AN ABSTRACT OF THE THESIS OF

Zheng Zhou for the degree of Master of Science in Computer Science presented on May 26, 2017.

Title: Flex-Shape Convolution and an Interactive Image Segmentation System

Abstract approved: _____

Fuxin Li

Semantic image segmentation is a relatively difficult task in computer vision. With the advent of deep learning, semantic image segmentation is increasingly of interest for researchers because of the excellent predictions from Convolutional Neural Network (CNN). However, CNNs have proven to struggle with obtaining global context of image due to convolutions being a local operation. Recent researches have proposed several global context-aware approaches: Atrous convolution, V-Net, multi-scale architecture, etc. Different from previous perspectives, this paper proposes two novel approaches to solve the problem of locating context in CNN. The first approach is called flex-shape convolution that improves the scaling and deformation capabilities of the CNN. The second approach samples auxiliary positive object pixels and negative non-object pixels for network to infer the object mask. Additionally, a web-based application of interactive image semantic annotator has been developed that allows both classical image segmentation algorithms

and deep learning algorithms to assist researchers with annotating their images for semantic segmentation.

Flex-Shape Convolution and an Interactive Image Segmentation System

by

Zheng Zhou

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented May 26, 2017
Commencement June 2017

Master of Science thesis of Zheng Zhou presented on May 26, 2017.

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Zheng Zhou, Author

ACKNOWLEDGEMENTS

I would like to acknowledge all Deep Machine Vision (DMV) group members who gave me a plenty of critical insights on the study I was working on. This thesis would not be able to accomplish without the help and guidance, abundant useful resources that are provided, from DMV group.

First of all, I would like to express my sincere appreciation to my major advisor, Dr. Fuxin Li, for the continuously mental and technological support on my study and researches during my master of science program. His personality, patience, enthusiasm, professional knowledge greatly inspired me and encouraged me to overcome the adversity in my research career at OSU. His guidance boosted me all the time to fulfill the requirements as a master student. This thesis will never be completed without aids of Dr. Fuxin Li. He is the best advisor and mentor I have ever had by far.

Besides my major advisor, I would like to appreciate the rest of my committee, Dr. Sinisa Todorovic, for his insightful computer vision course which taught me the fundamental knowledge to proceed my researches; Dr. Steven Strauss, who is the excellent professor from department of Forest Ecosystems & Society, for providing me the opportunity to work on the NSF project with his brilliant team; Dr. Joseph Louis, who comes form department of Civil and Construction Engineering, for his interest on my research and willingness to spend his time attending my defense.

Last but not least, I would like to thanks my family for their spiritual guidance throughout my life. This accomplishment would be impossible without my family.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

Chapter 1: Introduction

As human beings, it is certainly a trivial task to understand a provided scene using the vision system. However, not every visual scene can be correctly understood by everybody. To understand a scene, an observer requires prior knowledge about the information contained in the scene, which then help the observer to infer higher level information that is needed. For example, the category of the objects, the correlation between objects, the complex activities among objects, etc. The process of understanding the scene can be fundamentally reduced to extracting semantic information from the scene. Semantic information basically constitutes a scene and an image, along with what it should be semantically. Humans are able to associate the semantic information with the actual scene or image without any difficulties as long as prior knowledge has been utilized properly.

Nevertheless, from machine's perspective, everything is essentially digitized. How should machine effectively and accurately explain digital scenes or images? The discipline that researches how to make computers gain high-level understanding from digital images is called computer vision. Specifically, computer vision is primarily concerned with approaches that extract information from any format of images, including 2-D images, video sequences, multi-dimensional scenes, cameras, etc. Throughout the past years, computer vision researchers have achieved remark-

able progress on guiding computers to simulate or even surpass what human visual system can do. These researches cover various sub-tasks of computer vision, for example, object detection, face detection, edge detection, semantic segmentation, image classification, etc. Actually, most of the sub-tasks are substantially related to information extraction of image data. Despite each task having a particular objective, the process to obtain thorough representations of the data is quite similar. And this partially interprets why a learning-based method, deep learning (DL), is such a powerful technique to nicely solve a plenty of computer vision problems.

One related field that plays an important role in computer vision research is neurobiology, Specifically, the sub-domain of neurobiology that researches on biological vision system. The biological structures such as eyes, neurons, and the brains devoted to processing the visual stimuli have been broadly researched over the past century. The persistent studying on biological vision system brings a huge revolution in computer vision research. The study of neural science inspired computer scientists algorithmically and artificial neural network (ANN) has been invented, which largely impact the old-school computer vision algorithms (e.g. probabilistic graphical model). ANN is named after its artificial representation of working of a human beings nervous system. Interestingly, the birth of ANN derives a new branch of traditional machine learning (ML), which has a meaningful name, deep learning. DL uses layers of the algorithm to process data and develop abstractions, then images will be able to visually understand by machine.

The technical term "deep learning" based on the context of "artificial neural network" was presented by Igor Aizenberg and colleagues in 2000[12]. This expression perfectly and briefly concluded how an ANN models were trained and constructed. Nowadays, DL has been extensively applied to various complex computer vision tasks, for instance, image segmentation and boundary detection. The importance of image segmentation and boundary detection is emphasized by the demands of the applications that need to infer knowledge from images. The applications such as autonomous driving, image search engines, robotics, even virtual reality are highly dependent on the deeper analysis of the digital vision. DL as an advanced algorithm plays an important role in these sophisticated tasks with its comprehensive ability to learn from data. This paper will explicitly focus on DL for image segmentation and boundary detection. Specifically, we will propose the novel approaches to improve the standard DL algorithm on image segmentation and boundary detection. Besides, we will introduce an image segmentation application developed to collect data for semantic image segmentation.

1.1 Boundary Detection

Boundary detection, a.k.a. edge detection, is a computer vision task that seeks for sharp and meaningful intensity changes in images. Usually edges in an image carry out important semantic associations, for example, the boundary of the object. The rapid intensity or color variation indicates edges occurred. Detecting

edges from image source have plentiful practical applications. One imaginable case in the modern era is to find lanes in self-driving car vision system. Edge detection is considered as a non-trivial task because the there is no explicit threshold for the intensity changes to be an edge unless the objects or areas in the image are obviously distinguishable. Figure 1.1 shows an image with edge detected.

(a) Image (b) Edges

Figure 1.1: Edge detection used Canny edge detector.

1.2 Image Segmentation

Image segmentation is the procedure of separating the image into multiple meaningful parts according to the demands. With segmented image, the knowledge contained in the image will be easier to analyze and extract. Image segmentation

extracts the useful information from images by separating the objects from the background. Furthermore, semantic image segmentation, the sub-task of image segmentation, is the task to no only split the objects from the background, but also among each other. Instance Segmentation, the derivation of semantic segmentation, even asked for segmenting each instance of objects away from each other in the image. Usually, different objects or instances are labeled with different colors to distinguish. The difficulty of image segmentation truly depends on the distinguishability of the image. For example, segmenting camouflaged animals in the jungle is relatively more challenging than separating the pixels of bird from the sky. Figure 1.2 is an example of semantic image segmentation.

Figure 1.2: Semantic image segmentation that labels people as pink, bike as green.

1.3 Objective

Image segmentation and boundary detection will be the primary topic to dig into in this study. Two heuristic approaches will be presented to solve the problems existing in computer vision, specifically in semantic image segmentation and boundary detection, and propose several innovative ideas from scratch to provide other insights on the image segmentation of images. The proposed ideas will mainly focus on solving the local ambiguity problem that current convolutional neural network (CNN) is failed to deal with. In addition to this, a novel image annotation system will be introduced as well, which takes advantage of the ingenious algorithm designed from the ideas. The image annotator is a web-based application for the purpose of assisting researchers to accurately and efficiently label the raw image dataset that they wish to apply DL algorithm on.

1.4 Overview

As discussed above, image segmentation is a challenging task in computer vision. This paper will explore an algorithm to enlarge the receptive field to grab global knowledge from images without losing important information, which consequently segments the objects in the image more precisely. The algorithm is a variation of convolution which we called it as flex-shape Convolution (FSC). FSC takes the local extrema into account and only convolves the local extrema with corresponding weights. Furthermore, we utilize the revised version of sampling strategies from deep interactive object selection (DIOS)[31] to provide auxiliary information for

DL algorithm to better understand both global and local knowledge. The trained DL model with auxiliary channels will be further applied on the image annotator to segment objects by user interactions.

The remainder of this thesis is organized as follow: Chapter 2 presents the related work on our proposed methods. Chapter 3 briefly introduces the fundamental mathematical theories of CNN and discuss the potential drawback of CNN in detail. With these in mind, FSC, the redesigned CNN, will be presented with both the effect and mathematical calculation behind. The results applying FSC on BSDS500 boundary detection[1], which is a standard benchmark for boundary detection will show at the end of the chapter as well. Chapter 4 explores an entirely different strategy to assist computer understanding global information of images. By fine-tuning pre-trained DeepLab model with manipulated image data, the model will be able to segment foreground (object) and background precisely. The manipulated image data consist of two more channels that construct by positive samples and negative samples sampled with the redesigned strategies in DIOS[31], which serve as complementary information for DL model to learn the object-level context. Chapter 5 illustrates the development of image annotator user interface deployed as a web application. The image annotator integrates our DIOS and refined GrabCut[22] algorithm together to enhance the flexibility of the UI in which users have their options to choose the algorithms. The image annotated in the system can be further used to train the network, which abstractly composes a self-sufficient ecosystem. We will finally sum up on chapter 6 and discuss the

future works on this research.

1.5 Contributions

Our key contributions are summarized below:

- We propose a simple yet effective Flex-Shape convolution algorithm, which more effectively captures the global information, as an alternative to normal CNNs that are considered to struggle with interpreting global context information.
- We fine-tune the DeepLab-ResNet[3][4] pre-trained model for object segmentation on our five channels image data which consist of two extra sampling channels, and evaluated the model on PASCAL VOC 2012 validation dataset.
- We develop an image annotator system containing DL algorithm and many useful modules to help researchers label raw image data with semantic information more flexibly and accurately.

Chapter 2: Literature Review

2.1 Classification Model

In 2009, Fei-Fei Li, who is an AI professor at Stanford introduced ImageNet[5]. ImageNet is a free dataset of more than 14 million labeled images while most of the images on the Internet were raw and unlabeled. The advent of ImageNet potentially provided ample "food" for the neural network to consume. As a data-driven learner, DL would gain much more improvements with the support of ImageNet. ImageNet also offered the opportunity for researchers to compete against each other in large scale visual recognition challenge[25] (ILSVRC). The ImageNet ILSVRC evaluates models for object detection and image classification at large scale. ILSVRC continuously resulted in the innovation of sophisticated DL models. A variety of excellent ideas on DL were proposed as time goes on.

AlexNet developed by Alex Krizhevsky et al.[16] won the challenge in 2012 by decreasing the top-5 error to 16.4%, even could reduce to 15.3% with extra data, which surpassed the second place error rate 26.3% by significant amount. AlexNet had basically similar architecture as LeNet[17] but with more layers (8 layers) and deeper structure. The success of AlexNet significantly increased in popularity of CNNs. Not surprisingly, the winner of ILSVRC 2013 was a variation of CNN as

well known as ZFNet, which achieved top-5 error rate 14.8%. Specifically, ZFNet had same structure as AlexNet with finer tuning on the hyper-parameters. The winner of ILSVRC 2014 competition was GoogLeNet[28]. It improved top-5 error rate down to 6.67% which was extremely closed to human-level performance 5.1%. The distinctive part of GoogLeNet was the sub-module embedded in the architecture known as inception module. This module drastically reduced the number of parameters needed with a block of multiple small convolutional layers, for the sake of building a deeper network while still keeping the reasonable training speed. Compared with AlexNet which had 60 million parameters, GoogLeNet only had 4 million parameters. Even though GoogLeNet won the ILSVRC 2014, The runner-up VGGNet by Simonyan and Zisserman[27] achieved top-5 error rate 7.3% was definitely worth mentioning. VGGNet consisted of 16 convolutional layers (another variation composed of 19 convolutional layer called VGGNet19) with nicely uniform architecture. It only applied 3×3 convolution and 2×2 pooling all the way down to the output layer. Such configuration making VGGNet become perfect choice for extracting features. One potential problem of VGGNet was the large number of parameters, which reached to 140 million. Lastly, the ILSVRC 2015 in December, the so-called residual neural network (ResNet) was presented by Kaiming He et al[11] which achieved amazing top-5 error rate 3.57%. ResNet introduced a novel architecture which took advantage of the gated unit technique used in recurrent neural network (RNN) reformed as the skip connection between a set of convolution blocks. Skip connection served as a memorization purpose for deeper layers to memorize previous output of convolutional layers. Above CNN models

greatly boosted the development of DL in the sense that their pre-trained models can be repeatedly used by new architectures. In recent years, these proven classification architectures have been effectively applied as encoder to extract features for more complex tasks, semantic image segmentation and boundary detection, for instance.

2.2 Prior Work of Boundary Detection

AlexNet developed by Alex Krizhevsky et al.[16] won the challenge in 2012 by decreasing the top-5 error to 16.4%, even could reduce to 15.3% with extra data, which surpassed the second place error rate 26.3% by the significant amount. AlexNet had the similar architecture as LeNet[17] but with more layers (8 layers) and deeper structure. The success of AlexNet significantly increased in popularity of CNNs. Not surprisingly, the winner of ILSVRC 2013 was a variation of CNN as well known as ZFNet, which achieved top-5 error rate 14.8%. Specifically, ZFNet had the same structure as AlexNet with finer tuning on the hyper-parameters. The winner of ILSVRC 2014 competition was GoogLeNet[28]. It improved top-5 error rate down to 6.67% which was extremely close to human-level performance 5.1%. The distinctive part of GoogLeNet was the sub-module embedded in the architecture known as inception module. This module drastically reduced the number of parameters needed for a block of multiple small convolutional layers, for the sake of building a deeper network while still keeping the reasonable training speed.

Compared with AlexNet which had 60 million parameters, GoogLeNet only had 4 million parameters. Even though GoogLeNet won the ILSVRC 2014, The runner-up VGGNet by Simonyan and Zisserman[27] achieved top-5 error rate 7.3% was worth mentioning. VGGNet consisted of 16 convolutional layers (another variation composed of 19 convolutional layers called VGGNet19) with nicely uniform architecture. It only applied 3×3 convolution and 2×2 pooling all the way down to the output layer. Such configuration was making VGGNet become a perfect choice for extracting features. One potential problem of VGGNet was the large number of parameters, which reached to 140 million. Lastly, the ILSVRC 2015 in December, the so-called residual neural network (ResNet) was presented by Kaiming He et al[11] which achieved amazing top-5 error rate 3.57%. ResNet introduced a novel architecture which took advantage of the gated unit technique used in the recurrent neural network (RNN) reformed as the skip connection between a set of convolution blocks. Skip connection served as a memorization purpose for deeper layers to memorize previous output of convolutional layers. Above CNN models greatly boosted the development of DL in the sense that their pre-trained models can repeatedly be used by new architectures. In recent years, these proven classification architectures have been effectively applied to the encoder to extract features for more complex tasks, semantic image segmentation, and boundary detection, for instance.

2.3 Prior Work of Image Segmentation

Likewise, image segmentation obviously benefits by DL as well. There were plentiful works that tried to intelligently generalize image segmentation such that the generalization error of algorithm could be in a reasonable bound. Before DL, a plenty of probabilistic graphic model and clustering algorithms have been applied to image segmentation problem, such as conditional random field (CRF), K-means, graphcut, GrabCut[22], etc. GrabCut estimates the color distribution of the target object and background by using Gaussian mixture model. It is primarily a maximum a posterior algorithm used to solve CRF over the pixel labels. This algorithm will be refined and applied in our work. With DL algorithm, image segmentation becomes a pixel-wise prediction problem by constructing the DL networks. Currently, the most common and fruitful segmentation DL architecture is the fully convolutional network (FCN) developed by Long et al.[18]. FCN capitalized the well-known classification DL models that described above to learn the hierarchies of image features. It reconstructed AlexNet[16], VGGNet[27], GoogLeNet[28] and ResNet[11] to the fully convolutional hierarchy by substituting fully connected layers for convolutional layers in order to compute a score maps for pixels instead of a single class score. The maps would then deconvolve to the same size as input to finalize pixel-wise classification result. FCN is respected as the milestone that trained an end-to-end dense pixel predictor for semantic segmentation. Albeit FCN achieved notable results, the problem hiding behind image segmentation using FCN should not be neglected: FCN did not consider global context knowl-

edge from the image. Specifically, vanilla CNN itself were not object-awareness and able to balance the local and global knowledge due to linearly stacked convolutional layers. DeepLab models[3][4] tried to solve this potential problem by taking advantages of several techniques, for example, atrous convolution[33] and denseCRF[15], to fetch global information. Atrous convolution could be considered as normal convolution with exponentially expandable gaps. These gaps effectively amplify the receptive field without losing resolution, which somehow globalizes the output features. DenseCRF served as a network-independent module that is a post-processing stage to refine the segmentation result of the network globally. There were abundant researches putting efforts on improving the performance of segmentations and integrating the global context information, for example, multi-scale image pyramid prediction[20] [23]. Current reported state-of-the-art model on semantic segmentation reaches 80.6% IoU[1] by a Reset-based architecture[29]. While semantic segmentation achieved great success, the study on instance segmentation barely got expected outcomes. The automation process of this task is ambiguous in the sense that the number of instances is originally unknown in images and it is hard to perform pixel-wise estimation in the same class. The best performance reported for now on instance segmentation is Mask R-CNN[9] with 32.0% AP. Mask R-CNN is extended from Faster R-CNN[21], which added a branch for the purpose of predicting object mask in parallel with the bounding box and class predictor. Most of the architectures mentioned above depend upon

[1]IoU gives the evaluation of performance, commonly known as PASCAL VOC intersection-over-union metric

classification models as feature extractors and are restricted by the performance of those models.

Overall, image segmentation is considered to be a very challenging research topic that requires deeper exploration of both data and architectures. Additionally, it is crucial to prepare higher quality and more general data for DL, a data-driven algorithm, to improve the performance of models.

Chapter 3: Flex-Shape Convolution

3.1 Overview of Convolution and CNN

Mathematically speaking, convolution is an operation that applies to two functions by integral of point-wise multiplication, after that producing a third function which is essentially a linear operator. In computer vision, convolution mostly operates on discrete domain since the intensity of the image is usually discrete. Owing to the integral or summation of point-wise multiplication, principally convolution is a linear operation.

As the core unit in CNN, the linearity of convolution is not sufficient to accomplish the complex task. Hence convolution blocks (convolution-pooling) in CNN usually introduce non-linearity into the network. Non-linear activation function and pooling layer are inserted as a major non-linear setup of the convolution block. Regarding backpropagation, the added non-linear activations have to be differentiable. The most popular differentiable activation functions are ReLU, hyperbolic tangent and sigmoid. Compared to other functions, ReLU is preferable despite it is not differentiable at zero, which can be solved by computing sub-gradient. In next section, we will go through in detail about convolution and CNN in more formal manner.

3.1.1 Definition

Suppose * stands for convolving, the convolution operator for function **x** and **w** is defined by:

$$(x * w)(p) = \int x(i)w(p-i)di \tag{3.1}$$

The discrete convolution is defined as:

$$(x * w)(p) = \sum_i x(i)w(p-i) \tag{3.2}$$

in which cases **p** is the output index, **a** is the input index. This is actually 1D convolution formula. In DL terminology, the convolution learnable parameters **w** are often referred as the **kernel**, and the output is referred to the **feature map**. The formula tells the story that convolution no more than integral (continuous) or summation (discrete) of point-wise multiplication within the kernel range. $p - i$ is apparently the index of the kernel. Since we are considering in the computer vision field, the convolution will only refer to discrete convolution. With these, 2D convolution is derived as:

$$(x * w)(i,j) = \sum_m \sum_n x(m,n)w(i-m, j-n) \tag{3.3}$$

where **(i, j)** are 2D index of output and **(m, n)** are valid 2D index of input within the kernel size. In CNNs, the input is mostly a tensor[1], even image is a tensor with three dimensions (width, height, RGB channels). Thus the true story happened in the convolution block of CNN is slightly more complicated, which we can write as:

$$(x * w)(i, j, c') = \sum_m \sum_n \sum_c x(m, n, c) w(i - m, j - n, c, c') \tag{3.4}$$

where **c** is the index of input channel and **c'** is indexing output channel. The convolutional layers or general speaking the layers in CNN have the 3D volume of input and output neurons, which organize as width, height, and channel. In fact, convolution through convolutional layer will work on entire input volume by sliding the kernel window with certain stride at a time to result in a complete output channel. From this manner, the kernel for each output channel is shared across the process of sliding one input volume, which means the same high-order filter is used for each pixel patch in the layer. The kernel can be considered as 3D volume as well so that the kernel weights are usually combined with different values. Sharing weights across the sliding process is one of the major advantages to using convolutional layer rather than the fully connected layer in DL for computer vision problem. It not only significantly reduces the total number of parameters which is memory-saving and time-saving, but also hugely improves the performance as a result of considering spatial context. The fully connected layer will never take spatial information into account where it is pixel-to-pixel mapping. Figure3.1 con-

[1]tensor is merely multi-dimensional array

cisely demonstrates how it really works. Different color patches are convolved by

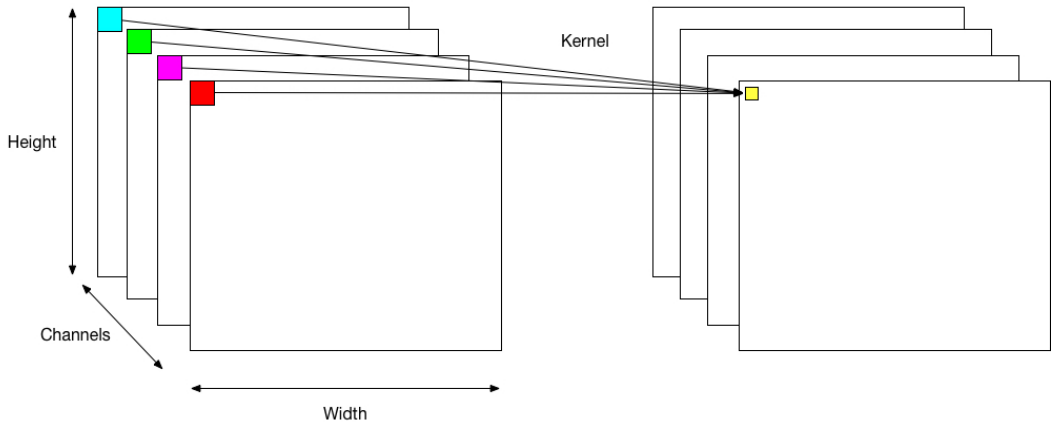

Figure 3.1: How Convolutional layer works on 4 channels input

kernel and eventually expressing on a single output neuron.

For the sake of breaking the linearity of the convolutional layer to learn more complex function than just linear regression, non-linear activation functions are introduced after convolving. Most of modern CNN models heavily rely on ReLU to activate computed feature where this has been proven efficient without downgrading the generalization accuracy. ReLU has the relatively simple form that takes an input and output either the original input or zeroes if the input is less than zero. Mathematically it can be expressed as follow:

$$f(x) = max(0, x) \tag{3.5}$$

where $\mathbf{x}$ is the input of ReLU. Figure 3.2 is the plot of ReLU near $x = 0$. This

function transforms all negative response to zero so that large response will be activated. It merely silences the negative response and never affects the receptive fields of the convolution layer.

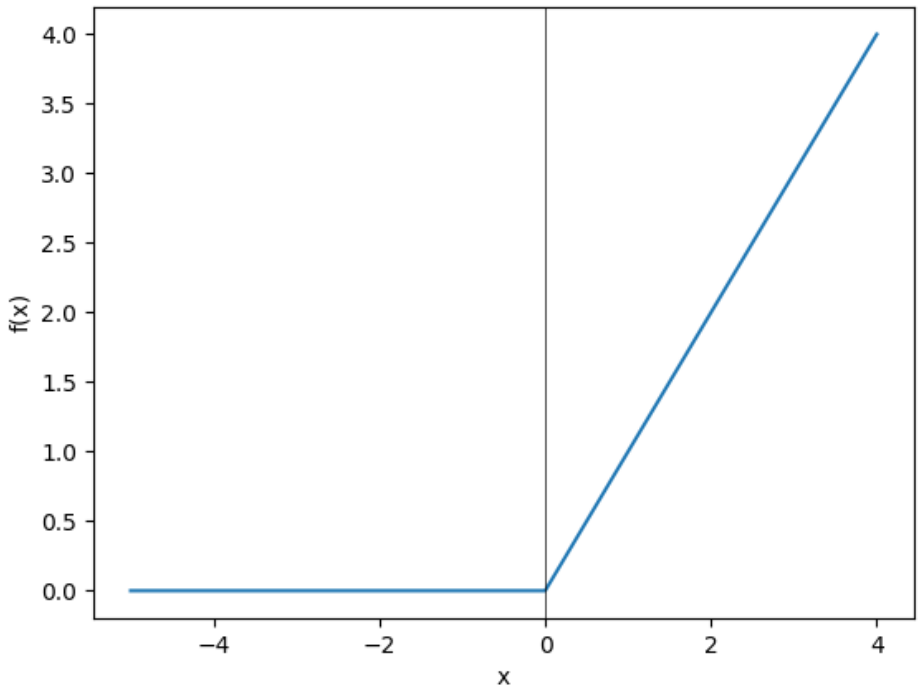

Figure 3.2: ReLU function

Pooling layers are another important part in CNNs. The most accessible strength of pooling layer is that it halves the spatial size of the feature map from the previous layer to reduce the number of parameters for the fully connected layer and accelerate the computation of network. It is important to note that pooling

layer does not reduce for convolutional layer because the number of parameters of the convolutional layer does not depend on the number of input neuron, it only depends on how many filters to convolve on the input. There are multiple types of pooling operations such max pooling, average pooling, the region of interest (RoI) pooling[8], etc. The most commonly used one is max pooling which takes the maxima from pooling window and discards the rest of activations. In the following contents, we will refer max pooling as pooling for simplicity. Pooling operates independently on every channel of input. Pooling layer constitutes the property of translation invariance by spatially downsizing the input in which the appearance of RoI still proportionally locates at the same position. Given pooling filter size $p \times p$ and stride s, the final output volume size for the input with size $w_i \times h_i \times c_i$ can be calculated as:

$$w_o = \frac{(w_i - p)}{s} + 1 \tag{3.6}$$

$$h_o = \frac{(h_i - p)}{s} + 1 \tag{3.7}$$

$$c_o = c_i \tag{3.8}$$

where the output volume has dimension $w_o \times h_o \times c_o$. Figure 3.3 shows the pooling operation on volume and one channel.

Convolutional layer, ReLU activation function, and pooling layer consist of the principal component of CNNs. Most of the modern CNN models are constructed by stacking these fundamental units, for example, VGGNet[27]. Undeniably, there

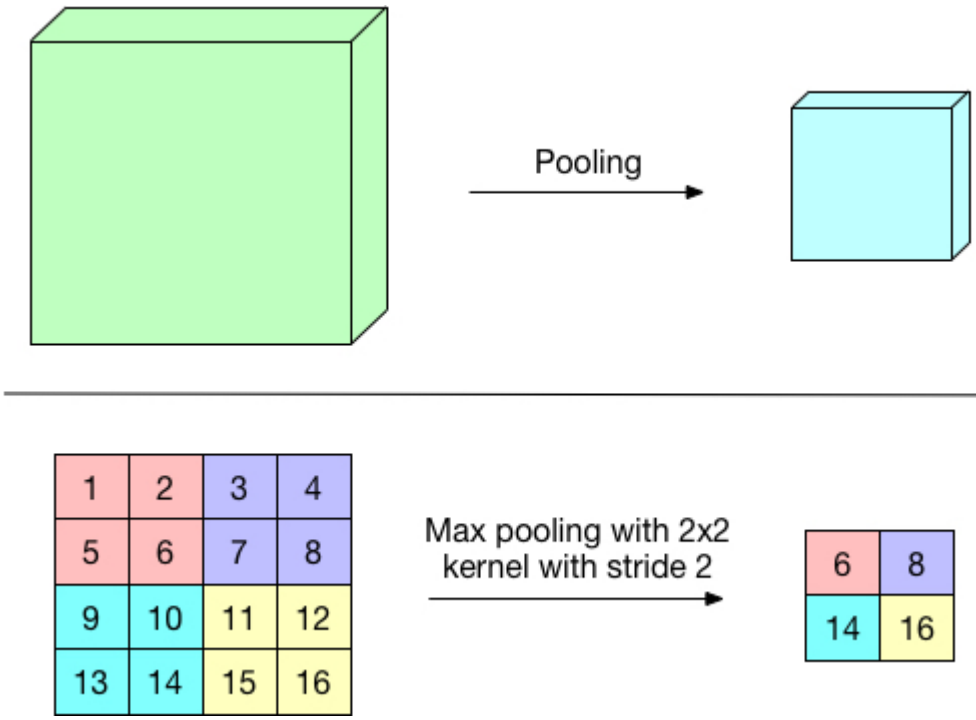

Figure 3.3: Pooling layer on the top and max pooling operation below.

are plentiful models that contribute more complicated architecture than the simple components above, yet they can never avoid to take advantage of the convolution blocks as image feature extractor in computer vision. In next part, we will walk through a complete process of computation in CNN.

3.1.2 Feed-forward and Back-propagation

There are two primary steps for completing a training process from one sample or mini-batch, Feed-forward, and back-propagation. Feed-forward, as the name self-

defined, is the procedure to feed input into network forward till the ending layer, in most case, loss layer to compute the loss (also called cost) of the prediction. It merely embeds the functions in network one by one. Given a cross entropy cost function below:

$$\mathcal{L}(x, y) = -\frac{1}{n}\sum_{i=1}^{n} y_i \ln a(x_i) + (1 - y_i)\ln(1 - a(x_i)) \tag{3.9}$$

where x, y are the input from the previous layer and ground truth respectively. $a(x_i)$ stands for the output of the activation function for ith input. x itself is an embedded function which is composed of all the functions before calculating x. For example, x computed from the pooling layer before, and the output of pooling layer is computed by the output of convolutional layer, etc. After the loss has been calculated, the back-propagation will start.

The back-propagation algorithm was initially introduced around 1970, but it does not become popular until 1986[24]. That paper described several neural networks where back-propagation works much faster than previous approaches to learning, which provided the possibility to solve the problem by the neural network. Back-propagation works with stochastic gradient decent (SGD) and chain rule of calculus to compute the derivative of parameters for the purpose of updating parameters, and backpropagate the error to the previous layer for computing the derivative of parameters of previous layers. Mathematically, for a loss function

$\mathcal{L}(X, Y)$, we calculate the amount of weight update of layer l by:

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial w^l} = \frac{\partial L(\mathbf{w})}{\partial f^L} \frac{\partial f^L}{\partial x^L} \frac{\partial f^{L-1}}{\partial x^{L-1}} ... \frac{\partial f^l}{\partial w^l} \tag{3.10}$$

where f, x a block of gates that contain all possible layer such as pooling and ReLU, and w^l is the weight in layer l. This equation can be further simplified as

$$\frac{\partial L(\mathbf{w})}{\partial w^l} = \epsilon \frac{\partial f^l}{\partial w^l} \tag{3.11}$$

while ϵ is the error propagated from previous layers. With the quantity of updates, classic SGD changes the weight of layer l by:

$$w^l = w^l - \eta \frac{\partial L(\mathbf{w})}{\partial w^l} \tag{3.12}$$

while η is learning rate to adjust the learning speed. SGD is not the only way to update weights. There are abundant algorithm works even faster and more effective than SGD in DL such as Adagrad[6], Adam[14], Adadelta[32], etc. But most of them are gradient-based learning approaches.

The algorithm starts with iteratively going through above two steps until the network converges, which means the loss can not be improved anymore and bounce back and forth in a certain range. For CNNs, above algorithm gives sufficient chances to learn the feature in the dataset with a few parameters compared to the fully connected neural network. This potentially practicalizes the deeper architec-

ture in which the features extracted are more expressive.

3.1.3 Drawback

Looking back to the calculation of convolutional layer and pooling layer, they heavily rely on the localized filters with certain kernel size to obtain image features. Regarding image segmentation, on the one hand, local information is crucial in the sense that it provides better pixel-level accuracy. On the other hand, in the restricted receptive field, local information can be extremely ambiguous. The downside of normal CNNs is that it barely considers the diverse spatial scales of segments. Since objects in an image usually are multi-sized, it readily imagines that fixed local information will not precisely segment the objects. Pooling layer has some degree of spatial invariance. But with losing some information in the local range, pooling layer will become ambiguous when the network is stacking up.

FCN[18] is a typical example of such CNN. As Hyeonwoo Noh et al. pointed out[19], FCN only handles on a single scale semantics within image due to the fixed-size receptive field, which can be problematic on considerably larger or smaller objects than the receptive field. figure 3.4, 3.5 give the example of mislabeled and fragmented on the large and small objects.

While most of the works concentrate on fixing the intrinsic issues of CNNs by innovating new architectures such as Deconvolution network[19], or modifying

Figure 3.4: Incorrect predictions on large size object.

Figure 3.5: missing predictions on small size object.

existing architectures to adapt multi-scale recognition like multi-scale convolutional architecture[20] and four multi-scale CNNs[23], we will attempt to work on the computation of convolution and pooling themselves to achieve some degree of scale invariance. Following sections will discuss the details about the Flex-Shape Convolution that we introduced to be an alternative to the conventional convolution-pooling unit.

3.2 Introduction of Flex-Shape Convolution

We proposed flex-shape convolution (FSC) which is a variant of Convolution that is expected to resolve above problems that CNNs suffer. The intuitions behind FSC are quite straightforward. We desire to obtain fine-grained pixel-level information but do not miss the context information in the global extent or larger receptive field. More generally, we want to correctly classify each pixel in either large object or small object within the image. The basic idea of FSC is that we extract the most significant value in a range such that each pixel in an original image map to the maximum and minimum value in that range based on the sign of the value in filters. By doing so, we can get a maximum mapping and a minimum mapping of the initial image. When convolving the image, we simply find the corresponding mapping value according to the sign of the weights and summing the multiplications. Consequently, with such flexed way to convolve feature map, the features are more expressive where the shape and scale of objects are invariant. And that is why we call it flex-shape convolution. By introducing shape-invariant and scale-invariant, FSC performs significantly better than standard convolution in recognizing objects which have only fixed size receptive field.

dilated convolution[33], a.k.a. Atrous convolution, has similar effects as our FSC. As we described in the background section, dilated convolution introduced gaps among each position within the kernel. According to the dilation rate, dilated convolution supports exponentially expanding receptive fields. However, the

expanded receptive fields are always squared as the result of the squared kernel, which potentially mismatches object shapes in many situations. Unlike the dilated convolution, even though FSC is using squared kernel, the way it generates max-mapping and min-mapping allows FSC to match most of the non-squared shapes and amplify the receptive field as well. In the next section, we will go into the detail of FSC mathematically to discover the secret of shape-invariance and scale-invariance properties.

3.3 Mathematical Formula

In this section, we will separately construct the formula of feed-forward and back-propagation of FSC. The feed-forward in FSC can be thought as typical convolution except the input data will be chosen from max-mapping and min-mapping of original inputs. The max-mapping and min-mapping are computed by:

$$x^{max}(i,j,c) = \max(x(i-m, j-n, c)...x(i+m, j+n, c)) \tag{3.13}$$

$$x^{min}(i,j,c) = \min(x(i-m, j-n, c)...x(i+m, j+n, c)) \tag{3.14}$$

where $m \times n$ is the flex kernel size. These mappings will be stored in the temporary memory and freed after all computations are completed. Subsequently, mapping recipes are fed into the convolution operation with slight difference from

the formula 2.4, which is feed-forward step and has following form:

$$\hat{y}(i,j,c') = \sum_{abc} \begin{cases} x(a,b,c)^{min} w(i-a, j-b, c, c') & for\ w(i-a, j-b, c, c') \leq 0 \\ x(a,b,c)^{max} w(i-a, j-b, c, c') & for\ w(i-a, j-b, c, c') > 0 \end{cases} \tag{3.15}$$

while $i-a, j-b$ are the spatial index of kernel centered at 0, and $\hat{y}$ is the output of FSC. Thus it is possible that network tries to retrieve both min-mapping and max-mapping in a single output neuron. The reason that FSC multiplies the negative weight with min-mapping and positive weight with max-mapping is to activate large response as much as possible and deactivate the passive response, namely, negative values. By doing so, the features that are extracted from FSC will be more representative. It can produce smoother result as well where deactivated values somehow balance the activate large response.

In the original CNN, we use the chain rule to back-propagate error down to each layer and update weights by SGD. Likewise, we can also apply chain rule and SGD to perform back-propagation. In formula 2.10, we have a term $\frac{\partial f^l}{\partial w^l}$ which is the derivative for the layer where weights will be updated. This term is reduced to x^l. Since in FSC, there are two input mappings, hence this term will be retrieved

from mappings instead of the initial input map. Mathematically, we have:

$$\frac{\partial f^l}{\partial w(i,j,c)^l} = \begin{cases} x(i,j,c)^{min} & for\ w(i,j,c)^l \leq 0 \\ x(i,j,c)^{max} & for\ w(i,j,c)^l > 0 \end{cases} \tag{3.16}$$

The error back-propagated layer-by-layer remains the same as a normal CNN since they have no dependencies with their activation values. Figure 3.6 graphically illustrate how FSC works. The min-max mapping convolution imports some level of balancing into network which avoid large input excessively dominating the output. Next section will exhibit results to better illustrate FSC.

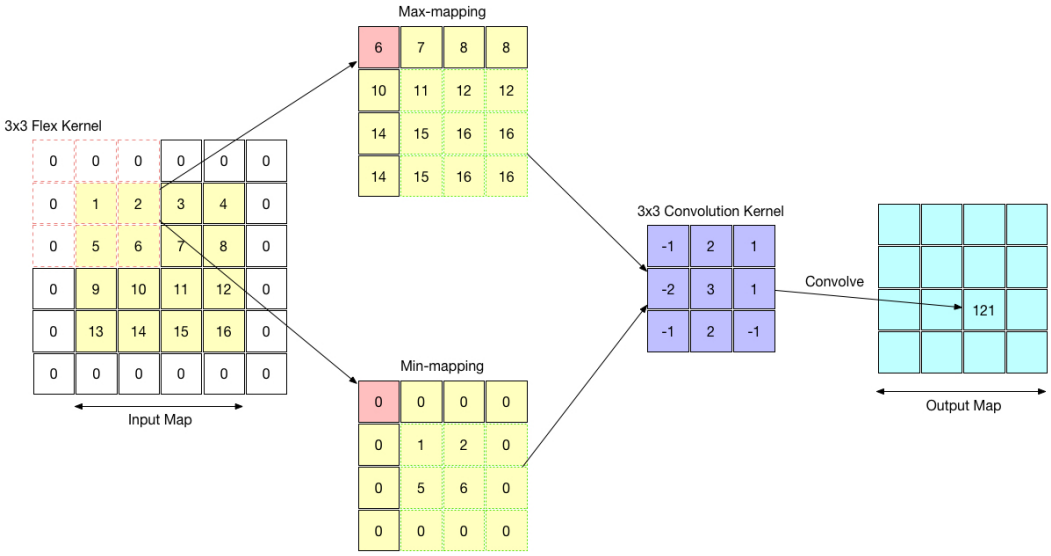

Figure 3.6: Feed-forward of FSC. Write zero rectangles are paddings. Red dash rectangle area on the left represents flexing area and it is using 3×3 kernel size. Green dot area is an example area to convolve based on the rule we defined above.

3.4 Implementation and Result

We implemented the FSC on Caffe DL framework[13] with both CPU and GPU CUDA versions. Caffe is a fantastic DL framework that has strong modularity and fast speed. We extended the layer interface of Caffe by adding our FSC layer. For the demonstration purpose, figures 3.7 shows the effect of min-mapping and max-mapping writing in Numpy version. As we expected, the max-mapping tries to brighten the picture while min-mapping tends to darken the picture. By applying in the neural network, process the lower level feature maps with FSC will focus more on the activated neurons and balance the activated neurons by a certain amount that is related to the deactivated neurons. Also, the values in two mappings come from arbitrary positions within the flexing area that meets the rules, which potentially infers the shape of the target object.

Furthermore, we were benchmarking FSC on the most famous boundary detection dataset BSDS500 to figure out how better it is to perform binary segmentation. Boundary detection is also one kind of image segmentation which has only two categories for each pixel: edge and not the edge. We constructed two different network architectures as shows in figure 3.8, 3.9 and trained on the BSDS500 training dataset. The first network is stacking the convolutional layer and flex-shape convolutional layer one by one to extract both local information and larger global context information with flexed shape. We trained the network from scratch with 20 epochs using cross entropy loss function.

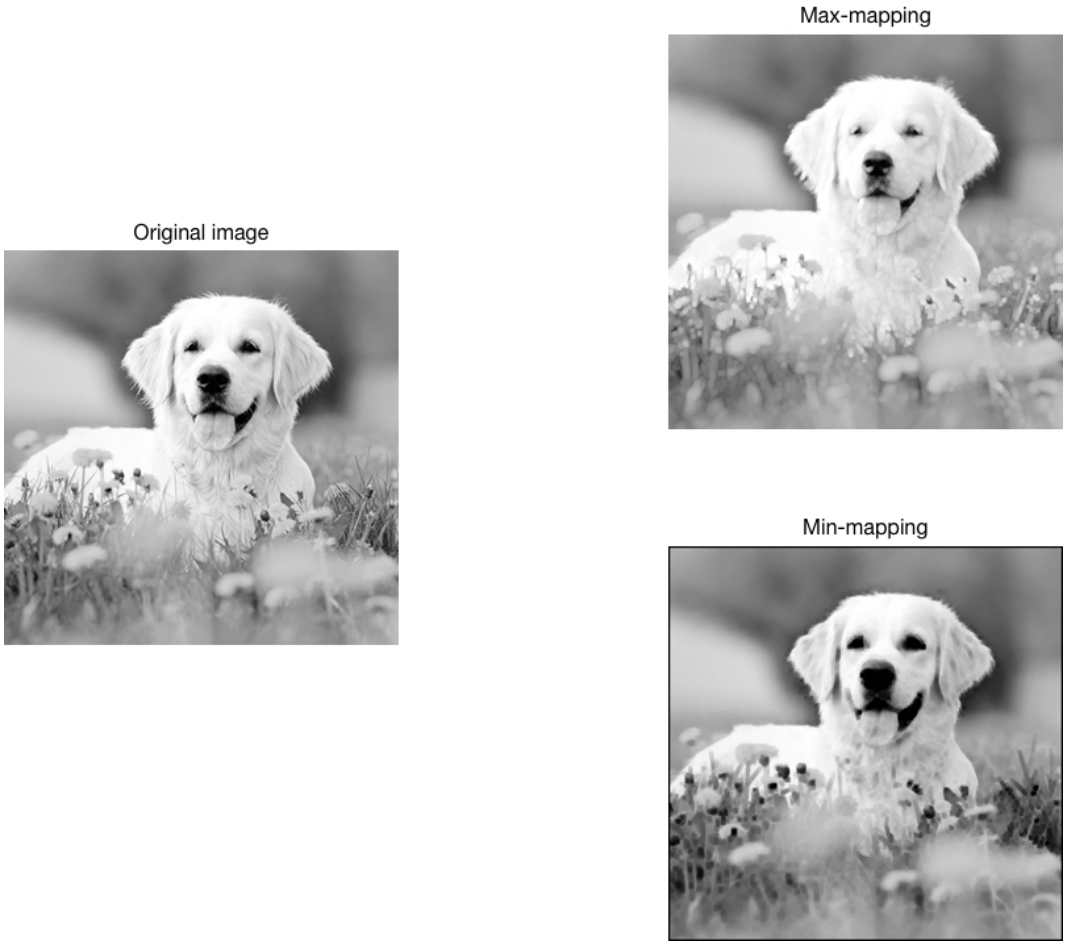

Figure 3.7: 5×5 flex kernel to obtain mappings.

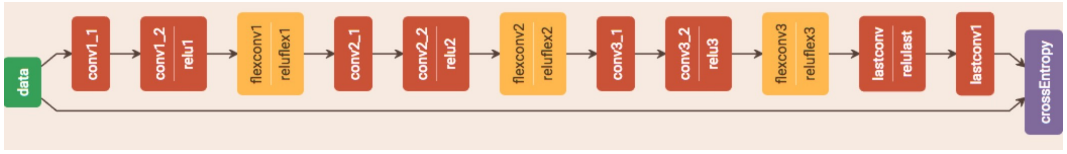

Figure 3.8: Stacked architecture with 3 flex-shape convolutional layers.

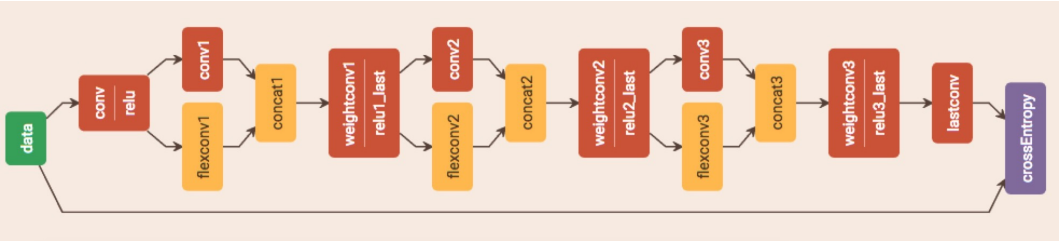

Figure 3.9: refined architecture that concatenate 1 by 1 convolutional layers together with the result of flex-shape convolutional layers.

Considering that the edges are comparatively sparser than the background, we weighted the positive loss ten times more than the negative loss, in other words, the model will punish more on falsely predicting the edge pixels than non-edge pixels. We did not insert any pooling layer as FSC is expected to perform the effect of pooling layer. Since there is no any 2-stride operation, the resolution will never be reduced. To this end, we can not apply deep architecture due to the comparatively larger number of parameters. Figure 3.10 gives the loss plot of 4000 iterations where 200 iterations are 1 epoch. From the plot, it is clear that the network is converged around 1200 iterations, which proves the correctness of mathematical formulas. We test the trained model on BSDS500 testing set, figure 3.11 is the result from one random example. The problem we observed from the result is that edge lines are isolated to two directions which result in a gap between two lines. By checking the value map, we found that the gap holds a fairly large value, yet it is not large enough compared to the two isolated lines. This problem happened because after the edge is detected by the certain level of flex-shape convolutional layer, applying more flex-shape convolution will transit

the large intensity from the center edge of two other sides. Meanwhile, the small intensity will affect the large response of the center edge from min-mappings.

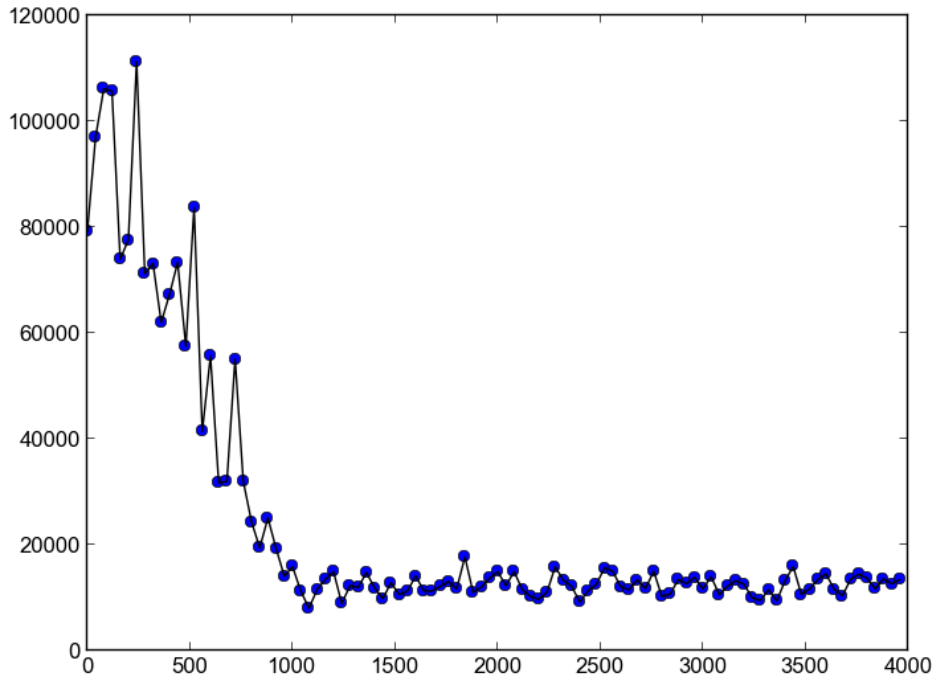

Figure 3.10: Training loss on BSDS500 with 4000 iterations (20 epochs). x-axis is the number of iterations while y-axis is the total training loss.

By means of observing this problem, we invented another architecture which demonstrates on figure 3.9. This architecture resolved the problem appeared in the first network by adding a parallel 1×1 convolutional layer concatenated with flex-shape convolutional layer to enhance the center pixels. Figure 3.12 shows the

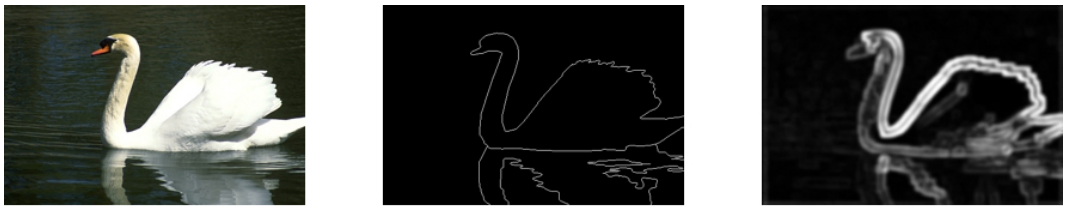

Figure 3.11: (a) Original image; (b) edge ground truth; (c) our edge detector result.

result of the same example. As the result showed, the gap in the center is disappeared and the edge is much clearer. The blurred effects in both results are expected because of the balancing calculation of min-max mapping. For the sake of comparison, we trained a stacked convolutional neural network with four convolutional layers and same configuration as the refined network on 20 epochs, which produces non-ideal result that shows on figure 3.13. Most of the edges are totally missed from the output layer. One of the reason might be the inadequate training process, the other reason is that purely convolutional layer focuses two much on the local receptive field. Consequently, most of the edge information are lost.

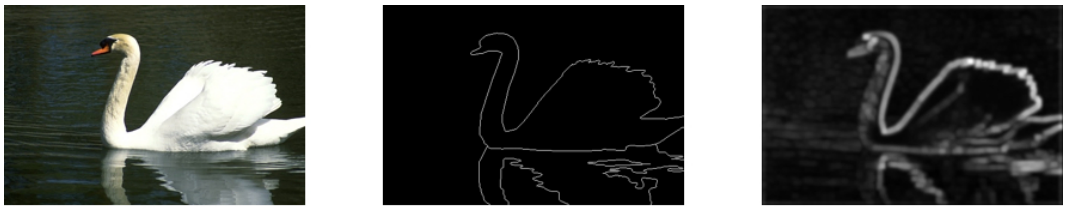

Figure 3.12: (a) Original image; (b) edge ground truth; (c) our edge detector result.

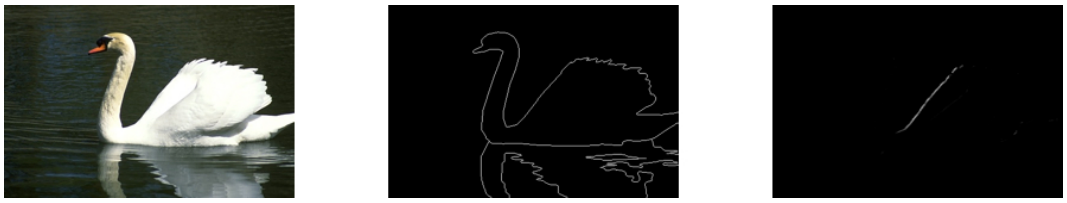

Figure 3.13: (a) Original image; (b) edge ground truth; (c) pure CNN edge detector result.

By testing the whole testing dataset on both detectors, table 3.1 shows quantitative results on CNN detector and FSC detectors along with several popular edge detectors. The HED[30] produces state-of-the-art result which even beats human level perception. Our refined FSC had similar performance as Canny edge detector[2] but got better performance than stacked CNN.

Detector	ODS	OIS	AP
HED	**0.782**	**0.804**	**0.833**
Canny	0.600	0.640	0.580
gPb	0.700	0.720	0.660
FSC-refined	0.569	0.602	0.572
CNN	0.430	0.454	0.412

Table 3.1: Boundary benchmark on BSDS500. Shown are the F-measures when choosing an optimal scale for entire dataset(ODS) or per image(OIS), and the average precision (AP)

In conclusion, FSC explored a new approach to resolving the localization problem that CNN intrinsically exists. It achieves this by applying min-max mappings on the convolutional kernel, which somehow automatic deformed the receptive field. Then it smoothers the extrema by neutralizing the large response with nega-

tively weighted small response to prevent over-focus of large response. If the small response is negative, it will, in turn, try to emphasize the small response a little bit for re-concentrating the parts that lack attentions. With larger receptive field, FSC can obtain global context in some degree by only changing the computation of convolutional layer itself instead of revising the entire network architecture, which is potentially useful in the future for the tasks that heavily require scale and shape invariance. Benchmarking FSC on BSDS500 visually reveals its functionalities. In the next chapter, we will introduce another method to gain global information through training a new model.

Chapter 4: Deep Sampling Object Segmentation

In the previous chapter, FSC put the break-point of local ambiguity and square-shaped problem within the convolutional layer and pooling layer, which achieved certain improvement regarding performance and visualization. Starting from this chapter, we will introduce another completely different approach to tackling the problem that exists in CNNs, or more specifically, FCN-like models.

The problem we will work on is object segmentation, which can also be considered as a binary segmentation problem: object and non-object. In this problem, we will be no longer interested in what the class of the object is, instead, figuring out the objectiveness of pixels will be the major task for predictor. The objectiveness predictor has many high-level applications. In this study, we trained an objectiveness predictor for user interactive segmentation, which will describe in chapter 4. It is even potentially useful for unresolved problems such as for instance segmentation. Following sections discuss the base model we employed and the methods that we attempted to encapsulate objectiveness context information into the network. Eventually, the last section exhibits the results that we got from proposed approach.

4.1 Overview of DeepLab-ResNet and Deep Interactive Object Selection

4.1.1 DeepLab-ResNet

While semantic segmentation world exists a plenty of excellent models in DL field, DeepLab-ResNet is no doubt to be one of the leading performance architecture. The DeepLab-ResNet is built on a fully convolutional variant of ResNet-101 with using dilated (atrous) convolutions that we discussed before and applied atrous spatial pyramid pooling[10] for multi-scaled inputs. We leveraged the re-implemented open source project from tensorflow-deeplab. This implementation trained the model with mini-batch and concatenate 21 classes softmax classifier at the end of the network. In this implementation, the inputs are not multi-scaled. Network downsampled the input to the size matching the output of the network, which is 321×321. At the time of inference, bilinear upsampling has been applied to obtain the output with size as input. The model achieved 79.7% IoU.

The pre-trained model of DeepLab-ResNet can extract very expressive features because of the application of several techniques to capture global and class-level information, for example, as we discussed before, atrous convolution and atrous spatial pyramid pooling. Figure 4.1, 4.2 shows more details about these operations. With the pre-trained parameters where semantic information is richly embedded, providing auxiliary information will be greatly helpful for detecting objects pixel-

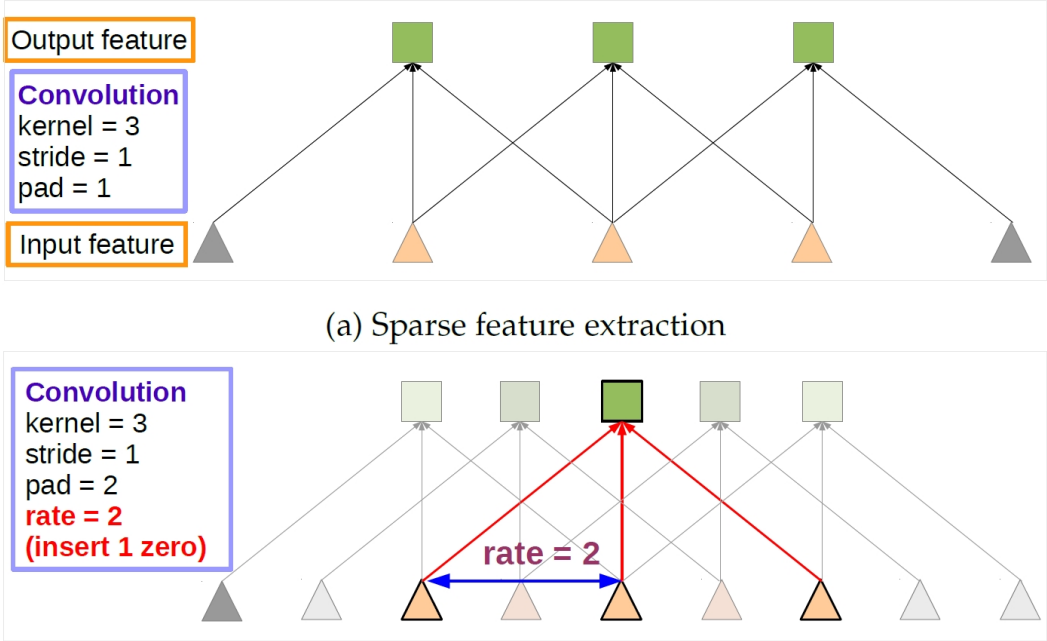

Figure 4.1: (a)Sparse feature extraction with standard convolution (b)Atrous convolution for dense feature extraction

wise.

4.1.2 Deep Interactive object Selection

Referring from deep interactive object selection (DIOS)[31], we noticed that the points sampling from user interactions as prior served as useful assistant information for estimating the objectiveness of pixels. Their algorithm transforms user-provided positive and negative clicks into two Euclidean distance maps which are

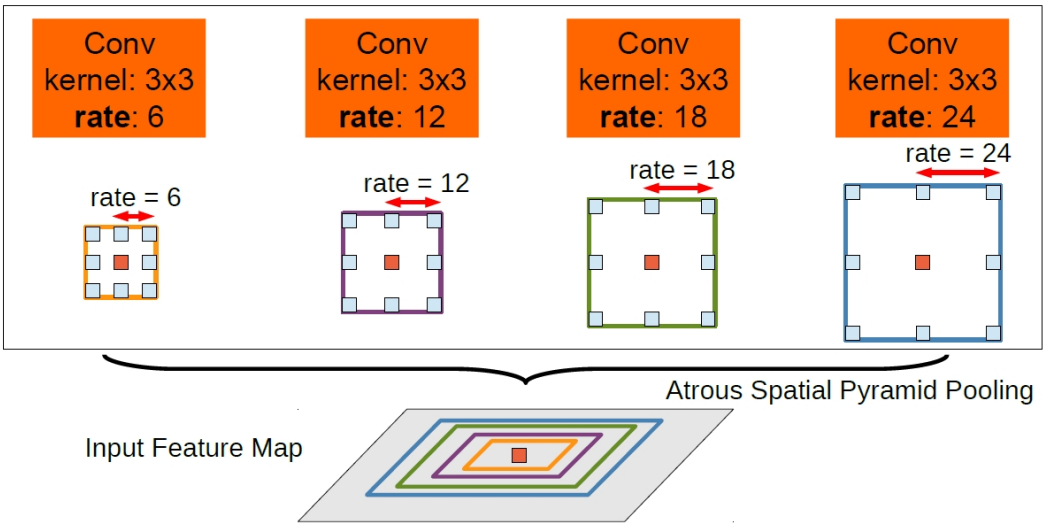

Figure 4.2: Atrous Spatial Pyramid Pooling exploits multi-scale features by employing multiple parallel filters with different rates.

then concatenated with the RGB channels of images to compose (image, user interactions) pairs. They used following distance transformation to construct two interactive channels:

$$f(p_{mn}|\mathcal{A}) = min_{\forall p_{ij} \in \mathcal{A}} \sqrt{(m-i)^2 + (n-j)^2} \tag{4.1}$$

In other words, the function f computes the minimum distance between a point and all other points. The paper provided more details about this. Through re-implementation of this algorithm, we discovered that the problem with this kind of distance transformation is that the overlap of positive distance channel and negative distance channel has relatively high percentage so that the network will be commonly confused about the auxiliary information. To simulate user interactions,

they automatically generate the pairs by randomly sampling positive points inside the object with certain distance among each other, and negative points outside object with three criteria to pretend different user's operations. Suppose $\mathcal{O}$ is the set of ground truth pixels of the object, and $\mathcal{G} = \{p_{ij} | p_{ij} \in \mathcal{O} \quad \text{or} \quad f(p_{ij}|\mathcal{O}) \leq d\}$, The complementary set of $\mathcal{G}$, denoted $\mathcal{G}^c$, covers the background pixels that are within a certain distance range to the object. n positive points are randomly picked from $\mathcal{O}$ where $n \in [1, 5]$ and each sampled pixels have to be a certain distance away from each other as well as object boundaries. Regarding negative points sampling, Strategy 1 of DIOS randomly sampled n negative pixels in the set $\mathcal{G}^c$, where $n \in [0, 10]$. Strategy 2 randomly chooses n_i negative clicks on each negative object $\mathcal{O}_i$ in the same image, where $n_i \in [0, 5]$. The points sampled in strategy 1 and strategy 2 must have a certain distance from other points in the sampled set as well as object boundaries. The strategy 3 tried to sample the negative points around objects, but our experiment show it is not always nicely surrounding the entire object according to the given formula in the paper. It sampled sequential clicks by $p_{next} = argmax_{p_{ij} \in \mathcal{G}^c} f(p_{ij}|S^0 \cup \mathcal{G})$. Figure 4.4 gives the visual example of the three sampling strategies for negative points. We revised strategy 3 to achieve smoother rings, which will be described in next section. By applying formula 4.1 on the sampled pixels, DIOS constructed two extra channels for each image and concatenated them after the RGB channels to fit into FCN. Figure 4.3 demonstrates this process.

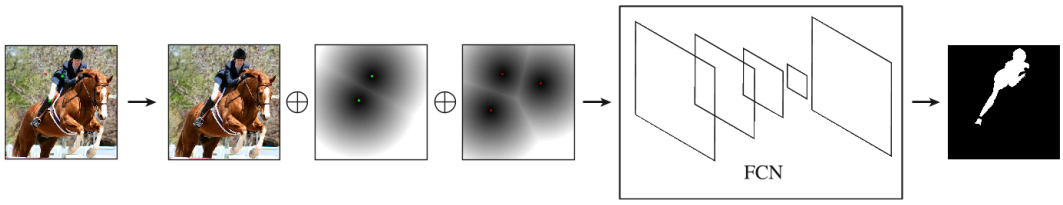

Figure 4.3: The learning process of DIOS. $\oplus$ represents concatenation of channels.

Instead of training on FCN, we fine-tuned our sampled pairs on DeepLab-ResNet. As we introduced, DeepLab-ResNet has already been able to extract context-aware features in some degree. Two more informative channels will further introduce object-aware features to the model which improves the performance.

(a) Strategy 1 (b) Strategy 2 (c) Strategy 3

Figure 4.4: **Three negative points sampling strategies.** Here person is the foreground object.

4.2 Preparing Sampling Data

We augmented the data on the famous PASCAL VOC 2012 segmentation dataset[7] by separating out each object from every image and sampling the positive-negative pairs. Original PASCAL VOC 2012 contains about 9993 segmentation images. Each image will correspond multiple object label files. Figure 4.5 demonstrate the process.

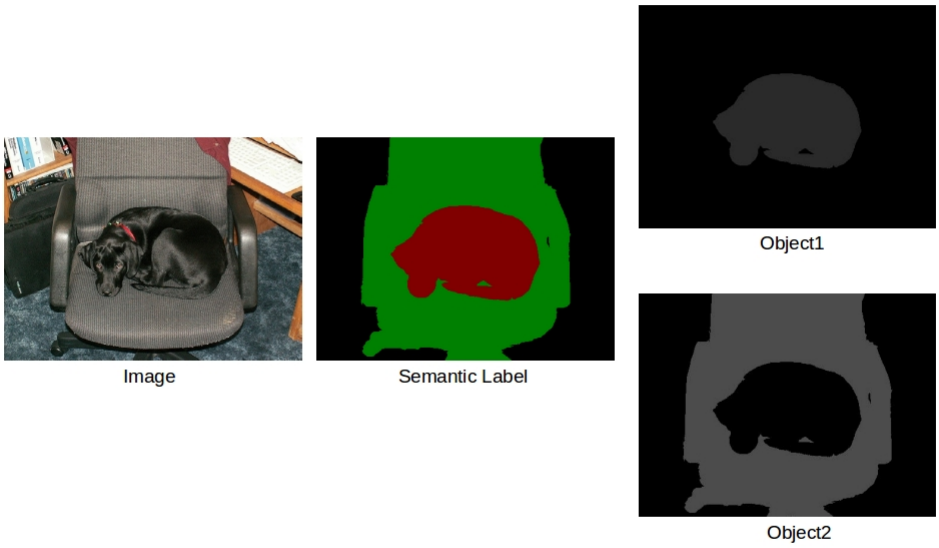

Figure 4.5: Image contains two objects and split into two individual object image.

There are three negative points sampling strategies described in DIOS: 1. sampling randomly outside the object area within the certain distance; 2. sampling in other objects area; 3. sampling surrounding the object. Since each object itself is unaware of other objects, sampling in other objects area has no different from randomly sampling outside the object area. In our method, we used strategy 1 and strategy 3 to sample negative points. We modified strategy 3 to get better surroundings by using morphology transformations. First, dilation is performed on the object mask with 40×40 kernel to get a rough surrounding area. Then, erosion is applied on the rough surrounding area with 15×15 kernel to obtain narrow surrounding area with the width of 10. Eventually, sampling the points inside this fine-grained surrounding area and the distance restriction that each sampling point should have as least 10 pixels away from each other is set to force the points circling the object. 1 to 5 positive points will be randomly chosen, at most 10 negative points will be sampled in strategy 1 and at most 20 points will be sampled in strategy 3 in order to circle the object. Besides, instead of doing distance transformation, we performed scaled distance transformation based on following formula:

$$f(p_{mn}|\mathcal{A}) = \eta \times min_{\forall p_{ij} \in \mathcal{A}} \sqrt{(m-i)^2 + (n-j)^2} \tag{4.2}$$

where η is scale factor. In our experiment on this dataset, we found set scale factor to 4 will avoid most of the overlaps without losing generalization. A scaled

distance transformed channel will be generated after applying formula 4.2. The positive channel will concatenate with two negative channels obtained from strategy 1 and revised strategy 3 to form interaction pairs. Figure 4.6 shows the object and transformed sampling channels.

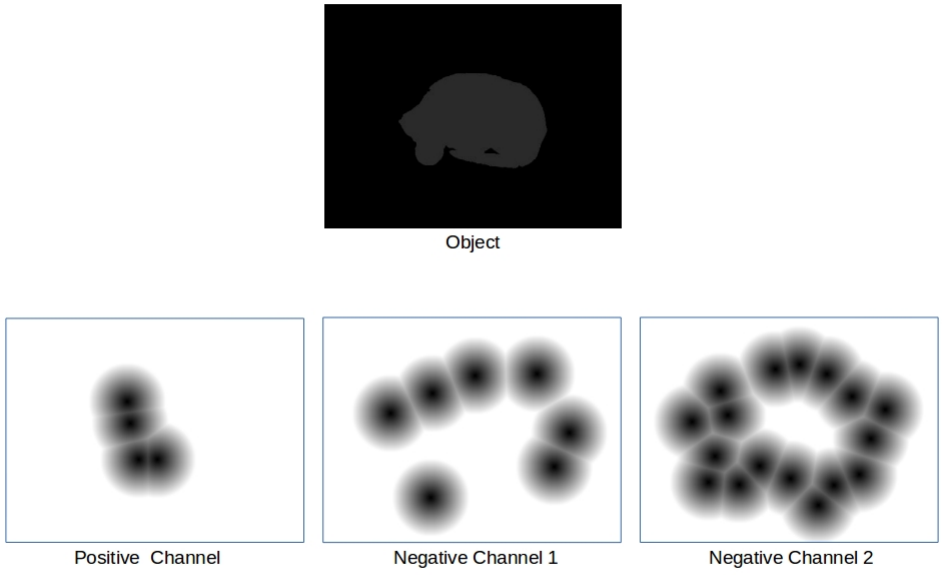

Figure 4.6: Positive and negative channels after apply scaled distance transformation

Wherever the value of distance transformation larger than 255, we truncate it to 255. For the storage efficiency, each image will be stored as a single file and

referred as many as the number of positive-negative pairs they have. Next section will briefly describe the process of how we fit our five channels data into revised DeepLab-ResNet to train the model.

4.3 Training

Initial DeepLab-ResNet is fitted with three channels images. As a result, the input channels are needed to specify as five for matching our five channels data. Additionally, the task is object segmentation instead of 21 classes semantic segmentation, therefore the last four aggregations atrous convolution is altered to be 1 output channel. Consequently, the loss function is changed to be sigmoid cross entropy loss instead of softmax cross entropy loss. We loaded the pre-trained DeepLab-ResNet model which has 79.7% IoU to initialize. Considering the input layer and last four layers are different, we slightly increased the learning rate of the first layer and last four layers. With this manner, these layers will be able to learn faster to match the learned parameters in the rest of the layers. We trained the network with 60000 steps and mini-batch of 10.

4.4 Result

We inferred the object mask of the testing dataset of sampled PASCAL VOC 2012. Each image feed with positive-negative channels into the network and the last aggregation layer output the energy map which shows comparatively high intensity

on the object area and low intensity otherwise. Figure 4.7 presents three randomly selected results.

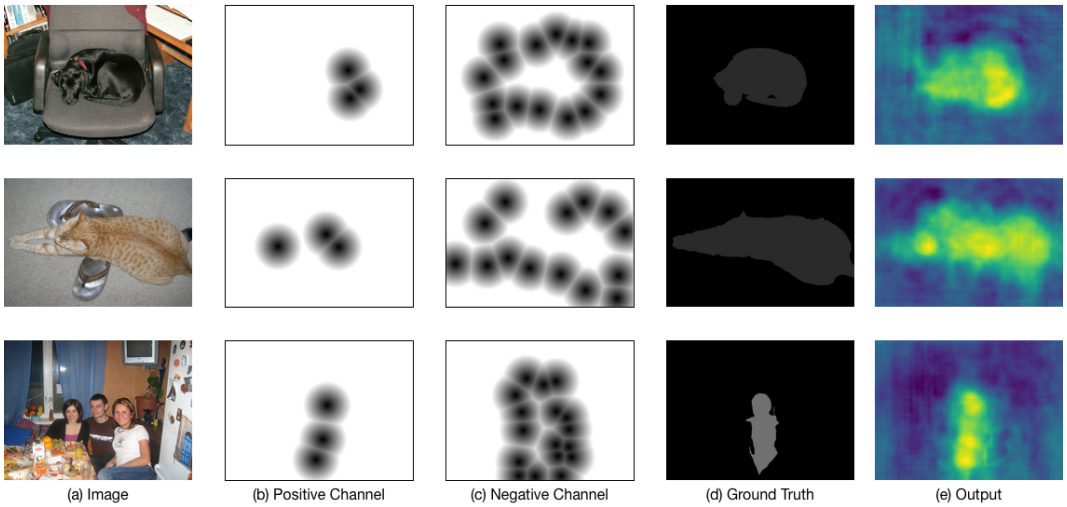

Figure 4.7: **results from trained network.**(a). input RGB images; (b). positive points sampled from object area; (c). negative points sampled outside object. (d). Ground truth of object; (e). Output energy map in last layer.

We further processed the energy map by normalizing them and set threshold with 0.5; the coarse object mask showed up in figure 4.8.

Even though the result is not fine enough to get the exact correct shape of the object, but quantitatively the overall average IoU is about 75.2% in the validation dataset of PASCAL VOC 2012 including 1449 annotated images. We used our sampling strategies on the validation dataset to create about 8938 five-channel images and evaluate them. This proves the effectiveness of our approach, and the

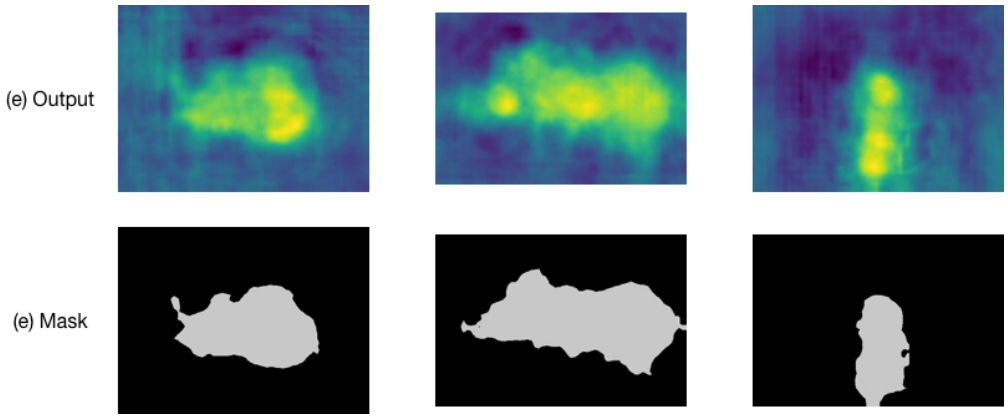

Figure 4.8: Coarse mask after set the threshold to 0.5.

coarse result is quite enough for applying on our Image Annotator web application to conduct object segmentation with user interactions.

4.5 Conclusion

In conclusion, the complementary information is always useful for helping neural network to grab the object-level, even global information of the image. Our method provided the extra information by sampling the positive-negative channel pairs to let network be aware of the position and coarse shape of the object. We believe that the model can be improved by training more sufficient to get better finer shape-aware predictor without any further refinement.

Chapter 5: Image Annotator Web Application

DL in computer vision requires large amounts of manually annotated data to learn difficult visual patterns. However, the purely manual annotation is too expensive to afford for many of researchers and tiny computer vision groups especially for annotating semantic information of objects. On the one hand, researchers expect the data to label as precise and flexible as possible. On the other hand, an automatic system for doing this task has to be assisted by humans as the prior knowledge[1] are necessary. Sometimes, researchers even desire to create their dataset instead of only working on benchmarks. Therefore, an efficient and algorithm-based annotator is badly in need. For this purpose, we initiate the development of a novel image annotator for building semantic segmentation dataset.

5.1 Introduction

In existing simple mask annotators, most of them rarely process the inference from prior knowledge that user provided form user interface (UI). js segment annotator merely applied super pixel analysis on the images which only allows the user to label with super pixels. This approach has obvious drawbacks. Firstly, the super pixels are pre-computed with which users have no choice to annotate other

[1]Here the prior knowledge indicates the user-provided foreground and background annotations.

desired area if the pre-computed areas are not clear. Secondly, it is extremely time-consuming to label such large amount of super pixels by clicks. Lastly, js segment annotator only contains predefined class label color, which will be useless when it comes to annotating another scenario. LabelMe[26] is a multi-functional annotator that is worth mentioning, which allows researchers to label images and share the annotations with the others. In consequence of its multi-functionality, LabelMe performs very poorly on masking the images for the reason that it ignores the pixel-level information. This tool is mostly used to draw the bounding box or boundary of objects instead of directly masking the objects with colors.

- Designed self-defined nestable class labels. Considering users might require the inter-relationship among classes, self-defined classes are invented to be able to nest with each other for the purpose of displaying the subordination.
- Implemented back-end algorithm interface and corresponding algorithms. Users have three annotation modes: Manually, GrabCut[22], deep sampling object segmentation. Each mode links to an entry of algorithm in the back-end. Manually is the most fundamental mode which merely masks what user has drawn. GrabCut mode applied the refined GrabCut algorithm to learn foreground and background model to segment objects. Deep sampling object segmentation mode takes advantages of the object segmentation model described in chapter 3 to extract the object from user's click or strokes.
- Simplified tools for easy-to-use. We provided pen, polygon, rectangle tools

for assisting annotation. Specifically, pen, polygon tools are actual user-provided prior while rectangle specifies the bounding box within which the user will interact.

- Developed object hierarchies for aggregating inserted classes. The user has the choice to specify the object name, which might contain several classes. For example, a plant object may consist of leaf, stem, root classes. The information can be further stored as XML file for the user to analyze the annotation. It does not algorithmically change the possible annotations, merely a supplement of information.

- Developed history panel for undoing and redoing misoperations and visualizing the interaction sequence. Tracking history of interactions is more reasonable than eraser regarding user experience.

- Implemented multi-file importing and visualizing gallery. While switching between the images in the gallery, all previous annotation state will be saved and can be restored after user click back to that image.

- Designed IO system. Users should add images readily during annotating. Also, all annotated information can be stored as either XML file including the original image which also can be imported into the system in the future to preserve footprints, or the image format label if in need.

  Comparing to above mask annotators, our image annotator (IA) web application is exceptionally powerful and flexible for annotating semantics in

many regards:

5.2 Implementation Details

The IA web application splits into two parts: front-end and back-end. We implemented the front-end web page through HTML, Javascript, and jQuery. Especially, the image editor was built upon HTML Canvas, which is an incredibly powerful component for achieving user-image interactions. On the back-end, Python Flask web framework was used to receive annotation request from front-end. OpenCV3 was imported as well for image processing and several basic algorithms. Tensorflow DL framework was introduced for developing the network-based algorithm. When working on the editor, each user interaction will be packed as a request to the back-end, thereafter a response containing an abstract level of the label will be received in front-end, eventually parsed as visualization mask and data structure.

The IA incorporate several functional modules. These modules cooperate with each other to build the user-friendly workflow. Figure 5.1 presented the overview of the most important five modules. These models are not algorithm-related. Here merely shows how they affect each other. Upcoming sections will focus on the implementation details of modules as well as back-end algorithms.

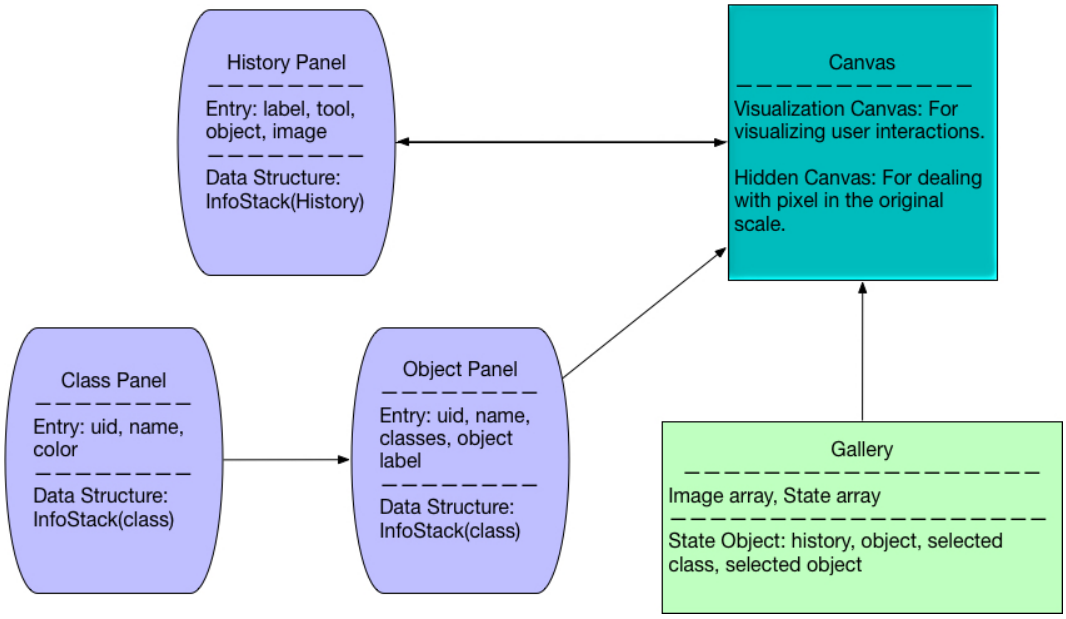

Figure 5.1: Figure shows the important information in each module. The arrows indicate inter-influence between modules. InfoStack is a self-defined stack for store the entries. There are two types of InfoStack. Class InfoStack has no difference from standard stack while history stack implements the **pop** function as move top index back by one for the purpose of redoing.

5.2.1 Object and Class Panels

Object and class panels provide the function of self-defined labeling. In terms UI, class panel primarily consists of five components: defining name and color, adding class or subclass, class tree view frame, adding to object panel, deleting the class. Object panel includes: defining the name, adding the object, object tree view frame, deleting the object. Figure 5.2 shows the appearance of panels. There are tabs on the top for switching between two panels. Behind the screen, we implemented the nested structure using a jQuery widget module called jqTree. jqTree

is an exceptionally powerful module for displaying tree structure and storing the necessary information for further usage. We stored the name and color information companying with a unique id into the tree node and implemented basic insertion, deletion, retrieval functions for it. To retrieval parents or children, we create a stack structure as well called InfoStack which allows nested data structure that the nodes of jqTree do not support. Each adding or deleting step of class or object will affect both the jqTree nodes and InfoStack to update the information.

5.2.2 Toolkit

For simplicity purpose, the toolkit contains only pen, polygon and rectangle tools that drawable on canvas. Specifically, pen and polygon are used to trigger the back-end algorithm to segment objects while rectangle allows the user to draw the bounding box around objects which are for accelerating the algorithm. With bounding box, either GrubCut or deep sampling object segmentation will only focus on the pixels within bounding box and ignore the pixels outside. Line width can also be set by the drop down menu. Mode should be able to select in mode drop down menu. Figure 5.3 is the UI of the toolkit. All of these settings are stored in global variables and will be used as need.

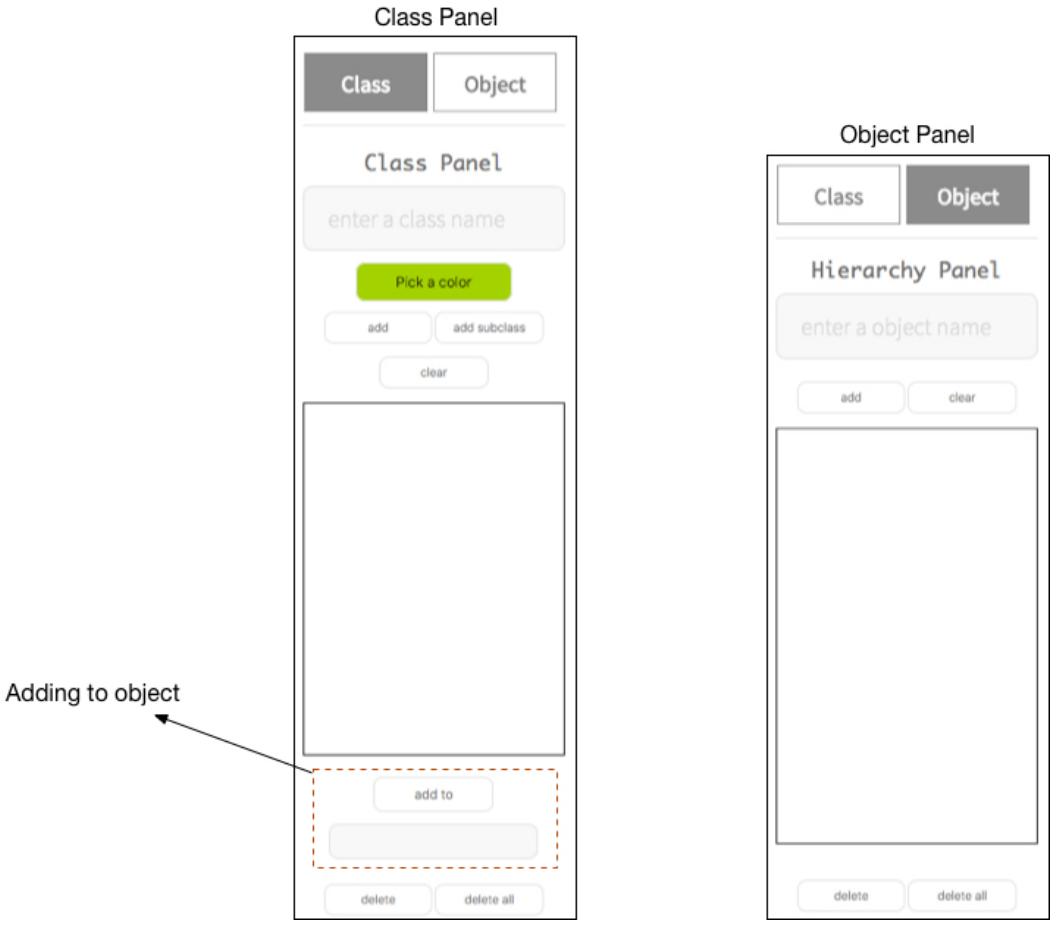

Figure 5.2: The UI of object and class panels. the red dotted rectangle is the component of class panel for adding class to object. Emphasizing it is for avoiding ambiguity.

5.2.3 Canvas

The central editor is implemented by HTML5 Canvas. The canvas in front of the screen is for drawing lines or polygons depending on chosen tool. For finer drawing or overview of the image, zooming has been implemented by scale function of the

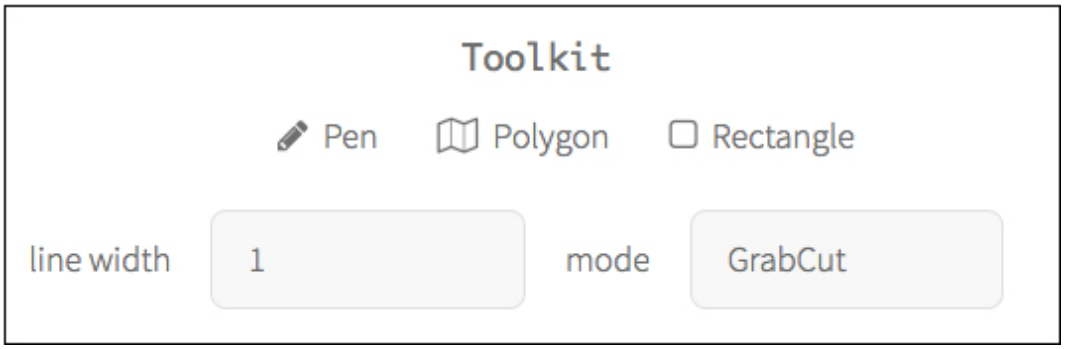

Figure 5.3: The UI of toolkit.

canvas as two buttons on the top of the visible canvas, one for zooming in, the other for zooming out. We also hid an invisible canvas behind the screen for the reason that the scale function of the canvas is not recoverable in resolution-wise. The hidden canvas contains the image with original scale. When ones ask to zoom the canvas, the hidden canvas will provide the original image data then scaling it to desired resolution. We upscaled or downscaled the coordinates of scaled canvas to the hidden canvas in order to map to the correct position. The hidden canvas will be drawn as the visible canvas is drawn. Besides, the editor has tabs for switching the canvas and labels that are calculated from the back-end. The visualization label and canvas are individually existed, which is slightly troublesome. We will combine these two parts together in the future, in another word, the labels sent back from back-end will overlap on the image of the visible canvas. For now, we kept them separately. Figure 5.4 gives the overview of the visible canvas.

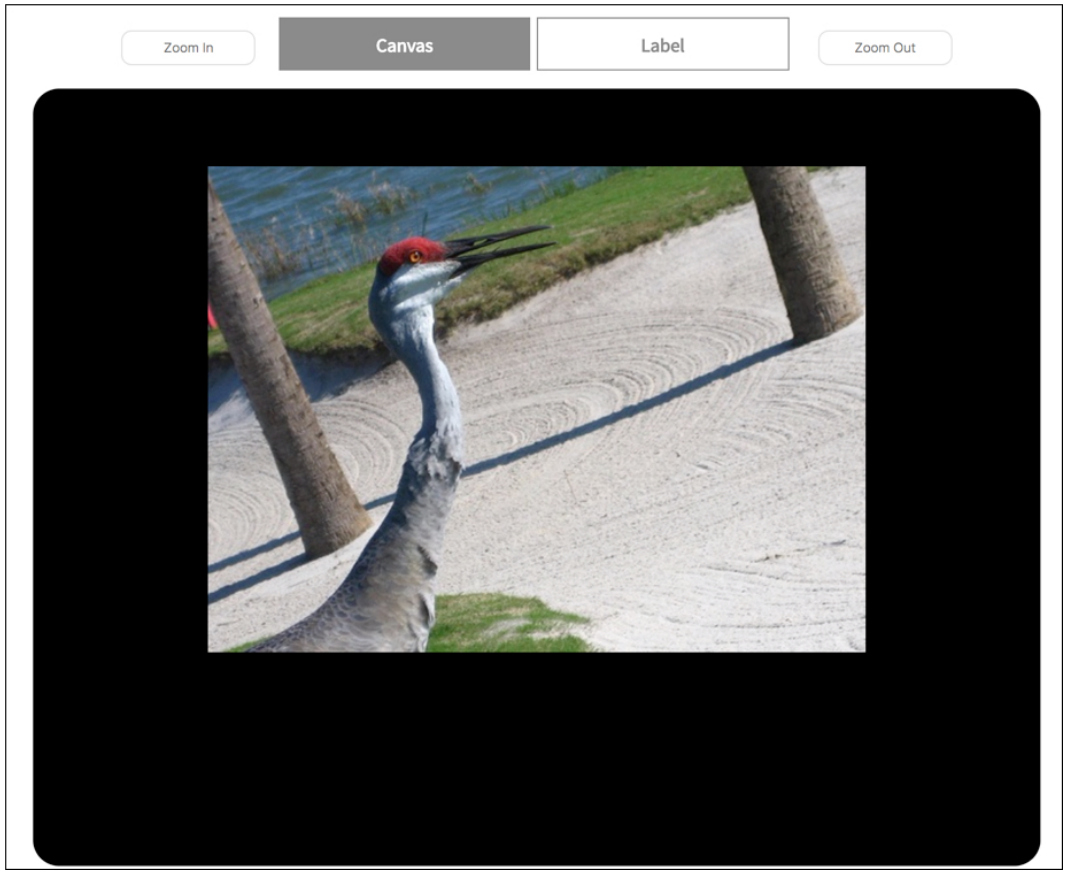

Figure 5.4: The UI of central editor.

5.2.4 History Panel

In order to avoid misoperation, history panel has been integrated to the IA. It allows the user to undo and redo previous drawings and annotations. We designed the InfoStack data structure with "history" type served as underlying implementation to store the history information. With history type, the stack differed from the standard stack in the sense that the pop function will move the top index of

stack back by one instead of completely popping out the first element. Thus when ones undo a drawing, the stack will keep the operation that will be undone for future redoing. History panel also provides a clear button to clear out all operation and return to initial state. Figure 5.5 is the UI of history panel.

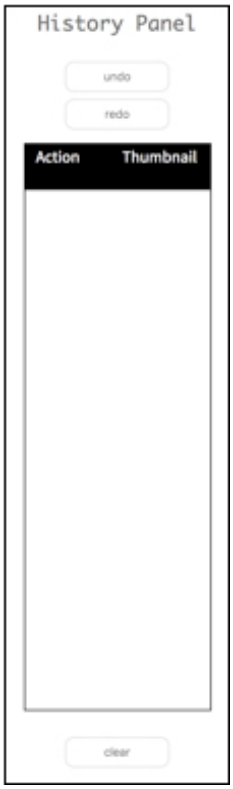

Figure 5.5: The UI of history panel.

5.2.5 File Gallery, Importing and Exporting

File gallery of the IA provides the overview of importing images. Ones should be able to switch the annotating image by simply click the thumbnails. During annotating, the user can import new image files into the gallery as well. The clear gallery button allows the user to clear the entire gallery. After the user completes the image annotating, they are able to save the information either with an image file or XML file that includes all the interactions and masks during the annotation. The underlying XML saving functionality is implemented by Javascript XMLSerializer. Figure 5.6 shows the UI of file gallery.

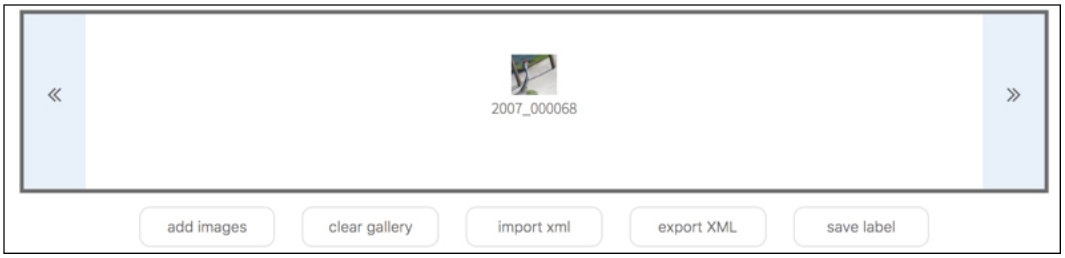

Figure 5.6: The UI of file gallery. User is also able to import multiple files into the gallery.

5.2.6 GrabCut

GrabCut was initially used for interactive foreground extraction. Theoretically, GrabCut is the iterative maximum a posterior (MAP) estimation which is the inference problem of conditional random field (CRF). It does an initial labeling based on user interactions (strokes, polygon in IA). There are four possible mask-

ing cases: foreground, background, probable foreground, probable background. User-drawn pixels will be treated as hard foreground, and other pixels will be labeled as either probable foreground or probable background. Then algorithm trains two Gaussian mixture models (GMM), foreground and background model respectively. Depending on the color statistics, two models will iteratively refine the pixel distributions. A graph then will be built from the pixel distribution. Each node in the graph corresponds to a pixel in the original image. Additionally, every foreground nodes and background connect with the extra-added source node and sink node respectively. The edges in graph carry a weight that defines from the foreground and background model, namely, the probability of the node being foreground/background. After graph is completely constructed, the min-cut algorithm will be applied to cut the graph. It cuts the graph for separating the source node and sink node to two unlinked components. All pixel nodes connect with source node will be foreground while pixel nodes connect with sink node will be background. Above process iteratively runs until the algorithm converges.

In IA's Grab Cut, each user interaction is treated as an independent mask. That is, each request sent to the back-end will train the foreground and background model from scratch to segment the pixels that are related to user's drawing if the annotating class is different. The concept of the foreground in IA is defined as the class. In order to implement multi-class labeling, each GrabCut result from user drawing will be integrated to the existed mask, and the overlap part follows the most recent annotation rule, in other words, the newest class mask computed

from GrabCut will cover the overlap part of previous class masks. With this manner, the final result will be a multi-class label map where the pixel distribution of each class is individually learned. Other than that, Our refined GrabCut performs a post-processing step, which runs breadth first search to detect the connectivity starting with the position that user has drawn after labeling the foreground (class). Standard GrabCut does not take user's intention into account where the resulting masking may discontinuously distribute as segments over the image. Presumably, the object that user wants to label should be around the drawings instead of spread over the entire image. Figure 5.7 shows the result of GrabCut with and without connectivity. The refined GrabCut is selectable in the mode menu.

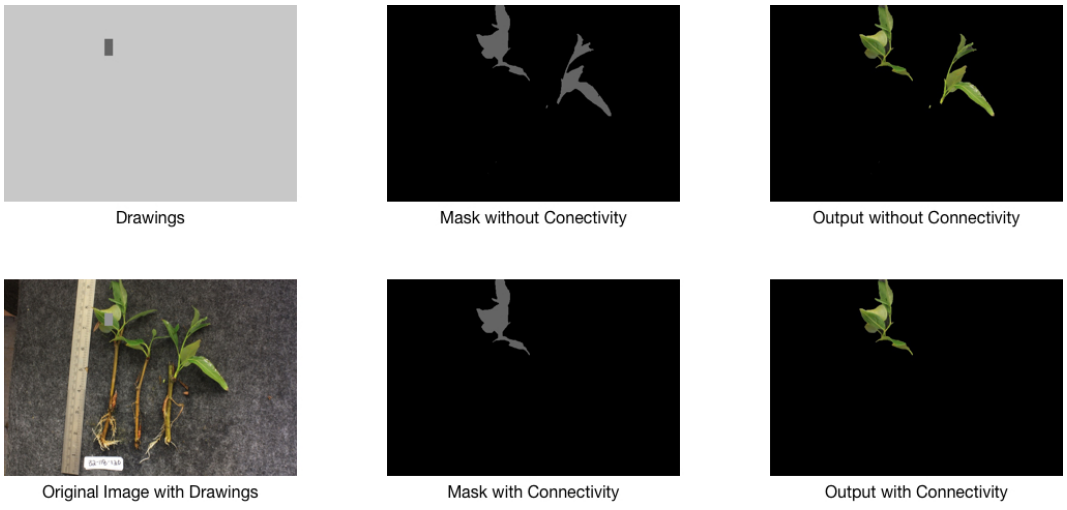

Figure 5.7: The result of GrabCut and refined GrabCut.

5.2.7 Deep Sampling Object Segmentation

Deep sampling object segmentation takes advantages of the model we trained in chapter 3 to output the coarse mask of the object according to user clicks. The user clicks some positive points and negative points, and in the back-end, the algorithm will construct new channels as described in chapter 3 then infer the object mask. We introduced the algorithm on mode. The user simply switches the mode, and the algorithm will be applied to every user interaction. Figure 5.8 shows a working example in our application. The user initially creates people object, cloth class, pant class and hand class. By clicking the hand, cloth, and pant, and clicking the points outside those classes, IA will then output nicely masked image.

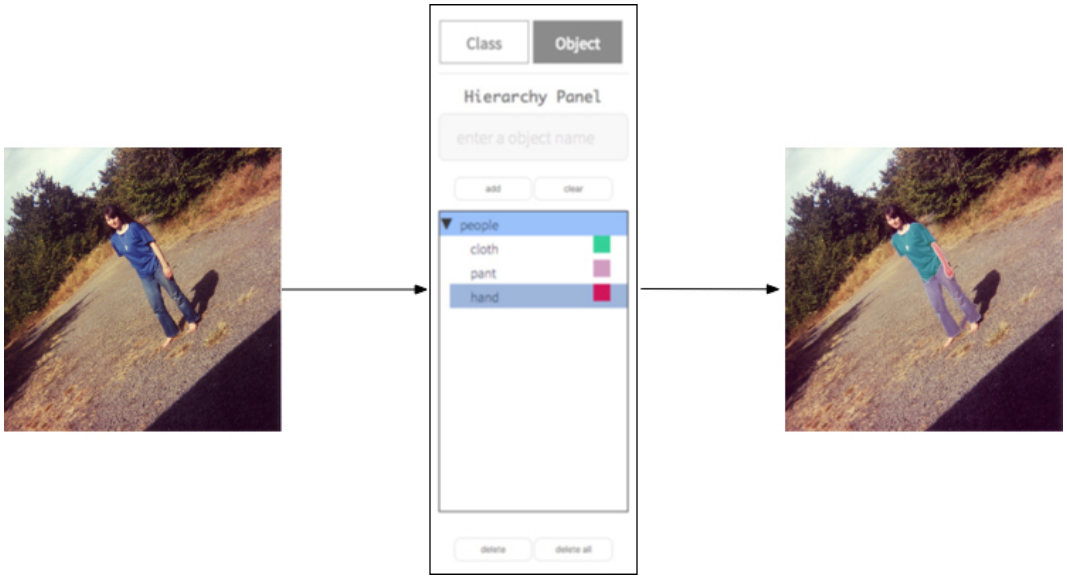

Figure 5.8: Applying the Deep Sampling Object Segmentation model on image.

5.3 Workflow

This section will demonstrate the fundamental workflow of IA. The main studio of IA is showed in figure 5.9. The top is the toolkit that can be chosen by the user to annotate the images. Most of them are already introduced in previous sections. The scrollable canvas is in the middle of the page. The image is also scalable with zoom-in and zoom-out button. Class and object panels are on the left and history panel on the right. In the bottom of the page, there is gallery for showing thumbnails of imported images and IO button for performing IO operations. The user first inserts the class and object into the panel. For annotating on the image, the user needs to choose an object first, then click the desired class to the label. Figure 5.10 shows this step. Next, the user chooses an annotation tool and a mode to draw on the image. The annotation request will subsequently send to the server where the algorithm is executed. After algorithm is done, the server sends a respond to the client side. Client deals with the respond and visualizes the information to the web page. In addition, an entry will be added into history panel for tracking the interactions. This step illustrates in figure 5.11. Finally, the user should be able to export their annotation as XML file that contains all information of the annotation process, or merely jpg file which contains only the label, with the bottom buttons. The IA can be more powerful than above description. It is even capable of restricting the annotation area with the rectangle tool which hugely reduces the algorithm's overhead. But overall, the basic functionalities of IA are mostly introduced.

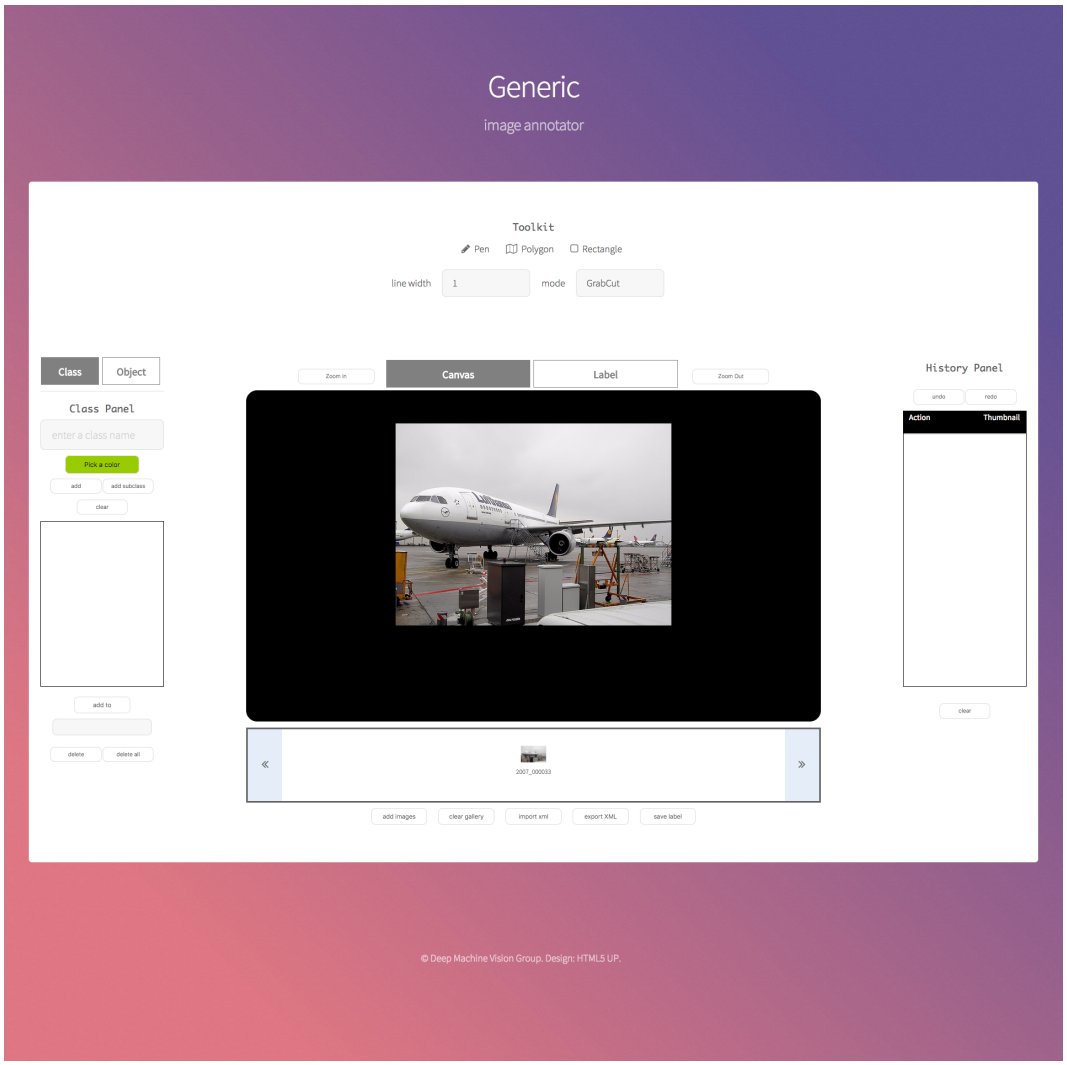

Figure 5.9: IA's studio.

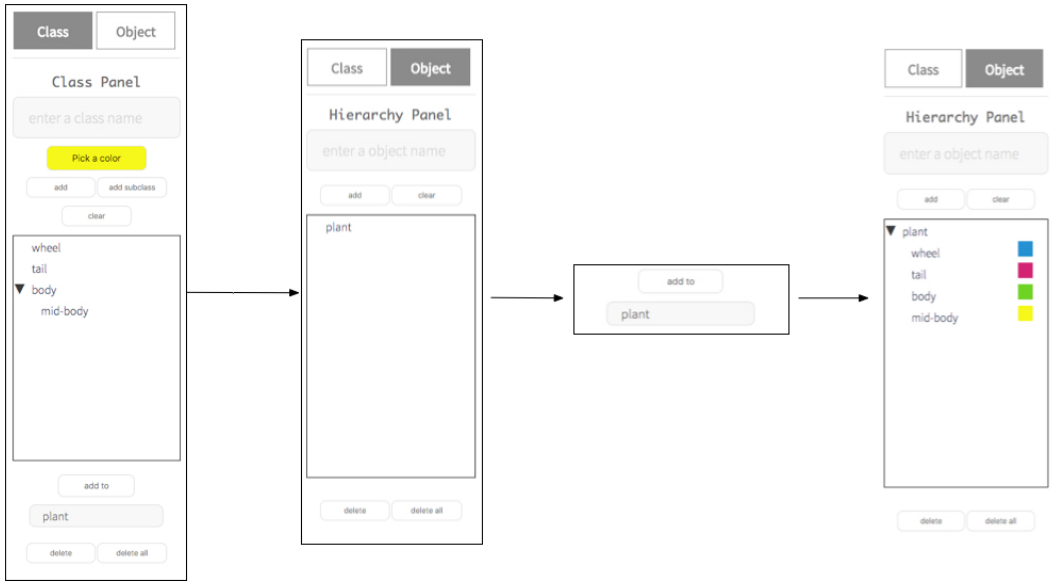

Figure 5.10: Self-defined classes and objects.

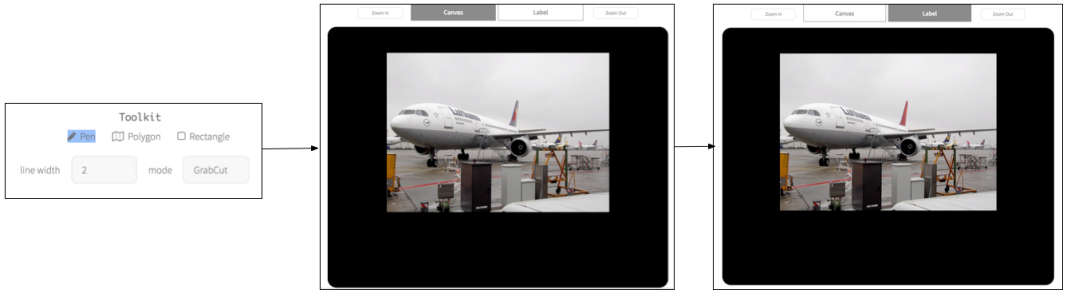

Figure 5.11: Drawing on the image and segment chosen class.

5.4 Conclusion

To sum up, annotation plays a critical role in the data-driven algorithm. The IA web application provides algorithm-based and DL-based annotation which largely reduce the time consumption on marking the labels. Furthermore, with DL algo-

rithm applied, the final label will be finer without increasing overhead. In terms of flexibility, IA allows the user to define their labels even with the hierarchy information contained, and history tracking is certainly useful in many cases. Therefore, IA is no doubt to be a very powerful and flexible annotation system.

Chapter 6: Conclusion

6.1 Summary

In this study, we described two novel approaches to solving the existing disadvantages of the current convolutional neural network, which is of lacking global context information as a local operator. The first approach called flex-shape convolution that made the change of underlying computation of convolution to obtain context information for classifying with scale-invariance and automatic deformation. The second approach took advantage of the strategies described in DIOS to sample positive and negative points and construct the auxiliary data for helping network to explore the object coarse shape and position. These proposed new approaches provide a solid starting for exploring the problems of convolutional neural network and tackling these problems.

Besides, we developed a brand new annotation system called IA to support the effective and efficient semantic image annotation. The IA also applied the DL model that we trained in the second approach to improving the accuracy of annotations. The IA works reasonably well for the researchers who want to flexibly and efficiently create image segmentation dataset. It is Not only a useful tool for annotating any type of dataset, but also one of real-world application of deep

learning algorithm.

6.2 Open Problem and Future work

Even though our work achieved some notable results, there are still arguable problems with them. First, in the case of FSC, we used only max-mapping and min-mapping to perform convolution for deforming the image. This is a bit problematic since the extrema will blur the image. We might consider to use average-mapping or integrate the idea in the bilateral filter which not only takes the value of pixel into account but also weighted by the position of the neighbor pixels. Furthermore, we will work on the self-sufficient ecosystem of the IA. That is, we will provide the interface of semantic segmentation for segmenting the image automatically and using the data that is annotated from user to fine-tune the back-end model. We believe that our algorithm and system will benefit many researchers with these further improvements and hopefully, the community of computer vision and deep learning will be much stronger in the upcoming years.

Bibliography

[1] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011.

[2] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1986.

[3] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *CoRR*, abs/1412.7062, 2014.

[4] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *CoRR*, abs/1606.00915, 2016.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[6] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. Technical Report UCB/EECS-2010-24, EECS Department, University of California, Berkeley, Mar 2010.

[7] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015.

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

[9] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. *arXiv preprint arXiv:1703.06870*, 2017.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR*, abs/1406.4729, 2014.

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

[12] Joos P.L. Vandewalle Igor Aizenberg, Naum N. Aizenberg. Multi-valued and universal binary neurons: Theory, learning and applications. *Springer Science & Business Media*, 2000.

[13] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pages 675–678, New York, NY, USA, 2014. ACM.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[15] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 109–117. Curran Associates, Inc., 2011.

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

[17] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, December 1989.

[18] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[19] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. *CoRR*, abs/1505.04366, 2015.

[20] Aman Raj, Daniel Maturana, and Sebastian Scherer. Multi-scale convolutional architecture for semantic segmentation. Technical Report CMU-RI-TR-15-21, Pittsburgh, PA, October 2015.

[21] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[22] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "grabcut": Interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, August 2004.

[23] Anirban Roy and Sinisa Todorovic. A multi-scale CNN for affordance segmentation in RGB images. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, pages 186–201, 2016.

[24] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986.

[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[26] Bryan C. Russell, Antonio Torralba, Kevin P. Murphy, and William T. Freeman. Labelme: A database and web-based tool for image annotation. *Int. J. Comput. Vision*, 77(1-3):157–173, May 2008.

[27] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[28] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[29] Zifeng Wu, Chunhua Shen, and Anton van den Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. *CoRR*, abs/1611.10080, 2016.

[30] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. *CoRR*, abs/1504.06375, 2015.

[31] Ning Xu, Brian L. Price, Scott Cohen, Jimei Yang, and Thomas S. Huang. Deep interactive object selection. *CoRR*, abs/1603.04042, 2016.

[32] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

[33] Shuchang Zhou, Jia-Nan Wu, Yuxin Wu, and Xinyu Zhou. Exploiting local structures with the kronecker layer in convolutional networks. *CoRR*, abs/1512.09194, 2015.