June 2018

Learning Regularization Weight for CRF Optimization

Jiaxiao Wu
The University of Western Ontario

Supervisor
Veksler, Olga
The University of Western Ontario

Graduate Program in Computer Science

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

Abstract

In recent years, convolutional neural networks (CNNs) are leading the way in many computer vision problems. Since the development of fully convolutional networks, CNNs have been widely employed for low-level pixel-labeling problems, and successfully pushed the performance to a new level. Although CNNs are able to extract highly discriminative features, they typically assign a class label to each image pixel individually. This leads to various spatial inconsistencies. Therefore, CNNs are commonly combined with graphical models, such as conditional random fields (CRFs), to impose spatial coherence. CRFs were invented precisely for the task of imposing spatial coherence among image pixels. The coherence regularization weight serves an important role of controlling the regularization strength in the CRF optimization, and has a great influence on the quality of the final result. Traditionally this weight value is set to a fixed number for all images.

In this thesis, we propose a novel approach to learn the coherence regularization weight for each individual image using a CNN, and then apply this per-image-learned weight in the CNN+CRF system. We first construct a dataset where the optimal regularization weight for the CRF optimization has been pre-computed for each image. We adopt convolutional regression networks with standard architecture for learning, and tailor the input according to our problem. We test the effectiveness of our approach on the task of salient object segmentation where a graph-cut based CRF optimizer can generate globally optimal solution. We show that consistent performance improvements can be achieved by using the regularization weight learned on per-image basis as opposed to a fixed regularization weight for all images in the dataset.

Keywords: parameter learning, regularization weight, convolutional neural network, graph cut optimization

Acknowledgements

I would like to thank my supervisor, Prof. Olga Veksler, for her encouragement and advice throughout my graduate study. She inspired me to solve computer vision problems using deep learning models. It has been a privilege to work with her and other members of the UWO vision group. I would also like to send my special thanks to David Szeto who has been an excellent idea generator and good company. Our constant discussions and arguments led to many interesting ideas. Last, I want to express my appreciation for my family and friends who encouraged and supported me unconditionally.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Image segmentation

The problem of image segmentation has been, and still is, a fundamental research topic in computer vision. Classical image segmentation is the process of partitioning a digital image into a set of regions that are visually distinct and homogeneous with respect to some characteristic or computed property, such as color, intensity, or texture. Image segmentation can also be formulated as a classification problem where each image pixel is assigned with one label from a set of pre-defined class labels [6, 51]. The resulting regions, or segments, consist of pixels with the same label. The criteria of the labeling encourage homogeneity between pixels of the same class and generally consider geometric information and chromatic characteristics. Image segmentation has many applications in various research fields, including medical image analysis such as exam result classification [57], tumour localization [47] and surgery planning [68]; object detection in satellite images [60] and thermal images [62]; and recognition tasks in video surveillance [71] and fingerprint identification system [53].

Given the ambiguity and the complexity of general purpose image segmentation, in this work, we instead focus on the most common type of segmentation: binary segmentation. This involves segmenting the object of interest from its background. In particular, we focus on segmenting the object with the property of *visual saliency* [3, 39].

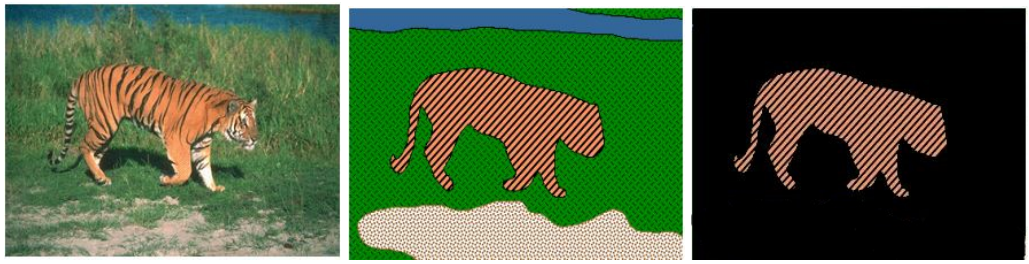
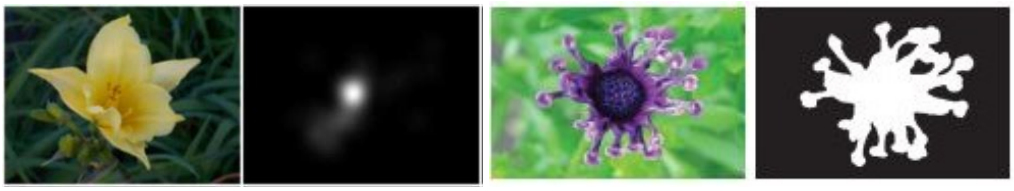


Figure 1.1: Example of different types of segmentation. From left to right: the original image, a semantic segmentation where image pixels are clustered together if they belong to the same object class, the salient object segmentation where the visually distinguishable object is separated from the background. Images edited based on original images from Boykov.



(a) Salient fixation map (b) Salient object segmentation

Figure 1.2: Examples of salient fixation map and salient object segmentation. Left image from [27], right image from [39].

1.2 Visual saliency and salient object segmentation

Visual attention is the ability of the human visual system to select only the *salient* visual stimuli for further processing. The saliency problem has been studied in various disciplines such as cognitive psychology, neuroscience, and computer vision. In computer vision, the purpose of *visual saliency detection* is to identify the most visually distinct object in an image. The traditional approaches express saliency as human eye gaze when input images are displayed to human experimental subjects. Then the task of saliency detection is formulated as finding a saliency model that predicts the probability distribution of the location of the eye fixation over an image, i.e. the *fixation map* [8, 20, 27]. However, an increasing number of recent studies have found that *salient object segmentation*, which focuses on saliency prediction at the object level, is more useful in many computer vision and image processing problems, such as content-aware image resizing and photo visualization [42, 3, 39]. In Figure 1.2 (a) is an image with its fixation map, and an example of salient object segmentation result is provided in Figure 1.2 (b).

1.3 Convolutional neural network approach for salient object segmentation

In recent years, *convolutional neural networks* (CNNs) have achieved remarkable successes in many computer vision tasks. Initially developed for image classification tasks, CNNs have expanded their influence into other high-level vision problems such as object detection, face recognition, and visual object tracking. Around 2015, Long, Shelhamer and Darrell [46] proposed the idea of the *fully convolutional network* for semantic segmentation. Since then, the usage of CNNs has been extended to pixel-labeling problems such as image segmentation, stereo correspondence, and optical flow [56, 72, 1, 15].

Just as with other pixel-labeling tasks, in recent years, CNNs have led the way in salient object segmentation research. Some of these data-driven models [38, 75, 65] aimed to capture spatial contrast and regional coherence by independently modeling image patches at different scales. In 2015, Li and Yu [38] proposed a neural network architecture that extracts features at different scales and evaluates the possibility of a proposed region to be salient based on the contrast between the considered region and its surroundings. Zhao *et al.* [75] presented a

multi-context learning deep network that integrates both global and local contextual information to classify saliency for designated regions, and superpixels were used to ensure regional coherence.

Another category of CNN-based saliency models uses a fully convolutional architecture and end-to-end training [10, 39, 43, 66]. To our knowledge, one of the first multi-scale fully convolutional networks that infer a pixel-level saliency map directly from the raw input image was proposed by Li and Yu [39] in 2016. This deep contrast model was paired with a segment-wise spatial pooling stream and a fully connected CRF to improve coherence. Almost simultaneously, Liu and Han [43] presented a 2-stage end-to-end deep hierarchical model in which a coarse prediction map was produced in the first stage, followed by a hierarchical refinement network to recover image details progressively.

More recently, there were attempts to combine the task of salient object segmentation with other closely related vision tasks such as fixation prediction and semantic segmentation. A multi-task fully convolutional network was developed by Li *et al.* [41] to explore the intrinsic correlations between saliency detection and semantic image segmentation. The proposed model collaboratively learns features for the two correlated tasks in the hope of improving object perception for saliency detection and reducing feature redundancy. Moreover, the idea of developing a unified network that is capable of solving multiple pixel-wise binary problems was explored by Hou *et al.* [26]. With the observation that salient object segmentation, skeleton extraction and edge detection can all benefit from multi-level features in various degrees, Hou *et al.* proposed a horizontal cascade where highly abstract, deep representations are learned in the classic bottom-up structures and the multi-scale features are learned through the horizontal *side paths* and *transition nodes*.

Including but not limited to the above approaches, deep convolutional neural networks have achieved significant progress in salient object segmentation through the last two years. However, regardless of their large capacity of representation, CNNs usually assign a label to each pixel individually, with no explicit enforcement of spatial coherence between the labeling of adjacent pixels. Therefore, there is still room for improvement over the generic CNN models that do not explicitly model this dependency of pixel labeling.

1.4 Challenges of spatial coherence and boundary precision

Convolutional neural networks have demonstrated many successes in high-level vision tasks, including salient object segmentation. However, a main challenge of salient object segmentation is to preserve the spatial details while accurately detecting saliency. The *receptive field* of a neuron is the region in the input image that is directly or indirectly connected to the neuron. The neurons in the deeper layers of a classic network usually have larger receptive field than those in the earlier layers. Having large enough receptive fields is critical to saliency detection because the lack of complete information might cause incorrect classification. Although pooling layers and downsampling layers can effectively enlarge the size of receptive field, they

also reduce the resolution of feature maps. As a consequence, the loss of accuracy in spatial information is inevitable. The resolution loss in feature maps is even more influential for the more recent deep neural nets which tend to consist of more layers and more complex architectures. For image segmentation, losing resolution usually results in the loss of accuracy in the boundary area between salient object and background.

Problems of similar nature happen for other low-level vision tasks such as semantic segmentation, and relative impacts and current attempted solutions are reviewed in a survey [21].

Many new layers and structures were developed with the intention to preserve or restore spatial information to enhance the highly abstract features obtained from deep layers. The two common approaches are

1. combining outputs from different CNN layers by summation, concatenation or skip architectures [25]

2. modeling spatial relationships between nearby pixel labeling with some graphical models [16, 9]

Before CNNs are exploited to image segmentation, graph-based models including Markov Random Fields (MRFs) and Conditional Random Fields (CRFs) [37] were often used to encode relationships between pixels or superpixels based on hand-crafted features. With the expansion of CNNs, they are often incorporated in the CNN learning scheme as a post-processing step that models pixel label dependencies and improve spacial coherence.

When incorporating CRFs with CNNs, the probability measures learned by the CNN are often converted to the unary term in the CRF model, and some learned or fixed pairwise term is added to model the relationship between neighboring nodes. To control the regularization strength of the pairwise coherence term, it is a common practice to impose a weight factor, which, in this work, we refer to as regularization weight λ. With the weighted pairwise term, final segmentation result tends to be more spatially coherent, leading to an improved labeling accuracy.

In fact, the value of this regularization weight parameter λ plays a key role in the spatial coherence of the final segmentation result. If λ is set too small, the length of the boundary between the object and background regions tends to be too long, resulting in over-segmentation. An over-segmented result consists of too many segments or the object boundary is not smooth enough. If λ is set too large, the length of the object boundary tends to be too short, resulting in under-segmentation. In under-segmentation, the object boundary is too smooth and important shape details are omitted.

The conventional way to set the regularization weight λ is either by hand or learned through the training dataset. In both cases, once chosen, the value of λ is fixed as the same value for the entire dataset and future testing images.

1.5 Our approach

Even though a fixed value of the regularization weight λ is currently used for all images in CRF models, a better result might be achieved if each individual image can determine the amount of coherence regularization they need for themselves. This is because the appropriate amount of regularization required for a particular image can be influenced by multiple factors, including the image content, the edge contrast, and also on the quality of the unary term produced by CNN for that particular image. If unary term is noisy, the weight parameter should be increased to make spatial coherence a stronger regularization. There are other non-obvious relationships between the best regularization weight and the image content and the CNN unary terms that one can attempt to learn.

Based on the above observations, in this work, we proposed to learn an appropriate value of the regularization weight λ on a per-image basis. To learn the weight of coherence regulation, we exploited a convolutional network with a contemporary architecture, referred to as *Weight-CNN*. The input to weight-CNN is the color image, the unary probability measures used in the CNN-CRF system, and a tentative segmentation map based solely on the unary term. The output is used to predict an appropriate value for the regularization weight λ which we then use during the CRF optimization using the graph cut algorithm [7].

Due to the novelty of learning the regularization weight in a supervised manner, we were not able to find any publicly accessible dataset for this problem. We constructed new datasets for training and testing based on the conventional saliency segmentation benchmarks. The examples in our augmented datasets consist of the color images coupled with their saliency probability measure maps and preliminary segmentation maps computed solely with unary saliency probabilities. The labels are the optimal regularization weight values estimated empirically from the ground truth saliency segmentations. We showed that, with our per-image-learned regularization weight, we can improve the F-measure metric of a CNN-CRF system in saliency segmentation.

The main contributions of this work can be summarized as following:

- We propose to learn the optimal weight of the coherence regularization on a per image basis for a CNN-CRF system. We employ a fully convolutional neural network to model the intrinsic relationship between the image content, the CNN-generated saliency probability measures and a good regularization weight. The proposed approach selects a regularization weight adapting to each specific input image, instead of a fixed weight for all images in the dataset, and achieves performance improvement in F-measure metric. It is worthy to mention that the proposed approach of regularization parameter learning can be extended beyond salient object segmentation, to any CRF-refined neural network training scheme.

- We demonstrate the adaptability of the proposed weight-learning approach by evaluating our models on four contemporary salient object segmentation datasets, and compare the

F-measures achieved by the fixed regularization weights and the learned variable regularization weights. Consistent F-measure improvements observed on different datasets suggest the effectiveness of regularization weight learning.

- We present a procedure to augment the standard benchmark datasets of salient object segmentation with image-specific optimal coherence regularization weight that a CRF model can use to refine the CNN-generated segmentation results. Our experiment design and procedure might inspire further studies on other CRF parameter learning.

1.6 Thesis outline

This thesis is organized as follows: In Chapter 2, we review the related work, which includes the layers and structures of convolutional neural networks used in this work; the graph cut algorithm which was used as the CRF optimizer for the binary segmentation problem; and prior work of using CRFs to refine CNN-generated results. In Chapter 3, we provide details on the acquisition of the optimal regularization weights on a per-image basis, and introduce the architecture of our weight-learning network Weight-CNN. In Chapter 4, experimental results are presented with performance analysis and configurations of learning. In Chapter 5, we conclude the thesis and discuss potential further improvements and future research directions.

Chapter 2

Related work

In this chapter, we review some fundamental architectures and techniques of convolutional neural networks that we employed for our regularization weight learning networks. We also discuss the conventional approach of modeling the image segmentation problem with CRFs and transferring the image segmentation problem into an energy minimization problem. Then, we review the graph cut algorithm, the optimizer in our energy minimization framework.

We are not aware of any prior work that applies deep learning to obtain the optimal regularization weight for CRF optimization in a supervised manner. However, there is prior work on combining CRFs with CNNs and prior work on parameter learning for CRFs, which we review in the latter two sections of this chapter.

2.1 Neural networks

Convolutional neural networks have led to major advances in computer vision in recent years [35]. In this section, we give an overview of the fundamentals of neural network design.

In general, neural networks are a machine learning model typically used to solve supervised learning problems such as regression or classification. They consist of an arrangement of multiple layers of artificial neurons, which loosely simulate the biological neurons of animals. The particular arrangement of layers depends on the type of neural network. This work only discusses multi-layer perceptrons and convolutional neural networks, but the topology of neural networks in common use is not limited to these two.

Within a given layer, each neuron takes an input vector and computes a linear function on it, giving an output scalar. The linear function is a set of weights learned as part of the training process for the neural network. The output scalar is typically then processed via a non-linear function known as an *activation function* in order to increase the expressive power of the neural network. A more detailed discussion on activation functions is given in Section 2.1.3. The particular arrangement of layers, the format of the input vectors, and the choice of activation function depend on the type of neural network being employed. In other words, they are hyperparameters of the general neural network model.

2.1.1 Traditional neural networks

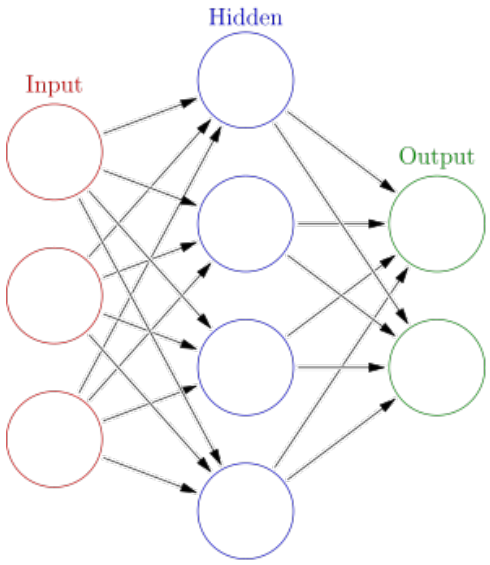


Figure 2.1: A multi-layer perceptron (MLP) with one hidden layer. The input, hidden, and output layers have three, four and two neurons respectively. Image from [22].

A traditional neural network, also known as a *multi-layer perceptron* (MLP), consists of a sequence of layers of neurons. The output of the first layer is simply the input to the neural network. For example, in an image classification task, the image is the input. Within each layer after the first layer, the input vector to every neuron is the vector of outputs (after activation function) from the previous layer. We say that such layers are *fully connected* because every neuron in a given layer is connected to every neuron from the previous one. We call the intermediate layers *hidden layers*.

The output of the overall neural network is simply the output of the last layer. Thus, the choice of the number of neurons in the final layer, as well as the final activation function, determine the format of the output of the neural network. For example, if the neural network is expected to output a probability, one would choose to employ a single neuron in the final layer and a logistic activation function in order to guarantee that the output will be between 0 and 1. Figure 2.1 shows a MLP with one hidden layer [30].

2.1.2 Depth vs breadth

The expressive power of any neural network is correlated with the number of neurons in each layer, i.e., the width or breadth of the layer and the number of layers, i.e., the depth of the network. However, in recent years neural network researchers have focused more on increasing the depth of their neural networks than the breadth for computer vision tasks [30]. For example, in 2012, a neural network with eight learned layers [35] was employed to achieve then-cutting

edge performance on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [59]. Then, in 2014, Simonyan and Zisserman [61] employed several networks with 16 to 19 learned layers to achieve newly-cutting edge performance in ISLVRC. The submission [23] to the 2015 iteration of the challenge employed networks up to 152 layers deep to earn 1st place.

Thus, in this work, we will treat increasing the depth of a neural network as being arguably synonymous with increasing its expressive power.

2.1.3 Activation functions

One employs non-linear activation functions in the intermediate layers of a neural network to increase its expressive power and the degree of potential non-linearity the network can compute. If one were to eschew activation functions, the resulting network would consist of a sequence of layers which computes a sequence of compositions of linear functions. However, from the associative property of matrix multiplication, one can derive the fact that a composition of linear functions is itself a linear function. Linear models are simply not powerful enough to solve complicated vision problems. Therefore, a deeper neural network without activation functions would be no more powerful than a shallower one, all else being equal. Thus, employing activation functions in conjunction with increased depth allow a network to compute more abstract features. Here, we discuss the logistic, hyperbolic tangent, and ReLU activation functions.

Logistic function

The *logistic function*, also known as the *sigmoid function*, was a popular choice of activation function in the past. It is defined as:

$$\sigma(x) = \frac{L}{1 + e^{-k(x-x_0)}}, \tag{2.1}$$

where L, k, and x_0 are constants affecting the behaviour of the function. The *standard* logistic function uses $L = 1, k = 1, x_0 = 0$. In this work, all references to the logistic function will refer to the standard logistic function. A plot of the function is available in Figure 2.2.

The output of the function, being bounded between 0 and 1, simulates a biological neuron either firing or not (or somewhere in between). Another common interpretation of its output, especially for the final layer, is that of a probability. This interpretation is convenient because probabilities also must be between 0 and 1.

Hyperbolic tangent function

Another formerly popular choice was the *hyperbolic tangent function* (tanh), defined as:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.2}$$

A plot is available in Figure 2.3. The tanh function behaves similarly to the logistic function,

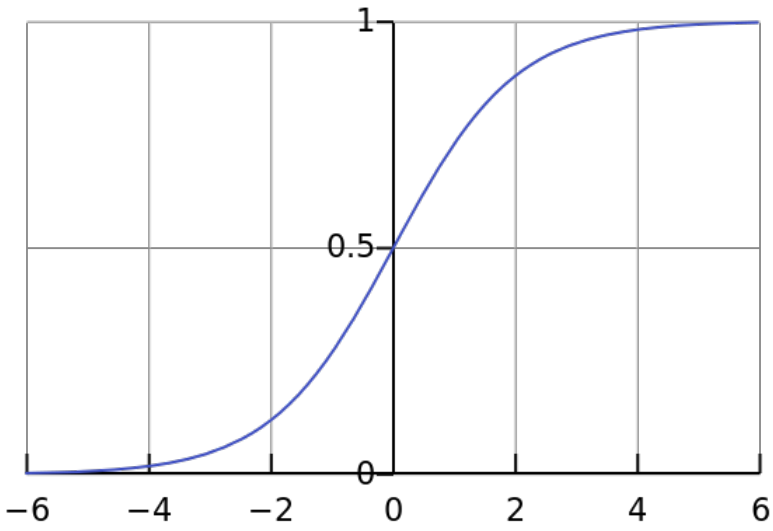


Figure 2.2: A plot of the (standard) logistic function. Image from Wikipedia, public domain.

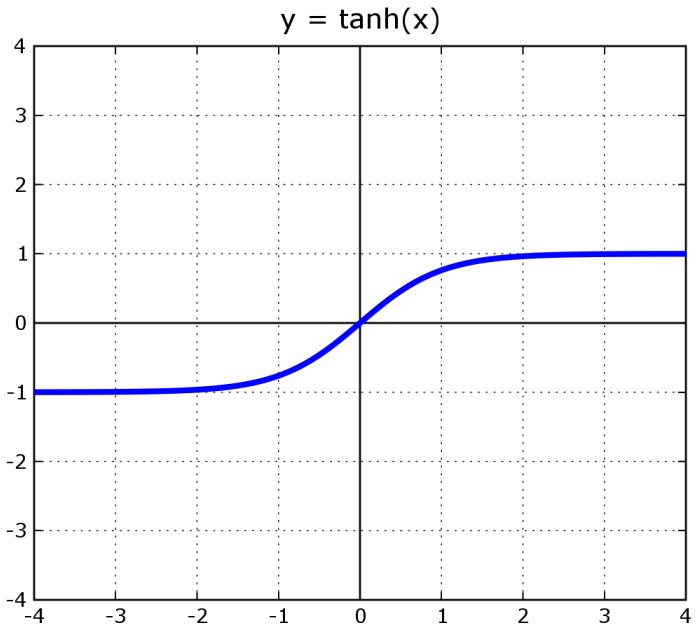


Figure 2.3: A plot of the hyperbolic tangent (tanh) function. Image from Wikipedia, public domain.

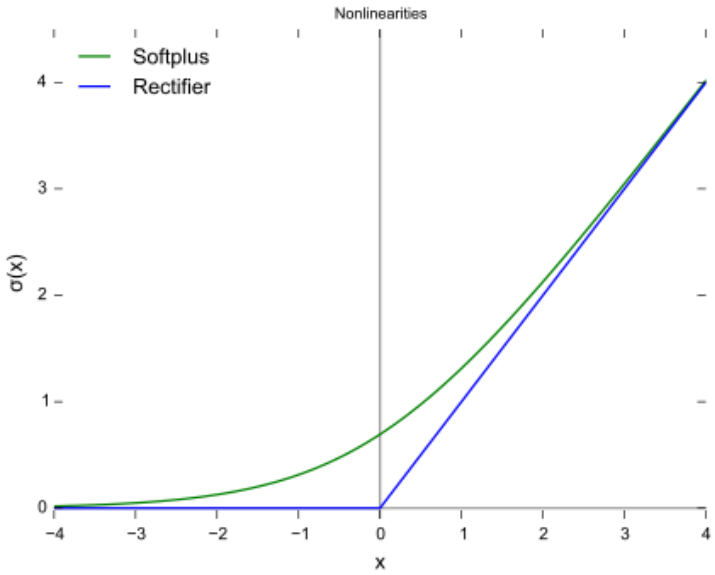


Figure 2.4: A plot of the ReLU function (in blue). Image from Wikipedia, public domain. Discussion of the softplus function (in green) is beyond the scope of this work.

except that its output is bounded between -1 and 1. Indeed, it can be seen that tanh is simply a shifted and re-scaled version of the logistic function:

$$tanh(x) = 2\sigma(2x) - 1. \tag{2.3}$$

For deeper networks, both the logistic and tanh functions suffer from the *vanishing gradient problem*. Because neural networks are typically trained with backpropagation, the rate of update for a given weight is proportional to the gradient of its activation function. However, both functions have a near-zero gradient for inputs with large absolute value. Thus, training can become excessively slow because of mini-scale per-iteration updates. This effect is magnified in deeper networks: a small update in the backpropagation process for a single layer will translate to small updates in earlier layers.

ReLU

The *Rectified Linear Unit* (ReLU) activation function [35] became a popular choice for deep networks because it doesn't suffer from the aforementioned problem. It is defined as:

$$f(x) = x^+ = max(0, x). \tag{2.4}$$

A plot of this function is available in Figure 2.4. It avoids the vanishing gradient problem because its gradient (for non-zero outputs) is always 1. It is also popular because both itself and its gradient are inexpensive to compute.

In the neural networks we train in this work, we exclusively use the ReLU activation function.

2.1.4 Training process

When employed in a supervised learning task, neural networks are typically trained using a process called *backpropagation*, an efficient implementation of gradient descent [30]. Backpropagation for a single training example works as follows.

1. The input training example $\boldsymbol{x}$ is fed into the input layer of the neural network.

2. The neural network performs its computation layer-by-layer until its output $\hat{\boldsymbol{y}}$ is finally computed. We call this process *feeding forward*.

3. The output $\hat{\boldsymbol{y}}$ is compared to a label $\boldsymbol{y}$ using a *loss function*, which computes a scalar C. For example, the squared error loss would compute $C = \sum_i (\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i)^2$.

4. The weights of each layer are updated layer-by-layer in reverse order (hence the name backpropagation). The update rule for a single weight w is $w_t = w_{t-1} - \eta \frac{\partial C}{\partial w_{t-1}}$ where η is the *learning rate* hyperparameter. In this manner, the weight updates perform gradient descent on the cost C.

The process described above constitutes gradient descent for a single training example. In practice, neural networks are typically trained on mini-batches of training examples via *stochastic gradient descent* (SGD) or one of its variants [30]. Mini-batches are more convenient than training on single instances at a time because they are computationally efficient in practice on modern GPUs and they approximate the process of training on the entire dataset at a time better because the batches are randomly sampled from the overall set. At the same time, they are more convenient than the batch method (training on every instance simultaneously) because most modern datasets are too large to train on the entire batch at the same time. Also, normalization only updates the weights after processing the entire dataset, the learning is slower and less responsive to training samples.

2.1.5 Optimizer

As mentioned in Section 2.1.4, neural networks are typically trained using a variant of stochastic gradient descent. Modern optimizer additionally takes advantage of the ideas of *momentum* and following the *sign* of the gradient instead of both its *sign and magnitude*.

Following the sign via RMSProp

When one adjusts a weight in the process of training a neural network, one uses an algorithm like backpropagation and SGD to determine whether to adjust the weight, which is a scalar value, upwards or downwards, and by *how much*. The Rprop optimizer [55] proposes to instead follow a fixed step size for each weight in the direction of the gradient. It does this by dividing weight adjustment by its absolute value, then multiplying by a *step size* scalar hyperparameter. To put it simply, unlike SGD, Rprop only decides whether to adjust the weight upwards or downwards. The magnitude of the step is held fixed.

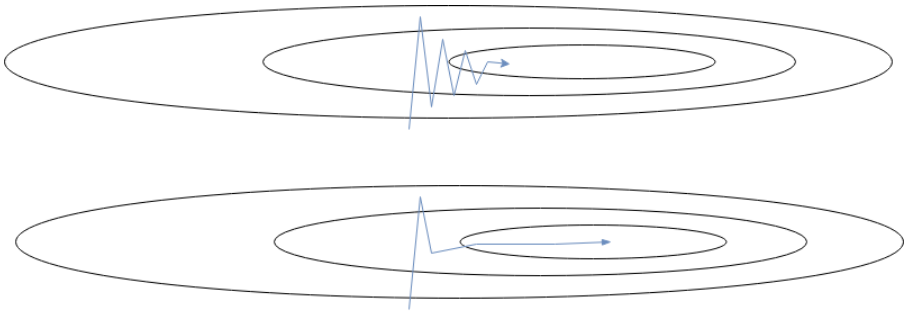


Figure 2.5: Top: a visualization of a potential oscillation in the process of gradient descent. Bottom: a visualization of how momentum could smooth out the oscillation and help the algorithm descend more directly towards the local minimum.

However, the Rprop algorithm was not designed with mini-batch-based training in mind. It assumes that the gradient is fixed, which is not the case for this kind of training. For example, the gradient of nine mini-batches may move a weight upwards by a value of 0.1, and then the tenth mini-batch may move the weight downwards by 0.9. The Rprop algorithm, assuming with a fixed step size of 1, would move this weight up by a value of 8 after the ten mini-batches, but clearly we would not want the weight to explode upwards like this. Or vice versa, weights could also explode downwards.

The RMSProp algorithm [64] attempts to fix this by dividing the weight update by the square root of a moving average of the squared gradient, instead of merely the magnitude of the current weight update as in Rprop. The exponential moving average of the squared gradient for a given weight w at time step t is calculated as follows:

$$MeanSquare(w,t) = decay \times MeanSquare(w,t-1) + (1-decay) \times (\frac{\partial C}{\partial w}(t))^2, \tag{2.5}$$

where *decay* is a hyperparameter denoting how slowly the old values of *MeanSquare* decay exponentially. Tieleman and Hinton give a default value of 0.9.

Momentum

The strategy of gradient descent with *momentum* [52] attempts to reduce oscillation in the process of gradient descent (stochastic or otherwise). To borrow terms from physics, applying the momentum strategy treats the vector of the weights as a massive object with inertia and friction. Loosely speaking, the weight update process updates the *velocity* instead of the position of the object. Thus, it damps out oscillating forces over time. Figure 2.5 demonstrates the way momentum can change the path taken by a gradient in order to reduce oscillation.

More precisely, the weight update V_t at training step t can be defined as an exponentially-weighted moving average of the previous updates:

$$V_t = \beta V_{t-1} + (1-\beta)\frac{\partial C}{\partial w}, \tag{2.6}$$

where β is a decay hyperparameter affecting the proportion of the update coming from the previous momentum.

Adam

The *Adam* optimization strategy [31] is related to both RMSProp and SGD with momentum. Its name comes from *adaptive moment estimation*.

In addition to keeping track of a moving average of the gradient for each weight, it keeps track of an exponentially-weighted moving average of the *squared gradient*. The latter is used as an estimate of the un-centered variance of the gradient updates. During the weight updates, the average update is divided by the square root of this variance. As a result, if the variance of recent updates is high, the resulting update to the weight will be reduced.

The average gradient is calculated as follows:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial C}{\partial w} \\ \hat{m}_t &= m_t/(1-\beta_1), \end{aligned} \tag{2.7}$$

where m_t is a biased estimate of the moving average, $\hat{m}_t$ is a biased-corrected estimate of the same, β_1 is an exponential decay hyperparameter, C is the cost, and w is the weight in question. Likewise, the average squared gradient is calculated as follows:

$$\begin{aligned} v_t &= \beta_2 v_{t-1} + (1-\beta_2)\left(\frac{\partial C}{\partial w}\right)^2 \\ \hat{v}_t &= v_t/(1-\beta_2), \end{aligned} \tag{2.8}$$

where v_t is a biased estimate of the moving average, $\hat{v}_t$ is a biased-corrected estimate of the same, and β_2 is an exponential decay hyperparameter.

The overall weight update is as follows:

$$w_t = w_{t-1} - \eta\frac{\hat{m}}{\sqrt{\hat{v}}+\epsilon}, \tag{2.9}$$

where η is a learning rate hyperparameter and ϵ is a very small number included to avoid division by zero.

In practice, Adam works very well compared to the other popular variations of gradient descent for neural network optimization [58]. For this reason, we chose to use it for the training of all of our neural networks.

2.1.6 Convolutional neural networks

Convolutional Neural Networks (CNNs) have been applied to many problems in recent years [14]. CNNs employ *convolutional layers* in addition to the fully connected layers employed by traditional neural networks. Convolutional layers are simply layers which replace the linear function with a convolution. The convolutional layers typically appear before the fully connected layers. For example, the VGG-16 network [61] contains 13 convolutional layers (alongside some max pool layers, described in Section 2.1.7) followed by 3 fully connected layers.

Convolutional layer

The expressive power of a neural network is arguably bounded by the number of learnable weights it contains, which itself is bounded by the breadth and depth of the network. Fully connected layers contain many weights. In a fully connected layer with m input and n output neurons, there are mn weights. This can be computationally expensive to train and prone to overfitting, because even a single fully connected layer is so powerful.

As a more concrete example, suppose we want to analyze a $224 \times 224 \times 3$ image using a traditional MLP neural network. The input to the first fully connected layer would contain $224 \times 224 \times 3 = 150528$ neurons. If the output of the layer has the same dimensions, the number of trainable weights will be nearly 2.3×10^{10}. Clearly, training even a single fully connected layer for a relatively small image is computationally prohibitive. Comparing to fully connected layers, *convolutional layers* take advantages of spatial regularities in images in order to avoid employing an inordinate number of weights.

A convolutional layer extracts useful information and computes *feature maps* from an image or from feature maps computed by a previous layer by convolving a bunch of filters with learnable weights. Because the filter's weights are learned, the layer can learn to perform simple tasks such as edge detection. Then, the extracted edges can be further processed in later convolutional layers for more complex tasks such as detection of parallelism. This can go on to be used in high level tasks such as detection of specific shapes. This process of solving low-level tasks in early layers and gradually progressing to higher-level tasks in later layers loosely reflects how the human visual system works, where early areas of the visual system solve lower level tasks and later areas focus on higher level ones.

A single feature map is computed as follows. Given an input of dimension $h \times w \times c$, where h, w, and c stand for the height, width, and number of channels respectively, a $n \times n \times c$ dimensional[1] filter is slid vertically and horizontally across the h and w dimensions[2]. At each step in the sliding process, the weight associated with each cell in the filter is multiplied with the value it is currently positioned over in the input. These products are summed together

[1]It is possible for the filter to have different first and second dimensions, but in practice they are often the same.

[2]This is the two-dimensional convolution. One- and three-dimensional versions exist; they simply require inputs with two or four dimensions respectively.

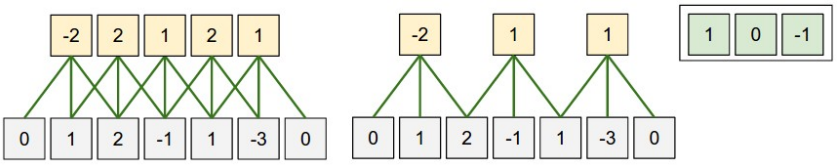


Figure 2.6: One-dimensional convolution examples with different stride values. The input is the lower layer of neurons and the feature map is the upper one. From left to right: convolution operation with stride 1; the same convolution, but with stride 2; the filter used to generate the feature maps. Image from [30].

to produce the value in the cell of the feature map at the given position. This operation is called a *convolution*. In order to compute multiple feature maps from a given input, one simply performs the convolution multiple times using different filters.

Note that the third dimension of the filter is always the same as the number of channels from the input, so is typically omitted when stating the size. Thus, one would often say "the filter has size $n \times n$".

A major advantage of convolutional layers is the fact that they use fewer parameters because the parameters of each filter are shared to compute all cells in a feature map. This reduces training time and the chance that the neural network might overfit. At the same time, there is an intuition behind this kind of parameter sharing: if a feature is useful in one image location, it should be useful in all other locations. The convolution takes advantage of these regularities.

Stride

One can use *stride* to generate a feature map significantly smaller than the input. Specifically, the stride is the number of pixels by which to translate the filter at every step of the convolution. A stride of 1 is the unaltered convolution described above. A stride of 2 slides the filter by two pixels every step in order to generate a feature map roughly half the size of the input in each dimension, and so on. See Figure 2.6 for a demonstration of how stride can reduce the size of a feature map.

In influential networks such as AlexNet [35] and the VGG nets [61], max pooling was used to reduce the size of successive feature maps rather than stride. These networks only used a stride of 1 in their convolutions. The neural networks in this work follow the same practice. See Section 2.1.7 for more information on max pooling.

Padding

By default, the convolution operation places the filter at the top left of the input and slides it until it reaches the top-right corner, then continuing to the bottom left and right corners. However, this process will cause the feature map to be smaller than the input. For example, when convolving a 3x3 filter with a $h \times w \times c$ dimensional input, the resulting feature map will

have size $(h-1) \times (w-1)$. While there is nothing wrong with this, the most influential CNNs such as AlexNet [35] and VGG nets [61] have preferred to keep the feature map the same height and width as the input via the use of *zero padding*. Other padding strategies have been used by other neural networks, but the discussion of them is beyond the scope of this work.

Zero padding implicitly extends the boundaries of the image with zero values. Suppose the filter size is n in the dimension of interest. The amount of zero padding required to keep the feature map the same size as the input would be $\lfloor \frac{n}{2} \rfloor$.

The neural networks in this work follow the strategy used by the aforementioned CNNs: zero padding is used before convolutions in order to make the feature maps have the same dimensions as their inputs.

2.1.7 Max Pooling

A *pooling* layer is used to reduce the dimensionality of the feature maps using some method to aggregate the responses. The AlexNet [35] and VGG nets [61] both use *maximum pooling* (max pooling). Other pooling strategies exist, such as average pooling, but their discussion is not the focus of this work.

Max pooling aggregates responses of cells within a feature map by taking their maximum. Like convolutions, they use strides. A max pool with a 2×2 filter and the stride of 2 will create a feature map by placing the filter on every other pixel positions and taking the max in the region covered by the filter. Thus, each dimension of the input will be halved. Likewise, the stride of 3 will take the max in the designated filter region placed on every 3 pixels, giving a feature map roughly one-third the size of the input in each dimension.

AlexNet and VGG nets use max pooling with stride 2 in order to halve the height and width of their feature maps. The neural networks in this work follow this strategy. The VGG nets use a total of five of these max pooling layers, giving feature maps $1/32$ as high and wide as the original input images. See Figure 2.7 for a diagram of the architecture of the VGG-16 network.

2.1.8 Regularization

Neural networks can incorporate a massive number of trainable parameters and are thus highly prone to overfitting. For example, the VGG-19 network [61] has around 144 million learnable parameters. This section details the regularization strategies of batch normalization, dropout, and weight decay which attempt to reduce the problem of overfitting. These strategies were used to regularize the neural networks in this work.

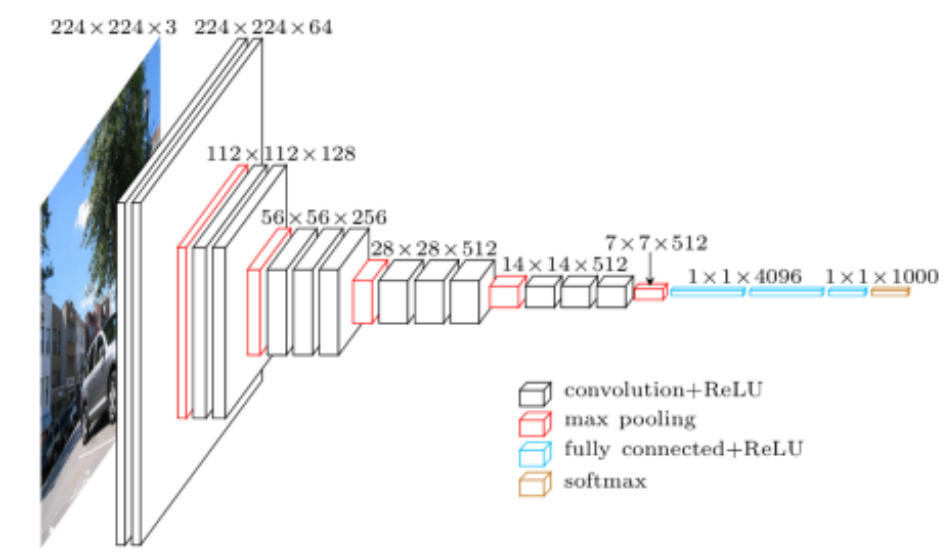


Figure 2.7: The architecture of the VGG-16 neural network. Note that the network uses a *softmax* function as output. Softmax function, also known as normalized exponential function, is often used in image classification tasks to represent a categorical distribution. Detailed discussion of softmax is beyond the scope of this work because we focus on training a regression network. Image from [19].

Batch normalization

Batch normalization [28] is a form of regularization applied to the output of a layer within a neural network. It can be applied to an image, a feature map (as output by a convolution or max pool), or simply a vector of data (as used by a fully connected layer). Batch normalization on the output of a given layer ensures that the data will have a consistent mean β and variance γ, which are learned parameters that change slowly over time. Thus, the next layer will receive data with a relatively consistent distribution, as a result, allowing a deep network to train with a higher learning rate while reducing fluctuations.

As its name suggests, batch normalization normalizes on a per-mini-batch basis. A mini-batch is a subset of the training samples, often randomly selected; and the size of it is usually treated as a hyperparameter. As previously mentioned, neural networks are typically trained on mini-batches of its training dataset. Batch normalization ensures a relatively consistent mean and variance for each mini-batch.

Batch normalization applies to the following calculation to input x in order to obtain output y:

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} \times \gamma + \beta, \tag{2.10}$$

where $E[x]$ and $Var[x]$ represent the mean and variance of the data respectively on a per-mini-batch basis at training time. At testing time, they represent an estimate of the population mean and variance respectively, where these estimates are bias-corrected moving averages of

the mini-batch means and variances collected at training time. ϵ represents a small number included in order to avoid division by zero.

Dropout

As previously mentioned, neural networks can contain hundreds of millions of learnable parameters. If a dataset doesn't contain sufficient training examples, then many of these parameters may not be properly trained before the neural network begins to overfit its training data. The *dropout* regularization strategy [63] attempts to encourage the neural network to train more of these parameters.

Like batch normalization, dropout is applied to the output of a neural network layer. It randomly zeroes out some of these elements with a probability p, where p is a hyperparameter. The elements which are zeroed out are then excluded from all feed-forward and back-propagation. The selected elements are chosen randomly on every forward call.

Dropout is employed only during the training process. It encourages the neural network to train using more parameters effectively, because different parameters will be zeroed out at different training steps.

Weight decay

Weight decay is an additional term added to the training loss in order to discourage any learnable weight from growing too large. An unusually large weight is a symptom of overfitting, a problem where a model performs well on training data but does not generalize to unseen data. Thus, with weight decay, a weight will only grow large if necessary.

The modified loss L is calculated as follows:

$$L = L_0 + \frac{1}{2}\lambda \sum_i w_i^2, \tag{2.11}$$

where L_0 is the original loss, w_i represents the ith weight in the network, and λ is a hyperparameter.

2.2 Segmentation as energy minimization

Image segmentation is not a well defined problem as different segmentations can be acceptable under different criteria. Many more precise segmentation tasks, such as semantic segmentation and salient object segmentation, can be formulated as a pixel-wise labeling problem. Let Ω be the set of all image pixels, L be a finite set of all possible labels, then the task of segmentation is, for every pixel $p \in \Omega$, assign a label $f_p \in L$. For binary segmentation, the label set L is $\{0, 1\}$, where 0 represents the background and 1 represents the object of interest. Let $f = \{f_p \in L | \forall p \in \Omega\}$ denote a labeling, namely a label assignment to all pixels, then the challenge is to find a labeling f that follows the observed data while generating visually smooth

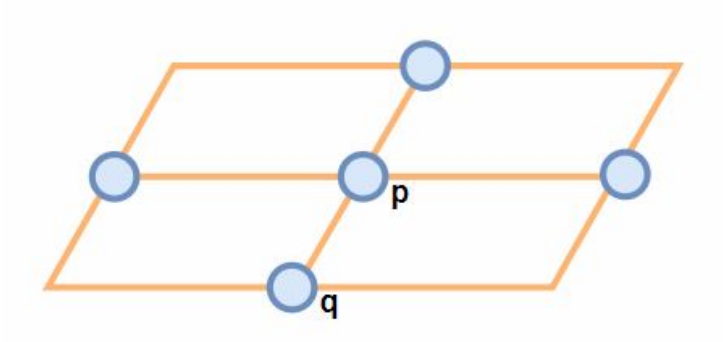


Figure 2.8: The 4-neighborhood system. The pixels to the left, right, top, bottom of the pixel p are considered the neighbors of p.

boundaries. Given the above notations, the problem of image segmentation can be expressed as an energy minimization problem [4, 6, 7] with the following energy

$$E(f) = \sum_{p \in \Omega} D_p(f_p) + \lambda \sum_{p,q \in N} V_{pq}(f_p, f_q), \tag{2.12}$$

where $D_p(f_p)$ is a unary function that models the cost of assigning label f_p to the pixel p, $V_{pq}(f_p, f_q)$ is a binary function that penalizes label discontinuities, λ is the weight factor balancing the strength of coherence regularization, and N is a neighbouring system. In this thesis, we employ a 4-neighborhood system as demonstrated in Figure 2.8.

2.2.1 Unary data term

$D_p(f_p)$ is a function that measures how well pixel p fits label f_p, and is often referred to as the *data term*. One of the most commonly adopted data terms is the negative log likelihood of the probability distribution for pixel p to have f_p as the correct label [4, 51, 45]. The mathematical expression is

$$D_p(f_p) = -logP(I_p \mid f_p), \tag{2.13}$$

where I_p usually represents some appearance information at pixel p, such as intensity or colour. When the log-likelihood function has a large value, the pixel p is more likely to be assigned with the label f_p, and a small penalty should be applied to the energy function. The minus sign transfers the likelihood maximization task into a energy minimization problem.

For binary segmentation, the label set is $L = \{0, 1\}$, and the data terms are calculated as follows:

$$\begin{aligned} D_p(0) &= -logP(I_p \mid 0) \\ D_p(1) &= -logP(I_p \mid 1). \end{aligned} \tag{2.14}$$

However, it is not always possible in practice to acquire an accurate estimation of the probability distribution $P(I_p \mid f_p)$ for pixel features in different segments. Consequently, the label

(a) (b) (c)

Figure 2.9: A Graph Cut segmentation example. (a) input image, (b) segmentation with data term only, (c) segmentation with data term and smoothness regularization. Image from Y. Boykov.

assignment of pixel p obtained solely by log likelihood test as in Equation (2.15) is not sufficient for a high quality segmentation.

$$f_p = \begin{cases} 0 & \text{if} - log\frac{P(I_p \mid 1)}{P(I_p \mid 0)} > 0 \\ 1 & \text{otherwise} \end{cases} \tag{2.15}$$

Figure 2.9 demonstrates the case where appearance model alone is insufficient to distinguish object pixels from background pixels when some background pixels have similar colour features as the object pixels. The lack of spatial coherence results in a noisy segmentation in Figure 2.9 (b), and a prominent improvement was achieved in Figure 2.9 (c) by considering the label choices of surrounding pixels when assigning the label to a pixel.

2.2.2 Binary smoothness term

To encourage spatial coherence, a smoothness function $V_{pq}(f_p, f_q)$ is employed to Equation (2.12), which introduces penalties when neighbouring pixels are assigned with different labels. This can be thought of as the cost of discontinuity in segmentation. Adding a smoothness term into the energy function encourages the segmentation results to have a shorter boundary. However, we also want to respect the observed data, so it is reasonable for the smoothness term to penalize more when the neighboring pixels with similar appearance features are assigned with different labels, but penalize less when neighboring pixels exhibiting distinct features are labeled differently. Thus, we adopt a widely used smoothness term [4, 51] in this thesis, which is defined as

$$V_{pq}(f_p, f_q) = w_{pq} \cdot I[f_p \neq f_q], \tag{2.16}$$

where

$$I[x] = \begin{cases} 1 & \text{if condition } x \text{ is true} \\ 0 & \text{otherwise} \end{cases} \tag{2.17}$$

and

$$w_{pq} = exp(-\frac{(I_p - I_q)^2}{2\sigma^2}). \tag{2.18}$$

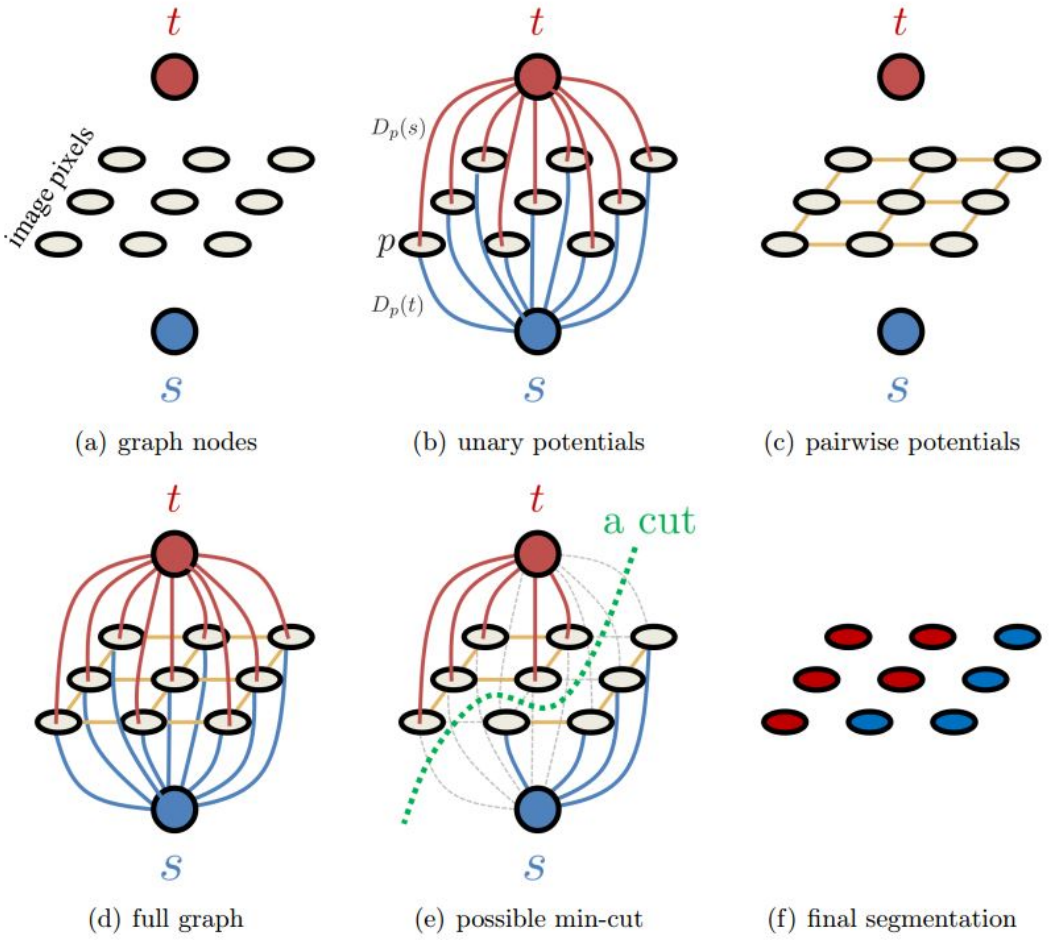


Figure 2.10: Graph construction and Graph Cut segmentation. (a) shows all nodes in the graph. (b) and (c) demonstrate the graphical representations of the t-links and n-links in the graph. (d) shows a fully constructed graph. A cut is illustrated in (e) with the severed edges painted grey and a segmentation of the cut is shown in (e). Image from [48].

The cost weight function w_{pq} describes the similarity in intensity between pixels p and q, namely w_{pq} is large when p and q possess similar chromatic features and close to zero when the adjacent pixels are very different.

2.2.3 Optimization via Graph Cuts

The graph cut algorithm provides a guaranteed global minimum solution in polynomial time for a certain type of energy functions, including Equation (2.12). This can be done by constructing a graph which encodes the data terms and the smoothness terms as edge weights and then finding a partition of the graph nodes associated with the lowest cut cost [6]. In a CNN-CRF system, graph cut can be applied as the segmentation algorithm after neural networks estimate the data terms, i.e., the probability of individual pixel label preference.

Graph construction

First, we establish the graphical representation of the energy minimization problem. Let $G = \{V, E\}$ be a graph consisting of a set of nodes V and a set of undirected[3] edges E. As shown in Figure 2.10 (a), the node set $V = V' \cup \{s, t\}$ consists of a node set V' where each node represents a image pixel and two special terminal nodes, the *source* s and the *sink* t.

An undirected edge (p, q) is called a *t-link* or *unary potential* if $p \in V'$ and $q \in \{s, t\}$. As in Figure 2.10 (b), for every $p \in V'$, both edges (p, s) and $(p, t) \in E$. In the case of binary segmentation, suppose the source s and the sink t represent the object label and the background label respectively, and then the edge weight w_{ps} of the t-link (p, s) can be interpreted as the cost of assigning the pixel p to be the background. Similarly, the t-link (p, t) encodes the penalty for assigning p to be the object. Intuitively enough, the cost of assigning pixel p to one label should positively correlate to the possibility for p to take the alternative label. In computation, these costs are estimated by the data terms, and the positive correlations can be observed in Equation (2.19).

$$\begin{aligned} w_{ps} &= D_p(t) = D_p(0) = -logP(I_p \mid 0) = -log(P(I_p) - P(I_p \mid 1)) \\ w_{pt} &= D_p(s) = D_p(1) = -logP(I_p \mid 1) = -log(P(I_p) - P(I_p \mid 0)) \end{aligned} \tag{2.19}$$

An undirected edge (p, q) is called an *n-link* or *binary potential* if $p, q \in V'$. Between each pair of adjacent pixels, an n-link is added to E to encode the penalty of label discontinuity as shown in Figure 2.10 (c). The edge weights are calculated based on the similarity of pixel features as in Equation (2.18).

The min-cut problem

An *s-t cut*, $C = \{S, T\}$, is a binary partition of graph nodes V, such that the source s and sink t are not in the same subset. That is, a cut separates the nodes in V into two disjoint subsets S and T, such that $s \in S$, $t \in T$ and $V = S \cup T$. The cut C can also be thought of as a set of edges, such that C is a subset of the graph edge set E and the edge set $E - C$ does not contain any path from source terminal s to sink terminal t. The cost of a cut is defined as the sum of the weights of all severed edges, and the min-cut problem is to find the cut for a graph with the smallest cost. The green dash line in Figure 2.10 (e) demonstrates the min-cut on the given graph where the different thicknesses of the edge lines represent the values of the edge weights and the severed edges in the cut are painted grey.

A group of polynomial time algorithms that are widely adopted to solve the min-cut problem is based on the Ford-Fulkerson method [17]. The idea behind this method is to recursively find augmenting paths from the source to the sink, until no path with available capacity can be found anymore. Then the maximum flow from the source s to the sink t was found, and the saturated edges divide the nodes into two disjoint subsets S, T which is the cut with the minimum total

[3]Here, we construct an undirected graph because the employed edge weight function is commutative. One can construct a directed graph if the weight of edge (p, q) differs from the weight of the reverse edge (q, p). Note that the cost of a cut on a directed graph changes if the terminals are swapped.

edge weight. Therefore, the max-flow and min-cut problems are equivalent, and the maximum flow passing from the source to the sink is equal to the total edge weight in the minimum cut[4]. For many vision applications where a graph model possesses the shape of two or high dimensional grid, Boykov and Kolmogorov [5] developed a fast augmenting path algorithm of linear running time to realize the Ford-Fulkerson method.

Submodularity

The graph cut algorithm can only provide globally optimized result for certain classes of energy functions. Kolmogorov and Zabih [33] have proved that a pairwise energy with binary labels can be optimized by graph cut if and only if the energy satisfies the submodularity criterion. A pairwise energy function is considered to be submodular if it satisfies the following inequality

$$V_{pq}(0,0) + V_{pq}(1,1) \leq V_{pq}(1,0) + V_{pq}(0,1) \quad \forall p, q \in N. \tag{2.20}$$

We can easily demonstrate that Equation (2.12) satisfies the above inequality using the definition of the pairwise terms in Equation (2.16).

$$\begin{cases} V_{pq}(0,0) + V_{pq}(1,1) = 0 + 0 \\ V_{pq}(1,0) + V_{pq}(0,1) = w_{pq} + w_{pq} \qquad \forall p, q \in N \\ w_{pq} \geq 0 \end{cases} \tag{2.21}$$
$$\Rightarrow \quad V_{pq}(0,0) + V_{pq}(1,1) \leq V_{pq}(1,0) + V_{pq}(0,1) \quad \forall p, q \in N$$

2.3 Combining CRFs with CNNs

Convolutional neural networks are often used to estimate the possibilities for a pixel to take on different labels and assign the label with the highest possibility to each pixel individually. Even though the overlaps of the receptive fields have the potential of encoding some connection between nearby pixels, the label dependencies are not explicitly modeled by CNNs. To fill this insufficiency, CRFs are often employed to refine the CNN-generated results either as a post-processing step [16, 9] or as a part of a trainable system [76].

The earliest and most intuitive approach to incorporate CRFs is to use them as a post-processing labeling strategy. The typical method is to construct a CRF energy function with the label assignment probability learned by a CNN as the unary term and a pairwise term based on appearance to enforce local consistency, and then this energy is minimized by some optimization algorithms, such as [5, 7, 34]. As early as in 2013, Farabet *et al.* [16] paired a multi-scaled CNNs with the CRF refinement as one of their proposed labeling strategies for scene labeling, and managed to produce more visually satisfying results than no refinement and superpixel refinement. Chen *et al.* [9] used a fully connected CRF to model the pixel dependency after obtaining the pixel-wise label probabilities for semantic segmentation, and the pairwise potentials were calculated using Gaussian kernels in different feature spaces, i.e, the RGB colour

[4]This is known as the max-flow min-cut theorem, and was proven by multiple individual work [17, 13] in 1956.

and the pixel positions. An approximate inference of the CRF was computed using the mean field approximation [34]. The same fully connected CRF system was also employed for salient object segmentation problem [39, 25] to improve spatial coherence.

Another way to integrate CRFs with CNNs is to include the CRF optimizer into the end-to-end training scheme of the neural networks. Zheng and Jayasumana [76] formulated the mean field inference [32] of the CRFs with Gaussian pairwise potentials as recurrent neural networks. However, the performance of this method is limited by the mean field approximation, whose effectiveness was shown to be arguably questionable [67].

2.4 Parameter learning for CRFs

Before CNNs are widely applied to image segmentation, probabilistic graphic models, including CRFs and Markov Random Fields (MRFs), are commonly used for modeling the connections among nodes which correspond to pixels or super-pixels in images. Here we summarize some popular approaches for CRF parameter learning.

One approach to CRF parameter learning is to use the classical method of maximum likelihood estimation [36]. In the context of CRFs, however, maximum likelihood estimation is often intractable due to the presence of the joint conditional distributions. So, various approximations, such as pseudo-likelihood [40], are used instead. When applying, many statistical assumptions are made about the underlying data by the approximation algorithm, which might over simplify the problem and influence the quality of the estimation.

Zhang and Seitz [74] formulated the stereo correspondence problem as a maximum a posterior (MAP) problem in which both a disparity map and CRF parameters were estimated from the stereo pair itself. More precisely, they learned CRF parameters using statistical estimation based on the expectation maximization (EM) algorithm. Their approach learned parameters on a per-image basis, but it was computationally intensive, as the proposed EM procedure had to be run for multiple iterations for estimation and the stereo algorithm was required to be executed at least once in each iteration.

In another attempt to choose the best regularization weight for each specific image, Peng and Veksler [51] first trained an AdaBoost [18] classifier to evaluate the quality of a segmentation given the original image. Given a new input image, they ran the CRF optimization multiple times with different regularization weights from a fixed pre-defined set, and employed the trained classifier to choose the segmentation with the best quality from the candidates to be the final output. This approach is intuitively effective, but very costly since both the CRF optimization and the AdaBoost classification need to be run several times for each input. In addition, the quality of the result may be limited by the pre-defined weight candidates. This empirical approach does not investigate the intrinsic relationship between the image properties and the optimal regularization weight.

Chapter 3

Learning coherence regularization weight for graph cuts using CNNs

Convolutional neural network has led the way in many pixel-wise labeling problems of vision, including salient object segmentation, semantic segmentation and scene understanding [56, 72, 1]. Essential to the success of CNNs is their ability to learn more abstract data representations and extract more compact features from images than traditional hand-crafted features [73]. However, when applied to dense prediction tasks, deep CNNs usually suffer from loosing spatial information with the loss of feature map resolution, which might result in the reduction of localized accuracy. For segmentation problems, this accuracy reduction mainly occurs in the boundary regions where more structural details are presented along with more noise.

As discussed in Section 2.3, a common approach to recover boundary accuracy in segmentation problems is to explicitly represent pixel dependencies with some graphical models, such as CRFs, and encourage spatial coherence. This involves formulating an energy function where the desired properties are encoded into one or several energy terms. Then the energy function is optimized and the labeling corresponding to its minimum gives the final segmentation.

Unlike in some previous work [39, 25] where fully connected CRFs were built and optimized using the mean field inference, we propose to build a CNN-CRF system where the CRF graph can be optimized by the graph cut algorithm as described in Section 2.2. A wide class of binary energy functions can be optimized by the graph cut algorithm [33]. For binary problems where pixels can take on one of the two possible labels, graph cuts yield an exact, globally optimal solution; while graph cuts cannot solve exactly the *multi-label* problems where pixels can be assigned with more than two distinct labels, but the produced approximation are usually near the global optimum [7]. The energy function we need to optimize for binary image segmentation belongs to the class of problems that can be optimized exactly [33]. Therefore, our proposed system benefits from the optimal solution generated by graph cuts, whereas the mean field inference used in [39, 25] is not a very effective optimization algorithm [67].

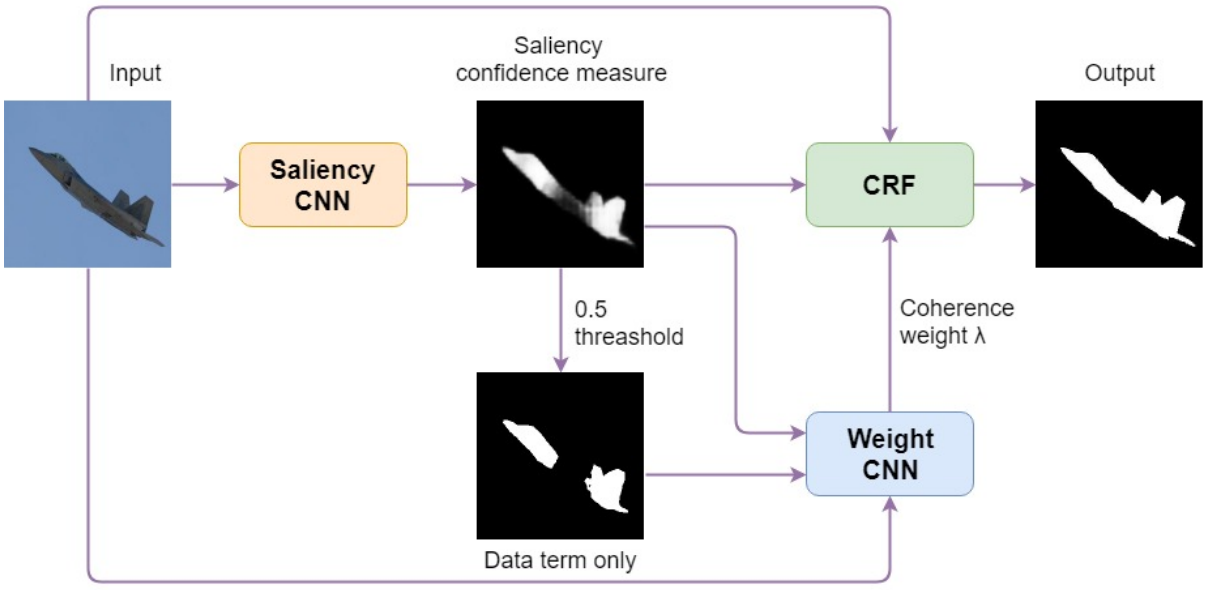


Figure 3.1: Our CNN-CRF integration system.

An important but open problem in graph cut segmentation is parameter selection. Different choices of parameter values might result in large differences in the performance of the segmentation result. The standard approach is to select a *fixed* value for these parameters when running the graph cut segmentation algorithm on an entire dataset. This fixed value is set by hand, learned from the training data, or a combination of both. However, a segmentation with better visual performance can be expected if the regularization weight of coherence is chosen adaptive to each specific image.

This work investigate the possibility of using a CNN to choose an appropriate parameter value on a *per-image* basis; the focus of this work is on the selection of the regularization weight λ for the graph cut algorithm in the context of salient segmentation problems.

3.1 CNN-CRF system

We propose a simple system to integrate the CNN and the CRF as outlined in Figure 3.1, where the CRF is used to post process the CNN-generated results. The network to learn the saliency probability measure is referred to as *saliency-CNN*, which can be substituted by any convolutional neural network that generates dense predictions for saliency segmentation. Saliency-CNN plays a key role in our overall system, as the quality of the saliency probability casts a direct influence on the value of the regularization weight. More details on the saliency network we use for this work are provided in Section 3.2.1. *Weight-CNN* is the network learning the appropriate strength of the regularization based on information possessed by each image, and it helps to predict a regularization weight that can be used in the energy function of the CRF.

3.2 Dataset construction

To solve the problem of learning an appropriate value for the regularization weight λ, one first needs to obtain a dataset from which a learning model can be trained on. Due to the novelty of the problem, we can not find any benchmark dataset and have to construct one for this new task. We start with a standard salient object segmentation dataset, which contains RGB images and the ground truth segmentations of typically one salient object labeled by hands. We augment these salient benchmark datasets to include more components.

The two critical components in our augmentation are a saliency measure map that evaluates the probability of assigning a pixel as foreground object, and the desired λ values with respect to the above saliency measure. The desired λ values serve as the output labels in our dataset and are referred to as λ_{opt} in this thesis. The calculation of the saliency measure is important because the quality of the probability map directly influence the desired λ values. The more accurate the probability map is, the less coherence information should be considered by a segmentation algorithm and the smaller the optimal λ value tends to be. In an ideal case, if we could obtain pixel-wise probability maps perfectly distinguishing the object from the background, we would not need to consider coherence between neighboring pixels and set λ value to be 0. However, in real applications, it is almost impossible to consistently obtain perfect saliency measure and a segmentation algorithm needs to consider blob coherence and boundary smoothness. Also, the saliency probability measures provide the data term in the energy function of graph cut image segmentation.

3.2.1 Acquisition of the saliency probability measures

As mentioned in the previous section, the network for saliency prediction in our CNN-CRF system, Saliency-CNN, can be substituted by any CNNs trained to give dense saliency probability measures. With the increased use of CNNs in various computer vision problems, many CNN architectures have been proposed specifically for salient object segmentation [29, 75, 65, 38, 39] and have achieved competitive performances in different saliency benchmarks. We chose to adopt a VGG-16 based multi-scale fully convolutional network (MS-FCN) to obtain the saliency probability measures after examining different network architectures based on criteria, such as the rationality and effectiveness of structures, performance, reproducibility.

MS-FCN was proposed by Li and Yu [39] in their salient object detection work as a substream that infers pixel-level saliency maps directly from the raw input images. Figure 3.2 illustrates the network architecture of MS-FCN. In MS-FCN, multi-scale feature maps are extracted by a horizontal cascade structure where the shallower side outputs capture the spatial information and the deeper side outputs infer the abstract properties of salient objects. Feature maps of different scales are then fused together by a convolutional layer with 1×1 kernels. Different strides, "a trous algorithm" [24, 9], and bilinear interpolation are adopted to preserve, maintain or recover the resolution of the multi-scale feature maps.

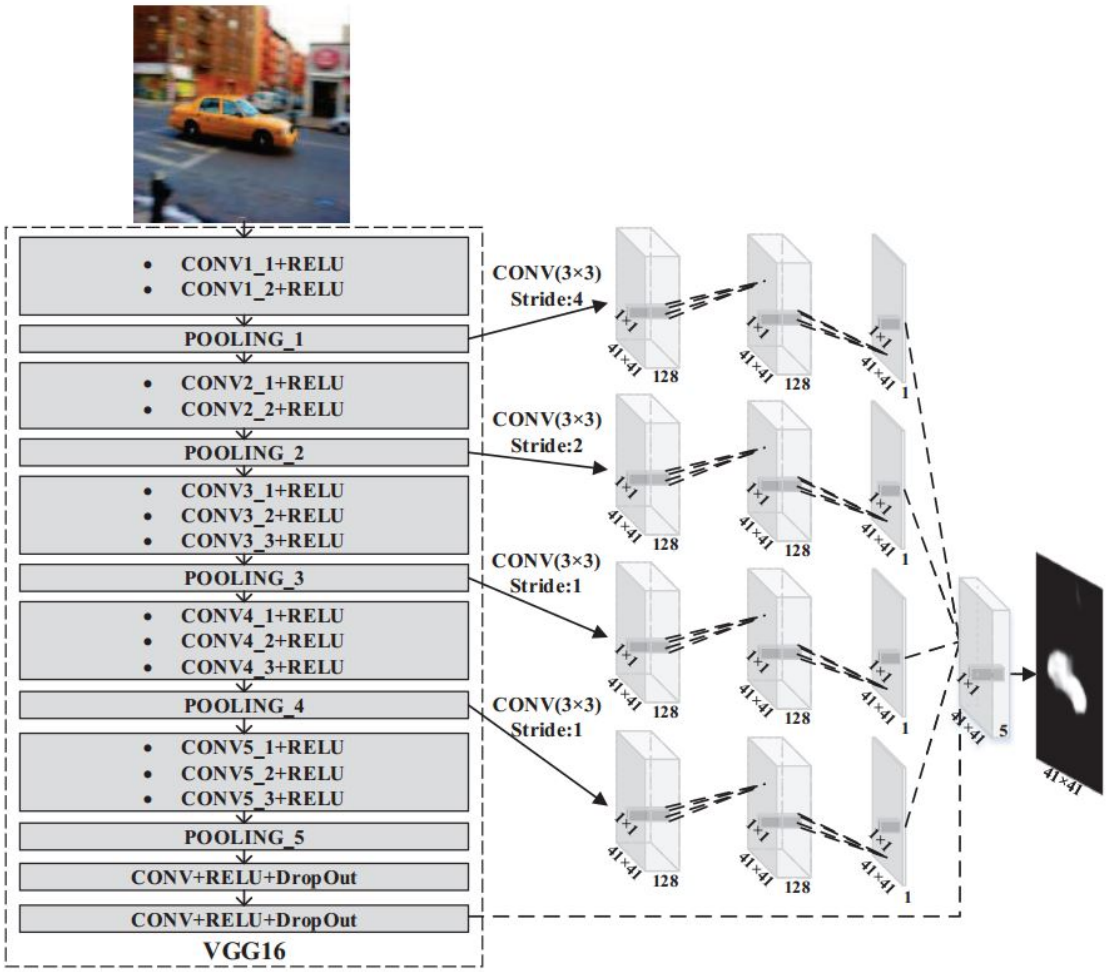


Figure 3.2: The architecture of MS-FCN. Image from [39]

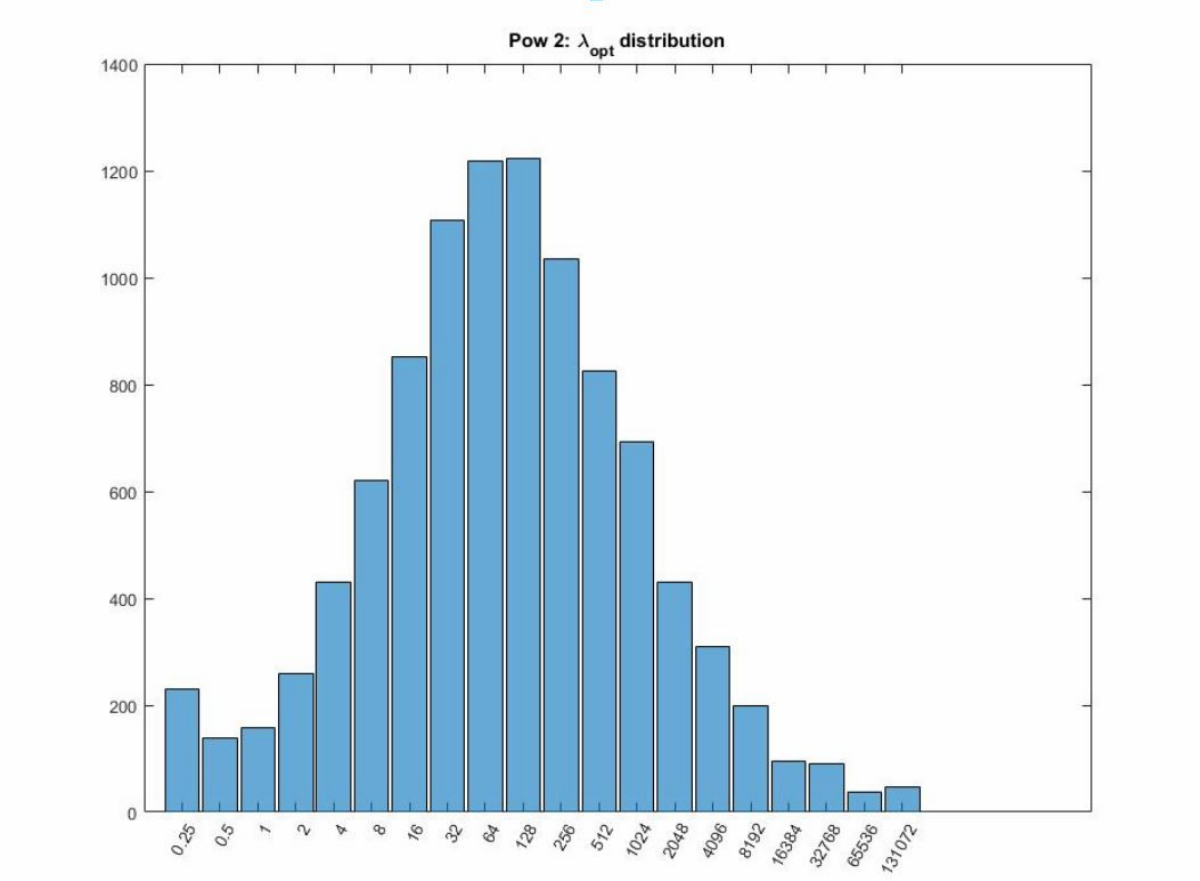


Figure 3.3: The numbers of images assigned with each λ_{opt} label. The image samples are from MSRA dataset of size 10K. The upper and lower limits of the λ_{opt} spectrum are 2^{-2} and 2^{17} respectively. The λ_{opt} label that fits the largest number of images is $128 = 2^7$, which fits 1224 images. The λ_{opt} label that fits the least number of images is $65536 = 2^{16}$.

3.2.2 Acquisition of the ground truth regularization weight

The value of λ controls the smoothness of the segmentation boundaries, i.e., how many details and noises will be included in the segmentation boundary. An appropriate λ value encourages boundary smoothness and suppresses noises caused by inaccurate saliency probability measure. However, if λ is set to be too big, the details on the object boundary might be lost. On the contrary, if the λ value is set to be too small, too many noises will be captured, which leads to over-segmented result.

In order to limit the search space when determining a ground truth value of λ, we decide to test a fixed set of values which we refer to as λ *spectrum*. We execute the graph cut algorithm multiple times with every value from the set $\{2^i\}$ where i is every integer in $[-2, 17]$. The ground truth label λ_{opt} is chosen to be the value that generates the best F-measure evaluation for each specific image.

We choose the lower and upper limits of the spectrum empirically: we determine that λ_{opt} rarely drops below 2^{-2} and rarely rises above 2^{17}. We decide that the interval between each value of λ in our spectrum should grow exponentially because the *relative* error seems to affect our results more than *absolute* error. For example, the difference between $\lambda = 10$ vs $\lambda = 11$ was observed to matter far more than that of $\lambda = 10000$ vs $\lambda = 10001$. The distribution of the calculated λ_{opt} values of segmentation dataset MSRA10K [44] is shown in Figure 3.3.

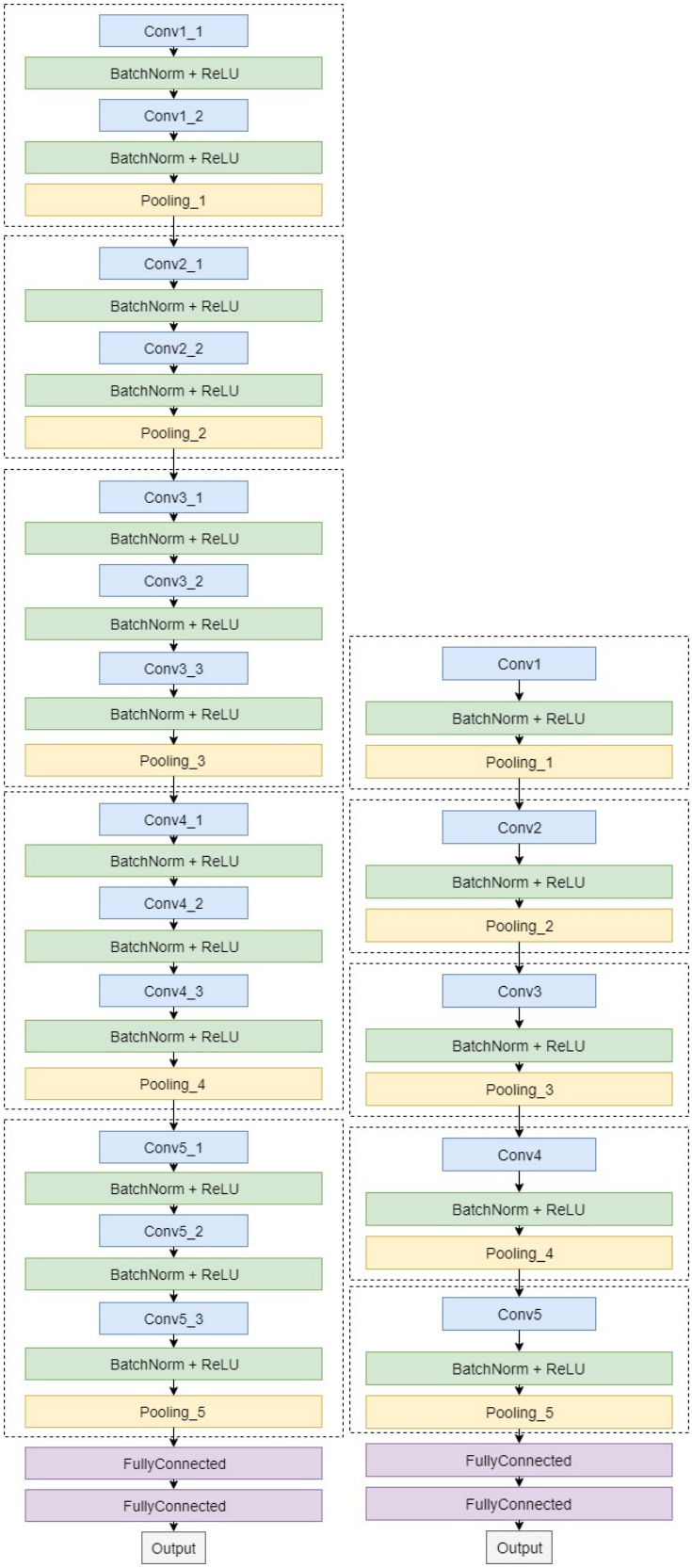
3.3 Network architectures

We exploit two regression network architectures for learning the regularization parameter λ in the CRF energy function on a per-image basis. Both networks are alterations of the Oxford VGG-16 classification nets [61]. We adopt the main characteristics of VGG-16 network structure because VGG-16 and its variations have achieved remarkable performance in various vision tasks other than object classification which VGG-16 was developed for. Their recent success in semantic segmentation [46, 49] demonstrated the expressive power and the potential of VGG-16 structure to learn for other vision tasks.

We modify the VGG-16 net to our Weight-CNN by reducing the number of neurons in the output layer from the number of classes (for classification purpose) to one which outputs a real scalar value serving as the exponent of the predicted regularization weight value. We take the number of neurons in the fully connected layers as a hyperparameter to tune during training. We preserve the 5 max pooling layers with stride 2 in the original VGG-16 net. This is to reduce the feature map resolution in the hope of extracting more compact, positional invariant feature representations. Each convolutional layer has 3×3 kernels with stride 1, and the number of kernels grows exponentially between each convolutional group as in the VGG nets. Zero padding is employed before every convolutional layers for easy calculation of dimension changes.

One major difference between our two networks is the number of convolutional layers before each pooling layer. As shown in Figure 3.4 (a), the full Weight-CNN network (Full-WCNN) has 2 to 3 convolutional layers in each *stage*. A *stage* is a group of network structures that consist of one or more convolutional layers, batch normalization layers, ReLU activation layers, and followed by a max pooling layer. For the trimmed Weight-CNN network (Trimmed-WCNN) in Figure 3.4 (b), there is only 1 convolutional layer in each stage. Additionally, in Full-WCNN, the two fully connected layers have the same number of neurons; meanwhile, in Trimmed-WCNN, the second fully connected layer has half as many neurons as in the first fully connected layer.

Trimmed-WCNN network is much slimmer than Full-WCNN in term of the number of trainable weights, which forces Trimmed-WCNN to learn more efficiently and reduces the potential overfitting problem. The slimness also significantly contributes to the training time reduction. Holding other configurations consistent, Trimmed-WCNN networks generally needed a third less time to train than Full-WCNN.



(a) Full-WCNN (b) Trimmed-WCNN

Figure 3.4: The architectures of the two regression networks.

Based on empirical experiments, we decide to include 5 input channels for Weight-CNN including the RGB channels of an image, the saliency probability map and a tentative segmentation obtained by segmenting the saliency probability map using a 0.5 threshold. The output of Weight-CNN is a real scalar value, which serves as the exponent of the predicted regularization weight. Let $\hat{y}$ denote the raw network output, then the λ value that should be used as the regularization weight for graph cut algorithm is

$$\lambda_{pred} = 2^{\hat{y}}. \tag{3.1}$$

The networks are trained to minimize the mean squared logarithmic error (MSLE) and a L_2 weight decay term. Let L denote the ground truth label of λ_{opt} selected from a spectrum of exponential growth, let $\hat{y}$ denote the raw network output, then, for a training batch of size n, the MSLE loss function can be expressed as

$$L_{MSLE} = \frac{1}{n}\sum_{i=1}^{n}(log(L_i) - \hat{y}_i)^2. \tag{3.2}$$

We choose the mean squared logarithmic error instead of mean squared error because we want to penalize the relative error instead of the absolute error. We do not want a small number of training samples with large scales to dominate the training process, especially considering our λ_{opt} had been obtained from a search with exponential increment.

Chapter 4

Experiments

4.1 Dataset

MSRA10K is a popular salient segmentation dataset introduced by Cheng and Borji *et al* [11, 12, 2], and it proposed a solution to the problem that traditional salient object datasets with bounding box annotations only provide coarse evaluations for the salient segmentation algorithms. In the contrast, MSRA10K provides pixel-level saliency labeling for 10,000 images from MSRA Salient Object Dataset [44].

We choose MSRA10K for our experiments because of its large size. By the time our experiments were conducted, MSRA10K was the largest publicly accessible salient segmentation dataset. MSRA10K consists of a variety of images, including challenging samples shown in Figure 4.1 (b) where the boundary is richly detailed, Figure 4.1 (c) where the pixels of the salient object have colors similar to the background pixels colors, and Figure 4.1 (e) where the boundary between the salient object and the background is weak. The richness of the dataset is also helpful to reduce the potential overfitting when we experiment with different network structures.

(a) (b) (c) (d) (e) (f)

Figure 4.1: Image examples from MSRA10K dataset with their ground truth segmentation.

We augment MSRA10K dataset with saliency probability measures and λ_{opt} for each image as described in Section 3.2. We feed RGB images through the MS-FCN network to obtain the saliency probability maps and, for the purpose of easy calculation, we re-size the RGB images, ground-truth segmentation labelings and their probability maps to a universal dimension 256×256 using bilinear interpolation. Then we run the graph cut algorithm multiple times with an extensive set of the regularization weight values and select the weight value that gives the best performance on the evaluation metric to be our λ_{opt}.

4.2 Evaluation metrics

Selecting appropriate metrics is a critical step for the evaluation of an algorithm. As a direct measure of the performance, the accuracy is the most intuitive choice. Besides, there are other measures of quality that are commonly adopted to compare segmentation results. In this section, we provide an overview of the quality measures that we have considered, and the main measure we select for the effectiveness analysis of learning the regularization weight with our Weight-CNN.

Let $c \in N$ be the number of class labels, n_{ij} $i, j \in 1, ..., c$ be the number of pixels which belong to class i and are labeled as class j.

- Accuracy: A direct measure to show the correctness of a labeling, which is defined as the number of pixels classified correctly divided by the total number of pixels in the image:

$$\text{accuracy} = \frac{\sum_{i=1}^{c} n_{ii}}{\sum_{i=1}^{c} \sum_{j=1}^{c} n_{ij}}. \tag{4.1}$$

- Precision: A measure that can be considered as evaluating the exactness or quality, and is defined as:

$$\text{precision}_i = \frac{n_{ii}}{\sum_{j=1}^{c} n_{ji}}. \tag{4.2}$$

- Recall: A measure that can be considered as evaluating the completeness or quantity, and is defined as:

$$\text{recall}_i = \frac{n_{ii}}{\sum_{j=1}^{c} n_{ij}}. \tag{4.3}$$

- F-measure: A widely-used measure for binary labeling, which considers both precision and recall, and is defined as:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{recall}) + \text{precision}}, \tag{4.4}$$

where we chose a value of $\beta^2 = 0.3$. Recall that, in this thesis, we used F-measure as our criterion to choose the ground truth for λ values.

Accuracy is the most intuitive performance measure but it might leave a biased impression of performance for problems that have unbalanced classes. For example, in the edge detection problem, the number of pixels that are actually edges is typically less than 5% of the total number of pixels in an image, which means an algorithm that blindly assigns every pixel to the non-edge label achieves an accuracy of more than 95%. Obviously, a constant labeling algorithm is insufficient in this case. For a similar reason, we usually prefer a quantitative measure that is more sensitive to the labelling of the object pixels in saliency segmentation because the fraction of pixels representing the salient object in most images is much smaller than the fraction of background pixels. The F-measure metric is more preferable when the problem involves an uneven class distribution because it takes into account both the precision and recall measures. Therefore, in this thesis, we focus our result analysis on the comparison of the F-measure scores.

4.3 Details of learning

For training Weight-CNN, we randomly partition the augmented MSRA10K dataset into the training set, validation set and testing set, and each contains $8,000$, $1,000$ and $1,000$ images respectively. During the training stage, the image mean is calculated from the training set and subtracted from the training samples. The training set is randomly shuffled after every time an epoch has been completed. An *epoch* is one complete presentation of all data samples in the training set to the learning networks.

The networks are trained with the Adam optimizer to minimize a loss function consisting of MSLE loss and L_2 weight decay with a weight decay factor of 0.0005. The mini-batch size is set to 20 for one training step. We set the training time to be at least 40K steps, which is 100 epochs. This is usually more than enough for the validation loss to show any signs of overfitting. Most of the time, the lowest value of validation loss occurs before 50 epochs and many of them occur within 25 epochs. We train for longer to make sure that training time is not a limiting factor of the performance.

Weight initialization

Training algorithms for neural networks are usually iterative and require some initialization to specify the starting point of the optimization. The initialization strategies can heavily affect the training time and the final result. For the training of Weight-CNN, we adopt the weight initialization that draws samples from a truncated normal distribution centered on 0 with the standard deviation of 0.02.

We also attempted to exploit the pre-trained weights for VGG nets [61] in the hope of taking advantage of the powerful features learned for classification tasks and reducing training time. However, due to the non-negligible differences in the nature of the problems, the random initialization and the initialization with pre-trained weights showed insignificant difference in performance.

Learning rate decay

The learning rate is a hyperparameter that controls how much the weights of the network should be adjusted according to the gradient of the loss function. If the learning rate is too small, gradient descent can be very slow and the learning is not effective enough with respect to training time. If the learning rate is too large, gradient decent can overshoot the minimum and fail to converge. When training a network, it is common to lower the learning rate as the training progresses. We apply an exponentially decayed learning rate with a decay factor of 0.96 and a lower bound of 10^{-5}. A lower bound on learning rate is set to ensure networks are still responsive to training in the later stage. The initial learning rate we use for training Weight-CNN is 10^{-4}, which is determined by the validation data during the parameter selection process in Section 4.6.

Dropout regularization

One of the biggest challenges we encounter during training Weight-CNN is to overcome the problem of overfitting. Given the limited number of training samples and training resources, it is very easy for a powerful network model to overfit the training samples. Applying dropout to both convolutional layers and fully connected layers is one of the most effective methods we experiment with to avoid overfitting. The dropout rates we impose on the convolutional layers and the fully connected layers are 0.1 and 0.3 respectively.

4.4 Results

In this section, we demonstrate the performance comparison of the per-image learned λ and the fixed λ chosen by the training set. The model hyperparameters are selected by the mean squared logarithmic error on the validation set. More details about training and hyperparameter selection are provided in Section 4.3 and 4.6. Table 4.1 compares the F-measures of different λ acquisition methods evaluated on the test set of the augmented MSRA10K.

Method		F-measure	Improvement ($\times 10^{-2}$)
Variable λ	(Ground truth)	0.94391557	3.6047
Variable λ	(Full-WCNN)	0.91660978	0.8741
Variable λ	(Trimmed-WCNN)	0.91550659	0.7639
Fixed λ	($\lambda_{fixed} = 64$)	0.90786851	0

Table 4.1: F-measures of different λ acquisition methods evaluated on the MSRA10K test set and improvements relative to λ_{fixed}

As shown in the last row of Table 4.1, the F-measure is 0.9079 when a fixed regularization weight is used for all images in CRF optimization. The fixed regularization weight is chosen to be the value that generates the best average F-measure for all images in the training set. This can serve as a baseline for our comparison and the improvements of using variable λ values are calculated with respect to this value. The F-measure in the first row defines the

biggest improvement we could possibly achieve using the given set of ground truth λ_{opt} values, which is 3.6×10^{-2}. Between the two boundaries are the F-measures evaluated using the variable λ values predicted by our best performing Full-WCNN and Trimmed-WCNN. Our best performing Full-WCNN achieved a F-measure of 0.9166, which is 24% of the total possible improvement.

4.5 Effectiveness of learning

4.5.1 Distribution similarities between λ_{opt}, λ_{pred} and λ_{fixed}

To provide more quantitative analysis on the effectiveness of Weight-CNN learning, we measure the similarity between the distributions of the ground truth label λ_{opt} and the learned regularization weight λ_{pred}, and compared it with the similarity between the distributions of λ_{opt} and the fixed weight λ_{fixed}.

We model the distributions with histograms as shown in Figure 4.2 (a)-(c). The distribution of λ_{pred} manifestly estimates the distribution of the ground truth λ_{opt} better than that of λ_{fixed}, as illustrated in Figure 4.2 (d) where the distributions are placed in the same scale.

The similarities between histograms are measured numerically by some bin-to-bin distances and histogram intersection. The bin-to-bin distances are calculated by the differences of the corresponding bins in the compared histograms. Let $\mathbf{h} = \{h_i\}_{i=1}^{n}$ be a histogram with n bins where h_i denote the count of the i-th bin. In this thesis, we examined the L_1 distance and the χ^2 distance [54] between two histograms of interest. The L_1 distance is defined as:

$$d_{L_1}(\mathbf{h1}, \mathbf{h2}) = \sum_{i=1}^{n} |h1_i - h2_i|. \tag{4.5}$$

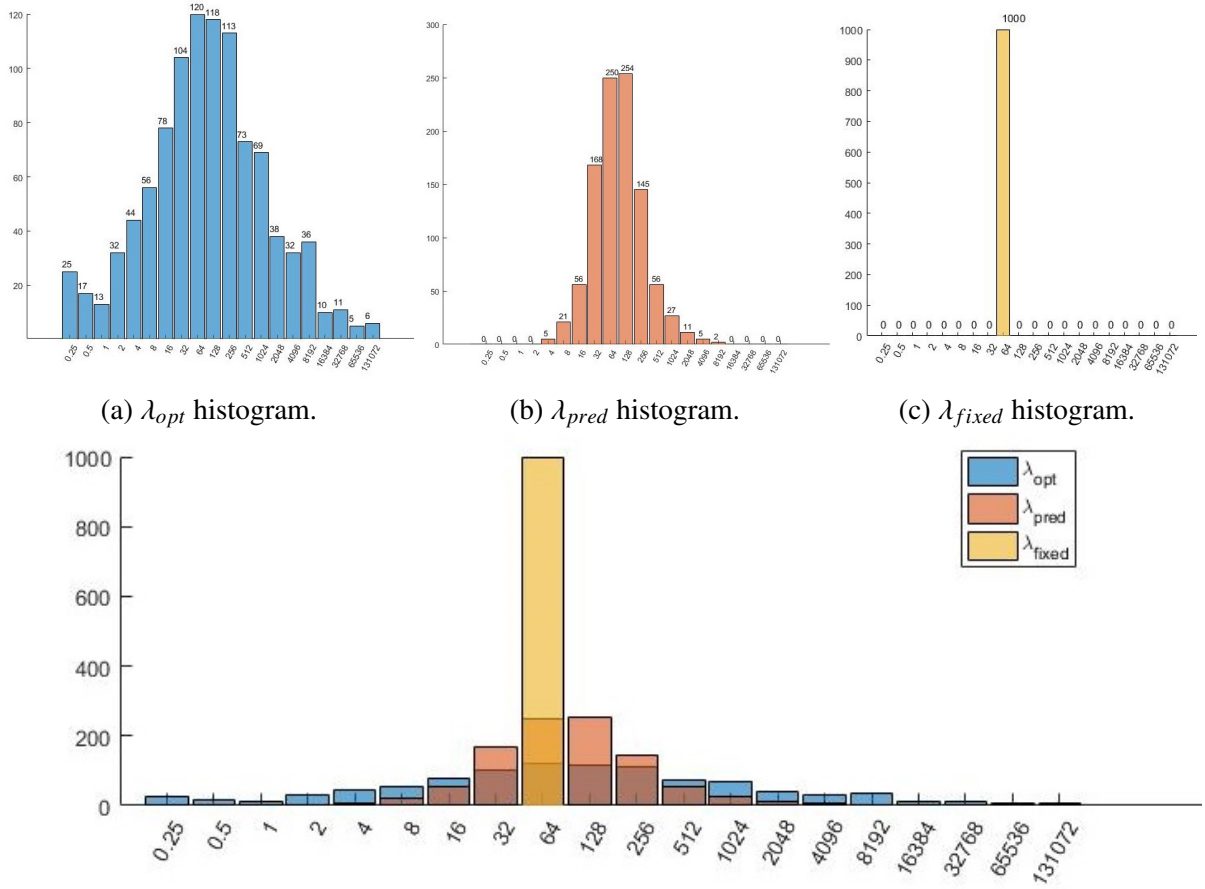
The χ^2 distance can be calculated as:

$$d_{\chi^2}(\mathbf{h1}, \mathbf{h2}) = \sum_{i=1}^{n} \frac{(h1_i - h2_i)^2}{h1_i + h2_i}. \tag{4.6}$$

The above mentioned distances are computed for the histograms of λ_{opt}, λ_{fixed} and the histograms of λ_{opt}, λ_{pred}, and the results are listed in Table 4.2. Besides the bin-to-bin distances, we also used the histogram intersection to compare the similarities of the distributions of λ_{opt}, λ_{fixed} and the distributions of λ_{opt}, λ_{pred}. The histogram intersection can be expressed as:

$$\mathbf{h1} \cap \mathbf{h2} = \sum_{i=1}^{n} min(h1_i, h2_i). \tag{4.7}$$

Unlike the distance measures, a bigger value of the histogram intersection suggests a stronger similarity. The numerical similarity measures in Table 4.2 support our observations that λ_{pred} learned by Weight-CNN gives a better estimation of λ_{opt} than the fixed λ_{fixed}.



(a) λ_{opt} histogram. (b) λ_{pred} histogram. (c) λ_{fixed} histogram.

(d) Comparison in the same scale.

Figure 4.2: The histograms of the number of images preferring different regularization weights. The horizontal axis lists out the bin values of regularization weights, and the vertical axis shows the image counts in the test set of the MSRA10K dataset.

		L_1 distance	χ^2 distance	Histogram intersection
λ_{opt} & λ_{opt}	(Perfect match)	0	0	1000
λ_{opt} & λ_{pred}		724	369.6024	638
λ_{opt} & λ_{fixed}	($\lambda_{fixed} = 64$)	1760	1571.4286	120

Table 4.2: The numerical similarity comparison of the λ_{opt}, λ_{pred} and λ_{fixed} histograms.

4.5.2 Extensive evaluation: performance on other datasets

An effective learning model should have steady performance on different benchmark datasets. As another main contribution of this thesis, we evaluate the Weight-CNN model, which is trained with the MSRA10K dataset, on another 3 representative saliency segmentation datasets, including PASCAL [42], ECSSD [69], and DUT-OMRON [70].

PASCAL was built from the validation set of the PASCAL VOC 2010 segmentation dataset, and contains 850 images with 20 object categories and complex scenes. The result of Weight-CNN is displayed in Table 4.3.

Method		F-measure	Improvement ($\times 10^{-2}$)
Variable λ	(Ground truth)	0.85261991	5.8582
Variable λ	(Full-WCNN)	0.79429388	0.0256
Variable λ	(Trimmed-WCNN)	0.79748042	0.3443
Fixed λ	($\lambda_{fixed} = 64$)	0.79403781	0

Table 4.3: F-measures of different λ acquisition methods evaluated on the PASCAL dataset and the improvements relative to λ_{fixed}

ECSSD consists of 1000 semantically meaningful images obtained from various sources. An interesting characteristic of this dataset is that it contains a large proportion of images with natural scenes, including some natural camouflage. The performance of Weight-CNN on this dataset is illustrated in Table 4.4.

Method		F-measure	Improvement ($\times 10^{-2}$)
Variable λ	(Ground truth)	0.92567453	3.7746
Variable λ	(Full-WCNN)	0.89112503	0.3196
Variable λ	(Trimmed-WCNN)	0.89188917	0.3960
Fixed λ	($\lambda_{fixed} = 16$)	0.88792895	0

Table 4.4: F-measures of different λ acquisition methods evaluated on the ECSSD dataset and the improvements relative to λ_{fixed}

DUT-OMRON contains 5168 images which have one or more salient objects and relatively complex background. The scale of the salient objects varies greatly from image to image,

making DUT-OMRON one of the most challenging saliency benchmarks. The result of Weight-CNN on DUT-OMRON is listed in Table 4.5.

Method		F-measure	Improvement ($\times 10^{-2}$)
Variable λ	(Ground truth)	0.76539855	8.7199
Variable λ	(Full-WCNN)	0.68927901	1.1079
Variable λ	(Trimmed-WCNN)	0.68416296	0.5963
Fixed λ	($\lambda_{fixed} = 1$)	0.67819961	0

Table 4.5: F-measures of different λ acquisition methods evaluated on the DUT-OMRON dataset and the improvements relative to λ_{fixed}

4.5.3 Demonstration of result examples

We collect a few examples to visually illustrate the effectiveness of the CNN-learned regularization weights. In Figure 4.3, the regularization weights learned by our Weight-CNN significantly improve the visual quality of the segmentation results by imposing an appropriate amount of spatial coherence.

We also study the failure samples where the F-measures achieved by the per-image learned regularization weights λ_{pred} are less than 0.8, and the ones where λ_{fixed} outperforms λ_{pred}. Although we realize certain level of randomness exists in the occurrences of the failure samples, we observe two main reasons that often cause the per-image learned regularization weight to fail to give satisfying results. When the quality of the saliency probability measures is not reliable, the positive effect of coherence regularization is very constrained. In this case, as shown in the row 4-6 of Figure 4.4, neither λ_{pred} nor λ_{fixed} produces segmentation results with good visual quality. Saliency ambiguity is another reason that sometimes causes low F-measure scores for some images. As illustrated in the row 7-9 of Figure 4.4, when there are more than one object or region with strongly distinguishable characteristics appearing in an image, it is hard for the learning system, even for humans, to determine which blob should be considered as salient. For example, in row 10 of Figure 4.4, the segmentation result generated by the learning system makes semantic sense, but it does not agree with the ground truth.

4.6 Hyperparameter selection

Another influential factor of neural network training is hyperparameter tuning. The idea behind it is to optimize the model parameters by the training process: neural networks are trained with different combinations of hyperparameter values, and the parameter values are adjusted according to some accuracy measure of the resulting predictions evaluated on the validation set, until achieving the best performance. For training Weight-CNNs, we modify the network composition and training parameters to minimize the mean squared logarithmic error in the loss function.

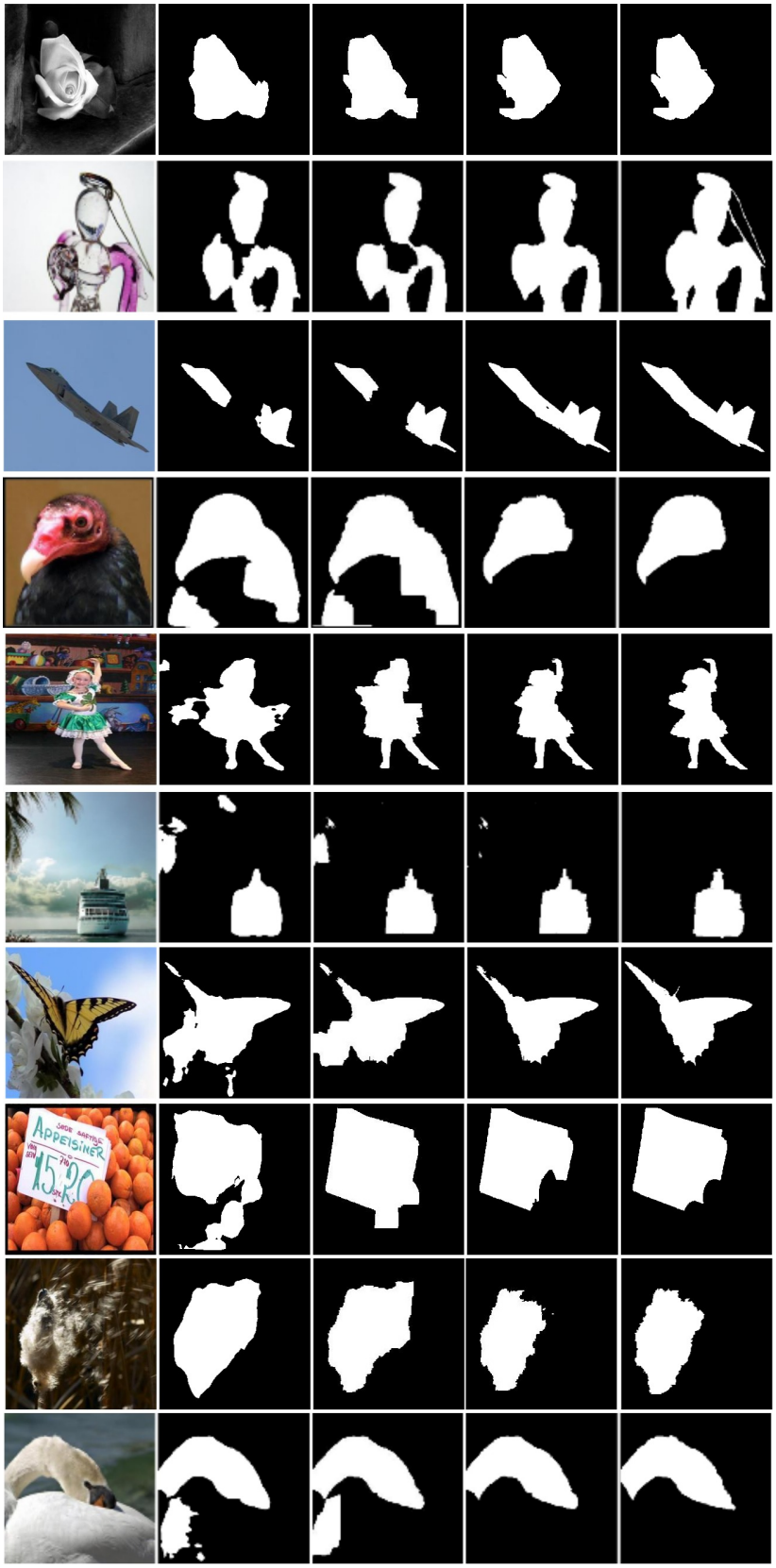


Figure 4.3: Some successful examples. From left to right: original image, segmentation produced by saliency-CNN, segmentation of CNN+CRF with the same regularization weight λ_{fixed} for all images, segmentation of CNN+CRF with λ_{pred} learned by our weight-CNN, ground truth

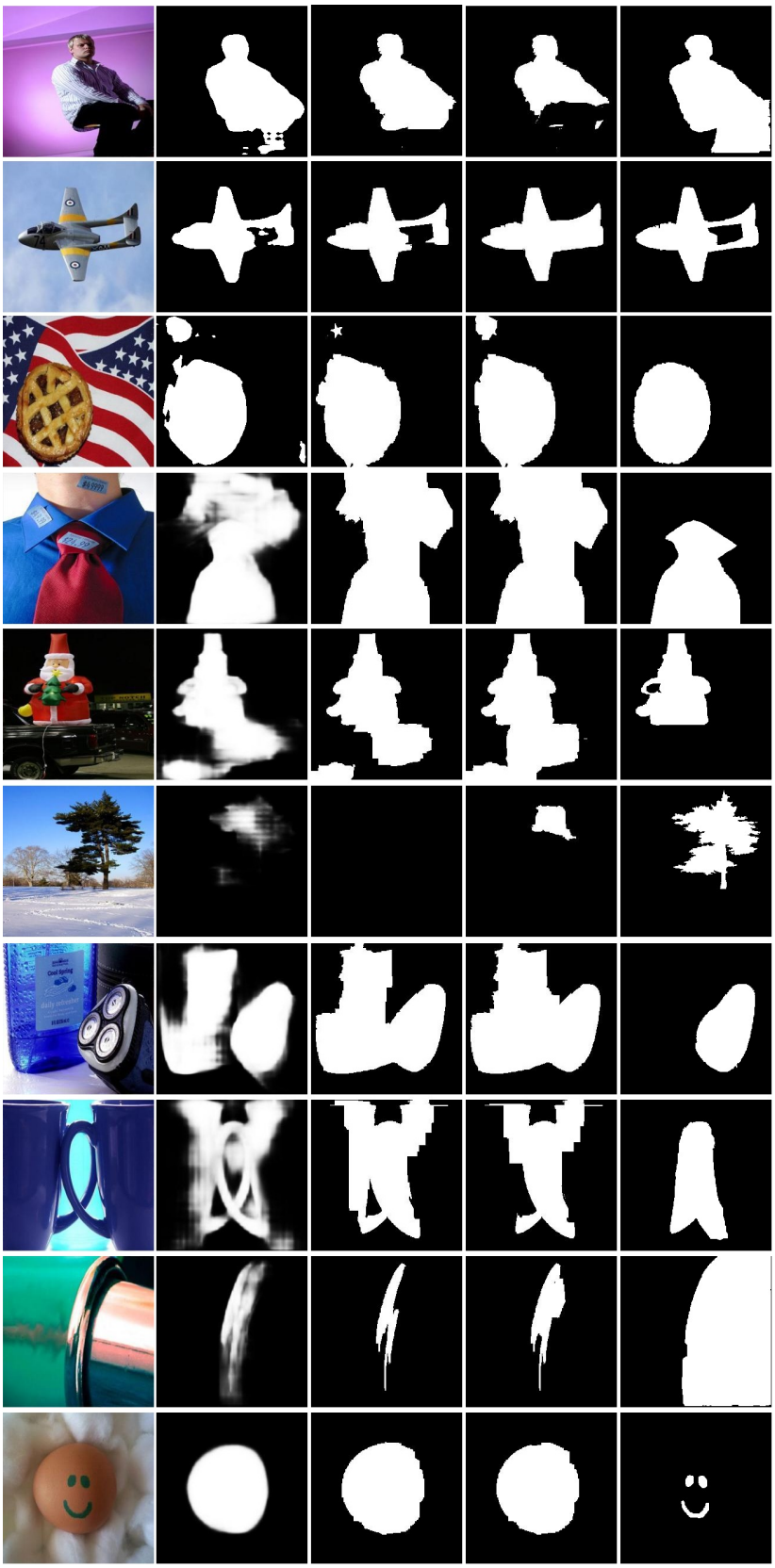


Figure 4.4: Some failure examples. From left to right: original image, saliency probability map produced by saliency-CNN, segmentation of CNN+CRF with the same regularization weight λ_{fixed} for all images, segmentation of CNN+CRF with λ_{pred} learned by our weight-CNN, ground truth

Init lr	# of fc units	MSLE loss	F-measure	accuracy
10^{-3}	(4096, 4096)	16.5415	0.9166	0.9540
10^{-3}	(1024, 1024)	11.3876	0.9242	0.9530
10^{-3}	(256, 256)	11.1328	0.9234	0.9530
10^{-3}	(64, 64)	11.0100	0.9224	0.9526
10^{-4}	(4096, 4096)	11.3814	0.9230	0.9525
10^{-4}	(1024, 1024)	10.9316	0.9203	0.9528
10^{-4}	(256, 256)	10.8673	0.9207	0.9516
10^{-4}	(64, 64)	**10.4809**	0.9239	0.9529

Table 4.6: Hyperparameter selection for Full-WCNN nets. Evaluation metrics are computed on the validation set of MSRA10K across different values of initial learning rate, number of neurons in the fully connected layers. The number of filters in convolutional layers are kept invariant as in VGG-16 nets. The parameters were selected based on MSLE loss.

The hyperparameter tuning process can be very computationally costly as the amount of time needed for training and evaluating the network grows exponentially with respect to the number of parameters considered for tuning. Limited by time and computational resource, we only made in-depth investigation of the following hyperparameters: the initial learning rate, the number of filters in each convolutional layers and the number of neurons in the fully connected layers. To determine an appropriate initial value of the learning rate, we have tried selective network structures with initial learning rate of $\{10^i\}, i \in [-8, -2]$ and narrowed down our searching space to $[10^{-4}, 10^{-3}]$. The results of hyperparameter selection are shown in Table 4.6 and Table 4.7.

4.7 Runtime

We measure the runtime of our Weight-CNN implementation on a computer with a single NVIDIA GeForce GTX 1070 graphics processing unit. Table 4.8 contains the runtime measurements across different datasets. The network training time varies from one experiment to another, depending on the network architectures and the parameter settings, but never exceeds three days.

Init lr	# of filters in conv	# of fc units	MSLE loss	F-measure	accuracy
10^{-3}	(64, 128, 256, 512, 512)	(2048, 1024)	11.4150	0.9218	0.9530
10^{-3}	(64, 128, 256, 512, 512)	(512, 256)	10.8145	0.9226	0.9533
10^{-3}	(64, 128, 256, 512, 512)	(128, 64)	10.6780	0.9242	0.9536
10^{-3}	(64, 128, 256, 512, 512)	(32, 16)	11.1171	0.9232	0.9510
10^{-3}	(32, 64, 128, 256, 256)	(2048, 1024)	11.1618	0.9227	0.9526
10^{-3}	(32, 64, 128, 256, 256)	(512, 256)	11.4883	0.9210	0.9500
10^{-3}	(32, 64, 128, 256, 256)	(128, 64)	10.9428	0.9235	0.9511
10^{-3}	(32, 64, 128, 256, 256)	(32, 16)	10.8208	0.9232	0.9526
10^{-3}	(16, 32, 64, 128, 128)	(2048, 1024)	11.0888	0.9247	0.9524
10^{-3}	(16, 32, 64, 128, 128)	(512, 256)	11.0671	0.9244	0.9533
10^{-3}	(16, 32, 64, 128, 128)	(128, 64)	11.1416	0.9209	0.9508
10^{-3}	(16, 32, 64, 128, 128)	(32, 16)	11.0723	0.9237	0.9534
10^{-4}	(64, 128, 256, 512, 512)	(2048, 1024)	11.2435	0.9222	0.9525
10^{-4}	(64, 128, 256, 512, 512)	(512, 256)	10.8728	0.9223	0.9513
10^{-4}	(64, 128, 256, 512, 512)	(128, 64)	10.8238	0.9237	0.9518
10^{-4}	(64, 128, 256, 512, 512)	(32, 16)	**10.4734**	0.9233	0.9518
10^{-4}	(32, 64, 128, 256, 256)	(2048, 1024)	11.1600	0.9227	0.9512
10^{-4}	(32, 64, 128, 256, 256)	(512, 256)	10.9624	0.9215	0.9511
10^{-4}	(32, 64, 128, 256, 256)	(128, 64)	10.9300	0.9224	0.9508
10^{-4}	(32, 64, 128, 256, 256)	(32, 16)	10.5843	0.9232	0.9523
10^{-4}	(16, 32, 64, 128, 128)	(2048, 1024)	10.9400	0.9227	0.9529
10^{-4}	(16, 32, 64, 128, 128)	(512, 256)	10.9840	0.9228	0.9521
10^{-4}	(16, 32, 64, 128, 128)	(128, 64)	10.7627	0.9229	0.9514
10^{-4}	(16, 32, 64, 128, 128)	(32, 16)	10.5237	0.9225	0.9531

Table 4.7: Hyperparameter selection for Trimmed-WCNN nets. Evaluation metrics were computed on the MSRA10K validation set across different values of initial learning rate, number of filters in the convolutional layers and number of neurons in the fully connected layers. The parameters were selected based on the MSLE loss.

Networks	MSRA10K	PASCAL	ECSSD	DUT-OMRON
Full-WCNN	0.0351	0.0349	0.0227	0.0345
Trimmed-WCNN	0.0151	0.0158	0.0155	0.0144

Table 4.8: The runtime, in seconds, required to compute the regularization weight using Weight-CNN.

Chapter 5

Conclusion and future work

5.1 Conclusion

In this thesis, we propose a novel deep learning approach to obtain a per-image based regularization weight for a CNN-CRF system, and apply the CNN-CRF system with the per-instance learned regularization weight to the problem of salient object segmentation.

The thesis is motivated by the observation that when applying CRF optimization algorithm to vision problems, the regularization weight is usually set to be a fixed number. However, better optimization result can usually be achieved if the regularization weight is chosen based on the features or contents of each specific sample instance.

In order to learn a per-image based regularization weight for the salient segmentation problem, we design a convolutional regression network, Weight-CNN, that takes as input a color image along with the saliency probabilities used as unary terms in the CRF and a segmentation computed solely with the saliency probability map. This is because the regularization weight depends not just on the image content, but also on the quality of the unary terms.

First, we prepare the dataset for regularization weight learning by augmenting the standard saliency datasets with the optimal weight values as the ground truth labels. The ground truth weight values was found by running the graph cut algorithm, the CRF optimizer we employ for this work, multiple times with a wide range of pre-defined regularization weights.

Next, we train the above-mentioned networks, Weight-CNNs, for regularization weight prediction. We adopt some features of the famous VGG classification net, which has a classic bottom-up network structure, and we adjust the network to meet our requirements. A combination of different regularization techniques are used to prevent overfitting, including L_2 weight decay, dropout, and batch normalization. Different network compositions and training parameters are tuned to find the best performer, and 8.74×10^{-3} improvement in F-measure has been achieved on the test set, which is around 25% of the total possible improvement defined by the ground truth performance.

Last, we analyze the effectiveness of learning by Weight-CNNs, and evaluate the performance of the per-image learned regularization weights on datasets which Weight-CNN is not trained on. By comparing distribution similarities, we find that Weight-CNNs are capable of capturing some intrinsic relationships between the image content, saliency probability measures and a good value of regularization weight. Consistent improvements of various scales are observed on all other public available saliency benchmarks, including some challenging datasets. These observations give us the confidence to conclude that the regularization weights learned on a per-image basis by our Weight-CNN consistently outperform the fixed regularization value set by the conventional approach.

5.2 Future work

Learning regularization weights for CRF optimization on a per-instance basis is quite a new problem. Even our Weight-CNN manages to outperform the conventional way to set the weight value, we believe there is much room for further improvement. In this section, we discuss some potential improvements to our current work and suggest possible future research directions.

5.2.1 Obtaining more accurate ground truth labels

Regardless of our current best effort, the accuracy of the ground truth regularization weights has a great potential to be improved. In this work, we have tried various ways to decide the optimal value of regularization weights, and decide the acquisition method based on empirical results.

Further investigation is needed on the searching range and searching method to obtain more accurate ground truth labels, and the learning models can benefit from the noise reduction in training data. Using evaluation metrics other than the F-measure alone to determine the optimal regularization weight is another potential approach to improve the quality of the ground truth label.

5.2.2 Exploring different types of network architectures

In this work, we focus our investigation solely on the traditional bottom-up neural networks to learn the regularization weights. As explained in Section 3.3, as the first work for this problem to our knowledge, we choose the bottom-up structure for its generality and extendibility to learn for different tasks. However, this may be a limiting factor for pushing the network performance for a specific task. There might be other types of network or network structures more suitable for regularization weight learning, such as recurrent neural networks [50] and multi-task neural networks [41].

5.2.3 Learning regularization weight in an end-to-end system

In our work, we present the great potential of learning regularization weights for CRF optimization using convolutional neural networks. Regardless of the consistent improvement and

short runtime for inference, the current approach has a non-negligible drawback: the regularization weight is trained on a separate network outside of the CNN-CRF system for saliency segmentation.

An interesting direction for further exploration is to integrate the regularization weight learning into the CNN-CRF system. One potential approach is to combine the network that learns for saliency prediction with the network that learns for regularization weight, and form a multi-task model [66]. The lower features are usually shared between two sub-nets in a multi-task network. The weight learning sub-net might benefit from being indirectly exposed to the segmentation ground truth labeling, which might provide valuable information for weight learning as well.

Bibliography

[1] Min Bai, Wenjie Luo, Kaustav Kundu, and Raquel Urtasun. Exploiting semantic information and deep matching for optical flow. In *European Conference on Computer Vision*, pages 154–170. Springer, 2016.

[2] Ali Borji, Ming-Ming Cheng, Huaizu Jiang, and Jia Li. Salient object detection: A survey. *ArXiv e-prints*, 2014.

[3] Ali Borji, Ming-Ming Cheng, Huaizu Jiang, and Jia Li. Salient object detection: A benchmark. *IEEE Transactions on Image Processing*, 24(12):5706–5722, 2015.

[4] Yuri Boykov and Gareth Funka-Lea. Graph cuts and efficient nd image segmentation. *International journal of computer vision*, 70(2):109–131, 2006.

[5] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE transactions on pattern analysis and machine intelligence*, 26(9):1124–1137, 2004.

[6] Yuri Boykov and Olga Veksler. Graph cuts in vision and graphics: Theories and applications. In *Handbook of mathematical models in computer vision*, pages 79–96. Springer, 2006.

[7] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 23(11):1222–1239, 2001.

[8] Neil Bruce and John Tsotsos. Saliency based on information maximization. In *Advances in neural information processing systems*, pages 155–162, 2006.

[9] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2018.

[10] Tianshui Chen, Liang Lin, Lingbo Liu, Xiaonan Luo, and Xuelong Li. Disc: Deep image saliency computing via progressive representation learning. *IEEE transactions on neural networks and learning systems*, 27(6):1135–1149, 2016.

[11] Ming-Ming Cheng, Niloy J. Mitra, Xiaolei Huang, Philip H. S. Torr, and Shi-Min Hu. Global contrast based salient region detection. *IEEE TPAMI*, 37(3):569–582, 2015.

[12] Ming-Ming Cheng, Jonathan Warrell, Wen-Yan Lin, Shuai Zheng, Vibhav Vineet, and Nigel Crook. Efficient salient region detection with soft image abstraction. In *IEEE ICCV*, pages 1529–1536, 2013.

[13] G Dantzig and Delbert Ray Fulkerson. On the max flow min cut theorem of networks. *Linear inequalities and related systems*, 38:225–231, 2003.

[14] Li Deng, Dong Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

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

[16] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1915–1929, 2013.

[17] Lester Randolph Ford Jr and Delbert Ray Fulkerson. *Flows in networks*. Princeton university press, 2015.

[18] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[19] Davi Frossard. Vgg in tensorflow, 2016.

[20] Anton Garcia-Diaz, Victor Leboran, Xose R Fdez-Vidal, and Xose M Pardo. On the relationship between optical variability, visual saliency, and eye fixations: A computational approach. *Journal of vision*, 12(6):17–17, 2012.

[21] Qichuan Geng, Zhong Zhou, and Xiaochun Cao. Survey of recent progress in semantic image segmentation with cnns. *Science China Information Sciences*, 61(5):051101, 2018.

[22] Glosser.ca. Colored neural network.

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

[24] Matthias Holschneider, Richard Kronland-Martinet, Jean Morlet, and Ph Tchamitchian. A real-time algorithm for signal analysis with the help of the wavelet transform. In *Wavelets*, pages 286–297. Springer, 1990.

[25] Qibin Hou, Ming-Ming Cheng, Xiaowei Hu, Ali Borji, Zhuowen Tu, and Philip Torr. Deeply supervised salient object detection with short connections. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5300–5309. IEEE, 2017.

[26] Qibin Hou, Jiangjiang Liu, Ming-Ming Cheng, Ali Borji, and Philip HS Torr. Three birds one stone: A unified framework for salient object segmentation, edge detection and skeleton extraction. *arXiv preprint arXiv:1803.09860*, 2018.

[27] Xun Huang, Chengyao Shen, Xavier Boix, and Qi Zhao. Salicon: Reducing the semantic gap in saliency prediction by adapting deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 262–270, 2015.

[28] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[29] Huaizu Jiang, Jingdong Wang, Zejian Yuan, Yang Wu, Nanning Zheng, and Shipeng Li. Salient object detection: A discriminative regional feature integration approach. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 2083–2090. IEEE, 2013.

[30] Andrej Karpathy. Cs231n convolutional neural networks for visual recognition. *Neural networks*, 1, 2016.

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

[32] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[33] Vladimir Kolmogorov and Ramin Zabin. What energy functions can be minimized via graph cuts? *IEEE transactions on pattern analysis and machine intelligence*, 26(2):147–159, 2004.

[34] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *Advances in neural information processing systems*, pages 109–117, 2011.

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

[36] Sanjiv Kumar and Martial Hebert. Man-made structure detection in natural images using a causal multiscale random field. In *Computer vision and pattern recognition, 2003. proceedings. 2003 ieee computer society conference on*, volume 1, pages I–I. IEEE, 2003.

[37] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[38] Guanbin Li and Yizhou Yu. Visual saliency based on multiscale deep features. *arXiv preprint arXiv:1503.08663*, 2015.

[39] Guanbin Li and Yizhou Yu. Deep contrast learning for salient object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 478–487, 2016.

[40] Stan Z Li. *Markov random field modeling in image analysis*. Springer Science & Business Media, 2009.

[41] Xi Li, Liming Zhao, Lina Wei, Ming-Hsuan Yang, Fei Wu, Yueting Zhuang, Haibin Ling, and Jingdong Wang. Deepsaliency: Multi-task deep neural network model for salient object detection. *IEEE Transactions on Image Processing*, 25(8):3919–3930, 2016.

[42] Yin Li, Xiaodi Hou, Christof Koch, James M Rehg, and Alan L Yuille. The secrets of salient object segmentation. Georgia Institute of Technology, 2014.

[43] Nian Liu and Junwei Han. Dhsnet: Deep hierarchical saliency network for salient object detection. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 678–686. IEEE, 2016.

[44] Tie Liu, Zejian Yuan, Jian Sun, Jingdong Wang, Nanning Zheng, Xiaoou Tang, and Heung-Yeung Shum. Learning to detect a salient object. *IEEE Transactions on Pattern analysis and machine intelligence*, 33(2):353–367, 2011.

[45] Xiaoqing Liu, Olga Veksler, and Jagath Samarabandu. Graph cut with ordering constraints on labels and its applications. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

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

[47] Bjoern H Menze, Andras Jakab, Stefan Bauer, Jayashree Kalpathy-Cramer, Keyvan Farahani, Justin Kirby, Yuliya Burren, Nicole Porz, Johannes Slotboom, Roland Wiest, et al. The multimodal brain tumor image segmentation benchmark (brats). *IEEE transactions on medical imaging*, 34(10):1993–2024, 2015.

[48] Karin Ng. Sol: Segmentation with overlapping labels. 2018.

[49] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1520–1528, 2015.

[50] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[51] Bo Peng and Olga Veksler. Parameter selection for graph cut based image segmentation. In *BMVC*, volume 32, pages 42–44, 2008.

[52] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[53] K. Krishna Prasad and P. S. Aithal. Fingerprint image segmentation: A review of state of the art techniques. *International Journal of Management, Technology, and Social Sciences*, 2(2):28–39, 2017.

[54] Jan Puzicha, Thomas Hofmann, and Joachim M Buhmann. Non-parametric similarity measures for unsupervised texture segmentation and image retrieval. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 267–272. IEEE, 1997.

[55] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pages 586–591. IEEE, 1993.

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

[57] Rahimeh Rouhi, Mehdi Jafari, Shohreh Kasaei, and Peiman Keshavarzian. Benign and malignant breast tumors classification based on region growing and cnn segmentation. *Expert Systems with Applications*, 42(3):990–1002, 2015.

[58] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[59] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[60] Rachid Sammouda, Nuru Adgaba, Ameur Touir, and Ahmed Al-Ghamdi. Agriculture satellite image segmentation using a modified artificial hopfield neural network. *Computers in Human Behavior*, 30:436–441, 2014.

[61] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[62] U Snekhalatha, M Anburajan, V Sowmiya, B Venkatraman, and M Menaka. Automated hand thermal image segmentation and feature extraction in the evaluation of rheumatoid arthritis. *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine*, 229(4):319–331, 2015.

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

[64] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[65] Lijun Wang, Huchuan Lu, Xiang Ruan, and Ming-Hsuan Yang. Deep networks for saliency detection via local estimation and global search. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 3183–3192. IEEE, 2015.

[66] Xiang Wang, Huimin Ma, Xiaozhi Chen, and Shaodi You. Edge preserving and multi-scale contextual neural network for salient object detection. *IEEE Transactions on Image Processing*, 27(1):121–134, 2018.

[67] Yair Weiss. Comparing the mean field method and belief propagation for approximate inference in mrfs. *Advanced mean field methods: theory and practice*, pages 229–240, 2001.

[68] Dijia Wu, Michal Sofka, Neil Birkbeck, and S Kevin Zhou. Segmentation of multiple knee bones from ct for orthopedic knee surgery planning. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 372–380. Springer, 2014.

[69] Qiong Yan, Li Xu, Jianping Shi, and Jiaya Jia. Hierarchical saliency detection. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 1155–1162. IEEE, 2013.

[70] Chuan Yang, Lihe Zhang, Huchuan Lu, Xiang Ruan, and Ming-Hsuan Yang. Saliency detection via graph-based manifold ranking. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 3166–3173. IEEE, 2013.

[71] Ezrinda Mohd Zaihidee, Kamarul Hawari Ghazali, and Ali Abd Almisreb. Comparison of human segmentation using thermal and color image in outdoor environment. In *Systems, Process and Control (ICSPC), 2015 IEEE Conference on*, pages 152–156. IEEE, 2015.

[72] Jure Zbontar and Yann LeCun. Stereo matching by training a convolutional neural network to compare image patches. *Journal of Machine Learning Research*, 17(1-32):2, 2016.

[73] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[74] Li Zhang and Steven M Seitz. Estimating optimal parameters for mrf stereo from a single image pair. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(2):331–342, 2007.

[75] Rui Zhao, Wanli Ouyang, Hongsheng Li, and Xiaogang Wang. Saliency detection by multi-context deep learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1265–1274, 2015.

[76] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1529–1537, 2015.

Curriculum Vitae

Name:	Jiaxiao Wu
Post-Secondary Education and Degrees:	University of Western Ontario London, ON M.Sc. in computer science, 2016 - Present
	University of Toronto Toronto, ON B.Sc. in computer science, 2011 - 2015
Honours and Awards:	Western Graduate Research Scholarship University of Western Ontario, 2016 - 2017
	Scarborough College General In-Course Scholarships University of Toronto, 2011 - 2012
Related Work Experience:	Teaching Assistant University of Western Ontario, 2016 - 2018
	Research Assistant, Computer Vision Group University of Western Ontario, 2016 - 2017