Learning-Based Segmentation Using Superpixel Pairs

Jin-Yu Huang

Advisor: Jiang-Jung Ding

Graduate Institute of Electronics Engineering

National Taiwan University

Taipei, Taiwan

June 2020

Abstract

Recently, the CNN has been widely adopted in image segmentation. However, the existing CNN-based segmentation algorithms are pixel-wise. It is hard to apply superpixels into the CNN architectures directly due to the irregular shape and size of superpixels. In this paper, we proposed different kinds of transformation techniques that leverage the CNN for learning superpixel-based image segmentation. The first proposed algorithm takes a square patch that contains two superpixel as the input of the CNN, and then the output of the CNN is whether the two superpixels should be merged or not. Additionally, one can obtain huge amount of training data even if there are only a few training images. Inspired by the first algorithm, we further proposed the second algorithm that utilizes the fully convolutional networks (FCN) to solve the problem from different perspective.

The second proposed algorithm takes a multi-channel image consisted of stacked color image and several feature maps such as superpixel boundary map and edge detection result as the input of a deep neural network, and the network outputs the prediction of superpixel boundary map that indicates whether the boundary of two adjacent superpixel should be keep or not, in a way, merging suprepixels. That is, by solving all the subproblems with just one forward pass, the FCN facilitates the speed of the whole segmentation process by a wide margin meanwhile gaining higher accuracy. Overall, simulations show that both proposed algorithms can achieve highly accurate segmentation results and outperforms state-of-the-art image segmentation methods in all evaluation metrics.

Keywords: Image segmentation, fully convolutional networks, superpixel.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Image segmentation is fundamental and important in many image processing applications. There are many existing image segmentation algorithms, including the region growing method [1], the mean shift method [2], the watershed [3, 4], the normalized cut [5, 6], the graph-based method [7], and the superpixel-based method [8, 9, 10].

In recent years, deep learning techniques have also been adopted in image segmentation [11, 12, 13, 14]. With supplicated deep learning architectures, one can achieve good segmentation results with enough training time.

In this thesis, two novel superpixel-based image segmentation algorithms based on deep neural networks are proposed. Classical superpixel-based algorithms [8, 9, 10] utilized several grouping cues to determine whether two superpixels should be merged. In this thesis, instead of applying these grouping cues, we use deep neural networks to decide whether two superpixels should be merged.

For the first proposed algorithm, different from other learning-based algorithms, instead of applying the whole image as the input, we apply the patch containing only two superpixels as the input of the deep neural network. Moreover, instead of outputting the segmentation result directly, the output of the network in the proposed algorithm is a label to indicate whether two superpixels should be merged. With such techniques, we successfully leveraged the CNN for learning generic

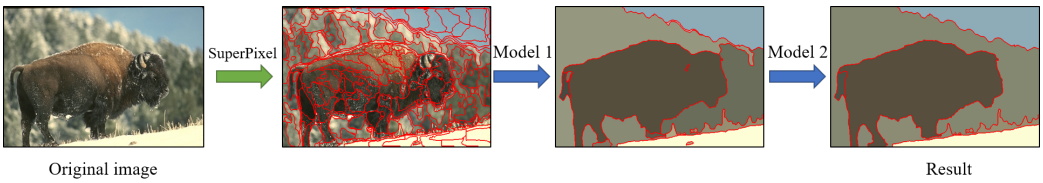


Figure 1.1: **Overview of the proposed algorithm.** An initial over-segmented image generated by Superpixel Sampling Network (SSN) is fed into series of superpixel-based deep learning models to produce the segmentation result.

image segmentation. We call it Deep Merging Models for Superpixel-Segmentation (*DMMSS*). Simulations show that the proposed *DMMSS* is much better than that of state-of-the-art methods in all evaluation metrics.

Fig. 1.1 shows the overview of the proposed method. Initially, an image is over-segmented by superpixels. Then, two learning-based merging models are applied to combine superpixels and obtain the segmentation result. The detail of the proposed algorithm will be illustrated in Chapter 3.

As for the second proposed algorithm, which is the refined version of the proposed *DMMSS*. Recently, the Fully Convolutional Networks (FCN) have been widely adopted in image segmentation for its high efficiency and accuracy. However, the existing FCN-based segmentation algorithms are designed for semantic segmentation. It is hard to integrate FCN into generic image segmentation. Before learning-based segmentation algorithms were developed, many advanced generic segmentation algorithms are superpixel-based. Unfortunately, due to the irregular shape and size of superpixels, there is no generic image segmentation algorithm that utilize both superpixel and FCN. In this thesis, we combined the merit of FCN and superpixel-based segmentation and proposed a highly accurate and extremely fast generic image segmentation algorithm. We call it *DMMSS-FCN* where *FCN* stands for Fully Convolutional Networks.

We treat the image segmentation as multiple superpixel merging decision problems. By correctly solving each subproblems, we can eventually get the

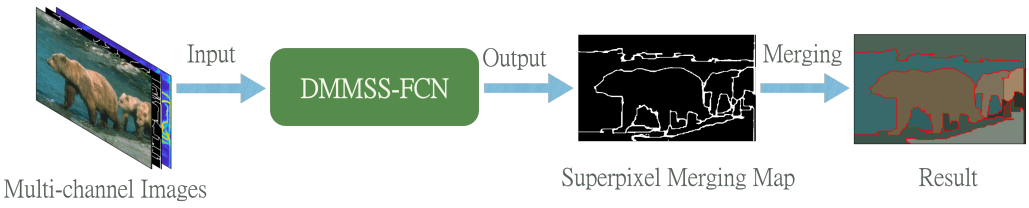


Figure 1.2: **Overview of the proposed *DMMSS-FCN* algorithm.** Multiple feature maps including an original RGB image are stacked together to form the input data, then being fed into the fully convolutional networks, the output is the superpixel merging map that tells which boundaries need to be keep. With such information, we perform superpixel merging based on the output of the model to produce the final segmentation result.

segmentation result. Furthermore, we converted each superpixel merging decision problems into boundary keeping problems. In other words, if we remove the boundary of two adjacent superpixels, it would be like merging those two superpixels.

The proposed algorithm takes a multi-channel image consisted of stacked color image and several feature maps such as superpixel boundary map and edge detection result as the input of a deep neural network, and the network outputs the prediction of superpixel merging map that indicates whether the boundary of two adjacent superpixel should be keep or not, in a way, making decision about merging suprepixels. That is, by solving all the subproblems with just one forward pass, the FCN facilitates the speed of the whole segmentation process by a wide margin meanwhile gaining higher accuracy.

Additionally, with the use of multiple superpixel generation algorithms, one can obtain a huge amount of training data by varying the parameters of different superpixel algorithms. That is to say, our method has low requirement for human-annotated groundtruth. Overall, simulations show that the proposed algorithm has favorable runtime, meanwhile achieve highly accurate segmentation results and outperforms state-of-the-art image segmentation methods in all evaluation metrics.

We show the overview of the second proposed *DMMSS-FCN* algorithm in Fig. 1.2. We will discuss the detail of *DMMSS-FCN* in Chapter 4.

Chapter 2

Related Work

2.1 Superpixels

Superpixels are a group of pixels with similar colors and locations. There are many types of superpixels, including the entropy rate superpixel (ERS) [15] and the simple linear iterative clustering (SLIC) [16] superpixel. In [2], the superpixel generated by mean shift was proposed. It has good edge-preserving property and the number of superpixels have not to be specified in advance. Moreover, its boundaries highly match the borders of objects. Recently, deep learning-based superpixels like Superpixel Sampling Network (SSN)[17], and Pixle Affinity Net (PAN) with Segmentation-aware Affinity Loss (SEAL) superpixels[18] were proposed. They both outperform non deep learning-based superpixels by a wide margin. In the following article, we will review the superpixel algorithms that used in our proposed method.

2.1.1 Mean Shift Superpixel

About Mean Shift Superpixel

Mean Shift [2] forms clustering by shifting every data point to its corresponding local maximum point with kernel density function to find modes in data. Since

Mean Shift is a nonparametric clustering method, it does not require prior information from user to define the number of clustering. The main parameter in the Mean Shift algorithm is the kernel bandwidth value. Different kernel bandwidth values change the result of the algorithm and can be chosen by specific domain. In image segmentation, we treat the kernel bandwidth value as the threshold of distance in color space for points within the same clustering.

In the Mean Shift superpixel generation process. Initially, each pixel of an image is considered to be a center of a window containing other pixels. Then, according to the mean value derived from pixels within the window, it updates the new center by shifting. Iteratively, the shifting process will continue until all center points are converged. In the end, the pixels that converge to the same mode form the superpixels.

Therefore, the choosing of kernel bandwidth value can be crucial for downstream tasks, for example, larger kernel bandwidth result in a small number of clustering which could be insufficient to be the underlying representation of the whole data. On the other hand, smaller kernel bandwidth might cause too many clusters that do not generalize the local grouping information. There are two parameters can adjust the kernel bandwidth, h_s, which is the kernel bandwidth for spacial domain, and h_r, the kernel bandwidth for range domain also known as color space such as gray level.

The Algorithm

For every input image, let x_i $i = 1, ..., n$ be the pixel in the joint spatial-range domain, and L_i be the label for i^th pixel in the result image .

Step 1

Initialize a window for x_i.

Step 2

Update the center of the window by computing the mean shift from the pixels in the window.

Step 3

Iteratively repeat *Step 2* until the window center of x_i converges.

Step 4

For every x_i whose convergence point is within h_s, h_r in the spatial domain and range domain respectively, assigning x_i to the cluster C_p in the joint domain.

Step 5

Assign $L_i = p$ for every pixel within C_p.

Step 6 (Optional)

Remove clusters that contain less than M pixels.

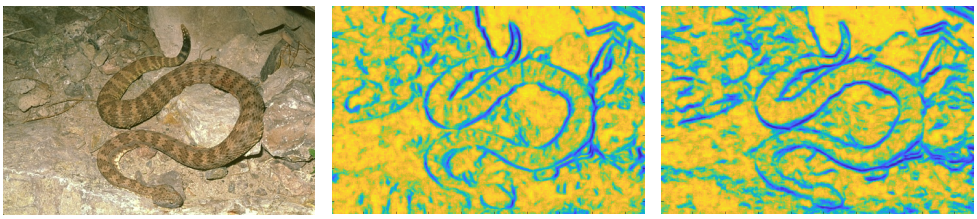
Simulations

In Fig.2.1, we show the superpixel results from Mean Shift algorithm using parameters of $(h_s, h_r, M) = (5, 7, 100)$ where h_s represents the spatial kernel bandwidth, h_r is the range kernel bandwidth, and M is the minimum size of superpixels.

Figure 2.1: Superpixels generated from Mean Shift algorithm.

2.1.2 Superpixel Generation With Segmentation-Aware Affinity Loss (SEAL) Using Pixel Affinity Net (PAN)

Tu et al.[18] proposed a deep-learning based superpixel algorithm that leverage the CNN for learning superpixel generation. First, they transform the pixel label transition in the segmentation ground truth into horizontal and vertical binary affinity map as ground truth affinities. Then the Pixel Affinity Net (PAN) model is built and trained to predict two-channel pixel-wise affinity map with horizontal and vertical affinities, see Fig2.2 for examples of predicted affinity maps. The output affinity maps represent the edge weights to construct the image graph. Therefore, the predicted affinities are used in a graph-based superpixel algorithm like Entropy Rate Superpixel[15] to generate superpixels. To further improve the affinity prediction, a task-specific loss called Segmentation-aware Affinity Loss (SEAL) based on binary cross entropy (BCE) loss is proposed to take segmentation errors into account.



(a) Original image. (b) Vertical affinity. (c) Horizontal affinity.

Figure 2.2: Examples of pixel affinity map.

Segmentation-Aware Affinity Loss (SEAL)

Training to predict affinity maps is similar to training edge prediction models where the vanilla binary cross entropy loss defined in 2.1 fails to be effective due to the imbalance of edge and non-edge classes. Hence, in [19], when training a deep-learning based edge detector, Xie et al. proposed a weighted BCE loss to

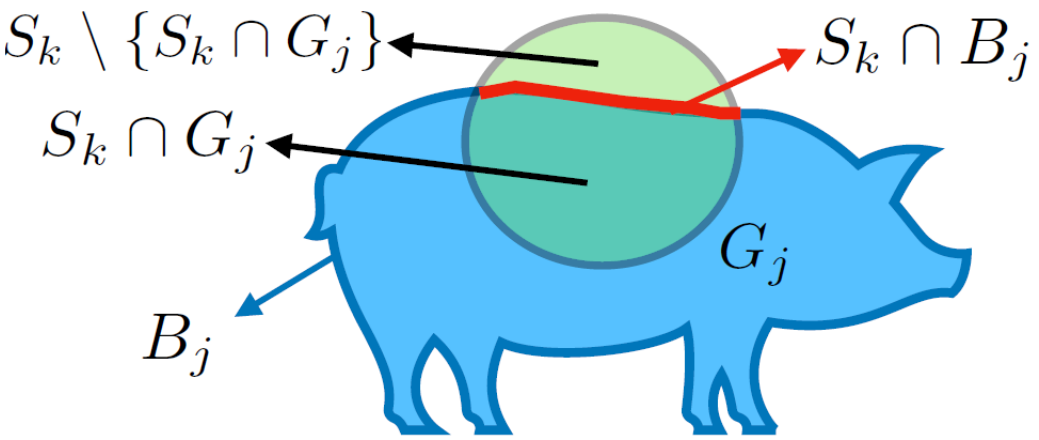


Figure 2.3: Segmentation errors illustration. S_k(green region) is the generated superpixel. G_j(blue) is the groundtruth segment. Pixels that applied SEAL calculation is marked as red line.

overcome the imbalance between two classes.

$$BCE(T, A) = -\sum_i \Big(T_i log(A_i) + (1 - T_i) log(1 - A_i)\Big) \tag{2.1}$$

where A is the output affinity map, and $A_i \in (0, 1)$ denotes the predicted affinity value at i-th pixel in A. T is the ground truth affinity map, and $T_i \in \{0, 1\}$ denotes the ground truth affinity value in T.

By introducing extra weights to BCE loss, a weighted BCE loss can effectively prevent training from favouring one class. The weighted BCE loss is defined as follow:

$$weighted - BCE(T, A) = -\sum_i \Big(w_0(T_i log(A_i) + w_1(1 - T_i) log(1 - A_i)\Big) \tag{2.2}$$

To consider segmentation error as loss, they replaced the weighting constant with segmentation-aware weights with:

$$\omega_{S_k} = |S_k \setminus \{S_k \cap G_j\}| = |S_k| - |S_k \cap G_j| \tag{2.3}$$

As in Fig.2.3, S_k refers to a superpixel generated from a graph-based algorithm, G_j refers to a segment from the groundtruth that superpixel S_k mostly overlapped with, and B_j refers to the object contour with respect to G_j.

If the generated superpixels using the predicted affinity map greatly align with object contours, ω_{S_k} should be small as possible. Otherwise, it means that some incorrectly predicted affinities cause the superpixel algorithm making segmentation errors. Thus, the SEAL function based on modified BCE loss is proposed.

$$SEAL(T, A) = -\sum_{i}(1+\gamma_i)\Big((T_i log(A_i) + (1-T_i) log(1-A_i)\Big) \tag{2.4}$$

where $\gamma_i = \omega_{S_k}$ for $i \in S_k \cap B_j$, otherwise $\gamma_i = 0$. That is, they only applied SEAL to those who are one the segmentation groundtruth boundaries, to invoke stronger loss during training while an over-segmentation occurred.

The Algorithm

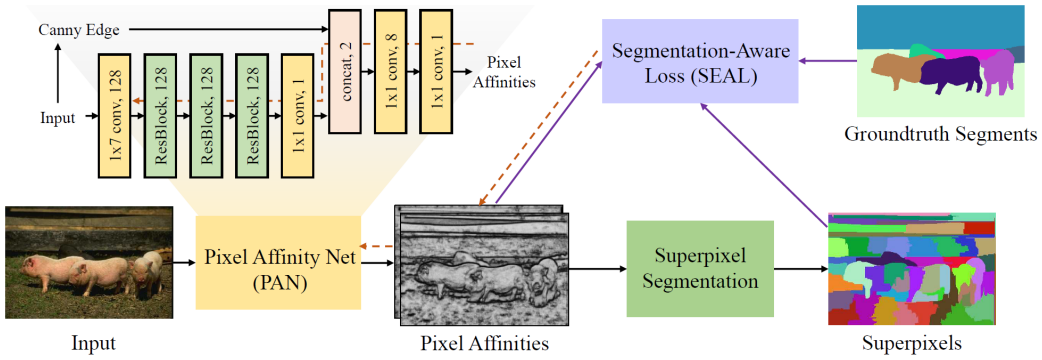


Figure 2.4: Overview of affinity learning framework.

Figure 2.4 shows the overview of the training procedure of the Pixel Affinity Net. The detail training steps are listed below:

Step 1

The input image is preprocessed to form a horizontal and vertical orientation image pair. And then forwarding through the Pixel Affinity Net, in the last two layers, the Canny edge map is join for training.

Step 2

The predicted affinities are passed into a graph-based superpixel algorithm like Entropy Rate Superpixels [15] (in this case) to produce superpxiels.

Step 3

Computing loss for each pixels in generated superpixels using SEAL function in Eq 2.4 with groundtruth segmentation.

Step 4

Back propagate the computed loss through the Pixel Affinity Net model to update parameters.

Superpixel Generation

Following we list the steps on how to produce superpixels using Pixel Affinity Net or SEAL-ERS for short.

Step 1

Preprocessing the input image to form horizontal and vertical image pair. Furthermore, compute the Canny edge map for later inference.

Step 2

Pass the predicted affinities to ERS to produce superpixels.

Simulations

In Figure 2.5 we show some examples of SEAL-ERS with the number of superpixels set to 100.

Figure 2.5: Superpixels generated from SEAL-ERS algorithm.

2.1.3 Superpixel Sampling Network (SSN)

Since conventional superpixel algorithms are not differentiable, it is hard to be integrated in the modern CNN architectures. As we mentioned in the Sec 2.1.2, the ERS algorithm is not differentiable , that is, Tu et al. use deep neural networks to learn pixel affinities as input of ERS algorithm to improve the performance. In [17], Jampani et al. modified the simple linear iterative clustering [16] into a differentiable SLIC. Therefore, they proposed an end-to-end trainable network to produce superpixels. The overview of the SSN is in Figure 2.6.

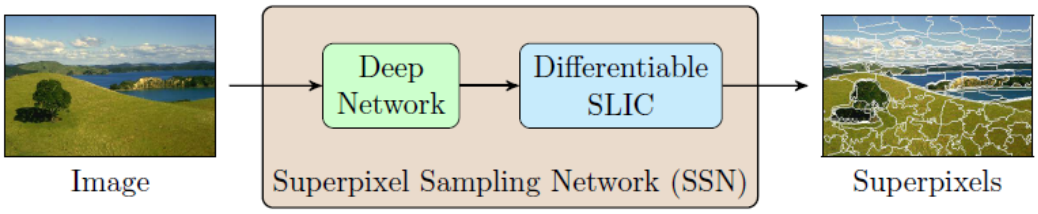


Figure 2.6: Overview of Superpixel Sampling Network.

The input image is first fed into a DCNN to extract k-dimensional features for each pixels, afterwards, the differentiable SLIC algorithm is applied to perform clustering on the predicted feature map.

Differentiable SLIC

Due to non-differentiable properties of traditional superpixel algorithms, it is hard to be integrated into modern end-to-end trainable network architecture. Jampani et al. proposed a differentiable version of SLIC that can be joined to convolutional networks for training.

Unlike original SLIC algorithm that calculate the distances on XYLab feature space, SSN performs clustering over k-dimensional feature space extracted from the CNN for each pixel. Therefore, the entire image with n pixels can now be represented as a feature matrix $\mathbf{F}_{(n\times k)}$.

The center matrix $\mathbf{C}_{(m\times k)}$ is initially distributed over a grid like the original

SLIC, where m is the number of superpixels. The distance from pixel i to the clustering center j is defined as:

$$D_{ij} = \|F_i - C_j\|^2 \tag{2.5}$$

As in original SLIC algorithm, every iteration will recalculate the distances over a specific range, and update the cluster centers. That is, each pixel is **hard** assigned to a cluster center which makes the algorithm non-differentiable. Therefore, instead of hard pixel-cluster association, they proposed soft association matrix $\mathbf{Q}_{(m\times n)}$ between pixels and clusters. The association between pixel i and cluster center j is defined as follows

$$Q_{ij}^T = e^{-D_{ij}} \tag{2.6}$$

Therefore, for each iteration, the center matrix $\mathbf{C}$ is updated using the weighted sums of pixel features as follows

$$C_j = \frac{\sum_{i=1}^{n} Q_{ij}^T F_p}{\sum_{i=1}^{n} Q_{ij}^T} \quad \text{or} \quad C = \hat{Q}^T F \tag{2.7}$$

where i is the pixel in the image, j is the cluster center, n is the number of pixels, and $\hat{Q}$ is the column normalized of Q^T.

Algorithm

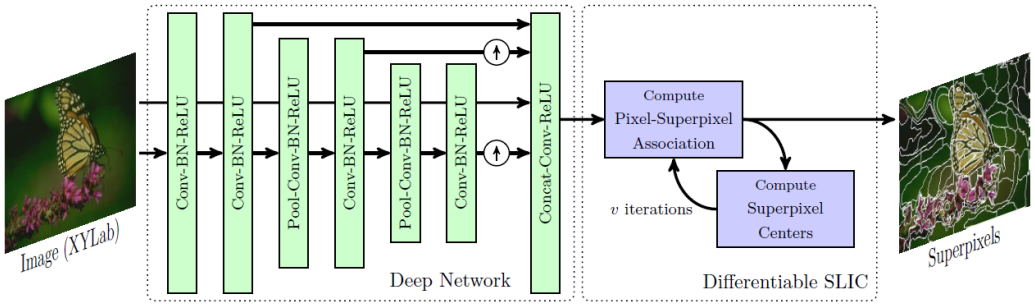


Figure 2.7: Computation flow of SSN.

Fig 2.7 shows the flowchart of superpixel generation in SSN. First, the input image is converted to XYLab feature space denoted as $I_{(n\times 5)}$, where n is the number of pixels, and then passed it into the Superpixel Samping Network to produce the final superpixels. The detailed procedures are defined as follows

Step 1

The input image is passed through a deep network to extract k-dimensional feature for each pixel, denoted by $\mathbf{F}_{(n\times k)}$.

Step 2

Superpixel centers are initialized based on average features in regular grid. The initial center matrix is denoted by $\mathbf{C}^0_{(m\times k)}$

Step 3

Compute the distances between each pixel i and the adjacent superpixel j with Eq 2.5, and construct the association matrix $\mathbf{Q}^T$ using Eq 2.6.

Step 4

Update the center matrix $\mathbf{C}$ with computed $\mathbf{Q}^T$ using Eq 2.7.

Step 5

Repeat Step 3 and Step 4 for v iterations.

Step 5 (Optional)

Hard assign every pixel to a superpixel based on the resulting association matrix $\mathbf{Q}_{(m\times n)}$ with spatial connectivity constraint.

Simulations

In figure 2.8, we show some examples of SSN generated with m as the initial clusters set to 100, and k as the hidden feature dimension of 20, and v as the iteration time set to 10.

Figure 2.8: Superpixels generated from Superpixel Samping Network.

2.2 Classical Segmentation

Most classical segmentation methods utilize hand-crafted features such as color, histogram, gradient, or texture to perform segmentation. Many of them are still widely used today, such as the graph-based method [7] and the normalized cut [6]. Arbelaz et. al. [3, 4] proposed a method based on the global information to perform the oriented watershed transform and generated an ultra-metric contour map for hierarchical segmentation.

Moreover, superpixel-based segmentation algorithms like the method of segmentation by aggregating superpixels (SAS) [8] perform segmentation based on merging superpixels using some local grouping cues. Kim et al.[9] used a full range affinity model and Yang et al.[10] proposed a spectral clustering method based on Gaussian Kernel similarity measure for image segmentation. These superpixel-based segmentation algorithms have good performance. However, due to the irregular shapes and sizes of superpixels, it is hard to embed deep learning techniques in superpixels-based algorithms.

In this section, we will review some state-of-the-art image segmentation algorithms.

2.2.1 Segmentation Using Superpixel (SAS)

In [8], Li et al. proposed an novel graph-based image segmentation algorithm. Since superpixels are groups of pixels which can provide powerful local grouping cues, Li et al. utilized the superpixels to perform image segmentation by treating the superpixels as nodes to be tailored to a bipartite graph structure. Therefore, by varying the parameters with different existing superpixel generation algorithms, multi-range and multi-scale of local information can be acquired by superpixels. Then, multi-layer superpixels are aggregated using grouping rules based on superpixel cues and smoothness cues. Favorable run time can be achieved due to the unbalanced structure of resulting bipartite graph. The overview of the SAS

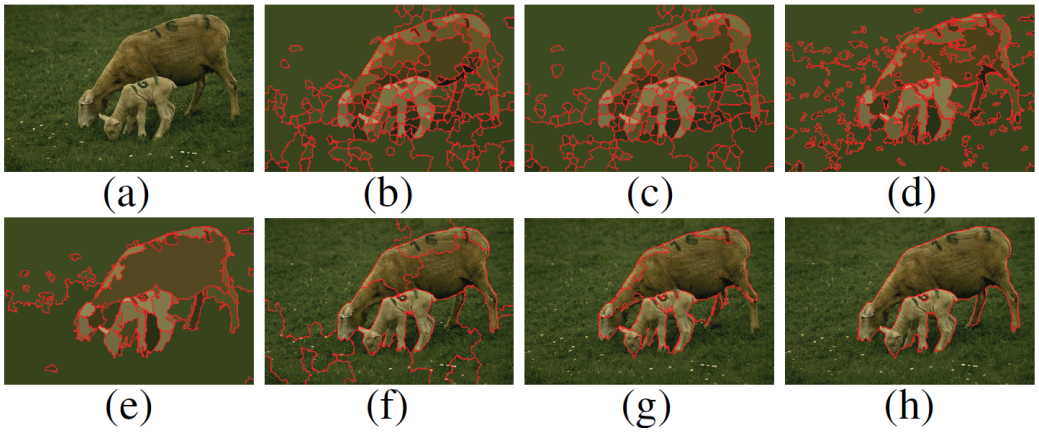


Figure 2.9: **Overview of SAS algorithm.** (a) Original input image. (b-c) over-segmentation (superpixels) results from Mean Shift. (d-e) Over-segmentation (superpixels) results from FH. (f-g) Segmentation results from Mean Shift and FH. (h) Segmentation results from SAS algorithms by aggregating superpixels from (b-e).

algorithm is shown in Fig 2.9.

The Algorithm

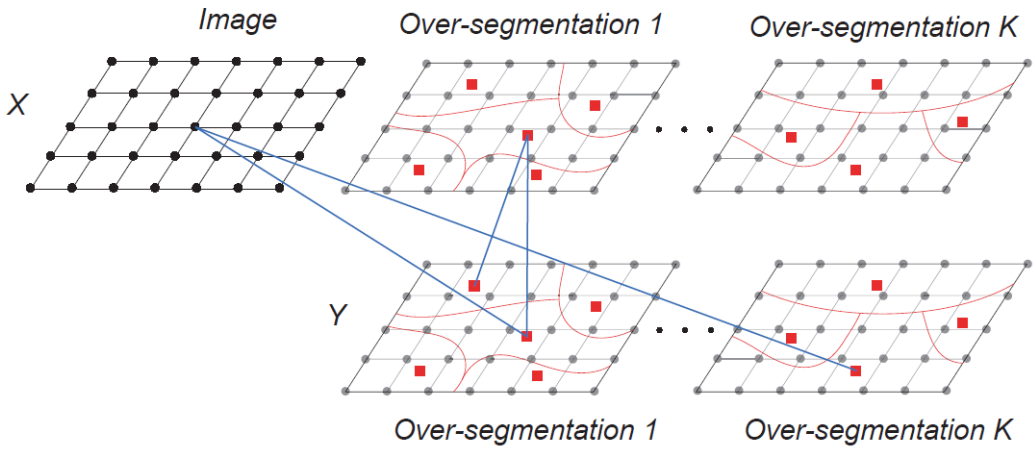


Figure 2.10: **SAS bipartite graph structure.** Formation of SAS bipartite graph with K different over-segmentations. A black dot represents pixel while a red square represents a superpixel within an image.

In fig 2.10 shows how the pixels and superpixels are connected. Mainly, superpixel cues and smoothness cues are used as grouping cues which are defined as follows

1. Superpixel cues: pixels located in the same superpixel are more likely to be grouped.

2. Smoothness cues: two adjacent pixels tend to be grouped if they are close in feature space.

For an image I, we define the desired number of region k, and perform different superpixel generation algorithms to obtain a bag of superpixels S.

Step 1

Construct a bipartite graph $G = \{X, Y, B\}$ where $X = I \cup S = \{x_i\}_{i=1}^{N_X}$, and $Y = S = \{y_j\}_{j=1}^{N_Y}$. with $N_X = |I| + |S|$ denotes the number of nodes in X, and $N_Y = |S|$, denotes the number of nodes in Y. And the across-affinity matrix $B = (b_{ij})_{N_X \times N_Y}$ within node set $X \cup Y$ is constructed as follows

$$b_{ij} = \alpha, \text{if } x_i \in y_j, x_i \in I, y_j \in S; \tag{2.8}$$

$$b_{ij} = e^{-\beta d_{ij}}, \text{if } x_i \sim y_j, x_i \in S, y_j \in S; \tag{2.9}$$

$$b_{ij} = 0, \text{otherwise}, \tag{2.10}$$

where d_{ij} is the l2-norm feature space distance of superpixel x_i and y_j, $\sim$ denotes association of superpixels, and α and β are weighting factors for superpixel and smoothness cues respectively.

Step 2

Perform transfer cuts to partition G into k groups.

Step 2-1

Construct $D_X = diag(B\mathbf{1}), D_Y = diag(B^T\mathbf{1}), W_Y = B^T D_X^{-1} B$, and $L_Y = D_Y - W_Y$.

Step 2-2

Calculate the bottom k eigenpairs $\{(\lambda_i, \mathbf{v}_i)\}_{i=1}^{k}$ of $L_Y\mathbf{v} = \lambda D_Y\mathbf{v}$

Step 2-3

Acquire γ_i which $0 \leq \gamma_i < 1$ and $\gamma_i(2 - \gamma_i) = \lambda_i$ for $i = 1, ..., k$.

Step 2-4

Calculate $\mathbf{f}_i = (\mathbf{u}_i^T, \mathbf{v}_i^T)^T$ with $\mathbf{u}_i = \frac{1}{1-\gamma_i} D_X^{-1} B \mathbf{v}_i$ for $i = 1, ..., k$.

Step 2-5

Obtain k partitions of $X \cup Y$ from $\mathbf{f}_1, ..., \mathbf{f}_k$.

Step 3

Derive the final segmentation by grouping the pixels.

The algorithm steps are referenced from [8].

Simulations

Fig 2.11 shows the examples from SAS algorithm. The parameter k that indicates the number of regions is provided by the authors.

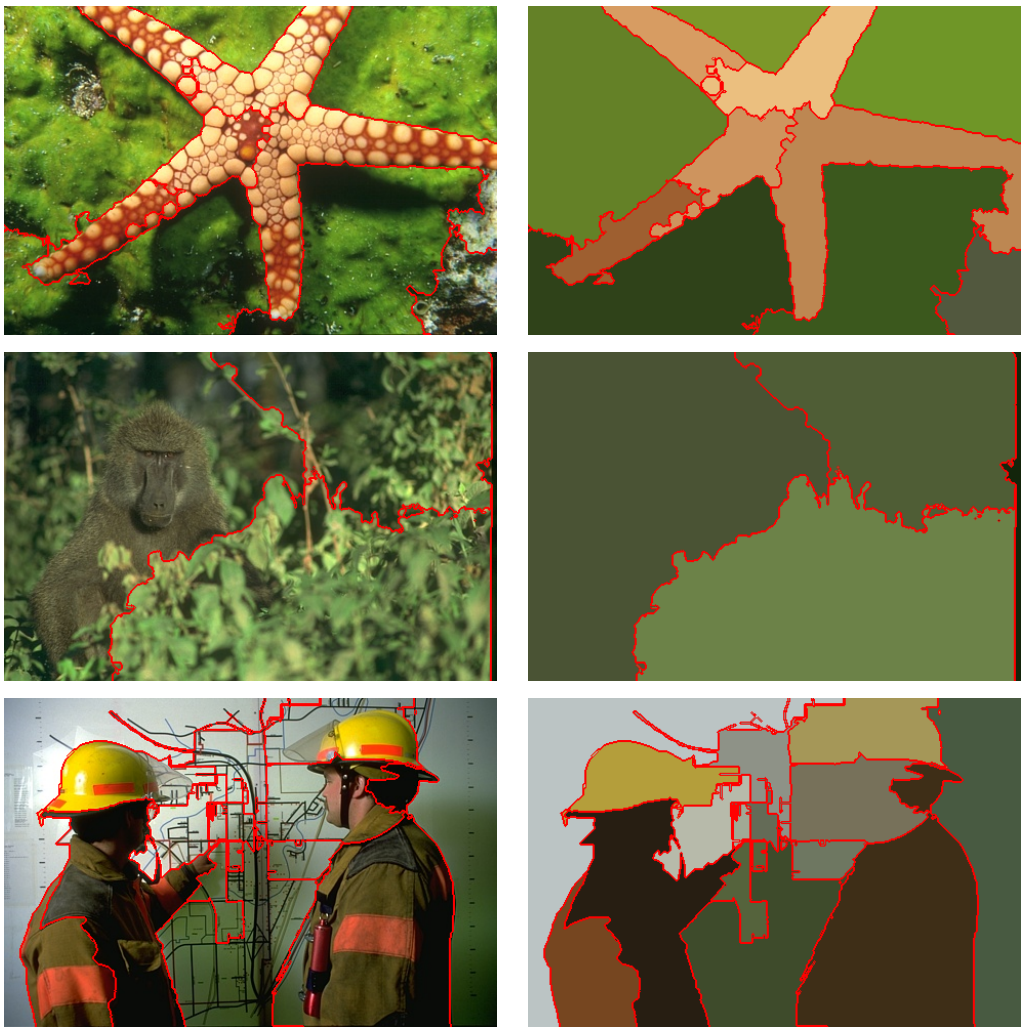


Figure 2.11: Segmentation results from SAS algorithm.

2.2.2 Hierarchical Image Segmentation

In [3, 4], Pablo et al. proposed an effective image segmentation algorithm to produce hierarchical region tree. Different levels of segmentation results can be easily accessed by varying the thresholding value. The algorithm consists of three steps, first, gPb (global Pb), a contour detector that extracts the contour information by computing the probability of each pixel as a boundary. Secondly, followed by Oriented Watershed Transform (OWT), which perform over-segmentation by transforming the gPb results into several closed regions. Last, Ultrametric Contour Map (UCM) converts the resulting region sets into a hierarchical tree.

The Algorithm

In the following section, we will review the steps of gPb-OWT-UCM, which composed of three parts.

Part 1 (global Pb)

Step 1

Convert the input image into 4 different feature channels of brightness, color a, color b (Lab), and texture. The mPb of each pixel at location (x, y) is defined as follows

$$mPb(x, y, \theta) = \sum_{s} \sum_{i} \alpha_{i,s} G_{i,\sigma(i,s)}(x, y, \theta) \tag{2.11}$$

where α is the weighting of different channels, s is the scales, i is the feature channels, $G_{i,\sigma(i,s)}(x, y, \theta)$ is the histogram difference within a circle of radius $\alpha_{i,s}$ divided at angle θ.

Step 2

Compute sPb by applying the Gaussian directional derivative filters on the eigenvectors of affinity matrix W_{ij} represents the maximum weight of two pixel connected as a boundary.

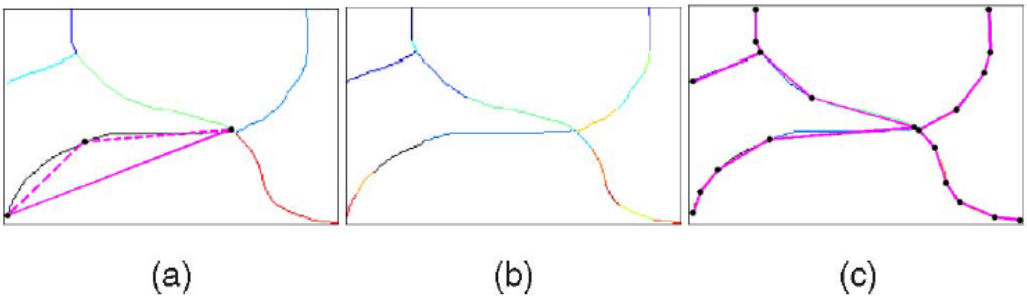


Figure 2.12: Arc subdivision. (a) Original arcs. (b) Final arcs after recursively update the subdivisions. (c) The resulting subdivision of line segments.

$$W_{ij} = exp(-max_{p \in ij}\{mPb(p)/\rho\}) \tag{2.12}$$

where i, j index the pixels whose distance between them is within radius (r = 5 pixels), p is on the boundary connected by two pixels. ρ is a constant.

Step 3

Calculate gPb by adding mPb and sPb.

$$gPb(x, y, \theta) = \sum_s \sum_i \alpha_{i,s} G_{i,\sigma(i,s)}(x, y, \theta) + \gamma \cdot sPb(x, y, \theta) \tag{2.13}$$

where γ is the weighting of sPb.

Part 2 (Oriented Watershed Transform)

Calculate $E(x, y) = max_\theta E(x, y, \theta)$ using the gPb formula 2.13 with 8 different angles. Then for every arc in E, recursively subdivide the arc into several line segments as in Fig 2.12, calculating the orientation of the line $o(x, y)$, and update $E(x, y)$ by $E(x, y, o(x, y))$.

Part 3 (Ultrametric Contour Map)

The finest segmentation is obtained by OWT, the authors defined the dissimilarity measure across each regions. They constructed a graph by treating regions as nodes connected by the same arc. Therefore, the final hierarchical region tree is build in the order of dissimilarity measure.

Simulations

In Fig 2.13, we show some gPb-OWT-UCM results thresholding at ODS (Optimal Dataset Scale) and OIS (Optimal Image Scale) respectively.

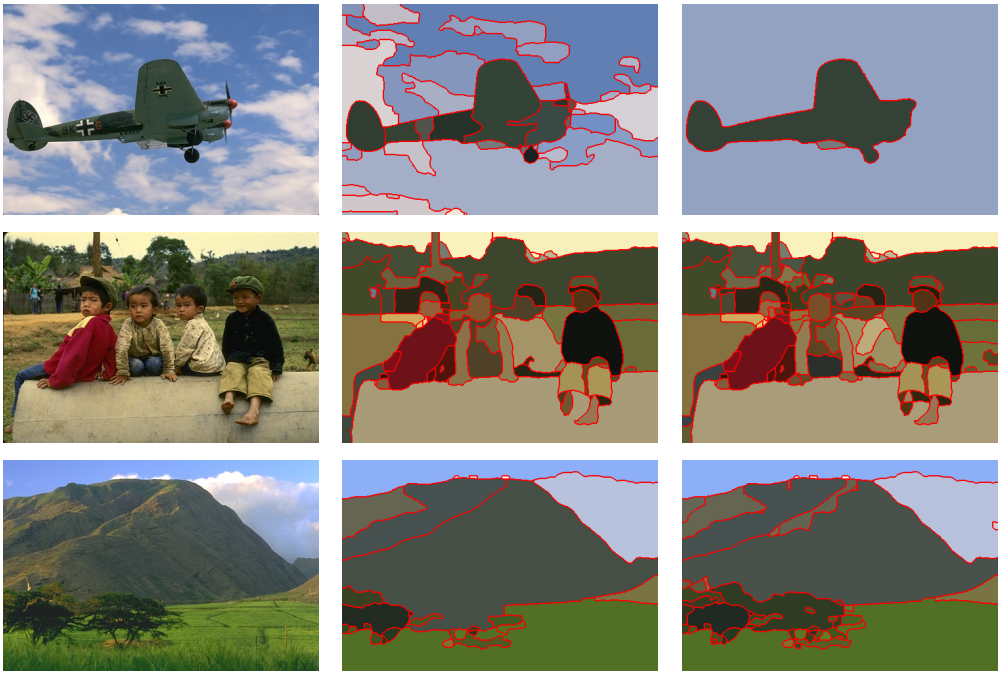


Figure 2.13: Results from gPb-OWT-UCM algorithm. Left: Original images. Middle: Results thresholding at ODS. Right: Results thresholding at OIS.

2.3 Deep Learning in Image Segmentation

Recently, many semantic segmentation algorithms applied the deep neural network were developed. In [12, 13], the Fully Convolutional Networks (FCN) were proposed to improve the performance of image segmentation and object detection. In [14], the conditional random field (CRF) was applied in the pixel-wise segmentation method of DeepLab. In [11], Xia and Kulis proposed the W-Net based on the FCN to perform segmentation. In [20], Haeh et. al. introduced a method based on the CNN to detect split error in segmented biomedical images.

2.3.1 Fully Convolutional Networks (FCN)

The appearance of FCN was first used in semantic segmentation task. The difference between FCN and the CNN is that the fully connected layers in CNN are all replaced with convolutional layers. In other words, FCN do not use fully connected layers to perform classification. The CNN has its strength at learning high-level features to describe an object. However, such high-level feature is usually insensitive to spatial information or local details. Therefore, as for semantic image segmentation task, which requires pixel-wise accuracy over the whole image, integrating the CNN in semantic segmentation will lose spatial details and being inefficient.

In [13], Long et al. proposed FCN to be implemented in semantic segmentation. With the use of upsampling layers, accurate pixel-wise prediction can be achieved without losing many spatial information. Moreover, without the fully connected layers to perform classification on each pixel, FCN are highly efficient since convolutional layers are in charge of classify and extracting features at the same time.

See Fig. 2.14 for the convolutionalization of the CNN.

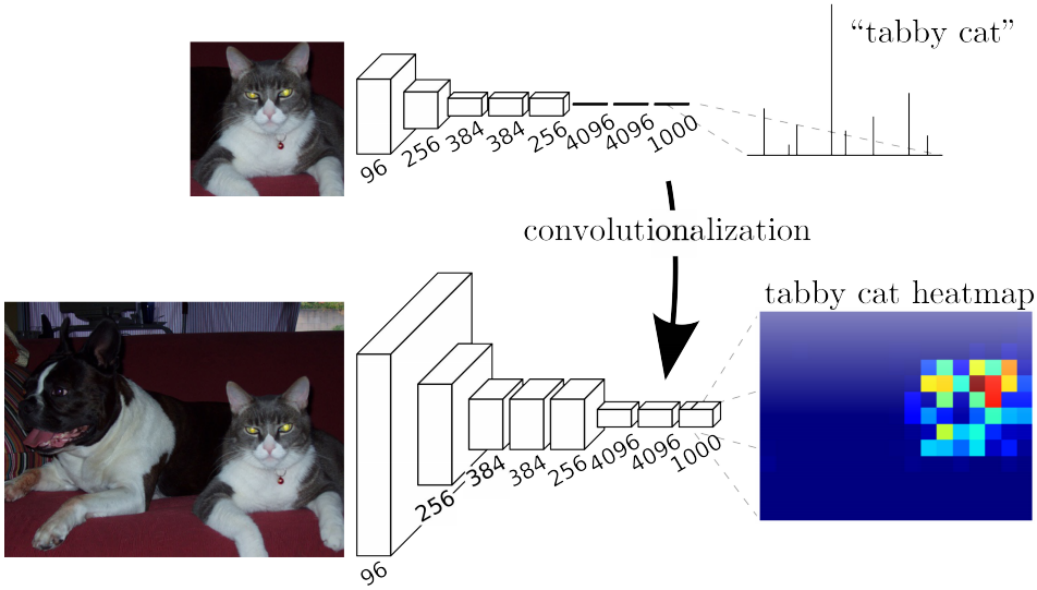


Figure 2.14: **Convolutionalization of CNN.** By replacing the fully connected layers in the CNN with convolutional layers, making the CNN into FCN, an end-to-end trainable networks for pixel-wise learning.

Chapter 3

Proposed Algorithms: DMMSS

In this chapter, we illustrate the architecture of the proposed algorithm in detail. It consists of four parts: (i) two-superpixel patch generation, (ii) the training architecture, (iii) superpixel pairing, and (iv) the merging procedure.

Different from other learning-based methods, which take the whole image as the input and output the segmentation result directly, in the proposed algorithm, the input and the output of the deep neural network are

- Input: A patch containing only two adjacent superpixels, as in Fig. 3.2.
- Output: A label to indicate whether the two superpixels should be merged.

That is, we break down the image segmentation problem into several parts and deep learning is applied to decide whether two superpixels should be merged.

With the above strategy, we convert an image segmentation problem into a binary classification problem. Moreover, since every image contains many superpixel pairs, a huge amount of training data can be acquired.

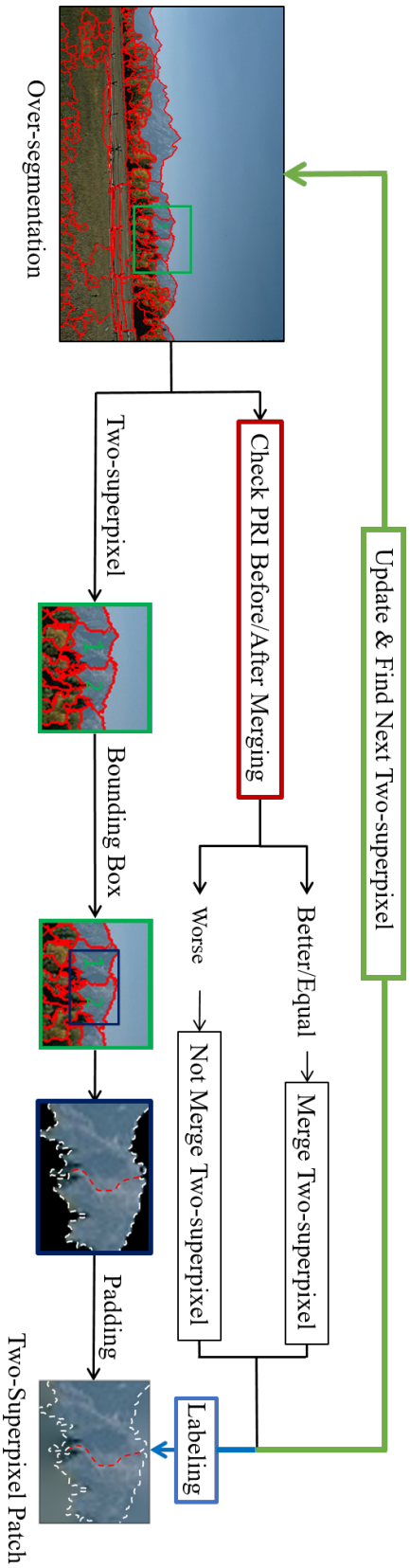


Figure 3.1: **Flowchart of generating two-superpixel patches.**

3.1 Two-Superpixel Patch Generation

To ensure the robustness of the deep learning model, we obey the following two rules when extracting the two-superpixel patch:

1. The patch should contain only two adjacent superpixels.

2. If the two superpixels do not fill the whole patch, as in the bottom right part of Fig. 3.1, the image inpainting technique is applied to pad the blank region.

Fig. 3.1 shows the flowchart of the two-superpixel patch extraction process and some examples of the extracted two-superpixel patches are shown in Fig. 3.3.

First, we apply superpixel generation algorithms to acquire the initial over-segmented image. In the training set, to increase the diversity of two-superpixel patches, three different algorithms for superpixel generation are applied, including the mean-shift algorithm [2] and deep-learning-based superpixel algorithms like the SEAL-ERS [18] and the SSN [17]. Furthermore, by varying the numbers of superpixel in SEAL-ERS and SSN, one can obtain multi-scales of superpixel patches.

In the labeling process of the training phase, the metric the probabilistic rand index (PRI)[3] is adopted to determine whether two superpixels should be merged. If the merging makes the PRI higher or unchanged, then we consider that merging these two superpixels is appropriate and label the corresponding two-superpixel patch as “to be merged”. Whereas the PRI drops after merging, we label the corresponding two-superpixel patch as “not to be merged”. In other words, whether two superpixels should be merged is treated as a label and the label in the ground truth is generated according to whether the PRI increases or decreases after merging the superpixel pair.

Note that, each two-superpixel patch will be treated as a training data. Since there are many superpixel pairs within an image, a huge amount of training data can be acquired even if there are limited number of training images.

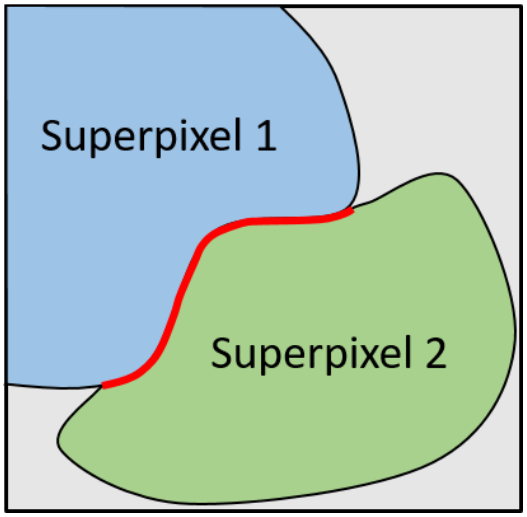


Figure 3.2: Example of a two-superpixel patch. The blue and green regions represent two adjacent superpixels, respectively, and the boundary between them is marked by a red curve. The gray region represents the area outside the two superpixels.

Then, two-superpixel patches are trimmed to follow the two rules defined at the beginning of Section 3.1. First, after applying a bounding box to capture two adjacent superpixels, the anchor point, the width, and the length of the bounding box are recorded. As mentioned in the first rule, the bounding box should cover only two superpixels. Therefore, we choose the middle point on the boundary between the two superpixels as the anchor point.

Then, we find the centroids of two superpixels and calculate the Euclidean distances between the two centroids and the anchor point. The smaller distance is denoted by d. Since better performance can be achieved if the areas of the two superpixels are roughly equal, we set the width and length of the bounding box as $2d$. Note that the smaller distance to the anchor point means that the superpixel is smaller than the other one. This method can ensure that the part of the larger superpixel within the bounding box has almost the same area as that of the smaller superpixel. After generating the bounding box, it is inevitable that there are some pixels within the bounding box that do not belong to the two

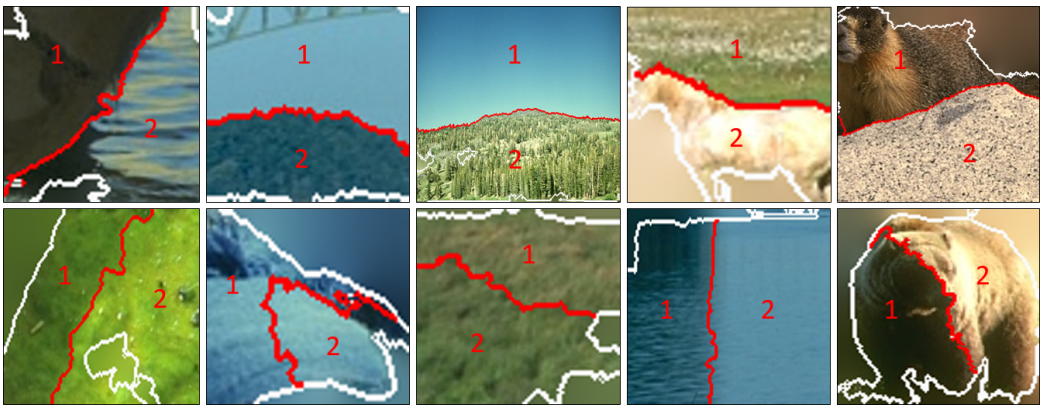


Figure 3.3: **Examples of extracted two-superpixel patches.** The upper row are the two-superpixel patches labeled "not to be merged" and the lower row are the two-superpixel patches labeled "to be merged". The red curves are the boundaries between two superpixels and the white lines are the borders of superpixels.

superpixels. To perform blank space padding, the naive solutions like padding with zeros or padding with the mean value will cause discontinuities and artifact edges. Moreover, the padded zeros will be confused with the black color. In this thesis, we apply the technique called inpainting [21] to fill the blank regions by solving the following Laplace equation with two independent variables:

$$\frac{\partial^2 \Phi}{\partial x^2} = \frac{\partial^2 \Phi}{\partial y^2} \tag{3.1}$$

Given a region R, the Dirichlet problem tries to find the solution where the harmonic function Φ satisfies the boundary conditions. In other words, Φ is dominated by the boundaries. In digital images, one can approximate the 2^{nd} order partial differentiations in (3.1) by the following central difference operations:

$$\frac{\partial^2 u}{\partial x_{i,j}^2} \approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{(\Delta x)^2} \tag{3.2}$$

$$\frac{\partial^2 u}{\partial y_{i,j}^2} \approx \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{(\Delta y)^2} \tag{3.3}$$

Since in images, pixels are discrete, $\Delta x = \Delta y = 1$. Hence, the discretized form of the Laplace equation can be rearranged as follows:

$$-4u_{i,j} + u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} = 0 \tag{3.4}$$

Figure 3.4: Padding the blank region of the two-superpixel patch using different techniques.

For pixel (i, j) in an image of each channel, $u_{i,j}$ represents the color intensity of the pixel. The color intensities of the pixels surrounding the blank space are treated as the Dirichlet boundary condition. After performing inpainting by (3.2)-(3.4) together with the Dirichlet boundary condition, the blank regions are filled with the values come from the original superpixels. Furthermore, this will not generate artifact edges around the borders of the blank spaces. See Fig. 3.4 for the effects of different padding techniques.

With all the procedures described in this section, 710,000 two-superpixel patches with labels can be extracted from 300 training and validation color images in the BSDS500 dataset. Examples of the two-superpixel patches after padding are shown in Fig. 3.3.

3.2 Training Architecture

To achieve an even better segmentation result, we developed a two-stage superpixel merging process. We train two deep models with different amount of data. One applies balanced labeled data, that is, the numbers of training two-superpixel patches in two classes are roughly the same. The other one uses unbalanced labeled data and the two-superpixel patches labeled by “to be merged” is few times more than those labeled by “not to be merged”. As a result, the first model performs merging cautiously to avoid over-merging. Then, the second model is adopted to obtain the final segmentation results.

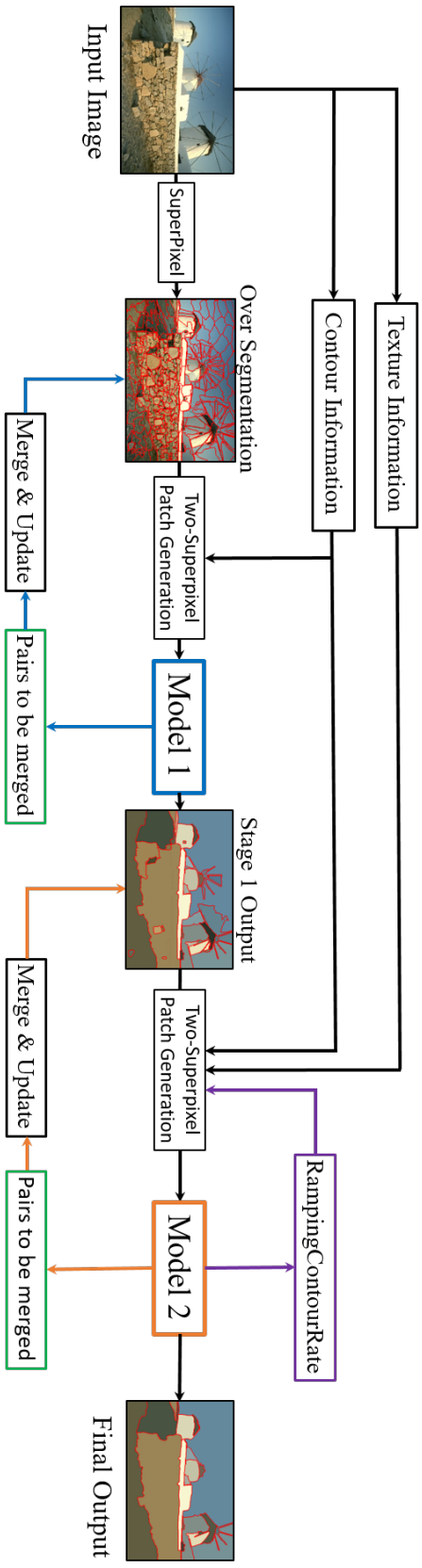


Figure 3.5: **Flowchart of the learning-based merging procedure.**

3.3 Superpixel Pairing

To ensure that the segmentation result is edge-preserving, we apply some criteria to sift the superpixel pair that are impossible to be merged. These criteria are described as follows.

The $ContourRate$ is to indicate the percentage of pixels on the boundary of two adjacent superpixels that have high responses for edge detection. If the $ContourRate$ is high, it means that the boundary of the two adjacent superpixels may be the edge of some object and one should avoid merging these two superpixels.

We apply the RefineContourNet(RCN) algorithm proposed by Kelm et al. [22] to generate the edge map of the input image and threshold it to get a binary contour map. Then, the $ContourRate$ is defined as:

$$ContourRate(i,j) = \frac{\#\text{ of pixels of (long contours} \cap Bnd(i,j))}{\#\text{ of pixels of } Bnd(i,j)} \tag{3.5}$$

where long contours are the contour on the edge map with contour length larger than certain threshold and the $Bnd(i,j)$ is the boundary between two adjacent superpixels i and j.

Moreover, in image segmentation, texture is one of the commonly used feature to compare the similarity between two regions. Smaller texture difference means that two regions are similar. Therefore, we use the texture difference as one of the criteria for superpixel pair sifting. The Log-Gabor filter proposed by [23] is widely used to extract texture features.

$$G(f_x, f_y) = \exp\left(- \frac{\Big(\log\Big((f_x \cos\phi + f_y \sin\phi)/f_0\Big)\Big)^2}{2\Big(log(\sigma/f_0)\Big)^2} \right) \tag{3.6}$$

Usually, two scales and four orientations are used to extract total of 8 texture images by changing the values of σ and ϕ. The difference of the texture is determined from:

$$dTex(i,j) = \sqrt{\sum_{k=1}^{8} \Big(T_k(i) - T_k(j)\Big)^2} \tag{3.7}$$

where $T_k(i)$ and $T_k(j)$ are the mean texture of adjacent superpixels i and j, respectively, and k denotes the k^th texture feature.

To avoid merging two superpixels that are connected by only a few pixels, we introduced the $ContactRate$.

$$ContactRate(i,j) = \frac{\#\text{ of pixels of } Bnd(i,j)}{\min\Big(BL(i), BL(j)\Big)} \tag{3.8}$$

where $BL(i)$ means the perimeter of superpixel i.

Moreover, traditional features like the area of regions and the color difference are also adopted. In the first stage, the criteria of the $ContourRate$, the $ContactRate$, and the area are applied with fixed thresholds, whereas in the second stage the $ContourRate$, the $dTex$ in (3.7) , the $ContactRate$, the $area$, and the $color\ difference$ are applied with adaptive thresholding.

3.4 Merging Procedure

After training the superpixel merging model, image segmentation is performed. As described in Section 3.2, we apply a two-stage merging procedure. The initial superpixels are fed into Model 1. It aims to merge the adjacent superpixel pairs selected by the criteria in Section 3.3. Then, the output is fed into Model 2. It performs superpixel merging using adaptive criteria. The overview of merging procedure is shown in Fig. 3.5.

First, each superpixel will be compared to all its adjacent superpixels. Once there is an adjacent superpixel pair satisfying the criteria in Section 3.3, we utilize the techniques in Section 3.1 to generate a two-superpixel patch and then pass it to Model 1 to decide whether the two adjacent superpixels should be merged. After examining all neighboring superpixels, the system continues to process the next superpixel. The iteration is stopped until all the remaining superpixels cannot be merged. As for now, the first-stage merging process is finished.

Next, the output of Model 1 is fed into Model 2. In this stage, instead of using fixed threshold criteria for superpixel pairing, we apply the adaptive criterion to

the $ContourRate$ called the $RampingContourRate$. That is, the initial threshold for the $ContourRate$ starts from a low value. It is increased bit-by-bit every time. The process continues until no two-superpixel pair can be further merged.

The reason why we apply the RampingContourRate in the second merging stage is that the first merging stage aims to merge most of the small superpixels. In other words, about $60\% \sim 80\%$ superpixels will be merged during this process. It is inefficient to apply the RampingContourRate in the first stage since much processing time is required due to the huge amount of superpixels. After the first merging stage, the number of superpixels is much reduced and the size of superpixel is increased. Then, the $RampingContourRate$ criterion is adopted to guarantee that the superpixels with small areas and small edge responses are easier to merged. By first merging all the small superpixels, one can prevent two large superpixels that are connected by a small superpixel from improper merging.

The second merging stage repeats until the $RampingContourRate$ criterion reaches the final threshold and no further adjacent superpixel pair can be merged. Then, the final segmentation result is obtained and the whole image segmentation process is completed.

We call the proposed image segmentation algorithm the *Deep Merging Model for Superpixel-based Segmentation* (DMMSS).

3.5 Experiments

We evaluate the proposed DMMSS algorithm on the popular Berkeley Segmentation DataSet 500, which consists of 500 color images. It is split into 200 test images, 200 training images, and 100 validation images. We extracted the two-superpixel patches on the 300 training and validation images and used them for training the network.

We mainly applied on the superpixels generated by the SSN [17]. Additionally, we also reported the segmentation results on the BSDS500 dataset using the mean

shift (MS) superpixels [2] with $(h_s = 5, h_r = 7, M = 100)$ and another deep-learning based superpixels SEAL-ERS [18]. The number of superpixels was set to 300 for both SSN and SEAL-ERS.

In the proposed architecture, the deep learning model of the ResNet101[24] is adopted with the last fully-connected layer replaced by a layer of two possible outputs, and the input image size is set to $64 \times 64 \times 3$.

We used a pre-trained model on the ImageNet and fine-tuned the networks using mini-batches of 700 images with the initial learning rate of 0.0001. The learning rate was divided by 2 every 10 epochs and the training process stopped after 80 epochs. The binary cross entropy loss was used as the objective function and the Adam optimizer was adopted. A dropout layer with probability 0.5 was added to the networks to prevent overfitting during the training phase. Moreover, we adopted the early-stopping technique with the validation patience set to 400 iterations.

3.5.1 Segmentation Evaluation

To compare the proposed *DMMSS* algorithm to the existing methods, we evaluate the performance on the standard metrics of segmentation covering (SC), the probabilistic rand index (PRI), and the variation of information (VI) [3]. A higher SC and PRI and a lower VI means better performance.

We compare the proposed *DMMSS* algorithm to the state-of-the-art methods, including the W-Net [11], gPb-OWT-UCM [4], DC-Seg-full [25], Taylor [26], Felzenszwalb and Huttenlocher (Felz-Hutt) [7], Mean Shift [2], Canny-OWT-UCM [4], Multiscale Normalized Cuts (NCuts) [5], fPb-OWT-UCM [9], and cPb-OWT-UCM [9]. As the proposed algorithm, all the algorithms compared in Table 3.2 have not to assign the number of regions in prior.

Table 3.2 shows the performance of the proposed *DMMSS* approach on the BSDS500 dataset. It can be seen that the performance of our proposed method is much better than that of state-of-the-art algorithms. One of the significant

advantages of the proposed *DMMSS* algorithm is that it does not require lots of annotated training images to train a fully functional deep model. Compared to other deep-learning based methods like the W-Net, which was trained on the PASCAL VOC2012 dataset [27] that contains 11,530 images and 6,929 segmentations, better results can be achieved by the proposed *DMMSS* algorithm with almost 35 times fewer training images.

3.5.2 Ablation Study

In the following section, we carried out lots of experiments to analyze the functionality of every parts of our proposed *DMMSS* algorithm. Including the importance of heuristic rules we adopted in the previous section, the order of forwarding input patches, the necessity of two models, superpixel variation, contour map, and CNN architectures.

Heuristic Rules and Models

In this subsection, we conduct some experiments to see the influence of deep models (Model 1, Model 2) and criteria we defined earlier in the proposed algorithm. In Table 3.1, first, we show the results that the algorithm is modified by removing the deep models from it and let the algorithm using the criteria in Section 3.3 to merge superpixels. Later on, we show the results that the criteria in Section 3.3 are removed from the algorithm and the deep models (Model 1 and Model 2) are adopted. In the last row, the results that Model 1, Model 2, and the criteria in Section 3.3 are all adopted are shown.

Table 3.1 shows that the deep models can still maintain good performance without the guidance of rules. Even though the criteria in Section 3.3 performs badly alone, however, it is helpful for improving the performance of the deep models due to that the superpixel pairs that have highly edge over-lapped boundaries are prevented from merging.

Table 3.1: **Ablation study on heuristic rules and models.**

Setting	**VI**	**PRI**	**SC**
w/o models, w rules	1.77	0.74	0.58
w models, w/o rules	1.57	0.82	0.62
w models, w rules	**1.46**	**0.86**	**0.63**

Model 1 and Model 2

In the proposed *DMMSS* algorithm, we design two different deep models to perform superpixel merging. Therefore, we look into the necessity of keeping two deep models. We modified the algorithm into several versions that can perform merging procedure using only Model 1 or Model 2,including one version that exchanges the position of two deep models.

The results of ablation study on models can be found in Table 3.3, therefore, we can find that each model is designed to deal with different kinds of merging situation on purpose. By observing the difference in SC and VI scores, higher SC and lower VI scores indicate that the segmentation is more compact. Hence, Model 1 aims to merge initial segments with caution to prevent future region leakages which will result in a lower SC and higher VI scores. After Model 1 forms a preliminary segmentation, Model 2 then further join those regions that are hard to be merged in many state-of-the-art algorithms by seeing more information contained in the two-superpixel patches since each region is larger after stage 1. As a result, not only these two models are needed in this algorithm, the order of placing each model is also crucial during merging procedure.

In Fig. 3.8, visual simulation results from different stages of the proposed algorithm are presented. During merging, each model is specifically designed to tackle with different situation. Therefore, we can observe the difference of segmentations between stages.

Table 3.2: **Results on the BSDS500 dataset.**

Method	**VI**	**PRI**	**SC**
Ncuts	2.23	0.78	0.45
Canny-OWT-UCM	2.19	0.79	0.49
Felz-Hutt	2.21	0.80	0.52
Mean Shift	1.85	0.79	0.54
Taylor	1.78	0.81	0.56
W-Net	1.76	0.81	0.57
fPb-OWT-UCM	1.70	0.82	0.58
DC-Seg-full	1.68	0.82	0.59
W-Net+UCM	1.67	0.82	0.59
gPb-OWT-UCM	1.69	0.83	0.59
cPb-OWT-UCM	1.65	0.83	0.59
DMMSS(MS)	**1.51**	**0.85**	**0.63**
DMMSS(SEAL)	**1.49**	**0.85**	**0.62**
DMMSS(SSN)	**1.46**	**0.86**	**0.63**
Human Drawing	1.17	0.88	0.72

ContourMap

In the proposed architecture, the RCN [22] is applied as the contour map, which plays an important role in the superpixel pairing procedure of Section 3.3. The contour map highly affects the performance of image segmentation because vanishing of the contour might cripple the criterion, causing regions to be join undesirably. In this subsection, we further study the impact of different contour/edge detection algorithms on the proposed algorithm. In the left part of Table 3.4, we reported the results that utilizing the classical contour detection of UCM [3, 4] and the structure edge detector [28] instead of the RCN for contour map generation. As one can see, better segmentation results can be achieved if the RCN is adopted for contour map

Table 3.3: **Ablation results on Model 1 and Model 2.**

Stage 1	**Stage 2**	**VI**	**PRI**	**SC**
Model 1	Model 1	1.7338	0.8515	0.5587
Model 2	Model 2	1.5260	0.8457	0.6196
Model 2	Model 1	1.5570	0.8508	0.6104
Model 1	Model 2	**1.4634**	**0.8597**	**0.6275**

Table 3.4: **(Left) Results of different contour/edge detection. (Right) Results of different depth in the CNN.**

Detection Method	**VI**	**PRI**	**SC**	**Depth of CNN**	**VI**	**PRI**	**SC**
Structure Edge	1.58	0.85	0.61	ResNet18	1.72	0.84	0.58
UCM	1.56	0.84	0.62	ResNet50	1.71	0.84	0.57
RCN	**1.46**	**0.86**	**0.63**	ResNet101	**1.46**	**0.86**	**0.63**

generation.

CNN Architecture

We mainly implemented the proposed *DMMSS* algorithm using the ResNet [24] architecture. There are various versions of the ResNet with different numbers of layers within the networks. Therefore, we tested the proposed algorithms on the ResNet18, the ResNet50, the ResNet101 and showed the results in the right part of Table 3.4. The results show that using ResNet101 can achieve the best performance.

Superpixel

In Table 3.2, three different kinds of superpixels are adopted (SEAL-ERS, MS, and SSN superpixels). Two of them (SEAL-ERS and SSN superpixels) are deep learning-based and adjustable to the number of superpixels. In Fig. 3.6, we

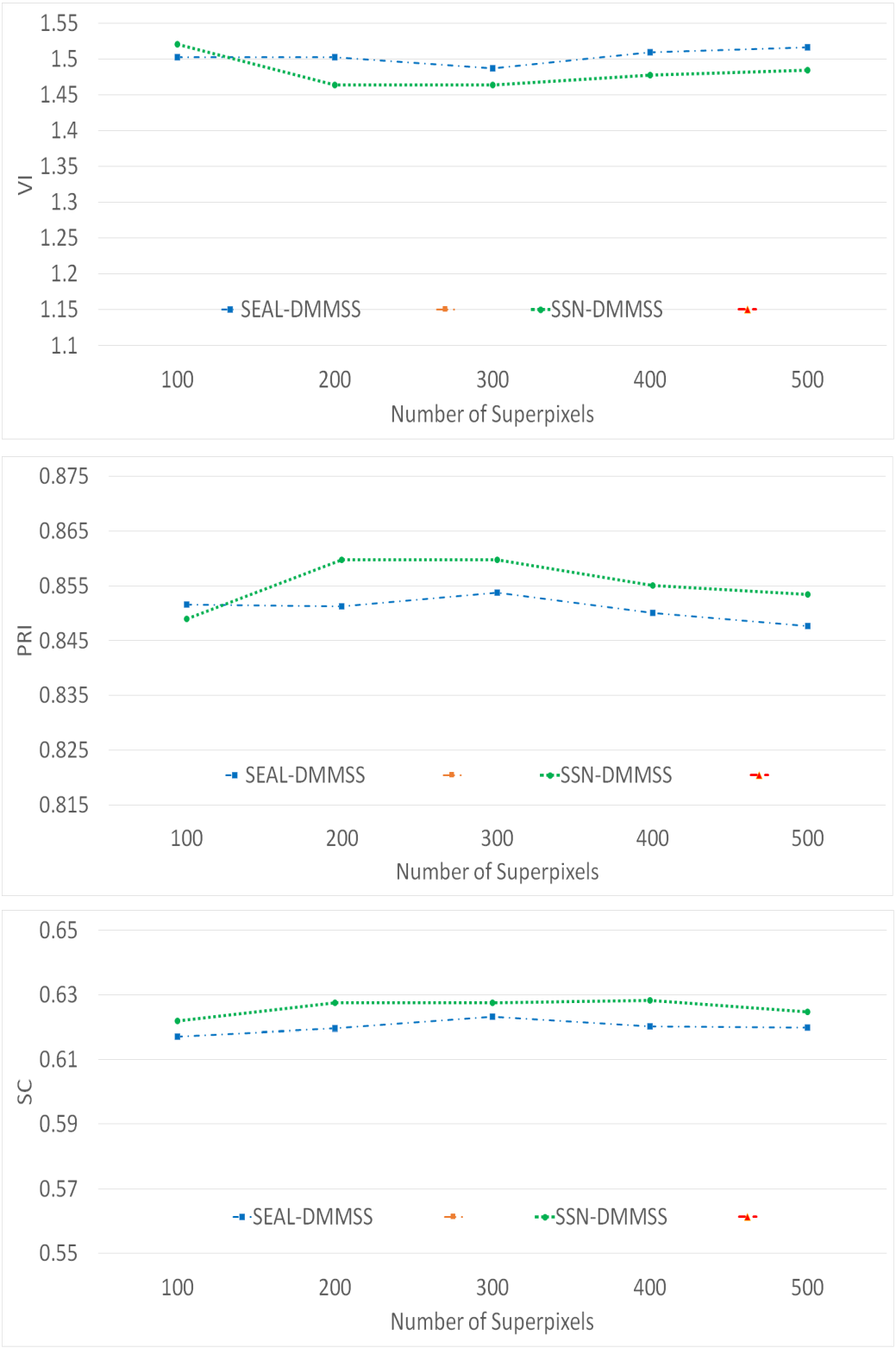


Figure 3.6: **Performance of different numbers of superpixel between SSN and SEAL-ERS.**

measured the performance difference by varying the number of initial superpixels. From Fig. 3.6, one can see that the performance slightly varies with the number of initial superpixels and the type of superpixels. Since our training two-superpixel patches were collected from the cases where the initial number of superpixels set to 300, it makes sense that the proposed algorithm reaches the best performance around 300. Moreover, the results in Table 3.2 and Fig. 3.7 show that, with the proposed *DMMSS* algorithm, good segmentation results can be achieved no matter which type of superpixels is adopted.

In this thesis, we trained our deep models using two-superpixel patches collected from three different superpixel generation algorithms including Mean-Shift(MS)[2], Superpixel Sampling Network (SSN)[17], and Segmentation-Aware Loss (SEAL-ERS)[18]. Therefore, experiments have been carried out to find out how different two-superpixel training patches could affect our proposed algorithm.

Table 3.5 shows the results using different types of superpixel algorithm to generate two-superpixel patches as training data, then testing on different superpixel patch types. As can be seen, no matter what type of superpixel algorithm that two-superpixel patches is collected from, our proposed *DMMSS* algorithm manages to produce highly accurate image segmentation results over different superpixel generation algorithms.

In Fig. 3.7, we present the simulations that apply the proposed *DMMSS* algorithm to merge the superpixels generated from different algorithms. The results show that, no matter which type of superpixels is applied, with the proposed *DMMSS* algorithm, very high-quality segmentation results can be achieved.

Table 3.5: **Influence of different two-superpixel training patches.**

Training Patch Type	**Testing Patch Type**	**VI**	**PRI**	**SC**
Mean Shift	Mean Shift	**1.5499**	**0.8435**	**0.6143**
	SEAL(num=100)	1.5768	0.8369	0.5954
	SEAL(num=200)	1.5735	0.8391	0.6005
	SSN (num=100)	1.5926	0.8377	0.5962
	SSN (num=200)	1.5773	0.8381	0.6041
SSN	Mean Shift	1.5627	0.8498	0.6067
	SEAL(num=100)	1.5868	0.8439	0.5885
	SEAL(num=200)	1.5567	0.8481	0.6049
	SSN (num=100)	1.5312	0.8501	**0.6140**
	SSN (num=200)	**1.5090**	**0.8525**	0.6125
SEAL	Mean Shift	1.5302	0.8498	0.6181
	SEAL (num=100)	1.5507	0.8498	0.5974
	SEAL (num=200)	1.5250	0.8505	0.6110
	SSN (num=100)	1.5103	0.8511	0.6188
	SSN (num=200)	**1.4891**	**0.8542**	**0.6206**
Mean Shift, SSN, SEAL	Mean Shift	1.5148	0.8544	0.6255
	SEAL (num=100)	1.5025	0.8516	0.6171
	SEAL (num=200)	1.5021	0.8513	0.6196
	SSN (num=100)	1.5207	0.8490	0.6219
	SSN (num=200)	**1.4634**	**0.8597**	**0.6275**

Figure 3.7: Results generated by different superpixel type as input prior.(1^{st} row): original images. Results produced by the proposed *DMMSS* algorithm using superpixels generated by (2^{nd} row): Mean-Shift; (3^{rd} row): SEAL-ERS; (4^{th} row): Superpixel Sampling Network (SSN).

Figure 3.8: Results from different stages of proposed *DMMSS*(1^{st} row): original images. (2^{nd} row): segmented into superpixels by SSN; (3^{rd} row): outputs of stage 1; (4^{th} row): final segmentation result of the proposed DMMSS algorithm; (5^{th} row) : segmentation result with boundaries of superpixels marked as red lines. As one can see, in stage 1, most superpixels belonging to the same region are merge. In stage 2, the model aims to merge the superpixels that are difficult to be merged in stage 1.

Chapter 4

Proposed Algorithms: DMMSS-FCN

In this chapter, we proposed a different architecture for generic image segmentation from previous *DMMSS*. Inspired by *DMMSS*, we take it to the next level by combining Fully Convolutional Networks (FCN) into *DMMSS*. Recall from *DMMSS*, what we are trying to decide is to decide whether two adjacent superpixels should be merged or not by forwarding a two-sueprpixel patch in the CNN. That is, each superpixel patch contains sufficient information for the CNN to make a decision for merging. In this case, by solving multiple subproblems like this, we could produce highly accurate image segmentation results. Therefore, each merging decision requires one forward pass in the CNN, which could be an effective yet inefficient solution. As a result, we rethink the superpixel merging from a different perspective, trying to solve the image segmentation with just one forward pass in the CNN to improve the efficiency. That is, back to two-superpixel patch merging problem as in fig 4.1, we convert this subproblem from deciding whether two adjacent superpixels should be merged into whether the boundary between two superpixels should be keep or not. Since the existence of boundary between two superpixels implies that they are separated from each other, in other words, they are not merged and the disappearance of boundary means they are merged.

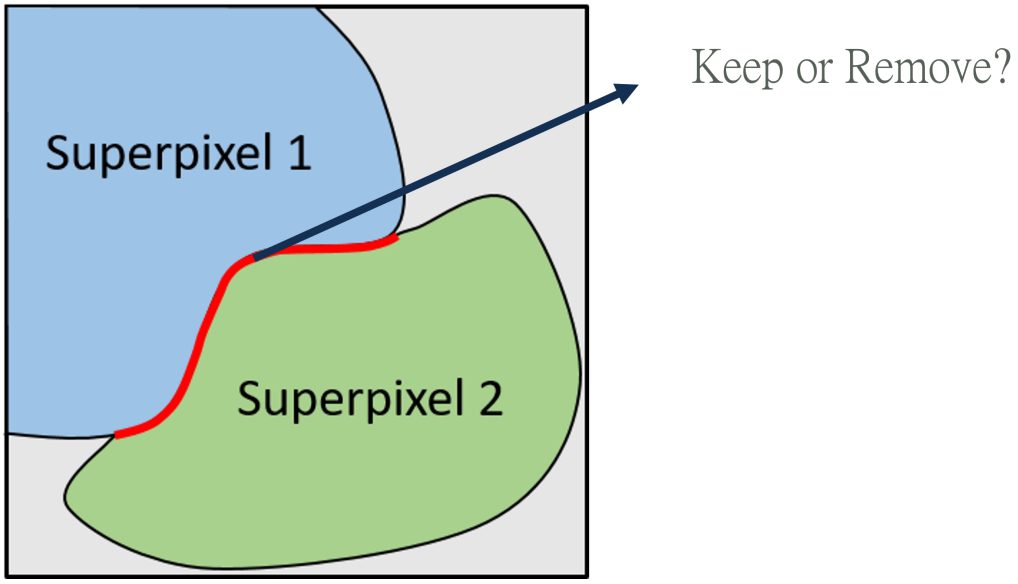


Figure 4.1: Rethinking two-superpixel patch.

Therefore, with the use of FCN, we perform pixel-wise prediction on the boundary and modified $ContourRate$ in Eq. 3.5 into Eq. 4.1 $BoundaryRate$ to quantize the tendency of keeping a boundary.

$$BoundaryRate(i,j) = \frac{\#\text{ of pixels of } (\textit{keep} \text{ label} \cap Bnd(i,j))}{\#\text{ of pixels of } Bnd(i,j)} \tag{4.1}$$

where *keep* label indexes the label of pixel prediction on the boundary pixel is to keep the boundary. For example in Fig. 4.2, there are 15 pixels along the boundary $Bnd(i,j)$, 10 of them are predicted as *keep* label (orange circles) while 5 of them(black circles) are *remove* label, resulting in a $BoundaryRate$ of 2/3. Therefore, we can thresholding on the $BoundaryRate$ to get the segmentation result from the FCN output.

The overview of the proposed *DMMSS-FCN* is illustrated in Fig. 4.3.

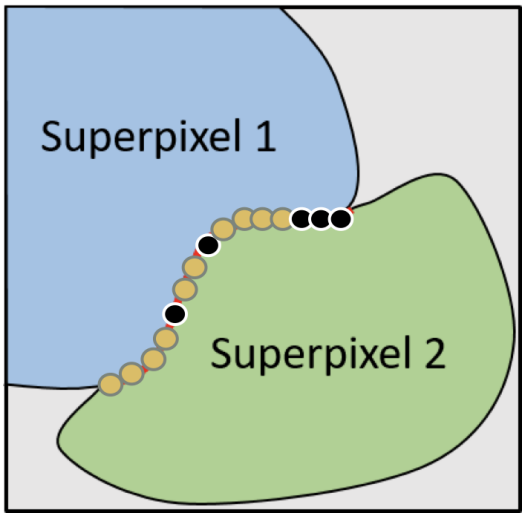


Figure 4.2: BoundaryRate example.

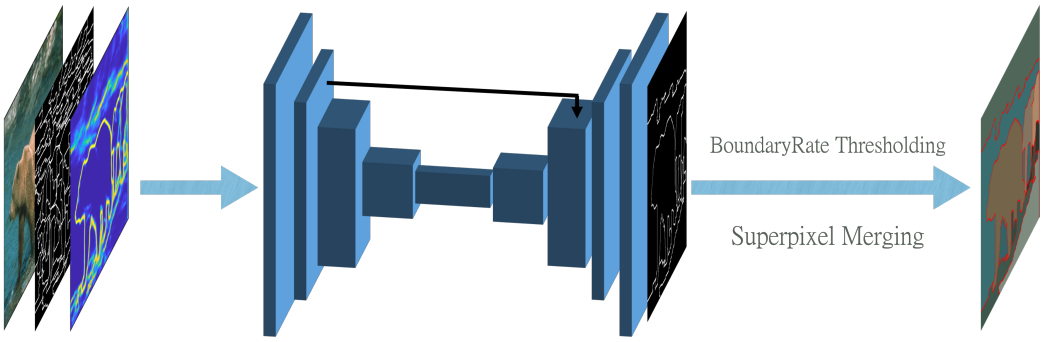


Figure 4.3: **Overview of DMMSS-FCN**. Left: We form the 5-channel input data by concatenating a RGB(3-channel) image with a superpixel boundary map(1-channel) and a edge-detection map(1-channel). Middle: Forward the input data to perform pixel-wise prediction and output the superpixel merging map. Right: Use $BoundaryRate$ in Eq. 4.1 to threshold each superpixel boundary to perform superpixel merging. Finally, the output is the segmentation result.

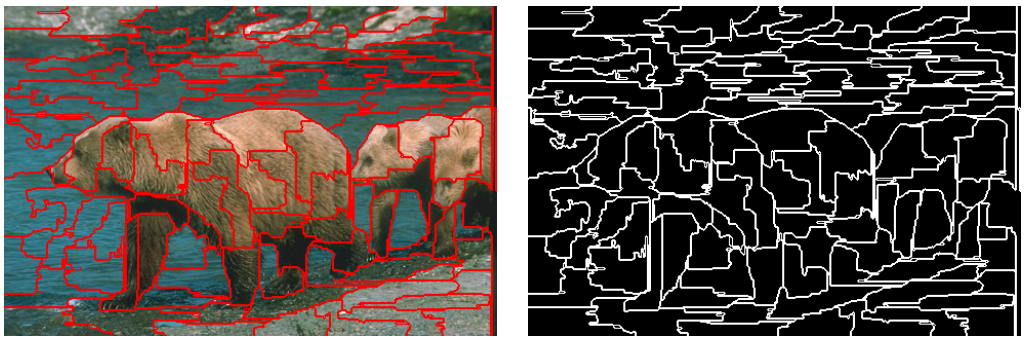


Figure 4.4: Superpixel boundary map generation. Left: Superpixel result from SEAL-ERS. Right: Binary image with superpixel boundaries marked as 1s(*keep* label), others are 0s(*remove* label).

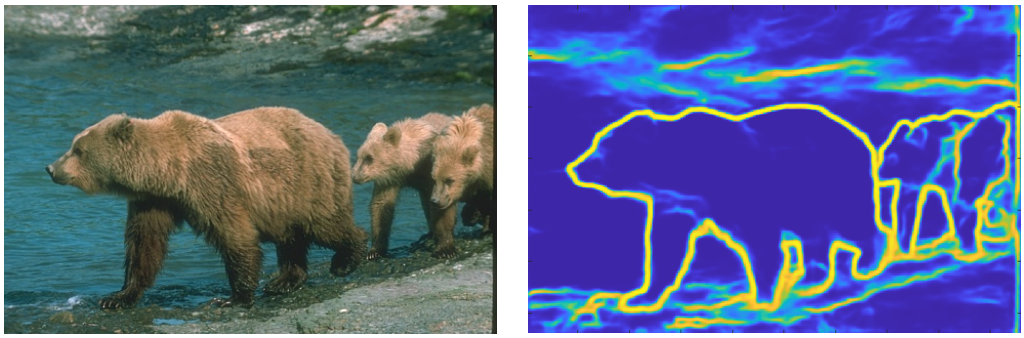


Figure 4.5: Example of RefineContourNet.Left: Original image. Right: RCN edge result.

4.1 5-channel Input Data

We defined our input data by concatenating a RGB(3-channel) image with a superpixel boundary map(1-channel) and a edge-detection map(1-channel). The RGB image indicates the original image while the superpixel boundary map is a binary image with only the boundary between two adjacent superpixels are marked as positive(*keep* label). The superpixel boundary map generation is shown in Fig. 4.4. And the edge-detection map is produced by RefineContourNet(RCN) [22]. Example of RCN is shown in 4.5.

The resulting input data is shown in Fig. 4.6. We have also tried different combination of concatenating input images, ablation studies have been carried out

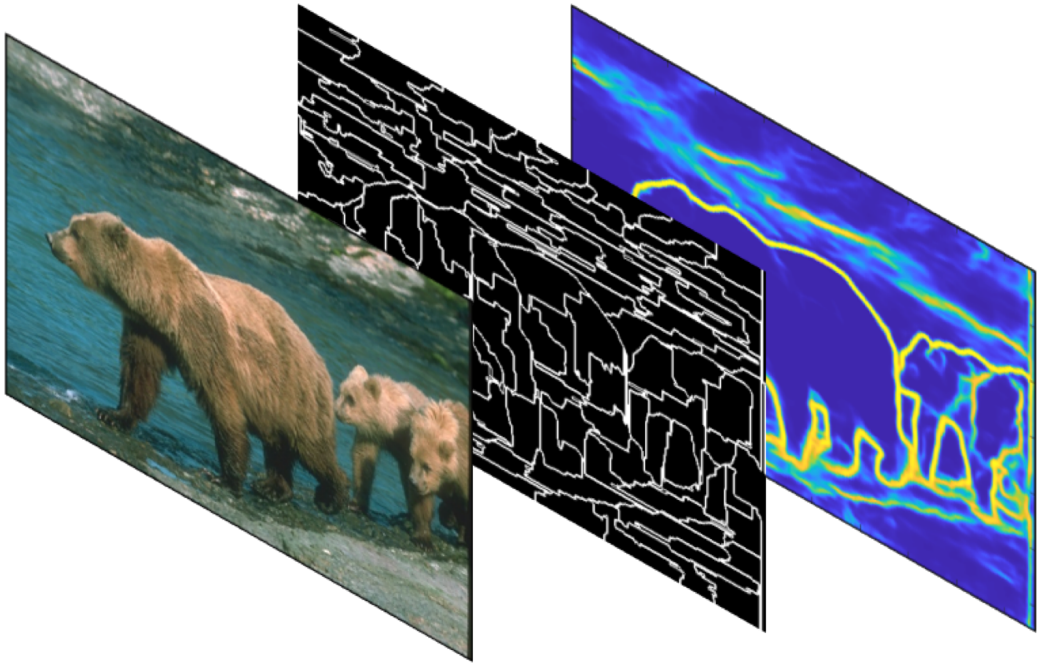


Figure 4.6: 5 channel input data.

to analyse the impact of different input channel in Sec. 4.5.3.

Since we want our model to be adaptive to any input superpixel type, the superpixel boundary map plays an important role in the whole process. As a result, it is essential to our input data. That is, with any superpixel result as prior input, our model can determine which boundary should be keep, with the superpixel boundary map as attention mechanism.

4.2 Output And GroundTruth

In this section, we defined the output of the Fully Convolutional Networks and the generation of superpixel boundary groundtruth map. As shown in Fig. 4.7,We generate our groundtruh of superpixel merging map by transforming the results acquired from Sec 3.1 into binary boundary map. With the superpixel boundary map as attention mechanism, the output of the networks should adhere to the input superpixel boundary map, in other words, same RGB image with different superpixel generation results should output response to their own superpixel type.

In Fig. 4.8, we show that different superpixel boundary input will correspond to different groundtruth of superpixel merging map. As a result, varying the parameters of different superpixel algorithm could bring us many superpixel groundtruth pairs, hence, increasing the training data. That is, we want the model to output superpixel merging map that adheres to the superpixel boundaries, since we use $BoundaryRate$ to perform thresholding to recover segmentation result, bad localization of predicted label could cripple the effectiveness of $BoundaryRate$.

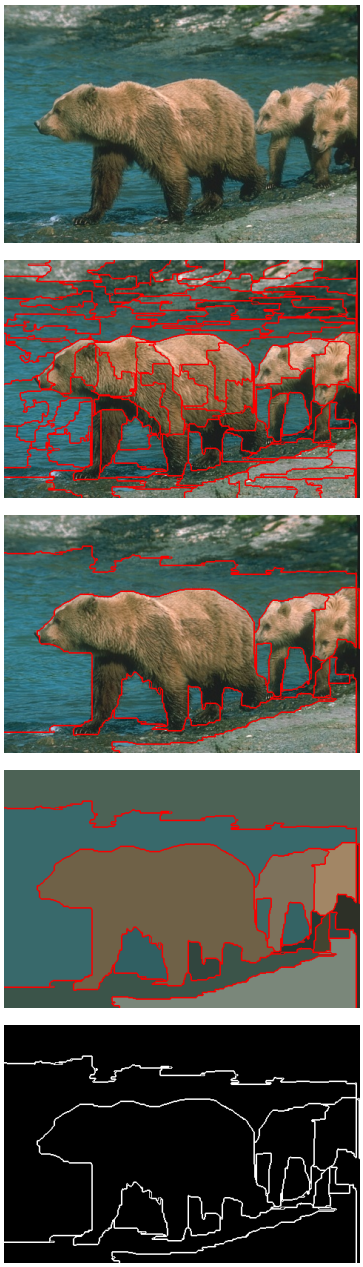
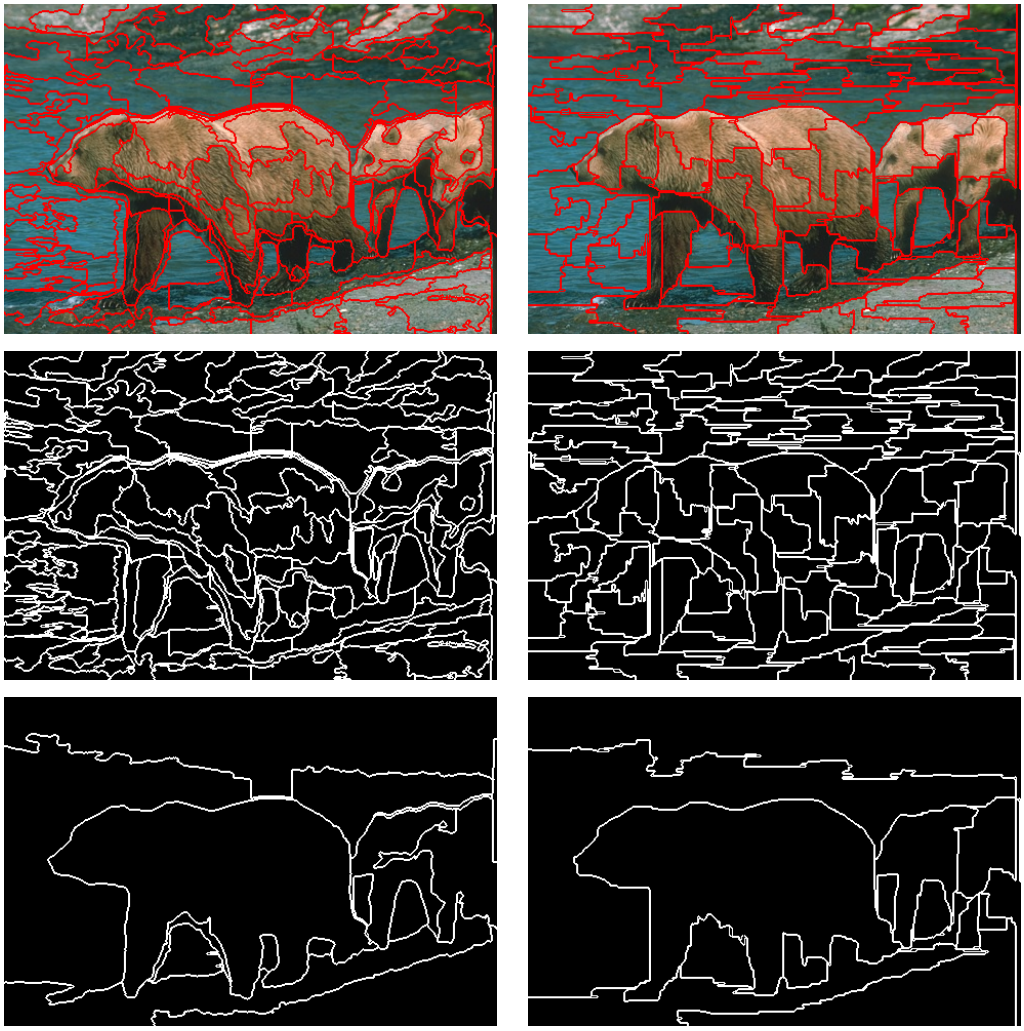


Figure 4.7: Groundtruth generation. 1^{st} row: Original image. 2^{nd} row: SEAL-ERS superpixel result. 3^{rd} and 4^{th} row: Results from Sec. 3.1. $5^{th}row$: Groundtruth superpixel boundary map by converting the resulting groundtruth into binary image.



(a) SSN 100 (b) SEAL-ERS 100

Figure 4.8: Different groundtruth of superpixel merging map with different superpixel input. **Left Col**: Groundtruth of superpixel merging map generated from SSN with the number of superpixel set to 100. **Right Col**: Groundtruth of superpixel merging map generated from SEAL-ERS with the number of superpixel set to 100. As can be observed, different superpixel algorithm produced different superpixel boundary map, since in Sec 3.1, the oracle-guided two-superpixel patches generation will produce the perfect segmentation result achieved by the current input superpixel result.

4.3 Training Architecture

Since our goal is to perform pixel-wise prediction on the image, and examine the $BoundaryRate$ along the boundary of two adjacent superpixels, the localization of predicted labels is crucial in our task. Therefore, we adopted the DeepLab V3+[29] FCN architecture proposed by Chen et al. , it utilized traditional Encoder-Decoder architecture for semantic segmentation with atrous convolution implemented in the encoder side for better field of view, and an simple but effective decoder with short cut skipped through the encoder part is added to this architecture, making it a highly accurate Fully Convolutional Networks structure while preserving good spacial information. The architecture of DeepLab V3+ is shown in Fig. 4.9

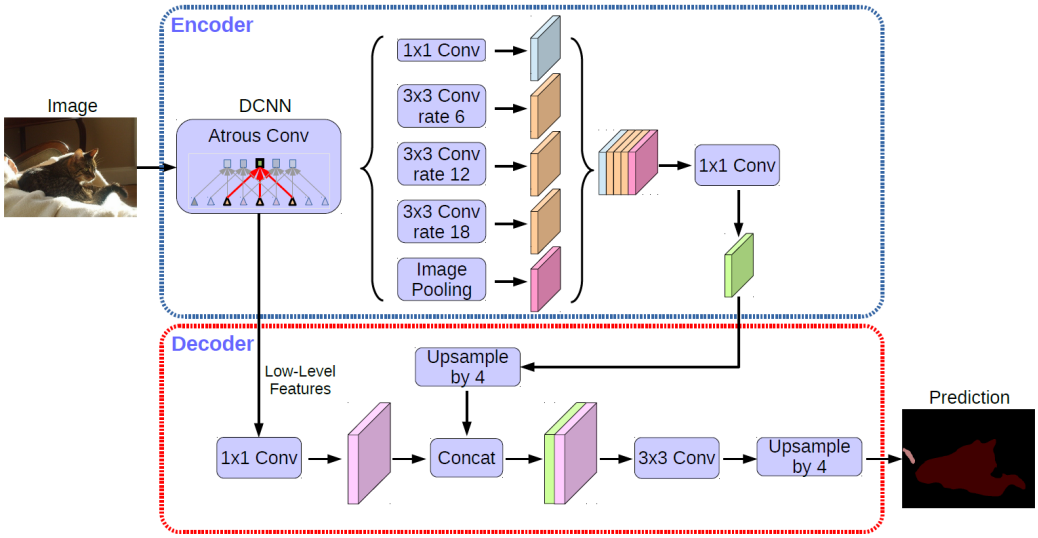


Figure 4.9: DeepLavV3+ architecture. Different from previous version of DeepLabV3 [14], a simple but effective decoder is added to the encoder for better localization of pixel-wise prediction.

In this thesis, we adopted the InceptionResNetV2[30] as the hidden encoder architecture, and the output stride of encoder is set to 16. And the batch size is set to 13 with the size of input data is $321 \times 481 \times 5$.

Since this is a binary classification problem of two labels, *keep* or *remove* labels, with the *keep* class is far less than the other one, we applied the weighted

binary cross entropy loss in 2.2 to calculate the loss. Furthermore, Adam optimizer is adopted to update the parameters of the networks with initial learning rate of 0.0001 and divided by 0.5 for every 10 epochs, and stopped training after 100 epochs.

We trained our *DMMSS-FCN* on the BSDS500 dataset, with 200 test images, 200 train images, and 100 validation images, the same dataset as we used to train our *DMMSS* models in Chap 3. Different from previous *DMMSS*, one FCN model is all we need. As we mentioned in Sec 4.2, different superpixel algorithm can produce different superpixel and groundtruth pairs. Therefore, we adopted SEAL-ERS [18] and SSN [17] both with the number of superpixels set to 100, and 200 to generate the training data, resulting in 800 training images, and 400 validation images in total.

4.4 Inference And Superpixel Merging

As we shown in Fig. 4.3, we first concatenate RGB image with superpixel boundary map and RCN edge map to form the 5-channel input data. Then we perform one forward pass through the networks to obtain superpixel merging map. Afterwards, we use the $BoundaryRate$ Eq. 4.1 to measure how many predicted $keep$ label are on each boundary of adjacent superpixels. Furthermore, $ContactRate$ Eq. 3.8 is also used as another thresholding criterion. Then, we perform Ramping thresholding merging procedure same as we do in Sec 3.4, first start merging by thresholding with the lowest threshold values and increase the threshold values bit-by-bit until there are no candidates for merging.

In Fig. 4.10, we show the input and output of the model. As can be seen, the top-3 rows are the input data that will eventually be joined together, the 4^th row is the output of the model that overlays with the original RGB image for better visualization. And the last row is the segmentation result after taking the output of *DMMSS-FCN* as prior information to perform BoundaryRate thresholding. It

is worth noting that as we take RCN edge-detection map as one of the input map, although the RCN has already produced highly accurate edge-detection result, our proposed model managed to suppress unnecessary part of RCN edge corresponding to the input superpixel boundary map. For example, the strong edge responses over the middle part of human palm, the top fin area, and the cheek area of the fish have been removed after forwarding to the *DMMSS-FCN*, producing a more general object contour result.

4.5 Experiments

In this section, we carried out tons of experiments and ablation studies to justify that our proposed *DMMSS-FCN* can outperform many state-of-the-art algorithms, even compared to our own *DMMSS*, the proposed *DMMSS-FCN* can still surpass it in all evaluation metrics while boosting the speed, making *DMMSS-FCN* a highly accurate and efficient generic image segmentation algorithms.

4.5.1 Segmentation Evaluation

To compare the proposed *DMMSS-FCN* algorithm to the existing methods, we evaluate the performance on the standard metrics of segmentation covering (SC), the probabilistic rand index (PRI), and the variation of information (VI) [3] same as we adopted in the previous chapter. A higher SC and PRI and a lower VI means better performance.

We compare the proposed *DMMSS-FCN* algorithm to the state-of-the-art methods, including the W-Net [11], gPb-owt-ucm [4], DC-Seg-full [25], Taylor [26], Felzenszwalb and Huttenlocher (Felz-Hutt) [7], Mean Shift [2], Canny-owt-ucm [4], Multiscale Normalized Cuts (NCuts) [5], fPb-owt-ucm [9], cPb-owt-ucm [9], and our own *DMMSS*. As the proposed algorithm, all the algorithms compared in Table 4.1 have not to assign the number of regions in prior.

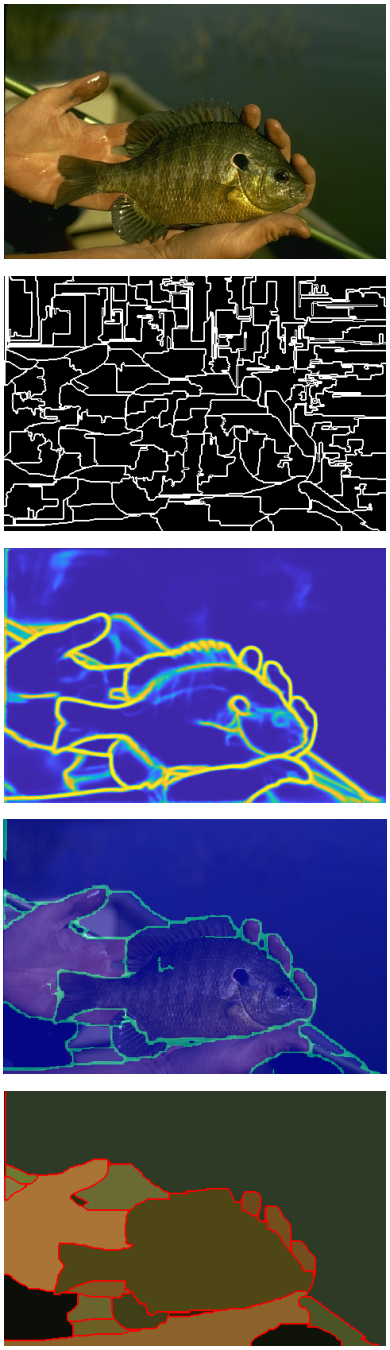


Figure 4.10: Input and Output of *DMMSS-FCN*. From top to bottom: 1^{st} **row:** Original RGB image. 2^{nd} **row:** Superpixel boundary map. 3^{rd} **row:** RCN edge-detection map. 4^{th} **row:** Output of the model overlay with original image. 5^{th} **row:** Segmentation result.

Table 4.1: **Results on the BSDS500 dataset.**

Method	**VI**	**PRI**	**SC**
Ncuts	2.23	0.78	0.45
Canny-owt-ucm	2.19	0.79	0.49
Felz-Hutt	2.21	0.80	0.52
Mean Shift	1.85	0.79	0.54
Taylor	1.78	0.81	0.56
W-Net	1.76	0.81	0.57
fPb-owt-ucm	1.70	0.82	0.58
DC-Seg-full	1.68	0.82	0.59
W-Net+ucm	1.67	0.82	0.59
gPb-owt-ucm	1.69	0.83	0.59
cPb-owt-ucm	1.65	0.83	0.59
DMMSS(SSN)	**1.46**	**0.86**	**0.63**
DMMSS-FCN(SSN)	**1.38**	**0.87**	**0.66**
Human Drawing	1.17	0.88	0.72

4.5.2 Run Time Analysis

In the following article, we analyze the run time of both *DMMSS* and *DMMSS-FCN* compare to the state-of-the-art segmentation algorithm gPb-OWT-UCM [4]. We evaluate our algorithms on SEAL-ERS superpixel, and perform inference on the BSDS500 test set. Furthermore, we also report the run time of our parallel-computing version of our proposed methods. In Table 4.2, the average run time of processing single image is presented. One can find out that although our proposed *DMMSS* has drastically reduce the run time compare to gPb-OWT-UCM, the proposed *DMMSS-FCN* managed to further decrease the run time up to x100 times less than the gPb-OWT-UCM and x30 times less than previous *DMMSS* and meanwhile boosting the performance. More on that, even we switch our

Table 4.2: **Run time result on the BSDS500 test set.**

Method	**SEAL-ERS 100**	**SEAL-ERS 200**
DMMSS(CPU+GPU)	30s	60s
DMMSS(parallel)(CPU+GPU)	15s	30s
DMMSS-FCN(CPU+GPU)	**0.94s**	**1.56s**
DMMSS-FCN(parallel)(CPU+GPU)	**0.26s**	**0.47s**
DMMSS-FCN(CPU)	8.2s	8.7s
gPb-OWT-UCM(CPU)	100s	100s

DMMSS-FCN to CPU mode, our proposed *DMMSS-FCN* still 12x faster than the gPb-OWT-UCM and almost twice as fast as the *DMMSS*.

4.5.3 Ablation Study

In this section, we will discuss the functionality of some key component in our proposed algorithm, and how they affect the overall performance. Including the different combination of input data, different superpixel generation algorithm, and different DeepLabV3 Plus implementation detail, furthermore, different inference techniques.

Combination of Input Data

Following, we show that different combination of input data could have a great impact on the performance of the model. In this section, we mainly use the SEAL-ERS 100 as the underlying superpixel representation. In Table 4.3, we show the difference of concatenating different feature map as input data.

In the early design stage of our proposed *DMMSS-FCN*, we first design the input data to have only 4-channel, that is, a RGB image(3-channel) concatenated with superpixel boundary map. During training, we found out that the model did not perform as we expected. As a result, we thought that concatenating another

Table 4.3: **Performance of different input data combination.**

Input Data	VI	PRI	SC
4-channel: RGB(3)+spixel bdry(1)	1.569	0.846	0.547
5-channel: RGB(3)+spixel bdry(1) +RCN edge(1)	**1.446**	**0.863**	**0.634**
5-channel: LAB(3)+spixel bdry(1) +RCN edge(1)	1.458	0.860	0.632
7-channel: RGB(3)+spixel bdry(1) +RCN edge(1)+AffinityXY(2)	1.472	0.861	0.629
7-channel: LAB(3)+spixel bdry(1) +RCN edge(1)+AffinityXY(2)	1.534	0.859	0.606

feature map as prior information might improve the performance. Therefore, as we added the RCN edge-detection map, we got a huge gain in performance. Hence, we tried to concatenated more feature map or change RGB to CIE LAB color space to further improve the performance. For example, we use the Affinity map that generated from the SEAL-ERS in Sec. 2.1.2 as candidates for feature map concatenation. Nevertheless, from the experimental results in Table 4.3, more channels of feature maps do not imply better performance. In the end, we adopted the 5-channel with RGB image as the input of the model.

Superpixel

Same as we did in Sec. 3.5.2, we measure the performance over different number of initial superpixels. In Fig. 4.11, the proposed *DMMSS-FCN* behaves the same as *DMMSS*. Therefore, our proposed *DMMSS-FCN* still maintains great performance over different numbers of initial superpixels even if some of them are not included during training.

In this work, we input the superpixel boundary map as attention mechanism for model to focus on the boundary part then make prediction on the pixels along

Figure 4.11: Performance of different numbers of superpixel between SSN and SEAL-ERS over *DMMSS* and *DMMSS-FCN*.

boundary. In Fig. 4.12, we show that the model do react to different type of superpixel boundary map input that specifically perform prediction along boundaries, where yellow lines index that the prediction overlapped with the corresponding groundtruth superpixel boundary map, the light blue parts indicate that the model predict *keep* label, and the dark blue parts represent the model predict *remove* label. Therefore, as we observe, different input superpixel type will lead the model to predict corresponding superpixel merging map.

As we discussed before, our model can suppress unnecessary edge response. In addition, from the result of RCN edge map in Fig. 4.12, we can find out that weak edge response can also be captured over the water area. In general, the model capture different level of contextual information to produce accurate prediction.

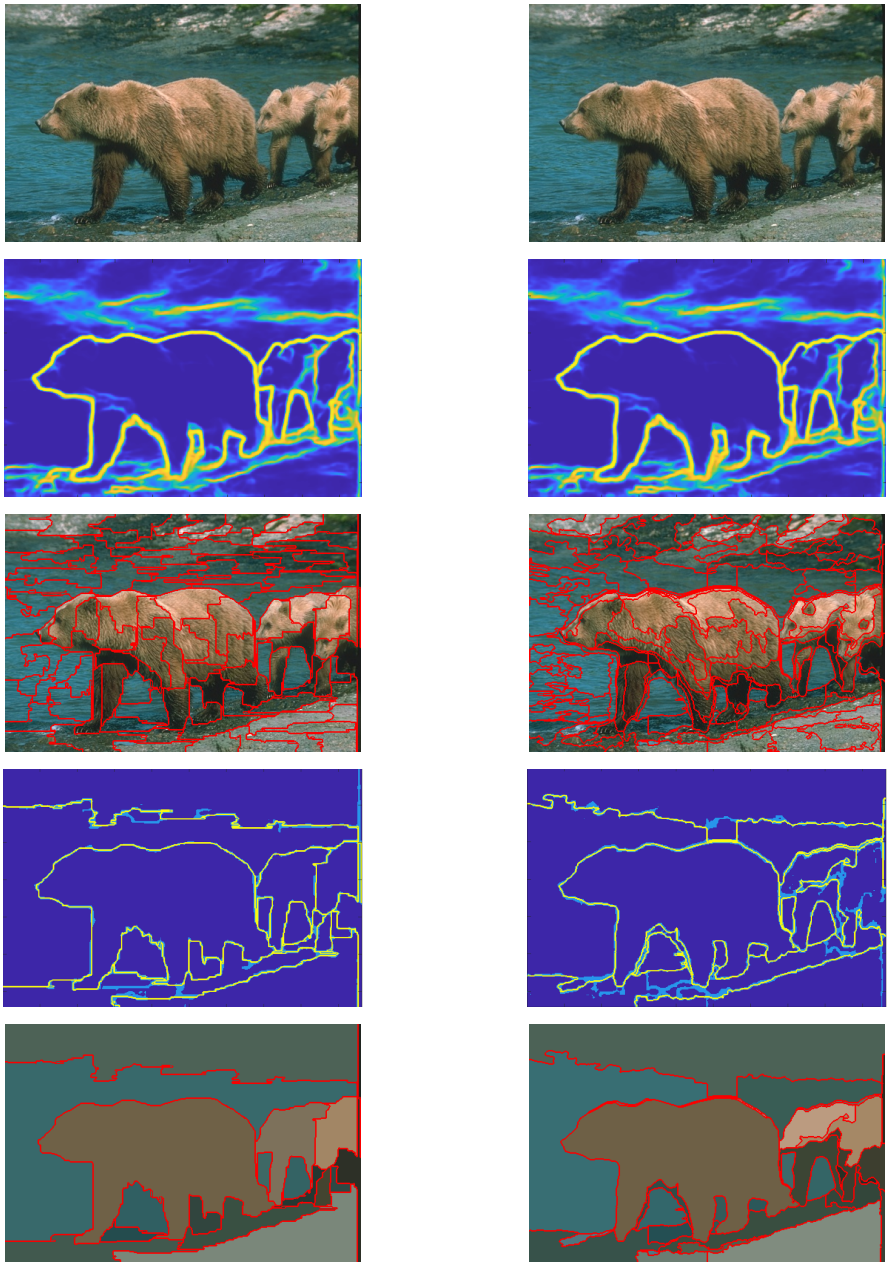
In Fig. 4.13, we present the simulations that apply the proposed DMMSS-FCN algorithm to merge the superpixels generated from different algorithms. The results show that, no matter which type of superpixels is applied, with the proposed DMMSS algorithm, very high-quality segmentation results can be achieved.

DeepLabV3 Plus Variation

In [29], Chen et al. proposed a powerful encoder-decoder-based fully convolutional networks. In the article, the author suggest different variation of implementation details, including changing the output stride of the model or replacing hidden encoder architecture with any existing CNN architecture such as ResNet [24] or Xception [31]. More on that, inference techniques like forwarding not one image but four up-down and right-left flipped images at once is also reported as legit method to gain performance. Therefore, the following article will discuss the difference of the performance among the variation of DeepLabV3 Plus applied in our work.

We choose SEAL-ERS 100 as the underlying superpixel representation to carry out the following experiments.

In Table 4.4, we reported the difference among two recent CNN architecture,



(a) SEAL-ERS 100 (b) SSN 100

Figure 4.12: Different superpixel boundary map input. (1^{st} **Row):** Original image. (2^{nd} **Row):** RCN edge map. (3^{rd} **Row):** Superpixel boundary map overlapped with original image. (4^{th} **Row):** Superpixel merging maps prediction overlapped with corresponding groundtruth merging maps. (5^{th} **Row):** Segmentation results.

Table 4.4: **Performance of different encoder architecture and output stride.**

Encoder Architecture	Output Stride	VI	PRI	SC
Xception	8	1.569	0.860	0.600
Xception	16	1.532	0.857	0.606
InceptionResNetV2	8	1.635	0.854	0.583
InceptionResNetV2	16	1.446	0.863	0.634

Table 4.5: **Result of adopting flipping technique.**

Encoder Architecture	Flip	VI	PRI	SC
InceptionResNetV2	No	1.446	0.863	0.634
InceptionResNetV2	YES	1.411	0.864	0.647

the Xception [31] and the InceptionResNetV2 [30] with two different output stride setting. In the original DeepLabV3 paper [14], the author indicated that smaller output stride could greatly increase the training time and memory usage with a negligible improvement which corresponds to our case here. As for different encoder architecture, InceptionResNetV2 has better performance over the Xception.

Furthermore, we also adopted the inference technique mentioned in [29] to improve the performance, that is, we generate extra images by flipped the original image, then feed them into the networks, then average the scores of the outputs to obtain the final superpixel merging map. In Table 4.5, we show that a simple but effective flipping technique could boost the performance.

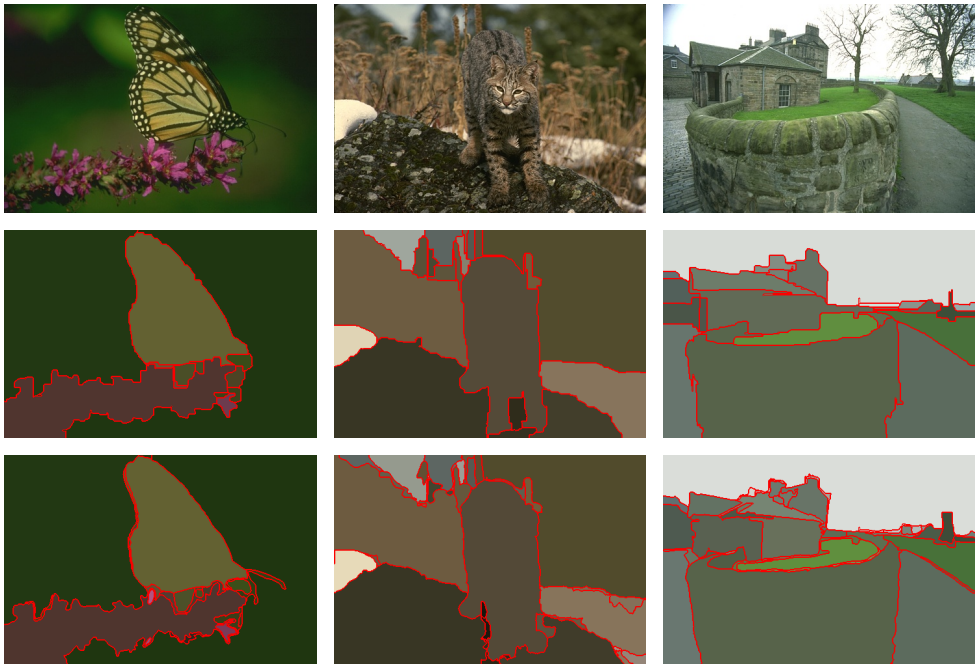


Figure 4.13: Results generated by different superpixel type as input prior.(1^{st} row): original images. Results produced by the proposed *DMMSS-FCN* algorithm using superpixels generated by (2^{nd} row): SEAL-ERS; (3^{rd} row): Superpixel Sampling Network (SSN).

Chapter 5

Simulations

In this chapter, comparisons towards other segmentation algorithms are presented. We compare our simulation results on the BSDS500 test set in Sec 5.1 with other algorithms. Moreover, we further run simulations on the real-world images that covers several kinds of scenarios in Sec 5.2 to justify that under any circumstances, both our proposed algorithms can perform highly accurate and efficient superpixel merging techniques to produce good generic image segmentation results.

5.1 BSDS500 Test Images

Fig. 5.1 and Fig. 5.2 show the comparison between the proposed methods and other methods like a deep-learning-based method, the W-Net[11], and classical segmentation algorithm gPb-OWT-UCM [4]. As we can see, both the results of the proposed algorithms are much better than that of state-of-the-art algorithms, since ours can produce more general and compact segmentation results compared to the others.

In Fig. 5.3 and Fig. 5.4, we show another visual comparison of the proposed algorithms to DC-Seg-full [25], which is a famous learning-based method. Compare to the results of DC-Seg-full, the proposed methods can produce more compact and reliable segmentation.

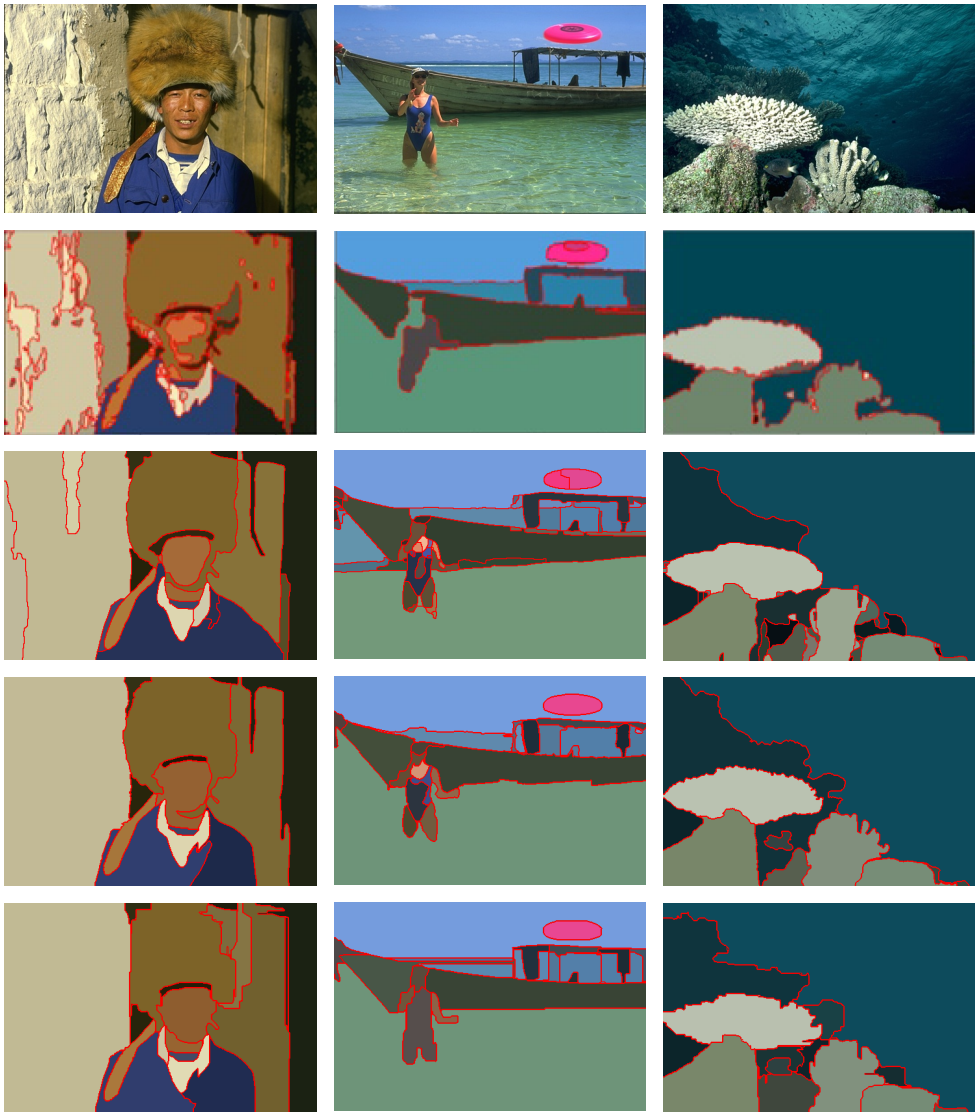


Figure 5.1: Visual comparison between WNet, gPb-OWT-UCM, and the proposed *DMMSS* and *DMMSS-FCN*. (1^{st} row): original images. Results produced by (2^{nd} row): WNet [11]; (3^{rd} row): gPb-OWT-UCM [4]; (4^{th} row): *DMMSS(ours)*; (5^{th} row): *DMMSS-FCN(ours)*.

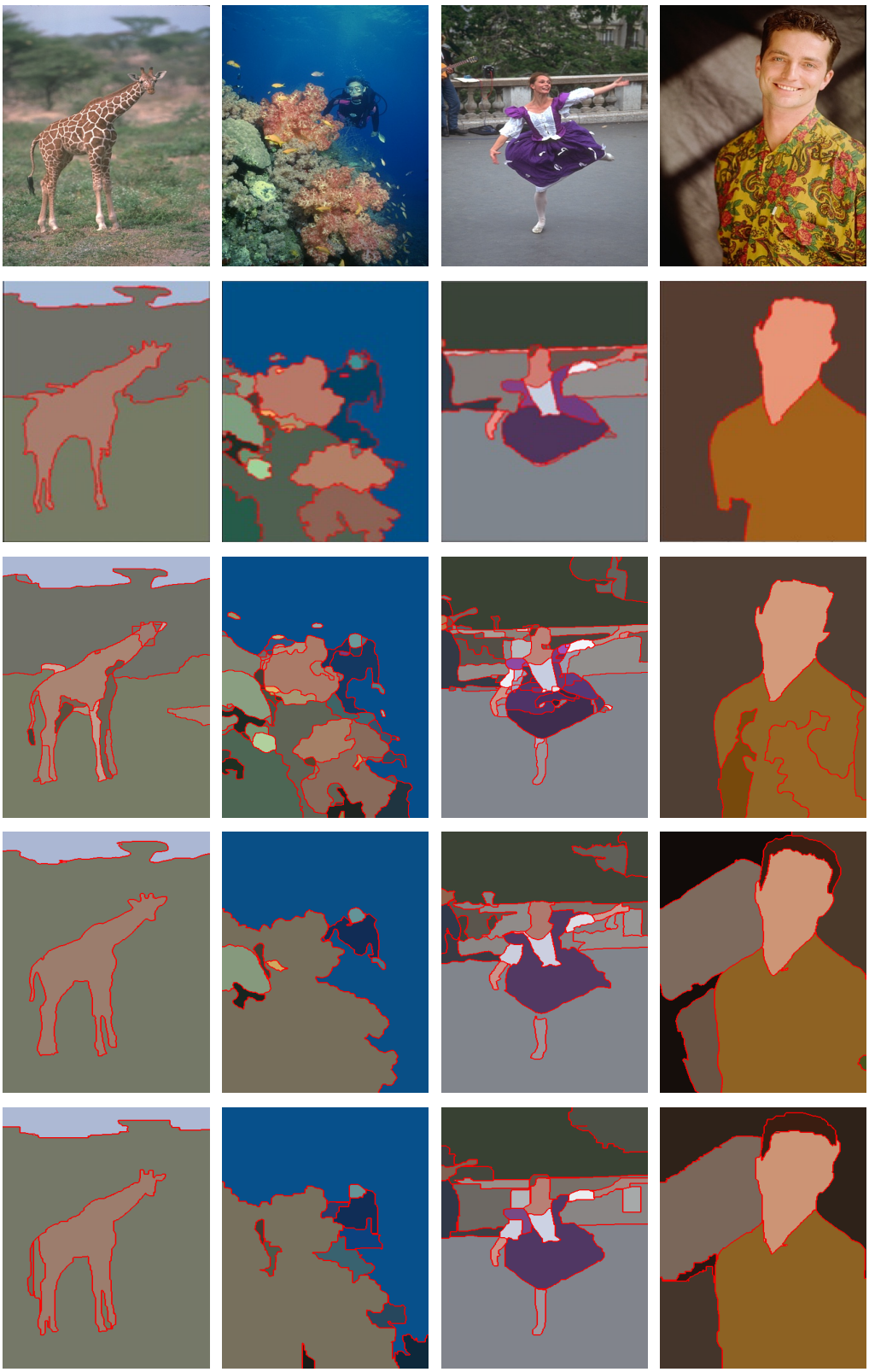


Figure 5.2: Visual comparison between WNet, gPb-OWT-UCM, and the proposed *DMMSS* and *DMMSS-FCN*. (1^{st} row): original images. Results produced by (2^{nd} row): WNet [11]; (3^{rd} row): gPb-OWT-UCM [4]; (4^{th} row): *DMMSS(ours)*; (5^{th} row): *DMMSS-FCN(ours)*.

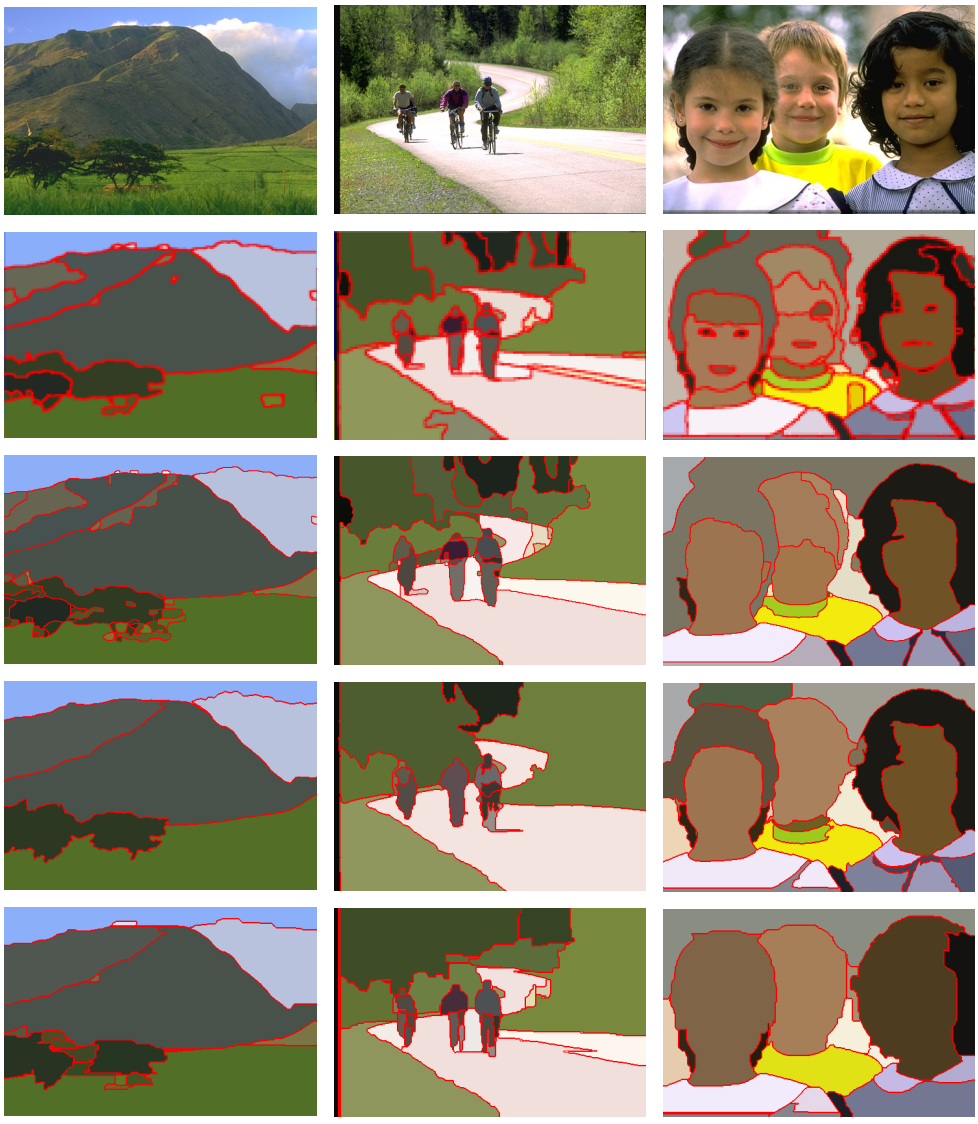


Figure 5.3: Visual comparison between DC-Seg-full, and the proposed *DMMSS* and *DMMSS-FCN*. (1^{st} row): original images. Results produced by (2^{nd} row): DC-Seg-full [25]; (3^{rd} row): gPb-OWT-UCM [4]; (4^{th} row): *DMMSS(ours)*; (5^{th} row): *DMMSS-FCN(ours)*.

Figure 5.4: Visual comparison between DC-Seg-full, and the proposed *DMMSS* and *DMMSS-FCN*. (1^{st} row): original images. Results produced by (2^{nd} row): DC-Seg-full [25]; (3^{rd} row): gPb-OWT-UCM [4]; (4^{th} row): *DMMSS(ours)*; (5^{th} row): *DMMSS-FCN(ours)*.

5.2 Real-World Images

In the following articles, we show that not only our models can outperform the state-of-the-art algorithms in BSDS500 dataset which we train our models from, but also in real-world scenarios. Therefore, we collected several types of images to prove that our proposed methods can maintain great performance over unseen data. In BSDS500 [4], there are many classes of nature images within this dataset, making it a great benchmark and training source for learning image segmentation tasks. Nevertheless, there are too many classes among this world, it is difficult for a dataset to cover them all. Therefore, it is important for every deep-learning model to be bias-invariant. To verify that, we test both our proposed algorithms on modern real-world images.

5.2.1 Buildings

We present examples of generic image segmentation of buildings from different distance. As we can see in Fig. 5.5 and Fig. 5.6, our models adapt to the distance of the images. That is to say, the further the buildings are, the more likely they are being segmented as an objects. On the contrary, the closer the buildings are, more details of buildings are being revealed, since in our perspective, human tend to look closer for details of objects. As in gPb-OWT-UCM, the overall segmentation results are fragmentary regardless of the distance of objects.

5.2.2 Animals

Since our training set contains many nature images, our proposed methods are able to produce outstanding segmentation results. See Fig. 5.7 for segmentation results on animal images compared to gPb-OWT-UCM.

5.2.3 Night View

In this section, we show some segmentation results on the images taken in the night to test the capability of our proposed methods of handling dark view scenario. In Fig. 5.8 and Fig. 5.9, one can see that, our models manages to merge blur and low-luminance parts into compact objects while the other method fails to recognize such information and cause severe segmentation leakage. Therefore, such examples prove that our models are robust not only in images taken under great exposures, but in those low-light and blurry scenarios.

5.2.4 Items and Objects

We specifically pick some items and objects that are few in number or never been in our training set for testing. In the left column of Fig. 5.10, we present simulations of an aircraft engine, which is difficult for another algorithm to perform accurate segmentation on the boundary of engine itself since the background information is similar to the engine. Nevertheless, both our proposed methods can generate compact segmentation without losing edge information. Moreover, food category such like coffee and snacks are never been shown in the training images. However, reliable segmentation result can still be produced by our methods. Fig. 5.11 shows more examples of items and objects, on the left-most column, our methods successfully segmented the saliency part of the image, which is the stone.

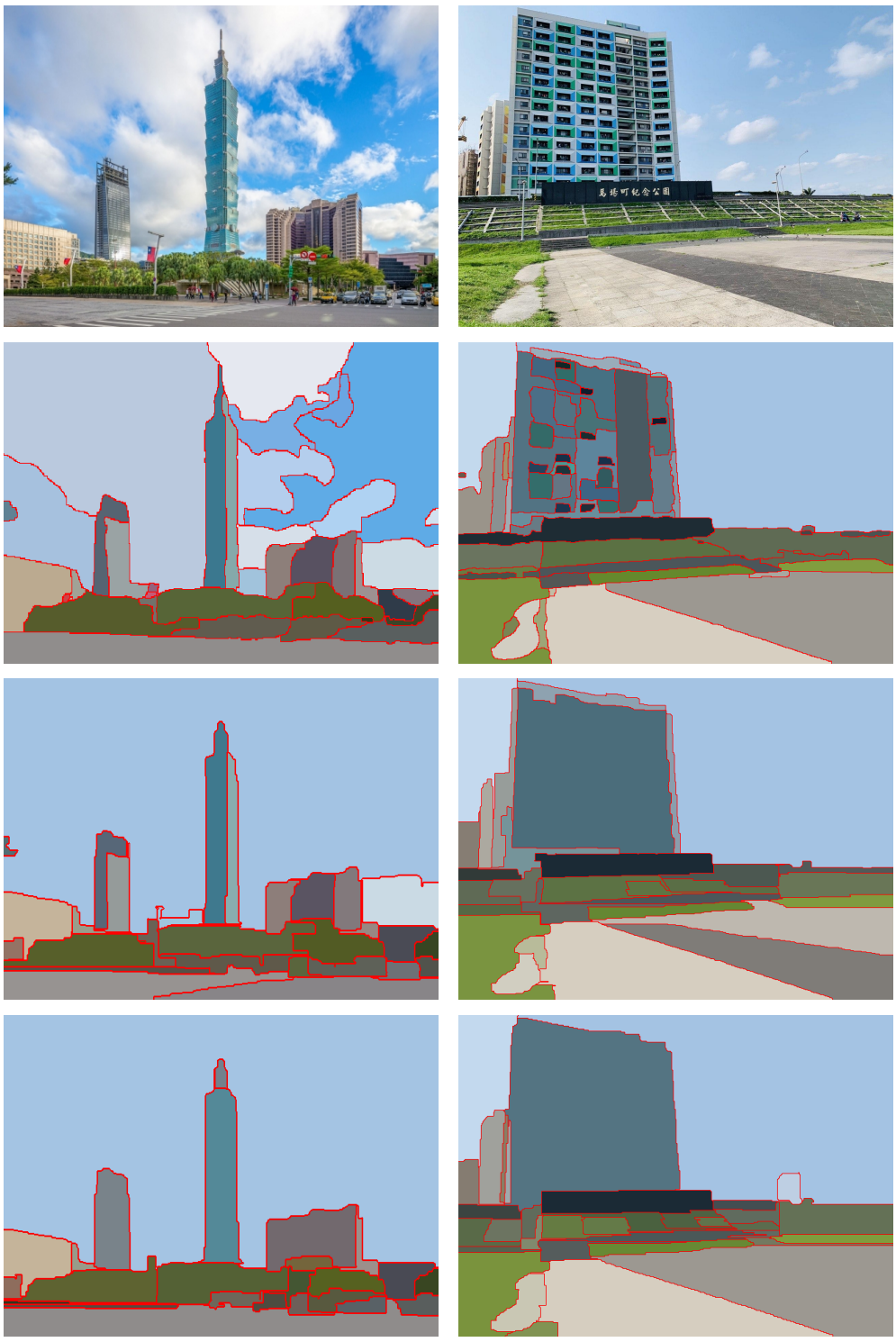


Figure 5.5: Real-world building image segmentation results. (1^{st} row): original images. Results produced by (2^{nd} row): gPb-OWT-UCM [4]; (3^{rd} row): *DMMSS(ours)*; (4^{th} row): *DMMSS-FCN(ours)*.

Figure 5.6: Real-world building image segmentation results. (1^{st} row): original images. Results produced by (2^{nd} row): gPb-OWT-UCM [4]; (3^{rd} row): *DMMSS(ours)*; (4^{th} row): *DMMSS-FCN(ours)*.

Figure 5.7: Real-world animal image segmentation results. (1^{st} row): original images. Results produced by (2^{nd} row): gPb-OWT-UCM [4]; (3^{rd} row): *DMMSS(ours)*; (4^{th} row): *DMMSS-FCN(ours)*.

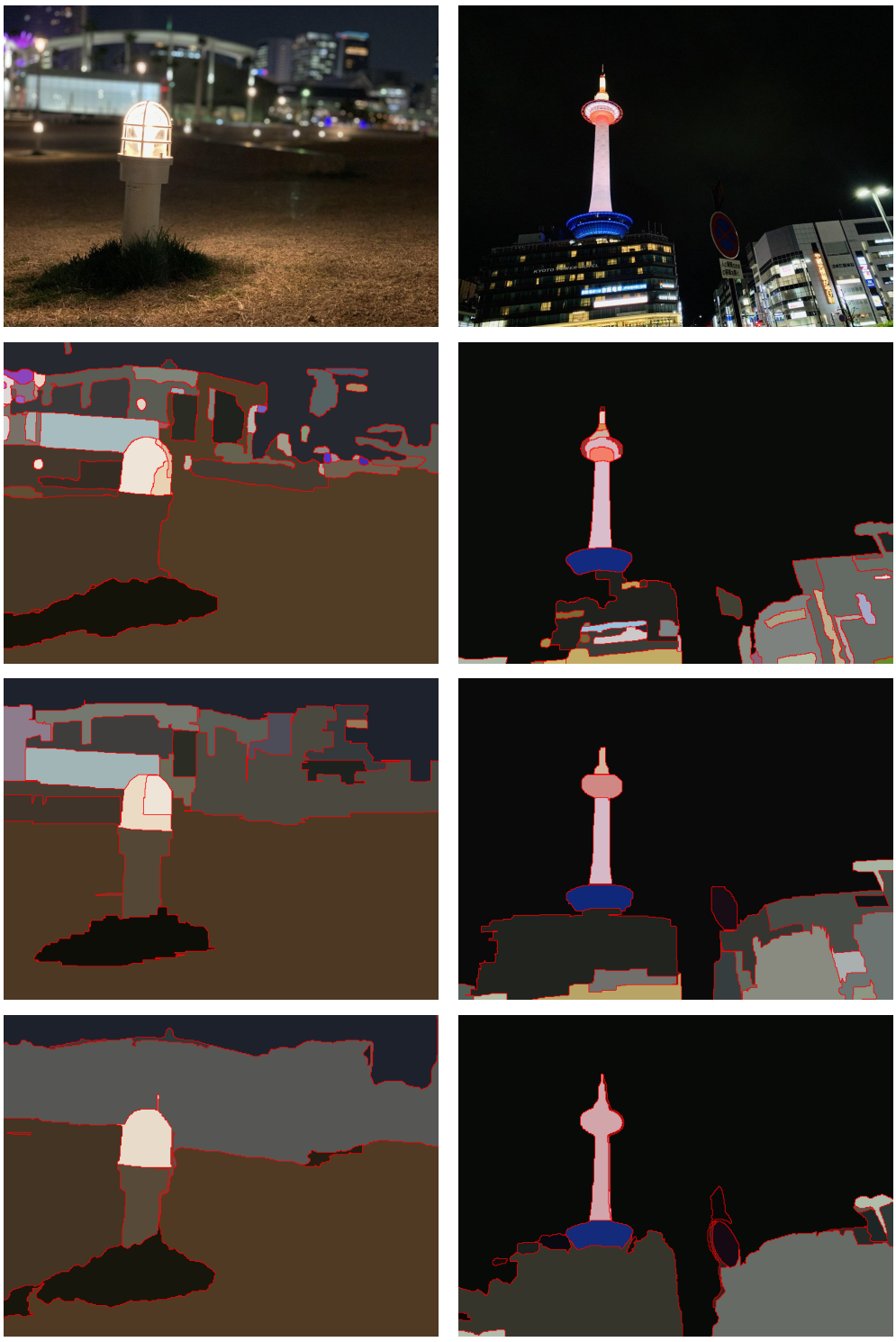


Figure 5.8: Real-world night view image segmentation results. (1^{st} row): original images. Results produced by (2^{nd} row): gPb-OWT-UCM [4]; (3^{rd} row): *DMMSS(ours)*; (4^{th} row): *DMMSS-FCN(ours)*.

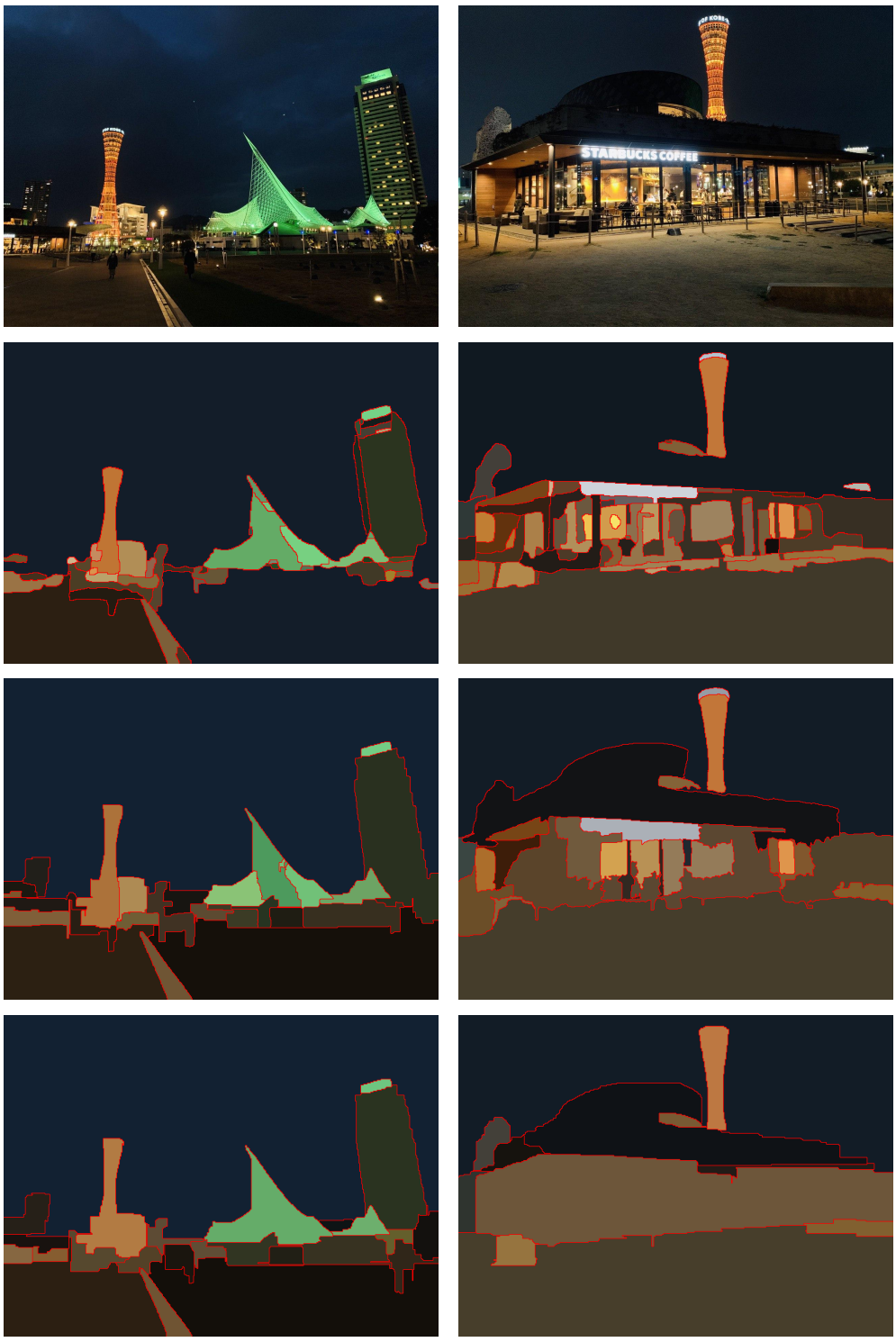


Figure 5.9: Real-world night view image segmentation results. (1^{st} row): original images. Results produced by (2^{nd} row): gPb-OWT-UCM [4]; (3^{rd} row): *DMMSS(ours)*; (4^{th} row): *DMMSS-FCN(ours)*.

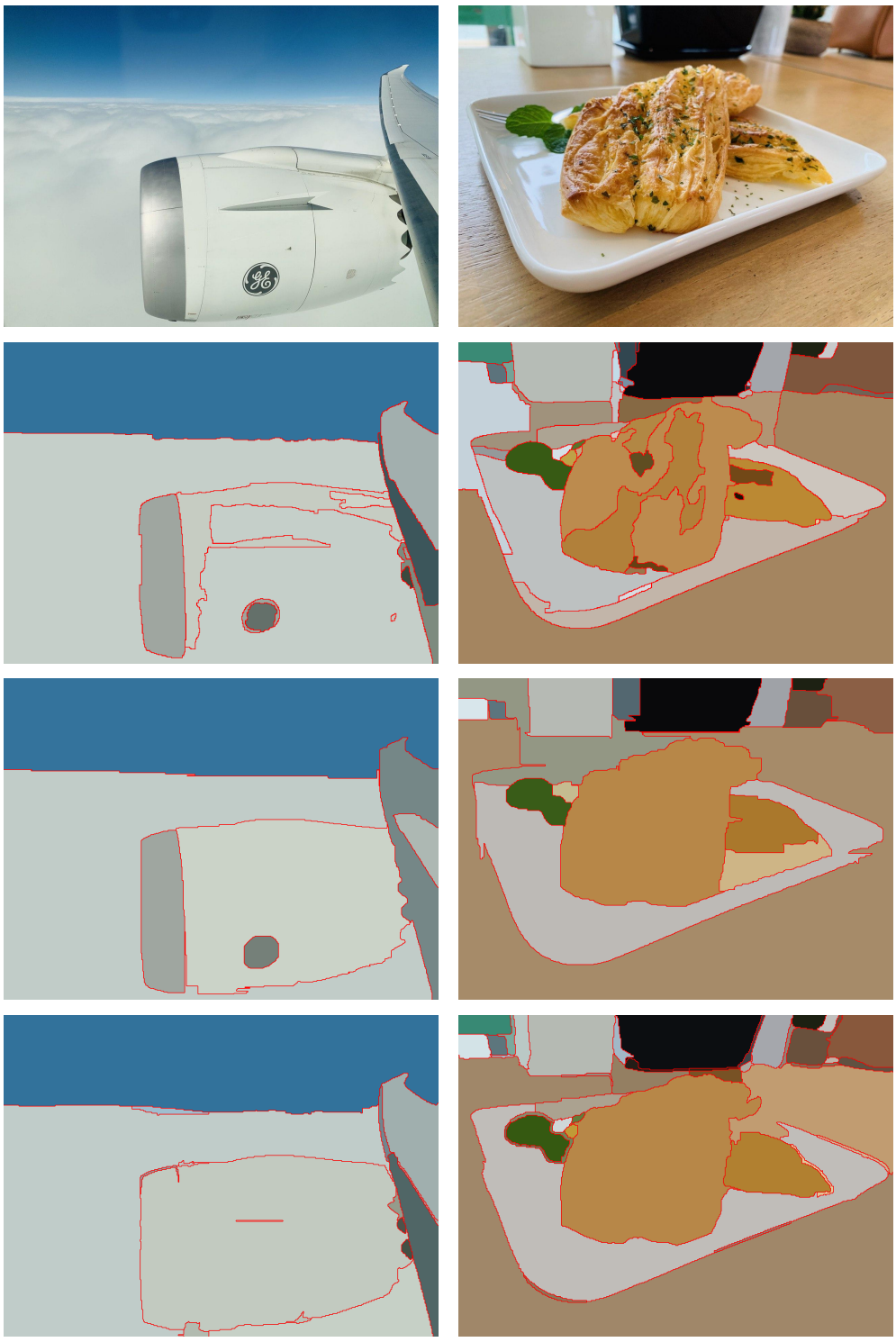


Figure 5.10: Real-world item and object image segmentation results. (1^{st} row): original images. Results produced by (2^{nd} row): gPb-OWT-UCM [4]; (3^{rd} row): *DMMSS(ours)*; (4^{th} row): *DMMSS-FCN(ours)*.

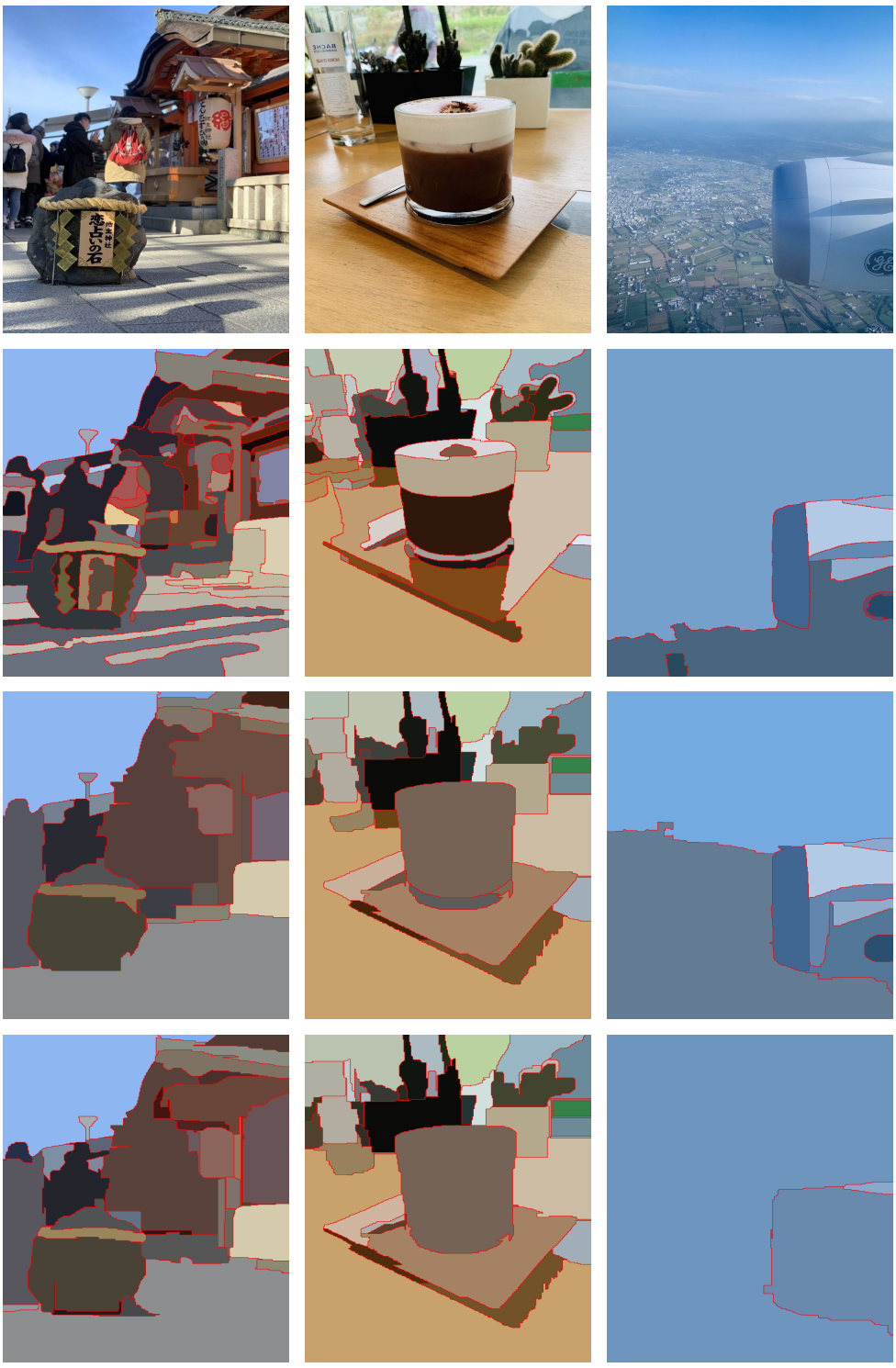


Figure 5.11: Real-world item and object image segmentation results. (1^{st} row): original images. Results produced by (2^{nd} row): gPb-OWT-UCM [4]; (3^{rd} row): *DMMSS(ours)*; (4^{th} row): *DMMSS-FCN(ours)*.

Chapter 6

Conclusion

In this thesis, two novel image segmentation algorithms that apply both deep learning architectures and superpixels are proposed. We call it *DMMSS* and *DMMSS-FCN*. The proposed *DMMSS* algorithm converts the image segmentation problem into a series of decision problems about whether two adjacent superpixels should be merged or not. The decision process can be well trained by the deep-learning architecture. The proposed method produces a segmentation by effectively merging superpixels from an over-segmented image and performs image segmentation fully automatically without any post-processing and further information from user.

For a learning-based method, to achieve better performance, sufficient amount of training data is required. Since in the proposed *DMMSS* algorithm the input of the network is an adjacent superpixel pair and there are many adjacent superpixel pairs within an image, one can acquire huge amount of data to train the network and obtain highly accurate segmentation results.

Simulation results show that the proposed *DMMSS* algorithm outperforms state-of-the-art image segmentation techniques, including both learning-based and rule-based algorithms. Moreover, the proposed DMMSS algorithm is fully automatic and the number of regions has not to be assigned in prior.

Inspired by *DMMSS*, we rethink the superpixel pairs in superpixel merging, and come up with an more elegant solution. That is, we covert the merging decision

into a boundary keeping problem across all superpixel pairs. Hence, with the use of Fully Convolutional Networks, we can solve all those boundary keeping problems with just one forward pass. As a result, the second proposed *DMMSS-FCN* algorithm is not only vastly faster than the first proposed *DMMSS* algorithm, but also more accurate.

Similar to the first proposed *DMMSS* algorithms, one can obtain large quantity of training data by varying the parameters of existing superpixel algorithms to produce various superpixel boundary maps as input training data.

In this thesis, we proposed two deep-learning-based generic image segmentation algorithms that leverage the CNN and the FCN for learning generic image segmentation. Both are very effective and simulations show that they are capable of producing reliable segmentation results under many circumstances. Since we utilize several techniques to overcome the lack of training data, they do not require many human-annotated data to be fully trained. Therefore, we hope this work could offer a great idea to be implemented in the downstream tasks, since segmentation is a quite useful technique in many computer vision applications.

Reference

[1] F. Y. Shih and S. Cheng, "Automatic seeded region growing for color image segmentation," *Image and vision computing*, vol. 23, no. 10, pp. 877–886, 2005. 1

[2] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 5, pp. 603–619, 2002. 1, 5, 31, 39, 45, 59

[3] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "From contours to regions: An empirical evaluation," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2009, pp. 2294–2301. 1, 18, 23, 31, 39, 42, 59

[4] ——, "Contour detection and hierarchical image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 5, pp. 898–916, 2010. 1, 18, 23, 39, 42, 59, 61, 69, 70, 71, 72, 73, 74, 76, 77, 78, 79, 80, 81, 82

[5] T. Cour, F. Benezit, and J. Shi, "Spectral segmentation with multiscale graph decomposition," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2. IEEE, 2005, pp. 1124–1131. 1, 39, 59

[6] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 8, pp. 888–905, 2000. 1, 18

[7] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International journal of computer vision*, vol. 59, no. 2, pp. 167–181, 2004. 1, 18, 39, 59

[8] Z. Li, X.-M. Wu, and S.-F. Chang, "Segmentation using superpixels: A bipartite graph partitioning approach," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 789–796. 1, 18, 21

[9] T. H. Kim, K. M. Lee, and S. U. Lee, "Learning full pairwise affinities for spectral segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 7, pp. 1690–1703, 2012. 1, 18, 39, 59

[10] Y. Yang, Y. Wang, and X. Xue, "A novel spectral clustering method with superpixels for image segmentation," *Optik*, vol. 127, no. 1, pp. 161–167, 2016. 1, 18

[11] X. Xia and B. Kulis, "W-net: A deep model for fully unsupervised image segmentation," *arXiv preprint arXiv:1711.08506*, 2017. 1, 26, 39, 59, 69, 70, 71

[12] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1520–1528. 1, 26

[13] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440. 1, 26

[14] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017. 1, 26, 57, 67

[15] M.-Y. Liu, O. Tuzel, S. Ramalingam, and R. Chellappa, "Entropy rate superpixel segmentation," in *CVPR 2011*. IEEE, 2011, pp. 2097–2104. 5, 9, 11

[16] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "Slic superpixels compared to state-of-the-art superpixel methods," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 11, pp. 2274–2282, 2012. 5, 14

[17] V. Jampani, D. Sun, M.-Y. Liu, M.-H. Yang, and J. Kautz, "Superpixel sampling networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 352–368. 5, 14, 31, 38, 45, 58

[18] W.-C. Tu, M.-Y. Liu, V. Jampani, D. Sun, S.-Y. Chien, M.-H. Yang, and J. Kautz, "Learning superpixels with segmentation-aware affinity loss," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 568–576. 5, 9, 31, 39, 45, 58

[19] S. Xie and Z. Tu, "Holistically-nested edge detection," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1395–1403. 9

[20] D. Haehn, V. Kaynig, J. Tompkin, J. W. Lichtman, and H. Pfister, "Guided proofreading of automatic segmentations for connectomics," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 26

[21] N. Agrawal, P. Sinha, A. Kumar, and S. Bagai, "Fast & dynamic image restoration using laplace equation based image inpainting," *J Undergraduate Res Innovation*, vol. 1, no. 2, pp. 115–123, 2015. 33

[22] A. P. Kelm, V. S. Rao, and U. Zolzer, "Object contour and edge detection with refinecontournet," in *International Conference on Computer Analysis of Images and Patterns*. Springer, 2019, pp. 246–258. 36, 42, 52

[23] D. J. Field, "Relations between the statistics of natural images and the response properties of cortical cells," *Josa a*, vol. 4, no. 12, pp. 2379–2394, 1987. 36

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778. 39, 43, 65

[25] M. Donoser and D. Schmalstieg, "Discrete-continuous gradient orientation estimation for faster image segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3158–3165. 39, 59, 69, 72, 73

[26] C. J. Taylor, "Towards fast and accurate segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 1916–1922. 39, 59

[27] M. Everingham and J. Winn, "The pascal visual object classes challenge 2012 (voc2012) development kit," *Pattern Analysis, Statistical Modelling and Computational Learning, Tech. Rep*, 2011. 40

[28] P. Dollár and C. L. Zitnick, "Structured forests for fast edge detection," in *Proceedings of the IEEE international conference on computer vision*, 2013, pp. 1841–1848. 42

[29] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017. 57, 65, 67

[30] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017. 57, 67

[31] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258. 65, 67