ADVANCED DEEP LEARNING METHODS FOR IMAGE PIXEL-WISE PREDICTION

By

YONGJUN CHEN

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WASHINGTON STATE UNIVERSITY
School of Electrical Engineering and Computer Science

DECEMBER 2018

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of YONGJUN CHEN find it satisfactory and recommend that it be accepted.

Shuiwang Ji, Ph.D., Chair

Haipeng Cai, Ph.D.

Hassan Ghasemzadeh, Ph.D.

ACKNOWLEDGMENTS

Firstly, I would like to express my sincere gratitude to my advisor Prof. Shuiwang Ji for his incredible mentor of my M.S. study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my M.S. study. Besides my advisor, I would like to thank all my thesis committee: Prof. Cai and Prof. Ghasemzadeh for their insightful comments and encouragement.

I had so much memorable time with my collaborators, friends and my family. I thank all my collaborators for the stimulating discussions, for the sleepless nights we were working together before deadlines, and for all the fun we have had in the last two years. I thank all the friends I met. They made my research life more enjoyable, happier and easier. Finally, I want to thank my family: my parents, my brothers and sisters for supporting me spiritually throughout writing this thesis and my life in general.

Overall, I want to thank everyone who has played a big or small role in my life. I am so gratefull for all yours help and support! And I am also looking forward the

next chapter of my life!

ADVANCED DEEP LEARNING METHODS FOR IMAGE PIXEL-WISE PREDICTION

Abstract

by Yongjun Chen, M.S.
Washington State University
December 2018

Chair: Shuiwang Ji

Deep learning methods have been highly successful in performing image pixel-wise prediction tasks. One of the most popular methods employs an encoder-decoder network with deconvolutional layer (DCL) for up-sampling feature maps. However, a key limitation of DCL is that they suffer from the checkerboard artifact problem, which decreases prediction accuracy. This artifact is caused by the independence among adjacent pixels on the output feature maps. Previous work has solved the checkerboard artifact issue of DCL only in 2D space. Because the number of intermediate feature maps needed to generate DCL grows exponentially with dimensionality, solving the checkerboard artifact issue is much more challenging with a higher number of dimensions. Furthermore, current deep learning methods make image pixel-wise predictions

by applying a model on regular patches centered on each pixel. Consequently, current methods are further limited for image analysis and prediction tasks because patches are determined from network architecture instead of data learning.

To address these limitations, I proposed the voxel deconvolutional layer (VoxelDCL) to solve the checkerboard artifact problem of DCL in 3D space. Additionally, I developed an efficient approach to implement VoxelDCL. To demonstrate the effectiveness of VoxelDCL, four variations of voxel deconvolutional networks (VoxelDCNs) were built based on U-Net architecture, but with VoxelDCL instead of DCL. Then, the proposed networks were applied to volumetric brain image labeling tasks, using the ADNI and LONI LPBA40 datasets. The experimental results showed that one variation, iVoxelDCNa, achieved the best performance across all experiments, reaching 83.34% in terms of dice ratio on the ADNI dataset and 79.12% on the LONI LPBA40 dataset. These values represent increases of 1.39% and 2.21%, respectively, compared with the classic U-Net method used as the baseline. Additionally, all VoxelDCN variations outperformed the baseline method on the two datasets, demonstrating their enhanced effectiveness. To relax the constraint of using patches with fixed shapes and sizes, I proposed dense transformer networks (DTNs), which can dynamically learn shapes and sizes of patches from input data. Experimental studies that applied these DTNs on natural and biological image pixel-wise prediction tasks also demonstrated superior performance compared to the baseline method.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Dedication

To Yongshuai, Mom, and Dad.

Yongshuai: for being the best brother and friend ever, and the most inspiring person

Mama: for your unbelievable love and belief in me

Baba: for always support to me

CHAPTER 1. INTRODUCTION

Deep learning methods have achieved promising performance in image pixel/voxel-wise prediction tasks. Current state-of-the-art networks employ encoder-decoder architectures. However, the deconvolutional layers (DCL), which are commonly employed in the decoder part, suffer from the checkerboard artifact issue existing in 3D space. Furthermore, the shapes and sizes of patches extracted for prediction are determined by the network architecture, which cannot be adjusted automatically based on input data. Both of these issues result in reduced prediction accuracy. Thus, I proposed voxel deconvolutional networks (VoxelDCNs) and dense transformer networks (DTNs) to resolve these limitations, and then designed experiments to evaluate their capacity to perform image pixel/voxel-wise prediction tasks. The experimental studies demonstrated the enhanced effectiveness of the proposed methods on natural, volumetric brain, and brain electron microscopy images, compared to the baseline method.

1.1 Problem Statement and Motivation

In recent years, deep learning methods have played an important role in various computer vision tasks such as image recognition [24, 23], object detection [37, 14], action recognition [20] and pixel-wise prediction [18, 39, 6]. Some key network layers

like convolutional layers [25], pooling/unpooling layers [49, 33], deconvolutional layers [41] and some well-known models like generative models [15] and encoder-decoder architectures [35, 33] are frequently used for these tasks. In pixel-wise prediction tasks, U-Net [35] with an encoder-decoder architecture is commonly used. The encoder path contains convolutional and down-sampling operations to extract high-level feature maps from raw images, while the decoder path recovers feature maps to the original spatial size using up-sampling operations. Encoder architectures are very similar to classic convolutional neural networks, which have been studied extensively [35, 27]. On the other hand, less attention has been paid to the decoder path. The three primary ways to up-sample feature maps are deconvolution, unpooling and resampling with interpolation [3]. In practice, deconvolutional layers are the most commonly used. However, a key problem with the deconvolutional layer is the checkerboard artifact issue [34, 1] caused by the independence among adjacent pixels on the output feature maps. In addition, current encoder-decoder architectures only act on regular patches centered on each pixel to make predictions, which means that patches used for pixel-wise prediction are completely determined by pre-defined network architectures. Both of these problems reduce prediction accuracy.

To solve the checkerboard artifact problem, some studies focused on improving the post-processing [8], which prevented the whole process from being fully trainable. Other work focused on adding smooth constraints [22], which resulted in a

more complex process. In contrast, the pixel deconvolutional networks [12] solved the checkerboard artifact problem by generating the intermediate feature maps sequentially, thereby building direct relationships among adjacent pixels on output feature maps. This is an effective way while keeping the learning process trainable. However, pixel deconvolutional networks only solved the artifact issue in 2D DCL. The checkerboard artifact issue also exists in 3D DCL, and is more challenging to solve because the number of intermediate maps needed to generate one DCL grows exponentially with dimensionality. For example, a total of 2^d intermediate feature maps are needed to generate a deconvolutional layer given a dimensional input.

To further improve performances of pixel/voxel-wise prediction tasks, it is highly desirable to resolve these limitations.

1.2 Summary of Results and Contributions

In this thesis, I proposed the voxel deconvolutional layer (VoxelDCL) to address the checkerboard artifact of 3D deconvolutional layers. Four variations of VoxelDCLs, named iVoxelDCLc, iVoxelDCLa, VoxelDCLc, and VoxelDCLa, were built based on different approaches for generating intermediate feature maps. iVoxelDCLc and iVoxelDCLa use the input along with the generated intermediate feature maps to generate new intermediate feature maps by concatenation and addition, respectively. The Vox-

elDCLc and VoxelDCLa use only the generated intermediate feature maps to generate new intermediate feature maps. I also developed an efficient implementation method to improve the computational efficiency by reducing unnecessary dependencies among the intermediate feature maps. To demonstrate the effectiveness of the proposed VoxelDCL, I first built the voxel deconvolutional networks (VoxelDCN) based on the U-Net with the four variations of VoxelDCL. I then applied the networks to volumetric brain image voxel-wise prediction task [52, 11, 47, 45, 43] using the Alzheimer's disease neuroimaging initiative (ADNI) [31] and the LONI LPBA40 [38] datasets. Results showed that the proposed methods significantly outperformed U-Net with regular DCL in terms of dice ratio. Specifically, iVoxelDCLa achieved the best performance of all four networks on both the ADNI and LONI LPBA40 datasets with an 83.34% and 79.12% dice ratio, respectively.

To allow the size and shape of every patch to be adaptive and data-dependent, I proposed the use of DTNs. Applying DTNs on natural and biological image pixel-wise prediction tasks in 2D space demonstrated the superior performance of these networks compared to the baseline method.

1.3 Thesis Organization

The remainder of this thesis is organized as follows:

- Chapter 2: Related Work. This chapter provides a brief overview of previous research relative to up-sampling operations, voxel-wise prediction, and spatial transformer networks.

- Chapter 3: Voxel Deconvolutional Networks for 3D Brain Image Voxel-Wise Prediction. In this chapter, I describe the primary limitation with current deep learning methods. That is, in encoder-decoder architectures, DCL are heavily employed for up-sampling operations, but introduce the checkerboard artifact issue in final outputs. I then describe my proposed resolutionvoxel deconvolutional layers (VoxelDCL)and the approach developed to efficiently implement VoxelDCL in 3D space. I then explain how I built four variations of voxel deconvolutional networks (VoxelDCNs) based on the U-Net architecture with VoxelDCL. Finally, results are presented from the experimental studies that demonstrated the effectiveness of the proposed methods by addressing brain image voxel-wise prediction tasks using the ADNI and LONI LPBA40 datasets.

- Chapter 4: Dense Transformer Networks for 2D Image Pixel-Wise Prediction. In this chapter, I describe the current problem associated with existing encoder-decoder architectures, that is, a regular patch centered on each pixel is employed for prediction. These methods are limited because these patches are determined by network architecture instead of learned from input data. I then present a

proposed solution for the problem of patch-based learning for pixel-wise prediction in 2D spaceDTNs, which can learn the shape and size of patches from data. Because dense transformer modules are differentiable, the entire network can be trained and, in turn, prediction accuracy can be increased. Finally, I show how the proposed networks can achieve superior performance on natural and biological image pixel-wise prediction tasks compared to the baseline method.

- Chapter 5: Conclusion. This chapter summarizes the main work of this thesis and highlights opportunities for future work.

CHAPTER 2. RELATED WORK

2.1 Operations for Up-Sampling

To recover the spatial size of the extracted feature maps to the original size as the input image, up-sampling layers are essential in the decoder path of a pixel-wise prediction network. Unpooling, resampling with interpolation, and deconvolution are the most commonly used operations for up-sampling. Unpooling layers [49, 33] put each activation back to its original location based on the recorded location of maximum activation selected during the corresponding pooling operations. Up-sampling layers using resampling with interpolation [3] scales an feature map to the desired size and calculates the value of each pixel using an interpolation method such as bilinear interpolation. In the above two methods, there is no learning parameters required in the operation. The third method, known as the deconvolutional layer [41, 34, 1], is the most popular method for up-sampling tasks. It up-samples feature maps by using operations that can be formulated as convolutional operations with learned kernels. More details are explained in Section 3.1.1. However, deconvolutional layers suffer from the checkerboard artifact issue. This is because there is no direct relationship among adjacent pixels on the output feature map. To solve this problem, pixel deconvolutional networks [12] built direct relationship among adjacent pixels

on the output of a deconvolutional layer by generating intermediate feature maps sequentially. This process is different in traditional deconvolutional layers where the intermediate feature maps are generated independently. Details of the concepts of intermediate feature maps and the sequential process are given in Section 3.1. The dependencies among the intermediate feature maps add direct relationships among adjacent pixels on the output feature map, thereby alleviating the checkerboard artifact in 2D deconvolutional layers.

2.2 Voxel-Wise Prediction

In computer vision area, voxel-wise prediction tasks [50, 53] have been a popular research field. Some existing methods [44, 36, 13, 46, 29, 51] use multi-atlas based voxel-wise prediction models to predict the labels of new images. These methods first transfer the pixel-wise prediction labels from pre-labeled atlases to a new image and then apply the label fusion method to combine the transferred labels as predicted results. There are also several learning-based voxel-wise prediction methods using random forests, support vector machine, and neural networks [30] to train a prediction model. Among these methods, neural networks have an unique advantage in that it does not require hand-crafted feature extraction in advance. It enables end-to-end learning, which improves efficiency of the training process. In this work, we introduce

a novel deep neural network architecture and apply the variations of this network on the 3D brain image voxel-wise prediction task.

2.3 Spatial Transformer Networks Based on Thin-Plate Spline

Spatial transformer networks [19] are deep models containing spatial transformer layers. These layers explicitly compute a spatial transformation of the input feature maps. They can be inserted into convolutional neural networks to perform explicit spatial transformations. The spatial transformer layers consist of three components; namely, the localization network, grid generator and sampler.

The localization network takes a set of feature maps as input and generates parameters to control the transformation. If there are multiple feature maps, the same transformation is applied to all of them. The grid generator constructs transformation mapping between input and output grids based on parameters computed from the localization network. The sampler computes output feature maps based on input feature maps and the output of grid generator. The spatial transformer layers are generic and different types of transformations, e.g., affine transformation, projective transformation, and thin-plate spline (TPS), can be used. Our proposed work is based on the TPS transformation, and it is not described in detail in the original paper [19]. Thus we provide more details below.

2.3.1 Localization Network

When there are multiple feature maps, the same transformation is applied to all of them. Thus, we assume there is only one input feature map below. The TPS transformation is determined by $2K$ fiducial points among which K points lie on the input feature map and the other K points lie on the output feature map. On the output feature map, the K fiducial points, whose coordinates are denoted as $\tilde{F} = [\tilde{f}_1, \tilde{f}_2, \cdots, \tilde{f}_K] \in \mathbb{R}^{2\times K}$, are evenly distributed on a fixed regular grid, where $\tilde{f}_i = [\tilde{x}_i, \tilde{y}_i]^T$ denotes the coordinates of the ith point. The localization network is used to learn the K fiducial points $F = [f_1, f_2, \cdots, f_K] \in \mathbb{R}^{2\times K}$ on the input feature map. Specifically, the localization network, denoted as $f_{loc}(\cdot)$, takes the input feature maps $U \in \mathbb{R}^{H\times W\times C}$ as input, where H, W and C are the height, width and number of channels of input feature maps, and generates the normalized coordinates F as the output as $F = f_{loc}(U)$.

A cascade of convolutional, pooling and fully-connected layers is used to implement $f_{loc}(\cdot)$. The output of the final fully-connected layer is the coordinates F on the input feature map. Therefore, the number of output units of the localization network is $2K$. In order to ensure that the outputs are normalized between -1 and 1, the activation function $\tanh(\cdot)$ is used in the fully-connected layer. Since the localization network is differentiable, the K fiducial points can be learned from data using error

back-propagation.

2.3.2 Grid Generator

For each unit lying on a regular grid on the output feature map, the grid generator computes the coordinate of the corresponding unit on the input feature map. This correspondence is determined by the coordinates of the fiducial points F and $\tilde{F}$. Given the evenly distributed K points $\tilde{F} = [\tilde{f}_1, \tilde{f}_2, \cdots, \tilde{f}_K]$ on the output feature map and the K fiducial points $F = [f_1, f_2, \cdots, f_K]$ generated by the localization network, the transformation matrix T in TPS can be expressed as follows:

$$T = \left(\Delta_{\tilde{F}}^{-1} \times \begin{bmatrix} F^T \\ \mathbf{0}^{3\times 2} \end{bmatrix} \right)^T \in \mathbb{R}^{2\times(K+3)}, \tag{2.1}$$

where $\Delta_{\tilde{F}} \in \mathbb{R}^{(K+3)\times(K+3)}$ is a matrix determined only by $\tilde{F}$ as

$$\Delta_{\tilde{F}} = \begin{bmatrix} \mathbf{1}^{K\times 1} & \tilde{F}^T & R \\ \mathbf{0}^{1\times 1} & \mathbf{0}^{1\times 2} & \mathbf{1}^{1\times K} \\ \mathbf{0}^{2\times 1} & \mathbf{0}^{2\times 2} & \tilde{F} \end{bmatrix} \in \mathbb{R}^{(K+3)\times(K+3)}, \tag{2.2}$$

where $R \in \mathbb{R}^{K\times K}$, and its elements are defined as $r_{i,j} = d_{i,j}^2 \ln d_{i,j}^2$, and $d_{i,j}$ denotes the Euclidean distance between $\tilde{f}_i$ and $\tilde{f}_j$.

Through the mapping, each unit $(\tilde{x}_i, \tilde{y}_i)$ on the output feature map corresponds to unit (x_i, y_i) on the input feature map. To achieve this mapping, we represent the units

on the regular output grid by $\{\tilde{p}_i\}_{i=1}^{\tilde{H}\times\tilde{W}}$, where $\tilde{p}_i = [\tilde{x}_i, \tilde{y}_i]^T$ is the (x, y)-coordinates of the ith unit on output grid, and $\tilde{H}$ and $\tilde{W}$ are the height and width of output feature maps. Note that the fiducial points $\{\tilde{f}_i\}_{i=1}^{K}$ are a subset of the points $\{\tilde{p}_i\}_{i=1}^{\tilde{H}\times\tilde{W}}$, which are the set of all points on the regular output grid.

To apply the transformation, each point $\tilde{p}_i$ is first extended from $\mathbb{R}^2$ space to $\mathbb{R}^{K+3}$ space as $\tilde{q}_i = [1, \tilde{x}_i, \tilde{y}_i, s_{i,1}, s_{i,2}, \cdots, s_{i,K}]^T \in \mathbb{R}^{K+3}$, where $s_{i,j} = e_{i,j}^2 \ln e_{i,j}^2$, and $e_{i,j}$ is the Euclidean distance between $\tilde{p}_i$ and $\tilde{f}_j$. Then the transformation can be expressed as

$$p_i = T\tilde{q}_i, \tag{2.3}$$

where T is defined in Eq. (2.1). By this transformation, each coordinate $(\tilde{x}_i, \tilde{y}_i)$ on the output feature map corresponds to a coordinate (x_i, y_i) on the input feature map. Note that the transformation T is defined so that the points $\tilde{F}$ map to points F.

2.3.3 Sampler

The sampler generates output feature maps based on input feature maps and the outputs of grid generator. Each unit $\tilde{p}_i$ on the output feature map corresponds to a unit p_i on the input feature map as computed by Eq. (2.3). However, the coordinates $p_i = (x_i, y_i)^T$ computed by Eq. (2.3) may not lie exactly on the input regular grid. In these cases, the output values need to be interpolated from input values lying

on regular grid. For example, a bilinear sampling method can be used to achieve this. Specifically, given an input feature map $U \in \mathbb{R}^{H \times W}$, the output feature map $V \in \mathbb{R}^{\tilde{H} \times \tilde{W}}$ can be obtained as

$$V_i = \sum_{n=1}^{H} \sum_{m=1}^{W} U_{nm} \max(0, 1 - |x_i - m|) \max(0, 1 - |y_i - n|) \tag{2.4}$$

for $i = 1, 2, \cdots, \tilde{H} \times \tilde{W}$, where V_i is the value of pixel i, U_{nm} is the value at (n, m) on the input feature map, $p_i = (x_i, y_i)^T$, and p_i is computed from Eq. (2.3). By using the transformations, the spatial transformer networks have been shown to be invariant to some transformations on the inputs. Other recent studies have also attempted to make CNNs to be invariant to various transformations [21, 16, 5, 7].

CHAPTER 3. VOXEL DECONVOLUTIONAL NETWORKS FOR 3D IMAGE VOXEL-WISE PREDICTION

3.1 Voxel Deconvolutional Networks

In this section, we introduce the concept of deconvolutional layers and other related operations, including transposed convolutional layers [41] and sub-pixel convolutional layers [34, 1, 41]. We show the equivalence among these concepts in the 1D case. After that, we describe 2D and 3D deconvolutional layers in the form of sub-pixel deconvolutional layer and show the checkerboard artifact that they suffer [34]. We discuss existing approaches to solve the checkerboard artifact and propose the voxel deconvolutional layers and networks, which can overcome the checkerboard problem of 3D deconvolutional layers. We show that our methods achieve better voxel-wise prediction performance on the Alzheimer's disease Neuroimaging Initiative (ADNI) [31] and the LONI LPBA40 [38] datasets as compared with other baseline methods.

3.1.1 Deconvolutional Layer

Deconvolutional layers can be viewed as a form of convolutional layers. The relationship between convolutional and deconvolutional layers can be best understood when both of them are considered as fully connected layers. Specifically, convolutional layers can be considered as fully connected layers with sparse connection matrices. Let us consider the following example in 1D. Given an input vector $x \in \mathbb{R}^8$, we pad the input by adding two zeros to both ends of x, yielding the padded input $\tilde{x} \in \mathbb{R}^{12}$. We then apply a 1D convolution with a stride of 2 and a kernel of size 4 to produce an output vector $y \in \mathbb{R}^5$. This convolution operation can be equivalently viewed as a fully connected operation as

$$y = C\tilde{x}, \tag{3.1}$$

with a 5×12 sparse connection matrix C, defined as follows:

$$C = \begin{bmatrix} w_1 & w_2 & w_3 & w_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 & w_4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 & w_4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 & w_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_1 & w_2 & w_3 & w_4 \end{bmatrix}. \tag{3.2}$$

In this example, y can be cropped by one pixel from the right end to make the down-sampling factor to be 2.

The above example reduces x to a lower-dimension by a convolution of stride 2. A similar operation can be used to convert a lower-dimensional input to a high-dimensional output, resulting in the deconvolutional operation. Specifically, assume y is the input and $\tilde{x}$ is the output, then their relationship can be expressed as

$$\tilde{x} = C^T y, \tag{3.3}$$

where C is defined in Eq. (3.2). It can be seen from Eq. (3.3) that, given a lower-dimensional input, a higher-dimensional output can be obtained via a fully connected layer with a sparse connection matrix. This results in the so-called deconvolutional layer. Since deconvolution layers use the transposed weight matrix of its corresponding convolutional operation C, it is also known as transposed convolutions.

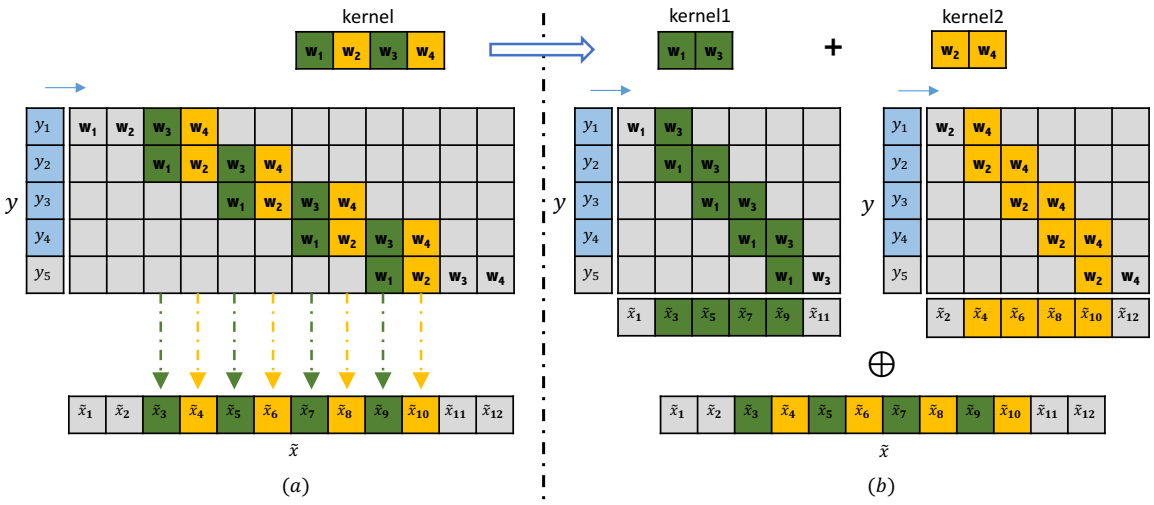


Figure 3.1: Illustration of a 1D deconvolutional layer. The green and yellow colors represent the odd and even columns of the kernel, respectively. $\oplus$ denotes the periodical shuffling and combination operation. y is the input and $\tilde{x}$ is the output from the deconvolutional layer. By multiplying each column of the weight matrix C with y, we can obtain its corresponding output $\tilde{x}_i$ (see (a)). Since all the odd columns of $\tilde{x}$ are computed only by $w1$ and $w3$ and all the even columns of $\tilde{x}$ are computed by $w2$ and $w4$, the operation can be decomposed into two convolutions with two independent kernels of sizes 1×2, resulting in kernel1 and kernel2. The intermediate results are then shuffled and combined to obtain the final output $\tilde{x}$. $\tilde{x}$ can be cropped by two pixels from both ends to make the up-sampling factor to be 2.

Since C is a sparse matrix, another way to understand the deconvolutional layer is to interpret it as a standard convolution. Specifically, it can be considered as

performing multiple independent convolutions on the input followed by a periodical shuffling operation [40]. An example is given in Figure 3.1 to illustrates this idea.

3.1.2 3D Deconvolutional Layer

In the previous section, we describe how to understand deconvolutional layers as a form of convolutional layers in 1D space. The same strategy can be used in 2D and 3D spaces as well. In 2D case, the first step is to generate 4 intermediate feature maps from padded input. This step simply requires 2D convolutional operations. Next, these feature maps are periodically shuffled and combined together to form the output. The same steps can be applied to perform deconvolutional operations in 3D space, which will be discussed next.

Assume $X \in \mathbb{R}^{l \times m \times n}$ is the input and $Y \in \mathbb{R}^{2l \times 2m \times 2n}$ is the output of a deconvolutional layer. Here the up-sampling factor is 2. We first perform eight 3D convolutional operations on the input. Each of such operation generates one intermediate feature map Y_i with size $l \times m \times n$ given by

$$Y_i = X \otimes k_i, \quad i = 1, 2, \cdots, 8, \tag{3.4}$$

where $\otimes$ denotes the convolutional operation, Y_i denotes the i^{th} feature map and k_i denotes the corresponding kernel. We add padding to the input in order to make sure that spatial size remains the same. Finally, these 8 feature maps are periodically

shuffled and combined together as

$$Y = Y_1 \oplus Y_2 \oplus Y_3 \oplus Y_4 \oplus Y_5 \oplus Y_6 \oplus Y_7 \oplus Y_8, \tag{3.5}$$

where $\oplus$ denotes the periodical shuffling and combination operation, as illustrated in Figure 3.2.

3.1.3 Voxel Deconvolutional Layer

When we consider a deconvolutional layer as the shuffled combination of several intermediate feature maps, discontinuities among adjacent pixels can be observed frequently. This is not surprising since the adjacent pixels are coming from independent feature maps. This is the checkerboard artifact problem that all deconvolutional layers suffer, no matter calculated in 2D or 3D space.

Compared to deconvolutional layers in 2D space, the number of feature maps needed in 3D space grows exponentially. In general, assume the up-sampling factor is 2 and the dimensionality is d, the number of intermediate feature maps need to generate one deconvolutional layer is 2^d. This indicates that deconvolutional layers in higher dimensional space are much more complex and computationally expensive than those in lower dimensional space. These are factors to consider when we introduce voxel deconvolutional layers in Section 3.1.3.

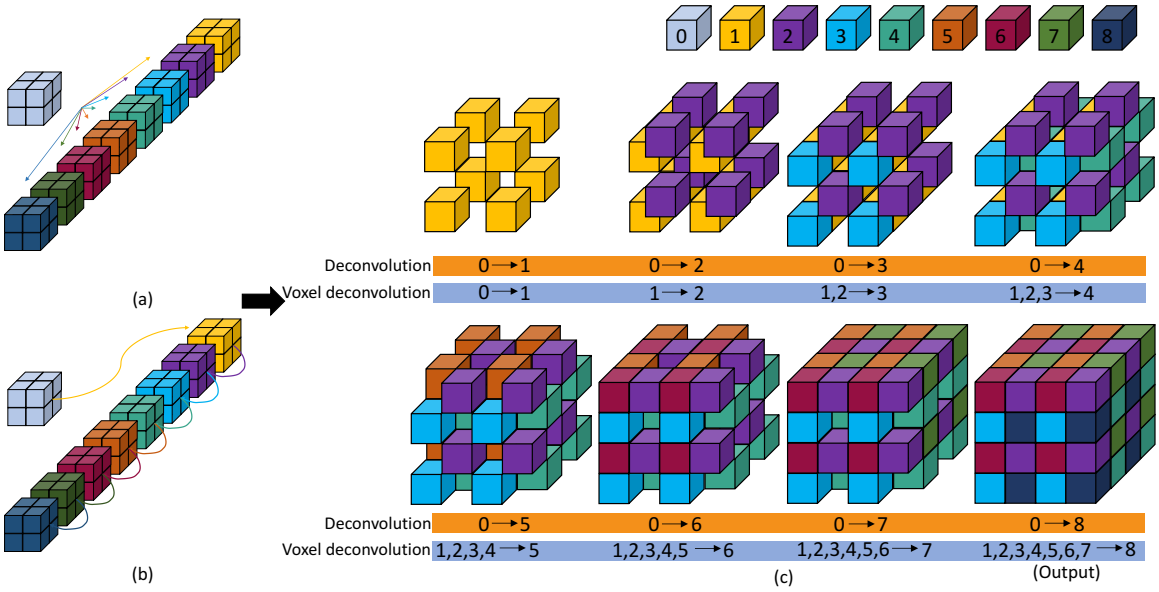


Figure 3.2: Illustration of the 3D deconvolutional layer and voxel deconvolutional layer. For convenience, we use number 0 to denote the input, and number 1 - 8 to denote the eight intermediate feature maps which are generated from convolutional operations. The input size is $2\times2\times2$ and the desired out size is $4\times4\times4$. The size of each intermediate feature map is also $2\times2\times2$. (a) shows how the intermediate feature maps are generated independently from eight different kernels in a regular 3D deconvolutional layer. This independent generation process is also indicated in the lower orange bar in (c). (b) shows how the intermediate feature maps are generated sequentially in a voxel deconvolutional layer. The i^{th} feature map are built on the $1^{st}, \cdots, (i-1)^{th}$ feature maps. In iVoxelDCLa and iVoxelDCLc layers, input data is also included in this process. This sequential generation process is also indicated in the lower blue bar in (c). (c) displays the periodically shuffling and combination process of eight feature maps to form the final output of a deconvolutional layer.

Gao *et al.* [12] solved the checkerboard artifact of deconvolutional layers in 2D space by adding relations to the intermediate feature maps. Inspired by [12], we propose the voxel deconvolutional layer to solve the same issue in 3D space. As we mentioned in Section 3.1.2, the computational cost and complexity of deconvolutional operations in 3D space are much higher than those in 2D space, thereby making this task challenging.

Back to the example in Section 3.1.2, we use $X \in \mathbb{R}^{l \times m \times n}$ to denote the input and $Y \in \mathbb{R}^{2l \times 2m \times 2n}$ to denote the output. The eight intermediate feature maps are $Y_1, Y_2, \cdots, Y_8$, each of size $l \times m \times n$.

In voxel deconvolutional layers, $Y_1, Y_2, \cdots, Y_8$ are formed sequentially to build direct relations in-between. Suppose $Y_1, Y_2, \cdots, Y_{i-1}$ are already generated, Y_i will be built on X together with $Y_1, Y_2, \cdots, Y_{i-1}$ instead of simple X as in traditional deconvolutional layers. The generation of Y_1 remains the same as it is the first intermediate feature map. There are mainly two ways to combine $X, Y_1, Y_2, \cdots, Y_{i-1}$ together; namely addition or concatenation.

We first introduce the naming conventions of the proposed voxel deconvolutional layers and corresponding networks in below:

- **Suffix c:** We use concatenation when combining multiple inputs to generate a new intermediate feature map.

- **Suffix a:** We use addition when combining multiple inputs to generate a new

intermediate feature map.

- **Prefix iVoxel:** Both the original input and existing intermediate feature maps are included to generate a new feature map.

- **Prefix Voxel:** Only existing intermediate feature maps are included to generate a new feature map.

We first introduce **iVoxelDCLc**. In iVoxelDCLc, the input X and intermediate feature maps $Y_1, Y_2, \cdots, Y_{i-1}$ are concatenated together to form the input to generate Y_i as

$$Y_1 = X \otimes k_1; Y_i = [X, Y_1, , \cdots, Y_{i-1}] \otimes k_i, \text{ for } i = 2, \cdots, 8, \tag{3.6}$$

where $\otimes$ denotes a convolutional operation, $[\cdot, \cdot]$ represents the concatenation and k_i denotes the corresponding convolutional kernel. By using concatenation, $X, Y_1, \cdots, Y_{i-1}$ are stacked as channels of a single input to generate Y_i. There will be weights assigned to each of $X, Y_1, \cdots, Y_{i-1}$, indicating their importance to Y_i. However, concatenation will cause memory and computational issues due to the large amount of parameters needed. This problem can be reduced by using addition when combining multiple related feature maps as input, which will be denoted as iVoxelDCLa.

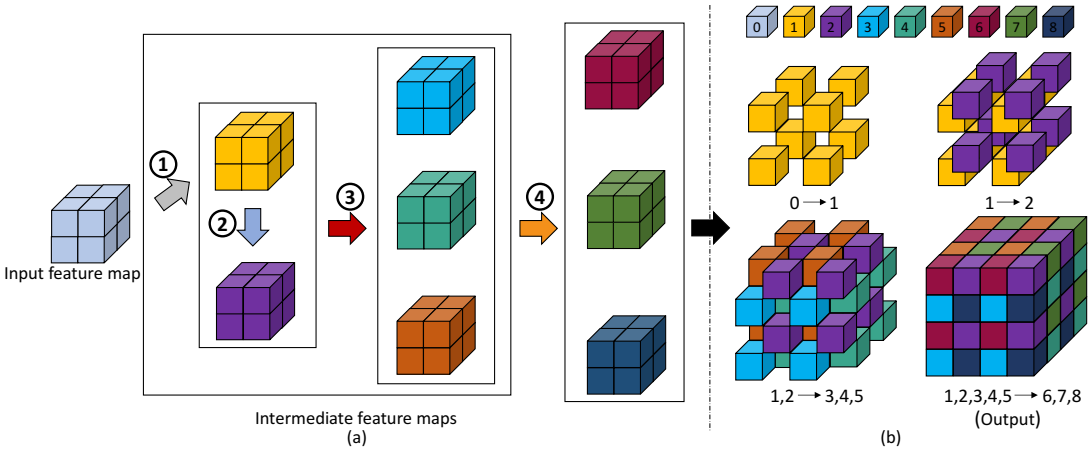


Figure 3.3: Illustration of the proposed efficient implementation of the VoxelDCLa and VoxelDCLc layers. The notation remains the same as introduced in Figure 3.2. In addition, each step in the efficient implementation is denoted as numbers with circles in (a). (a) shows the efficient generation process of intermediate feature maps step by step. Step 1 is to generate the 1^{st} feature map directly from the input. Step 2 is to generate the 2^{nd} feature map from the 1^{st} one. The 3^{rd} to 5^{th} feature maps are generated from the 1^{st} and 2^{nd} feature maps in step 3. Finally, the last three feature maps are generated from all previous five feature maps in step 4. (b) shows how these eight intermediate feature maps are shuffled and combined to form the final $4 \times 4 \times 4$ output. In VoxelDCLa layers, multiple feature maps are added together to form the input for the next feature map while in VoxelDCLc layers they are concatenated.

In **iVoxelDCLa**, the input X and intermediate feature maps $Y_1, Y_2, \cdots, Y_{i-1}$ are

added together to form the input to generate Y_i as

$$Y_1 = X \otimes k_1; Y_i = \left(X + \sum_{j=1}^{i-1} Y_j \right) \otimes k_i, \text{ for } i = 2, \cdots, 8. \tag{3.7}$$

In iVoxelDCLc and iVoxelDCLa, the input X is included in the generation of every intermediate feature map. This might be redundant since the information carried in X will be carried over in Y_1 to Y_8 through this sequential process. In the next two versions of VoxelDCL, we take out the input X when generating $Y_2, \cdots, Y_8$. Only the first intermediate feature map Y_1 is generated using X. Similarly, we first use concatenation to combine multiple inputs, which will be denoted as VoxelDCLc.

In **VoxelDCLc**, the intermediate feature maps $Y_1, Y_2, \cdots, Y_{i-1}$ are concatenated together to form the input to generate Y_i while Y_1 is generated from the input X:

$$Y_1 = X \otimes k_1; Y_i = [Y_1, \cdots, Y_{i-1}] \otimes k_i, \text{ for } i = 2, \cdots, 8. \tag{3.8}$$

To further reduce memory usage and computational complexity, we use addition instead of concatenation the same way as we described in iVoxelDCLa layer. This layer is known as VoxelDCLa.

In **VoxelDCLa**, intermediate feature maps $Y_1, Y_2, \cdots, Y_{i-1}$ are added together to form the input to generate Y_i while Y_1 is generated from the input X:

$$Y_1 = X \otimes k_1; Y_i = \left(\sum_{j=1}^{i-1} Y_j \right) \otimes k_i, \text{ for } i = 2, \cdots, 8. \tag{3.9}$$

Once all the eight intermediate feature maps are generated, we can obtain the

final output using Eq. (3.5). In Figure 3.2, we provide an example to illustrate how the VoxelDCLa and VoxelDCLc work.

3.1.4 Efficient Implementation

We propose an efficient way to implement all of the above voxel deconvolutional layers in order to scale down the computational cost. The goal is to reduce unnecessary dependencies among the intermediate feature maps. The basic principle is that, for voxels that are not adjacent to each other but linked to the same pivot voxel in the output, we generate their corresponding feature maps in parallel. In this way, we remove the dependency of unrelated inputs when generating new feature maps, hence simplifying the process and expediting the computation. In this new design, the intermediate feature maps are generated in 4 steps. Specifically, Y_1, which denotes the 1^{st} intermediate feature map, is generated from the input X. Then Y_2 is generated from Y_1. Since voxels from Y_3, Y_4, Y_5 are not adjacent to each other but are all linked to voxels from Y_1, we generate them in parallel using previously generated Y_1 and Y_2. The same strategy is used for generating Y_6, Y_7, Y_8. They are generated in parallel from all the previously generated $Y_1, \cdots, Y_5$. In iVoxelDCLa and iVoxelDCLc, the input X is also included in each step. By using parallel generation in steps 3 and 4, the computational time for VoxelDCL is reduced. In this way, we build a reason-

able relationship among these intermediate feature maps. Figure 3.3 illustrates how VoxelDCLa and VoxelDCLc work.

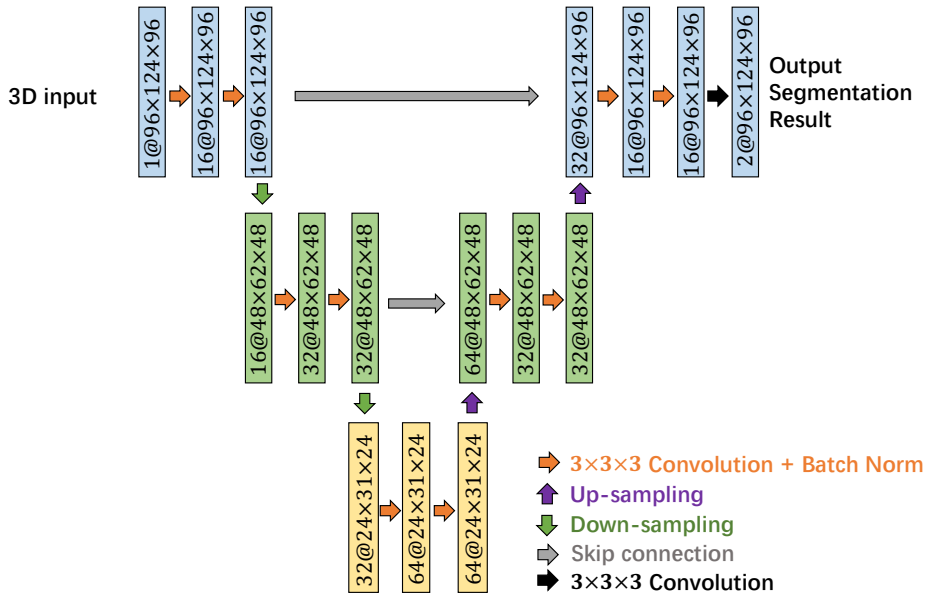


Figure 3.4: Network architecture implemented on the ADNI dataset. The encoder has two max pooling layers with sizes of $2 \times 2 \times 2$. We replace the regular deconvolutional layers in U-Net with variations of the proposed voxel deconvolutional layers in the decoder path.

3.1.5 Overview of Network Architectures

U-Net [35] is a convolutional neural network architecture commonly used for pixel-wise prediction. It has an encoder path which extracts high-level feature maps from raw images and a decoder path which recovers feature maps to the original spatial

size. In the classic U-Net architecture, there is repeated application of three convolutional layers, followed by a max pooling layer for down-sampling in each encoder block. Batch normalization [17] is used after each convolutional layer. The number of channels is doubled after each down-sampling operation. In each decoder block, there is a deconvolutional layer followed by two convolutional layers. The output of the deconvolutional layers are concatenated with corresponding feature maps from encoder blocks before going to the next block. The proposed architecture in this work, which is denoted as voxel deconvolutional network (VoxelDCN), is based on this U-Net framework. But we replace the deconvolutional layers in the decoder path with voxel deconvolutional layers. So there are five architectures implemented for comparison:

- **U-Net:** We use U-Net as our baseline method. It is a classic neural network architecture with encoder and decoder parts. It uses deconvolutional layers for up-sampling in decoder path.

- **iVoxelDCNc:** In this method, we replace all deconvolutional layers in U-Net with iVoxelDCLc layers.

- **iVoxelDCNa:** In this method, we replace all deconvolutional layers in U-Net with iVoxelDCLa layers.

- **VoxelDCNc:** In this method, we replace all deconvolutional layers in U-Net

with VoxelDCLc layers.

- **VoxelDCNa:** In this method, we replace all deconvolutional layers in U-Net with VoxelDCLa layers.

Figure 3.4 shows the architecture we implemented on the ADNI dataset. On the LONI LPBA40 dataset, we use a similar architecture with more blocks in both encoder and decoder paths to avoid under-fitting problem.

3.2 Experimental Studies

In this section, we demonstrate the performance of our proposed methods on two public datasets that have been widely used for brain images voxel-wise prediction, namely the Alzheimer's disease neuroimaging initiative (ADNI) dataset [31] and the LONI LPBA40 dataset [38]. We select these two datasets since they represent different types of brain image labeling. The ADNI dataset has rich brain MR images for labeling hippocampal regions. It can be treated as a pixel-wise prediction task with binary classes. On the contrary, the LONI LPBA40 dataset has various regions of interest, mostly inside the cortex of the brain. It can be treated as a multi-class pixel-wise prediction task. Examples from these datasets are provided in Figure 3.5. Our code is publicly available[1].

[1]https://github.com/divelab/VoxelDCN

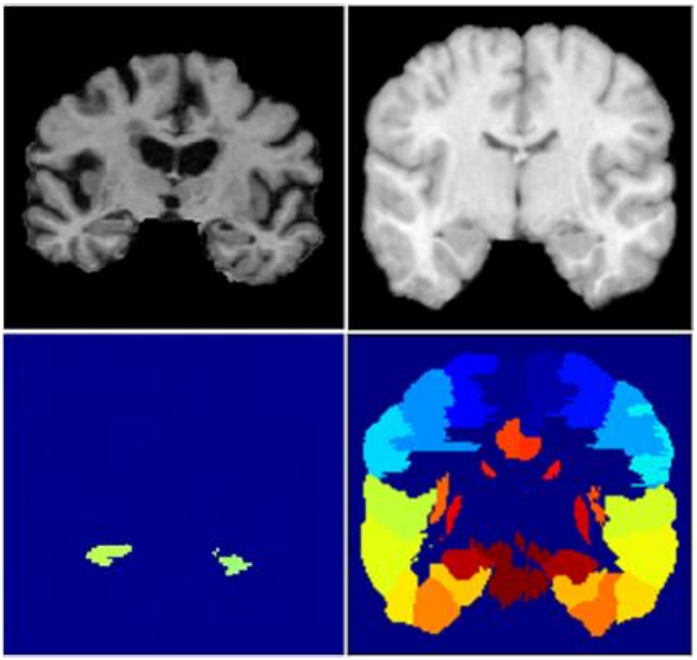


Figure 3.5: Examples of MR images from the ADNI dataset (left) and the LONI LPBA40 dataset (right). The top row displays the intensity images, and the bottom row displays the manually segmented labels.

We use classic U-Net as the baseline method to evaluate the performance of our proposed methods . Dice ratio is used to measure the voxel-wise prediction accuracy. It calculates the degree of overlap between the predicted area and its corresponding ground truth for each target class. It is defined by

$$DICE(A, B) = \frac{2|A \cap B|}{|A| + |B|}, \tag{3.10}$$

where A is the predicted result and B is the corresponding true label. For k different

regions of interest (ROI), an averaged dice ratio is calculated, given by

$$DICE = \frac{1}{k}\sum_{i=1}^{k} DICE(A_i, B_i) = \frac{1}{k}\sum_{i=1}^{k} \frac{2|A_i \cap B_i|}{|A_i| + |B_i|}, \tag{3.11}$$

where A_i represents the prediction for the i^{th} ROI and B_i represents the corresponding ground truth label.

3.2.1 ADNI Dataset

The ADNI dataset is mostly used to label hippocampus in brains. The task is to determine whether a certain part of the brain is hippocampus or not. We consider it as a pixel-wise prediction task with binary classes.

Data Preparation: There are 64 samples in the ADNI dataset with sizes of $96 \times 124 \times 96$. We randomly split the dataset into training data with 59 samples and testing data with 5 samples. We repeat this process for five times and build five pixel-wise prediction models respectively. The averaged dice ratio is the reported performance of this experiment.

Experimental Setup: We build the baseline model using U-Net architecture as shown in Figure 3.4. The number of output channels in the first encoder block is 16. The convolutional kernel size is $3 \times 3 \times 3$ and the stride size used in the max pooling layers is $2 \times 2 \times 2$. The last convolutional operation in the final decoder block has 2 output channels, which corresponds to the total number of classes. We apply batch

normalization after each convolutional layer, except for the output layer. The training batch size is 4. We implement our methods on Tensorflow and use AdamOptimizer to update the network with moment estimates $\beta 1 = 0.9$ and $\beta 2 = 0.999$.

Analysis of Results: Table 3.1 shows the overall dice ratios of the baseline method and the proposed methods. We can see that all the proposed methods have higher dice ratios than the baseline method. Model iVoxelDCNa achieves 83.34% in dice ratio, which improves the baseline method by 1.39%. The result demonstrates that the proposed methods, which intend to build relationships among intermediate feature maps, can help networks better capture local information of images than the baseline method.

3.2.2 LONI LPBA40 Dataset

The LONI LPBA40 dataset is designed as a multi-class voxel-wise prediction task. It requires 54 regions of interest to be labeled in brain images automatically.

Data Preparation There are 40 brain image samples with sizes of $220 \times 220 \times 220$ in the dataset. Each image has 54 manually labeled ROIs along with cerebrum, brainstem, and background. When calculating result, only dice ratios of the 54 ROIs are included. Since we are not interested in predicting background, we first crop the image to drop the excessive background area around boundaries. The cropped image

Table 3.1: Experimental results on the ADNI and LONI LPBA40 datasets.

Dataset	Model	Dice Ratio (%)
ADNI	U-Net	82.1985
	iVoxelDCNc	82.8784
	iVoxelDCNa	**83.3403**
	VoxelDCNc	82.2777
	VoxelDCNa	82.9553
LONI LPBA40	U-Net	77.4209
	iVoxelDCNc	78.3524
	iVoxelDCNa	**79.1294**
	VoxelDCNc	78.7590
	VoxelDCNa	77.7654

size is $180 \times 145 \times 137$. Then we split the dataset into training set with 16 samples, validation set with 4 samples and testing set with the remaining 20 samples. We flip the images across all three directions on training and validation datasets. This increases the number of training samples to 128 and the number of validation samples to 32. The size of testing data remains 20 as we do not apply data augmentation on testing data.

Experimental Setup: The architecture of U-Net implemented on this dataset consists of five blocks in both the encoder path and decoder path. The number of output channels in the first encoder block is 32. The convolutional kernel size is $3\times3\times3$ and stride size used in the max pooling layers is $1\times2\times2$. The last convolutional

operation in the final decoder block has 57 output channels, which corresponds to the total number of classes (54 ROIs, cerebrum, brainstem and background). The training batch size is 1. Since each data sample provided is too large to be stored in memory, we use training patch with size of $128 \times 128 \times 32$ as input in the training process. Each training patch are randomly cropped from training data. At the testing time, we pick one patch at a time from a single testing data sample with the same size in turn. The predicted patches are concatenated together with a linear weighted summation on overlapped part to get the final prediction results. We implement our models on Tensorflow and use AdamOptimizer to update the network with moment estimates $\beta 1 = 0.9$ and $\beta 2 = 0.999$.

Analysis of Results: The dice ratio of ROIs in the left and the right hemispheres are provided in Figure 3.6. Compared with the baseline method, iVoxelDCNa shows improvement in 44 out of 54 ROIs; VoxelDCNc shows improvement in 35 out of 54 ROIs; iVoxelDCNc shows improvement in 30 out of 54 ROIs; VoxelDCNa shows improvement in 29 of 54 ROIs. Table 3.1 shows the experimental results in terms of average dice ratio. All the proposed methods have better performance than the baseline model. Network iVoxelDCNa has the best performance and achieves 2.21% improvement compared with baseline. The result demonstrates that the proposed methods can help networks better capture the local information of images than the baseline method by eliminating the checkerboard artifact, yielding a better voxel-wise

prediction result.

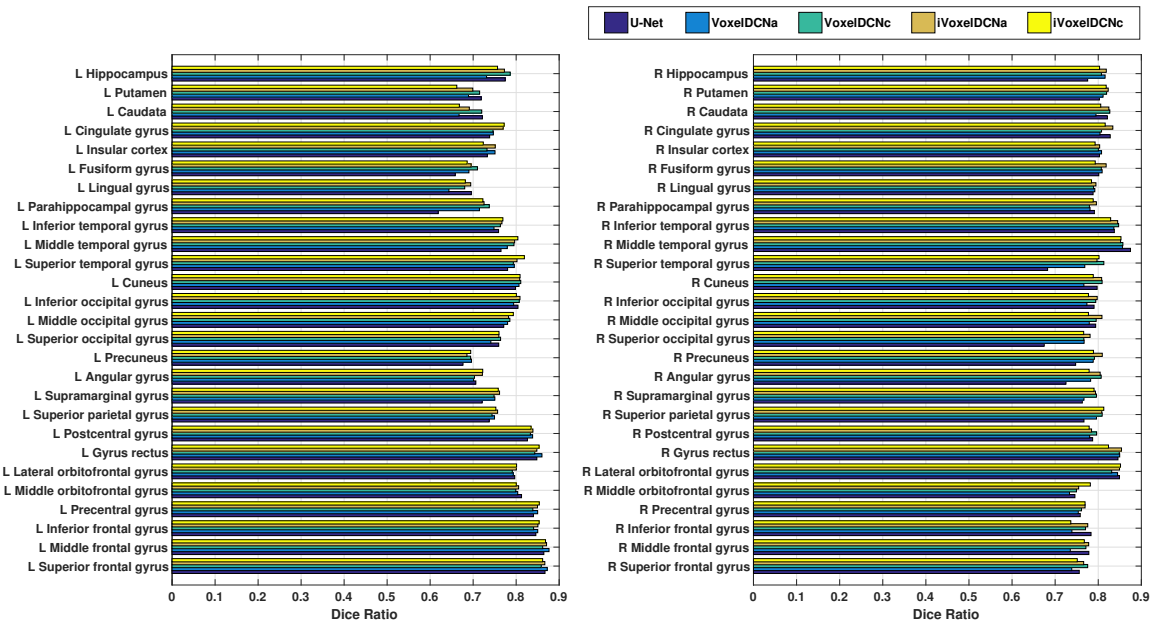


Figure 3.6: Experimental results on the LONI LPBA40 datasets. Each set of bars represents the dice ratios of 3D U-Net, VoxelDCLa, VoxelDCLc, iVoxelDCLa, iVoxelDCLc on testing data for each ROI.

3.2.3 ***Timing Comparison***

Table 3.2 shows the comparison of the training and testing time between baseline and proposed methods. We can see that iVoxelDCLa and iVoxelDCLc take more time to train than VoxelDCLa and VoxelDCLc in that they take original input when generating each intermediate feature map. In VoxelDCL, the use of concatenation

Table 3.2: Training and testing time on the two datasets using Tesla K80 GPU. We compare the training time of 10,000 (ADNI) and 100,000 iterations (LONI LPBA40), and testing time of 5 (ADNI) and 20 testing subjects for the baseline U-Net and the proposed methods.

Dataset	**Model**	**Training time**	**Testing time**
ADNI	U-Net	45 h 43 min	19.45 sec
	VoxelDCNa	54 h 57 min	20.80 sec
	VoxelDCNc	57 h 12 min	19.18 sec
	iVoxelDCNa	59 h 31 min	19.87 sec
	iVoxelDCNc	59 h 54 min	19.65 sec
LONI LPBA40	U-Net	52 h 05 min	12 min 12 sec
	VoxelDCNa	60 h 37 min	20 min 35 sec
	VoxelDCNc	63 h 24 min	19 min 85 sec
	iVoxelDCNa	71 h 12 min	20 min 32 sec
	iVoxelDCNc	75 h 40 min	20 min 05 sec

slightly increases the training and testing time. All the proposed methods take more time to train and test than the baseline model, but the increase is not dramatic. Overall, we do not expect this to be a major bottleneck of the proposed methods.

CHAPTER 4. DENSE TRANSFORMER NETWORKS FOR 2D IMAGE PIXEL-WISE PREDICTION

4.1 Dense Transformer Networks

The central idea of CNN-based method for pixel-wise prediction is to extract a regular patch centered on each pixel and apply CNNs to compute the label of that pixel. A common property of these methods is that the label of each pixel is determined by a regular (typically square) patch centered on that pixel. Although these methods have been shown to be effective on pixel-wise prediction problems, they lack the ability to learn the sizes and shapes of patches in a data-dependent manner. For a given network, the size of patches used to predict the labels of each center pixel is determined by the network architecture. Although multi-scale networks have been proposed to allow patches of different sizes to be combined [10], the patch sizes are again determined by network architectures. In addition, the shapes of patches used in CNNs are invariably regular, such as squares. Ideally, the shapes of patches may depend on local image statistics around that pixel and thus should be learned from data. In this work, we propose the dense transformer networks to enable the learning of patch size and shape for each pixel.

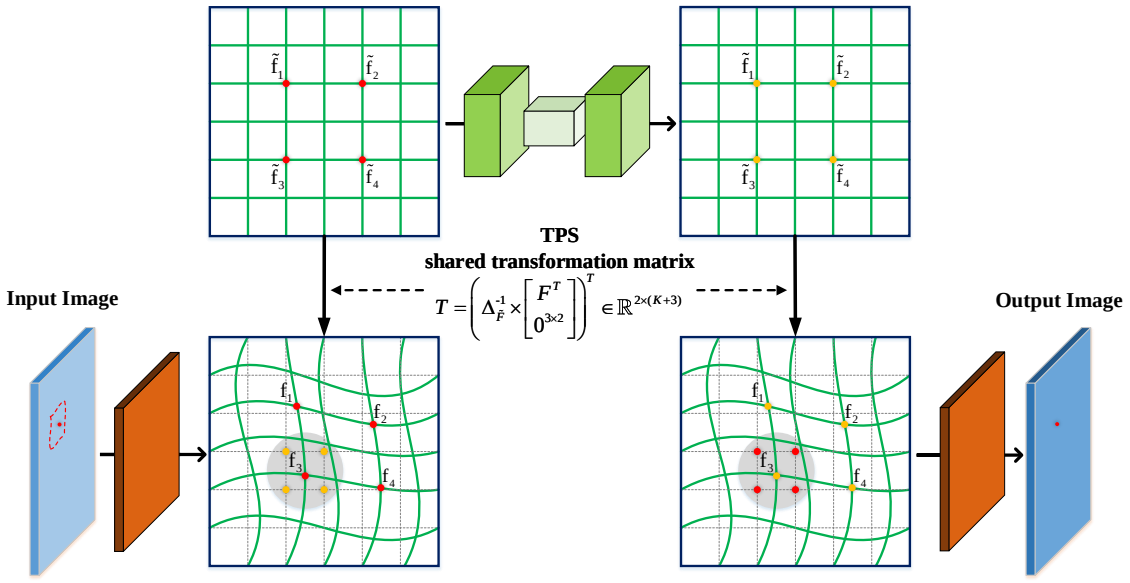


Figure 4.1: Illustration of the dense transformer networks.

As illustrated in Figure 4.1, features extracted by the original convolutional operation corresponds to a regular path on input feature maps. We employ a nonlinear TPS transformation in our model. Therefore, a regular patch $\widetilde{f}_1\widetilde{f}_2\widetilde{f}_3\widetilde{f}_4$ corresponds to an area $f_1f_2f_3f_4$ with different shape and size. The parameters of nonlinear transformation are learned by a network based on inputs. Therefore, our model can learn an appropriate transformation to achieve better performance. To tackle the spatial correspondence problem, a reverse transformation is used to restore the spatial correspondence. The reverse TPS transformation share parameters with the TPS transformation in the encoder network.

4.1.1 An Encoder-Decoder Architecture

In order to address the above limitations, we propose to develop a dense transformer network model. Our model employs an encoder-decoder architecture in which the encoder path extracts high-level representations using convolutional and pooling layers and the decoder path uses deconvolution and un-pooling to recover the original spatial resolution [33, 35, 2, 32]. To enable the learning of size and shape of each patch automatically from data, we propose to insert a spatial transformer module in the encoder path in our network. As has been discussed above, the spatial transformer module transforms the feature maps into a different space using nonlinear transformations. Applying convolution and pooling operations on regular patches in the transformed space is equivalent to operating on irregular patches of different sizes in the original space. Since the spatial transformer module is differentiable, its parameters can be learned with error back-propagation algorithms. This is equivalent to learning the size and shape of each patch from data.

Although the patches used to predict pixel labels could be of different sizes and shapes, we expect the patches to include the pixel in question at least. That is, the patches should be in the spatial vicinity of pixels whose labels are to be predicted. By using the nonlinear spatial transformer layer in encoder path, the spatial locations of units on the intermediate feature maps could have been changed. That is, due

to this nonlinear spatial transformation, the spatial correspondence between input images and output label maps is not retained in the feature maps after the spatial transformer layer. In order to restore this spatial correspondence, we propose to add a corresponding decoder layer, known as the dense transformer decoder layer. This decoder layer transforms the intermediate feature maps back to the original input space, thereby re-establishing the input-output spatial correspondence.

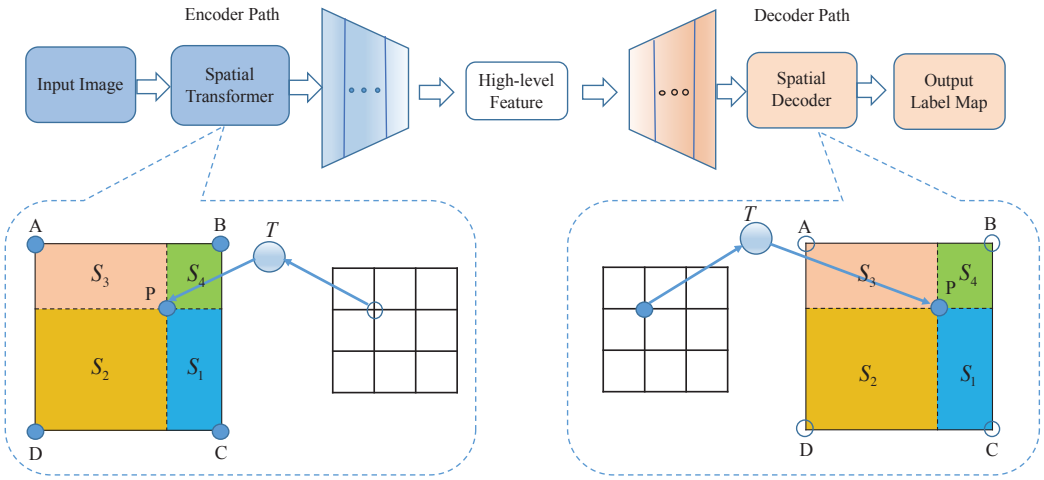


Figure 4.2: The proposed dense transformer networks. A pair of dense transformer modules are inserted into each of the encoder and decoder paths. In the spatial transformer module, values at points A, B, C, and D are given from the previous layer, and we need to estimate value for point P. In contrast, in the decoder layer, value at point P is given from the previous layer, and we need to estimate values for points A, B, C, and D.

The spatial transformer module can be inserted after any layer in the encoder path while the dense transform decoder module should be inserted into the corresponding location in decoder path. In our framework, the spatial transformer module is required to not only output the transformed feature maps, but also the transformation itself that captures the spatial correspondence between input and output feature maps. This information will be used to restore the spatial correspondence in the decoder module. Note that in the spatial transformer encoder module, the transformation is computed in the backward direction, i.e., from output to input feature maps (Figure 4.2). In contrast, the dense transformer decoder module uses a forward direction instead; that is, a mapping from input to output feature maps. This encoder-decoder pair can be implemented efficiently by sharing the transformation parameters in these two modules.

A technical challenge in developing the dense transformer decoder layer is that we need to map values of units arranged on input regular grid to another set of units arranged on regular output grid, while the decoder could map to units at arbitrary locations on the output map. That is, while we need to compute the values of units lying on regular output grid from values of units lying on regular input grid, the mapping itself could map an input unit to an arbitrary location on the output feature map, i.e., not necessarily to a unit lying exactly on the output grid. To address this challenge, we develop a sampler method for performing interpolation. We show that

the proposed samplers are differentiable, thus gradients can be propagated through these modules. This makes the entire dense transformer networks fully trainable. Formally, assume that the encoder and decoder layers are inserted after the i-th and j-th layers, respectively, then we have the following relationships:

$$\begin{aligned} U^{i+1}(p) &= \text{Sampling}\{U^i(Tp)\}, \quad U^{j+1}(Tp) = U^j(p), \\ U^{j+1}(p) &= \text{Sampling}\{U^{j+1}(Tp)\}, \end{aligned} \tag{4.1}$$

where U^i is the feature map of the i-th layer, p is the coordinate of a point, T is the transformation defined in Eq. (2.1), which maps from the coordinates of the $(i+1)$-th layer to the i-th layer, Sampling$(\cdot)$ denotes the sampler function.

From a geometric perspective, a value associated with an estimated point in bilinear interpolation in Eq. (2.4) can be interpreted as a linear combination of values at four neighboring grid points. The weights for linear combination are areas of rectangles determined by the estimated points and four neighboring grid points. For example, in Figure 4.2, when a point is mapped to P on input grid, the contributions of points A, B, C, and D to the estimated point P is determined by the areas of the rectangles S_1, S_2, S_3, and S_4. However, the interpolation problem needs to be solved in the dense transformer decoder layer is different with the one in the spatial transformer encoder layer, as illustrated in Figure 4.2. Specifically, in the encoder layer, the points A, B, C, and D are associated with values computed from the previous layer, and the interpolation problem needs to compute a value for P to be propagated

to the next layer. In contrast, in the decoder layer, the point P is associated with a value computed from the previous layer, and the interpolation problem needs to compute values for A, B, C, and D. Due to the different natures of the interpolation problems need to be solved in the encoder and decoder modules, we propose a new sampler that can efficiently interpolate over decimal points in the following section.

4.1.2 Decoder Sampler

In the decoder sampler, we need to estimate values of regular grid points based on those from arbitrary decimal points, i.e., those that do not lie on the regular grid. For example, in Figure 4.2, the value at point P is given from the previous layer. After the TPS transformation in Eq. (2.3), it may be mapped to an arbitrary point. Therefore, the values of grid points A, B, C, and D need to be computed based on values from a set of arbitrary points. If we compute the values from surrounding points as in the encoder layer, we might have to deal with a complex interpolation problem over irregular quadrilaterals. Those complex interpolation methods may yield more accurate results, but we prefer a simpler and more efficient method in this work. Specifically, we propose a new sampling method, which distributes the value of P to the points A, B, C, and D in an intuitive manner. Geometrically, the weights associated with points A, B, C, and D are the area of the rectangles S_1, S_2, S_3, and

S_4, respectively (Figure 4.2). In particular, given an input feature map $V \in \mathbb{R}^{\tilde{H}\times\tilde{W}}$, the output feature map $U \in \mathbb{R}^{H\times W}$ can be obtained as

$$\begin{aligned} S_{nm} &= \sum_{i=1}^{\tilde{H}\times\tilde{W}} \max(0, 1-|x_i - m|)\max(0, 1-|y_i - n|), \\ U_{nm} &= \frac{1}{S_{nm}} \sum_{i=1}^{\tilde{H}\times\tilde{W}} V_i \max(0, 1-|x_i - m|) \times \\ &\qquad \max(0, 1-|y_i - n|), \end{aligned} \tag{4.2}$$

where V_i is the value of pixel i, $p_i = (x_i, y_i)^T$ is transformed by the shared transformation T in Eq. (2.1), U_{nm} is the value at the (n, m)-th location on the output feature map, S_{nm} is a normalization term that is used to eliminate the effect that different grid points may receive values from different numbers of arbitrary points, and $n = 1, 2, \cdots, N$, $m = 1, 2, \cdots, M$. More details are given in Figure 4.1.

In order to allow the backpropagation of errors, we define the gradient with respect to U_{nm} as dU_{nm}. Then the gradient with respect to V_{nm} and x_i can be derived as

follows:

$$dV_i = \sum_{n=1}^{H}\sum_{m=1}^{W} \frac{1}{S_{nm}} dU_{nm} \max(0, 1-|x_i - m|) \times \max(0, 1-|y_i - n|), \tag{4.3}$$

$$dS_{nm} = \frac{-dU_{nm}}{S_{nm}^2} \sum_{i=1}^{\tilde{H}\times\tilde{W}} V_i \max(0, 1-|x_i - m|) \times \max(0, 1-|y_i - n|), \tag{4.4}$$

$$dx_i = \sum_{n=1}^{H}\sum_{m=1}^{W} \left\{ \frac{dU_{nm}}{S_{nm}} V_i + dS_{nm} \right\} \max(0, 1-|y_i - n|) \times \begin{cases} 0 & \text{if} \quad |m - x_i| \geq 1 \\ 1 & \text{if} \quad m \geq x_i \\ -1 & \text{if} \quad m \leq x_i \end{cases}. \tag{4.5}$$

A similar gradient can be derived for dy_i. This provides us with a differentiable sampling mechanism, which enables the gradients flow back to both the input feature map and the sampling layers.

4.2 Experimental Studies

We evaluate the proposed methods on two image pixel-wise prediction tasks. The U-Net [35] is adopted as our base model in both tasks, as it has achieved state-of-the-art performance on image pixel-wise prediction tasks. Specifically, U-Net adds residual connections between the encoder path and decoder path to incorporate both low-level and high-level features. Other methods like SegNet [2], deconvolutional

networks [48] and FCN [28] mainly differ from U-Net in the up-sampling method and do not use residual connections. Experiments in prior work [35, 48, 28] show that residual connections are important while different up-sampling methods lead to similar results. The network consists of 5 layers in the encoder path and another corresponding 5 layers in the decoder path. A stack of two 3×3 convolutional layers have the same receptive field as a 5×5 convolutional layer, but with less parameters [42]. Therefore, we use 3×3 kernels and one pixel padding to retain the size of feature maps at each level.

Table 4.1: Comparison of pixel-wise prediction performance between the U-Net and the proposed DTN on the PASCAL 2012 segmentation data set. Three different performance measures are used here. An arrow is attached to each measure so that ↑ denotes higher values indicate better performance, and ↓ denotes lower values indicate better performance.

Data Set	Model	Loss↓	Accuracy↑	Mean-IOU↑
PASCAL	U-Net	0.9396	0.8117	0.4145
	DTN	**0.7909**	**0.8367**	**0.5297**

In order to efficiently implement the transformations, we insert the spatial encoder layer and dense transformer decoder layer into corresponding positions at the

same level. Using spatial transformation layers at a deep position in the encoder-decoder network can increase the receptive field of the spatial transformation operation. Therefore, the layers are applied to the 4th layer, and their performance is compared to the basic U-Net model without spatial transformations. As for the transformation layers, we use 16 fiducial points that are evenly distributed on the output feature maps. In the dense transformer decoder layer, if there are pixels that are not selected on the output feature map, we apply an interpolation strategy over its neighboring pixels on previous feature maps to produce smooth results.

4.2.1 Natural Image Pixel-Wise Prediction

We use the PASCAL 2012 segmentation data set [9] to evaluate the proposed methods on natural image pixel-wise prediction task. In this task, we predict one label out of a total of 21 classes for each pixel. To avoid the inconvenience of different sizes of images, we resize all the images to 256×256. Multiple performance metrics, including loss, accuracy, and mean-IOU, are used to measure the pixel-wise prediction performance, and the results are reported in Table 4.1. The state-of-the-art models for the PASCAL 2012 segmentation task employ the VGG or ResNet network that is pre-trained on the ImageNet or MSCOCO datasets.

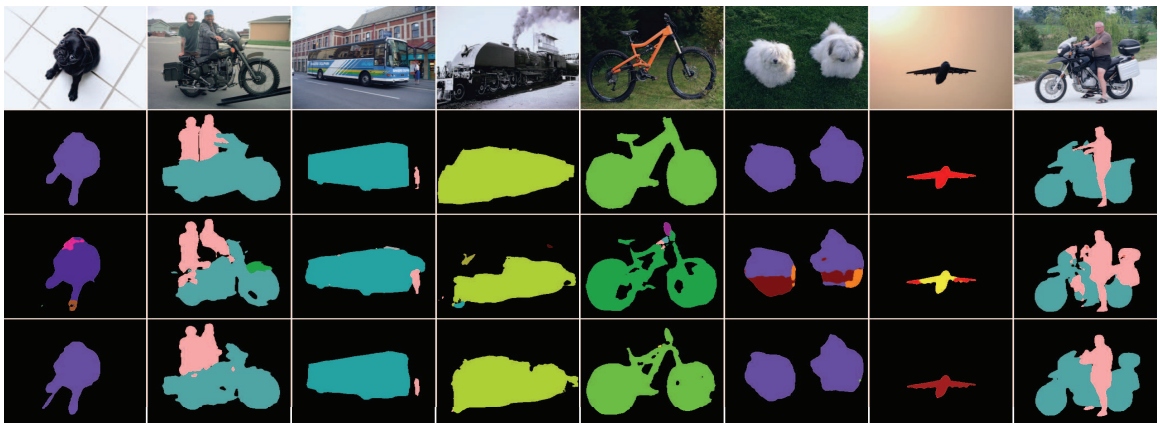


Figure 4.3: Sample pixel-wise prediction results on the PASCAL 2012 segmentation data set. The first and second rows are the original images and the corresponding ground truth, respectively. The third and fourth rows are the pixel-wise prediction results of U-Net and DTN, respectively.

These models have many layers and a large number of parameters. In the experiments, we use 5 layers encoder-decoder architecture with random initial parameters. The number of parameters in our model is significantly less than those of the pre-trained models. Our experiments focus on demonstrating that our spatial transformer layers can improve the performance for encoder-decoder architecture. We can observe that the proposed DTN model achieves higher performance than the baseline U-Net model. Especially, it improves the mean-IOU from 0.4145 to 0.5297. Some example results along with the raw images and ground truth label maps are given in Figure 4.3. These results demonstrate that the proposed DTN model can boost the pixel-wise

prediction performance dramatically.

4.2.2 Brain Electron Microscopy Image Pixel-Wise Prediction

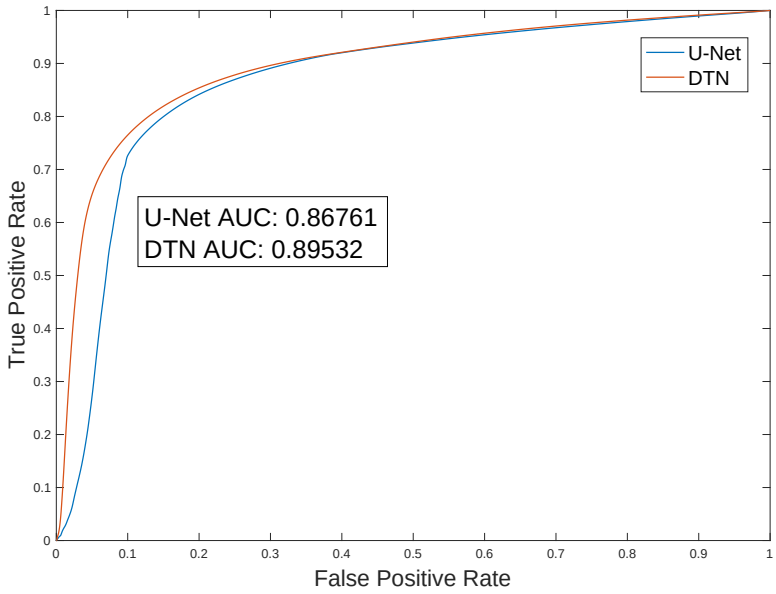
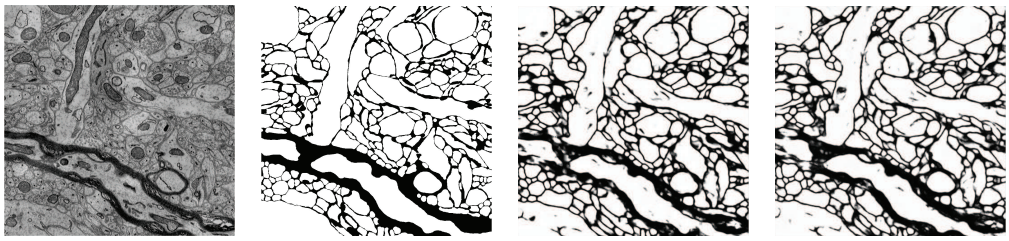


Figure 4.4: Comparison of the ROC curves of the U-Net and the proposed DTN model on the SNEMI3D data set.

We evaluate the proposed methods on brain electron microscopy (EM) image pixel-wise prediction task [26, 4], in which the ultimate goal is to reconstruct neurons at the micro-scale level. A critical step in neuron reconstruction is to segment the EM images. We use data set from the 3D Segmentation of Neurites in EM Images (SNEMI3D, http://brainiac2.mit.edu/SNEMI3D/). The SNEMI3D data set consists

of 100 1024×1024 EM image slices. Since we perform 2D transformations in this work, each image slice is segmented separately in our experiments. The task is to predict each pixel as either a boundary (denoted as 1) or a non-boundary pixel (denoted as 0).



Raw image Ground truth U-Net output DTN output

Figure 4.5: Example results generated by the U-Net and the proposed DTN models for the SNEMI3D data set.

Our model can process images of arbitrary size. However, training on whole images may incur excessive memory requirement. In order to accelerate training, we randomly pick 224×224 patches from the original images and use it to train the networks. The experimental results in terms of ROC curves are provided in Figure 4.4. We can observe that the proposed DTN model achieves higher performance than the baseline U-Net model, improving AUC from 0.8676 to 0.8953. These results demonstrate that the proposed DTN model improves upon the baseline U-Net model, and the use of the dense transformer encoder and decoder modules in the U-Net

architecture results in improved performance. Some example results along with the raw images and ground truth label maps are given in Figure 4.5.

4.2.3 Timing Comparison

Table 4.2 shows the comparison of training and prediction time between the U-Net model and the proposed DTN model on the two data sets. We can see that adding DTN layers leads to only slight increase in training and prediction time. Since the PASCAL data set is more complex than the SNEMEI3D data set, we use more channels when building the network of natural image pixel-wise prediction task. That causes the increase of training and prediction time on the PASCAL data set as compared to SNEMEI3D.

Table 4.2: Training and prediction time on the two data sets using a Tesla K40 GPU. We compare the training time of 10,000 iterations and prediction time of 2019 (PASCAL) and 40 (SNEMI3D) images for the base U-Net model and the DTN.

Data Set	Model	Training Time	Prediction Time
PASCAL	U-Net	378m57s	14m06s
	DTN	402m07s	15m50s
SNEMI3D	U-Net	14m18s	3m31s
	DTN	15m41s	4m02s

CHAPTER 5. CONCLUSION

In this thesis, I proposed voxel deconvolutional layers (VoxelDCL) to address the checkerboard artifact problem of 3D deconvolutional layers (DCL) in image analysis and prediction tasks. VoxelDCL generate the intermediate feature maps of 3D DCL sequentially, thereby building relationships among the adjacent voxels on the output feature maps. I built four variations of voxel deconvolutional networks (VoxelDCN) and applied them to the ADNI and LONI PBA40 datasets for volumetric brain image voxel-wise prediction tasks. Experimental results demonstrated the increased effectiveness of these new methods. All of the proposed networks outperformed U-Net with regular 3D DCL used as the baseline method. These results indicate that developing dependencies among the intermediate feature maps in DCL indeed alleviate the checkerboard artifact issue, thereby improving prediction accuracy. I applied the VoxelDCL specifically to U-Net for voxel-wise prediction tasks. But, in general, VoxelDCL can also be applied to other 3D deep network architectures with up-sampling operations. Thus, I plan to apply the proposed methods to models such as 3D generative adversarial networks for image generation tasks in the future.

I also proposed dense transformer networks to enable automatic learning of patch sizes and shapes in 2D pixel-wise prediction tasks. A unique challenge in pixel-wise prediction tasks is to preserve the spatial correspondence between inputs and outputs

in order to make pixel-wise predictions. To this end, I developed dense transformer decoder layers to restore this spatial correspondence. Experimental results showed that adding the spatial transformer and decoder layers to existing models leads to improved performance. In this work, I developed a simple and efficient decoder sampler for interpolation. However, a more complex method, based on irregular quadrilaterals, may increase prediction accuracy and will be explored in the future.

In summary, I developed two novel, state-of-the-art deep learning methods for image pixel/voxel-wise prediction in both 2D and 3D space. Experimental results demonstrated significant improvements in the performance of pixel/voxel-wise prediction by resolving the existing limitations in current deep learning methods.

BIBLIOGRAPHY

[1] Andrew Aitken, Christian Ledig, Lucas Theis, Jose Caballero, Zehan Wang, and Wenzhe Shi. Checkerboard artifact free sub-pixel convolution: A note on sub-pixel convolution, resize convolution and convolution resize. *arXiv preprint arXiv:1707.02937*, 2017.

[2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*, 2015.

[3] 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. *arXiv preprint arXiv:1606.00915*, 2016.

[4] Dan Ciresan, Alessandro Giusti, Luca M Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *Advances in neural information processing systems*, pages 2843–2851, 2012.

[5] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 2990–2999, 2016.

[6] Angela Dai, Daniel Ritchie, Martin Bokeloh, Scott Reed, Jürgen Sturm, and Matthias Nießner. Scancomplete: Large-scale scene completion and semantic segmentation for 3d scans. *arXiv preprint arXiv:1712.10215*, 2017.

[7] Sander Dieleman, Jeffrey De Fauw, and Koray Kavukcuoglu. Exploiting cyclic symmetry in convolutional neural networks. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1889–1898, 2016.

[8] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2758–2766, 2015.

[9] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[10] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1915–1929, 2013.

[11] Christine Fennema-Notestine, Donald J Hagler, Linda K McEvoy, Adam S Fleisher, Elaine H Wu, David S Karow, and Anders M Dale. Structural mri

biomarkers for preclinical and mild alzheimer's disease. *Human brain mapping*, 30(10):3238–3253, 2009.

[12] Hongyang Gao, Hao Yuan, Zhengyang Wang, and Shuiwang Ji. Pixel deconvolutional networks. *arXiv preprint arXiv:1705.06820*, 2017.

[13] Yaozong Gao, Shu Liao, and Dinggang Shen. Prostate segmentation by sparse representation based classification. *Medical physics*, 39(10):6372–6387, 2012.

[14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[16] João F Henriques and Andrea Vedaldi. Warped convolutions: Efficient invariance to spatial transformations. *arXiv preprint arXiv:1609.04382*, 2016.

[17] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating

deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[18] Md Amirul Islam, Neil Bruce, and Yang Wang. Dense image labeling using deep convolutional neural networks. In *Computer and Robot Vision (CRV), 2016 13th Conference on*, pages 16–23. IEEE, 2016.

[19] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025, 2015.

[20] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3D convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, 2013.

[21] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. In *Advances in Neural Information Processing Systems*, pages 667–675, 2016.

[22] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711. Springer, 2016.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification

with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[25] 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.

[26] Kisuk Lee, Aleksandar Zlateski, Vishwanathan Ashwin, and H Sebastian Seung. Recursive training of 2D-3D convolutional networks for neuronal boundary prediction. In *Advances in Neural Information Processing Systems*, pages 3573–3581, 2015.

[27] Kisuk Lee, Jonathan Zung, Peter Li, Viren Jain, and H Sebastian Seung. Superhuman accuracy on the snemi3d connectomics challenge. *arXiv preprint arXiv:1706.00120*, 2017.

[28] 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.

[29] Guangkai Ma, Yaozong Gao, Guorong Wu, Ligang Wu, and Dinggang Shen. Nonlocal atlas-guided multi-channel forest learning for human brain labeling. *Medical physics*, 43(2):1003–1019, 2016.

[30] Vincent A Magnotta, Dan Heckel, Nancy C Andreasen, Ted Cizadlo, Patricia Westmoreland Corson, James C Ehrhardt, and William TC Yuh. Measurement of brain structures with artificial neural networks: two-and three-dimensional applications. *Radiology*, 211(3):781–790, 1999.

[31] Susanne G Mueller, Michael W Weiner, Leon J Thal, Ronald C Petersen, Clifford Jack, William Jagust, John Q Trojanowski, Arthur W Toga, and Laurel Beckett. The alzheimer's disease neuroimaging initiative. *Neuroimaging Clinics of North America*, 15(4):869–877, 2005.

[32] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European Conference on Computer Vision*, pages 483–499. Springer, 2016.

[33] 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.

[34] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

[35] 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.

[36] Gerard Sanroma, Guorong Wu, Yaozong Gao, and Dinggang Shen. Learning to rank atlases for multiple-atlas segmentation. *IEEE transactions on medical imaging*, 33(10):1939–1953, 2014.

[37] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated recognition, localization and detection using convolutional networks. In *Proceedings of the International Conference on Learning Representations*, April 2014.

[38] David W Shattuck, Mubeena Mirza, Vitria Adisetiyo, Cornelius Hojatkashani, Georges Salamon, Katherine L Narr, Russell A Poldrack, Robert M Bilder, and Arthur W Toga. Construction of a 3d probabilistic atlas of human cortical structures. *Neuroimage*, 39(3):1064–1080, 2008.

[39] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(4):640–651, 2017.

[40] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken,

Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1874–1883, 2016.

[41] Wenzhe Shi, Jose Caballero, Lucas Theis, Ferenc Huszar, Andrew Aitken, Christian Ledig, and Zehan Wang. Is the deconvolution layer the same as a convolutional layer? *arXiv preprint arXiv:1609.07009*, 2016.

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

[43] Tong Tong, Robin Wolz, Joseph V Hajnal, and Daniel Rueckert. Segmentation of brain mr images via sparse patch representation. In *MICCAI Workshop on Sparsity Techniques in Medical Imaging (STMI)*, 2012.

[44] Hongzhi Wang, Jung W Suh, Sandhitsu R Das, John B Pluta, Caryne Craige, and Paul A Yushkevich. Multi-atlas segmentation with joint label fusion. *IEEE transactions on pattern analysis and machine intelligence*, 35(3):611–623, 2013.

[45] Li Wang, Yaozong Gao, Feng Shi, Gang Li, John H Gilmore, Weili Lin, and Dinggang Shen. Links: Learning-based multi-source integration framework for segmentation of infant brain images. *NeuroImage*, 108:160–172, 2015.

[46] Guorong Wu, Minjeong Kim, Gerard Sanroma, Qian Wang, Brent C Munsell, Dinggang Shen, Alzheimer's Disease Neuroimaging Initiative, et al. Hierarchical multi-atlas label fusion with multi-scale feature representation and label-specific patch partition. *NeuroImage*, 106:34–46, 2015.

[47] Guorong Wu, Qian Wang, Daoqiang Zhang, Feiping Nie, Heng Huang, and Dinggang Shen. A generative probability model of joint label fusion for multi-atlas based brain segmentation. *Medical image analysis*, 18(6):881–890, 2014.

[48] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, and Rob Fergus. Deconvolutional networks. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2528–2535. IEEE, 2010.

[49] Matthew D Zeiler, Graham W Taylor, and Rob Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2018–2025. IEEE, 2011.

[50] Tao Zeng, Bian Wu, and Shuiwang Ji. DeepEM3D: Approaching human-level performance on 3D anisotropic EM image segmentation. *Bioinformatics*, 33(16):2555–2562, 2017.

[51] Lichi Zhang, Qian Wang, Yaozong Gao, Guorong Wu, and Dinggang Shen. Automatic labeling of mr brain images by hierarchical learning of atlas forests. *Medical physics*, 43(3):1175–1186, 2016.

[52] Shaoting Zhang, Yiqiang Zhan, Maneesh Dewan, Junzhou Huang, Dimitris N Metaxas, and Xiang Sean Zhou. Towards robust and effective shape modeling: Sparse shape composition. *Medical image analysis*, 16(1):265–277, 2012.

[53] Wenlu Zhang, Rongjian Li, Houtao Deng, Li Wang, Weili Lin, Shuiwang Ji, and Dinggang Shen. Deep convolutional neural networks for multi-modality isointense infant brain image segmentation. *NeuroImage*, 108:214–224, 2015.