Weakly-supervised Semantic Segmentation with Regularized Loss Hyperparameter Search

by

Zongliang Ji

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2021

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Weakly supervised segmentation significantly reduces user annotation effort. Recently, regularized loss was proposed for single object class segmentation under image-level weak supervision. Regularized loss consists of several components. Each component, if used in isolation, would lead to some trivial solution. However, a weighted combination of the loss components introduces a balance between the individual biases. The weight of each component in regularized loss is controlled by a hyperparameter. We propose an approach that searches for regularized loss hyperparameters. The main idea is to set the most important regularized loss component to a high weight while ensuring the other loss components are set to weights just sufficiently high to prevent the trivial solution favoured by the most important component. Our approach results in a significantly improved performance over prior work with fixed hyperparameters and improves the state of the art in salient and semantic image level supervised segmentation.

In addition to image level weak supervision, we propose a new approach for semantic segmentation with weak supervision using bounding box annotations. Our new approach to weak supervision from bounding boxes also makes use of hyperparameter search regularized loss. Previous work on weak supervision from bounding boxes constructs pseudo-ground truth by segmenting each box into the object and the background for each box independently from all the other boxes in the dataset. We argue that the collection of boxes for the same class naturally provides a dataset from which we can learn the appearance of that object class. Learning a good appearance model, in turn, leads to a better segmentation of each individual box. Thus for each class, we propose to train a segmentation CNN as from the dataset consisting of the bounding boxes for that class using our proposed single object approach. After we train these single-class CNNs, we apply them back to the training bounding boxes to obtain object/background segmentations and merge them to construct pseudo-ground truth. The obtained pseudo-ground truth is used for training a standard segmentation CNN. We improve the state of the art on Pascal VOC 2012 benchmark in bounding box weak supervision setting.

Acknowledgements

I would like to thank all the people who made this thesis possible. This thesis would not exist if Professor Olga Veksler didn't accept me as a research master student to Waterloo. Thank you Olga for all the help during the past two years. I enjoy working with you and I truly received a lot of precious guidance. I also would like to thank my thesis reader Prof. Yaoliang Yu and Prof. Jeff Orchard. Thank you for attending my presentation and provide feedback to my work. Thanks to Prof Yuri Boykov for his vision class to help me kick off this research project.

Back in 2013, when I was still a high school student in Qingdao, David Scott Lewis helped me registered the ML class by Andrew Ng. I stopped watching that class after the housing price prediction example since I had no experience in programming not to mention Matlab or Octave. Thanks to Prof Matthew Anderson, Prof John Rieffel and other CS and Math faculty back in Union. I got to do research, learn how to code and taking the deep learning specialization again by the end of my Junior year. Thanks to Prof. George Todd, who guided me to reprove the Universal Approximation Theorem to have the basic understanding of neural networks. I chose to do my research in either ML for healthcare, robotics, or Vision and here I am.

I truly appreciate those who helped me to during this journey. Hope everyone stays safe and hope I can travel back to my hometown in the near future.

Dedication

This is dedicated to my family.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Image segmentation is one of the fundamental problems in computer vision and image processing. It is the process of partitioning an image (color or grayscale) into multiple segments (subsets of pixels). An individual segment (subset of pixels) is usually given a discrete label, see Figure 1.1. In general purpose image segmentation, labels do not carry any special meaning. Each label is just a generic name for a group of pixels, such as '1', '2', etc.

The purpose of image segmentation is to summarize and change the representation of digital image into more meaningful representation that is easier to analyze [107]. Segmented image can be used to identify object locations and boundaries (lines, curves, etc) within the image. In more details, image segmentation assigns a label to pixels within an image so that pixels with the same label have certain characteristics in common. The output

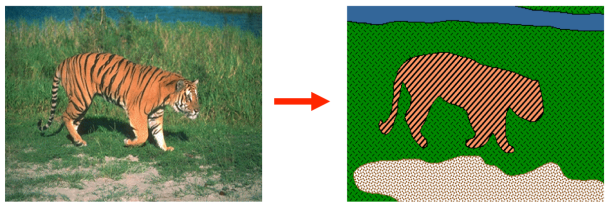

Figure 1.1: An example of image segmentation based on color [4]. There are 4 distinct segments, and, therefore, 4 discrete labels found in this image. The discrete labels are visualized with different color textures. These regions roughly correspond to the tiger, sand, grass and water.

of image segmentation is a set of segments (subset of pixels) that collectively cover the entire image. A pixel in a segment should have some similarity to other pixels in the same segment, based on appearance properties such as color, intensity, or texture. Adjacent segments should differ from each other in appearance, and the boundary between image segments usually has a strong contrast in either color or image intensity.

Image segmentation has applications in different disciplines from filming industry to the field of medicine. When filming scenes that are difficult to film or would be hazardous to film in real life, the software behind green screens uses image segmentation to crop out the foreground and place it on a different background [12]. In satellite imaging, image segmentation can be used to classify terrains and track objects within images, such as petroleum reserves [90]. The segmentation technique is also used for the identification of injured muscles [100], measurement of bone and tissue [68], detection of suspicious structures to assist radiologists in their diagnosis (Computer Aided Diagnosis, or CAD) [38], etc.

Unlike for human visual system, which seems to perform image segmentation without much difficulty, image segmentation is a challenging problem for machine vision. General image segmentation has been studied by researchers for decades and various methods have been developed. Among these methods, thresholding is the most straight-forward. It is based on using a threshold value to partition a grayscale image into a binary image [79]. There are also more sophisticated methods, such as clustering-based methods like Hierarchical Agglomerative Clustering (HAC) [29], K-Means [91], and Mean shift [18] methods, compression-based methods [72], histogram-based methods [77], region-growing methods [76], partial differential equation-based methods [19], and graph partitioning methods that utilize graph cut [14], MRF [39], or normalized cut [97].

Despite the multitude of methods developed for general image segmentation, discrete labels have no meaning attached to them and, therefore, are hard to interpret. This means that after we apply any of these general purpose algorithms to segment an image, each particular image segment is not endowed with any meaning. Further analysis must be applied to understand what each segment corresponds to. Furthermore, it is very hard to formulate the general image segmentation problem as a learning problem [118, 47]. This is unfortunate as supervised machine learning has truly moved the state of the art for most applications in computer vision. The main problem with using supervised machine learning for general purpose image segmentation is that the discrete labels assigned to image segments have no particular meaning. A segment labeled '1' in one image can correspond to a very different object than segment labeled '1' in another image.

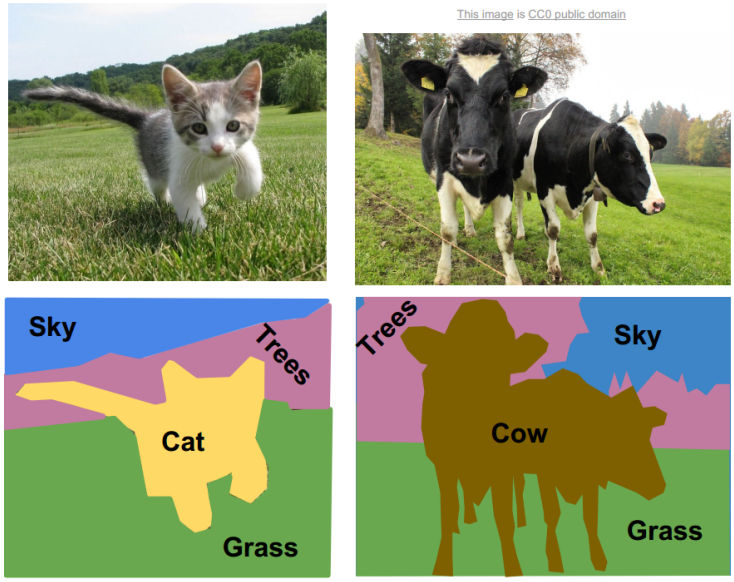

Figure 1.2: An example of Semantic Segmentation.

1.1 Semantic Segmentation

To address the issue of labels having no particular meaning in the general purpose segmentation, semantic segmentation has been proposed. Figure 1.2 shows an example of semantic segmentation, where each pixel in the image is assigned a predefined label (cat, dog, grass, cow, tree) with semantic meaning rather than segments without semantic labels in Figure 1.1. In other words, semantic segmentation attempts to allow machines to recognize and comprehend the content of an image at the pixel level. Since the predefined labels (objects) normally share similar characteristics (e.g. grass is typically green or yellow and sky is normally blue and white), we can formulate semantic segmentation as a learning problem. Given a predefined label set, one can "learn" the representation of each label across multiple images using the common features of the same label class.

Semantic segmentation has been shown useful in many applications like image editing (e.g. remove, replace background [74]), image interpolation (e.g. camera panning and zooming [70]), medical image analysis (e.g. analysis angiographic images [67], detecting mitotic cells [80]), autonomous driving (e.g. pedestrian identification, road line, road signs detection [15]), augmented and mixed reality (e.g. content creation [20]), and agriculture (e.g. precision farming robot [71] and aerial inspection of lands [7]).

The study of semantic segmentation dates back to 70s, when researchers were interested to separate or classify regions in a scene (image) with different semantic descriptions. In 1972, David Walts developed an algorithm to generate descriptions from drawings of scenes with shadows where he predefined different types of junction types of lines in the drawings [114]. Later, Feldman et al. developed a statistical method that could semantically analyze each region of specified images [37]. For the images of a car driving on a road, they defined six possible labels: sky, road, roadside vegetation, car, shadow of car and tree. For x-ray images of the left ventricular of heart, they defined the labels as heart interior, chest cavity background and the dark frame border. There were also researchers from the computational vision and pattern recognition field who utilized labeled satellite image dataset to classify different types of objects on the ground like rivers, lakes, bridges, lands, and islands [10], or to classify type of land into old growth, second growth, recent logging and water [103]. Ohta et al. proposed a hierarchical structure of region growing method to segment city scene into building, windows, sky, tree and road [78].

Most of early stage methods focus on utilizing the information within a single image and information across different images is not considered. Since the problem of semantic segmentation can be posed as a learning problem, researchers started to view it as a supervised learning classification problem in the field of machine learning. Given a set of images (a set of sets of pixels) with each pixel assigned a label from a set of predefined semantic classes, we want to find a method that can learn from these pixel-level labelled images in order to correctly classify each pixel in an unseen image into correct semantic class. To that end, various methods that combine traditional machine learning algorithms like random forest [16], supported vector machine (SVM) [105], conditional random field (CRF) [55] with basic computer vision image feature extractor like SIFT [66] were developed in the 2000s and early 2010s. Csurka et al. used a pipeline of probabilistic model with three components: a local appearance model, a local consistency model and a global consistency model [30]. Approaches like superpixels, SIFT [66], and MRF [39] were utilized in [69, 119]. Carreira et al. and Lempitsky et al. tackled semantic segmentation with pool-based method and MRF [39, 17, 59]. Arbeláez et al. developed a method that is based on region-based object detectors and scanning-windows model [9].

Even though applying traditional machine learning methods improved the performance of semantic segmentation, there was still a large amount of hand-crafted feature extraction involved in previous methods, and the accuracy of semantic segmentation had a significant room for improvement.

However, with a wider spread of Internet, cheaper price of electronic devices that can obtain and store data, the field of Artificial Intelligence and Big Data has been growing exponentially over the past few years. One of the subfields in artificial intelligence,

deep learning (DL) [40], has improved performance in various applications by orders of magnitude, due to advances in algorithms, computational power, and availability of large datasets. Using deep learning concepts, researchers developed a new generation of semantic segmentation methods that significantly improve results compared to the traditional methods. In addition to the giant leap in performance, the biggest difference between deep learning-based methods and previous methods is simplicity. Deep learning methods allow researchers to avoid the hand-design for feature extraction and selection phase which substantially reduces the research and development time. This thesis will focus on works that tackle semantic segmentation problem with deep learning.

1.1.1 Fully-supervised semantic segmentation

Deep learning addresses semantic segmentation problem in the same way as machine learning, but it employs artificial neural networks, such as convolutional neural networks (CNNs) [56], to summarize information from thousands of images at the same time without hand-engineering features. A key aspect of the success of deep learning methods on semantic segmentation problem is that artificial neural network has the universal approximation property [31] and CNNs can learn features and representations from thousands of real-world images at once within a reasonable time frame [53]. In order to train the neural networks, we provide them with a large number of images each with pixel-level labelled ground-truth [1]. During their training process, these neural networks uncover hidden representations of each label class. Therefore, when we offer them a new image that has never been seen before, the trained neural network models are supposed to identify which class each pixel in the new image belongs to. This specific setting, where the neural network is provided with images and their pixel-level labels, is known as fully-supervised semantic segmentation. In recent years, semantic segmentation with CNNs like FCN [65], U-Net [92], Mask R-CNN [42], and DeepLab [26] achieves excellent performance in fully supervised setting. We introduce some of relevant fully-supervised semantic segmentation works in more detail in Chapter 2.

1.1.2 Weakly-supervised semantic segmentation

Even though fully supervised semantic segmentation is highly successful, it is extremely labour-intensive to obtain image labels at the pixel level which makes the effort that goes into dataset preparation formidable. Therefore there is a significant research effort on

[1] We use label, annotation and ground-truth interchangeably in this thesis.

semantic segmentation in a weakly-supervised setting. In weakly-supervised semantic segmentation setting, pixel precise ground truth is not provided. Instead, only some information related to the object classes present in the image is provided.

There are various types of weak supervision. The least annotation demanding is the *image level* or *image tag* supervision. In this case, for each image, one provides only the object classes (i.e. *tags*) present in that image. Some examples of image tag semantic segmentation are [86, 82, 50, 5, 57]. The majority of these methods use Class Activation Maps (CAMs) [133, 95] to generate pseudo-ground truth which is used instead of the actual ground truth to train a CNN. CAMs usually provide only a sparse set of "seed" pixels. These need to be extended spatially, which is challenging. In the bounding box setting [82, 121, 32, 48, 61, 101, 54], annotation is in a form of bounding box with the corresponding class label placed around each object from the class of interest. Most of these methods rely on segmenting an object inside each box, using, for example [93] or [87]. The resulting segmentations are either used as pseudo-ground truth, or as part of a specially designed loss function to train a CNN. In "scribble" supervision setting [121, 63, 112, 108, 109], a small set of pixels are annotated in each image. Some of these approaches are similar to those based on pseudo-ground truth generation from a bounding box. Different from the rest, [108, 109] use regularized loss instead of pseudo-ground, and are the first methods to use regularized loss for weak supervision. However the approach in [108, 109] is designed for weak supervision with seeds, it is not directly applicable for image tag and bounding box weak supervision. Figure 1.3 shows an example of these three different types of weak supervision.

This thesis contains two contributions that improve weakly-supervised semantic segmentation. The first approach contributes to weak supervision with image-level tags. The second approach contributes to weak supervision from bounding boxes.

1.2 Contributions

We summarize our contributions in this thesis into two parts:

(1) Recently [111] proposed an approach for image level weak supervision based on regularized loss. We propose an improvement to [111]. Regularized loss encourages shorter object boundaries that are well aligned with the intensity edges in an image. The main idea in [111] is to train CNN to produce object segments with low regularized loss. However, in the absence of seeds, regularized loss cannot be used by itself, since the optimal solution would be an empty object (or empty background).

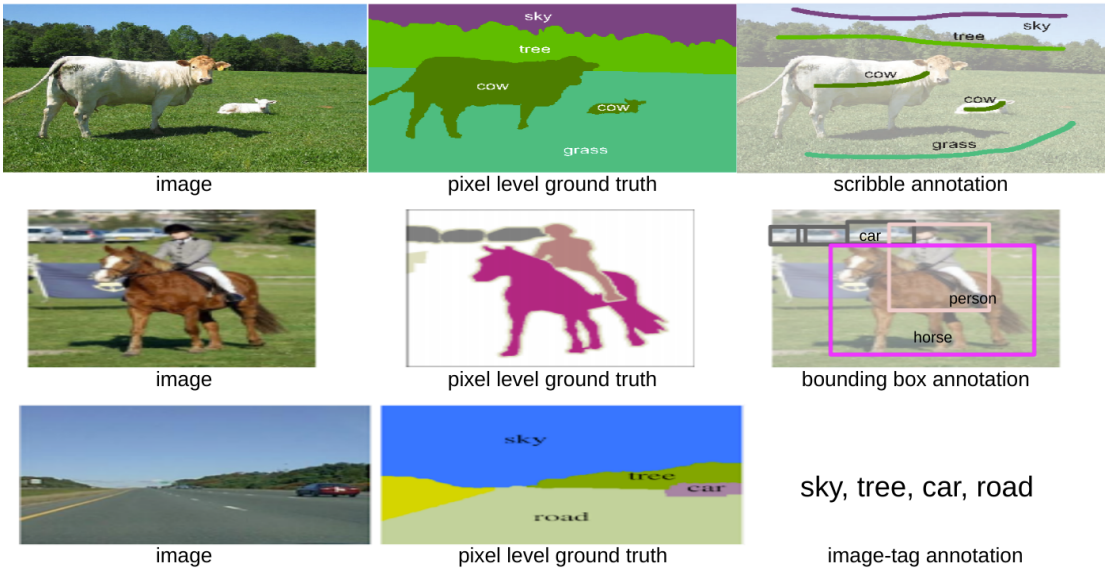

Figure 1.3: Different type of weak supervision for semantic segmentation.

In [111], they develop additional losses to be used with regularized loss. One of the main helper losses is the *volumetric loss* [113] that prohibits too small objects.

One drawback of [111] is that it uses fixed weights for the different regularized loss components. However, for each dataset, there is an optimal setting of hyperparameters that will, in general, differ from the optimal settings for other datasets. In weakly supervised setting, searching for an appropriate setting of regularized loss hyperparameters is far from trivial, as pixel precise ground truth is not available. Thus one cannot select hyperparameters through grid search that computes an accuracy metric using ground truth.

We propose a method for regularized loss hyperparameter search that does not rely on pixel precise ground truth. We observe that there are two important components of regularized loss in [111]: sparse-CRF and volumetric loss. In fact, we show that other regularized loss components can be omitted with only a minor decrease in performance. Therefore we focus on the weights of these two loss components. Sparse-CRF should have a higher weight compared to volumetric loss. However, volumetric loss weight should be sufficiently large to prevent collapse to the trivial solution favoured by sparse-CRF. We can compute this sufficiently large weight from the current solution, assuming it is not trivial.

In the first part, we propose an iterative algorithm for estimating the relative weight of the two loss components (i.e searching hyperparameters for regularized loss). We show that our approach significantly outperforms the fixed hyperparameters one [111], and improves state-of-the-art in weakly supervised salient and semantic segmentation.

(2) We propose a novel method for weakly-supervised semantic segmentation with bounding box annotations that achieves state-of-the-art result. This new method also utilizes regularized hyperparameter search we developed in our first contribution of the thesis.

Previous approaches to weak supervision with bounding boxes [82, 121, 32, 48, 61, 101, 54] consist of two stages. In the first stage, they construct pseudo-ground truth from bounding boxes. In the second stage, they train a segmentation CNN, with pseudo-ground truth instead of ground truth, using either a standard cross-entropy loss, or a loss that handles noisy pseudo-ground truth better [101, 54]. All prior methods treat each bounding box independently of the other boxes of the same class in the first stage.

Our approach is also based on constructing pseudo-ground truth from bounding box annotations. However, we observe that segmenting each bounding box separately is sub-optimal, since each bounding box has a rather limited information about the appearance of the object. Instead we should use the data from all boxes of the same class collectively, to construct a better appearance model for that class. The better appearance model, subsequently, can be used to segment each bounding box of that class more accurately, leading, in turn, to a more accurate pseudo-ground truth. In particular, for each object class, we propose to train a segmentation CNN using the bounding boxes from that class as training data. Note that this step transforms the collection of bounding boxes (for each class separately) into a dataset for weak supervision with image-tag annotations. Each such dataset contains only one object class of interest. We choose [111] for our image-tag weak supervision method since it was specifically designed for datasets that have a single object class of interest (our setting) and shows state of the art performance.

After training single-class CNNs, we apply them back to each bounding box in the training dataset to obtain pixel precise object masks. Then we develop and evaluate two strategies for constructing pseudo-ground truth from the obtained masks. Finally we train a standard semantic segmentation CNN on our pseudo-ground truth. In essence, we reduce bounding **box** supervision to image-**tag** supervision and apply the results **back** for the original bounding box supervision task.

We are the first to explore the information across all bounding boxes of the same class to learn a better object appearance model for that class. The advantage of our approach over prior work is its simplicity. We use standard CNN architecture, our loss function is intuitive and easy to interpret, there is a natural pipeline with no complex stages. Our approach sets a new state-of-the-art in weak segmentation with bounding boxes on Pascal VOC benchmark [35].

1.3 Thesis Organization

Chapter 2 introduces basic concepts in machine learning, deep learning and their applications in semantic segmentation and the related work in weakly-supervised semantic segmentation including the models this thesis is based on. Chapter 3 describes our method that improves regularized loss for weakly supervised single class segmentation. Chapter 4 describes our approach for bounding box weakly-supervised semantic segmentation that improves the state-of-the-art result. We call this method Box to Tag and Back (BTB). Chapter 5 summarizes and concludes our work in this thesis and proposes some possible future research directions.

Chapter 2

Related work

2.1 Deep Learning for Computer Vision

Artificial intelligence, machine learning, and deep learning have become the "buzzwords" in media for the past few years. In short, machine learning is a subfield of artificial intelligence, and deep learning is a subfield of machine learning. In this thesis, we are primarily deal with the semantic segmentation task as a machine learning and deep learning problem. Machine learning is a subject that lies at the intersection of computer science and statistics. It allows the machine to automatically learn from past experience (i.e. data) without programming explicitly. A fundamental goal of machine learning is to develop algorithms[1] that can utilize statistical analysis to predict an output based on the received input data while updating outputs as new input data becomes available.

There are three main types of machine learning: supervised learning, unsupervised learning, and reinforcement learning. Unsupervised learning deals with data that has no labels. For instance, general image segmentation can be considered as an unsupervised learning problem. Reinforcement learning studies how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward like robot navigation. We focus on supervised learning, which studies algorithms that are given labeled data as input.

There are two types of supervised learning problems: classification and regression. Regression is the task that allows the model to predict a continuous quantity like tomorrow's weather temperature or average housing price in certain areas for the next year. The

[1] We use machine learning model and machine learning algorithm interchangeably in this thesis.

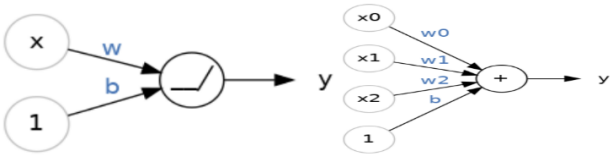

Figure 2.1: Example of two simple artificial neurons [1].

datasets provided for the regression problem are also labeled with a continuous number. Classification, on the other hand, is the task that allows the model to predict a discrete class label. Semantic segmentation is categorized as a classification problem as given an image, the model is going to assign (classify) each pixel within the image a predefined class label.

There are also different types of supervised learning: fully-supervised learning, weakly-supervised learning, and semi-supervised learning. For fully-supervised learning, each data point in the dataset is given a correct label (e.g. pixel-level labeled data provided for each image). In terms of weakly-supervised learning, each data point is given a label, but the label is noisy and not exactly correct (e.g. each image is provided with a box labeled ground-truth). For semi-supervised learning, some of the data points are provided with correctly labeled data and others with noisy labels. As discussed in the introduction, we are interested in the weakly-supervised semantic segmentation setting since obtaining pixel-level labeled data for each image is labor-intensive. In order to save human effort on the data labeling task and achieve acceptable semantic segmentation results, we focus on the single-class tag-labeled and multi-class box-labeled data in this thesis.

Deep learning is a subfield of machine learning that improves semantic segmentation performance by a large margin for the past few years. It is an approach characterized by deep stacks of computations. This depth of computation has enabled deep learning models to disentangle complex and hierarchical patterns found in the most challenging real-world datasets. Artificial neural networks (ANNs) have become the defining model of deep learning through their power and scalability. ANNs are engineered systems inspired by the biological brain but are not designed to be a realistic model of biological function [40]. They are composed of artificial neurons, where each neuron individually performs only a simple computation. However, the power of a neural network comes from the complexity of the connections these simple artificial neurons can form.

The fundamental component of a neural network is the individual neuron. Figure 2.1 shows two examples of a neuron (or unit). The neuron on the left has one input x, which

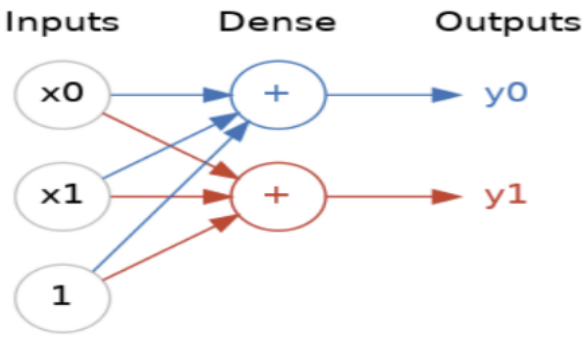

Figure 2.2: Example of a dense layer [1].

represents a linear unit $y = wx + b$. The input x connects to the neuron with a **weight** w. Whenever a value flows through a connection, we multiply the value by the connection's weight. For the input x, $w \cdot x$ reaches the neuron. *A neural network "learns" by modifying its weights.* The b here is a special kind of weight we call the **bias**. The bias does not have any input data associated with it; instead, we put a 1 in the diagram so that the value that reaches the neuron is just b. The bias enables the neuron to modify the output independently of its inputs.

The y is the value the neuron ultimately outputs. To get the output, the neuron sums up all the values it receives through its connections. The output of the neuron is also called **activation**, and this neuron's activation is $y = w \cdot x + b$. Though individual neurons will usually only function as part of a larger network, it is often helpful to start with a single neuron model as a baseline. Single neuron models are *linear* models. The neuron can also take multiple inputs with multiple weights at once. The neuron on the right of figure 2.1 is an example of neuron with multiple inputs and represents the formula $y = w_0x_0 + w_1x_1 + w_2x_2 + b$.

One of the advantages of the neural network is its modality, building up a complex network from simpler functional units. We can combine and modify single neurons to model more complex relationships. Neurons are typically organized into **layers** in a neural network. When we collect together linear units having a common set of inputs, we get a **dense** layer. Figure 2.2 shows an example of a dense layer with two linear units receiving two inputs and a bias. We can interpret each layer in a neural network as performing some kind of a relatively simple transformation. Through a deep stack of layers, a neural network can transform its inputs in more and more complex ways. In a well-trained neural

network, each layer is a transformation that propagates a little bit closer to a solution. Different types of layers can represent different types of transformations.

Each dense layer represents a linear transformation from a vector to another vector where the weight that connects the input and the neurons in the dense layer act as the transformation (weight) matrix. It is not difficult to observe that stacking two dense layers without anything in between still represents a linear transformation. Thus, with only dense layer, we cannot model transformation other than linear. In order to diversify the capability of neural network and introduce *non-linearity*, we need **activation functions**. An activation function is simply some function we apply to each of a layer's outputs (its activations).

The most common activation function is the *rectifier* function $\max(0, x)$. The rectifier function has a graph that is a line with the negative part "rectified" to zero. Applying the function to the outputs of a neuron will put a *bend* in the data, which introduces non-linearity. When we attach the rectifier to a linear unit, we get a **rectified linear unit** or **ReLU**. Applying a ReLU activation to a linear unit means the output becomes $\max(0, w * x + b)$ which is represented on the left of Figure 2.3. If we stack these non-linear units together, we obtain a "fully-connected" network, also known as multi-layer perceptron (MLP).

Since we are primarily interested in classification, we set the output layer as a rectifier function for binary classification. We can also use a linear unit to perform regression tasks that are able to predict some arbitrary numeric values. We use the fully-connected network on the right of Figure 2.3 as an example of a binary classifier. Binary classification is a common machine learning problem, like whether an email should go into a spam folder or not. We assign the two classes with labels 0 and 1, which can be potentially used by the neural network.

With a basic understanding of how basic neural network is constructed and its potential to model complex transformations, it brings us the question on how to allow neural networks to "learn" with labeled data. When first created, all of the network's weights are set randomly – the network doesn't "know" anything yet. As with all machine learning tasks, we begin with a set of training data. Each example in the training data consists of some features (the inputs) together with an expected target (the labeled output). Training the network means adjusting its weights in such a way that it can transform the features into the target. If we can train a network with this capability, its weights must represent in some way the relationship between those features and that target as expressed in the training data. In addition to the training data, we need two more components:

- A "loss function" that measures how good the network's predictions are.

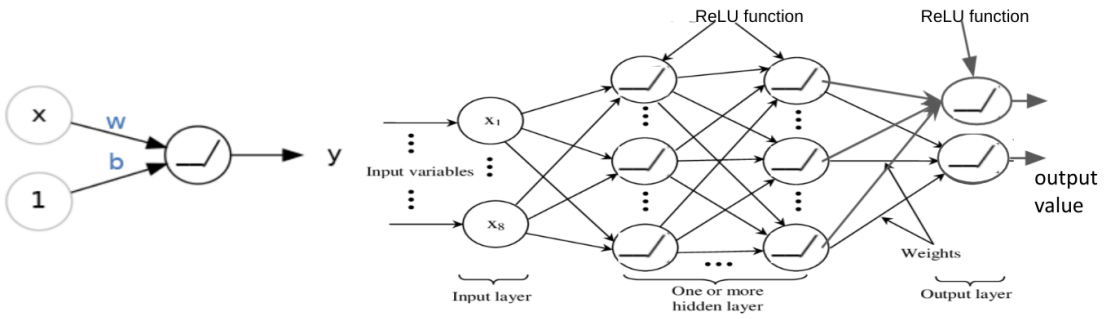

Figure 2.3: Example of ReLU and a fully-connected network [1].

- An "optimizer" that can tell the network how to change its weights.

The loss function measures the disparity between the the target (label)'s true value and the value the model predicts. Different problems call for different loss functions. For binary classification problem, where the task is to predict a class label either 0 or 1, we consider the output of the neural network as the probability of each class since neural networks are simply combinations of simple transformations and the output values are arbitrary. Instead of restricting the outputs to be a strictly discrete value, we allow the neural network to predict the probability of each class that each input data point could potentially belong to. At last, the class with highest predicted probability will be the final class prediction of the network. For the example network on the right of Figure 2.3, each of the final two neurons in the output layer can produce value from 0 to any finite positive number x. To convert these arbitrary output values into probability of each class that sums to 1, we use Softmax function

$$\sigma(o_i) = \frac{e^{o_i}}{\sum_{j=1}^{n} e^j},$$

where o_i is the prediction of single neuron in the final output layer represented by vector $[o_1, o_2, ..., o_n]$. Here, $n = 2$ for binary classification. After converting the network outputs into probabilities, a common loss function for classification problem is the cross-entropy (CE) function

$$-\sum_{c=1}^{M} y_{d,c} \log(p_{d,c}),$$

where M is the total number of classes, y is an binary indicator(0 or 1) if class label c is the correct classification for the data point d, and p is the predicted probability of data

point d for class c. For binary classification the CE function can be simplified to

$$-(y \log(p) + (1 - y) \log(1 - p))$$

where p is the predicted probability for the class labeled 1 and y is the true class label. From this simplified binary CE function, we know that the greater the predicted probability of the data point for the correct class, the lower the value CE function will output. This allows CE function to measure the disparity between probability distribution of model predicted results and the probability distribution represented by the discrete target labels. During training, the model will use the loss function as a guide for finding the correct values of its weights (lower loss is better). In other words, the loss function tells the network its objective[2].

After understanding the objective of the neural network, we would like to allow the network to gradually improve itself to meet its objective. The **optimizer** is an algorithm that adjusts the weights of neural network to minimize the loss. Virtually all of the optimization algorithms used in deep learning belong to a family called **stochastic gradient descent** (SGD). They are iterative algorithms that train a network in steps. One step of SGD-based algorithms contains:

- Sample some data points from training data.
- Use the neural network to make predictions from these data points.
- Measure the loss between the predictions and the true values using loss functions.
- Adjust the weights of the network so that the loss smaller based on gradient.

The algorithm halt when the loss becomes small enough or stops decreasing any further. Each iteration's sample of training data is called a **minibatch** (or often just "**batch**"), while a complete round of the training data is called an **epoch**. The number of epochs represents how many times the network will see each training example. Each epoch contains a number steps of the optimizer algorithm based on the size of minibatch (batchsize) and the total number of training data. Instead of halting the optimizer when loss stops decreasing, we can also manually choose the number of epochs to set an end point for the optimizer. The gradient is the gradient vector that represents a direction the weights need to go. More precisely, it indicates how to change the weights to make the loss change fastest. It is calculated by an algorithm called back-propagation [94]. The process is

[2]Loss function is also called objective function

called gradient descent because it uses the gradient to descend the loss function towards a minimum. The training is stochastic because the minibatches are random samples from the dataset.

With a desired supervised machine learning task, labeled training data, a neural network design, an objective function, and an optimizer, we are ready to solve a supervised learning problem with a deep learning approach. Previous paragraphs show a simple example for everything needed to solve a supervised classification problem with deep learning.

Deep learning for semantic segmentation that this thesis focuses on also includes these necessary parts, but the data are images with pixel-level, tag-level, or box-level ground truth, a more complex design of neural network and objective functions and a choice of optimizer. The goal of deep learning for semantic segmentation is to utilize the labeled data, loss function and optimizer to train a special design of neural network that is able to accurately classify each pixel within an unseen image from the test data. The special design of neural networks that revolutionized performance of image classification and later semantic segmentation is called convolutional neural network (CNN) [56].

2.1.1 Convolutional Neural Network

Convolutional neural network (CNN) [56] is a special design of artificial neural network architecture like the multi-layer perceptron. CNN was originally designed for image classification[3]. In 1989, LeCun et al. proposed a neural network based on CNN architecture to classify images of hand-written digits [56]. Later in 2012, Krizhevsky et al. [53] proposed another more complex CNN design called AlexNet that can classify images out of 1000 predefined classes from ImageNet [33]. AlexNet became the basis for modern CNN architecture design.

When first designed, CNN architectures make the explicit assumption that the inputs are images. This assumption allows the operations between layers to be more efficiently implemented and vastly reduces the number of parameters in the network. As introduced earlier, MLPs receive an input (a single vector) and transform it into an output vector after a series of hidden layers. Each hidden layer comprises several artificial neurons. These neurons have connections with every neuron in the previous layer and the next layer, but neurons within the same layer are not connected to each other. The last fully-connected layer is called the "output layer" and in classification settings, it represents the class scores.

[3]Given an image with one object inside, assign the class to the whole image from a predefined set of classes

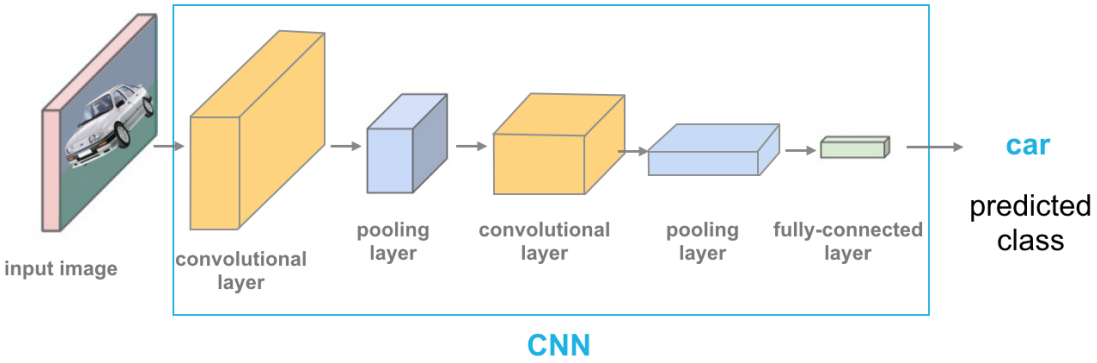

Figure 2.4: A general example of CNN architecture for image classification [3].

However, when dealing with images of larger sizes, fully-connected layers start to show the issue of scaling. Consider a small color image with size $32 \times 32 \times 3$ (32 wide, 32 high, 3 color channels) from the CIFAR-10 dataset [52]. A single fully-connected neuron in a first hidden layer of the MLP architecture would have $32 \cdot 32 \cdot 3 = 3072$ weights. This amount still seems manageable, but clearly this fully-connected structure does not scale to larger images. For example, if the input image has a bigger but still reasonable size, e.g. $300 \times 300 \times 3$, then this would lead to $300 \cdot 300 \cdot 3 = 270,000$ neuron weights. Moreover, from the design of MLP, each hidden layer requires many such neurons so that the parameters would add up quickly. Thus, full connectivity is wasteful, and the considerable number of parameters would quickly lead to overfitting[4]. The role of the CNNs is to transform images into a different form of representation regardless of image sizes and preserve the features that are crucial for the model to learn. This property is important when we want to design a neural network architecture that is expandable to massive datasets and learns important features simultaneously.

With this specific purpose in mind, Convolutional Neural Networks are specifically designed for image inputs, allowing CNNs to constrain the architecture more sensibly. In particular, unlike MLP, the layers of CNNs have neurons arranged in 3 dimensions: width, height, depth (channel). In addition to this 3-dimensional setup, each neuron in a layer only has connections with a partial region of the neurons in previous layer, instead of connecting to all neurons like the fully-connected layer. Moreover, for image classification, the final output layer has dimensions $1 \times 1 \times C$ where C is the number of classes the input image can belong to. At the last layer of CNN architecture, we will reduce the entire image

[4]Model performs well on train data but not on the test data.

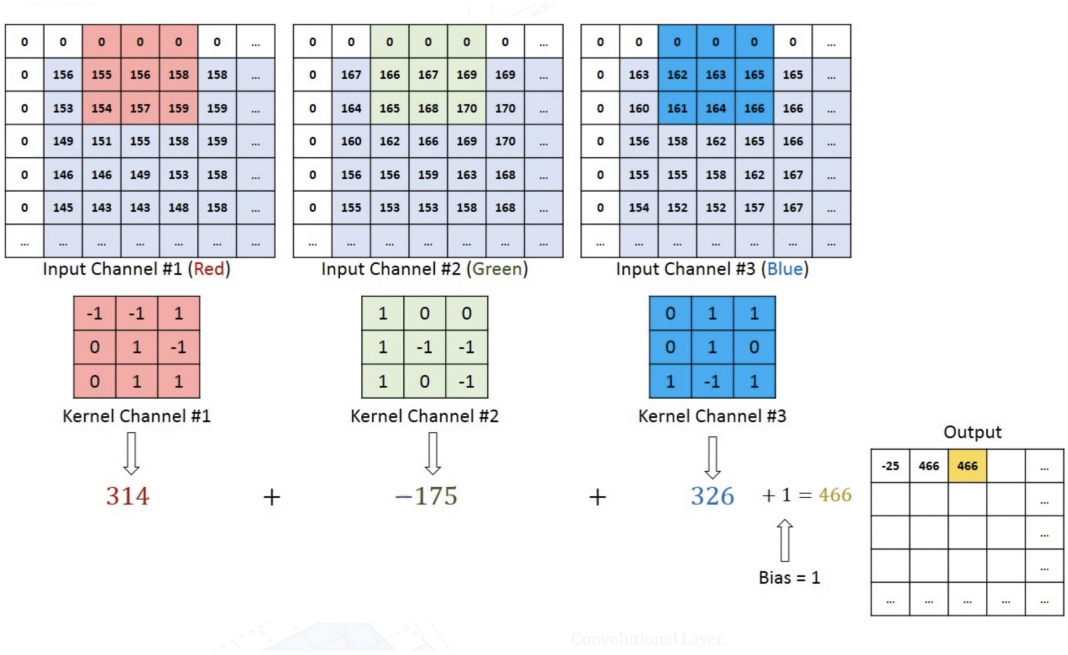

Figure 2.5: An example of convolution operation between an image with 3 channels and a $3 \times 3 \times 3$ filter [2].

into a single vector of class scores, arranged along the depth dimension. Figure 2.4 shows an example of CNN architecture for image classification. From the figure, we find that the bounding blocks of CNN architectures are convolutional layers and pooling layers.

Convolutional Layer

The convolutional layer is an essential part of a Convolutional Network, and it contains most of the computations required for CNN. A convolutional layer is based on convolution operation where the input 3D tensor will convolve with the parameters in the convolution layer to output a 3D tensor. These parameters consist of a set of learnable filters (kernels). Each filter is a 3-dimensional matrix (tensor) with a small spatial size (width and height) but the same depth size as the input volume. For example, a typical filter in the first layer of a CNN might have size $5 \times 5 \times 3$ (i.e. 5 pixels width and height, and 3 because images have depth 3, i.e. the color channels). When passing the input to the convolution layer, each filter is slid (more precisely, convolved) with the input volume through width

and height. We compute the dot products between the filter and the input volume in each spatial position during this sliding process. Each such dot product will produce a single value. As we slide along the width and height spatially, a 2-dimensional activation map that contains all these dot products will be formed.

Figure 2.5 shows an example of how a 2D activation map is generated after convolving a 3D tensor with a single $3 \times 3 \times 3$ filter. This particular filter (kernel) has $3 \cdot 3 \cdot 3 + 1 = 10$ weights that are trainable. There are different hyperparameters that we can manually adjust for the convolution operation in the convolutional layer. For instance, in Figure 2.5, each channel of image tensor is surrounded by 0's, this is called **zero-padding**. Since after convolution operation, the output 2D activation map may not be same size as the input tensor size, to adjust the width and height of the output volume, we pad zeros around the input tensor before convolving. Here, the number of rows and columns to pad is a hyperparameter as well as the **filter size**[5] (3 by 3 in this example). There is another hyperparameter called **stride** which determines how many pixels we move the filter each time when performing the convolution operation. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 then the filters jump 2 pixels at a time as we slide them around. This will produce smaller output volumes spatially. If the input tensor has width or height X, with filter size K, padding size P and stride size S, then the output width or height will become $(X - K + 2P)/S + 1$.

Intuitively, the final weights learned by the filters in each convolutional layer can recognize some visual features such as a particular orientation of edges or a representation of some specific colors, or eventually entire circle-like patterns in deeper layers of the network. Typically, we choose to use several unique but same-sized filters in each convolutional layer. These filters will each produce a unique 2-dimensional activation map. We then stack these activation maps along the depth dimension and produce the output volume. The number of filters used in each convolutional layer is also a hyperparameter called **output dimension** or **output channels**, as this number determines the depth dimension of the output volume. Unlike the fully connected layer, where the size of the input vector determines the number of weights, the number of weights in a convolutional layer is merely determined by the size of kernels and the number of kernels used. The design of convolutional layer vastly reduces the number of trainable weights, compared to the fully connected architecture.

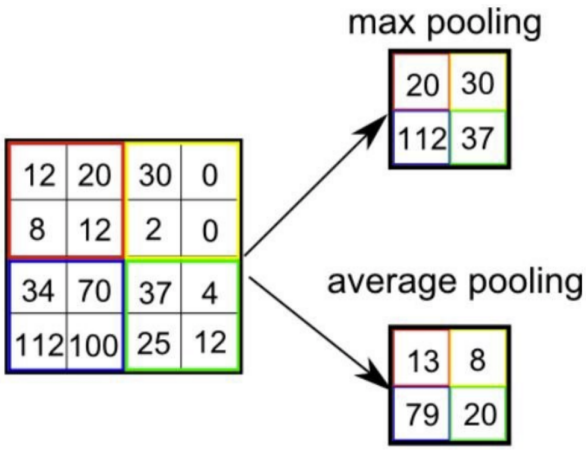

Figure 2.6: An example of two pooling operations [2].

Pooling Layer

After applying the convolution operation, the activation of convolutional layers is passed to the pooling layer. The pooling layer can further reduce the spatial size of activation maps from the convolutional layer. The existence of the pooling layer allows the CNN architecture to process data with less computation required via dimensionality reduction. Moreover, it is useful for extracting dominant features, which are rotationally and positionally invariant, thus maintaining the process of effectively training the model.

There are two types of pooling: average pooling and max pooling. The base operations of these two methods are indicated in their names. Given a filter with a small spatial size (i.e. 2×2), average pooling returns the average of values from within the portion of the input covered by the small filter. On the other hand, max pooling returns the maximum value from the portion of the input covered by the filter. Figure 2.6 is an example of these two different pooling operations. Max pooling is also known to act as a noise suppressant. By discarding all values but only the maximum value, it simultaneously performs denoising along with dimensionality reduction. However, average pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, max pooling normally performs a lot better than average pooling in practice.

[5]Filter size is also called kernel size or receptive field.

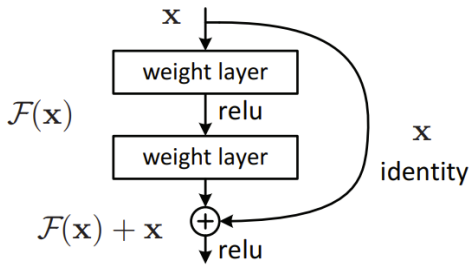

Figure 2.7: An overview of residual block.

Benefits of CNNs

CNNs can largely improve the performance on computer vision tasks compared to MLPs because they are designed for image inputs and can successfully summarize the spatial and temporal information in an image. Because of the reusability of network weights and the reduction in parameter usage comparing to MLP, the CNN architecture performs better on image datasets. In other words, the CNNs can be trained to understand images better and produce a more meaningful representation via convolution layers and pooling layers.

2.1.2 ResNet and ResNeXt

CNNs enrich the modeling ability of neural networks when stacking different convolution layers and improve the performance of models, especially in applications of computer vision. The most representative application of CNNs is image classification. In 1989, LeCun et al. proposed a CNN design called LeNet to recognize hand-written digits [56]. In 2012, Krizhevsky et al. won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [33] with a CNN design called AlexNet [53]. Since then, lots of new network architecture for improving image classification task has been developed [99, 106, 43, 120]. Among them, ResNet [43] is another groundbreaking work in the design of CNNs, which makes it possible to train up to hundreds or even thousands of layers and achieves compelling results. Here, we briefly introduce ResNet [43] and its improvement ResNeXt [120] that we use in this thesis.

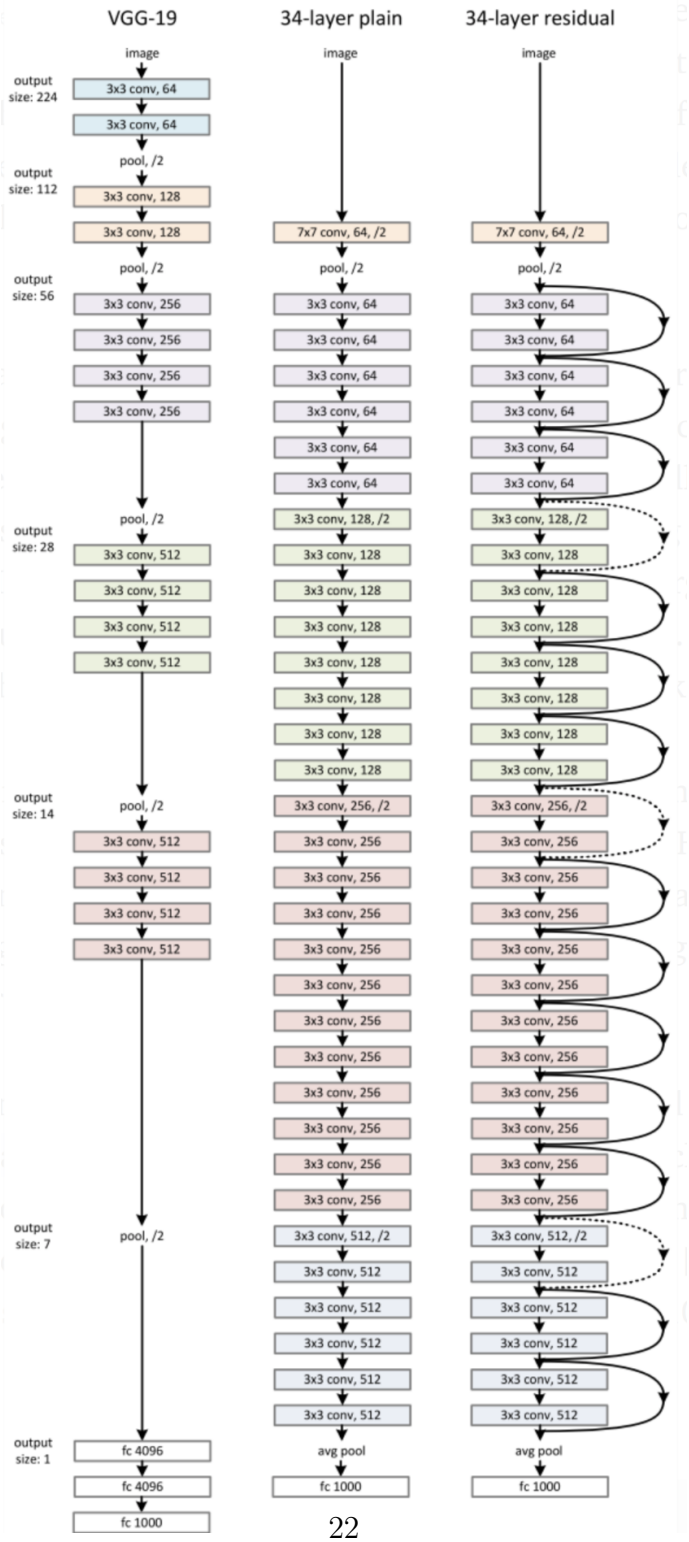

Figure 2.8: Left: VGG-19 network. Middle: Plain 34-layer CNN. Right: ResNet 34-layer design.

ResNet

After AlexNet [53] was developed, researchers started to extend the design of AlexNet with more convolutional layers incorporated in the network architecture. Instead of using 5 convolutional layers as AlexNet does, the VGG [99] network put 19 layers in their network design, and GoogleNet [106] put 22 layers.

However, stacking the number of layers in the design of CNNs does not necessarily increase the network's performance. The deep neural network has a notorious problem called vanishing gradient, where the gradients calculated from the later layers by the back-propagation algorithm become infinitely small after repetitions of multiplications. This problem eventually makes deep neural nets hard to train, and the performance of networks becomes saturated over time or even starts degrading rapidly.

To address the vanishing gradient issue, ResNet [43] proposed a novel idea called "identity shortcut connection" that connects the activation of a layer to a layer much deeper in the architecture by skipping several layers in between, as shown in Figure 2.7. The authors of ResNet [43] argue that stacking layers should not decrease the network performance because we could simply stack identity mappings (layers that model identity transformation) upon the current network, and the resulting architecture would perform the same. This indicates that the model with more layers should not produce a training loss higher than its shallower counterparts. The authors of ResNet [43] assume that asking the stacked layers with this identity short connection to learn this "residual" function is easier than asking them to learn the desired underlying transformation directly. The proposed residual block above explicitly allows it to do precisely that. Figure 2.8 shows an overview of the architectures of a 19-layer VGG network [99], a plain 34-layer CNN, and the 34-layer ResNet design. The dotted shortcuts increase dimensions (channels). With this new design of CNN architecture, ResNet won first place on the ILSVRC 2015 [33] image classification task. Because of its compelling results, ResNet quickly became one of the most popular architectures in various computer vision tasks.

ResNeXt

In 2016, Xie et al. proposed a variant of ResNet called ResNeXt [120]. The model name, ResNeXt, contains Next. It means the next dimension, on top of the ResNet. This next dimension is called the "cardinality" dimension. Figure 2.9 shows a comparison between a regular ResNet block and a ResNeXt block with similar model complexity. The building block of ResNeXt utilizes the idea of split-transform-merge from the Inception network [106] but with less effort on choosing hyperparameters. It aggregates a set of transformations

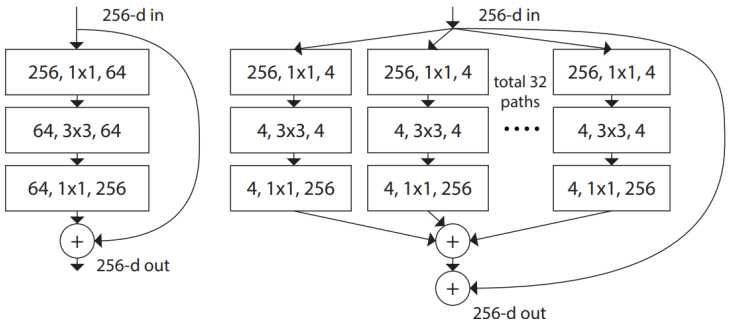

Figure 2.9: Left: A block of ResNet [43]. Right: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

with the same topology. Compared to a ResNet, it exposes a new dimension C, cardinality (the size of the set of transformations), as an essential factor in addition to the dimensions of depth and width. Same as ResNet, ResNeXt also has shortcuts, but it has several parallel stacking layer rather than sequential layers. Since the architecture was "split" into smaller parallel blocks, the trainable weights of each block are roughly the same as a ResNet block. ResNeXt can perform better than a typical ResNet block by taking advantage of the introduced cardinality dimension with similar model complexity.

2.2 Deep Learning for Semantic Segmentation

After achieving groundbreaking results in image classification, researchers started to use CNN architectures to approach another classical problem that can be thought of as a supervised learning problem: semantic segmentation. For semantic segmentation, the neural network still takes an image as input, but instead of outputting a vector with class scores, it has to output a tensor of class scores. The width and height dimensions represent the original image size. Recall CNN architecture for the image classification task. The last layer must output tensor of size $1 \times 1 \times C$, where C is the number of possible image classes. For neural networks designed for semantic segmentation, which deal with pixel-level classification, we would like the output layer to produce a tensor with dimension $W \times H \times C$, where W, H are the input image's width and height. This difference in the output size requires us to design architectures that can potentially "upsample" the abstract small feature volumes back to the input image size.

Since supervised learning utilizes the neural networks to "memorize" information from thousands of images, even though CNNs designed for image classification are not directly usable for semantic segmentation task, the features (weights) learned by convolutional layers before the output layer still capture useful visual information (like identifying colors and edges). Thus, instead of initializing weights randomly for the new NN architectures for semantic segmentation, it is customary to load the weights of CNNs trained on the ImageNet [33] challenge first. This idea of reusing weights from a previously trained network to start the training for a new task is called transfer learning [40]. With this idea, most of the architectures [65, 92, 42, 26] designed for semantic segmentation embed a basic CNN architecture [99, 43] designed for image classification as a feature extractor[6].

In this thesis, we choose U-Net [92] with ResNeXt [120] encoder for our single-object segmentation network and DeepLab [26] with ResNet [43] as the backbone for our bounding box weak supervision network.

2.2.1 UNet

The U-Net [92] was first developed by Ronneberger et al. for biomedical image segmentation. It was inspired by the fully convolutional network (FCN) [65]. There are two paths in the U-Net architecture. The first path works as the encoder called contraction path, which summarizes the features represented in the input image. The contraction path is similar to regular CNN architectures, which comprise convolutional and pooling layers. The second path acts as an upsampler or decoder called symmetric expanding path, which uses transposed convolutions [34] and allows the network to localize each part of the input image for precise pixel-level classification. Unlike FCN, U-Net has no dense layers but only convolutional layers, enabling U-Net to input images of any size. Figure 2.10 shows the overview of the original network design of U-Net. In order to output a tensor with the same width and height as the input image, the decoder path is necessary to increase the spatial size from the encoded stack of activation maps. Since the encoder path decreases the spatial size on the left-hand side of the figure and the decoder path increases the spatial size on the right-hand of the figure, the final network design visually resembles the letter U.

For the upsampling/decoder path, essentially, the reverse of the downsampling path is carried out. Each step in the decoder part of the U-Net has to fill extra pixels around

[6]This feature extractor is normally referred as the "encoder" or "backbone" of the semantic segmentation network.

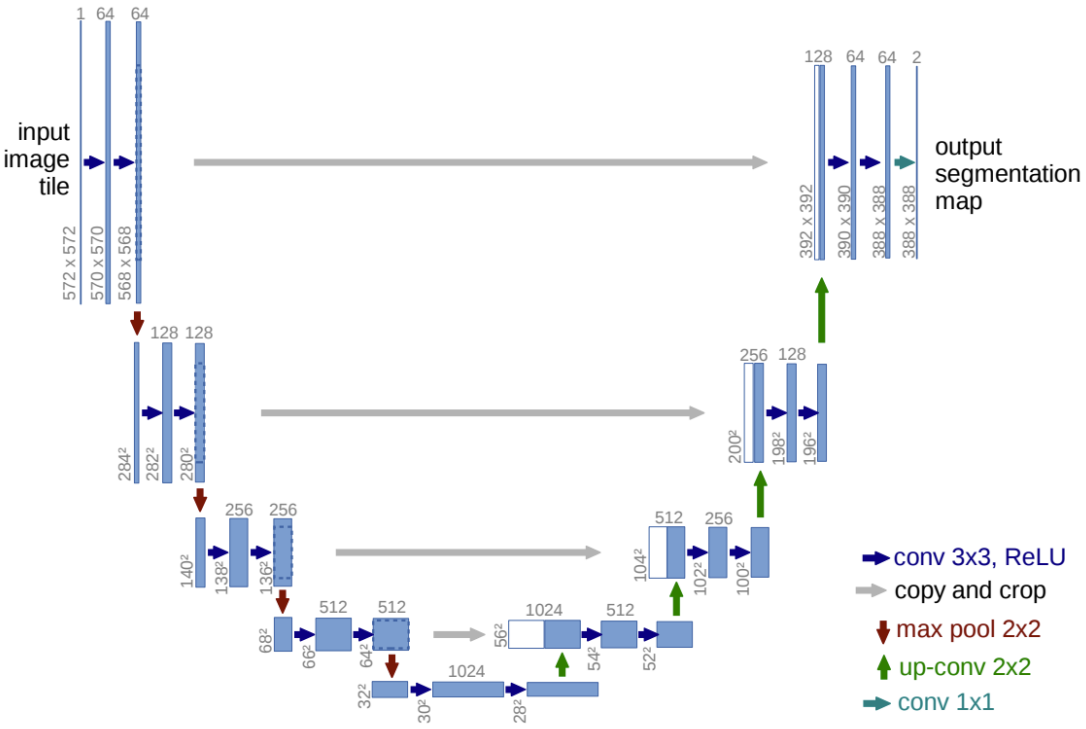

Figure 2.10: Overview of original U-Net architecture [92].

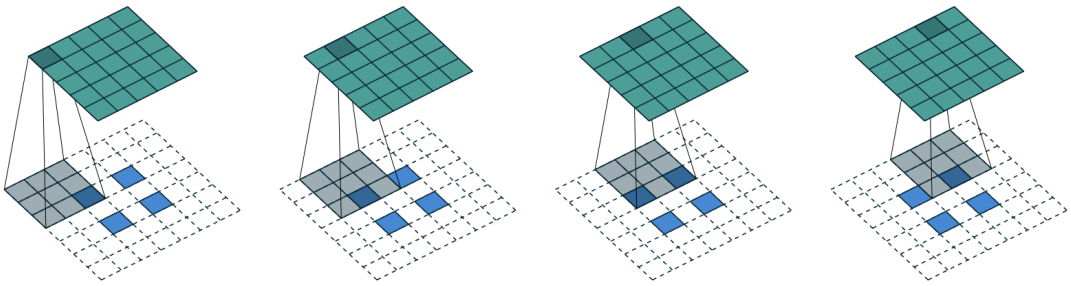

Figure 2.11: An example of transposed convolution which takes a 2×2 pixels as input and outputs a 5×5 feature map [34].

and in-between the existing pixels in the activation from previous layers. This transformation is the transposed convolution operation visualized in Figure 2.11, was introduced by Dumoulin et al. [34]. Here, zeros are added between the pixels. The blue pixels are the original 2×2 patches being expanded to 5×5 patches. Each pixel in the original map is padded with two pixels outside and another pixel between its neighbor. Here, the padded pixels are zeros (white). In practice, instead of zeros, we use some simple initialization of the new pixels like the weighted average of the input pixels (e.g. bilinear interpolation) to improve the performance of the transposed convolution.

Note that the U-Net architecture has cross-connections. The corresponding intermediate features from the encoder path are concatenated to upsampled features in the decoder path. At the time when U-Net was first proposed, ResNet-like [43, 120] structures were not available. However, ResNet and ResNeXt, by their design, are suitable encoders for U-Net architecture, and the combination yields better performance than the original network design. We choose ResNeXt as the encoder for U-Net in this thesis.

2.2.2 DeepLab Network

DeepLab network [22, 23, 25, 26] represent a family of CNNs designed for semantic segmentation, improved over time. The main features that make DeepLab network shine are atrous convolution and Atrous Spatial Pyramid Pooling (ASPP). Finally in the most recent versions [25, 26], DeepLabv3 and Deeplabv3+ incorporate the encoder-decoder design with all previously developed features.

Atrous convolution [34, 23], also called dilated convolution, introduces another param-

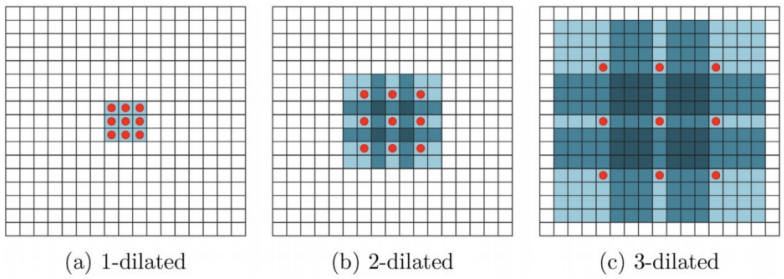

Figure 2.12: An example of atrous convolution of a 3×3 filter with different dilated rate.

eter to convolutional layers, the dilation rate. The dilated convolution of a signal $x(i)$ is defined as

$$y(i) = \sum_{k=1}^{K} x[i + rk]w[k],$$

where r is the dilation rate that defines a spacing between the weights of the filter w. For example, like in Figure 2.12 , a 3×3 filter with a dilation rate of 2 will have the same size receptive field as a 5×5 kernel while using only 9 parameters, thus enlarging the receptive field with no increase in computational cost. Atrous convolution was utilized in Deeplabv1 [22] .

To further improve the performance of the Deeplab architecture, the next challenge is existence of objects at multiple scales. In Deeplabv2 [23], they proposed Atrous Spatial Pyramid Pooling (ASPP). The idea is to apply multiple atrous convolution with different dilation rates to the input feature map, and fuse the results together. As objects of the same class can have different sizes in the image, ASPP helps to account for different object sizes which can improve the accuracy.

The former versions of Deeplab networks are able to encode multi-scale contextual information by probing the incoming features with filters or pooling operations (atrous convolution) at multiple rates and multiple effective fields-of-view (ASPP). In Deeplabv3 [25], the authors rethink the use of atrous convolution for semantic segmentation task. They incorporate the atrous convolution in to ResNet features extraction. They also improved the ASPP techniques by considering the image-level feature in addition to different features extracted by atrous convolution at different rate. Later, the authors of Deeplabv3+ want to address the challenge of capturing sharper object boundaries by gradually recovering the spatial information. Since general encoder-decoder networks have been successfully applied

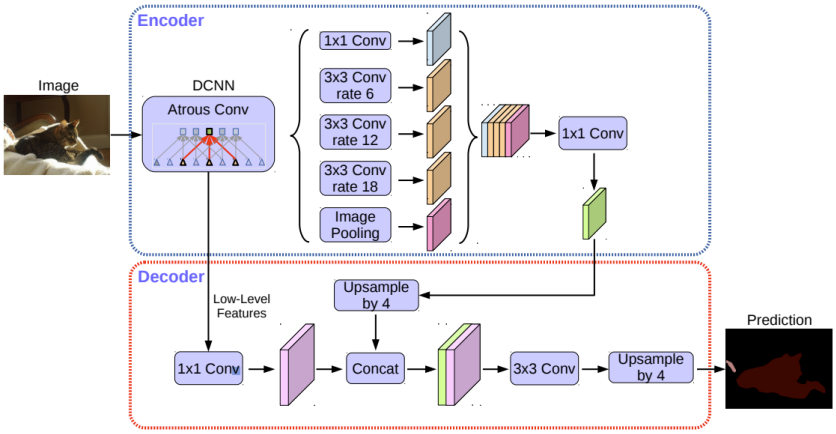

Figure 2.13: Overview of Deeplabv3+ network [26].

to many computer vision tasks, including object detection, human pose estimation, and also semantic segmentation, Deeplabv3+ [26] architecture combines the encoder-decoder design with the ASPP design to address the above issue. Instead of directly upsampling features output from the ASPP to the desired spatial dimension, they add a decoder that concatenates low-level features before ASPP and the features from ASPP then upsample to desired output dimension. The encoder-decoder model design of Deeplab network is able to obtain sharp object boundaries. The final network setup of Deeplab network is shown in Figure 2.13. We choose DeeplabV3+ with ResNet backbone as our bounding box weak supervision network.

2.3 Weakly-supervised Semantic Segmentation

Weakly-supervised semantic segmentation deals with non-pixel-level labeled ground truth but the neural network architecture is still similar to the fully-supervised setting. The loss function and the training of CNN is typically changed in weakly supervised setting to deal with the absence of pixel-level ground truth.

There are three common types of weak supervision: image-level tags [82, 84, 86, 8, 50, 44, 5, 57, 36, 21, 104, 36, 125, 124], image scribbles [121, 63, 112, 108, 109], and bounding boxes [82, 121, 32, 48, 61, 101, 54]. Here, we focus on related works for weakly-supervised salient segmentation and weakly-supervised bounding box segmentation since they are

Figure 2.14: Example of salient object segmentation.

most related to this thesis.

2.3.1 Weakly-supervised Salient Segmentation

Weakly-supervised salient segmentation is a special form of image-tag weak supervision like shown in Figure 2.14. It assumes that each input image only contains one object class, thus pixels in the input image are to be classified into two classes: foreground or background. Salient object is usually an object that draws the attention of the viewer [85, 123, 28]. More generally, salient object segmentation is a type of semantic segmentation where there is only one semantic class and the background. Single object class segmentation is especially useful for us later in the bounding box weak supervision because we can train a CNN on images that contain only one object class, for example on images containing only cats or images containing only airplanes.

There are three prior works for CNN based salient object segmentation without human annotation [127, 130, 75].

Zhang et al. proposed a method that generates proper supervisory signals (pseudo-ground truth) for training an existing deep salient object detector [127]. They summarized their method as "supervision by fusion." Since the methods developed for general image segmentation can perform salient segmentation without any learning and these methods are computationally efficient, they utilize several low-level (no learning involved methods) salient segmentation methods [128, 131, 98] to fuse a pseudo-ground truth for each image. Instead of directly using each generated pseudo-ground truth to train the deep salient object detector, they proposed a fusion process that considers the difference between training

images and the difference between low-level salient object detector predictions for each training image.

The fusion process that they introduced does not simply take the majority vote of segmented results or merge the results from each low-level salient object detector directly. For each training image, they generate a superpixel-level confidence score based on the difference of the pseudo-ground truth map generated by each low-level salient object detector. They also develop a superpixel-level fusion map based on the GLAD fusion models [117] to weight each low-level salient detector's prediction. Besides summarizing intra-image information, they also summarize inter-image information by applying GLAD fusion model to obtain image-level learning confidence and image-level fusion map from a set of training images. Finally, they use the superpixel-level fusion map and image-level fusion map to generate pseudo ground truth for each training image and use the superpixel-level confidence and image-level confidence as targets for their cross-entropy training loss. Their work shrinks the gap between unsupervised salient segmentation to supervised salient segmentation.

In [130], Zhang et al. proposed a new design of salient segmentation without human annotations. Their method is end-to-end and consists of two main parts: a "latent" saliency prediction module, and a noise modeling module. Given an input image I, they use several handcrafted feature-based low-level methods [128, 131, 98] to generate several saliency predictions. However, these predictions are noisy. The idea of this method is to use these noisy predictions as targets, train a deep CNN module as the "latent" saliency prediction module that generates an accurate prediction of salient segmentation, and a probabilistic model to create a noise. After adding the noise to the CNN salient prediction, it should match the noisy targets that those low-level salient detectors generate.

Let I be an input image, and ϕ_Θ be the saliency prediction module with weights Θ, and f_Σ be the noise modeling module with distribution Σ. For the input image I, $y = \phi_\Theta(I)$ is the CNN prediction of saliency map and $n = f_\Sigma(I)$ is the predicted noise based on the distribution. Fixing Σ, we can compute loss between $y+n$ and $y_{low-level}$ using cross-entropy to update the network weight Θ, where $y_{low-level}$ is the noisy predictions from the low-level salient detectors. Fixing Θ, we can calculate loss between $y_{low-level} - y$ and n to update the noise distribution. Through iteratively updating the network weights and the noise model, they are able to have a final deep salient object detector ϕ_Θ. Their design is more straightforward than the "Supervision by fusion" method [127] and achieves better results.

In [75], Nguyen et al. proposed another method that first generates pseudo-ground truth labels using various low-level salient detectors and then trains a final deep saliency detector using these labels. However, different from the previous two methods, they proposed to use

self-supervised learning to improve the quality of their pseudo-ground truth labels into high-quality pseudo labels. Given training images without labels, they choose n handcrafted low-level methods and generate a low-quality saliency map for each image with each method. Then within the labels generated by the same method, they first train an auxiliary CNN using the coarse low-quality saliency maps to generate consistent label outputs, which they refer to as *inter-images consistency*.

Then each generated label is further refined with their self-supervision technique in an iterative manner. This iterative manner includes training a CNN that generates consistent label outputs. During each epoch of training, for each image, they store the intermediate saliency map output. After few epochs, they can use these saved snapshots for each image to generate a historical moving average prediction (MVA-prediction) by weighted summing over the post-processed snapshots using dense-CRF[51]. After obtaining these MVA-predictions, they will replace the targets in the previous step and train the CNN again. They iterate through training and generating the MVA-predictions until the MVA-predictions become stable.

After generating stable MVA-predictions (refined labels) for the same set of training images but different low-level methods, they have n refined labels for each training image. Then they use these n sets of refined labels to train a final deep CNN salient object detector which they refer to as *inter-methods consistent* predictions.

All three methods are based on exploiting multiple saliency segmentation methods that are not based on machine learning like general image segmentation. These approaches are very specific to the saliency segmentation, whereas method proposed by [111] is generic and applies to any single object segmentation scenario. This method [111] also utilized regularized loss instead of normally used cross-entropy loss to train CNN architecture for salient segmentation. We will review it in detail in later section since this thesis makes improvement on [111] and utilizes this improvement to propose new method to tackle bounding box weak supervision problem.

2.3.2 Weakly-supervised Bounding Box Segmentation

As described in the introduction and shown in Figure 1.3, instead of having pixel-level labeled ground truth, bounding box weak supervision provides object bounding boxes and their corresponding semantic class. Each bounding box completely encloses the the object from the semantic class of interest. Thus we can think of the box as an approximate shape, or approximate ground truth for the object. However, most of the objects have shapes different than a rectangle. Thus, if we view bounding box as providing an approximate

labeling for the object it contains, then such ground truth has many incorrect or noisy labeled pixels.

Approaches to segmentation with bounding box annotations consist of two main stages. The first stage is to construct pseudo-ground truth. The second stage is to train a semantic segmentation CNN, such as [24] using pseudo ground truth instead of ground truth. When generating pseudo ground truth, a pixel which does not belong to any bounding box can be safely marked as the background. There are different approaches for dealing with pixels inside the boxes.

The simplest approach is to consider each pixel inside the bounding box as a positive example for the corresponding object class [82]. Conflicts, i.e. pixels that fall inside two bounding boxes, are usually resolved by assuming the smaller box is in front of the larger box. A variation which is more robust to mislabeling at the expense of density is to consider only a certain percentage of pixels centrally located in the box as positive examples, and to label the rest as a void class (i.e. to be ignored) [48]. While these approaches are the most simple, they are the least accurate, as they are based on the least accurate object/background segmentation inside each box.

To get better results, one needs a more accurate estimate of which pixels inside the box belong to the corresponding object class. A natural idea is to apply to a box an algorithm designed for object/background separation in a single image. In [82], they use denseCRF [51], in [48] they use GrabCut [93]. Another approach is to use MCG method [87] to generate image segment proposals and to chose the segment that has the largest overlap with the bounding box [48, 32]. Yet another idea is to use both GrabCut [93] and MCG [87] and to mark as the object only those pixels that both methods agree on being foreground, while marking disagreeing pixels as void [48, 61]. All of these approaches segment a box into object/background separately from the other boxes, unlike our approach. Our approach can get more accurate segmentations in each box because we learn the appearance of an object class across all boxes that contain it.

An additional improvement can be obtained by iterative refinement of the pseudo-ground truth [82, 32, 48, 61]. In particular, after training CNN on the initial pseudo-ground truth, the output of CNN after several iterations can be taken as the new pseudo-ground truth, and this process is iterated, sometimes also using post-processing with denseCRF [51] on the intermediate pseudo-ground truth. However, such iterative training is not guaranteed to improve accuracy as errors can be propagated from one iteration to the next.

Instead of cross entropy, one can design a loss function that is better suited for handling noisy pseudo-ground truth [101, 54]. In [101] they first use denseCRF [51], separately in each box, to segment the object from background. From these segmentations, they compute

the class specific *filling rate*, which is the object size relative to its box, averaged over the corresponding class. The loss function for training the final CNN uses a fraction of the most confident pixels in the box, and this fraction is equal to the class filling rate. The rest of the pixels in the box are ignored. Interestingly, the filling rate computation does use information across the different boxes of the same class, but in a rather limited way, i.e. for computing the average area of an object in a box. We use information across different boxes in a more general way, to model class appearance. In [101] they also propose to learn a class specific attention map using bounding boxes. The learned attention map is used to mask out features of the last layer in CNN for the corresponding class, which can help to filter out features from the background pixels. We could use the loss function from [101][7] (with the filling rate and attention mechanism) in the final stage of training segmentation CNN, possibly improving our perfromance. However, even with a simpler cross entropy loss, we outperform [101].

In [54], they generalize the loss from [101] and develop not only class specific, but also image specific filling rate loss and attention mechanism. The first stage is still based on segmentation masks obtained from each box using GrabCut [93], independently from the other boxes. In [54], they also introduce a feature embedding component in the loss function to learn the affinities between pixel pairs in order to encourage neighbors with similar affinities to be assigned to the same class. We could use the loss function from [54][8] to possibly improve our performance, but again, even with our simpler cross entropy training, we outperform [54].

2.3.3 Regularized Loss for Weak Supervision with Scribbles

As shown in Figure 1.3, scribble is another form of weak supervision. Instead of selecting a bounding box around the object, the human annotator draws lines (scribbles) within the objects of interest in the input image to obtain the annotation. Different from bounding box annotation which does not provide any definitive pixels that belong to the semantic classes (other than the background), the scribble annotation provides many pixels with definitive semantic classes. We can assume that every pixel that is labeled as an object class must belong to that object class.

Prior to Tang et al. [108, 109], approaches dealing with weakly supervised semantic segmentation all try to mimic fully supervised methods by first using the partial or noisy labels to generate labels that are as close as possible to pixel-level ground truth [63, 82]. These

[7]Note that [101] do not have a public implementation available yet.
[8]They plan to release the code but it is not publicly available yet.

approaches typically iterate between two steps: training CNN for semantic segmentation task and pseudo-ground truth generation step. Since pixel-level labels are not provided, general image segmentation methods like graph cut [14] or dense-CRF [51] are used in the pseudo-ground truth generation step. Then the generated pseudo-ground truth is used for training CNN. Next, they improve pseudo-ground truth and CNN performance iteratively.

However, these iterative approaches are time consuming and not necessarily guaranteed to generate high quality pseudo-ground truth. Tang et al. [108, 109] proposed methods that incorporate the general image segmentation losses like normalized cut [97] and dense-CRF [51] into the loss together with cross-entropy for training CNN for semantic segmentation. Thus, instead of iteratively improving the quality of pseudo-ground truth and the quality of CNN, they use scribble level labels directly as training targets and train CNN with a special loss that consists both of cross-entropy and a regularizer (e.g. normalized cut, dense-CRF) seen in general image segmentation. Their loss is named as "regularized" loss[9].

Dense-CRF [51] became popular in deep learning for semantic segmentation because the Deeplab [23, 26] authors used dense-CRF as a post-processing step to refine predictions from their CNN architecture and yield great results. As described earlier, neural network "learns" by updating its weights, and the weights are updated by SGD-like algorithm after comparing the network predicted results with ground truth targets via loss function like cross-entropy. One requirement for the loss function to be utilized by SGD-like algorithms is that the loss function must be differentiable. Thus, as long as the loss function is kept differentiable, we can add any terms to the original cross-entropy loss to help us guide the "learning process" of neural networks.

With this in mind, in [108], they use partial cross-entropy loss with normalized cut regularizer together as one loss for training the CNN and achieve better results than just using partial cross-entropy loss. Let $\boldsymbol{x}$ denote the output of CNN, where $\boldsymbol{x}$ has the same size as the input image. The output of CNN for pixel p is denoted by x_p. We know that cross-entropy loss for semantic segmentation can be represented as

$$-\frac{1}{|\mathcal{P}|}\sum_{p\in\mathcal{P}}\log(x_p^{y_p}),$$

where $\mathcal{P}$ is the set of all pixels in the input image and $x_p^{y_p} \in [0,1]$ is the scalar output for the ground truth class y_p of pixel p. Here, every pixel is taken into account for the final loss calculation. However, in weak supervision with scribbles, labels are available only for

[9]Here, the word "regularized" represents the extra regularizing components added into the normal cross-entropy loss to form a new total loss. It is not the regularization methods like batch normalization [45] or dropout[102] that used in the literature of deep learning to prevent model over-fitting.

the pixels that belong to scribbles. Thus, the partial cross-entropy loss in [108, 109] is represented as

$$-\frac{1}{|\mathcal{P}_l|}\sum_{p\in\mathcal{P}_l}\log(x_p^{y_p}),$$

where $\mathcal{P}_l$ is the set of pixels that has human labels (scribble pixels).

The joint regualarized loss that introduced in [108, 109] is denoted as

$$L_{pCE}(\boldsymbol{x}, y) + \lambda L_{reg}(\boldsymbol{x}),$$

where $L_{pCE}(\boldsymbol{x}, y)$ is the partial cross-entropy loss and L_{reg} can be any differentiable loss. On [108], they use normalized cut loss, and in [109], they experimented with dense-CRF loss. Above, λ is a hyperparamter that controls the weight of the regularizer.

In [109], they used this combined loss to train DeepLab model directly on scribble annotations and achieve the state-of-the-art result in weak supervision with scribbles. This is the first time where general image segmentation's losses are used directly as a part of loss function instead of in the post-processing step under the weakly supervised setting. This idea of using regularized loss to train neural network for weakly supervised tasks inspired this thesis.

2.3.4 Regularized Loss for Weak Supervision with Image Tags

Regularized loss has been used for weak supervision with scribbles [108, 109] and weak supervision with image-tags [111], but it has not been used for weak supervision with bounding boxes. Scribbles provide a definitive set of pixels that belong to an object class, and can be used with cross entropy in an objective function. With bounding boxes, there are no pixels that definitely belong to the object class and thus our method is substantially different from [108, 109]. Our method uses a modified version of [111] for image-tag weak supervision.

In this section we review the approach in [111]. It is primarily designed for a single object class segmentation, but is extended to multiple classes in a naive manner, by training on each class separately.

There are two classes: the object (class 1) and the background (class 0). Let $\boldsymbol{x}$ denote the output of CNN, where $\boldsymbol{x}$ has the same size as the input image. The output of CNN for pixel p is denoted by x_p. The last layer of CNN is softmax, so that $x_p \in (0, 1)$.

Sparse-CRF regularized loss is computed for $\boldsymbol{x}$ as

$$L_{crf}(\boldsymbol{x}) = \frac{1}{|\mathcal{P}|} \sum_{(p,q)\in\mathcal{N}} w_{pq} \cdot |x_p - x_q|, \tag{2.1}$$

where $\mathcal{P}$ is the set of all pixels in the image, and $\mathcal{N}$ is the set of neighboring pairs of pixels on a 4-connected grid. If CNN produces a sharp distribution, i.e. most outputs x_p are close either to 0 or to 1, then whenever two nearby pixels are not assigned to the same class, the loss of w_{pq} is incurred. In practice, CNN usually does produce a sharp distribution.

To align object boundaries to image edges, w_{pq} is set as:

$$w_{pq} = e^{-\frac{||C_p - C_q||^2}{2\sigma^2}}, \tag{2.2}$$

where σ controls the strength of the image color edges. Sparse-CRF regularized loss encourages shorter object boundaries that align to image edges [13].

Training with sparse-CRF loss alone, the lowest value is zero, which is incurred for a trivial solution: everything is classified as either the object or the background. Thus training with just regularized loss is infeasible, and "helper" losses are needed. The main helper loss is volumetric bias that penalizes empty solutions. Let $\bar{\boldsymbol{x}} = \frac{1}{|\mathcal{P}|}\sum_{p\in\mathcal{P}} x_p$, i.e. the normalized object size. Again, if CNN output $\boldsymbol{x}$ is sharp, this is a good approximation to the normalized object size.

There are two types of volumetric loss: batch and minimum volume losses. The batch volumetric loss L_b is defined for a batch of m images with outputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$:

$$L_b(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m) = \Big(\frac{1}{m}\sum_i \bar{\boldsymbol{x}}_i - 0.5\Big)^2 \tag{2.3}$$

This loss encourages the average size of objects in a batch to be half of the image size. Averaging object size over a batch makes the loss less strict: some batch objects can be significantly smaller than a half, some significantly larger.

The second volumetric loss is the minimum volume loss L_{min}. It acts on a single image output $\boldsymbol{x}$ and penalizes the result if the normalized object segment obtained for that image is less than obj_{min} of the image size.

$$L_{min}(\boldsymbol{x}) = \sum_i \big[\bar{\boldsymbol{x}}_i < obj_{min}\big] \cdot \big(\bar{\boldsymbol{x}}_i - obj_{min}\big)^2, \tag{2.4}$$

where $\big[\,\cdot\,\big]$ is 1 if the argument is true and 0 otherwise. In practice, we set $obj_{min} = 0.15$.

In the beginning of training, the regularized and volumetric loss are not sufficient, some cues are needed for possible spatial positions of the object/background. For a dataset with only one object class, the border pixels are likely to be the background, and the center pixel is likely to be the object. These priors are incorporated in positional loss. Let $\mathcal{B}$ be the set of pixels on the border of the image of width $w = 3$. Let $\mathcal{C}$ be the pixels in the center of the image in a box of size $c = 3$. The positional loss $L_p(\boldsymbol{x})$ is:

$$L_p(\boldsymbol{x}) = \Big(\frac{1}{|\mathcal{B}|}\sum_{p\in\mathcal{B}} x_p\Big)^2 + \Big(\frac{1}{|\mathcal{C}|}\sum_{p\in\mathcal{C}} x_p - 1\Big)^2. \tag{2.5}$$

The complete loss function for training is a weighted sum of regularized, both volumetric, and the positional loss:

$$L(\boldsymbol{x}) = \lambda_{crf} L_{crf}(\boldsymbol{x}) + \lambda_b L_b(\boldsymbol{x}) + \lambda_{min} L_{min}(\boldsymbol{x}) + \lambda_p L_p(\boldsymbol{x}). \tag{2.6}$$

If negative images (i.e. images containing only background pixels) are available, then they can also be included in the training with *negative* loss:

$$L_{neg}(\boldsymbol{x}) = \lambda_{neg}\frac{1}{|\mathcal{P}|}\sum_{p\in\mathcal{P}} x_p^2 \tag{2.7}$$

Training CNN with the loss function in Eq. (2.6) is difficult. In [111] they devise a two-stage strategy: annealing and normal stage. In the annealing stage, $\lambda_{crf} = \lambda_b = \lambda_p = 1$, $\lambda_{min} = 5$, and the value of parameter σ in Eq. (2.2) is increased from 0.05 to 0.15. Annealing σ helps CNN to find an easy object hypothesis first, and then refine it. Lower σ results in many color edges detected, and guides CNN to extract textured areas as an easy initial object hypothesis.

In the normal training stage, $\lambda_{crf} = 100$, $\sigma = 0.15$ and the rest of the parameters are kept the same as in the annealing stage. Most of the work is done by the regularized prior, given its high weight. This allows extraction of objects that significantly violate the positional and volumetric loss. To prevent object from collapsing to empty, $\lambda_{min} = 5$, higher than the parameters of the positional loss.

The annealing stage helps to find a reasonable initial hypothesis for the normal training stage. In [111], they find that the annealing stage can be replaced by weight transfer from another dataset. They use an easy-to-fit OxfordPet [110] dataset. It has the cat and dog object classes, which are combined into a single "pet" class since a single object dataset

is required. CNN is trained on OxfordPet dataset, first in the annealing, and then in the normal training stage. Then, for any new single object class dataset, instead of the annealing stage, the weights from CNN trained on OxfordPet are transferred, and training starts in the normal stage.

Chapter 3

Regularized Loss Hyperparameter Search for Weakly Supervised Single Class Segmentation

3.1 Introduction

Convolutional Neural Networks (CNNs) [92, 26] trained with pixel-level labeled images have largely improved the performance of semantic segmentation task. However, the effort to obtain such pixel-level labels is time-consuming. Collecting images with tag labels takes much less human effort in the data labeling process. Thus, we are interested in training CNNs without having pixel-level ground truth.

In this chapter, we are particularly interested in a special case of image-tag supervision called single class semantic segmentation or salient object segmentation, where each image in the dataset is assumed to have only one object class.

Image level weak supervision for salient object segmentation has also been explored [115, 127, 130, 61, 75]. Most of these methods use multiple weak saliency detection algorithms based on certain heuristics that do not rely on ground truth and then combine their results to train CNN.

Recently [111] proposed an approach for image level weak supervision without saliency heuristics. The main idea in [111] is to design a loss function that models common object shape properties and use it to train CNN without pixel precise annotations. They call this loss *regularized* as its main components are derived from CRF-based computer vision [113],

where regularizers impose coherence on noisy observations. The approach is designed for single object class segmentation, but is extended to multiple classes by training on each class separately.

Regularized loss in [111] consists of several components. Used in isolation, each component would prefer some trivial solution. The weighted combination of the components mitigates their individual biases and is suitable for training CNN without pixel precise ground truth.

The advantages of using regularized loss [111] for weak supervision is that we can use the same loss for different datasets and different applications without fine-tuning, as long as the training dataset contains a single object class. Another advantage is that there is no need to design a special purpose CNN architecture for each application. The method in [111] is conceptually simple, focusing on designing an effective loss function and its optimization.

One drawback of [111] is that it uses fixed weights for the different regularized loss components. However, for each dataset, there is an optimal setting of hyperparameters that will, in general, differ from the optimal settings for other datasets. In weakly supervised setting, searching for an appropriate setting of regularized loss hyperparameters is far from trivial, as pixel precise ground truth is not available. Thus one cannot select hyperparameters through grid search that computes an accuracy metric using ground truth.

In the rest of this chapter, we propose a method for regularized loss hyperparameter search that does not rely on pixel precise ground truth. We mainly focus on balancing the weights of sparse-CRF and volumetric loss, two important components of regularized loss in [111]. We show that our approach significantly outperforms the fixed hyperparameters one [111], and improves state-of-the-art in weakly supervised salient and semantic segmentation.

3.2 Regularized Loss Hyperparameter Search

In this section we describe our approach on the regularized loss hyperparameter search. In Sec. 3.2.1 we explain why we focus on hyperparameter search for sparse-CRF and minimum volume loss and explain how to set their weights relative to each other. In Sec. 3.2.2 we explain our iterative algorithm for hyperparameter search. In Sec. 3.2.3 we explore loss functions other than that in Eq. (2.6).

3.2.1 Relative Weights of Hyperparameters

As in [111], we start training in the normal stage with CNN weights initialized from CNN trained on OxfordPet dataset (Sec. 2.3.4). In [111] they use fixed weights for the different regularized loss components in the normal training stage. We wish to improve results by searching for a better hyperparameter setting. However, since there is no pixel precise ground truth, it is not obvious how to do it.

We observe that in Eq. (2.6), sparse-CRF is the main tool for discovering object segments. Indeed, the other losses tell us whether a segment has size close to a half of the image, or if it overlaps image center, or if it does not overlap image border. These tell us little about how likely a segment is to correspond to an object. Sparse-CRF, however, tells us whether a segment has image intensity edges on its boundary. This is informative, since object boundaries tend to cause intensity edges in the image.

Therefore, we would like to drastically increase the weight of sparse-CRF loss in Eq. (2.6), relative to other loss weights. However, if sparse-CRF weight is too large, the result would be a trivial empty solution. Note that we only have to handle the trivial (or 'empty') solution that sets every pixel to the background. We do not need to handle the other trivial (or 'full') solution that sets every pixel to the object. This is because the positional loss encourages the central pixel to be the object, and the border of the image to be the background. This means that producing a trivial 'empty' solution is cheaper than a trivial 'full' solution. To prevent an empty solution, we propose to increase the weight of the volumetric loss as well, but just by a sufficient amount, in order not to counteract the increase in the sparse-CRF weight.

There are two volumetric losses in Eq. (2.6): batch volume L_b and minimum volume L_m (L_{min}). Out of these two, L_b corresponds to a less realistic prior. It assumes that the average object size in a batch is half of the image size. This can be far from true for some datasets and thus increasing the weight of L_b is not promising [1]. Instead, we increase the weight of the minimum volume loss L_m. It gives a penalty to any individual image where the normalized object size is less than obj_{min}. Assuming that the object size is not exceedingly small, at least for most images, is more realistic.

Thus there are two hyperparameters in Eq. (2.6) that we focus on: sparse-CRF and minimum-volume loss weights. From now on we assume that the weights of other loss components are fixed to a low weight, namely of $\lambda_b = \lambda_p = 1$. In fact, we show in Sec. 3.2.3

[1] Experiments, which we omit here, confirm our intuition: increasing L_b in relation to sparse-CRF gives poor results, biased to equal split of object/background.

λ_m	100	200	300	400	500	600	700	800	900	10^3	10^4	estimated λ_m
ECSSD	0.	.591	.814	**.880**	.857	.864	.863	.864	.861	.852	.826	422
DUTS	0.	0.	0.	.743	**.784**	.748	.761	.769	.759	.762	.618	360
DUTO	0.	0.	0.	0.	.689	.784	**.785**	.753	.735	.738	.24	462

Table 3.1: Results in terms of F_β-measure on salient object datasets with fixed $\lambda_{crf} = 10^4$, and for different settings of λ_m. The last column shows λ_m estimated by our algorithm when $\lambda_{crf} = 10^4$.

that we can omit the other loss components without much decrease in performance.[2]

Assuming that our current solution is not empty, we can compute the weight of minimum volume loss that prevents the empty solution favoured by sparse-CRF loss. We need to make the weight of minimum volume loss large enough to make the average loss of initial solution smaller than the average loss of an empty solution. Thus switching to an empty solution from the initial one is too costly.

Given a loss function $L(x)$ and a solution $\mathbf{X} = (\boldsymbol{x_1}, ..., \boldsymbol{x_n})$ for the whole training dataset, let

$$L^{ave}(\mathbf{X}) = \frac{1}{n}\sum_{i=1}^{n} L(\boldsymbol{x}_i)$$

denote the average loss for all samples in $\mathbf{X}$.

Let $\mathbf{X}' = (\boldsymbol{x'_1}, ..., \boldsymbol{x'_n})$ be the solution (assumed non-empty) produced by the initial CNN for the whole training dataset. The average of regularized loss in Eq. (2.6) of the initial solution is $L^{ave}(\mathbf{X}')$.

Let $\mathbf{X}^0 = (\boldsymbol{x}^0, ..., \boldsymbol{x}^0)$ denote the trivial empty solution for the training dataset. Here $\boldsymbol{x}^0$ denotes the output of CNN which is all zeros, i.e. $x_p^0 = 0$ for all pixels p.

Now suppose we wish to use sparse-CRF weight λ_{crf}. We need to set $\hat{\lambda}_{min}$ to a value that makes collapse to an empty solution unlikely. To achieve this, we need to find large enough $\hat{\lambda}_{min}$ to ensure that

$$L^{ave}(\mathbf{X}') < L^{ave}(\mathbf{X}^0).$$

Since $\lambda_{crf} L^{ave}_{crf}(\mathbf{X}^0) + \lambda_m L^{ave}_m(\mathbf{X}^0) \leq L^{ave}(\mathbf{X}^0)$, it is enough to ensure that

$$L^{ave}(\mathbf{X}') < \lambda_{crf} L^{ave}_{crf}(\mathbf{X}^0) + \lambda_m L^{ave}_m(\mathbf{X}^0). \tag{3.1}$$

[2]Note that this is true only for the normal training stage. For the annealing stage (Sec. 2.3.4), the other loss components are still important, see [111].

	single λ_m	our backward algorithm			our forward algorithm		FixedReg [111]
	F_β	initial λ_m	final λ_m	F_β	final λ_m	F_β	F_β
ECSSD	.860	814	388	.856	422	**.884**	.835
DUTS	.768	765	355	.770	360	**.786**	.742
DUTO	.716	1035	447	.761	462	**.795**	.734

Table 3.2: Results in terms of F_β-measure of single λ_m algorithm, the backward, and the forward algorithms for hyperparameter estimation on salient object datasets. The last column shows the results of [111], i.e regularized loss with fixed hyperparameters.

Plugging $\boldsymbol{x}^0$ in Eqs.(2.1), (2.4) we obtain

$$L_{crf}^{ave}(\boldsymbol{x}^0) = 0, \quad L_m^{ave}(\boldsymbol{x}^0) = obj_{min}^2. \tag{3.2}$$

Plugging Eq. (3.2) into Eq. (3.1) and solving for $\hat{\lambda}_{min}$, we get our formula for setting $\hat{\lambda}_{min}$ in relation to any value of λ_{crf}

$$\hat{\lambda}_{min} > \frac{\lambda_{crf} L_{crf}^{ave}(\mathbf{X}') + L_p^{ave}(\mathbf{X}') + L_b^{ave}(\mathbf{X}')}{obj_{min}^2 - L_m^{ave}(\mathbf{X}')}. \tag{3.3}$$

We test experimentally the effect of the relative setting of λ_m and λ_{crf} on performance. We use saliency datasets ECSSD [122], DUTS [115] and DUTO [123]. For all experiments, we set $\lambda_{crf} = 10^4$ in Eq. (2.6), much higher than $\lambda_{crf} = 10^2$ used in [111]. We vary λ_m in regular intervals from 10^2 to 10^3, and also use one larger weight of 10^4. We resize images to 128×128 and train for 100 epochs. We divide each dataset in training and test folds. The results on the test fold are in Table 3.1, with performance metric F_β.

The first observation is that with such large λ_{crf}, to avoid a collapse to an empty object, λ_m has to be set to a much higher value than what is used in [111] for $\lambda_{crf} = 100$. Empty solution collapse results in F_β of zero. The peak performance for all datasets occurs at the smaller values of λ_m among those that do not suffer from empty solution collapse. This aligns with our intuition that λ_m should be set to a smaller, but sufficient value that prevents empty solution.

The peak performance also happens at different values of λ_m, so searching for a relative setting of hyperparameters is important. The last column in Table 3.1 shows the value of λ_m estimated by our algorithm (see Sec. 3.2.2) for $\lambda_{crf} = 10^4$. These values are larger than those leading to an empty solution collapse, and are in the lower range of λ_m values that do not lead to a collapse to an empty solution.

3.2.2 Iterative Hyperparameter Search

In previous section, we discussed our motivation for setting λ_{crf} to a high value. We can train CNN with the chosen λ_{crf} and with λ_m computed from Eq. (3.3). We call this *single* λ_m algorithm. However, while the initial CNN is a reasonable fit, it is not yet a good fit to the training data. Therefore for segmentations $\boldsymbol{x}'_i$ produced by the initial CNN, $L_{crf}(\boldsymbol{x}'_i)$ terms are likely to be high. In turn, the right hand side of Eq. (3.3) is likely to be high, and the computed value of λ_m high as well. This leads to an overestimated λ_m. As seen in Table 3.1, smaller λ_m that avoid empty solution collapse result in better performance.

A more accurate λ_m would result from a solution better than that produced by the initial CNN. As we keep training CNN, we get better (lower loss) solutions which can, in turn, be used to update λ_m using Eq. (3.3).

We explore two options for improving λ_m estimate. First option is to update λ_m after every few epochs of training. This leads to an iterative algorithm we call *backward* since the values of λ_m are decreasing. It is summarized in Alg. 1.

Algorithm 1: Backward algorithm for training with λ_m estimation

```
Initialize CNN from OxfordPet;
λ_crf = 10^4;
for i = 1 to 10 do
    (x'_1, ..., x'_n) is CNN output on training data ;
    use (x'_1, ..., x'_n) and Eq. (3.3) to compute λ_m;
    train CNN for 10 epochs with L
end
```

The second option is to start with a lower value of λ_{crf}, and iteratively increase it to the desired level. The values of λ_m are updated from the current solution whenever we raise the value of λ_{crf}. We increase λ_{crf} from 10^3 to 10^4 in 10 equal steps, trained for 10 epoch each. Alg. 2 summarizes this second approach, which we call *forward* because the values of λ_m are increasing (due to increasing λ_{crf}).

We compare single λ_m algorithm (for $\lambda_{crf} = 10^4$), the backward and forward algorithms in Table 3.2. The table shows the starting and final values of λ_m for the backward algorithm. The final values decrease significantly as the algorithm updates the current solution. For the single λ_m algorithm, the computed λ_m values are the same as in 'initial λ_m' column for the backward algorithm. For the forward algorithm, we show only the final λ_m as the initial estimates correspond to λ_{crf} smaller than the final value of interest.

Algorithm 2: Forward algorithm for training with λ_m estimation

Initialize CNN from OxfordPet;
for $\lambda_{crf} \in \{10^3, 2 \cdot 10^3, ..., 10^4\}$ **do**
    $\boldsymbol{x}'_i, ..., \boldsymbol{x}'_n$ is the output of CNN on training data;
    use $(\boldsymbol{x}'_1, ..., \boldsymbol{x}'_n)$ and Eq. (3.3) to compute λ_m ;
    train CNN for 10 epochs with L
end

Single λ_m algorithm is clearly inferior to both the forward and backward algorithms, due to an overestimated value of λ_m. The final estimates for λ_m are not that different between the backward and forward algorithms. The forward algorithm estimates of λ_m are slightly higher, and F_β scores are better. Even though both algorithms are based on the same principle, raising λ_{crf} values works better than setting a large λ_{crf} value from the start. The reason might be due to the particulars of gradient descent optimization. We chose the forward algorithm for all further experiments.

In the last column of Table 3.2 we also show the performance of the approach in [111] with fixed hyperparameters. Our results for both the backward and forward algorithm are significantly better. Fig. 3.3 illustrates some comparative results of the method in [111] and our approach on DUTO images. Our results are less noisy due to a much higher sparse-CRF weight while also λ_m is set appropriately to prevent collapse to an empty solution.

3.2.3 Other Loss Functions

Minimal Regularized Loss: In Sec. 3.2.1 we argued that sparse-CRF and minimum volume loss are the most important components of regularized loss in Eq. (2.6). We now experiment with a regularized loss where we only use these components, i.e. $\lambda_b = \lambda_p = 0$ in Eq. (2.6). The computation of λ_m is adjusted accordingly.

We use the forward version (Alg. 2) and all the other aspects are as in Sec. 3.2.2. The results, in terms of F_β are

	ECSSD	DUTS	DUTO
F_β	.874	.752	.754

Performance slightly decreases for all datasets, see Table 3.2, confirming our intuition that sparse-CRF and volumetric loss are the most important terms of the regularized loss function.

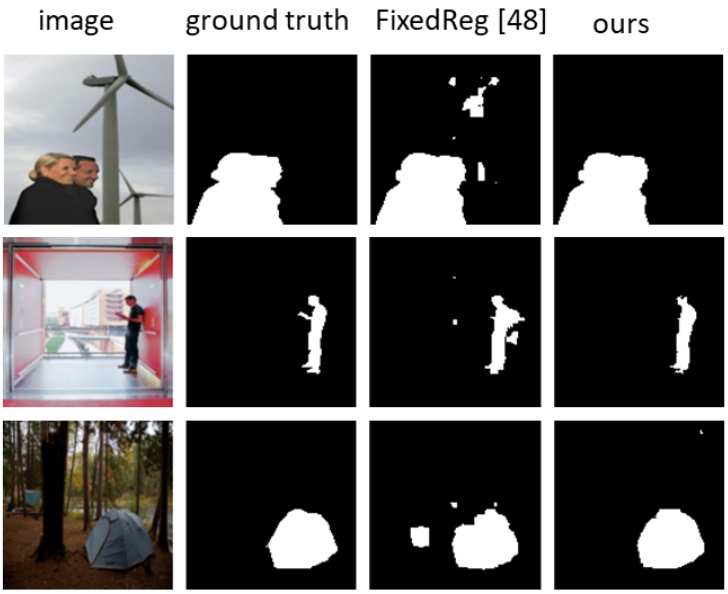

Figure 3.1: Comparison of fixed hyperparameter method [111] vs. our approach with hyperparameter search (with sparse-CRF). Our results are less 'noisy'.

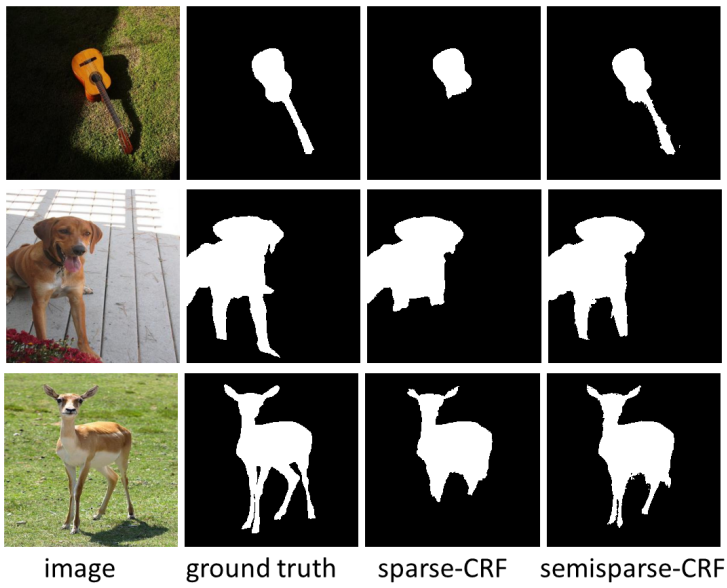

Figure 3.2: Semisparse-CRF captures thin structures better than sparse-CRF.

Semi-Dense CRF Sparse-CRF (Eq. (2.1)) is known for a short-cutting bias, which may result in missing long thin structures if they do not have sufficient intensity edge on the boundary. This is because a short low contrast boundary is cheaper than a long medium contrast boundary. Minimum volume loss may leave this bias uncorrected, if the area of an object without its thin parts is sufficiently large.

Dense-CRF [51], which connects every pair of image pixels, models thin parts better. But as in [111], we find that it gives results significantly worse than that of sparse-CRF.

We design a semisparse-CRF loss which has more edge connections than sparse-CRF but fewer connections than Dense-CRF. In particular, we let $\mathcal{N}$ in Eq. (2.1) consist of all pixel pairs that are at most distance r apart. In practice, we set $r = 5$. The w_{pq} weights are modified as [13]:

$$w_{pq} = \frac{1}{||p - q||} \cdot e^{-\frac{||C_p - C_q||^2}{2\sigma^2}}, \tag{3.4}$$

Fig. 3.2 shows a few examples of semisparse-CRF capturing thin structures better.

	MSRAB		ECSSD		DUTO		PascalS		THUR		SED2		SOD	
	F_β	MAE	F_β	MAE	F_β	MAE	F_β	MAE	F_β	MAE	F_β	MAE	F_β	MAE
SBF [127]	-	-	.787	.085	.583	.135	.680	.141	-	-	-	-	.676	.140
USD [130]	.877	.056	.878	.070	.716	.086	.842	.139	.732	.081	.838	.088	.798	.118
WSI [61]	.890	.067	.837	.110	.722	.101	.752	.152	-	-	-	-	.751	.185
DeepUSPS [75]	.903	.040	.874	.063	.736	.063			-	-	.845	.070	-	-
FixedReg [111]	.889	.048	.862	.075	.750	.077	.794	.110	.716	.090	.811	.100	.781	.146
ours sparse	.901	.043	.900	.061	.774	.069	.840	.091	.740	.074	.840	.083	.780	.140
ours semisparse	.904	.043	.890	.065	.786	.068	.824	.101	.741	.075	.852	.078	.800	.140

Table 3.3: MSRAB training dataset: image level weakly supervised saliency methods. Performance metrics are F_β (higher is better) and MAE (lower is better).

	MSRAB		ECSSD		DUTO		PascalS		SOD		DUTS	
	$maxF_\beta$	MAE	$maxF_\beta$	MAE	$maxF_\beta$	MAE	$maxF_\beta$	MAE	$maxF_\beta$	MAE	$maxF_\beta$	MAE
WSS [115]	.877	.076	.856	.104	.687	.118	.778	.141	.780	.170	-	-
MSW [126]	.890	.071	.878	.096	.718	.114	.790	.134	.799	.167	-	-
FixedRef [111]	.851	.058	.883	.055	.713	.084	.823	.086	.836	.112	.769	.065
ours sparse	.880	.055	.902	.057	.780	.071	.868	.072	824	.124	.835	.051
ours semisparse	.895	.045	.911	.051	.781	.069	.858	.080	.832	.127	.838	.050

Table 3.4: DUTS training dataset: image level weakly supervised saliency methods. Performance metrics are $maxF_\beta$ (higher is better) and MAE (lower is better).

3.3 Experimental Results

We use CNN architecture from [111][3], namely Unet [92] with ResNeXt [120] fixed features in the encoder. The features are pretrained on Imagenet [33]. Starting with CNN trained on OxfordPet [110], we first train on 128×128 images, and then on 256×256 images. Pretraining on 128×128 images gives slightly better results. Training is performed with Alg. 2, Adam optimizer [49], fixed learning rate 0.001, and batch size 16.

Results for salient object and semantic segmentation are in Sec. 3.3.1, Sec. 3.3.2, respectively. We also report ablation studies and comparison to fully supervised training in Sec. 3.3.1.

3.3.1 Salient Object Segmentation

We use the following datasets: DUTS [115], DUTO [123], ECSSD [122], MSRAB [64], THUR [28], SED2 [6], SOD [73], PascalS [62], and HKU-IS [60]. DUTS comes divided into the training and test folds.

[3] https://github.com/moduspordus/SingleClassRL

	ECSSD		DUTO		PascalS		HKUIS		THUR		DUTS	
	F_β	MAE	F_β	MAE	F_β	MAE	F_β	MAE	F_β	MAE	F_β	MAE
[129]	.880	.061	.750	.068	.813	.140	.870	.047	.837	.077	.777	.062
ours sparse	.900	.057	.776	.071	.865	.072	.902	.040	.748	.069	.835	.051
ours semisparse	.909	.051	.777	.069	.858	.080	.912	.037	.744	.070	.838	.050

Table 3.5: Result comparison to [129] who use scribbles, a stronger form of weak supervision than what we use. Performance metrics are F_β (higher is better) and MAE (lower is better). All methods are trained on training fold of DUTS dataset.

The most common metric for performance evaluation in saliency detection is F_β score, defined as,

$$F_\beta = \frac{(1+\beta^2) precision \times recall}{\beta^2 \times precision + recall},$$

with $\beta^2 = 0.3$. Another common metric is $maxF_\beta$, which is the maximum F_β across the binary maps of different threshold. The last metric is MAE [85], which is the average absolute per pixel difference between predicted saliency map and ground truth. In tables, red denotes the best and green second best results.

First we compare our work to previous image level weakly supervised saliency methods. We need to compare results consistently, as different methods use different datasets for training, and this affects the results. In [115, 126, 129] they train on DUTS. In [127] they train on MSRA10K [27]. In [130, 75] they train on 3000 images from MSRAB. In [61] they train on 2500 from MSRAB and 2500 images from HKUIS. To maximize the consistency, we first train our method (both with sparse-CRF and semisparse-CRF) on 3000 from MSRAB and compare it to [127, 130, 61, 75]. Then we train our method on DUTS, training fold, and compare it to [115, 126]. This is consistent except for [127] was trained on a much larger MSRA10K dataset and [61] which was trained on HKUIS in addition to MSRAB, making it a larger training dataset. We compare to the fixed hyperparameter method [111] for both MSRAB and DUTS training. For all prior work, we use their published performance metrics[4], or their code to generate results.

Comparison of our method to [127, 130, 61, 75, 111] is in Table 3.3. Our method with hyperparameter search is better than regularized loss with fixed hyperparameters [111] for all datasets. Our methods take the first or second place for all datasets, sometimes by a significant margin, especially for DUTO in terms of F_β measure.

Comparison to our method to [115, 126, 111] is in Table 3.4. Our method is better than fixed hyperparameter method [111] on all but one dataset, often by a significant

[4]Note that we took performance metrics for [115] from [126], since the later are better and are from the same authors.

	DUTS		ECSSD		DUTO		PascalS		SOD		HKUIS	
	$maxF_\beta$	MAE	$maxF_\beta$	MAE	F_β	MAE	$maxF_\beta$	MAE	$maxF_\beta$	MAE	$maxF_\beta$	MAE
BasNet [89]	.860	.047	.942	.037	.805	.056	.856	.076	.851	.114	.928	.032
GateNet [132]	.898	.035	.952	.041	.829	.061	.888	.070	-	-	.943	.035
Unet^2 [88]	.852	.054	.943	.041	.813	.060	.849	.086	.841	.124	.928	.037
ours sparse	.835	.051	.902	.057	.780	.071	.868	.072	.824	.124	.906	.040
ours semisparse	.838	.050	.911	.051	.781	.069	.858	.080	.832	.127	.916	.037

Table 3.6: Result comparison to fully supervised saliency methods in terms of $maxF_\beta$ measure (higher is better) and MAE (lower is better).

margin. Our method is better than [115, 126] across almost all datasets, in some cases by a large margin. Note that [126] uses additional sources of weak supervision, such as image captions, etc.

Next we compare to [129] in Table 3.5. In [129], they use scribbles, which is a much stronger form of weak supervision than what we use. Our results are significantly better in most cases. In Fig. 3.3 (third, forth columns) we provide a qualitative comparison of our method vs. [129]. We chose images where both methods do well overall. The gross form of salient object is well detected by both methods. However, our method extracts finer details better than [129]. This is likely because our regularized loss has a term L_{crf} that directly favours segments to align with intensity edges.

Lastly, we compare our performance to the fully-supervised methods for salient segmentation [89, 132, 88], in Table 3.6. Even though this comparison is unfair, as we do not use full ground truth, still, it is interesting to see how far behind our approach is. We train on DUTS dataset and compare to some most recent methods that are trained on the same dataset. Although our results are worse than the fully supervised methods, the gap is small. For PascalS dataset, our method is even the second best.

Ablation Studies

We test how the performance of our approach varies with different choices of the training protocol. First we test the performance of the multi-scale vs. single-scale approach, then the performance for different choices of λ_{crf} in Alg. 2 in the thesis, and finally the performance for different number of epochs inside the loop of Alg. 2 in the thesis.

For all experiments in this section, we use saliency datasets ECSSD [122], DUTS [115] and DUTO [123]. Each dataset is divided into training and validation folds: ECSSD has 700 and 300 images for training and testing, DUTO has 3,678 and 1,490 images for training and testing, and DUTS has 10,553 and 5,019 images for training and testing. Note that for ECSSD and DUTO datasets, the results reported in this section are slightly different

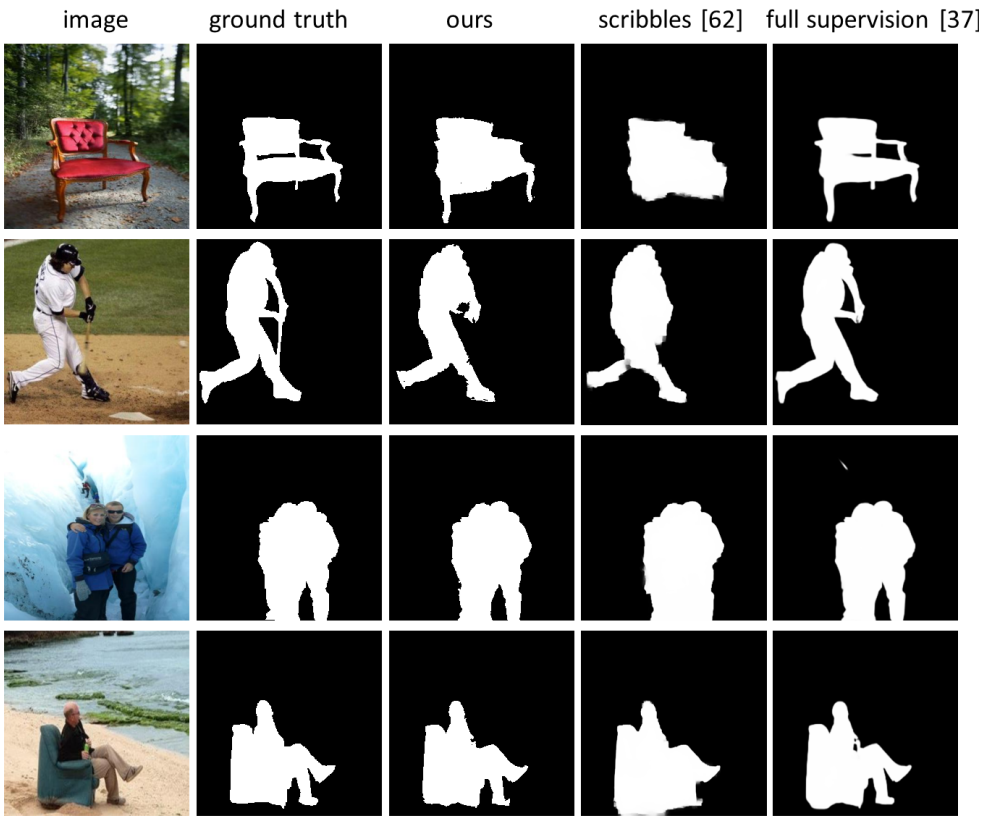

Figure 3.3: Qualitative comparison of our method (third column) to stronger forms of supervision, namely supervision with scribbles (forth column) and pixel precise supervision (last column).

	ours sparse		ours semisparse	
	1-stage	2-stage	1-stage	2-stage
ECSSD	.870	**.884**	**.900**	.881
DUTO	.778	**.800**	.754	**.772**
DUTS	.812	**.835**	.817	**.838**

Table 3.7: One-stage vs. two stage performance of our approach in terms of F_β measure.

from those reported in Section 4.2 of this thesis. This is because to be consistent with prior work we compare to, in Section 4.2 we train either on DUTS or MSRAB dataset, and test on the other datasets (including ECSSD and DUTO). In this section, we train and test on each of ECSSD, DUTO, and DUTS datasets separately.

One-stage vs. two-stage

For the experiments in the thesis, we train with two stage muti-scale approach. We start with CNN trained on OxfordPet [110], and first train on 128×128 images, and then transfer the weights and train on 256×256 images. We now compare the performance of this two stage approach to a one stage approach, where we train on 256×256 images starting with OxfordPet weights, without pre-training on 128×128 images first. The results of comparison are in Table 3.7, in terms of F_β score. Two stage training gives better results in all but one case. Training with a single stage is only slightly less accurate in most cases. Thus a single-stage approach can also be used with a slight loss of accuracy.

Value of λ_{crf}

We motivate using a large value of λ_{crf} in Algorithm 2. For all experiments in the thesis, we chose $\lambda_{crf} = 10^4$. We now test the performance of our approach for other large values of λ_{crf}, namely, 10^3 and 10^5. The results are in Table 3.8. Using a smaller value of $\lambda_{crf} = 10^3$ results in a slight decrease in performance in all cases, and using a larger value of $\lambda_{crf} = 10^5$ results in a slight decrease in performance in most cases. But the decrease in performance is small, showing that in general our method is stable across a large range of λ_{crf} values. The stability of our algorithm to a large range of λ_{crf} values is due to computing the weight of the minimum volume loss, λ_m, in relation to λ_{crf} to prevent a collapse to an empty solution.

Number of Epoch

We now compare the performance of our algorithm depending on the number of epoch used in Algorithm 2 of the thesis. The results of comparison are in Table 3.9. Performing 5 epochs for each setting of λ_{crf} is slightly worse in most cases, and performing 20 epochs

	ours sparse			ours semisparse		
λ_{crf}	10^3	10^4	10^5	10^3	10^4	10^5
ECSSD	.864	**.884**	.860	.856	.881	**.884**
DUTO	.776	**.800**	.782	.756	.772	**.780**
DUTS	.795	**.835**	.832	.809	**.838**	.827

Table 3.8: Performance of our method for different values of λ_{crf} in Algorithm 2 in terms of F_β measure.

	ours sparse			ours semisparse		
# epoch	5	10	20	5	10	20
ECSSD	.893	.884	**.900**	.892	.881	**.894**
DUTO	.777	.800	**.803**	.764	.772	**.779**
DUTS	.821	**.835**	.830	.833	**.838**	**.838**

Table 3.9: Performance, in terms of F_β measure, of our approach for different number of epochs in Algorithm 2 of the thesis.

is slightly better in most cases than using 10 epochs, the default setting in Algorithm 2. This is as expected, since longer training usually results in better performance.

Comparison to Fully Supervised Training

A natural question to ask is how well the network we use, Unet [92] with ResNeXt [120] fixed features, performs if the full ground truth is provided. This gives a bound on weakly supervised performance we can achieve in our approach, since we cannot outperform the same network trained with cross entropy on pixel precise ground truth.

We train the same CNN that we use for weak supervision, but now using pixel precise ground truth and cross entropy, training for 100 epoch on images of size 256×256. The results on test fold of each dataset are in Table 3.10. The gap between training with our method and with ground truth is negligible for ECSSD and MSRAB but could be improved for DUTS and DUTO. Thus there is still room to improve our approach without necessarily switching to a different CNN architecture. Thus, the direction for improving a regularized loss function still has promise, due to the performance gap.

Interestingly, the performance of the generic Unet architecture with cross-entropy loss is comparable to that of fully supervised methods on DUTS, see Table 3.6 in the thesis. This

	ECSSD	MSRAB	DUTO	DUTS
ground truth	.904	.916	.836	.867
ours sparse	.884	.901	.800	.835
ours semisparse	.881	.904	.772	.838

Table 3.10: Comparison of CNN trained with ground truth vs. CNN trained with our algorithm, in terms of F_β measure.

is surprising because the fully supervised methods in Table 3.6 have architecture and/or loss functions specifically designed for salient object detection.

Qualitative Results

In Figure 3.4 we show the performance of our hyper-parameter search method (semisparse-CRF), the method with fixed regularized loss parameters [111], and method that uses weak supervision with scribbles [129] on DUTO dataset, test fold. We sampled images where all methods perform reasonably well. Notice that our results are less noisy than that of [111, 129]. We capture object boundaries better than [129] despite using a weaker form of supervision.

Some interesting failure examples are in Figure 3.5. In the first two rows, our approach extracts areas corresponding to human figures, rather than signs marked as salient, showing that it learned to respond to people as salient. In the middle row, a human face is extracted rather than the whole figure covered by an unusual clothing. In the forth row, a salient object is extracted together with its reflection in the water. In the last row, the pool of water is not detected completely, likely because the water is often part of the background and not salient in the training dataset.

3.3.2 Semantic Segmentation

We now present our results for image-level weak semantic segmentation on Pascal VOC [35]. We follow the same approach as in [111]. They construct a single object class dataset[5] for each class separately by automatic web search on class keyword. The resulting dataset is called Web dataset. Then a single class CNN is trained, separately for each class on Web dataset. The resulting CNNs are used to construct pseudo-ground truth for Web images. These are used to train a standard multi-class segmentation CNN [24].

[5]Available at https://cs.uwaterloo.ca/oveksler/Data/PascalWebImages.zip.

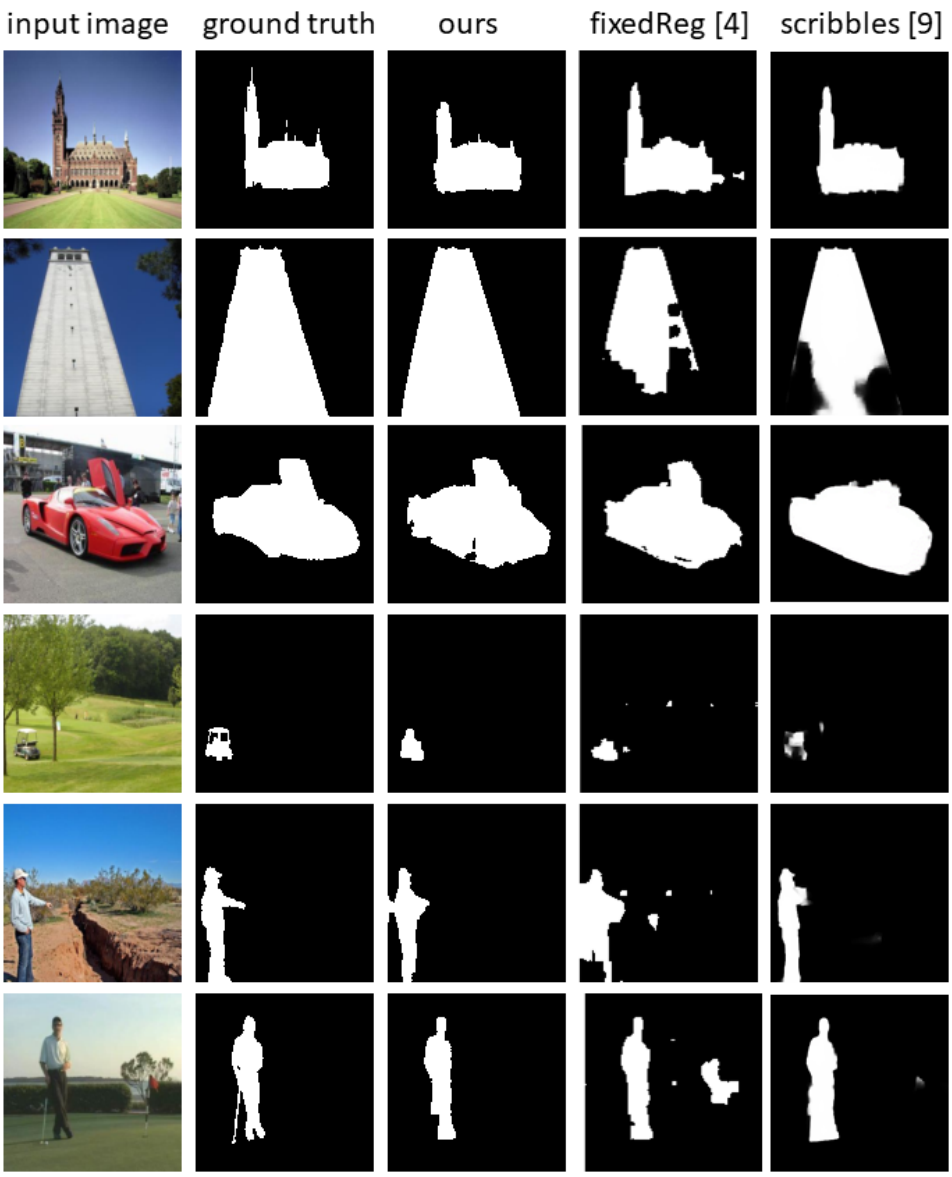

Figure 3.4: Some example results of our method vs. regularized loss with fixed hyperparameters [111] and salient object detection supervised with scribbles [129].

Figure 3.5: Some failure examples for our approach.

Method	Extra Data	mIoU
[5]	S	61.7
[8]	None	62.7
[96]	I	63.0
[125]	None	63.3
[57]	None	64.9
[21]	None	66.1
[104]	I	66.2
[124]	S	67.1
[111]	I	67.1
[36]	S	67.8
ours	I	68.4

Table 3.11: Comparison (mIoU metric) to other weakly supervised semantic segmentation methods on Pascal VOC 2012 val.

There are several variants in [111]. We use the one that gives the best results. Namely the single class CNNs trained on the Web images are applied to training Pascal images that have only one class present to construct the ground truth. This works slightly better since the Web images are not as close to validation Pascal images as training Pascal images.

Images are rescaled to 256×256 and we use our Alg. 2 and sparse-CRF loss for Web single-class dataset training. We use DeepLab-ResNet101 [24] pretrained on ImageNet [33] to train on the pseudo-ground truth. We train for 100 epochs. DeepLab is trained on images of size 513×513.

The results of our method, as well as comparison to recent prior work are in Table 3.11. The performance measure is mean Intersection over Union (mIoU). For each method we indicate if it uses extra data. If it uses saliency data with full ground truth, we mark it with 'S'. If it uses extra image data without full ground truth, we mark it with 'I'. All the better performing methods do use extra data. Our method falls into that category and outperforms all prior work.

We now evaluate our approach to image-level semantic segmentation on Pascal VOC 2012 segmentation challenge, test fold. Notice that our extra data (internet images) does not have pixel precise ground truth, it only has image labels. Methods that use extra salient object detection datasets do use pixel precise ground truth for saliency. We compare to the same methods as on the validation fold, Table 3.11. Some of these methods use CRF post-processing [51], and we use no post-processing. Our method outperform all others,

Method	Extra Data	+CRF	mIoU
[5]	S	✓	63.7
[8]	None	✓	64.3
[96]	I	✓	63.0
[125]	S	✓	63.3
[57]	None	✓	65.3
[21]	None	✓	65.9
[104]	I	✓	67.5
[124]	S		67.2
[111]	I		NA
[36]	S	✓	68.0
ours	I		69.2

Table 3.12: Comparison (mIoU metric) to other image-level weakly supervised semantic segmentation methods on Pascal VOC 2012 test fold.

the closest prior work [36] achieves $mIoU = 68.0$ while we achive $mIoU = 69.2$. We show our results vs. results in [36] on the examples chosen by [36]. We segment the shape of objects more accurately, in general. Our per-class IoU accuracies are in Table 3.13.

In Fig. 3.7 we show more qualitative results on Pascal VOC 2012 segmentation validation fold [35]. Even though the training data consists of images containing each object class separately, we can segment images containing multiple classes simultaneously. Some failure examples are in Fig 3.8. Sometimes, the segmentation is accurate but the object class is wrong (top row), sometimes, the two object classes are not separated from each other accurately (second, forth rows), reflection of an object is not segmented (middle row).

class	IoU
background	91.58
aeroplane	87.95
bicycle	33.62
bird	80.52
boat	54.54
bottle	64.1
bus	86.8
car	80.00
cat	88.24
chair	20.75
cow	86.02
diningtable	50.77
dog	84.74
horse	85.87
motorbike	81.37
person	71.90
pottedplant	56.10
sheep	85.28
sofa	38.94
train	76.88
tvmonitor	47.48
mean	69.21

Table 3.13: Per-class performance of our method on Pascal VOC 2012 test data.

Figure 3.6: Comparison to [36], on the examples chosen in [36].

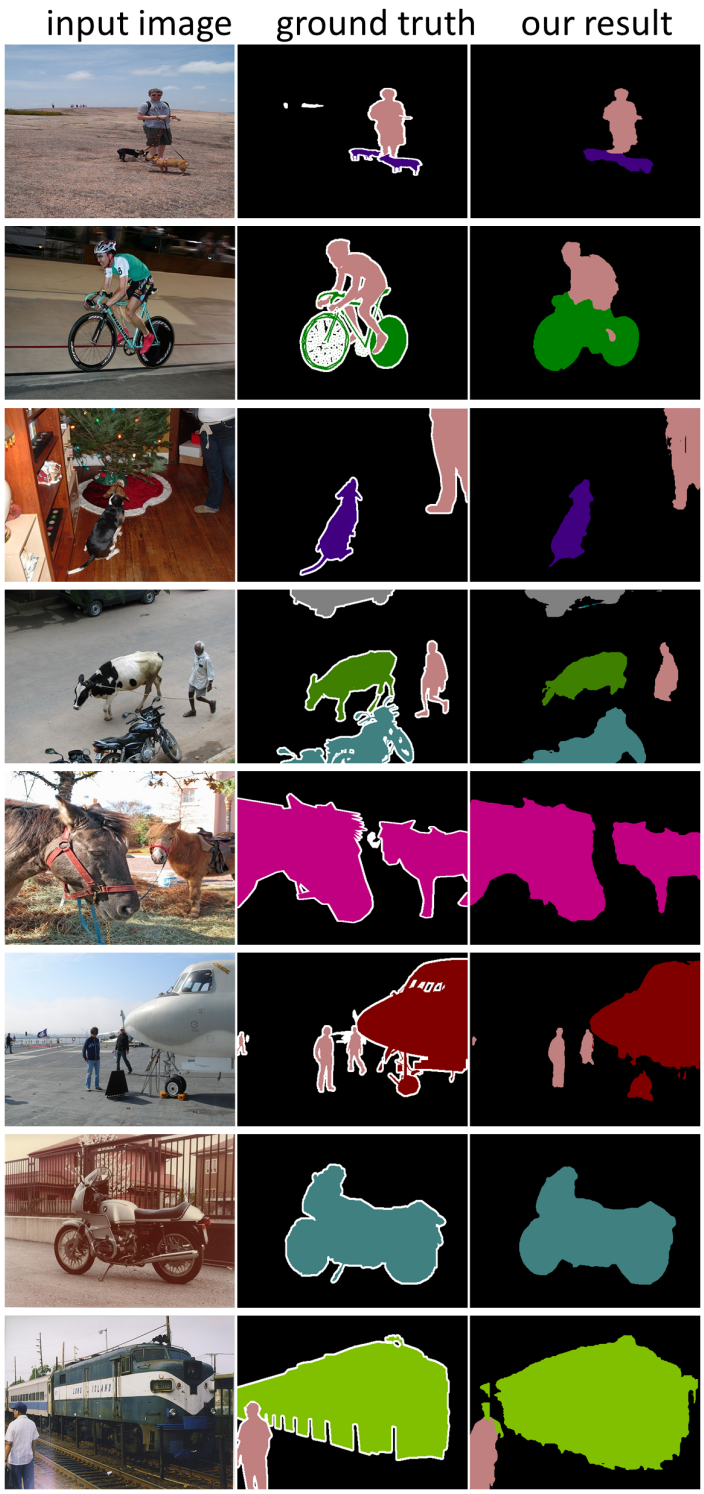

Figure 3.7: Some example results on Pascal VOC 2012 validation benchmark.

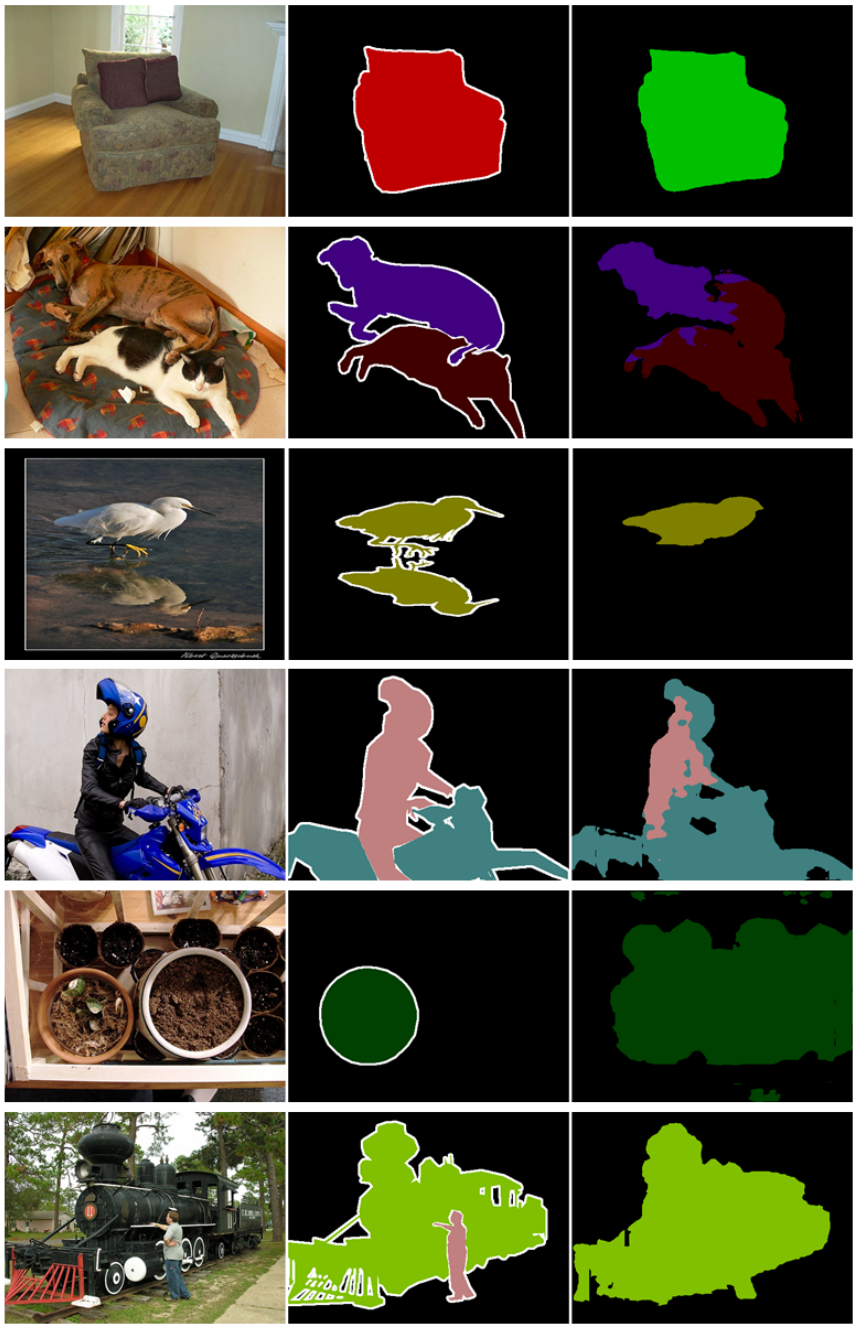

Figure 3.8: Some failure examples Pascal VOC 2012 validation benchmark.

Chapter 4

From Box to Tag and Back

In addition to weakly supervised single class segmentation, this thesis also looks into weak supervision with bounding boxes for semantic segmentation [82, 121, 32, 48, 61, 101, 54], where annotation is in a form of bounding box with the corresponding class label placed around each object from the class of interest.

Bounding boxes take approximately only 7 seconds per image to annotate [81], whereas pixel-precise annotations can take more than 4 minutes per image [11, 35]. Compared to other forms of weak supervision, such as image-level tags and scribbles, bounding boxes usually perform better [54].

Previous approaches to weak supervision with bounding boxes [82, 121, 32, 48, 61, 101, 54] consist of two stages:

1. Construct pseudo-ground truth.
2. Train a segmentation CNN with generated pseudo-ground truth.

When constructing pseudo-ground truth, dealing with pixels inside any bounding box differs previous work. A popular approach is to use an object/background segmentation algorithm such as GrabCut [93] on each box independently, and then paste the segmented foreground back to form pseudo-ground truth. At this second stage, all prior methods treat each bounding box independently of the other boxes of the same class.

Our approach is also based on constructing pseudo-ground truth from bounding box annotations. However, we observe that segmenting each bounding box separately is sub-optimal, since each bounding box has a rather limited information about the appearance

of the object. Instead we should use the data from all boxes of the same class collectively, to construct a better appearance model for that class. The better appearance model, subsequently, can be used to segment each bounding box of that class more accurately, leading, in turn, to a more accurate pseudo-ground truth. In particular, for each object class, we propose to train a segmentation CNN using the bounding boxes from that class as training data. Note that this step transforms the collection of bounding boxes (for each class separately) into a dataset for weak supervision with image-tag annotations. Each such dataset contains only one object class of interest.

For this step, we choose to use the state-of-the-art solution, which is an improvement to [111], that we proposed in Chapter 3. The regularized loss hyperparameter search approach from Chapter 3 was specifically designed for datasets that have a single object class of interest (our setting). The method in Chapter 3 is particularly simple, it uses generic CNN architecture and its main component is designing the loss function suitable for image-tag weak supervision. Furthermore, the loss function can be easily redesigned to be more suitable for single-class datasets arising from bounding box annotations.

After training single-class CNNs, we apply them back to each bounding box in the training dataset to obtain pixel precise object masks. Then we develop and evaluate two strategies for constructing pseudo-ground truth from the obtained masks. Finally we train a standard semantic segmentation CNN on our pseudo-ground truth. In essence, we reduce bounding box supervision to image-tag supervision and apply the results back for the original bounding box supervision task. The overview of our approach is in Fig. 4.1.

We are the first to explore the information across all bounding boxes of the same class to learn a better object appearance model for that class. The advantage of our approach over prior work is its simplicity. We use standard CNN architecture, our loss function is intuitive and easy to interpret, there is a natural pipeline with no complex stages. Our approach sets a new state-of-the art in weak segmentation with bounding boxes on Pascal VOC benchmark [35].

This chapter is organized as follows. We first describe single object class dataset construction in Section 4.1.1. Section 4.1.2 shows our approach for weakly supervised single class CNN training. Section 4.1.3 describes the pseudo-ground truth construction. Section 4.2 describes the final step of training a standard segmentation CNN with our pseudo-ground truth and the experiment results.

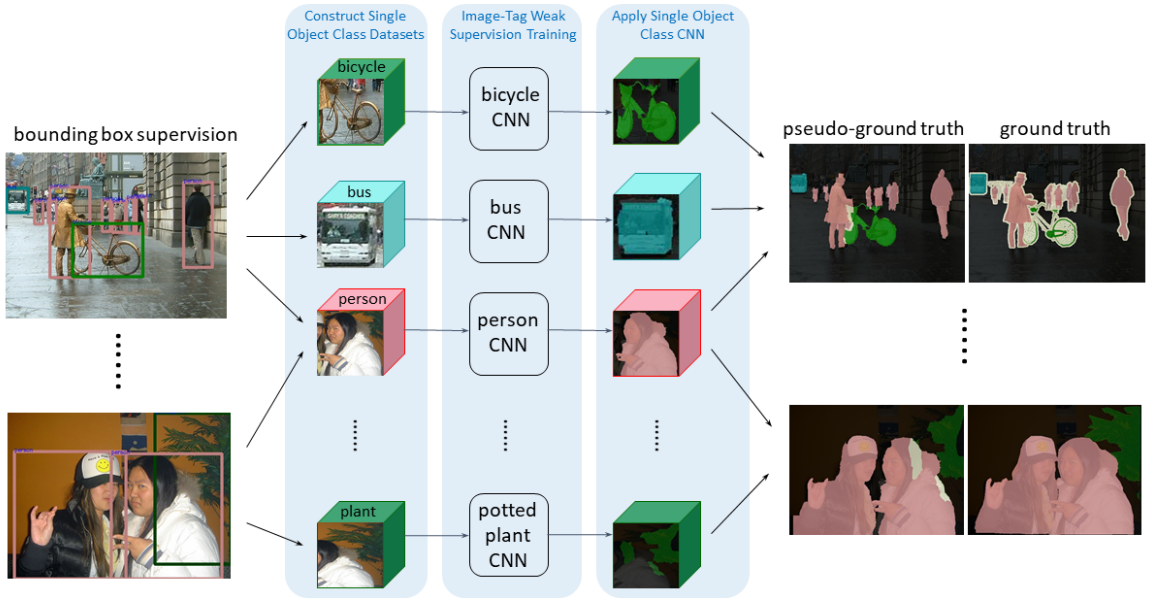

Figure 4.1: Overview of our approach. First we construct single object class datasets by cutting out the bounding boxes from the annotated training images and separating them by class. Then we train CNN in image-tag weakly supervised setting on each dataset. The next step is to apply the trained CNN back to the training bounding boxes. The obtained segmentations are combined with conflict resolution to construct pseudo-ground truth. The final step (not illustrated) is to train a standard segmentation CNN on pseudo-ground truth.

4.1 Our Approach

4.1.1 Single Object Dataset Construction

For each semantic class, we crop out its boxes and put them into a separate single object class dataset. When cropping a box, we take a border of 3 pixels in width from the surrounding image outside the box. The border is used for the positional loss, which encourages border pixels to be the background. If pixels in the border region contain boxes of another object, we mark them as void. The void class is ignored during loss calculation. Border pixels provide examples of the background class and make it easier to learn to segment the object from the background. Strictly speaking, this makes training on our single class dataset a mixture of image-tag and scribble supervision, where scribbles are provided for the background class only, and only at the border of each sample. Still, we refer to this setting as image-tag weak supervision, as the main source of supervision is through image-tags. In fact we could still learn without the border pixel labels, similar to how it is done in [111]. But since the box annotation data does provide labels for the pixels outside any box, it is advantageous to make use of it.

Suppose we are constructing a single object class dataset for, say, class *cat*. We could take all the boxes for the class *cat*. Some of the boxes for class *cat* intersect boxes of objects of other classes. In case of intersection with another class, the box of class *cat* may have pixels of that other object class. Learning appearance of the *cat* class is made harder by presence of another class, since we are learning only with image-tag annotations. Therefore, we only take the boxes for the class that do not intersect with boxes from the other classes. Note that in Pascal VOC [35] dataset which we use for evaluation, there are objects from the classes of interest without a bounding box annotations around them. Thus any box which does not have an intersection with a box from another class in the annotation, still may have pixels from the other classes. But excluding the boxes that are marked as intersecting with other classes does help to construct a cleaner dataset. Intersection with its own class does not present difficulties, since no mixing between different classes occurs in such case. Thus, for example, if a box of class *cat* intersects with another box of class *cat*, we still include it in the single class dataset for object *cat*.

Lastly, we take boxes in the single class dataset only if their longer side is bigger than 50 pixels. Otherwise a box is too small in resolution and is less helpful for learning appearance. We rescale all cropped boxes to be of size 256 by 256 for training CNN. Notice that the border region can either shrink or expand, depending on whether the original box is larger or smaller than the training image size.

4.1.2 Single Object Class Training

We train a single class CNN with a regularized loss function, which we redesign from Chapter 3 to better suit a single object class dataset derived from bounding box annotations.

In Chapter 3, we observe that in a single class dataset, the central pixel is likely to be the object. Since they deal with general images, the object size can be rather small. Therefore, we use a rather modest prior, only the small central patch is encouraged to be the object. In contrast, for bounding boxes, we can assume that the bounding box is likely to cover the object rather tightly, and can use a stricter, and, therefore, more useful prior on the object position.

We replace the central pixel prior by a tight box prior, inspired by [58]. Our tight box prior encourages either every row or every column in the box to contain at least one object pixel. This means that the object fills out the full width or the full height of the box. We do not expect the object to fill out both width and height, since often bounding boxes are placed around the object more loosely than that, for example, see the bounding box for the plant in Fig. 4.1, bottom left. However, usually a bounding box is tight in at least the horizontal or the vertical dimension.

An additional modification is also to the positional prior. In [111] and Chapter 3, the border of the image is not guaranteed to be the background since they consider general images. Therefore, we used mean squared loss, which has a smaller penalty than cross entropy in case when a border pixel is assigned to the object. Since we cropped the border of each sample from the outside of a bounding box, we know the border region belongs to the background, so we use cross entropy instead of a mean squared loss.

For a training image, let $\mathcal{B}$ be the border region (cropped from the outside of the corresponding bounding box), and $\mathcal{R}$ be the inner region (corresponding to the actual bounding box). Our positional prior, replacing the one in Eq. (2.5) is

$$\begin{aligned} L_p(\boldsymbol{x}) = &-\frac{1}{|\mathcal{B}|}\sum_{p\in\mathcal{B}}\log\left(1-x_p\right) \\ &+\left(\max\left\{\frac{1}{r}\sum_{row\in\mathcal{R}}\max_{x_p\in row}x_p, \frac{1}{c}\sum_{col\in\mathcal{R}}\max_{x_p\in col}x_p\right\}-1\right)^2, \end{aligned} \tag{4.1}$$

where r and c are the row and column lengths.

The last change to the regularized loss function is to raise the minimum size requirement for the object to $obj_{min} = 0.3$ in Eq. (2.4), since the relative size of objects in our single

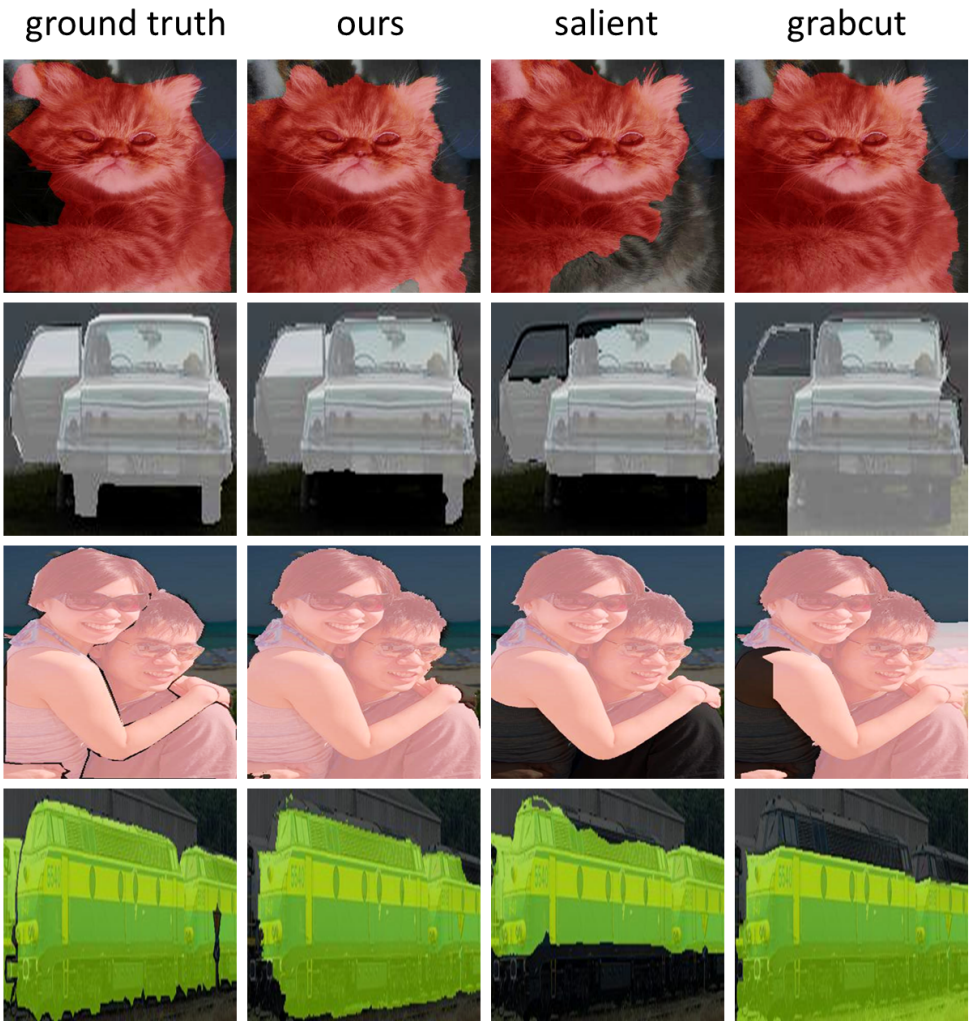

Figure 4.2: Examples of object/background segmentations for our method, salient object detection, and GrabCut.

object datasets is large. This is because the objects in the bounding box tend to be of larger size than for a general single object dataset case considered in [111] and Chapter 3.

Finally, we add the negative loss in Eq. (2.7) to the regularized loss in Eq. (2.6). Although we collect non-overlapping boxes in our single class datasets, still, for each object class, there are sometimes pixels from the other object classes due to the limited precision of bounding box annotations. Negative loss helps to be less sensitive to these by providing images from the other classes as negative examples. With negative loss, when we train for a class, the datasets for all the other classes are used in Eq. (2.7).

We train a CNN for each object class separately, initializing weights from OxfordPet dataset, as described in Sec. 3.1. The CNN architecture is the same as in [111][1]. They use Unet [92] with ResNeXt [120] fixed features in the encoder. The features are pretrained on Imagenet [33].

After single CNN classifiers are trained for each object class, they are applied back to the bounding boxes and a segmentation into object/background is obtained, which is used for constructing pseudo-ground truth (Sec. 4.1.3).

We now compare the accuracy of our box segmentations with those obtained by GrabCut [93] on Pascal VOC 2012 [35] dataset. We use GrabCut implementation in [46]. We optimize the size of the region outside the bounding box to give us the best performance. In addition to GrabCut, we also evaluate salient object detection for segmenting a box, since the state of the art in salient object detection improved tremendously in recent years. We use a recent method BasNet [89][2]. Note that BasNet was trained on pixel precise ground truth from salient object benchmarks. We use F_β performance metric, defined as

$$F_\beta = \frac{(1+\beta^2) precision \times recall}{\beta^2 \times precision + recall},$$

with $\beta^2 = 0.3$.

The comparison is in Table 4.1. There are four versions of our method: column **Ours** uses the original positional loss in Eq. (2.5) from [111], column **ours(T)** uses our new positional loss based on the tight box assumption (Eq. (4.1)), and column **ours(T+N)** uses our new tight box positional loss and the negative loss (Eq. (2.7)). Finally, we test how our method performs if instead on training on 'clean' boxes, we use all boxes in the training dataset. In particular, we take all boxes even if they are small and overlap with other boxes. For consistency, we test only on clean boxes, since the method in all other

[1] https://github.com/morduspordus/SingleClassRL
[2] https://github.com/xuebinqin/BASNet

columns were tested on clean boxes only. The results of training on all boxes and testing on clean boxes is in the new column **ours(all boxes)**. Note that this column is obtained with our loss function, namely the tight bounding box loss. For all four of **our** methods, we kept the $obj_{min} = 0.3$ in the minimum volume loss.

Our new tight box positional prior significantly outperforms the positional prior from [111] (column **ours** vs. **ours(T)**). Performance is improved for all classes. The largest improvement, by over 4 points, are for the table and plant classes. The appearance of these classes is harder to model in weakly supervised setting, so having a stronger prior helps them the most. Adding negative loss (column **ours(T)** vs. **ours(T+N)**) improves the performance for almost all classes, but only slightly.

Surprisingly, training on all boxes is not much worse than training on clean boxes. The chair fares the worst by far, probably due to high confusion with the table class. Interestingly, training on all boxes gives a significantly improved performance over all other of our methods for the table class. This is most likely due to the dataset size. Class table has only 65 clean bounding boxes, and this is a very small dataset to learn the appearance from. There are 715 training boxes overall, if we do not remove overlapping boxes, thus training on all boxes for the table is advantageous, even if they have overlap with other classes. In principle, we could decide to include 'not clean' boxes in our dataset if the number of clean boxes for the class is small.

All four versions of our method outperform MCG, denseCRF, GrabCut and salient object detection by a large margin in terms of the mean F_β score. Also our methods with tight box prior are better than GrabCut and saliency for each individual class.

Saliency, although not previously used for bounding box segmentation, performs best out of methods other than ours. Some classes are naturally more salient (easily identified) than others, for example, the bird class. Here saliency achieves almost the same performance as we do. However, some classes, such as sofa and table, are far from salient, and, the difference in performance is almost 15 points of F_β score.

Some example bounding box segmentations are in Fig. 4.2. GrabCut tends to join pieces of the background to the object, likely due to a poor model of object appearance constructed only from one box. Salient object detection is trained on a large dataset. However, it is trained for salient object detection, and, therefore, it tends to focus on more salient object parts, such as the skin of the person or the middle part of the train. Our method learns object appearance (with weak tag supervision) for each class from a large set of class-specific samples, and therefore is able to learn a class-specific appearance model that leads to a more accurate segmentation.

For implementing MCG, we follow the prior work [32, 61], and select the proposal

class	#boxes	MCG	denseCRF	grabcut	salient	ours(all boxes)	ours	ours(T)	ours(T+N)
aero	608	63.72	70.97	67.12	88.43	84.95	87.13	88.88	**89.12**
bike	243	58.41	69.56	69.60	80.00	84.40	84.37	85.59	**85.74**
bird	827	67.32	76.54	70.69	90.07	87.69	91.16	91.57	**91.70**
boat	444	65.73	76.68	76.58	77.57	82.89	84.91	**87.26**	87.14
bottle	376	78.61	85.19	84.97	82.50	91.52	89.28	92.45	**92.68**
bus	310	76.75	89.32	87.04	92.71	94.17	91.44	**94.73**	94.60
car	909	72.07	86.27	81.99	88.00	88.88	89.23	92.54	**92.90**
cat	901	73.86	86.49	80.44	84.62	92.55	92.69	94.71	**94.73**
chair	1003	58.73	69.41	63.48	68.28	68.72	74.71	75.99	**78.64**
cow	428	72.54	84.99	75.01	87.17	87.70	91.42	**92.56**	92.53
table	65	62.45	83.63	82.22	63.32	**85.26**	75.26	79.59	79.69
dog	953	72.58	85.95	79.40	89.25	92.68	93.98	94.36	**94.42**
horse	359	66.57	79.22	71.00	88.22	88.22	90.26	91.35	**91.53**
mbike	248	59.30	80.60	75.92	85.48	88.87	87.95	**89.61**	**89.61**
person	4029	70.02	80.01	78.29	84.60	80.56	85.83	88.02	**89.05**
plant	437	63.13	81.24	76.10	74.62	86.43	82.59	87.51	**87.69**
sheep	548	73.95	84.66	76.78	85.22	87.19	88.67	91.42	**91.68**
sofa	256	70.07	78.13	73.04	66.04	79.10	78.79	81.59	**81.99**
train	445	68.64	83.94	78.91	85.89	89.33	87.86	92.05	**92.08**
tv	455	82.09	86.96	85.00	83.52	90.16	88.18	92.20	**92.36**
mean F_β		68.83	81.00	76.68	82.28	86.56	86.79	89.20	**89.50**

Table 4.1: Comparison of object/background segmentation accuracy on training bounding boxes from Pascal VOC 2012 dataset using MCG [82], denseCRF [51] GrabCut [93], salient object detection [89] and four versions of our method. See text for explanation of four different versions of our method. Performance metric is F_β score (higher is better).

that has the highest IoU measure with the bounding box. We do not use MCG ranking of proposals, since the ranking was trained with pixel-precise supervision on Pascal VOC dataset. For implementing denseCRF [51] we use the same approach as in [82]. We select $\alpha\%$ of pixels in the center of the box and label them as preferring the object with probability p. Then we select a border of pixels outside the box to be set to the background. All other pixels are set to an unknown label. Then the unary CRF terms are set as suggested in the standard implementation package[3]. We chose the appropriate setting of α, p and denseCRF parameters on a small held-out set of training images (100 images, fully annotated).

[3] https://github.com/lucasb-eyer/pydensecrf

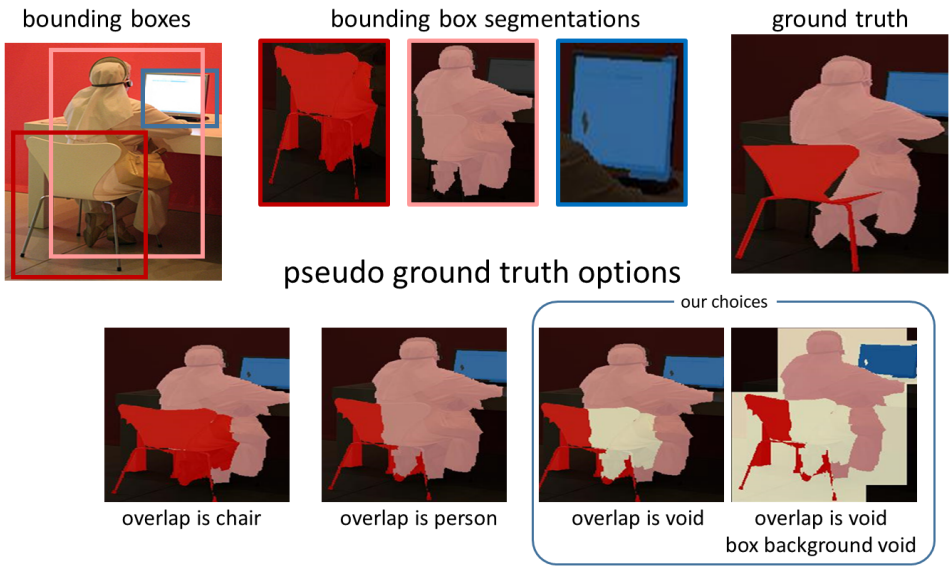

Figure 4.3: Illustrates pseudo-ground truth construction. Left: original image with bounding boxes for chair, person, and monitor classes. Top, middle: segmentations produced by our single class CNNs. Chair and person segmentations overlap. Bottom: four possible choices for pseudo-ground truth construction. We evaluate two approaches, as indicated in 'our choices' box.

4.1.3 Pseudo Ground Truth Construction

We now describe how we construct pseudo-ground truth after training a single-class CNN for each class. For each bounding box in the training data, we apply our trained CNN that corresponds to the class of the box. Note that at this stage, we apply trained CNN to all bounding boxes in the training data, whether they overlap the other boxes or not, and even if they have a small size. This is done in order to obtain as dense ground truth as possible.

Pixels that do not belong to any box are marked as background. Any pixel that gets segmented as an object in only one box, or in several boxes of the same class, gets assigned to the class corresponding to the box. If a pixel gets segmented as an object by two or more boxes of different classes, see Fig. 4.3, we have several choices to make. We could assign it to one of the overlapping classes, but this is error prone. In the example in Fig. 4.3, whether we assign pixels in the overlap to *person* or *chair* class, we will make mistakes. Therefore we assign pixels in the overlap to the void class, as illustrated on the bottom right.

For the pixels that get labeled as background inside a bounding box, we consider and evaluate two options. The first option is to assign them to the background (Fig. 4.3, left in the 'our choices' box). We call this option *background-dense*. The second option is to mark any background pixel inside the bounding box as void. This results in less samples for the background class, but with less object pixels erroneously assigned to the background. And we already have many samples of background pixels from the parts of the data that does not fall into any box. We call the second option *background-sparse*.

4.2 Experimental Results

We evaluate our approach on Pascal VOC 2012 dataset [35]. Following previous work, we use augmented annotations from [41] for a total of 10,582 training images. The number of validation images is 1,449. For training from our pseudo-ground truth, we use DeepLab-ResNet101 [24] pretrained on ImageNet [33]. We train on 513x513 images and use random horizontal flip, rescale, and Gaussian blur for data augmentation. We use stochastic gradient descent with the same parameters as in [24], except our initial learning rate is 10^{-4}. Our implementation is in PyTorch [83] and we use NVIDIA GeForce RTX 2080 Ti graphics card.

When we train on pseudo-ground truth which is *background-dense*, we train with standard cross entropy for 200 epochs. When we train with *background-sparse*, because the

method	**backbone**	**mIoU**
WSSL (CRF)[82]	VGG-16	60.6
BoxSup (CRF) [32]	VGG-16	62.0
SDI (CRF) [48]	Resnet-101	65.7
BCM (CRF) [101]	Resnet-101	70.2
GCMCG(CRF) [61]	Resnet-101	74.3
Box2Seg [54]	Resnet-101	74.9
Box2Seg (CRF)[54]	Resnet-101	76.4
Ours (bckg-sparse)	Resnet-101	75.7
Ours (bckg-dense)	Resnet-101	**76.6**
Full supervision	Resnet-101	77.8

Table 4.2: Comparison of our approach to previous bounding box weakly supervised semantic segmentation methods on PASCAL VOC 2012 validation set. Here (CRF) means the method uses denseCRF [51] post-processing. Performance metric is $mUoI$. The last line is performance of CNN we use when trained with pixel precise supervision.

method	bckg	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mean
GCMCG(CRF) [61]	93.3	85.0	35.9	88.6	70.3	77.9	91.9	83.6	90.5	39.2	84.5	59.4	86.5	82.4	81.5	84.3	57.0	85.9	55.8	85.8	70.4	75.7
Box2Seg [54]	92.5	66.5	31.7	78.9	65.5	83.4	90.4	86.7	86.0	55.1	81.8	59.9	80.5	74.1	76.0	75.7	65.3	85.1	72.5	87.8	77.7	74.9
Box2Seg (CRF) [54]	93.3	72.4	33.0	84.2	64.9	83.5	90.9	86.7	88.7	57.2	83.6	62.5	82.6	76.8	77.0	77.8	63.3	87.2	75.1	88.3	74.1	76.4
Ours (bckg-sparse)	92.0	81.3	36.5	86.9	73.1	79.5	90.5	85.9	88.6	41.7	86.3	63.6	85.9	83.4	81.3	81.3	62.9	81.3	48.2	83.2	77.3	75.7
Ours (bckg-dense)	92.9	88.5	39.9	89.4	78.7	79.9	89.3	87.8	91.8	36.0	84.8	51.9	88.2	84.7	85.3	82.1	64.3	83.7	50.8	85.1	74.2	76.6

Table 4.3: Per-class results on Pascal VOC 2012 validation set of our methods and prior methods that made per-class results available.

Figure 4.4: Example results (for *background-dense*). In each image pair, the left is the ground truth, the right is our result.

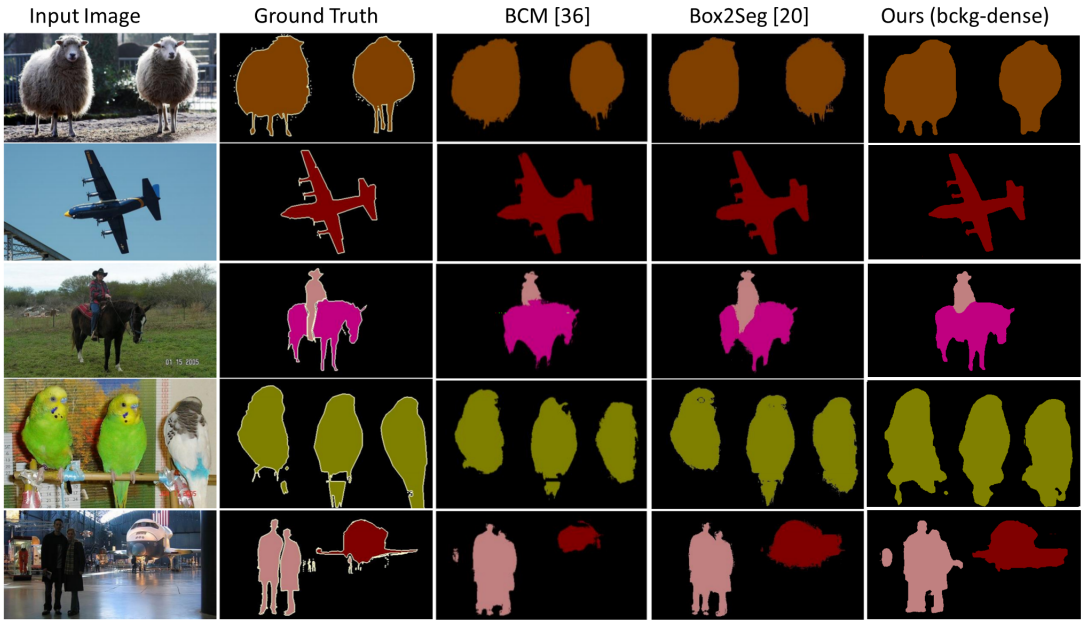

Figure 4.5: Comparison of our results (last column) to BCM [101] and Box2Seg [54]. BCM and Box2Seg results are with denseCRF post-processing. Our results are without post-processing.

number of void pixels is relatively large, we get slightly better results if we add denseCRF loss from [109][4] to the cross-entropy loss. We use the same setting of denseCRF and its relative weight as in [109]. When training with denseCRF loss, we use the same strategy as in [109]: first we train only with cross-entropy (for 100 epochs), and then with both cross entropy and denseCRF (for another 100 epochs). For comparison, we also train DeepLab-ResNet101 with pixel precise ground truth, for 200 epochs, and with the same setting of all other parameters.

We compare the accuracy of our algorithm to the recent bounding box weakly supervised methods in Table 4.5. The evaluation metric is $mIoU$. Methods using Resnet-101 backbone perform better. Most methods use denseCRF [51] as post-processing (marked with CRF in parenthesis in the table), and do not report results without post-processing. Typically performance with denseCRF post-procesing improves the results by about 2 points of $mIoU$ measure.

We do not use denseCRF postprocessing. Previously best performing method is Box2Seg [54], their $mIoU = 74.9$ without post-processing. Our result with *background-sparse* is $mIoU = 75.7$, and with *background-dense* is $mIoU = 76.6$, both are better than $mIoU = 74.9$ Box2Seg [54] achieves without denseCRF post-processing. Our result with *background-dense* is slightly better than Box2Seg [54] with denseCRF post-processing.

Somewhat unexpectedly, our *background-dense* approach works better than *background-sparse*, even though there are more pixels marked as background erroneously (see supplementary materials).

We are using a simple cross-entropy based loss function. In [54], they use a loss function that better handles noisy pseudo-ground truth. It is likely that if our masks learned with image-tag weak supervision are used in conjunction with the loss in [54][5], the performance would improve.

The last line in Table 4.5 gives the performance of the same CNN as ours but trained with pixel precise ground truth. Our method is only 1.2 points of $mIoU$ measure behind. Per-class result comparison with prior work that made per-class results available are in Table 4.3.

Some example segmentations on the images chosen by [101] are in Fig. 4.5. Our results are without any post-processing. We capture sheep legs (top row) and horse legs (middle row) better than prior work. In the last row, we have small scale human figures captured (to the right of the two main figures), while the other methods most likely smooth them out due to CRF post-processing. More examples of our segmentations are in Fig. 4.4.

[4] https://github.com/meng-tang/rloss

[5] Their code is not available online yet.

Appendices contain more experiments and ablation studies.

Additional Experiments

In Table 4.4 we provide experimental comparison on PASCAL VOC 2012 test set. About half of the related work does not provide results on the test set, including two most recent works [101, 54]. Most recent related work that provides test set results is [61]. They use denseCRF [51] post-processing and achieve $mIoU = 75.5$. Our score on test data is slightly better than that, $mIoU = 76.1$. In Figure 4.6 we show some example segmentations of our approach (*background-dense*) on the test data.

Next we compare different training regimes for *background-light* approach, where pixels segmented as background inside any box get labeled as void, to minimize the number of object pixels that are mistakenly labeled as background. Since many pixels are labeled as void (23.53%), training benefits from including denseCRF loss from [109], in addition to the cross entropy loss. The method in [109] was specifically designed for training with the scribble form of weak supervision, so that a large portion of image pixels are labeled as void. Adding denseCRF loss encourages pixels with void labels to get assigned to similar labels to those pixels that are similar in color and have ground truth labels for training. We use the same parameter setting as in [109]. They also observe that training with just cross entropy first and then adding denseCRF loss works better. This approach also works the best for us. Training with cross-entropy alone, we get $mIoU = 71.87$. Adding denseCRF loss to cross entropy from the beginning gets $mIoU = 74.99$. Adding denseCRF loss after training with cross entropy first gives the best result, $mIoU = 75.71$. For our *background-dense* approach, where pixels in the bounding boxes that are classified as background get labeled as background, there is no benefit in adding denseCRF loss. There are only 3.07% of pixels that are labeled as void in this case.

method	backbone	mIoU
WSSL (CRF)[82]	VGG-16	62.2
BoxSup (CRF) [32]	VGG-16	64.6
SDI (CRF) [48]	Resnet-101	-
BCM (CRF) [101]	Resnet-101	-
GCMCG(CRF) [61]	Resnet-101	75.5
Box2Seg(CRF) [54]	Resnet-101	-
Ours (bckg-dense)	Resnet-101	**76.1**

Table 4.4: Comparison of our approach to previous bounding box weakly supervised semantic segmentation methods on PASCAL VOC 2012 test set. Here (CRF) means the method uses denseCRF [51] post-processing. Performance metric is $mIoU$. Some prior work methods do not report results on test data.

method	backbone	mIoU Val	mIoU Test
WSSL (CRF)[82]	VGG-16	60.6	62.2
BoxSup (CRF) [32]	VGG-16	62.0	64.6
SDI (CRF) [48]	Resnet-101	65.7	67.5
BCM (CRF) [101]	Resnet-101	70.2	-
GCMCG(CRF) [61]	Resnet-101	74.3	75.5
Box2Seg [54]	Resnet-101	74.9	-
Box2Seg (CRF)[54]	Resnet-101	76.4	-
Ours (bckg-sparse)	Resnet-101	75.7	
Ours (bckg-dense)	Resnet-101	76.6	
Full supervision	Resnet-101	77.8	-

Table 4.5: Comparison of our approach to previous bounding box weakly supervised semantic segmentation methods on PASCAL VOC validation set. Here (CRF) means the method uses denseCRF [51] post-processing.

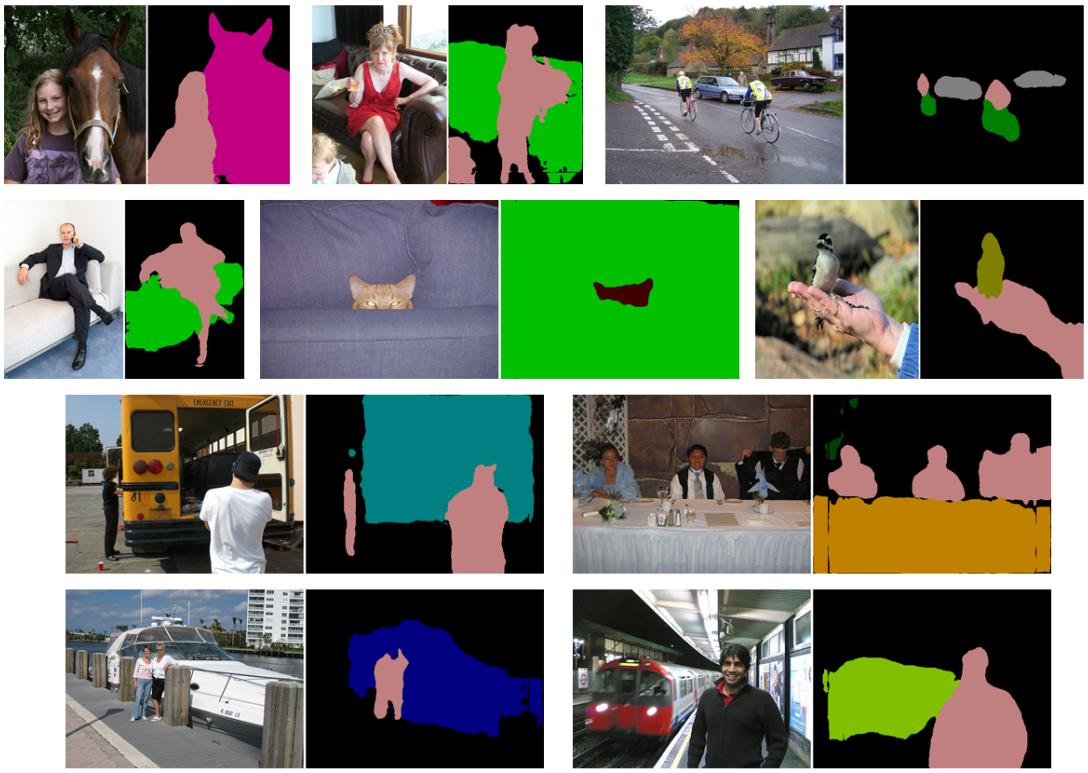

Figure 4.6: Examples of our segmentations on Pascal VOC 2012 test set. Each image pair shows the input image and our segmentation.

Chapter 5

Conclusion and Future work

In this thesis, we proposed a hyperparameter search method for regularized loss in the context of single object class dataset with image level weak supervision and showed it significantly outperforms the previous approach with fixed hyperparameters. Our method is simple, easy to transfer between different applications without changing architecture or loss function. We produce new state-of-the-art results in image level weakly supervised salient and semantic segmentation. In the future, we plan to improve regularized loss function to lessen the gap between weakly supervised and fully supervised performance for single object datasets.

With the help of our single class method, we proposed an approach for semantic segmentation with bounding box weak supervision that takes advantage of the collective information in the boxes of the same class to learn the appearance of the corresponding class. This, in turn, is used to segment each bounding box more accurately. All current bounding box weak supervision methods make use of some segmentation method in each bounding box, independently of other boxes. Thus all prior work could benefit from our approach for more accurate box segmentation. One promising simple direction for a further improvement is to add more image-tag data for supervision, for example, by web image search on class names.

References

[1] Diagram for simple neurons. https://www.kaggle.com/learn/intro-to-deep-learning. Accessed: 2021-6-21. viii, 11, 12, 14

[2] example of convolution operation. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. Accessed: 2021-6-21. viii, 18, 20

[3] An example of general image classification task using cnn. https://cezannec.github.io/Convolutional_Neural_Networks/. Accessed: 2021-6-21. viii, 17

[4] An example of general image segmentation from a tutorial. https://ai.stanford.edu/~syyeung/cvweb/tutorial3.html. Accessed: 2021-6-21. viii, 1

[5] Jiwoon Ahn and Suha Kwak. Learning pixel-level semantic affinity with image-level supervision for weakly supervised semantic segmentation. In *Conference on Computer Vision and Pattern Recognition*, pages 4981–4990, 2018. 6, 29, 58, 59

[6] Sharon Alpert, Meirav Galun, Achi Brandt, and Ronen Basri. Image segmentation by probabilistic bottom-up aggregation and cue integration. *IEEE transactions on pattern analysis and machine intelligence*, 34(2):315–327, 2011. 49

[7] Tanmay Anand, Soumendu Sinha, Murari Mandal, Vinay Chamola, and F Richard Yu. Agrisegnet: Deep aerial semantic segmentation framework for iot-assisted precision agriculture. *IEEE Sensors Journal*, 2021. 3

[8] Nikita Araslanov and Stefan Roth. Single-stage semantic segmentation from image labels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4253–4262, 2020. 29, 58, 59

[9] Pablo Arbeláez, Bharath Hariharan, Chunhui Gu, Saurabh Gupta, Lubomir Bourdev, and Jitendra Malik. Semantic segmentation using regions and parts. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3378–3385. IEEE, 2012. 4

[10] Ruzena Bajcsy and Mohamad Tavakoli. A computer recognition of bridges, islands, rivers and lakes from satellite pictures. In *LARS Symposia*, page 12, 1973. 4

[11] Amy Bearman, Olga Russakovsky, Vittorio Ferrari, and Li Fei-Fei. What's the point: Semantic segmentation with point supervision. In *European conference on computer vision*, pages 549–565. Springer, 2016. 64

[12] Moshe Ben-Ezra. intro-segmentation with invisible keying signal. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)*, volume 1, pages 32–37. IEEE, 2000. 2

[13] Yuri Boykov and Vladimir Kolmogorov. Computing geodesics and minimal surfaces via graph cuts. In *9th IEEE International Conference on Computer Vision (ICCV 2003), 14-17 October 2003, Nice, France*, pages 26–33, 2003. 37, 48

[14] 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. 2, 35

[15] Garrick Brazil, Xi Yin, and Xiaoming Liu. Illuminating pedestrians via simultaneous detection & segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4950–4959, 2017. 3

[16] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. 4

[17] Joao Carreira, Rui Caseiro, Jorge Batista, and Cristian Sminchisescu. Semantic segmentation with second-order pooling. In *European Conference on Computer Vision*, pages 430–443. Springer, 2012. 4

[18] Miguel A Carreira-Perpinan. Acceleration strategies for gaussian mean-shift image segmentation. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 1160–1167. IEEE, 2006. 2

[19] Vicent Caselles, Ron Kimmel, and Guillermo Sapiro. Geodesic active contours. *International journal of computer vision*, 22(1):61–79, 1997. 2

[20] Yosun Chang. sur. faced. io: augmented reality content creation for your face and beyond by drawing on paper. In *ACM SIGGRAPH 2019 Appy Hour*, pages 1–2. 2019. 3

[21] Yu-Ting Chang, Qiaosong Wang, Wei-Chih Hung, Robinson Piramuthu, Yi-Hsuan Tsai, and Ming-Hsuan Yang. Weakly-supervised semantic segmentation via sub-category exploration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8991–9000, 2020. 29, 58, 59

[22] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014. 27, 28

[23] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017. 27, 28, 35

[24] 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. 33, 55, 58, 74

[25] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017. 27, 28

[26] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *European Conference on Computer Vision*, pages 833–851, 2018. ix, 5, 25, 27, 29, 35, 40

[27] Ming-Ming Cheng, Niloy J Mitra, Xiaolei Huang, Philip HS Torr, and Shi-Min Hu. Global contrast based salient region detection. *IEEE transactions on pattern analysis and machine intelligence*, 37(3):569–582, 2014. 50

[28] Ming-Ming Cheng, NiloyJ. Mitra, Xiaolei Huang, and Shi-Min Hu. Salientshape: group saliency in image collections. *The Visual Computer*, 30(4):443–453, 2014. 30, 49

[29] Dongxiang Chi, Ying Zhao, and Ming Li. Automatic liver mr image segmentation with self-organizing map and hierarchical agglomerative clustering method. In *2010 3rd International Congress on Image and Signal Processing*, volume 3, pages 1333–1337. IEEE, 2010. 2

[30] Gabriela Csurka and Florent Perronnin. An efficient approach to semantic segmentation. *International Journal of Computer Vision*, 95(2):198–212, 2011. 4

[31] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989. 5

[32] Jifeng Dai, Kaiming He, and Jian Sun. Boxsup: Exploiting bounding boxes to supervise convolutional networks for semantic segmentation. In *International Conference on Computer Vision*, pages 1635–1643, 2015. 6, 8, 29, 33, 64, 71, 75, 80

[33] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 16, 21, 23, 25, 49, 58, 70, 74

[34] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016. viii, 25, 27

[35] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010. 9, 55, 59, 64, 65, 67, 70, 74

[36] Junsong Fan, Zhaoxiang Zhang, Chunfeng Song, and Tieniu Tan. Learning integral objects with intra-class discriminator for weakly-supervised semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4283–4292, 2020. ix, 29, 58, 59, 61

[37] Jerome A Feldman and Yoram Yakimovsky. Decision theory and artificial intelligence: I. a semantics-based region analyzer. *Artificial Intelligence*, 5(4):349–371, 1974. 4

[38] GMNR Gajanayake, Roshan Dharshana Yapa, and B Hewawithana. Comparison of standard image segmentation methods for segmentation of brain tumors from 2d mr images. In *2009 International Conference on Industrial and Information Systems (ICIIS)*, pages 301–305. IEEE, 2009. 2

[39] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984. 2, 4

[40] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016. 5, 11, 25

[41] Bharath Hariharan, Pablo Arbeláez, Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. Semantic contours from inverse detectors. In *International Conference on Computer Vision*, pages 991–998. IEEE, 2011. 74

[42] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. 5, 25

[43] 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. viii, 21, 23, 24, 25, 27

[44] Zilong Huang, Xinggang Wang, Jiasi Wang, Wenyu Liu, and Jingdong Wang. Weakly-supervised semantic segmentation network with deep seeded region growing. In *Conference on Computer Vision and Pattern Recognition*, pages 7014–7023, 2018. 29

[45] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. 35

[46] Itseez. Open source computer vision library. https://github.com/itseez/opencv, 2015. 70

[47] Xu Ji, Joao F Henriques, and Andrea Vedaldi. Invariant information clustering for unsupervised image classification and segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9865–9874, 2019. 2

[48] Anna Khoreva, Rodrigo Benenson, Jan Hendrik Hosang, Matthias Hein, and Bernt Schiele. Simple does it: Weakly supervised instance and semantic segmentation. In *Conference on Computer Vision and Pattern Recognition*, pages 1665–1674, 2017. 6, 8, 29, 33, 64, 75, 80

[49] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 49

[50] Alexander Kolesnikov and Christoph H. Lampert. Seed, expand and constrain: Three principles for weakly-supervised image segmentation. In *European Conference on Computer Vision*, pages 695–711, 2016. 6, 29

[51] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *Neural Information Processing Systems*, pages 109–117, 2011. xii, 32, 33, 35, 48, 58, 72, 75, 78, 79, 80

[52] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 17

[53] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012. 5, 16, 21, 23

[54] Viveka Kulharia, Siddhartha Chandra, Amit Agrawal, Philip Torr, and Ambrish Tyagi. Box2seg: Attention weighted loss and discriminative feature learning for weakly supervised segmentation. 2020. x, 6, 8, 29, 33, 34, 64, 75, 77, 78, 79, 80

[55] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001. 4

[56] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. 5, 16, 21

[57] Jungbeom Lee, Eunji Kim, Sungmin Lee, Jangho Lee, and Sungroh Yoon. Ficklenet: Weakly and semi-supervised semantic image segmentation using stochastic inference. In *Conference on Computer Vision and Pattern Recognition*, pages 5267–5276, 2019. 6, 29, 58, 59

[58] Victor Lempitsky, Pushmeet Kohli, Carsten Rother, and Toby Sharp. Image segmentation with a bounding box prior. In *2009 IEEE 12th international conference on computer vision*, pages 277–284. IEEE, 2009. 68

[59] Victor Lempitsky, Andrea Vedaldi, and Andrew Zisserman. Pylon model for semantic segmentation. In *Advances in neural information processing systems*, pages 1485–1493. Citeseer, 2011. 4

[60] G. Li and Y. Yu. Deep contrast learning for salient object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 478–487, June 2016. 49

[61] Qizhu Li, Anurag Arnab, and Philip HS Torr. Weakly-and semi-supervised panoptic segmentation. In *Proceedings of the European Conference on Computer Vision*, pages 102–118, 2018. 6, 8, 29, 33, 40, 49, 50, 64, 71, 75, 79, 80

[62] Yin Li, Xiaodi Hou, Christof Koch, James M Rehg, and Alan L Yuille. The secrets of salient object segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 280–287, 2014. 49

[63] Di Lin, Jifeng Dai, Jiaya Jia, Kaiming He, and Jian Sun. Scribblesup: Scribble-supervised convolutional networks for semantic segmentation. In *Conference on Computer Vision and Pattern Recognition*, pages 3159–3167, 2016. 6, 29, 34

[64] Tie Liu, Jian Sun, Nan-Ning Zheng, Xiaoou Tang, and Heung-Yeung Shum. Learning to detect a salient object. In *Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007. 49

[65] 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. 5, 25

[66] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004. 4

[67] Gloria Menegaz and Rosa Lancini. Semantic segmentation of angiographic images. In *Proceedings of 18th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, volume 2, pages 670–671. IEEE, 1996. 3

[68] David J Michael and Alan C Nelson. Handx: a model-based system for automatic segmentation of bones from digital hand radiographs. *IEEE transactions on medical imaging*, 8(1):64–69, 1989. 2

[69] Branislav Mičušĺík and Jana Košecká. Semantic segmentation of street scenes by superpixel co-occurrence and 3d geometry. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 625–632. IEEE, 2009. 4

[70] Pierangelo Migliorati, Federico Pedersini, L Sorcinelli, and Stefano Tubaro. Semantic segmentation applied to image interpolation in the case of camera panning and zooming. In *1993 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages 25–28. IEEE, 1993. 3

[71] Andres Milioto, Philipp Lottes, and Cyrill Stachniss. Real-time semantic segmentation of crop and weed for precision agriculture robots leveraging background knowledge in cnns. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 2229–2235. IEEE, 2018. 3

[72] Hossein Mobahi, Shankar R Rao, Allen Y Yang, Shankar S Sastry, and Yi Ma. Segmentation of natural images by texture and boundary compression. *International journal of computer vision*, 95(1):86–98, 2011. 2

[73] Vida Movahedi and James H Elder. Design and perceptual validation of performance measures for salient object segmentation. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Workshops*, pages 49–56. IEEE, 2010. 49

[74] Josh Myers-Dean and Scott Wehrwein. Semantic pixel distances for image editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 534–535, 2020. 3

[75] Tam Nguyen, Maximilian Dax, Chaithanya Kumar Mummadi, Nhung Ngo, Thi Hoai Phuong Nguyen, Zhongyu Lou, and Thomas Brox. Deepusps: Deep robust unsupervised saliency prediction via self-supervision. In *Advances in Neural Information Processing Systems*, pages 204–214, 2019. 30, 31, 40, 49, 50

[76] Richard Nock and Frank Nielsen. Statistical region merging. *IEEE Transactions on pattern analysis and machine intelligence*, 26(11):1452–1458, 2004. 2

[77] Ron Ohlander, Keith Price, and D Raj Reddy. Picture segmentation using a recursive region splitting method. *Computer Graphics and Image Processing*, 8(3):313–333, 1978. 2

[78] Yu-ichi Ohta, Takeo Kanade, and Toshiyuki Sakai. An analysis system for scenes containing objects with substructures. In *Proceedings of the Fourth International Joint Conference on Pattern Recognitions*, pages 752–754, 1978. 4

[79] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979. 2

[80] Şaban Öztürk and Bayram Akdemir. A convolutional neural network model for semantic segmentation of mitotic events in microscopy images. *Neural Computing and Applications*, 31(8):3719–3728, 2019. 3

[81] Dim P Papadopoulos, Jasper RR Uijlings, Frank Keller, and Vittorio Ferrari. Extreme clicking for efficient object annotation. In *International Conference on Computer Vision*, pages 4930–4939, 2017. 64

[82] George Papandreou, Liang Chieh Chen, Kevin P. Murphy, and Alan L. Yuille. Weakly and semi supervised learning of a deep convolutional network for semantic image segmentation. In *International Conference on Computer Vision*, pages 1742–1750, 2015. xii, 6, 8, 29, 33, 34, 64, 72, 75, 80

[83] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 74

[84] Deepak Pathak, Philipp Krahenbuhl, and Trevor Darrell. Constrained convolutional neural networks for weakly supervised segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1796–1804, 2015. 29

[85] Federico Perazzi, Philipp Krähenbühl, Yael Pritch, and Alexander Hornung. Saliency filters: Contrast based filtering for salient region detection. In *Conference on Computer Vision and Pattern Recognition*, pages 733–740. IEEE, 2012. 30, 50

[86] Pedro H. O. Pinheiro and Ronan Collobert. From image-level to pixel-level labeling with convolutional networks. In *Conference on Computer Vision and Pattern Recognition*, pages 1713–1721, 2015. 6, 29

[87] Jordi Pont-Tuset, Pablo Arbelaez, Jonathan T Barron, Ferran Marques, and Jitendra Malik. Multiscale combinatorial grouping for image segmentation and object proposal generation. *IEEE transactions on pattern analysis and machine intelligence*, 39(1):128–140, 2016. 6, 33

[88] Xuebin Qin, Zichen Zhang, Chenyang Huang, Masood Dehghan, Osmar R Zaiane, and Martin Jagersand. U2-net: Going deeper with nested u-structure for salient object detection. *Pattern Recognition*, 106:107404, 2020. 51

[89] Xuebin Qin, Zichen Zhang, Chenyang Huang, Chao Gao, Masood Dehghan, and Martin Jagersand. Basnet: Boundary-aware salient object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7479–7489, 2019. xii, 51, 70, 72

[90] Nilanjan Ray, Baidyanath Saha, and Scott T Acton. Oil sand image segmentation using the inclusion filter. In *2008 15th IEEE International Conference on Image Processing*, pages 2188–2191. IEEE, 2008. 2

[91] Siddheswar Ray and Rose H Turi. Determination of number of clusters in k-means clustering and application in colour image segmentation. In *Proceedings of the 4th*

international conference on advances in pattern recognition and digital techniques, pages 137–143. Citeseer, 1999. 2

[92] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention*, pages 234–241, 2015. viii, 5, 25, 26, 40, 49, 54, 70

[93] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. *ACM TOG*, 23:309–314, 2004. xii, 6, 33, 34, 64, 70, 72

[94] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. 15

[95] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *International Conference on Computer Vision*, pages 618–626, 2017. 6

[96] Tong Shen, Guosheng Lin, Chunhua Shen, and Ian Reid. Bootstrapping the performance of webly supervised semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1363–1371, 2018. 58, 59

[97] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000. 2, 35

[98] Jianping Shi, Qiong Yan, Li Xu, and Jiaya Jia. Hierarchical image saliency detection on extended cssd. *IEEE transactions on pattern analysis and machine intelligence*, 38(4):717–729, 2015. 30, 31

[99] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 21, 23, 25

[100] Lucas R Smith and Elisabeth R Barton. Smash–semi-automatic muscle analysis using segmentation of histology: a matlab application. *Skeletal muscle*, 4(1):1–16, 2014. 2

[101] Chunfeng Song, Yan Huang, Wanli Ouyang, and Liang Wang. Box-driven class-wise region masking and filling rate guided loss for weakly supervised semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3136–3145, 2019. x, 6, 8, 29, 33, 34, 64, 75, 77, 78, 79, 80

[102] 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. 35

[103] Dale W Starr and Alan K Mackworth. Exploiting spectral, spatial and semantic constraints in the segmentation of landsat images. *Canadian Journal of Remote Sensing*, 4(2):101–107, 1978. 4

[104] Guolei Sun, Wenguan Wang, Jifeng Dai, and Luc Van Gool. Mining cross-image semantics for weakly supervised semantic segmentation. *European Conference on Computer Vision*, 2020. 29, 58, 59

[105] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999. 4

[106] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 21, 23

[107] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010. 1

[108] Meng Tang, Abdelaziz Djelouah, Federico Perazzi, Yuri Boykov, and Christopher Schroers. Normalized cut loss for weakly-supervised CNN segmentation. In *Conference on Computer Vision and Pattern Recognition*, pages 1818–1827, 2018. 6, 29, 34, 35, 36

[109] Meng Tang, Federico Perazzi, Abdelaziz Djelouah, Ismail Ben Ayed, Christopher Schroers, and Yuri Boykov. On regularized losses for weakly-supervised CNN segmentation. In *European Conference on Computer Vision*, pages 524–540, 2018. 6, 29, 34, 35, 36, 78, 79

[110] Andrea Vedaldi. Cats and dogs. In *Conference on Computer Vision and Pattern Recognition*, 2012. 38, 49, 53

[111] Olga Veksler. Regularized loss for weakly supervised single class semantic segmentation. In *European Conference on Computer Vision*, pages 348–365. Springer, 2020. ix, xi, 6, 7, 8, 32, 36, 38, 40, 41, 42, 43, 44, 46, 47, 48, 49, 50, 55, 56, 58, 59, 65, 67, 68, 70, 71

[112] Paul Vernaza and Manmohan Chandraker. Learning random-walk label propagation for weakly-supervised semantic segmentation. In *Conference on Computer Vision and Pattern Recognition*, pages 2953–2961, 2017. 6, 29

[113] G. Vogiatzis, P.H.S. Torr, and R. Cipolla. Multi-view stereo via volumetric graph-cuts. In *Conference on Computer Vision and Pattern Recognition*, pages II: 391–398, 2005. 7, 40

[114] David L Waltz. Generating semantic descriptions from drawings of scenes with shadows. 1972. 4

[115] Lijun Wang, Huchuan Lu, Yifan Wang, Mengyang Feng, Dong Wang, Baocai Yin, and Xiang Ruan. Learning to detect salient objects with image-level supervision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 136–145, 2017. 40, 44, 49, 50, 51

[116] Lijun Wang, Huchuan Lu, Yifan Wang, Mengyang Feng, Dong Wang, Baocai Yin, and Xiang Ruan. Learning to detect salient objects with image-level supervision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 136–145, 2017.

[117] Jacob Whitehill, Ting-fan Wu, Jacob Bergsma, Javier Movellan, and Paul Ruvolo. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. *Advances in neural information processing systems*, 22:2035–2043, 2009. 31

[118] Xide Xia and Brian Kulis. W-net: A deep model for fully unsupervised image segmentation. *arXiv preprint arXiv:1711.08506*, 2017. 2

[119] Jianxiong Xiao and Long Quan. Multiple view semantic segmentation for street view images. In *2009 IEEE 12th international conference on computer vision*, pages 686–693. IEEE, 2009. 4

[120] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017. 21, 23, 25, 27, 49, 54, 70

[121] Jia Xu, Alexander G. Schwing, and Raquel Urtasun. Learning to segment under various forms of weak supervision. In *Conference on Computer Vision and Pattern Recognition*, pages 3781–3790, 2015. 6, 8, 29, 64

[122] Qiong Yan, Li Xu, Jianping Shi, and Jiaya Jia. Hierarchical saliency detection. In *Conference on Computer Vision and Pattern Recognition*, pages 1155–1162. IEEE, 2013. 44, 49, 51

[123] Chuan Yang, Lihe Zhang, Huchuan Lu, Xiang Ruan, and Ming-Hsuan Yang. Saliency detection via graph-based manifold ranking. In *Conference on Computer Vision and Pattern Recognition*, pages 3166–3173. IEEE, 2013. 30, 44, 49, 51

[124] Qi Yao and Xiaojin Gong. Saliency guided self-attention network for weakly and semi-supervised semantic segmentation. *IEEE Access*, 8:14413–14423, 2020. 29, 58, 59

[125] Yu Zeng, Yunzhi Zhuge, Huchuan Lu, and Lihe Zhang. Joint learning of saliency detection and weakly supervised semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7223–7233, 2019. 29, 58, 59

[126] Yu Zeng, Yunzhi Zhuge, Huchuan Lu, Lihe Zhang, Mingyang Qian, and Yizhou Yu. Multi-source weak supervision for saliency detection. In *Conference on Computer Vision and Pattern Recognition*, pages 6074–6083, 2019. 49, 50, 51

[127] Dingwen Zhang, Junwei Han, and Yu Zhang. Supervision by fusion: Towards unsupervised learning of deep salient object detector. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4048–4056, 2017. 30, 31, 40, 49, 50

[128] Jianming Zhang, Stan Sclaroff, Zhe Lin, Xiaohui Shen, Brian Price, and Radomir Mech. Minimum barrier salient object detection at 80 fps. In *Proceedings of the IEEE international conference on computer vision*, pages 1404–1412, 2015. 30, 31

[129] Jing Zhang, Xin Yu, Aixuan Li, Peipei Song, Bowen Liu, and Yuchao Dai. Weakly-supervised salient object detection via scribble annotations. In *Conference on Computer Vision and Pattern Recognition*, pages 12546–12555, 2020. ix, xi, 50, 51, 55, 56

[130] Jing Zhang, Tong Zhang, Yuchao Dai, Mehrtash Harandi, and Richard Hartley. Deep unsupervised saliency detection: A multiple noisy labeling perspective. In

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 9029–9038, 2018. 30, 31, 40, 49, 50

[131] 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. 30, 31

[132] Xiaoqi Zhao, Youwei Pang, Lihe Zhang, Huchuan Lu, and Lei Zhang. Suppress and balance: A simple gated network for salient object detection. 2020. 51

[133] Bolei Zhou, Aditya Khosla, Àgata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Conference on Computer Vision and Pattern Recognition*, pages 2921–2929, 2016. 6