E-stained tissue sample images captured at 40x magnification from The Cancer Genomic Atlas (TCGA), one image per patient. This data set contains six tissue samples of patients from each of the following types of cancer: lung, breast, kidney, prostate. We tested our methods on those images. Training data consists of 16 images (four lungs, four prostate, four breast & four kidney cancer) and test data consists of 8 images, two from each category. For the training purpose, we have split images into patches of size $128 \times 128 \times 3$. Figure 4.3 shows several normalized patches and corresponding manual segmentations from the data set.

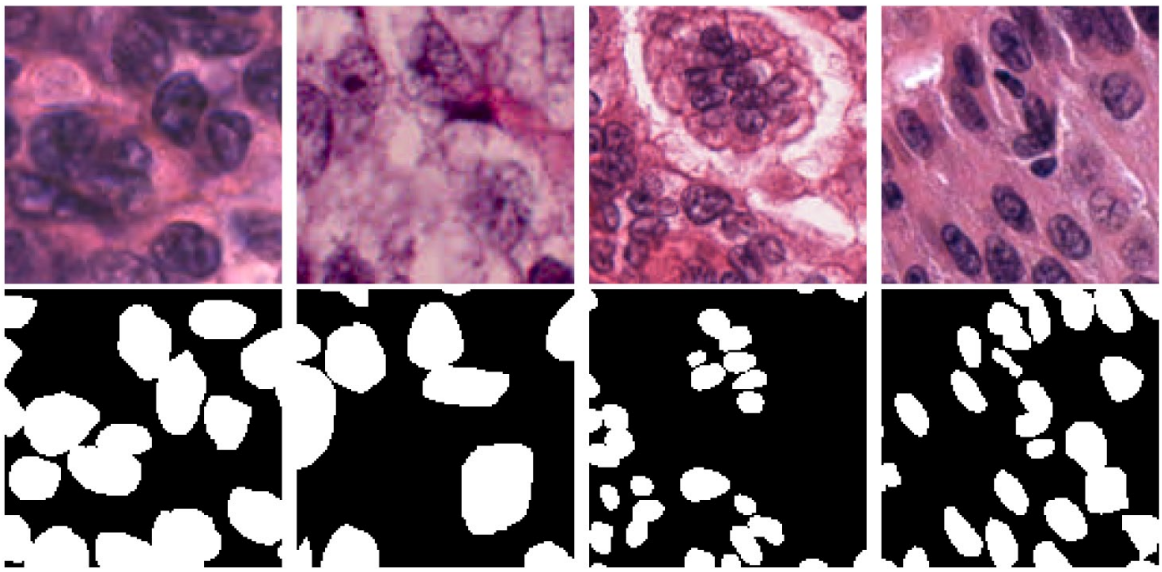

Figure 4.3: Sample normalized image patches and corresponding manual segmentations from the dataset.

4.2 Color Normalization

One of the challenges of nuclei segmentation is the color variation in histopathology images due to the difference in H&E reagents, staining process, and sensor response. Reinhard color normalization [46] is a method of color normalization where the mean and standard deviation of each channel of the image is matched to that of the target employing a set of linear transforms in Lab colorspace. First, we convert both the target image and source image from RGB color space to L*a*b* color space. Then we do the following computation in each channel to normalize the color:

$$\begin{aligned} \mathrm{S}(l)_{norm} &= \frac{(\mathrm{S}(l) - \overline{\mathrm{S}(l)})}{\sigma_{\mathrm{S}(l)}} * \sigma_{\mathrm{T}(l)} + \overline{\mathrm{T}(l)} \\ \mathrm{S}(a)_{norm} &= \frac{(\mathrm{S}(a) - \overline{\mathrm{S}(a)})}{\sigma_{\mathrm{S}(a)}} * \sigma_{\mathrm{T}(a)} + \overline{\mathrm{T}(a)} \\ \mathrm{S}(b)_{norm} &= \frac{(\mathrm{S}(b) - \overline{\mathrm{S}(b)})}{\sigma_{\mathrm{S}(b)}} * \sigma_{\mathrm{T}(b)} + \overline{\mathrm{T}(b)} \end{aligned} \tag{4.1}$$

where $\mathrm{S}(l)_{norm}$, $\overline{\mathrm{S}(l)}$, $\sigma_{S(l)}$, $\sigma_{\mathrm{T}(l)}$ and $\overline{\mathrm{T}(l)}$ denotes the normalized source, mean of source, standard deviation of source, standard deviation of target and mean of target respectively for the channel l and so on.

After this process, we revert the image thus obtained again to the RGB colorspace.

All images are color normalized by Reinhard method using the Stain Normalisation Toolbox for Matlab [24] before training and testing. Figure 4.4 shows an example of Reinhard color normalization.

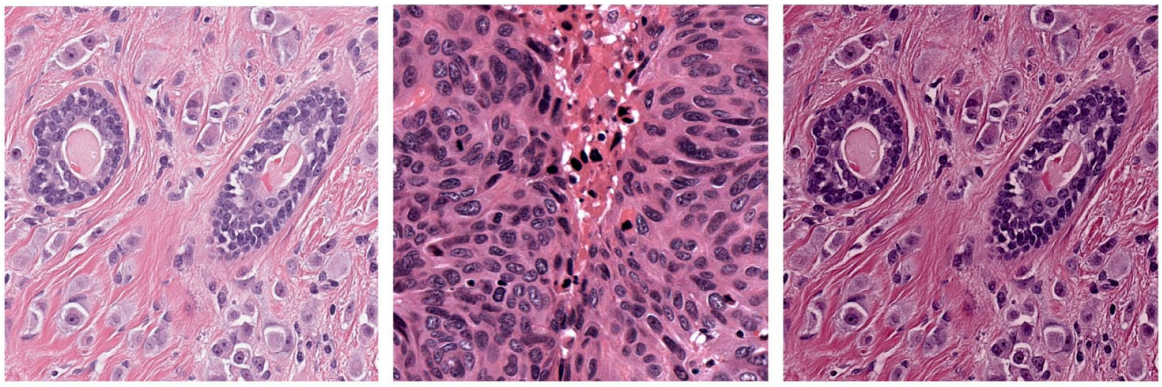

Figure 4.4: Left: Source image, Middle: Target image and Right: Normalized image

4.3 Data Augmentation

One of the challenges in medical image analysis is the availability of training data [27]. Typically we have fewer training data in medical image analysis tasks than in other computer vision tasks. However, since deep networks usually require lots of data to feed in, there is a high chance of being overfitted if we train the model using only the low amount of data available. Data augmentation is a regularization technique that can be used to create artificial training data. It is one of the many popular methods to prevent overfitting of the model, and generally, we get better performance after data augmentation [50].

During training, reflections about the x-axis are randomly performed on training data. In addition, random translations along the x-axis and y-axis (within ± 10 pixels) are also

performed. The corresponding transformations should also be applied to the corresponding training labels when we apply these transformations to the training images. Augmentation done this way will not increase the total number of training images at each iteration, making the training process more efficient.

CHAPTER 5

EXPERIMENTS AND RESULTS[1]

5.1 DCN training

We implement our method using Matlab deep learning toolbox [37]. Original tissue images and corresponding labeled ground-truth segmentation masks are partitioned into $128 \times 128 \times 3$ non-overlapping image patches. Each patch is normalized to have a zero mean. The extracted patches are partitioned randomly into two sets; 80% of the patches are for training, and 20% are for validation. To train the network, we try both the stochastic gradient descent with momentum (SGDM) optimizer [44], and the Adam optimization algorithm [26]. The momentum value used for SGDM is 0.9. A fixed learning rate of 0.001 is used. L_2 regularization with a factor of 0.0005 is used for the weights to the loss function to reduce overfitting. A Mini-batch of four observations is used at each iteration, and the maximum number of training epochs is set to 30. The training data and validation data are shuffled before each training epoch. The model is tested against the validation data after each epoch during the training. Our method is efficient enough to run on a personal laptop.

5.2 Evaluation Metrics

Let us first define the confusion matrix. The confusion matrix is shown in Figure 5.1.

[1]This chapter includes verbatim excerpts from
K C Khatri,R.; Caseria, B.; Lou, Y.; Xiao, G. and Cao, Y. "Automatic Extraction of Cell Nuclei Using Dilated Convolutional Network", Inverse PROBLEMS AND Imaging, Vol. 15, No. 1, 27-40, 2021.

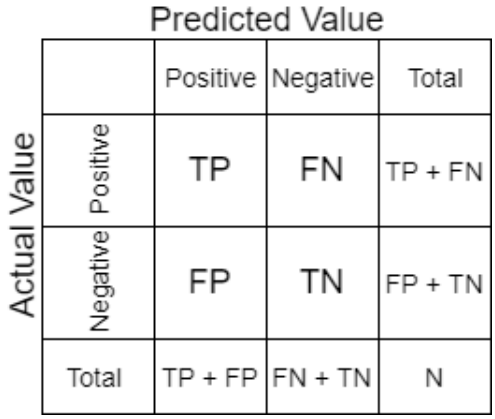

Figure 5.1: Confusion matrix.

Here TP is the number of true positives, i.e., the number of pixels that are ground truth pixels among the detected, FP is the number of false positives, i.e., the number of pixels that are not ground truth pixels among the detected, FN is the number of false negatives, i.e., the number of ground truth pixels that have not been detected, TN is the number of true negatives, i.e., the number of ground truth pixels that has been detected.

We used five metrics to quantitatively measure the segmentation results. The measures are defined as follows:

- GlobalAccuracy: the percentage of correctly classified pixels among all image pixels, regardless of whether the pixel is a ground truth nuclei pixel or a background pixel.

$$\text{GlobalAccuracy} = \frac{TP + TN}{TP + FN + FP + TN}. \tag{5.1}$$

- MeanAccuracy: the percentage of correctly identified pixels for each class, averaged over the two classes in the image, nuclei or background.

$$\begin{aligned} \text{MeanAccuracy} &= \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{FP + TN}\right), & (5.2) \\ &= \frac{1}{2}\left(\text{Sensitivity} + \text{Specificity}\right), & (5.3) \end{aligned}$$

where

$$\text{Sensitivity} = \frac{TP}{TP+FN}, \tag{5.4}$$

$$\text{Specificity} = \frac{TN}{FP+TN}. \tag{5.5}$$

- MeanIoU: Intersection over union (IoU), also known as the Jaccard similarity coefficient. For each class, IoUscore $= \frac{TP}{TP+FP+FN}$. MeanIoU is the average IoU score of all classes in the image.

$$\text{MeanIoU} = \frac{1}{2}\left(\frac{TP}{TP+FP+FN} + \frac{TN}{TN+FP+FN}\right). \tag{5.6}$$

- WeightedIoU: Average IoU of all classes in the image, weighted by the number of pixels in each class.

- MeanBFScore: For each class, BF (Boundary F1) contour matching score is defined by $BF = \frac{2TP}{2TP+FP+FN}$. MeanBFScore is the average BF score of each class in the image.

5.3 Experiments

5.3.1 Test on synthetic data sets

We tested our DCN model on two synthetic data sets of random triangles, one with a uniform foreground and a uniform background. The other was a textured foreground and a textured background. Our DCN is created according to the architecture shown in Table 3.1, with the input size as 64×64. All networks are trained using the SGDM optimizer with the same parameters as described in Section 5.1, except that U-Net3 and U-Net4 are trained with a smaller learning rate of 0.0001.

Figure 5.2 and Figure 5.3 show the segmentation results on the triangle data sets. Table 5.1 shows the quantitative metrics of the segmentation results. Here we compare the performance of SegNet, U-Net3, U-Net4, and our DCN. For the triangle data set with a

Table 5.1: Quantitative metrics of the segmentation results on triangle data sets. Best values are displayed in bold.

	Triangle Dataset	Global Accuracy	Mean Accuracy	Mean IoU	Weighted IoU	Mean BFScore
SegNet	Uniform	0.9325	0.9508	0.7829	0.8882	0.4172
U-Net3	Uniform	0.9694	0.9531	0.8784	0.9438	0.6572
U-Net4	Uniform	**0.9974**	**0.9953**	**0.9884**	**0.9949**	**0.9488**
Our DCN	Uniform	0.9952	0.9941	0.9786	0.9906	0.8946
SegNet	Textured	0.8764	0.9280	0.6818	0.8139	0.3605
U-Net3	Textured	0.8119	0.8614	0.5855	0.7359	0.2157
U-Net4	Textured	0.7250	0.8148	0.4945	0.6391	0.2000
Our DCN	Textured	**0.9658**	**0.9741**	**0.8728**	**0.9386**	**0.4638**

uniform foreground and uniform background, our method and U-Net4 have similar performance. Both methods achieve better results than SegNet and U-Net3. Our method is much better for the triangle data set with a textured foreground and a textured background than SegNet and U-Net. The main problem of U-Net in this situation is overfitting. The accuracy of the training set is very high; however, the performance on the test set is much worse. This is true partially because U-Net has a lot more learnable parameters than SegNet and our DCN.

5.3.2 Test on Real H&E-stained Pathology Images

We also test the proposed method on a publicly available dataset of H&E-stained pathology images [29]. All networks are trained using the Adam optimization algorithm with the same parameters as described in Section 5.1. Figure 5.4 shows the segmentation results on several patches along with the ground truth. Compared to SegNet and U-Net, our method produces better segmentation around the cell boundaries. Table 5.2 shows the quantitative metrics of the segmentation results. Our method achieves better results than SegNet and U-Net in general. Table 5.3 shows the training and testing time of SegNet, U-Net3, and our DCN using the H&E-stained image data set. The tests are performed on a Windows 10 personal

laptop with 2.8 GHz Intel(R) Core (TM) i7, 16GB RAM, and one GeForce GTX 1050 graph card. Since U-Net cannot handle input images with different sizes, the testing time of U-Net includes time to divide test images into patches and stitch image patches to the original dimensions.

Table 5.2: Quantitative metrics of the segmentation results on the H&E-stained image data set. Best values are displayed in bold.

	Image Set	Global Accuracy	Mean Accuracy	Mean IoU	Weighted IoU	Mean BFScore
SegNet	Lung	0.8819	0.8975	0.7324	0.8074	0.9204
U-Net3	Lung	0.8917	0.9013	0.7475	0.8190	0.9266
U-Net4	Lung	0.8929	0.8966	0.7484	0.8193	0.9343
Our DCN	Lung	**0.9045**	**0.9033**	**0.7690**	**0.8355**	**0.9448**
SegNet	Breast	0.8691	0.8990	0.7002	0.7917	0.8900
U-Net3	Breast	0.8829	0.9051	0.7183	0.8124	0.8743
U-Net4	Breast	0.8775	0.8977	0.7086	0.8057	0.8700
Our DCN	Breast	**0.9047**	**0.9123**	**0.7538**	**0.8415**	**0.9210**
SegNet	Kidney	0.9122	0.9249	0.7290	0.8634	0.9425
U-Net3	Kidney	0.9133	**0.9281**	0.7259	0.8639	0.9218
U-Net4	Kidney	0.8993	0.9145	0.7013	0.8462	0.9306
Our DCN	Kidney	**0.9329**	0.9277	**0.7725**	**0.8911**	**0.9634**
SegNet	Prostate	0.8956	0.9142	0.7533	0.8271	0.9105
U-Net3	Prostate	0.8949	0.9041	0.7496	0.8255	0.9047
U-Net4	Prostate	0.8961	0.9032	0.7510	0.8271	0.9090
Our DCN	Prostate	**0.9211**	**0.9163**	**0.7962**	**0.8632**	**0.9336**
SegNet	Overall	0.8897	0.9000	0.7383	0.8184	0.9159
U-Net3	Overall	0.8957	0.8976	0.7467	0.8264	0.9069
U-Net4	Overall	0.8914	0.8905	0.7380	0.8201	0.9110
Our DCN	Overall	**0.9158**	**0.9039**	**0.7815**	**0.8548**	**0.9407**

Table 5.3: Comparison of the training and testing time of SegNet, U-Net3 and our DCN.

	SegNet	U-Net3	Our DCN	Our DCN (efficient)
Training Time	15 min 14 sec	18 min 36 sec	26 min 47 sec	19 min 45 sec
Testing Time	7.6 sec	37.7 sec	10.1 sec	19.9 sec

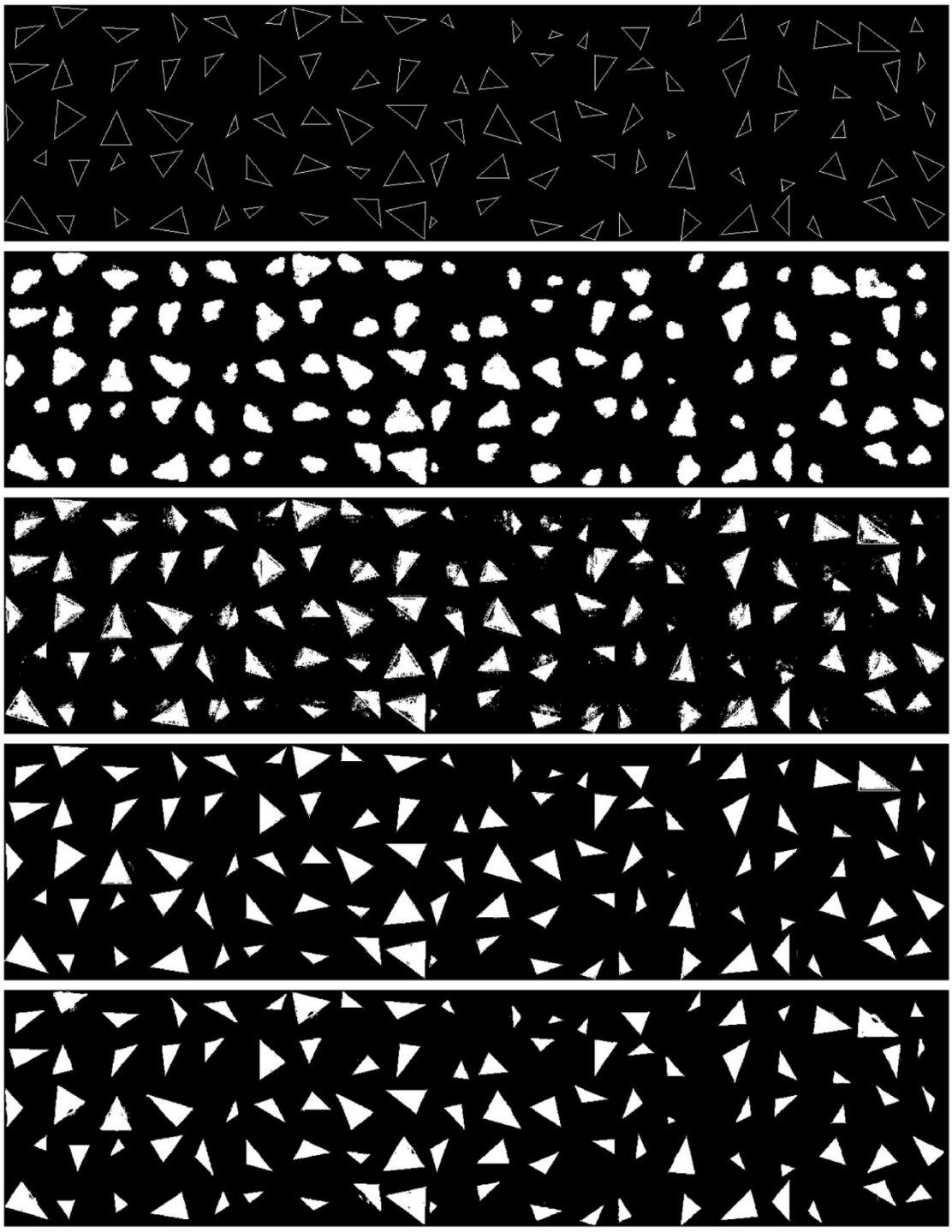

Figure 5.2: Test images (1st row) with corresponding segmentations using SegNet (2nd row), U-Net3 (3rd row), U-Net4 (4th row), and our DCN (5th row).

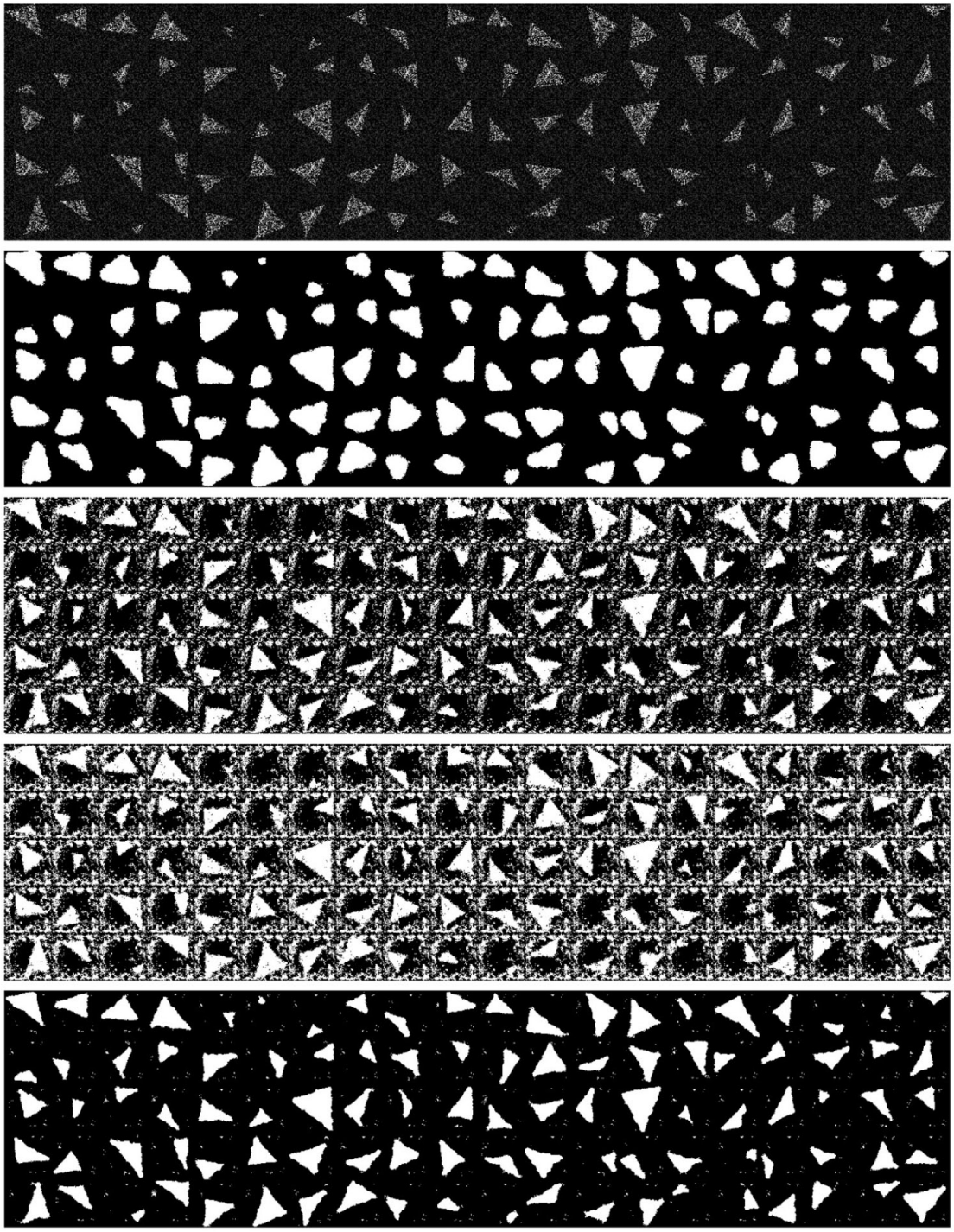

Figure 5.3: Test images (1st row) with corresponding segmentations using SegNet (2nd row), U-Net3 (3rd row), U-Net4 (4th row), and our DCN (5th row).

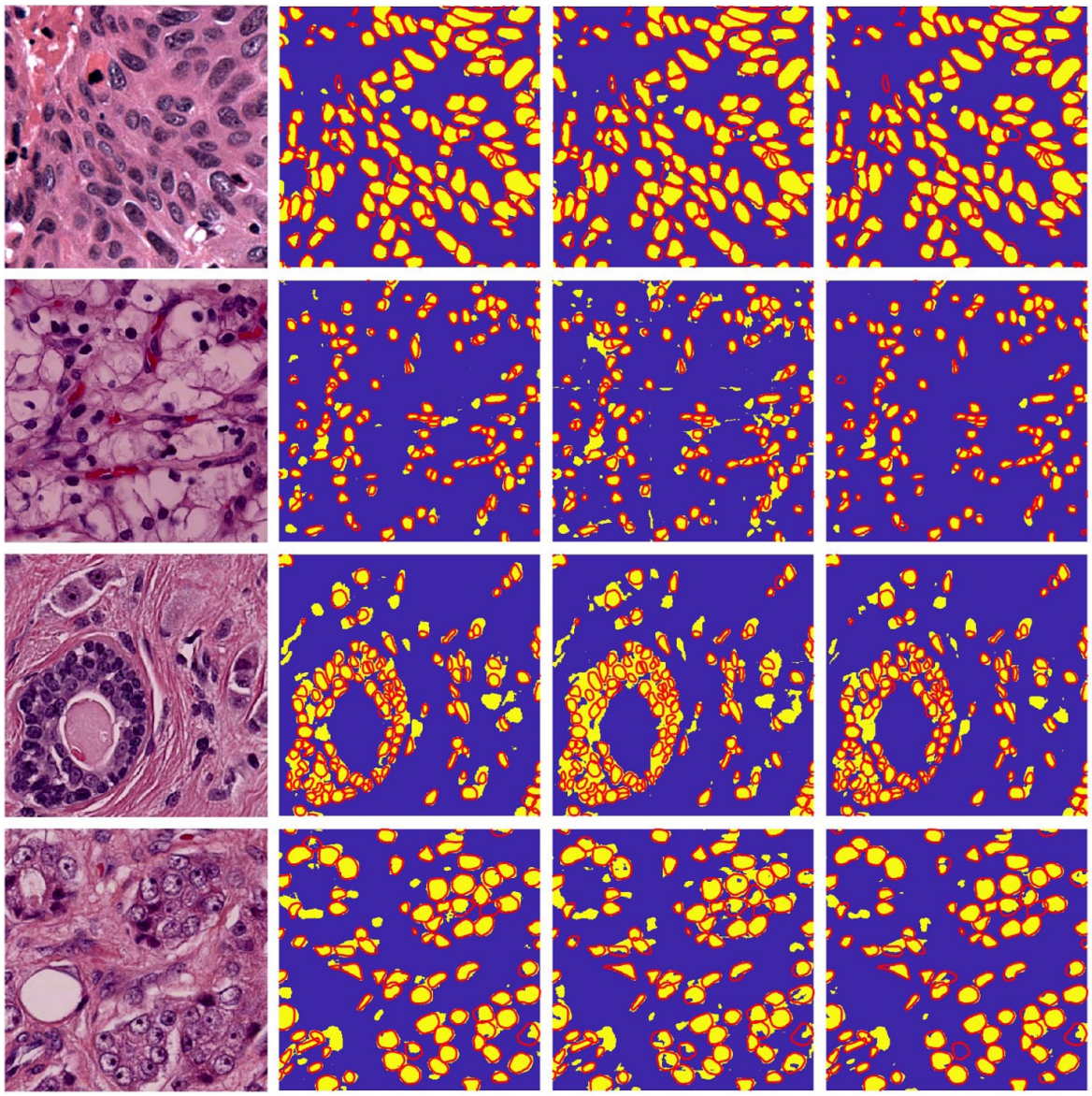

Figure 5.4: Test images (1st column) with corresponding segmentations using SegNet (2nd column), U-Net3 (3rd column), and our DCN (4th column). Ground truth contours are plotted in red.

CHAPTER 6

CELL NUCLEI DETECTION AND INSTANCE SEGMENTATION

6.1 Object Detection

Object detection involves two things:

- Object localization.
- Object classification.

Object localization is the task of recognition of one or more objects in an image and finding their locations by creating a bounding box around each object. Object classification is the task of classifying the objects detected by bounding boxes.

6.2 Single Shot MultiBox Detector (SSD)

SSD is a one-stage object detection model proposed by Liu et al. [34]. It is faster than two-stage models such as the Faster R-CNN detector [8, 47] and it outperforms a comparable Faster R-CNN detector. SSD is also more accurate than other one-stage object detectors such as YOLO [45].

SSD model also utilizes the CNN architecture for classification as the base network. The classification layer is removed from CNN, and an auxiliary structure is added to the network for the object detection part. Figure 6.1 illustrate an example of the model in [34]. The network structure of the SSD model consists of three parts: base network, extra feature layers, and the non-maximum suppression layer. In the example in Figure 6.1, VGG-16 is used as a base network. Classification layers were removed from VGG-16. The feature map "Conv 4-3" from base network VGG-16 is directly fed to the object detection part, and extra convolutional feature layers are added to the end of the truncated VGG-16 network ("Conv

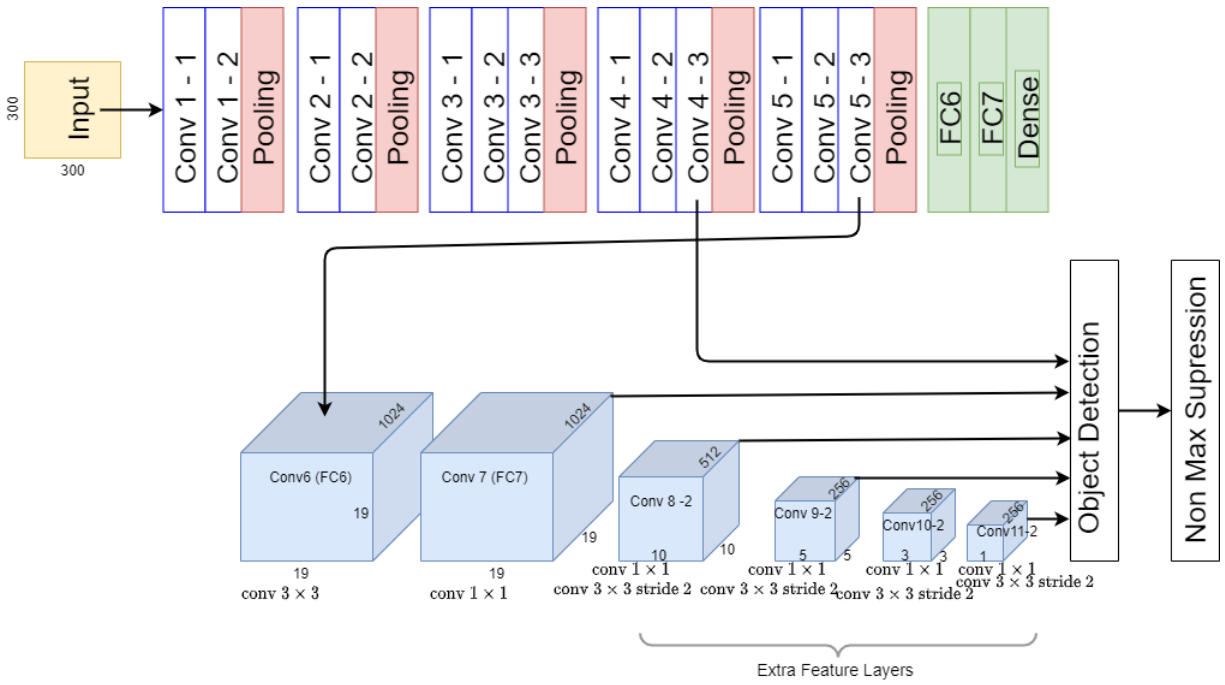

Figure 6.1: SSD Model architecture

5-3"). The sizes of these layers are decreased gradually to allow for detection at different scales.

A set of default bounding boxes (anchor boxes) are associated with each feature map cell for multiple feature maps to generate possible object locations. The network will output a classification score and the location offset for each anchor box. The anchor box strategy allows us to evaluate the object prediction at once, which makes SSD computational efficient. These boxes pass through a non-max suppression layer, which evaluates the intersection-over-union (IoU) distance metric with the ground truth box and keeps only the boxes with maximum IoU.

6.3 DCNSSD: Our DCN Based SSD

6.3.1 Network Architecture

Our DCN is very successful in semantic segmentation of the cell nuclei in H&E-stained pathology images. To detect each individual cell nucleus, we designed an SSD model based on our DCN, named DCNSSD. We implemented our method using Matlab deep learning toolbox [38]. Figure 6.2 shows the architecture of our DCNSSD.

6.3.2 Anchor Boxes Proposals

The cell nuclei have different shapes and sizes. To decide an optimal set of anchor boxes, we estimate the anchor boxes from the training data. To visualize the ground truth box distribution, we plot the box area vs. aspect ratio of the ground truth bounding boxes in Figure 6.3. It helps get an idea about the range of size of the bounding boxes in the data.

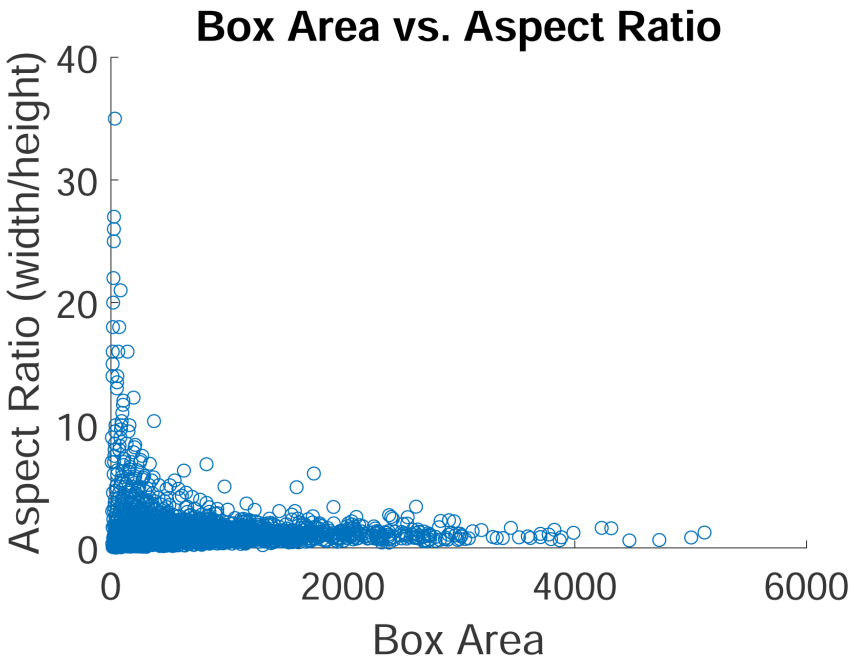

Figure 6.3: Box area vs aspect ratio of ground truth bounding boxes.

The number of the anchor boxes is a hyperparameter that we need to choose wisely. From Figure 6.3 above, it is difficult to determine the clusters manually, so we need to use some clustering algorithm. We use a k-means clustering algorithm using the IoU distance metric. Then the boxes of similar aspect ratios and sizes are clustered together. Figure 6.4 displays the number of anchors and their corresponding mean IoU score. It clearly shows that the 10 is a reasonable choice because it gives the highest mean IoU.

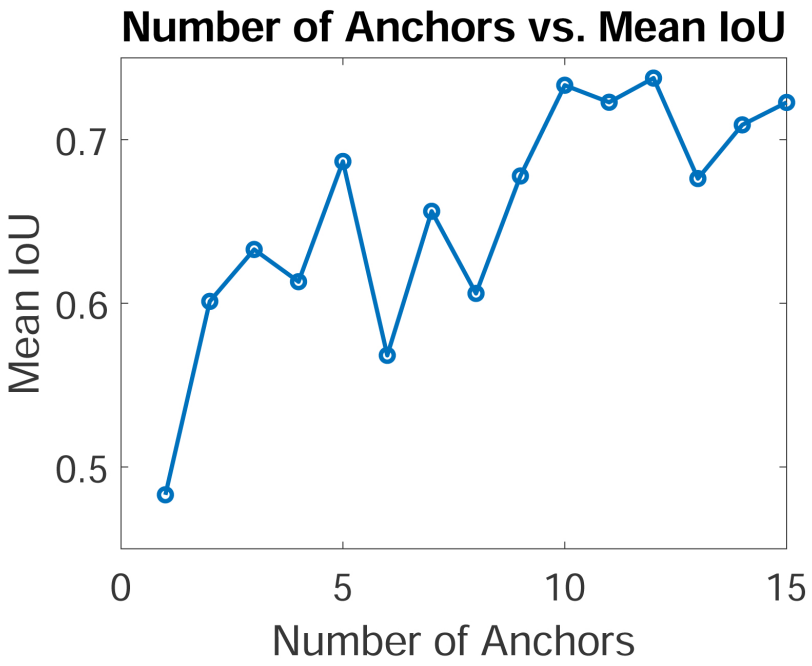

Figure 6.4: Numbers anchors vs mean IoU

We utilize those ten anchor boxes given by the k-means algorithm [39]. These anchor boxes are placed in a convolutional manner in such a way that the position of each box relative to its corresponding cell is fixed [34]. For each anchor box, the SSD network looks if there is an object and predicts the probability of each class, i.e., the confidence score for each category. Also, for each anchor boxes, it predicts box offsets with ground truth bounding box.

Our feature layers start from the layer relu10, which has dimension $128 \times 128 \times 64$. We have to calculate the prediction scores for two classes (cell and background) and four offsets

relative to the anchor box shape for each of 10 boxes, so a total of $(2+4) \times 10$ filters are applied around each location in the feature map. So total, we yield $(2+4) \times 10 \times 128 \times 128$ outputs for a 128×128 feature map. Among these, 10×2 (number of anchors $\times$ number of classes) filters are used to classification part, and $4\times$ number of anchors filters are used in bounding box regression.

6.3.3 Encode Ground Truth Boxes

After obtaining the anchor boxes, we need to encode the ground truth bounding boxes as anchor boxes. At each feature map cell, we compare the ten anchor boxes with all the ground truth bounding boxes and calculate the IoU distance. If the IoU is greater than 0.5, we consider it a match, and label the corresponding anchor box 1, otherwise, label it as 0. The location offsets (cx, cy, w, h) are calculated as in [34] as follows:

$$cx = (cx_g - cx_a)/w_a \tag{6.1}$$

$$cy = (cy_g - cy_a)/h_a \tag{6.2}$$

$$w = \log\left(\frac{w_g}{w_a}\right) \tag{6.3}$$

$$h = \log\left(\frac{h_g}{h_a}\right) \tag{6.4}$$

where (cx, cy) is the offsets of the centers and (w, h) is the offsets of the width and height. Here (cx_g, cy_g) and (cx_a, cy_a) represent the centers of the ground truth box and the anchor box respectively; (w_g, h_g) and (w_a, h_a) represent the width and height of the ground truth box and anchor box respectively.

6.3.4 Loss Function

The total loss L is a weighted sum of the focal loss L_{obj} for classification of the anchor boxes at each pixel and the smooth L_1 loss L_{loc} for regression of the location offsets.

$$L = \alpha L_{obj} + L_{loc}, \tag{6.5}$$

where the balancing parameter α was set to be 2 during the training.

The focal loss L_{obj} is defined as in [32] as follows:

$$L_{obj} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{c} y_{ij}(1-p_{ij})^{\gamma}\log(p_{ij}) \tag{6.6}$$

where N is the number of observations, c is the number of classes, y_{ij} are ground truth labels of the anchor boxes at each pixel, p_{ij} are the corresponding predictions from the network, and γ is the focusing parameter. In our case, N equals to the number of pixels 128×128 multiply the number of anchor boxes 10, that is, 163840. We have two classes, hence $c = 2$. The focusing parameter γ was set to be 0.25 during the training.

The smooth L_1 loss L_{loc} is defined as in [34] as follows:

$$L_{loc} = \sum_{i\in Pos}^{M}\sum_{m\in\{cx,cy,w,h\}} \text{smooth}_{L1}(l_i^m - g_i^m) \tag{6.7}$$

$$\text{smooth}_{L1}(z) = \begin{cases} 0.5z^2 & \text{if } |z| < 1; \\ |z| - 0.5 & \text{otherwise.} \end{cases} \tag{6.8}$$

where i is the index of anchor box with label 1, M is the number of matched anchor boxes, l is the predicted box offsets and g is the encoded ground truth box offsets.

6.3.5 DCNSSD Training and Experimental Results

We trained the model using a stochastic gradient descent with momentum (SGDM) algorithm [44]. The momentum value used for SGDM is 0.9. The initial learning rate is 0.1 and with a learning rate drop at every 30 epochs by the factor of 0.8. The maximum number of epochs for training is set to be 300 epochs and a mini-batch with 16 observations at each iteration was used. The training data are shuffled before each training epoch. Figure 6.5 shows some of the cell nuclei detection results of our DCNSSD on the testing data. The results is promising.

6.4 Instance Segmentaion

Instance segmentation involves object detection and semantic segmentation. Given an input image, we want to output all instances of those classes, and for each instance, we want to segment out the pixels that belong to that instance. Our problem of cell segmentation consists of segmenting cells and backgrounds. So basically, we have one class to detect. So instance segmentation problem involves detecting all nuclei objects and at the same time distinguish each of those cell nuclei. We can combine the results of DCN and DCNSSD to achieve instance segmentation of the cell nuclei.

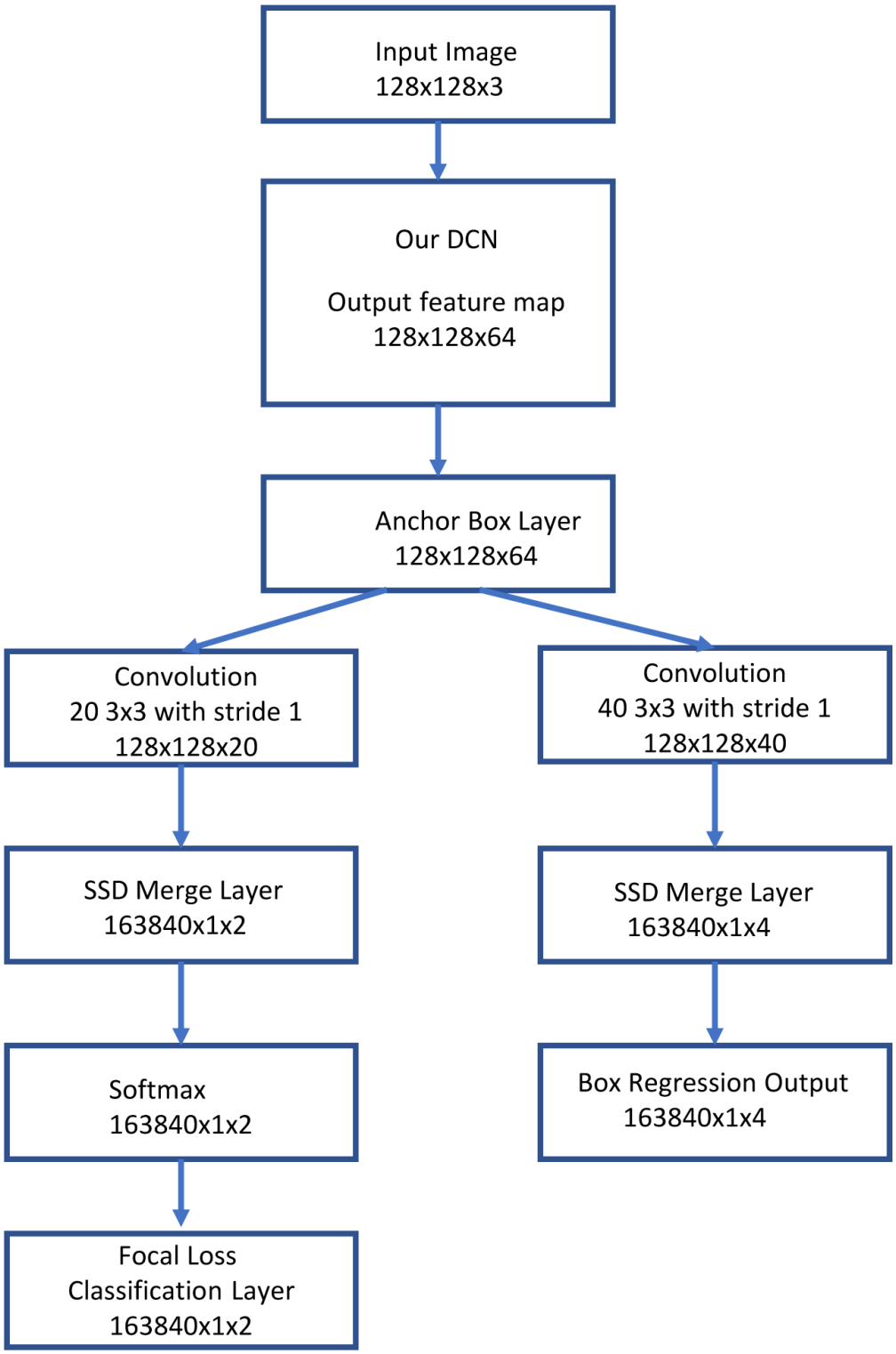

Figure 6.2: The architecture of our DCNSSD.

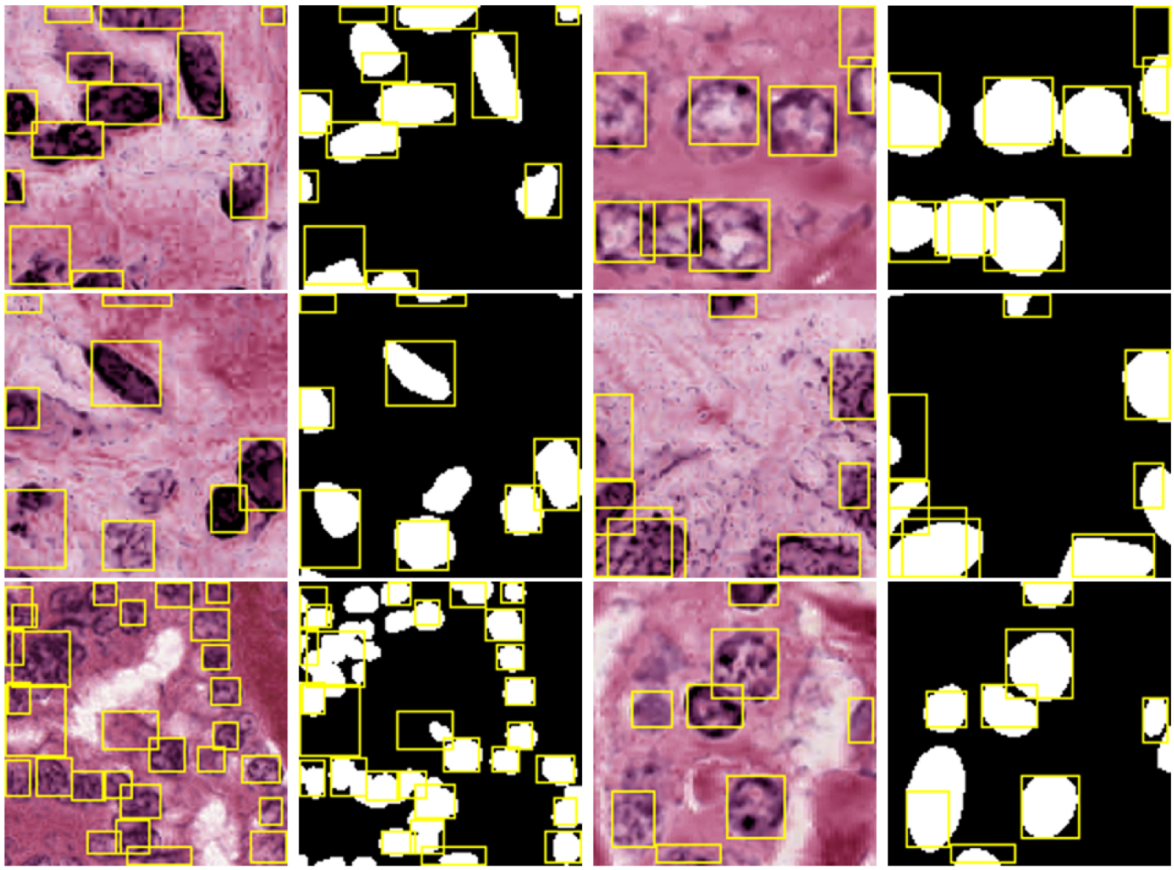

Figure 6.5: Some cell nuclei detection results of our DCNNSSD model on testing data. First column and the third column show the predicted cell nuclei locations on the H&E-stained input images, second column and the fourth column show the predicted cell locations on the ground truth mask.

CHAPTER 7

CONCLUSIONS

In this work, we developed a cell nuclei segmentation method using dilated convolutional neural network. We showed that dilated convolutional layers with increasing dilation factors and then followed by dilated convolutional layers with decreasing dilation factors is superior in extract useful information from textured images, which is ideal for extracting cell nuclei in H&E-stained pathology images. We also proposed a method to implement dilated convolutions efficiently.

We first designed a dilated convolutional neural network DCN for semantic segmentation of cell nuclei in H&E-stained pathology images. Experiments show that the proposed method achieves a higher level of accuracy than the state-of-the-art. In particular, the proposed method works better in detecting cell boundaries.

Semantic segmentation doesn't separate objects of the same class. There are some cell nuclei which are overlapped or stick together. To detect the instance of each cell nuclei, we designed a SSD cell nuclei detector DCNSSD using our DCN as the base network. The preliminary results is promising.

Future work involves adding the semantic segmentation branch in the network to get the pixel-wise cell nuclei instance segmentation in one shot. Another direction of future work is to use the segmentation results to analyze the morphological properties of the tumor cells and predict the progress of cancer.

REFERENCES

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[2] Christopher M Bishop. Pattern recognition. *Machine learning*, 128(9), 2006.

[3] Gabriel J Brostow, Julien Fauqueur, and Roberto Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88–97, 2009.

[4] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[5] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2018.

[6] Di Feng, Christian Haase-Schuetz, Lars Rosenbaum, Heinz Hertlein, Claudius Glaeser, Fabian Timm, Werner Wiesbeck, and Klaus Dietmayer. Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[7] Andrew H Fischer, Kenneth A Jacobson, Jack Rose, and Rolf Zeller. Hematoxylin and eosin staining of tissue and cell sections. *Cold spring harbor protocols*, 2008(5):pdb–prot4986, 2008.

[8] Ross Girshick. Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[9] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[11] Mohammad Havaei, Axel Davy, David Warde-Farley, Antoine Biard, Aaron Courville, Yoshua Bengio, Chris Pal, Pierre-Marc Jodoin, and Hugo Larochelle. Brain tumor segmentation with deep neural networks. *Medical image analysis*, 35:18–31, 2017.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *Signal Processing Magazine*, 2012.

[14] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[15] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[16] Peter J Huber et al. Robust regression: asymptotics, conjectures and monte carlo. *Annals of statistics*, 1(5):799–821, 1973.

[17] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[18] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*, 2017.

[19] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002.

[20] Feng Jiang, Aleksei Grigorev, Seungmin Rho, Zhihong Tian, YunSheng Fu, Worku Jifara, Khan Adil, and Shaohui Liu. Medical image semantic segmentation based on deep learning. *Neural Computing and Applications*, 29(5):1257–1265, 2018.

[21] Miyoun Jung and Myungjoo Kang. Efficient nonsmooth nonconvex optimization for image restoration and segmentation. *Journal of Scientific Computing*, 62(2):336–370, 2015.

[22] Konstantinos Kamnitsas, Christian Ledig, Virginia FJ Newcombe, Joanna P Simpson, Andrew D Kane, David K Menon, Daniel Rueckert, and Ben Glocker. Efficient multi-scale 3d cnn with fully connected crf for accurate brain lesion segmentation. *Medical image analysis*, 36:61–78, 2017.

[23] Baris Kayalibay, Grady Jensen, and Patrick van der Smagt. Cnn-based segmentation of medical imaging data. *arXiv preprint arXiv:1701.03056*, 2017.

[24] Adnan Mujahid Khan, Nasir Rajpoot, Darren Treanor, and Derek Magee. A nonlinear mapping approach to stain normalization in digital histopathology images using image-specific color deconvolution. *IEEE Transactions on Biomedical Engineering*, 61(6):1729–1738, 2014.

[25] Khalil Khan, Massimo Mauro, Pierangelo Migliorati, and Riccardo Leonardi. Gender and expression analysis based on semantic face segmentation. In *International conference on image analysis and processing*, pages 37–47. Springer, 2017.

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[27] Marc D Kohli, Ronald M Summers, and J Raymond Geis. Medical image data and datasets in the era of machine learning—whitepaper from the 2016 c-mimi meeting dataset session. *Journal of digital imaging*, 30(4):392–399, 2017.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[29] Neeraj Kumar, Ruchika Verma, Sanuj Sharma, Surabhi Bhargava, Abhishek Vahadane, and Amit Sethi. A dataset and a technique for generalized nuclear segmentation for computational pathology. *IEEE transactions on medical imaging*, 36(7):1550–1560, 2017.

[30] 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.

[31] Weijia Li, Conghui He, Jiarui Fang, Juepeng Zheng, Haohuan Fu, and Le Yu. Semantic segmentation-based building footprint extraction using very high-resolution satellite images and multi-source gis data. *Remote Sensing*, 11(4):403, 2019.

[32] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[33] Chunxiao Liu, Michael Kwok-Po Ng, and Tieyong Zeng. Weighted variational model for selective image segmentation with application to medical images. *Pattern Recognition*, 76:367–379, 2018.

[34] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[35] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[36] Jun Ma. Cutting-edge 3d medical image segmentation methods in 2020: Are happy families all alike? *arXiv preprint arXiv:2101.00232*, 2021.

[37] The MathWorks Inc., Natick, Massachusetts. *MATLAB version 9.5 (R2018b)*, 2018.

[38] The Mathworks, Inc., Natick, Massachusetts. *MATLAB version 9.10 (R2021a)*, 2021.

[39] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[40] Jawad Nagi, Frederick Ducatelle, Gianni A. Di Caro, Dan Cireşan, Ueli Meier, Alessandro Giusti, Farrukh Nagi, Jürgen Schmidhuber, and Luca Maria Gambardella. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, pages 342–347, 2011.

[41] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[42] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1520–1528, 2015.

[43] Zhuokun Pan, Jiashu Xu, Yubin Guo, Yueming Hu, and Guangxing Wang. Deep learning segmentation and classification for urban village using a worldview satellite image based on u-net. *Remote Sensing*, 12(10):1574, 2020.

[44] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[45] 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 (CVPR)*, June 2016.

[46] Erik Reinhard, Michael Adhikhmin, Bruce Gooch, and Peter Shirley. Color transfer between images. *IEEE Computer graphics and applications*, 21(5):34–41, 2001.

[47] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[48] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[49] Sagar Sharma and Simone Sharma. Activation functions in neural networks. *Towards Data Science*, 6(12):310–316, 2017.

[50] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

[51] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[52] 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.

[53] Carole H Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin, and M Jorge Cardoso. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. In *Deep learning in medical image analysis and multimodal learning for clinical decision support*, pages 240–248. Springer, 2017.

[54] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[55] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[56] Nima Tajbakhsh, Jae Y. Shin, Suryakanth R. Gurudu, R. Todd Hurst, Christopher B. Kendall, Michael B. Gotway, and Jianming Liang. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE Transactions on Medical Imaging*, 35(5):1299–1312, 2016.

[57] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[58] Dayong Wang, Aditya Khosla, Rishab Gargeya, Humayun Irshad, and Andrew H Beck. Deep learning for identifying metastatic breast cancer. *arXiv preprint arXiv:1606.05718*, 2016.

[59] 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*, 6(3):283–292, 2018.

[60] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[61] Zhou and Chellappa. Computation of optical flow using a neural network. In *IEEE 1988 International Conference on Neural Networks*, pages 71–78 vol.2, 1988.

BIOGRAPHICAL SKETCH

Rajendra K. C. Khatri was born in Makawanpur, Nepal in 1985. He is a PhD candidate in Mathematics at The University of Texas at Dallas. He is working on image segmentation problems in medical imaging under the supervision of Dr. Yan Cao. He received his Bachelor's and Master's degree in Mathematics from Tribhuvan University, Nepal, in 2007 and 2009. He graduated with another MS in Applied Mathematics from The University of Texas at Dallas in 2020. He has worked as a Teaching Assistant in the Department of Mathematical Sciences, UT Dallas, since 2015.

CURRICULUM VITAE

RAJENDRA K C KHATRI

EDUCATION

- **PhD in Mathematics – The University of Texas at Dallas** 2021
  Department of Mathematical Sciences
- **MS in Applied Mathematics – The University of Texas at Dallas** 2020
  Department of Mathematical Sciences
- **MA in Mathematics – Tribhuvan University, Nepal** 2009
  Central Department of Mathematics

TEACHING EXPERIENCE

- **Graduate Teaching Assistant** 2015 - 2021
  Department of Mathematical Sciences, UT Dallas
- **Lecturer** 2012 - 2015
  Prime College, Kathmandu Nepal

PUBLICATIONS

- **Automatic extraction of cell nuclei using dilated convolutional network**.
  Inverse Problems & Imaging, 2021, 15(1): 27-40, doi:10.3934/ipi.2020049
- **Predicting stock market index using LSTM**.
  Submitted , 2021.

RESEARCH INTEREST

Medical Image Analysis, Computer Vision, Artificial Intelligence, Data science

MEMBERSHIP

- Member of Society of Computational and Applied Mathematics (SIAM)
- Member of Nepal Mathematical Society (NMS)
- Member of Nepalese Mathematicians in America (ANMA)