# Volumetric Weak Supervision for Semantic Segmentation

by

Sharhad Bashar

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, 2022

## Author's Declaration

## Abstract

Semantic segmentation is a popular task in computer vision. Fully supervised methods are data hungry, they require pixel precise annotations for thousands of images. To reduce user annotation efforts, weak supervision for semantic segmentation is becoming an area of increasing interest. Weak supervision can take many forms: bounding boxes, scribbles, image level labels. Image level supervision is the least annotation demanding, as the user is asked to just name the object classes present in the image.

In this thesis, we propose a new type of weak supervision which is a generalization of image level supervision: **Volumetric Supervision**. In addition to providing the object classes present in the image, the user also provides the approximate size of each object class present in the images. This type of annotation is still very undemanding on the users time.

To incorporate volumetric information into weakly supervised segmentation, we develop two volumetric loss functions that penalize deviation from the object size annotated by the user. Almost any semantic segmentation method with image level weak supervision can be transformed into a segmentation method with volumetric supervision using volumetric loss functions. To show the usefulness of volumetric supervision, we chose four popular methods for image level weak supervision and transform them into volumetric supervision methods.

For evaluation, we create a simulated dataset that contains size information for the object classes. We also test the sensitivity of our approach to the possible mistakes in the size information dataset. Our experimental evaluation shows that volumetric supervision gives a significant improvement over image level supervision, however it is sensitive to mistakes in the size information provided by the user.

## Acknowledgements

I would like to thank everyone who made this thesis possible.

## Dedication

This is dedicated to everyone.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Image segmentation is one of the most fundamental topics in Computer Vision. The goal is to represent an image in a more meaningful and easier to analyze format. Image segmentation is the process of dividing an image into smaller regions or segments such that the pixels in each segment share similar properties like color, texture. Each segment, as shown in figure 1.1, is given a discrete label. The labels do not carry any special meaning, rather they are used to identify segment membership and boundaries. Segments with the same label have similar characteristics, while adjacent segments are significantly different with respect to those characteristics. Together, these segments cover the entire image.

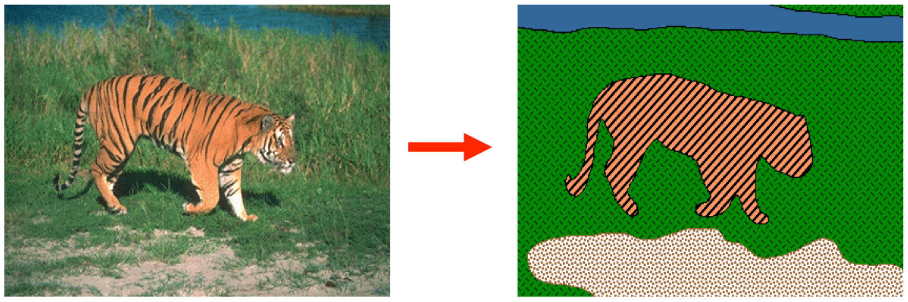


Figure 1.1: Example of image segmentation based on color. There are four distinct segments, each with its own discrete label [126]

By assigning a label to every pixel, image segmentation can perform deeper analysis of an image. Unlike other computer vision problems such as image classification and object detection, which produces a single classification label and an object bounding box, image segmentation can yield a pixel precise answer, revealing fine-grained information about image contents.

Earlier methods on image segmentation are usually based on local heuristics: threshold segmentation [29, 28, 63, 58], region segmentation [43, 119], edge segmentation [45, 106, 71, 130], texture features [62], clustering [131, 96] and so on [87]. Later methods for image segmentation tend to perform more global reasoning and/or rely on classification: graph theory [39], clustering [25, 114], classification and combination of clustering and classification [87].

**Semantic segmentation** is a type of image segmentation where labels have meaning: each label corresponds to a particular class out of a predefined set of classes of interest. Figure 1.2 shows the input and output of a semantic segmentation model. The model has detected three classes in the image: person, bicycle and background. The model has thus assigned each pixel with a label describing which class the pixel belongs to. Unlike general purpose image segmentation methods, most semantic segmentation methods require pixel precise ground truth for training.

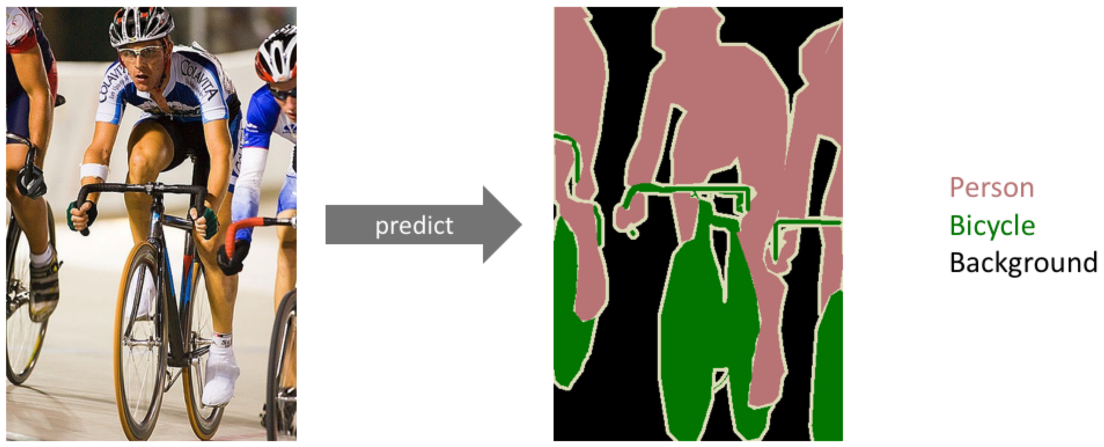


Figure 1.2: Input and output of semantic segmentation. The output has three labels: person, bicycle and background, each highlighted with a different color

The human visual system performs the semantic segmentation task seemingly without an effort, but it is a challenging problem for machine vision. Semantic segmentation is a field that has been researched for decades and hundreds of methods have been proposed. In 1972, David Walts developed an algorithm for generating semantic descriptions from drawings of scenes with shadows [128]. Later, Feldman developed a statistical method that could semantically analyze each region of specified images [38].

Since 2010, the rise of neural networks, and the development of deep learning, specifically Convolutional Neural Network (CNN), has radically changed how semantic segmentation is done [44]. CNN was initially proposed in the late 1980's by Yann LeCun. The network, called LeNet [66] was able to recognize and classify hand written digits. But due to a lack of sufficient amount of labeled data and computational power, it was not used widely. The popularity and importance of CNNs rose in 2012, when Krizhevsky et al. won the Imagenet large-scale visual recognition challenge [102] with their model AlexNet [60]. The success of CNN in classification led to its first use for semantic segmentation in 2014 by Liang-Chieh Chen et al. in their network DeepLab [18]. Since then CNN have become the gold standard for semantic segmentation. CNN's are further discussed in chapter 2.

Semantic segmentation has a wide range of applications. These include autonomous driving [14, 73], medical imaging and diagnosis [23, 74, 75], facial segmentation [16], handwriting detection and analysis [66], agriculture [77, 4], use cases in fashion [72, 73], video surveillance [124, 56], image editing [76, 80], and a lot more [88].

Training a neural network for semantic segmentation can either be fully supervised or weakly supervised. Both are further explored in the following sections.

## 1.1 Fully Supervised Semantic Segmentation

In the 21$^{st}$ century, semantic segmentation has primarily revolved around deep learning and artificial neural networks. Of these, Convolution Neural Networks (CNN) gained immense popularity in the task of segmentation. CNN's can extract features from a large number of images without hand engineering features. In order to train a fully supervised semantic segmentation network, we need to provide it with a dataset of images with pixel level annotation. This is where each pixel is labeled as part of a class from a set of predetermined classes. These are known as *ground truth.* Classes can range from a wide variety of objects. For example, the Pascal VOC 2012 segmentation dataset [35, 36, 34] has more than 10,000 images with 21 classes (20 class objects and 1 background). This dataset will be discussed further in later sections.

During training, a CNN finds features and representations of each class label. Once trained, the CNN is given a new image, and the neural network model identifies and labels which class each pixel belongs to. This process of training a CNN with images and their pixel level labels is known as *Fully Supervised Semantic Segmentation*. In recent years CNN models such as FCN [65, 70, 26, 123, 33], U-Net [33], Mask R-CNN [48, 33], ResNet [49, 50, 40, 122, 1] and DeepLab [19, 18, 73, 26, 56] achieved excellent results in fully supervised semantic segmentation.

## 1.2 Weakly Supervised Semantic Segmentation

While fully supervised semantic segmentation methods achieve excellent results, it is extremely difficult and time consuming to create datasets with pixel level annotations. Thus a lot of research is focused on creating semantic segmentation methods which do not require pixel level ground truth. This is known as *Weakly Supervised Semantic Segmentation* (WSSS).

In weakly supervised semantic segmentation, pixel level annotations are not present. Instead other types of information about the image and the objects present in them is provided. There are many different types of weakly supervised semantic segmentation. Of these, the most common is image level annotation [6, 92, 93, 59, 53, 3, 67, 37, 115]. For each image, we are only given the object classes that are present in each image. Figure 1.3 (a) shows an example of image level annotation. Most of the methods that use such image level weak supervision are based on Class Activation Maps (CAMs) [137] to generate pseudo ground truth labels that are in turn used to train a CNN. CAMs is a technique that highlights pixels of an image that are associated with a particular class. CAMs identify which regions in the image are relevant to detect a particular class. Figure 1.4 shows an input image, its image level annotation, pixel level ground truth and its CAM highlighting the different regions where the class is likely to be present. CAMs are further discussed in chapter 2.

There are other types of weak supervision such as bounding boxes [90, 132, 27, 57, 68, 112, 61] where one specifies a bounding box around each object, and image scribbles [132, 69, 125, 120, 121] where only some image pixels are annotated with their object class. Figure 1.3 (b) and (c) shows examples of bounding box and image scribble annotations respectively. Bounding boxes and scribbles require much more time to annotate, compared to image level labels.

## 1.3 Contributions

The summary of the contributions of this thesis are listed as follows:

1. We propose a new type of weak supervision: **Volumetric Supervision**.This is a type of weak supervision that generalizes previously proposed image level weak supervision. In addition to specifying which object classes are present in an image, the user also specifies rough size of each object class, relative to the image size. We do not expect the user to provide an accurate size estimate, but rather estimate the size bucket each object class belongs to, where "bucket" is quantization of all possible sizes. We use a coarse quantization with 10 evenly spaced buckets. Figure 1.5 shows an example of annotation for Volumetric Supervision for an input image. It also shows the ground truth, the results of a method [133] with image level supervision, and the results of our transformation of that method into volumetric supervision method using quadratic volumetric loss. As shown in the image, Volumetric Supervision annotation consists of image level labels and size of each class present in the images. The image is from the Pascal VOC 2012 dataset [36, 35, 34] and the classes present are background, bicycle, and person, with background and person each occupying roughly 40% of the image, and bicycle occupying roughly 20% of the image.

   When the user annotates the image with the classes present in it, they already have an approximate idea of the area occupied by the class, as they do a visual inspection of an image. Thus such additional information should be easy to provide. This additional volumetric information is helpful for weakly supervised setting, as CNN class estimates that are too large or too small in size can be penalized, providing an additional guidance for training and leading to more accurate results. As seen in figure 1.5, volumetric supervision helps to recover objects more accurately.

   The name "volumetric" instead of "size" is used for historical reasons. One of the first works to use constraints on size was [127], who used size constraints on 3D reconstructed volumes, therefore, the name "volumetric" is popular for size constraints.

2. In order to implement Volumetric Supervision, we propose two new volumetric loss functions based on the approximate size of each object class present in the images. These loss functions are named Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss respectively.

   (a) The first loss function,which we call **Quadratic Volumetric Loss** is based on Mean Square Error. For each pixel in the image, the network outputs a distribution over the classes it belongs to. We take this probability as an estimate of the class membership. For each class, we average the corresponding estimated probabilities over all image pixels, and this gives us an approximation for the size of the class. Then we take the square of the difference between the estimated size and the size of the bucket this class falls into (given by the user annotation).

    (b) For the second loss function, which we call **Outside Quadratic Volumetric Loss**, the idea is to penalize the network when its prediction is not within two end points of the bucket of the corresponding class. The end points are denoted as $a$ and $b$, where $b$ is greater than $a$. As for the Quadratic Volumetric loss, we first compute an approximation of the size of the class, derived from class probabilities for each pixel. Note that Quadratic Volumetric loss penalizes any class size which is not strictly equal to the size of the corresponding bucket. Since the user specifies only a course set of object buckets, we want to make the penalty less strict, i.e., zero when the estimated size falls inside its bucket, and increasing when the estimated object size falls outside its bucket. For object size estimates that fall outside their bucket, the penalty is quadratically increasing, depending on the distance to the bucket. Thus we call this loss Outside Quadratic loss, since it is zero inside the appropriate bucket, and quadratically increasing when the estimate size moves away from the bucket.

   These volumetric loss functions are further detailed in chapter 3.

3. We show how to incorporate our volumetric loss functions into four previously developed methods for weakly supervised semantic segmentation with image level labels [133, 6, 17, 22]. These approaches are described in more details in chapter 4. This converts these methods from weak supervision with image level labels into volumetric weak supervision methods.

4. We create a new simulated dataset containing size information for the classes for each image from a popular semantic segmentation benchmark [36, 35, 34]. The dataset is based on size buckets. We quantize possible object sizes into 10 buckets, and place each object class appropriately.

5. We evaluate the performance of the new volumetric supervision that was incorporated into the four methods [133, 6, 17, 22], mentioned in step 2. First we run these methods as is with no modifications. Then we incorporate the two loss functions described above into the methods and compare the results. Different experiments with different parameters are conducted, all detailed in chapter 5. The results from

the different experiments are compared to the original unmodified results. More detailed visualizations are shown in chapter 5.

6. We test if we can minimize the user annotation efforts by adaptively estimating class sizes without having the user specify them manually. We assume that the background occupies 50% of the image, and split the foreground evenly among all object classes present in the image. This is our initial size estimates, which we change adaptively during training of the algorithm, depending on whether the sizes of classes increase or decrease, compared to their initial size.

7. Finally we test the sensitivity of our approach. This is done by introducing errors into the volumetric datasets to emulate human error that might occur while creating the datasets. To do so, we take the bucket dataset and randomly place up to 20% of objects in incorrect buckets. The experiments conducted are outlined in chapter 5, where these results are compared to the unmodified results of the four recent image level weak supervision approaches [133, 6, 17, 22].

## 1.4 Thesis Organization

Chapter 2 introduces basic machine learning concepts, their applications in semantic segmentation and related work in both fully and weakly supervised semantic segmentation. In chapter 3, we introduce the two newly proposed volumetric loss functions, namely **Quadratic Loss** and **Outside Quadratic Loss**. In chapter 4, we introduce the four recently published weakly supervised semantic segmentation methods [133, 6, 17, 22] and describe how we modify each for volumetric supervision. In chapter 5, we evaluate our volumetric loss supervision using the four methods discussed in chapter 4. First we create an artificial dataset with volumetric annotations. Then we discuss our main evaluation measure, mean over intersection (mIOU). Next we discuss and analyze the results for Quadratic loss and Outside Quadratic loss. Finally, we summarize and conclude our work in this thesis in chapter 6.

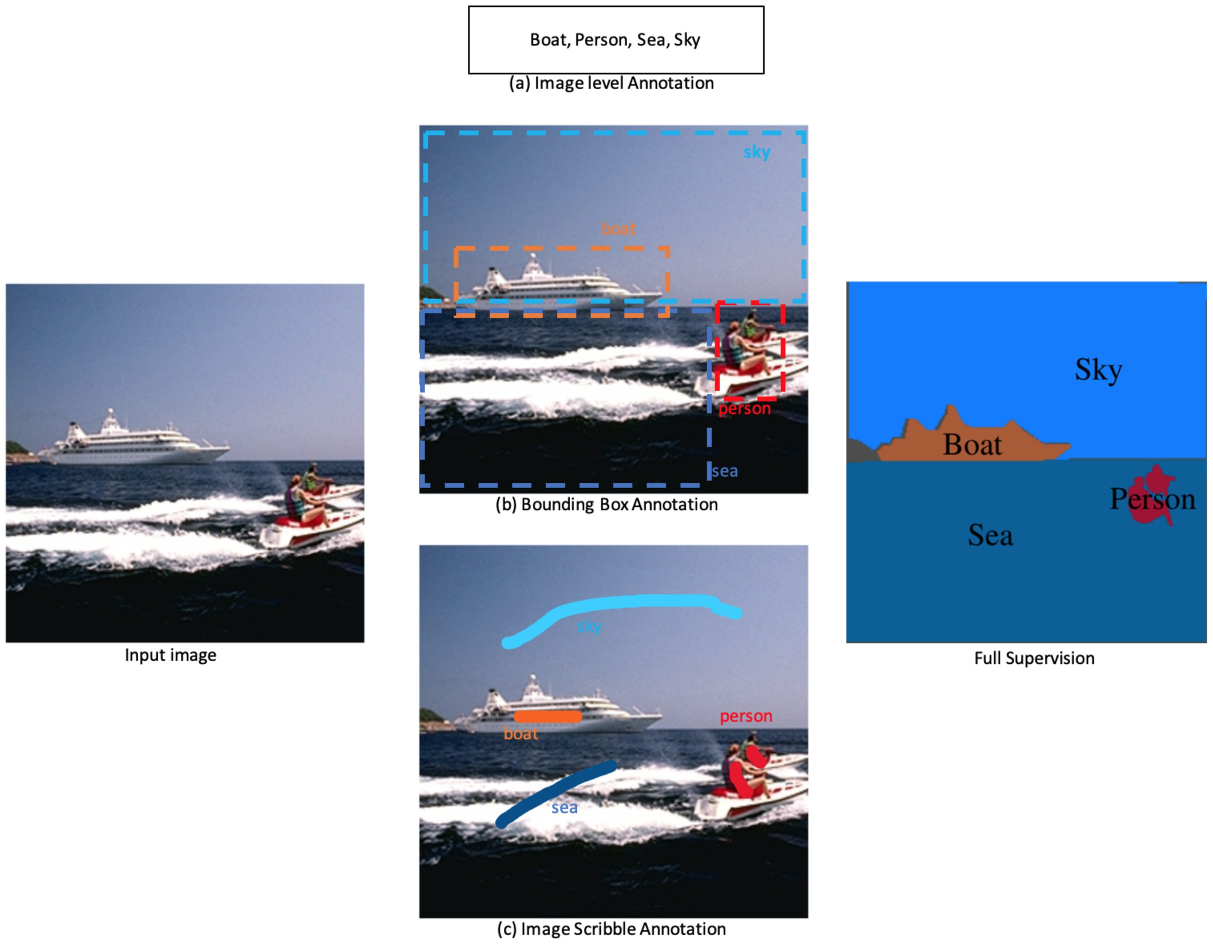


Figure 1.3: Different types of weak supervision for semantic segmentation for an input image. (a) is image level annotation. (b) is bounding box annotation. (c) is image scribble annotation. The image on the right is full supervision

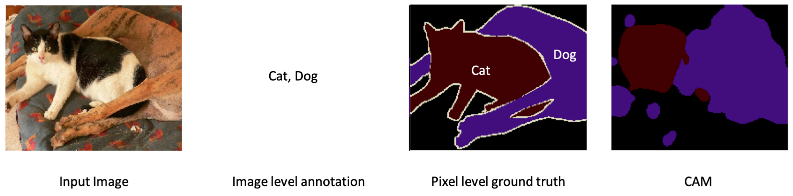


Figure 1.4: Input image, image level annotation, pixel level ground truth and the CAMs that show where the different class objects are located

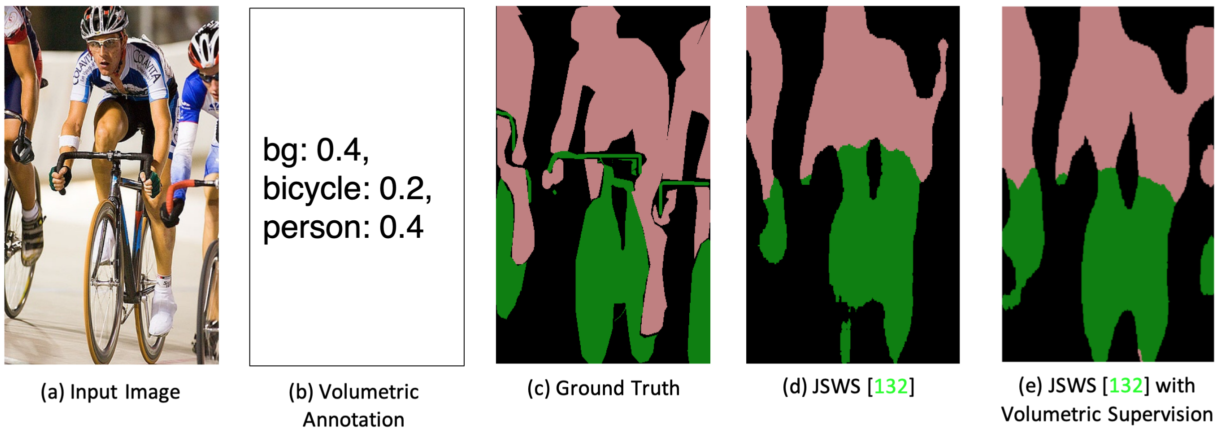


Figure 1.5: (a) Input image from the Pascal VOC dataset [36, 35, 34]. (b) Volumetric Supervision annotation. Volumetric Supervision consists of naming the object classes present in the image, together with a rough size of each object class relative to the image size. In this example, only three classes, background (bg), bicycle, and person are present out of 21 possible classes in the Pascal dataset. (c) Pixel level ground truth. (d) Output from Joint Learning of Saliency Detection and Weakly Supervised Semantic Segmentation (JSWS) [133] without any modification. (e) Output from JSWS [133] with Volumetric Supervision prediction

# Chapter 2

# Related Work

Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL) have become immensely popular in all circles in recent years. ML is a subset of artificial intelligence that uses computer science and statistics, and DL is a further subset of ML. The goal is for a machine to learn from past information without specific coding. Given a data set, data is fed as input, and the machine “learns” from the data using analysis methods and outputs predictions. In this thesis we tackle semantic segmentation as a ML and DL task.

The three main types of machine learning are: Supervised, Unsupervised and Reinforcement Learning. Supervised learning algorithms take labeled data as input. Unsupervised is the opposite, where the data do not have any labels. Reinforcement learning has an agent making a series of decisions in an environment, and is rewarded on the outcome of the actions. The goal is to maximize rewards. This thesis uses labeled data, thus the focus is on supervised learning.

Supervised learning algorithms can be applied to either classification or regression problems. *Classification* is predicting discrete labels, whereas *Regression* is for continuous data, for example predicting temperature. Semantic segmentation is a classification problem, since we need to predict discrete class labels for each pixel. Supervised learning has three different settings: Fully Supervised, Semi Supervised, Weakly Supervised. As mentioned previously, fully supervised has data inputs where each entry in the dataset has the correct label. Weakly supervised has noisy, and sometimes incorrect labels. Semi supervised is a mixture of the two. As mentioned above, the importance of weakly supervised semantic segmentation lies in the ease of creating a training dataset, compared to its fully supervised counterpart. In the following sections, we establish key machine learning fundamentals.

## 2.1 Fundamentals

The most important components of a neural network (NN) are the layers and neurons. Layers are vertically stacked components in a NN. Figure 2.1 shows a NN with different layers. The first layer is the input layer, where the data is passed into the next layer, the hidden layer. The hidden layers are where all the computation takes place. In figure 2.1, there are three hidden layers, but NN can have any number of hidden layers. The final layer is where all the information learned from the hidden layers are outputted, thus naming it the "output layer".

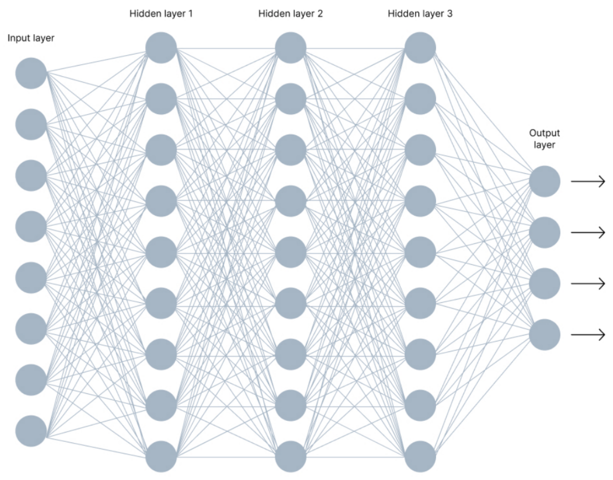


Figure 2.1: Different layers in a Neural Network [11]

Each layer consists of neurons. A neuron in a NN is similar to the neurons in the brain [2]. A neuron is made up of inputs, weights, biases and an activation function, all of which combine to produce an output. Figure 2.2 shows all the components of a neuron in detail. The inputs are $x_1$ and $x_2$, with their respective weights $w_1$ and $w_2$. The bias is $b$ with its weight $w_b$. The inputs and bias are multiplied by their respective weights, and then summed. The summation is passed into the activation function $f(x)$, which produces the output. In mathematical term, $y = f(x)$, where $x = x_1 * w_1 + x_2 * w_2 + b * w_b$ [2]. A neural network "learns" by modifying these weights.

Neural networks have two essential directions of movement, Feed Forward and Back Propagation [11]. In *feed forward*, the inputs and bias are multiplied by their respective weights, the result is fed into the activation function, which produces the output. The output is then sent to the next layer. In *back propagation*, the weights are repeatedly adjusted to minimize the difference between the actual network output and the desired output, also known as minimizing the cost function. Cost function gradients determine the level of adjustment to apply to parameters like activation function, weights and bias.

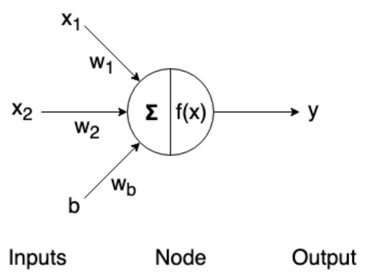


Figure 2.2: Different components of a neuron [11]

The activation function decides if a neuron should be activated and adds non-linearity to a neural network, and allows the network to learn complex tasks. There are three main types of activation function: Binary, Linear and Non linear [2]. A binary activation function activates a neuron if a certain threshold value is met. A binary activation function can be represented as:

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

A linear activation function is where the output is simply the sum of the product of the inputs and weights. It is represented as:

$$f(x) = x$$

The issue with binary and linear activation functions is that their gradient is constant. Back propagation requires non-constant gradient, and thus the model cannot learn to adjust weights. Non-linear activation functions solve this problem. A non-linear activation function allows back propagation and enables us to create complex combinations by stacking multiple layers. Stacking these non linear units together gives us a fully connected layer, as known as a *multi-layer perceptron* (MLP). Some non-linear activation functions and their equations are:

1. Sigmoid / Logistic function [9, 2]. Not used in Hidden layers. Used primarily for binary or multilabel classification

$$f(x) = \frac{1}{1 + e^{-x}}$$

2. Hyperbolic Tangent function [85]. Not used in Hidden layers

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

3. Rectified Linear Unit (ReLU) function [85, 41]. Only used in hidden layers. Figure 2.3 shows the graph of the ReLU function

$$f(x) = max(0, x)$$

4. Exponential Linear Units (ELUs) function [85]

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases}$$

5. Softmax function [85]. Used primarily for multiclass classification

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

6. Swish function [85]

$$f(x) = x * sigmoid(x) = \frac{x}{1 + e^{-x}}$$

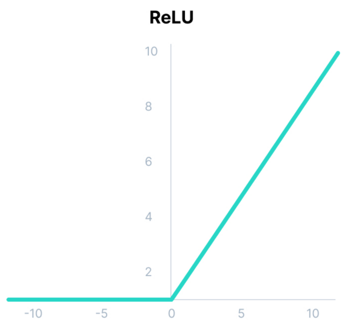


Figure 2.3: ReLU function

ReLU functions [41] are primarily used with CNNs, which is the focus of this thesis. With a basic explanation of the components of a neural network, we now explore how it functions, i.e., how does a NN learn from data. But first, we need to introduce two important components that a NN needs to learn how well it performed before back propagating: loss function and optimizer.

*Loss functions* tell a NN how well it is performing. A loss function quantifies the difference between the actual output in the dataset and the prediction that the NN outputs. In other words, it evaluated how well the NN models the given data. There are two main categories of loss functions, classification and regression [91]. Some common loss functions for classification are:

- Hinge loss [54]
- Cross Entropy Loss [81]

Some common regression loss functions are :

- Mean Absolute Error also known as L1 Loss [54]
- Mean Square Error also known as L2 Loss [54]
- Mean Bias Error [91]

The main contribution of this thesis are two new loss functions based on the size of objects in an image as discussed in chapter 4.

*Optimizers* work with the loss function to modify different parameters of a NN, such as weights, activation functions, learning rate and more. This optimization is done in conjunction with the loss function to minimize the latter as much as possible. Some common optimizers are [101]:

- Gradient Descent
- Stochastic Gradient Descent
- Mini-Batch Gradient Descent
- Adam
- AdaDelta
- Adagrad (Adaptive Gradient Descent)

- RMSProp (Root Mean Square)
- AdaMax
- Nadam

Before a ML algorithm starts, the weights of the NN are randomly set. The data set consists of features, and the expected output, or labels. Training the NN implies modifying the weights such that it can represent a correlation between the features and the label. A **sample** is a single row in a dataset, or for our purposes, it is an image along with its labels. For each iteration, we take a **batch** of samples from the training data and send them to the NN. The NN outputs predictions, which are then processed by the loss function. The loss functions measures the difference between the predictions and the true labels. The difference is passed to the optimizer, which uses those differences to adjust the weights so that the next iteration the loss will be smaller.

An **epoch** is the number of times the NN sees the entire dataset. For each epoch, we run the above steps a number of times based on the batch size. Optimizers specify the direction the weights need to go, as well as how big of a change is needed to decrease the loss. Figure 2.4 shows a graph of how optimizers work to reduce loss. We start with our initial weights, and the gradient vector from the optimizers represent the direction and speed of the weights such that they reach a global minimum and that loss is minimal. Optimizers can be configured to keep running until the loss is no longer decreasing (reached a global minimum as shown in figure 2.4), or can be manually set to halt after a number of epochs. All this information is conveyed back to the NN or back propagated to the NN so that it can rerun the model on the updated weights.

To summarize, the goal is to utilize input data, the loss functions and optimizers to train a NN. In weakly supervised semantic segmentation, the input data is the images, and their image level labels, as well as other information, which will be discussed in later sections. In computer vision, a special kind of neural network is used. This is known as Convolutional Neural Network.

## 2.2 Convolutional Neural Network

A *Convolutional Neural Network* (CNN) is a form of Artificial Neural Network (ANN), that has been specifically designed to work with images. CNNs were initially proposed in the late 1980's by LeCun et al., who called it "LeNet" [66]. This NN was able to recognize

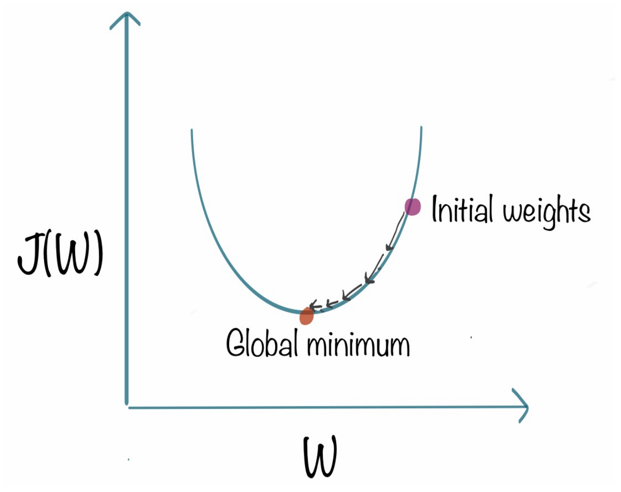


Figure 2.4: Graph representing how optimizers work to change weights such that loss is minimal. The gradient vector initially suggests bigger changes to the weights, but as they reach global minimum, the changes are smaller

and classify hand written digits. But due to a lack of image data and computational power, LeNet did not garner any interest from the research community. The popularity and importance of CNN rose in 2012, when Krizhevsky et al. won the Imagenet large-scale visual recognition challenge [102] with their model AlexNet [60]. This network classified images from ImageNet [30] with 1000 predefined classes. AlexNet would pave the path to future CNN model architectures.

A fully connected NN as discussed in the previous section can be used for small image classification, for example if we take a image of a hand written number of size 7 x 7 pixel from the MNIST dataset [31]. Assume the network has one hidden layer. The input layer will have 49 ($7 \times 7$) nodes, the hidden layer will have the same, and the output layer will have 10 nodes, for ten digit classification. The number of weights in the NN is 2891. Figure 2.5 displays such a NN and all its connections. But for larger images, such as an image of size $1920 \times 1080 \times 3$, there would be over 6 million weights in the first layer itself. Weights between inputs and the hidden layers would be over 24 million. This is excessive level of computation and parameters, which would lead to over fitting. Other disadvantages of ANN are it treats local pixels similarly to pixels far apart. And ANNs are sensitive to the location of objects in an image. These problems are solved by using a CNN. CNNs can represent images in a different form and preserve the features that are important for the model to learn.

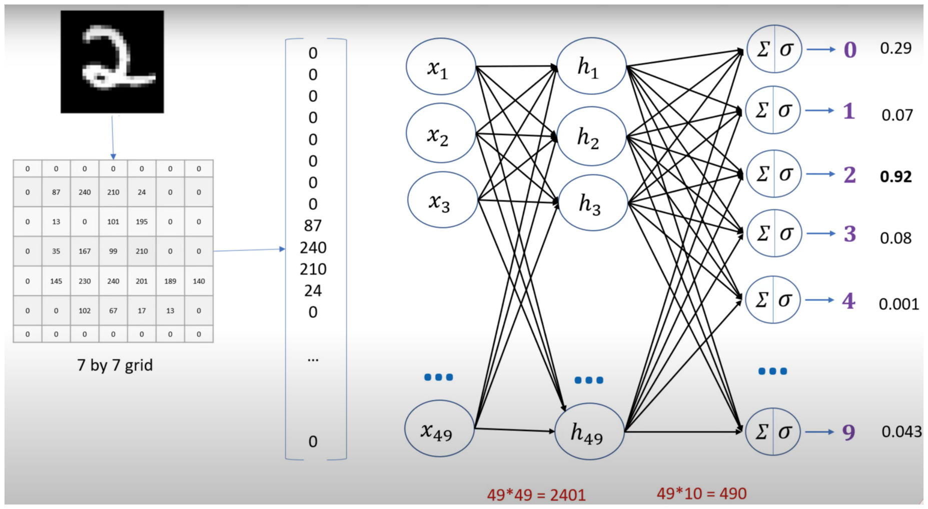


Figure 2.5: Fully connected NN with one hidden layer for classifying hand written 7 by 7 pixel image numbers

In CNNs, generally there are three main layers: input, hidden and classification layers [79]. The input layer contains the input image. The hidden layer contains the Convolution layer with an activation function (ReLU is most commonly used) and pooling layers, which reduce the output size. The classification layer is a fully connected layer where the image is classified. Figure 2.6 shows a simple architecture of a CNN used for image classification.

### Convolution Layer

The convolution layer is the main layer of a CNN. The input to a convolution layer is the image. Most images are RGB, and have three layers or depth (one for each color). Thus the input size is $h * w * 3$. In addition, each neuron has access to a partial number of neurons from the previous layer, unlike a fully connected NN.

A convolution layer consists of filters or kernels [103]. These filters are used to detect patterns or features in an image [55], for example shapes or edges. Commonly the first layers capture low level features such as edges, colors and such, whereas the deeper layers focus on higher level features. The filters are smaller than the image, but have the same depth as the image [12]. The convolution layer performs a convolution operation between the filter and a restrictive portion (spatial position) of the input of the same height and width as the filter [55].

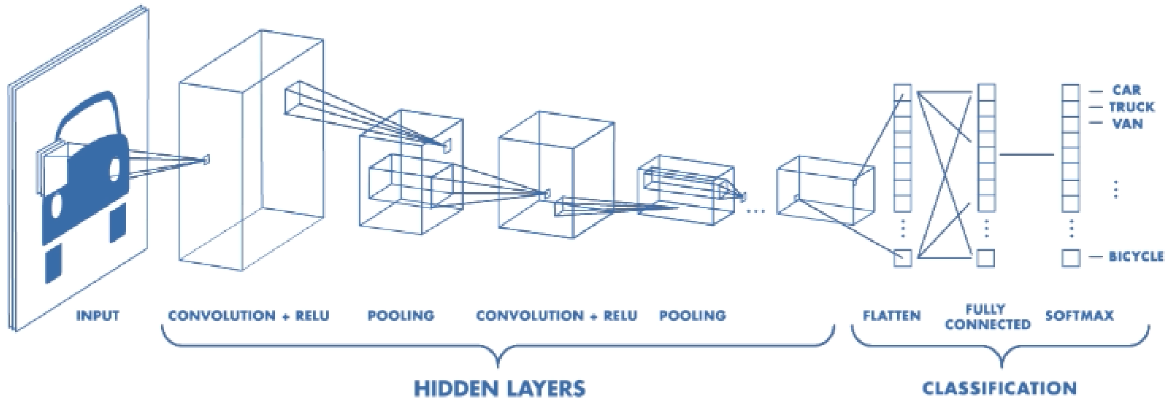


Figure 2.6: Simple architecture of a CNN used for image classification

A convolution operation is a dot product between the filter and the restrictive portion of the input. The filter starts at the top left of an image, where it performs a convolution operation and produces a single value as output. The filter then moves one step, or stride to the right and repeats [103]. The filters stride can vary from architecture to architecture. This process produces a 2D activation map of the input. Figure 2.7 shows this operation with an example. The input is 4 x 4. The filter is 2 x 2. The filter starts at the top left and moves with a stride of 1. At each spatial position, it performs a dot product. The result is stored in the 3 x 3 output shown below, as well as the values in each of the individual positions of the output 2D activation map with respect to the input and filter.

Figure 2.8 shows a convolution operation with an input of depth three.The filter has dimensions of $3 \times 3 \times 3$. And finally there is a bias of 1. Bias is an extra parameter used to adjust the output. Matrix multiplication (dot product) is performed between the input channel and the corresponding filter (or kernel), and the three products are summed up with the bias. Mathematically: $IC_1 \cdot KC_1 + IC_2 \cdot KC_2 + IC_3 \cdot KC_3 + bias$ to get the value in the corresponding output position [103].

The resulting operation reduces the dimensionality compared to the input. If we want to increase the dimensionality, we can use a method called **Padding** [103]. Padding is surrounding each layer of the input to increase its dimensions. In figure 2.8, the inputs are surround by a layer of zeros. This is known as Zero Padding.

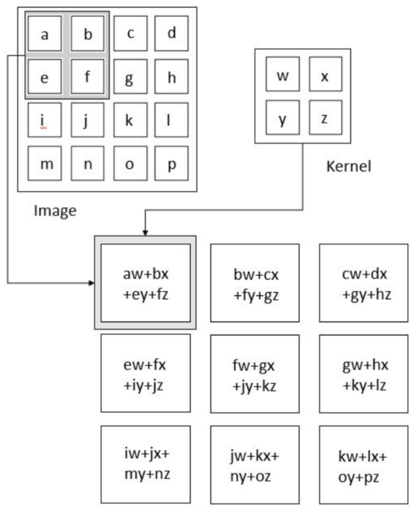


Figure 2.7: Convolution operation with a stride of one

The parameters in the convectional layer that can be controlled are the number of filters, the filter size, the filter stride and the padding width. The number of filters will determine the depth of the output or output channel. If the stride is one, then the filter moves one step to the right and then one step down. If the stride is two, then it moves two steps to the right and then two steps down. This will produce a smaller output spatially compared to a stride of 1. The output size is calculated as follows: If the input has a shape of $W \times W \times D$, with a padding of $P$, and the filter has a size $F \times F \times D$ and a stride of $S$, then the output size is $W_{out} \times W_{out} \times D$, where $W_{out}$ is:

$$W_{out} = \frac{W - F + 2P}{S} + 1[79]$$

## Activation Layer

The second layer is the activation layer, where an activation function is applied to the activation map [12]. ReLU is the most common activation function used. As mentioned earlier, this increases non linearity in the CNN [12]. Each filter is responsible for a certain detection. For example, a filter that detects loops in an image [109]. The activation map will have positive values for pixels that are part of the loop, and negative values for pixels that do not. ReLU activation will replace all the negative values with zero, and leave the positive values as is. Figure 2.9 shows the inputs and outputs of such an example.

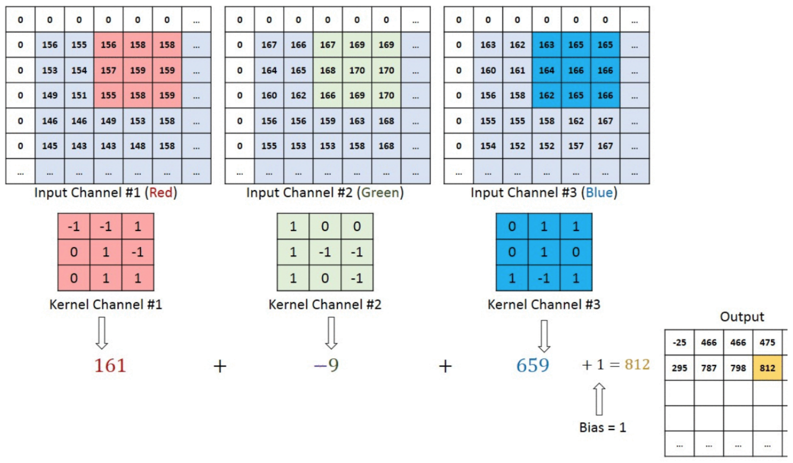


Figure 2.8: Convolution operation on a M x N x 3 image matrix with a 3 x 3 x 3 filter and padding of 1 [79]

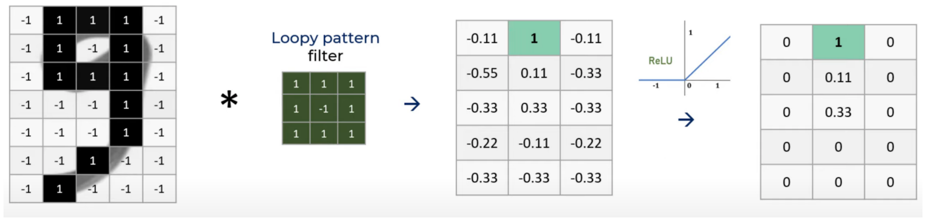


Figure 2.9: ReLU function to detect loops in images

## Pooling Layer

Up till this point, the output dimensions have not reduced significantly, or not reduced at all if it was padded. Dimension reduction is done in the pooling layer. The pooling layer is used to reduce the output size and involves downsampling of features, thus reducing the computation required [103]. Pooling is also useful for extracting dominant features, regardless of object location or rotation in an image [103]. Another benefits of pooling is it reduces overfitting since there are fewer parameters. Pooling is applied to every layer of the output channel. The pooling layer also has parameters such as filter size and stride [12]. But instead of doing a convolution operation, this layer performs a pooling operation. The output size is calculated as follows: If the activation map has a shape of $W \times W \times D$, and the pooling layer filter has a size $F \times F \times D$ and a stride of $S$, then the output size is

$W_{out} \times W_{out} \times D$, where $W_{out}$ is:

$$W_{out} = \frac{W - F}{S} + 1[79]$$

There are two types of pooling: max and average [103]. Suppose we have an activation map of $8 \times 8$ and a kernel of of $2 \times 2$ as shown in figure 2.10. The kernel starts on the top left and moves with the stride value. At each step, the kernel performs either a max pool or average pool operation. For max pool, the kernel takes the max value of the portion of input covered by the Kernel. For average pool, the kernel takes the average of all those values.

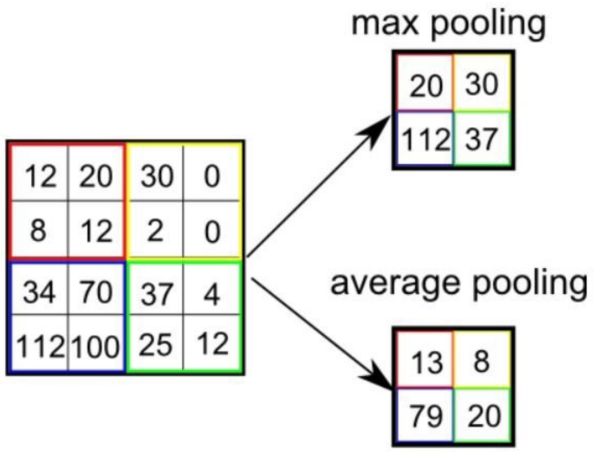


Figure 2.10: Max and average pooling on an input of size $8 * 8$ with a kernel of size $2 * 2$ [103]

Average pooling reduction does dimensionality reduction as a noise suppressing mechanism [103]. Max pooling acts as a noise Suppressant by discarding the noisy activations by performing denoising alongside dimensionality reduction [103]. Thus in most cases max pooling out performs average pooling [103]. Figure 2.11 shows the entire hidden layer with the convolution layer, the ReLU and finally a max pooling with a kernel of size 2 with stride 1.

### Classification Layer

The final layer is a fully connected dense classification layer. This layer helps representation between the input and the output [79]. Figure 2.12 shows a CNN architecture with an

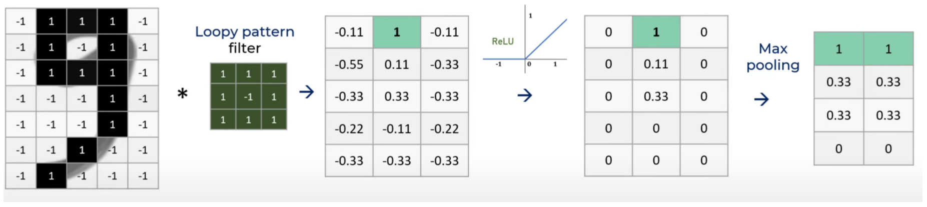


Figure 2.11: All the steps in a hidden layer of a CNN

emphasis on the classification layer. The hidden layer with the convolution, ReLU and pooling layers extract features from the image. These features are flattened into a column vector before being sent to the classification layer [103]. In figure 2.12 the output of max pool is 14 × 14 × 3, and this three-dimensional image becomes a column vector of size 588 after being flattened. The classification layer is a feed-forward neural network and backpropagation applied to every iteration [103]. Over a series of epochs (going over every image in the data set once), the model is able to distinguish features in images and classify them using Softmax or Sigmoid [12]. The final output layer has dimensions $1 * 1 * C$, where $C$ is the number of class objects.

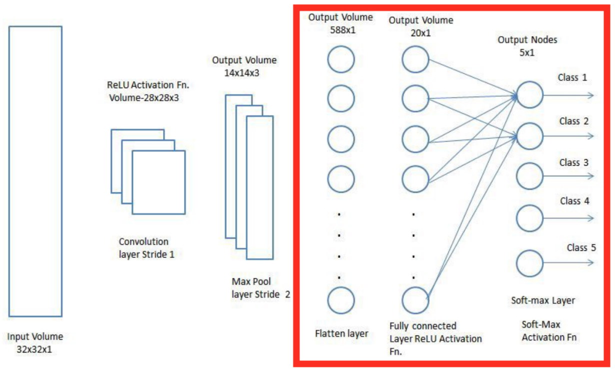


Figure 2.12: CNN architecture with emphasis on the fully connected classification layer (in red) [103]

### Benefits of CNNs

CNNs are designed and can extract spatial and temporal information from images. Benefits of CNNs include:

- Connections sparsity reduces over fitting

- Convolution and pooling allows for feature detection regardless of location or orientation
- Parameters can be shared. When a CNN learns the parameters for a filter, this filter can be applied to the entire image
- Using ReLU introduces non linearity, and it speeds up training
- Having a pooling layer decreases dimensionality and reduces computation
- Pooling also reduces overfitting and makes the model tolerant towards small distortions and variations

### 2.2.1 Famous Convolutional Neural Network Models

As mentioned earlier, LeNet was the first CNN proposed in 1989 by LeCun to recognize hand written numbers [66], but CNNs were made popular in 2012 when Krizhevsky created AlexNet [60] and won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). ImageNet is a dataset consisting of more than 14 million images belonging to 1000 classes [30]. Since then numerous CNN architectures for image classification task have been developed. Here we will briefly talk about some of the famous CNNs in computer vision.

#### Vanishing Gradient Problem

Before exploring famous CNN architectures, we should address **Vanishing Gradient Problem**. As more layers are added to a NN, the deeper the network gets. And more layers mean more activation functions. Certain activations functions such as sigmoid transforms a larger input to a smaller input space [129]. For shallow networks it is not a problem. But for deeper networks, the gradient (derivative) of sigmoid becomes small in saturated regions. The gradients are found by back propagation [129]. However, when $n$ hidden layers use an sigmoid activation function, $n$ small derivatives are multiplied together, resulting in small gradients [129]. Small gradient means weights and biases are not effectively updated. This is known as the **Vanishing Gradient Problem**.

#### AlexNet

As mentioned previously, AlexNet was introduced in 2012 by Krizhevsky. It won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). They scored Top-1 and

Top-5 error rates of 37.5% and 17.0% respectively [60, 64], which outperformed other approaches. AlexNet is a eight layer CNN, with 5 convolution layers and 3 dense layers [64]. Figure 2.13 shows the full architecture of AlexNet. The first layer is the input. The next five layers are the convolutional layers with max pooling. The first two convolution layers are connected to overlapping max-pooling to extract maximum features [64]. And the final three layers are the fully connected dense classification layers.

The network takes in an RGB image of the shape $247 \times 247 \times 3$. This image is passed to the first convolution layer. Conv 1 has 96 filters of shape $11 \times 11 \times 3$, with a stride of 4 [104]. Filtering with conv 1 creates a feature map of shape $55 \times 55 \times 96$ [104], which is passed through ReLU activation function to get the activation map. The activation map is passed to a max pooling layer with a kernel shape of $3 \times 3 \times 3$ and stride of 2 [104]. This pooling layer produces an output of shape $27 \times 27 \times 96$ from the first layer [104].

Conv 2 is the next layer. This convolution layer has 256 filters of shape $5 \times 5 \times 3$ with a stride of 1 and padding of 2 [104]. Filtering with conv 2 creates a feature map of shape $27 \times 27 \times 256$ [104], which is passed through ReLU activation function to get the activation map. The activation map is passed to a max pooling layer with a kernel shape of $3 \times 3 \times 3$ and stride of 2 [104]. This pooling produces an output of shape $13 \times 13 \times 256$ from the second layer [104].

The output of the last layer is passed to Conv 3. It has 384 filters of shape $3 \times 3 \times 3$ with a stride of 1 and padding of 1 [104]. Filtering with conv 3 creates a feature map of shape $13 \times 13 \times 384$ [104], which is passed through ReLU activation function to get the activation map. There is no pooling.

The output of the previous layer is passed to Conv 4, which is the same as Conv 3. Filtering with conv 4 creates a feature map of shape $13 \times 13 \times 384$ [104], which is passed through ReLU activation function to get the activation map. There is once again no pooling.

Conv 5 is the final convolutional layer and it has 256 filters of shape $3 \times 3 \times 3$ with a stride of 1 and padding of 1 [104]. Filtering with conv 5 creates a feature map of shape $13 \times 13 \times 256$ [104], which is passed through ReLU activation function to get the activation map. The activation map is passed to a max pooling layer with a kernel shape of $3 \times 3 \times 3$ and stride of 2 [104]. This pooling produces an output of shape $6 \times 6 \times 256$ from the final layer [104].

Then comes the first dropout layer with $p = 0.5$ [104]. The output from this layer is also $6 \times 6 \times 256$ [104], which is passed to the fully connected layers.

The first fully connected layer has 4096 neurons and uses the ReLu activation function [104]. Then comes another dropout layer with $p = 0.5$ [104]. This dropout is followed by the second layer which also has 4096 neurons and ReLU activation function [104]. The output of the second fully connected layer is connected to the final layer with 1000 neurons, since there are 1000 classes [104]. This layer uses the softmax function to make the final predictions.

This model has a total of 62.3 million parameters [104].

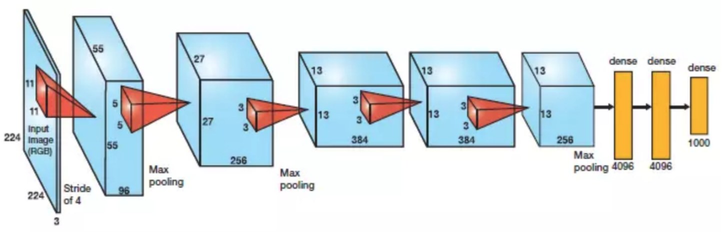


Figure 2.13: AlexNet Architecture [1]

The key take away from this architecture was the use of ReLU over Sigmoid. The advantages of ReLU over sigmoid is that it trains faster by avoiding the vanishing gradient problem [1].

Another key advantage AlexNet has was the use of **Dropout** layers [1]. Dropout layers applies a probability $p$ to each individual neuron in the activation map separately, and randomly switches off the activation with probability $p$. This is illustrated in figure 2.14

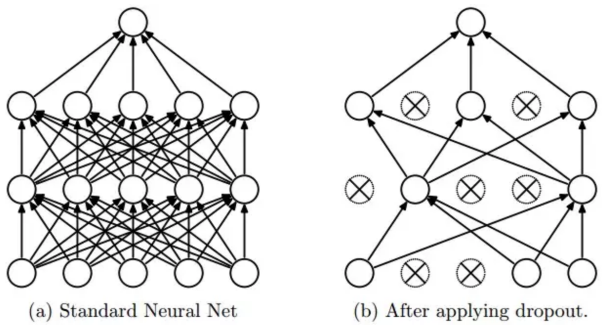


Figure 2.14: Standard Neural Network with and without Dropout Layer [1]

## VGG

VGG stands for Visual Geometry Group. This method was proposed in 2014 by Zisserman and Simonyan. There are two model architectures: VGG 16 and VGG 19 [100]. The numbers 16 and 19 stand for the number of layers respectively [13]. The VGG16 model achieves almost 92.7% Top-5 test accuracy in ImageNet [47, 13]. VGG made significant improvements over AlexNet by replacing the larger sized filters with several $3 \times 3$ sized continuous filters [1].

Figure 2.15 and 2.16 shows the architecture and filter/pool shapes for each layer of VGG 16 respectively. There is the input layer, a RGB image of shape $224 \times 224 \times 3$. This layer is followed by 13 conv layers [111, 64]. The unique thing about VGG is that each filter has a stride of 1, and each max pool has a stride of 2, and all the layers have the same padding [13]. Thus after a convolution layer, the height and width always remains the same as shown by the black layers in figure 2.15 , only the depth changes based on the number of filters in each layer, which is shown in figure 2.16. The height and width are only cut in half whenever a max pooling layer is reached, as shown by the red layers in figure 2.15.

After the final convolution layer, the output is passed to a fully connected dense NN. This network follows the same structure as AlexNet, with 4096 neurons in the first two layers and 1000 neurons, one for each class in the final layer with a sigmoid function [13].

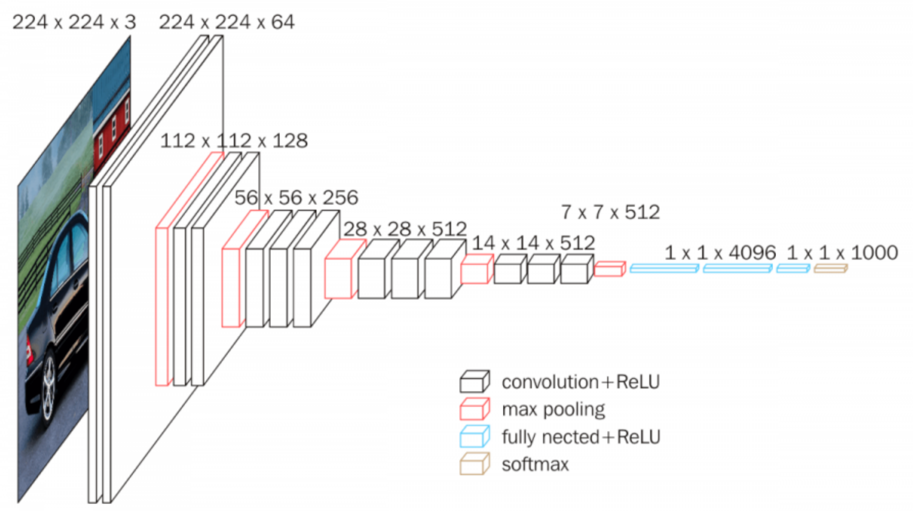


Figure 2.15: Architecture and output shape of each layer of VGG 16 [47]

Figure 2.17 shows the difference between VGG 16 and VGG 19. There are three more convolution layers for VGG 19, which can be found in conv layers 4 and 5.

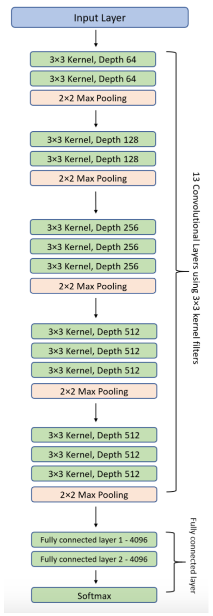


Figure 2.16: Filter and max pool size of each layer of VGG 16 [13]

### GoogleNet/Inception

While VGG has exceptional performance, it is a computational heavy architecture. For example a convolution layer with a $3 \times 3$ filter with an input and output of 512 channels, has $9 \times 512 \times 512 = 2359296$ operations [1]. However most of the activations in a deep network are redundant because they have a value of zero or do not have any correlations between them [1]. Thus there are sparse connections between the activations, which implies that all 512 input and output channels will not be connected to each other. GoogleNet created a module called Inception that approximates a sparse CNN [117]. The idea behind Inception is to have multiple size filters on the same level. This arrangement will make the network wider rather than deeper.

Figure 2.18 shows two versions of the inception module. The figure on the left is the naive version, with three different size filters ($1 \times 1, 3 \times 3, 5 \times 5$). The native version performs convolution on its input, with these three filters, alongside max pooling [117]. The outputs are concatenated and sent to the next layer [94]. To make it further cheaper, computationally, the input channels are limited by adding an extra $1 \times 1$ convolution layer before the multi size filters, which is shown in the right image of figure 2.18 [94].

The Inception module is the core of GoogleNet. Figure 2.19 shows it has 22 layers (27

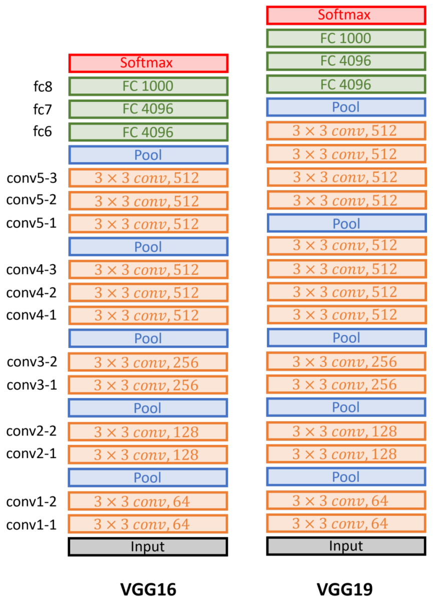
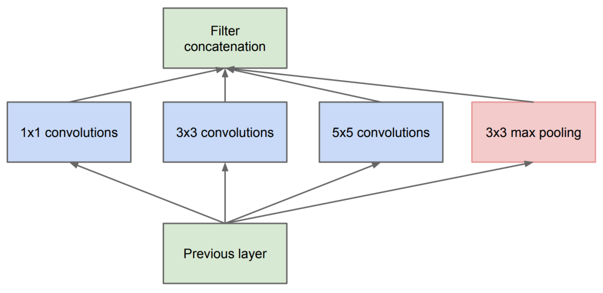


Figure 2.17: VGG 16 vs VGG 19 [100]

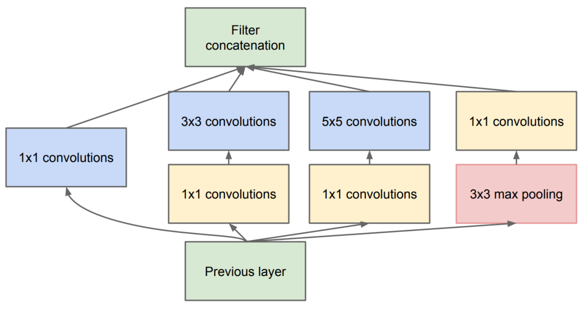
with the max pooling) [117]. The input is fed into three preliminary layers, which is called the stem [94], shown in the green box in figure 2.19. The output is a fully connected layer with global average pooling at the end, shown in the blue box in figure 2.19. In between are nine inception module stacked linearly, shown in the black boxes in figure 2.19. As mentioned before, this is a deep network, and thus suffers from the vanishing gradient problem. To solve this problem, the network has two auxiliary classifiers, shown in the red boxes in figure 2.19 [94]. It applies softmax to the outputs of two of the inception modules. The total loss function is a weighted sum of the auxiliary loss and the real loss, given by the following formula:

$$total_{loss} = real_{loss} + 0.3 \times (aux_{loss_1} + aux_{loss_2})[117,\ 94]$$

The above model in figure 2.19 is the first version, Inception V1. It finished top in the classification tasks in the ILSVRC 2014 classification challenge, with a top-5 error of 6.67% [64]. Currently there are a few other models as well: Inception V2, Inception V3 and Inception V4, all essentially building on the previous version to improve performance [118, 116].



(a) Inception module, naïve version



(b) Inception module with dimension reductions

Figure 2.18: Inception module. (a) is the naive version, and (b) has dimension reductions [117]

### ResNet

ResNet, short for Residual Network is a CNN that was introduced in 2015. It won first place in the ILSVRC 2015 classification competition with a top-5 error rate of 3.57% and came out on top at the COCO 2015 competition in ImageNet Detection, ImageNet localization, Coco detection and Coco segmentation [122].

The main goal of ResNet was to combat the vanishing gradient problem [40]. As mentioned previously, as more layers are added to a NN, the network hits a threshold and its performance degrades. The last two architectures show a few ways to deal with this problem, but ResNet incorporates a brand new method: the addition of a residual module [49].

The main idea of a residual module is the identity shortcut connection, also known as skip layers [50, 122]. Skip layers, as the name suggests, skips one or more layers. Figure 2.20 shows a diagram of the residual module. The input is x, which is both fed to the next layer, and has a direct connection, which skips a number of layers in between. Due to the skip connection, the output of the layer is changed. The output of the layer if it were a regular NN would be $H(x) = f(x)$, where $f()$ is the activation function [122]. But with the skip layer, the new output is $H(x) = f(x) + x$ [122]. The number of layers that are skipped is adjustable.

Skip connections help alleviate the vanishing gradient problem by creating a shortcut path for the gradient to flow through [122]. This skipping also allows the model to learn

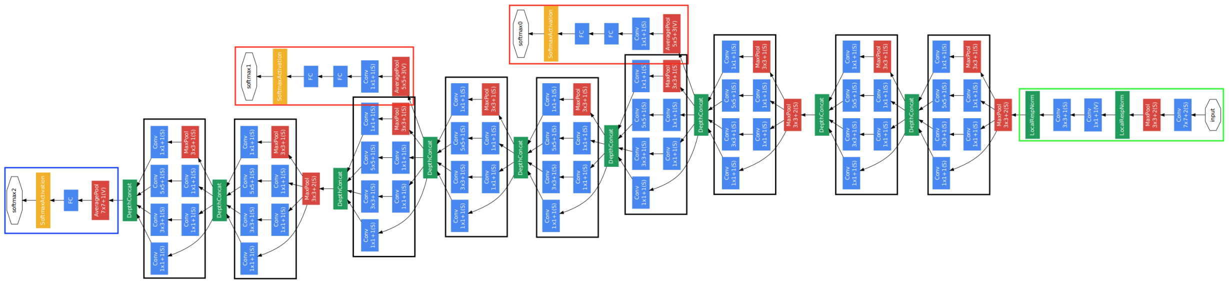


Figure 2.19: GoogleNet model architecture with the different components. The stem is in the green box. Nine inception modules are in the black boxes. The two auxiliary classifiers are shown in the red boxes. And the fully connected layer with global average pooling is in the blue box [117]

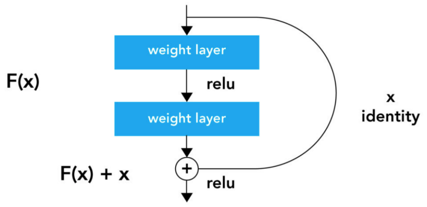


Figure 2.20: Residual module with skip layers [49]

identity functions ensuring that higher layers perform at least as good as lower layers [122].

The residual module allows stacked layers to fit a residual mapping, which is easier than directly fitting them to the underlying mapping. Researchers argue that deeper models should not produce higher training errors than its shallower models [122]. This indicates that stacking layers do not degrade network performance, because it stacks identity mappings, which are layers that do not do anything, on the current network [50].

Figure 2.21 shows a regular 34 layer CNN on the left and the ResNet34 architecture with 34 layers on the right [49]. ResNet34 has two layer skip connections, meaning that the identity mappings are skipped every two layers. Dotted skip connections indicate increase in channel dimensions. Similar to GoogleNet, ResNet also has an average pooling with a fully connected layer with 1000 outputs for classifying 1000 classes. ResNet comes in different variations: ResNet18, ResNet34, ResNet50, ResNet101, ResNet110, ResNet152, ResNet164, ResNet1202 and more [49, 122]. All of these follow the same format, just differ in the number of layers, which is indicated by the number. For example, ResNet50 has 50 layers.

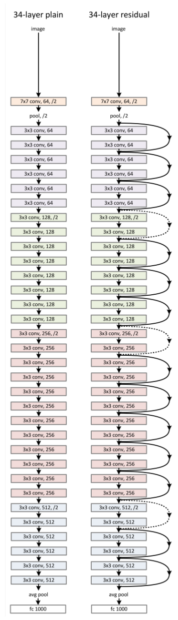


Figure 2.21: Left: Regular 34 layer CNN Resnet. Right: ResNet34 network with skip connections [49]

## 2.3 Deep Learning for Semantic Segmentation

The success of CNNs in image classification has led to its increased popularity in the task of semantic segmentation. For image classification, the output is in the shape of $1 \times 1 \times C$, where C is the number of classes in the dataset. For semantic segmentation, we need to classify each pixel as part of a class object set. Therefore, the input for a semantic segmentation model is still an image of shape $H \times W \times 3$, but the output is $H \times W \times C$, where $H$ and $W$ are the image dimensions [26]. The difference in output shape requires the model to up sample abstract small feature volumes back to the output shape [26].

In image classification the initial weights are randomly set. But this is not the case for semantic segmentation. Image classification CNN models are used to capture useful visual information like identifying colors, shapes, edges and curves. Thus the features learned by convolutional layers before the output layer are used as the initial weights for a semantic segmentation model. This image classifier is called *backbone* or *feature extractor*.

Most semantic segmentation models [70, 98, 48, 21] use an image classification CNN as a backbone to extract features [111, 49]. The backbone is trained on a classification dataset, such as ImageNet. This idea of reusing weights from a previously trained backbone CNN is called *transfer learning* [44]. ResNet [49] is one of the most commonly used backbones.

### 2.3.1 Fully Convoutional Network

A regular CNN has convolutional layers with pooling, where the image is down sampled, and then fully connected layers where it is classified. Fully Convolutional Networks were introduced in 2014, and it substitutes the fully connected dense layer with a $1 \times 1$ convolution layer, achieving the same results [73]. But the advantage of this substitution is the input size can now vary [65].

For a normal sized input image, the output is a class output. If the input image is bigger, the output will be a feature map, representing a heat map, shown in figure 2.22 [73]. This heat map can be further refined and used for segmentation.

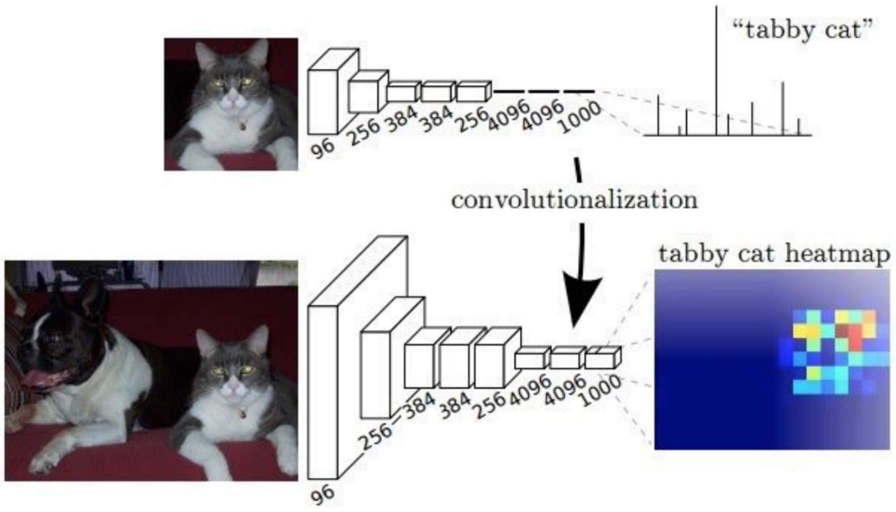


Figure 2.22: Difference between a regular sized image vs a bigger image. The regular size input outputs a classification, while the larger image outputs a heat map [70]

The heat map is the output down sampled. To get segmentation, the feature map needs to be up sampled. Down sampling is refereed to as *encoder*, and up sampling is decoder in a NN [73]. The input is reduced in size with the encoder and de-convoluted back up with the decoder. In FCN, specifically FCN-32, the input has been down sampled 32 times using convolution layers [70]. This requires up sampling by the same amount. However, down sampling 32 times results in loss of information at the final feature layer [73]. It is difficult to up sample 32 times with the resulting loss of information [73]. Thus the resulting segmentation is rough, shown in the left most image of figure 2.23.

The paper proposed two other model architectures: FCN-16 and FCN-8 [70]. In FNC-16, information is extracted from one pooling layer before the final convolution layer and is fused with the final feature map (element-wise addition) [123]. For FCN-8 information is extracted from the second to last pooling layer and is fused with the final feature map (element wise addition) [123]. A detailed diagram of these operations is shown in 2.24. For both these models, the output of the pooling layer is up sampled 16 and 8 times respectively. The segmentation results of both these models are shown in figure 2.23.

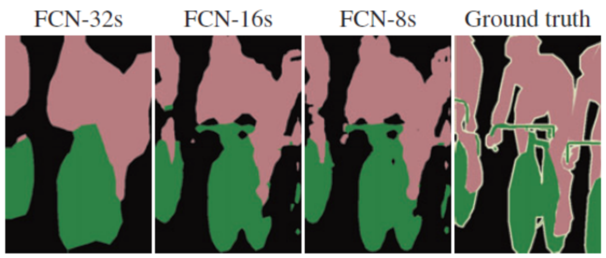


Figure 2.23: Outputs of FCN-32, FCN-16 and FCN-8 [70]

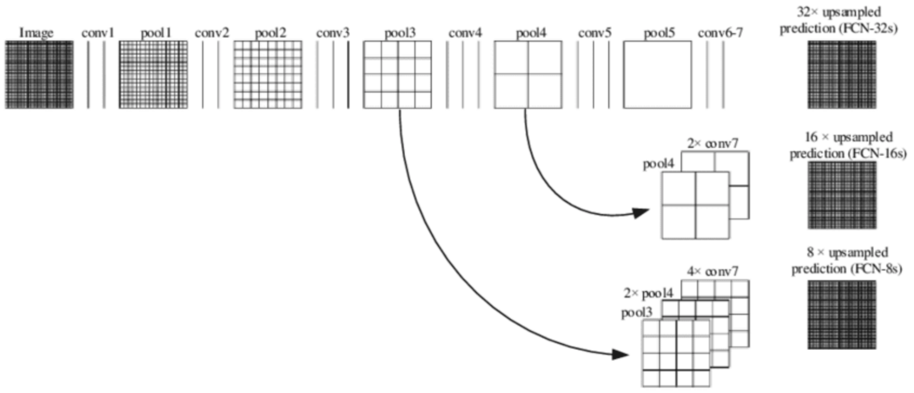


Image
conv1
pool1
conv2
pool2
conv3
pool3
conv4
pool4
conv5
pool5
conv6-7
32× upsampled prediction (FCN-32s)
2× conv7
pool4
16 × upsampled prediction (FCN-16s)
4× conv7
2× pool4
pool3
8 × upsampled prediction (FCN-8s)

Figure 2.24: Information extraction for FCN-16 and FCN-8 [70]

## 2.3.2 UNet

UNet was initially developed by Ronneberger for the medical field to identify tumors. UNet was inspired and built on top of FCN (Fully Convolutional Network) [70]. It only contains convolutional layers and no dense layers. UNets name comes from its architecture, which is shaped like a U, as shown in figure 2.25. The network has two parts, the encoder (on the left) and the decoder (on the right) as well as shortcut connections (grey horizontal arrows connecting layers from the encoder to the decoder) [56]. The encoder down samples the image to a feature map, and the decoder does the opposite, up samples the feature map to input image size, using deconvolution layers [73].

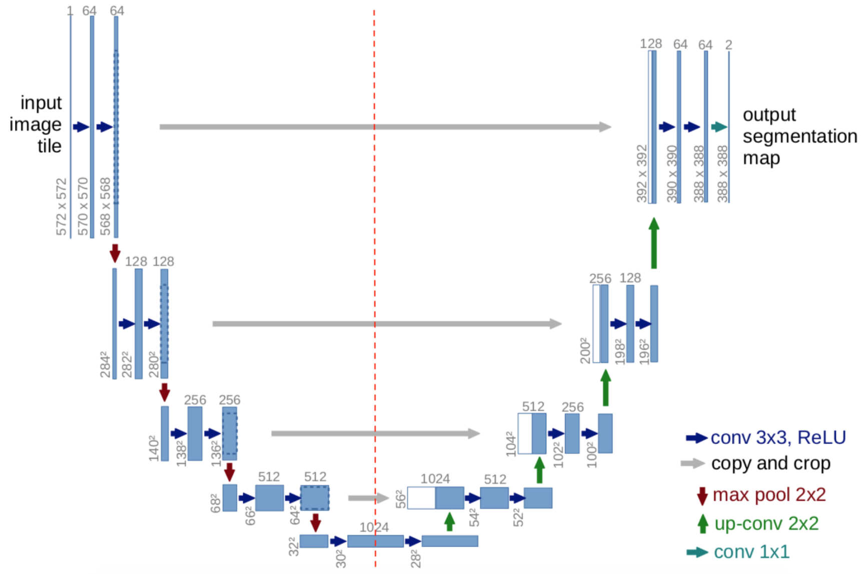


Figure 2.25: UNet Architecture. The left side of the red line is the encoder and the right side is the decoder. The horizontal grey lines connecting the encoder and decoder layers are the shortcut connections [98]

The encoder functions just like a normal CNN, with convolution layers and max pooling. Each level in the encoder portion of figure 2.25 consists of two $3 \times 3$ convolutional layers, followed by ReLU activation [98]. Transition between the levels is done by a $2 \times 2$ max pooling layer with a stride of 2 [98]. For every level in the encoder, the shape of the input halves, while the number of channels doubles. At the end of the encoder stage, the model knows the objects present in the image. The decoder handles the localization of these objects.

The decoder is essentially the reverse of the encoder. At each level, instead of max pooling, transpose convolutions or deconvolutions are carried out [32]. Figure 2.26 shows how a $3 \times 3$ input is up sampled to a $5 \times 5$ feature map. To do so, the decoder needs to add extra pixels in between and around the incoming pixels in the input from previous layers. The blue pixels are the original input. They are padded with zeros both on the outside and in between pixels, shown by the white squares. To improve performance, the zero padding is often replaced by weighted average of the input pixels such as bilinear interpolation [32].

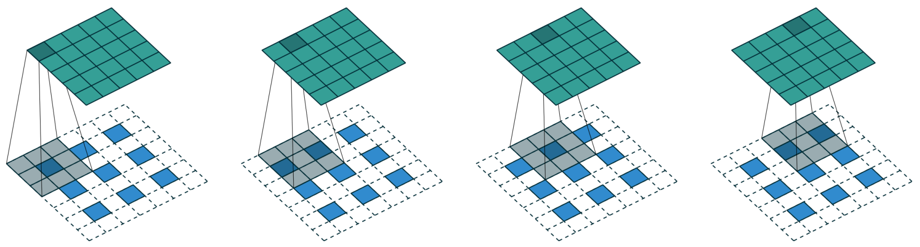


Figure 2.26: An example of transposed convolution which takes a $3 \times 3$ input and outputs a $5 \times 5$ feature map [32]

The final component of UNet is the shortcut connections. For localization of objects to be more precise, these shortcut connections or skip connections are used [73]. The feature maps of each level of the encoder are concatenated to the output of the transposed convolutions of each level in the decoder [73]. By sending information to every up sampling layer in decoder from the corresponding down sampling layer in the encoder the model can capture finer information. Layers at the beginning of the encoder have more information and would bolster the up sampling operation of decoder by providing finer details and improving the results [98].

### 2.3.3 DeepLab Network

DeepLab is a family of CNNs from Google. These networks focus on improving techniques to get better results at lower computational costs. These networks introduce three main improvements [56]:

1. Atrous convolutions
2. Atrous Spatial Pyramidal Pooling (ASPP)
3. Conditional Random Field (CFR)

As discussed previously with the FCN, consecutive pooling causes excessive down sampling. The down sampled image also needs to be up sampled to get the segmentation results. Down sampling results in a loss of information which is important for getting good segmentation results. Also deconvolution is computationally expensive since there are additional parameters involved in up sampling.

Atrous convolutions [20] or dilated convolutions was introduced to combat this issue. The idea is to increase the size of the filters by padding the weights around and in between with zeros or holes. The number of holes/zeroes in between the filter weights is known as **dilation rate** [73]. Figure 2.27 shows filters with different dilation rates. Dilation rate of 1 means no padding, and is thus a regular filter. Dilation rate of 2 means a padding of 1 between the filter weights, essentially turning the $3 \times 3$ filter to a $5 \times 5$ filter, while only using the same 9 parameters of a $3 \times 3$ filter. This enlarges the receptive field with no increase in computational cost.

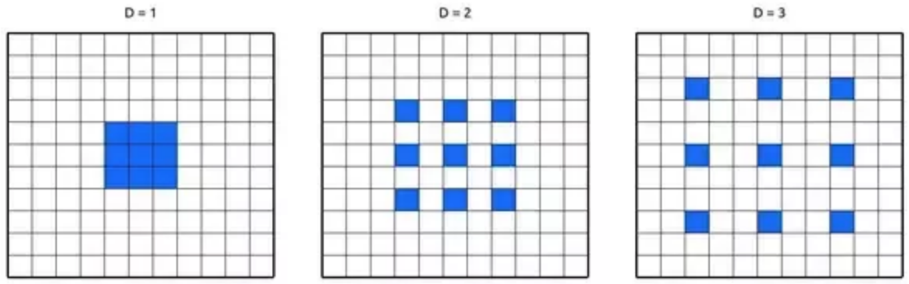


Figure 2.27: Atrous convolutions with dilation rate of 1, 2 and 3 [7]

Atrous Spatial Pyramidal Pooling (ASPP) [19] was introduced to capture multi-scale information from a feature map or capture objects at multiple scales. The idea is to apply Atrous convolutions with multiple dilation rates on an input, and put the results together [19]. Figure 2.28 demonstrates how its performed. ASPP helps account for the fact that objects will have different sizes in images, and thus improve results [73].

Pooling reduces the parameters, but also introduces invariance [73]. Invariance causes a NN to be unaffected by slight changes in input [19]. Thus the segmentation output obtained by a NN is coarse and the boundaries are not well defined. DeepLab proposes Conditional Random Field (CRF) to tackle this problem [19]. CFR is a post-processing step that tries to improve the results by defining sharper boundaries [73].

The Deeplabv3+ architecture combines the encoder-decoder design from UNet with the ASPP to get sharper boundaries. Figure 2.29 shows the model architecture. Instead of

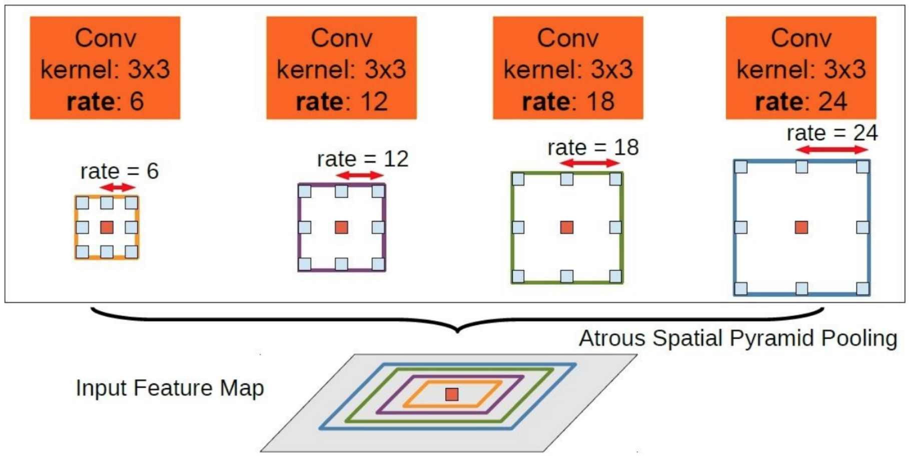


Figure 2.28: Atrous Spatial Pyramidal Pooling with Atrous convolutions of different dilation rates [19]

directly up sampling features from ASPP to the input dimensions, the decoder concatenates low-level features from both before and after ASPP and then upsamples to the output dimensions [19]. This encoder-decoder model design of Deeplab network is able to obtain sharp object boundaries.

## 2.4 Deep Learning for Weakly Supervised Semantic Segmentation

Weakly Supervised Semantic Segmentation (WSSS) deals with ground truth that does not have pixel level annotations. The neural network architecture and backbones are similar to Fully Supervised counterparts. The difference is the use of loss functions, the method of training the CNN and the use of additional data such as Saliency to combat the absence of pixel level annotation. In this section we explore a few WSSS Deep Learning methods that use image level annotation. Prior to that, we also introduce Class Activation Maps (CAMs) which are used as localization cues for most WSSS methods [3].

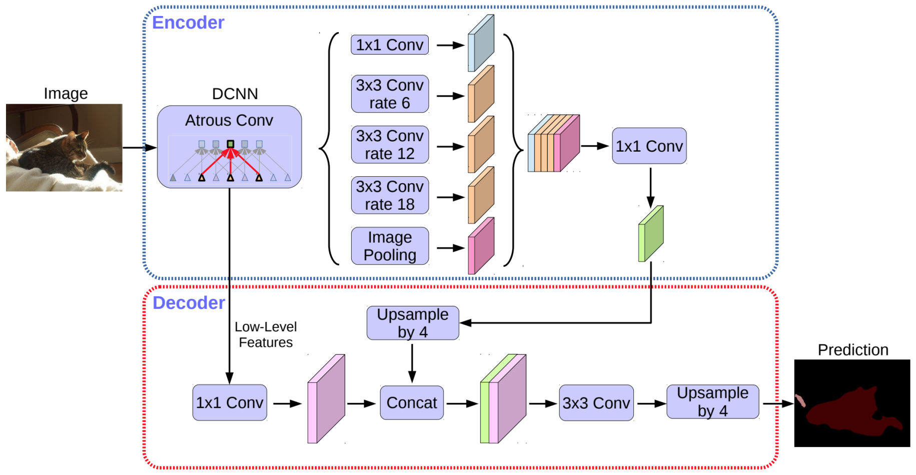


Figure 2.29: DeepLabV3+ model architecture

## Class Activation Map (CAM)

Given an input image, Class Activation Maps (CAM) highlights all the important regions of the image a NN uses to make decisions, such as output predictions, or classifications [8]. Visually, it looks like a heat map of objects present in an image, as shown in figure 2.30 [42]. Mathematically, it is the weighted activation map generated for each image [137]. CAMs were proposed by a team in MIT in 2015.

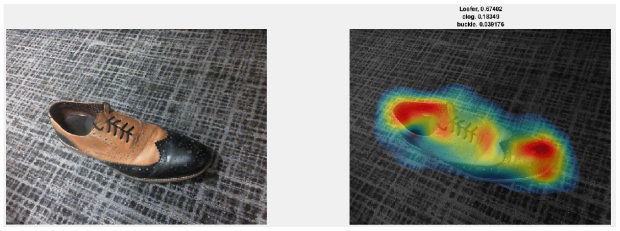


Figure 2.30: Class Activation Map (CAM) of image of a shoe. The highlighted heat map is the region of the image a NN looks at for classifications [42]

CAMs are trained in a weakly supervised manner [15]. Below in figure 2.31 is the model architecture for producing CAMs. The model consists of a few convolution layer. The fully connected layer at the end is removed to keep the spatial information that is present at the final convolution layer. A global average pooling (GAP) is performed before the final output layer. GAP is an average that is taken across all the activation maps which help find all the important regions of the image [24]. Global Max Pooling (GMP) is another layer that can be used, but it focuses on finding the top most important region, instead of all the important regions, and thus is less effective than GAP [24]. The final layer is a softmax layer, with neurons that match the number of classes in the classification list.

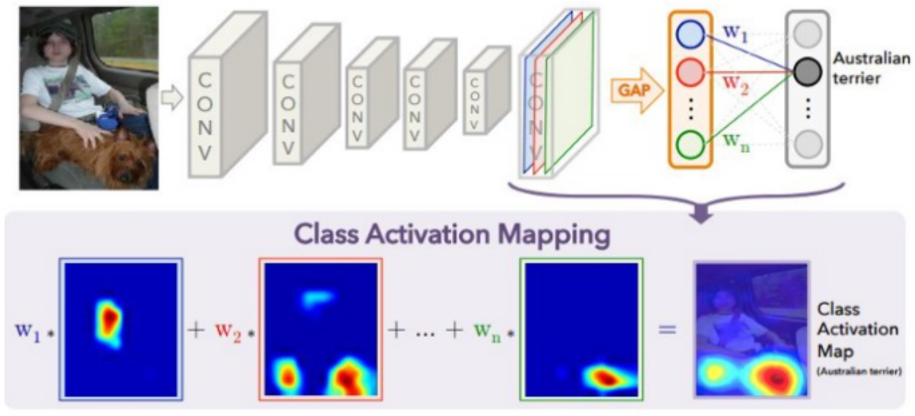


Figure 2.31: Class Activation Map (CAM) model architecture [137]

The importance regions of the image are found by projecting back the weights of the output layer obtained from the last conv layer onto the convolutional feature maps [15]. At the first iteration, the input image is fed to the network to output the image classification. In the next iteration the weights connected to the correct classification neuron are used. The output of the final convolution layer is stored and used to calculate CAM, by multiplying each depth from the output of the final convolution layer with corresponding weight connected to the correct neuron [15]. These products are summed and up sampled using bilinear up sampling to match the size of the input [15]. Figure 2.30 shows the input image and its corresponding CAM.

### 2.4.1 From Image-level to Pixel-level Labeling with Convolutional Networks

Pinheiro and Collobert [93] devised a method based on the Multiple Instance Learning Framework, which works as follows. Given an image and its object class, the image is known to have (or not) at least one pixel corresponding to the object class, and the segmentation task is inferring the pixels belonging to the object class. To this end the method uses a

CNN which is trained to put more weight on pixels that are important for classifying the image.

The task of segmentation is challenging as each object can generate an infinite amount of images with different position, pose, lightning, texture, background etc. Segmentation has to keep up with these variations with limited training data. CNNs are used because of their ability to learn general information and thus excel in transfer learning, which can be exploited for different vision tasks. However the disadvantage is the need for a large fully labeled training dataset.

Pinheiro and Collobert [93] uses a CNN, but it is not trained with segmentation labels, or bounding box annotations [93]. Instead, their method considers a single object class label for a given image, and the model puts more weight on important pixels for classification [93]. This approach can be seen as an instance of Multiple Instance Learning (MIL).

As mentioned before, CNN learns a hierarchy of filters. The type of features learned make it ideal for transfer learning. Since the amount of classification data is much bigger than segmentation data, it is natural to leverage the former to solve the later. Pinheiro and Collobert's method considers segmentation with a set of classes $C$ [93]. They assumes that the classification dataset contains at least the same classes. The classes that are in the classification dataset but not the segmentation dataset are considered to be a part of the background class.

Figure 2.32 shows the architecture of the model. The CNN has 10 layers with pooling. The CNN takes an RGB image of $400 \times 400$ and outputs $C + 1$ planes, one for each class and one for the background. During training, an extra layer, the Log-Sum-Exp layer is used to aggregate pixel level labels to image level labels [93].

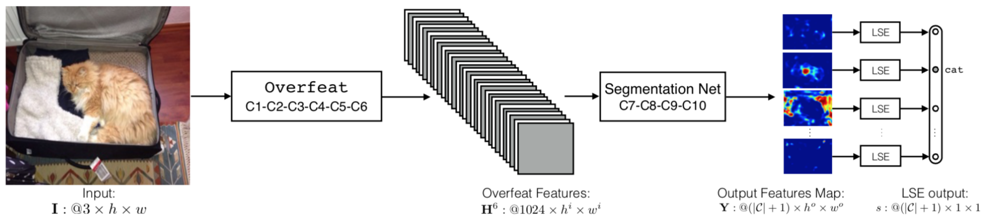

Overfeat
C1-C2-C3-C4-C5-C6
Segmentation Net
C7-C8-C9-C10
LSE
cat
Input:
$\mathbf{I} : @3 \times h \times w$
Overfeat Features:
$\mathbf{H}^6 : @1024 \times h^i \times w^i$
Output Features Map:
$\mathbf{Y} : @(|\mathcal{C}|+1) \times h^o \times w^o$
LSE output:
$s : @(|\mathcal{C}|+1) \times 1 \times 1$

Figure 2.32: The full RGB image is forwarded through the network (composed of Overfeat and four extra convolutional features), generating output planes of dimension $(C{+}1){\times}h{\times}w$. These output planes can be seen as pixel-level labels of a sub-sampled version of the input image. The output then passes through the Log-Sum-Exp layer to aggregate pixel-level labels into image-level ones. The error is backpropagated through layers C10-C7 [93]

The model starts with the Overfeat feature extractor developed by Sermanet el at [93]. This feature extractor has been trained on the ILSVRC13 challenge and corresponds to the first six layers of the model. The feature extractor takes the $400 \times 400$ RGB image and generates feature maps of dimensions $1024 \times h \times w$, where $h$ and $w$ are the size of the RGB input image, the convolution kernel sizes, convolution strides and max-pooling sizes [93]. After the first 6 convolution layers and 2 pooling layers of Overfeat, the RGB $400 \times 400$ image is transformed to a $1024 \times 29 \times 29$ feature representation [93].

The other 4 layers are for the feature planes coming from Overfeat. The first three layers are followed by a pointwise ReLU:

$$H^p = max(0, W^p H^{p-1} + b^p), p\epsilon(7, 8, 9)[93]$$

Parameters of layer $p$ are $(W^p, b^p)$. The last layer $\mathbf{Y}$ is:

$$Y = W^{10} H^9 + b^{10}[93]$$

There is no max pooling, but a drop out regulation strategy is applied on all layers. The network outputs $C + 1$ planes, one for each class and one for the background.

The network also produces a score $s^k_{i,j} = Y^k_{i,j}$ for each pixel location (i, j) in image I, for each class $k\epsilon C$ [93]. These scores need to be aggregate from pixel level into image level classification. Aggregation drives the network towards correct pixel-level assignments, such that the network could perform decently on segmentation tasks. This aggregation is done by the Log-Sum-Exp (LSE):

$$s^k = \frac{1}{r} log[\frac{1}{h * w} \sum_{i,j} exp(r, s^k_{i,j})][93]$$

The parameter $r$ controls how smooth the approximation is.

During inference, the padded and normalized RGB image is sent to the network, where the aggregation layer has been removed. This generates $C + 1$ planes of pixel-level scores $s^k_{i,j}$ and the CNN densely classifies every pixel of the image. The entire process is shown in figure 2.33.

### 2.4.2 Weakly and Semi Supervised Learning of a Deep Convolutional Network for Semantic Image Segmentation

Papandreau et al. [90] propose an online Expectation-Maximization (EM) method for training CNN semantic segmentation models from weakly annotated data [90]. The proposed algorithms alternate between estimating the latent pixel labels (subject to the weak

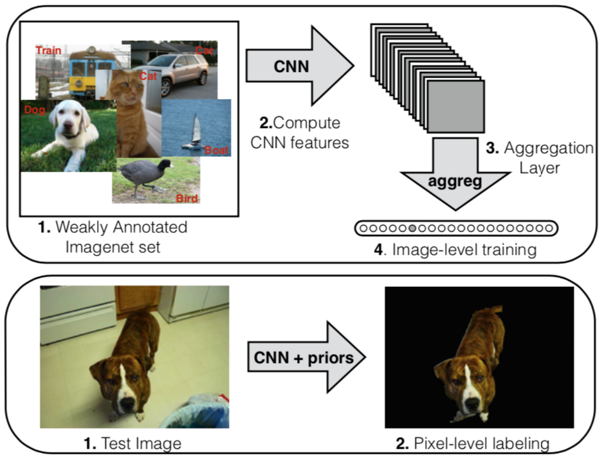


Figure 2.33: Top: (1) The model is trained using weakly annotated data. (2) The CNN generates feature planes. (3) These planes pass through an aggregation layer to constrain the model to put more weight on the right pixels. (4) The system is trained by classifying the correct image-level label. Bottom: During inference, the aggregation layer is removed and the CNN densely classifies every pixel of the image [93]

annotation constraints), and optimizing the CNN parameters using stochastic gradient descent (SGD) [90].

DeepLab for semantic segmentation uses a CNN to predict label distribution per pixel, followed by a fully connected dCRF to smooth predictions [90]. This method focuses on methods for training the CNN parameters using weak labels.

Figure 2.34 (a) shows how DeepLab model trains from fully supervised annotated images. The image is sent to a CNN and produces an output. The output is used with the ground truth to produce a loss which is used to optimize the model and its parameters. Figure 2.34 (b) shows the changes to the model when only image level annotation is available. The main difference is the addition of the weakly supervised E-step module. The method pursues a EM-approach in order to learn the model parameters from training data by adopting a hard-EM approximation, estimating in the E-step of the algorithm the latent segmentation by argmax [90].

The method described in the paper for image level labels shows results that are close to [93], without using any external objectness or segmentation module [90]. The main focus of this paper is semi supervised learning where having access to 2.9k pixel level labels and 9k image level annotated images yields results that 2% inferior to the performance of the

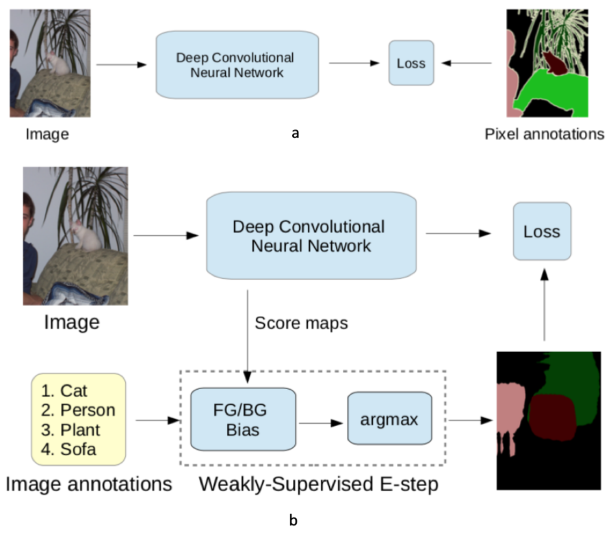


Figure 2.34: (a) DeepLab model training from fully annotated images (b) DeepLab model training using image level labels [90]

system trained with all 12k images strongly annotated at the pixel level [90].

### 2.4.3 FickleNet: Weakly and Semi-supervised Semantic Image Segmentation using Stochastic Inference

Pixel level annotations allow fully supervised methods to effectively learn location and boundaries of objects. Weakly labeled data on the other hand only indicates the existence of objects in an image. Most weakly supervised methods use localization maps such as Class Activation Maps (CAM) to bridge the gap between full and weak supervision. But these maps focus on small parts of an image and do not effectively capture boundaries. This problem can be addressed by diverting the classifier from its primary task of discrimination between objects to discovering the relations between pixels [67].

This paper [67] proposes a three step method to tackle the task of WSSS. First it introduces a new network, **FickleNet**, which using the dropout method, discovers the relationship between locations in an image and enlarges the regions activated by the classifier [67]. FickleNet uses stochastic selection of hidden units and is trained for multi class classification [67]. The classification scores from the first step are used to generate localization maps of training images. These two steps address the problem described above. Finally, the localization maps are used as pseudo ground truths to train a segmentation network [67].

FickleNet can generate a variety of localization maps from a single image using random combinations of hidden units in a convolutional neural network, as shown in figure 2.35 (a) [67]. Starting with a feature map created by a generic classification network such as VGG 16 or ResNet 101, the network chooses hidden units at random for each sliding window position, which corresponds to each stride in the convolution operation, realized by the drop out method , as shown in figure 2.35 (b) [67]. Selecting all the hidden units in a sliding window does not allow distinction between foreground and background. Random selection of hidden units produces regions of different shapes which can delineate objects more sharply [67]. Since the patterns of hidden units randomly selected by FickleNet include the shapes of the kernel of the dilated convolution with different dilation rates, FickleNet can be regarded as a generalization of dilated convolution, but FickleNet can potentially match objects of different scales and shapes using only a single network because it is not limited to a square array of hidden units, whereas dilated convolution requires networks with different dilation rates just to scale its kernel [67].

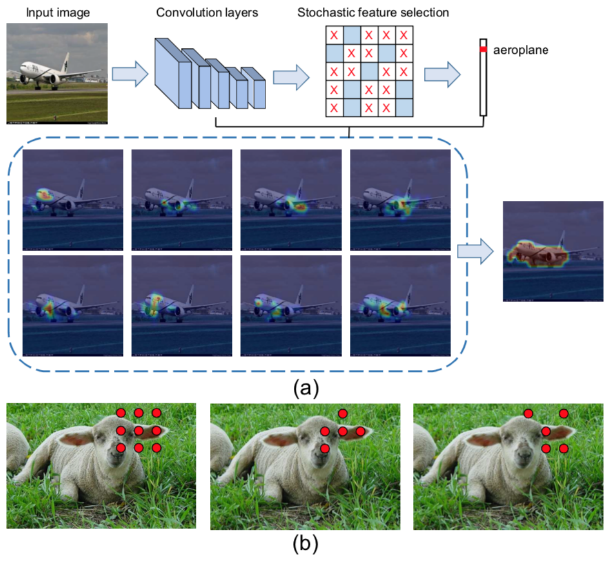


Figure 2.35: (a) FickleNet allows a single network to generate multiple localization maps from a single image. (b) Conceptual description of hidden unit selection. Selecting all hidden units produces smoothing effects as background and foreground are activated together. Randomly selected hidden units (stochastic, center and right) can provide more flexible combinations which can correspond more clearly to parts of objects or the background [67]

As mentioned before this method is a three step process:

- Stochastic Hidden Unit Selection
- Inference Localization Map
- Training the Segmentation Network

Stochastic hidden unit selection is used in FickleNet to discover relations between parts of objects by exploring the classification score computed from the randomly selected pairs of hidden units, with the aim of associating a nondiscriminative part of an object with a discriminative part of the same object. This process is realized by applying spatial dropout to the feature at each sliding window position, as shown in figure 2.36 (a). However to implement the method shown in figure 2.36 (a) would require calling the convolution and the dropout function $w \times h$ times in each forward pass [67]. By expanding the input feature map as shown in figure 2.36 (b), so that no sliding window overlaps, reduces to a single call to each function during each forward pass [67]. Zero padding is applied to the feature before expanding so that the size of the final output is the same as the input.

Stochastic hidden unit selection is realized by applying dropout method to spatial locations. Applying spatial dropout with a rate $p$ outputs feature map $x_p^{expand}$ [67]. Applying global average pooling and a sigmoid function to this map, outputs a classification score $S$ for each randomly selected hidden unit [67]. A combination of randomly selected hidden units generates various classification scores from a single image.

The second step is generating the localization maps. FickleNet allows many localization maps to be constructed from a single image, because different combinations of hidden units are used to compute classification scores at each random selection [67]. GradCAM, which is a generalization of CAM, is used to create these localization maps. N different localization maps are created from a single image and aggregated into a single localization map, as shown in figure 2.35 (a) [67].

The final step is training the segmentation network. The localization maps created above are used as pseudo ground truths to train a segmentation network. Using the same background cues as Deep Seeded Region Growing (DSRG) [53], the generated localization maps from FickleNet are fed into DSRG as the seed cues for weakly supervised segmentation [67]. Figure 2.37 shows comparisons between DSRG and FickleNet using both VGG 16 and ResNet 101 on Pascal VOC 2012 dataset [36, 35, 34].

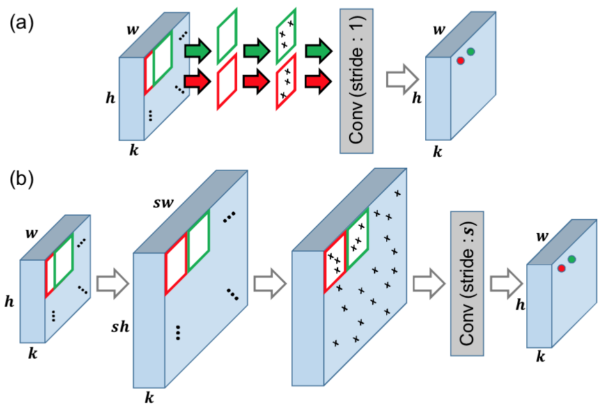


Figure 2.36: (a) Naive implementation of FickleNet, which requires a dropout and convolution function call at each sliding window position (the red and green boxes). (b) Implementation using map expansion: convolution is now performed once with a stride of s. The input feature map is expanded so that successive sliding kernels (the red and green boxes) do not overlap [67]

### 2.4.4 Learning Pixel-level Semantic Affinity with Image-level Supervision for Weakly Supervised Semantic Segmentation

In the weak supervision setting with only image level labels the model does not know object location or shape. The only information available is the existence of classes. A popular choice to supplement this gap of location cue knowledge is by using Class Activation Maps (CAM). CAMs, highlight discriminative parts of a target object by investigating the contribution of hidden units to the output of a classification Deep Neural Network (DNN) [3].

In weak supervision trained models are known to segment local discriminative parts rather than the entire object area. One solution is to propagate local responses to nearby areas with the same semantic entity. To do so, this method introduces a DNN called AfinityNet that predicts semantic affinity between a pair of adjacent image coordinates [3].

The pipeline for this method is shown in figure 2.38. It has three distinct stages:

1. First, CAMs of input images are created and used to generate semantic affinity labels, which in turn are used to train AffinityNet [3]
2. Second, the trained AffinityNet is applied to each training image to revise its CAM and synthesize segmentation labels. This revision is done by first building a neigh-

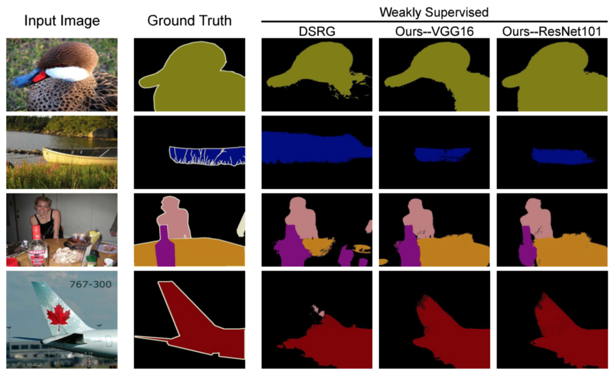


Figure 2.37: Comparisons between DSRG and FickleNet using both VGG 16 and ResNet 101 on Pascal VOC 2012 dataset [67]

borhood graph where each pixel is connected to its neighbors within a certain radius, and estimate semantic affinities of pairs connected in the graph through AffinityNet [3]. Sparce activations in CAMs are diffused by random walk, which revises CAMs significantly, producing fine object shapes. Finally, this process is applied to training images for synthesizing their segmentation labels by taking the class label associated to the maximum activation of the revised CAMs at each pixel [3]

3. Third, the segmentation labels are used to train a DNN for semantic segmentation

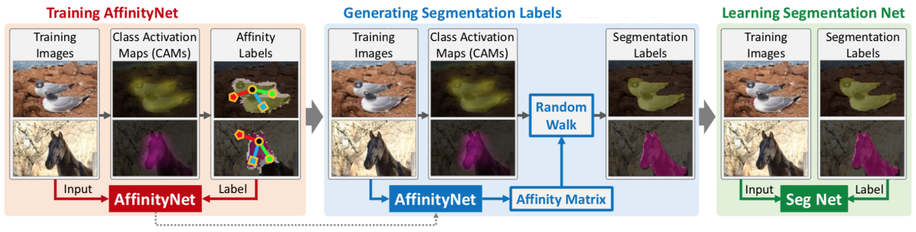


Figure 2.38: Pipeline for the method. Salient areas for object classes and background are first localized in training images by CAMs [3]

The entire framework is based on three DNNs:

- Network to compute CAMs
- AffinityNet

- Semantic Segmentation network

Each of these networks use ResNet-38 as the backbone. The first two networks are used to generate segmentation labels of training images for the last network which does semantic segmentation.

The architecture for computing CAMs, as described previously, is a typical classification network with global average pooling (GAP) followed by a fully connected layer, trained by a classification network with image-level labels [3]. For this method, three layers are added on to ResNet-38: a $3 \times 3$ conv layer with 512 channels, a GAP layer for feature aggregation, and finally a fully connected layer for classification [3].

AffinityNet is designed to predict a convolutional feature map $f^{aff}$ [3]. Figure 2.39 shows the architecture of AffinityNet. The network predicts class agnostic semantic affinity between a pair of adjacent coordinates in an image [3]. The predicted affinities are used in random walk as transition probabilities so that random walk propagates activation scores of CAMs to nearby areas of the same semantic entity, which improves the quality of CAMs significantly [3]. The network acquires semantic information at various field of views by aggregating multi level feature maps from the last three levels of the ResNet-38 network. Before aggregation, the channel dimensionalities are reduced to 128, 256, and 512, for the first, second, and third feature maps, respectively, by individual $1 \times 1$ convolution layers [3]. These are concatenated to a single feature map with 896 channels. Finally one more $1 \times 1$ convolution layer with 896 channels is added on the top for adaptation [3]. The trained AffinityNet is now used to revise CAMs for training images using random walk.

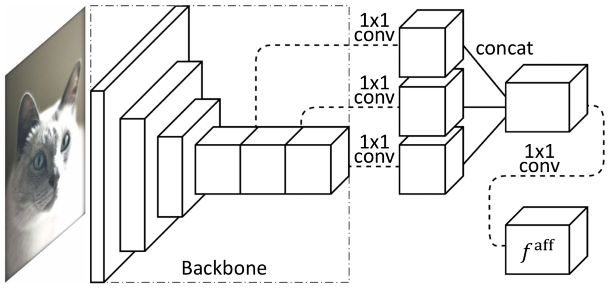


Figure 2.39: Architecture of AffinityNet.The output feature map $f^{aff}$ is obtained by aggregating feature maps from multiple levels of a backbone network so that $f^{aff}$ can take semantic information at various field of views [3]

Finally, the revised CAMs are used to train the segmentation network. The CAMs need to be upsampled by bilinear interpolation, to match the input image size. A segmentation label of a training image is then obtained simply by selecting the class label associated with the maximum activation score at every pixel in the revised and upsampled CAMs [3]. The segmentation network is created by adding two more atrous layers to ResNet-38. These layers have the same dilation rate of 12, and 512 and 21 channels respectively.

### 2.4.5 Seed, Expand and Constrain: Three Principles for Weakly-Supervised Image Segmentation

This paper [59] introduces a new loss function for weak supervision called **SEC**. This loss function is based on three principles: Seed, Expand and Constrain [59].

The first insight is to seed with weak localization cues. Classification networks can output object localization cues or seeds, but cannot predict their exact spatial extent. This method incorporates seeding loss that encourages a segmentation network to match localization cues [59].

The second insight is to expand objects based on the information about which classes can occur in an image. Global pooling layer is used to aggregate segmentation masks into image-level label scores. There are two prominent choices: global average pooling (GAP) and global max pooling (GMP). The former overestimates the quality of segmentation, while the latter underestimates it. This method generalizes max pooling and average pooling by using a global weighted rank pooling that is leveraged by expansion loss to expand the object seeds to regions of a reasonable size [59].

Finally the third insight is to constrain the segmentation to coincide with object boundaries. This method proposes a new constrain-to-boundary loss that alleviates the problem of imprecise boundaries at training time [59]. The method constraints predicted segmentation masks to respect low-level image information

Figure 2.40 shows the model architecture of SEC with the three paths, each corresponding to one of the three principles.

The top path, corresponding to seed, is shown in more details in figure 2.41. The method proposes to use a seeding loss to encourage predictions of the neural network to match only landmarks given by the weak localization while ignoring the rest of the image [59].

The middle path corresponds to expansion. To measure if a segmentation mask is consistent with the image level labels one can aggregate segmentation scores into classification scores and apply the standard loss function for multi label image classification [59]. Two

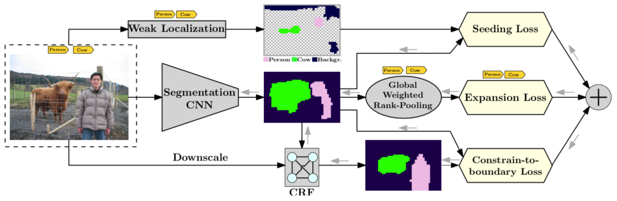


Figure 2.40: Schematic illustration of SEC that is based on minimizing a composite loss function consisting of three terms: seeding loss, expansion loss and constrain-to-boundary loss [59]

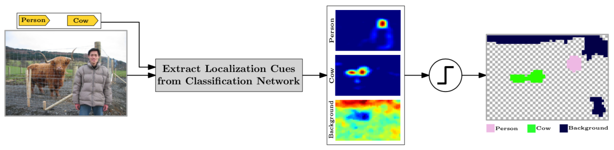


Figure 2.41: Schematic illustration of the weak localization [59]

prominent techniques are used: GMP and GAP. For classes which are present in an image GMP only encourages the response for a single location to be high, thus resulting in a segmentation network that often underestimates the sizes of objects while GAP encourages all responses to be high, which overestimates the sizes [59].

The method proposes Global Weighted Rank Pooling (GWRP) that is a combination of GMP and GAP. GWRP computes a weighted average score for each class, where weights are higher for more promising locations [59]. This encourages objects to take up a certain fraction of an image without overestimating their size.

### 2.4.6 Weakly supervised image semantic segmentation using graph convolutional networks

As seen in previous methods, one common approach to WSSS is to use Class Activation Maps (CAMs) and a random-walk mechanism to create pseudo ground truths for training a fully supervised semantic segmentation network. However, the feed forward nature of the random walk imposes no regularization on the quality of the resulting pseudo ground truths. To overcome this issue, Pan et al. proposes a Graph Convolutional Network (GCN) based feature propagation framework [89].

This paper [89] uses a two stage framework that is common with most weak supervision methods. The first stage is producing pseudo ground truths, and the second is using these labels to train a segmentation network. Producing pseudo ground truths is done in three steps:

1. First, predict crude estimates of labels using CAMs. These are called partial pseudo labels [89]
2. Second, use these partial labels to train an affinity network, where pixels with the same labels have similar features
3. Third, the affinity network is applied to evaluate a Markov transition matrix for propagating the activation scores of CAMs through a random-walk mechanism, with the aim of producing complete pseudo labels for all the pixels [89]

This paper addresses step 3 through a GNN. The high level semantic features of image pixels are propagated on a graph using a 2-layer graph convolutional network, followed by decoding the propagated features into semantic predictions [89]. A separate GCN is learned for every training image by back-propagating a Laplacian loss and an entropy loss to ensure the consistency of semantic predictions with image spatial details [89].

GCN is chosen over CNN because of the affinity relations between features. Figure 2.42 shows the overall architecture of the method, with the two stages. Stage I generates the pseudo ground truths and Stage II trains the segmentation network.

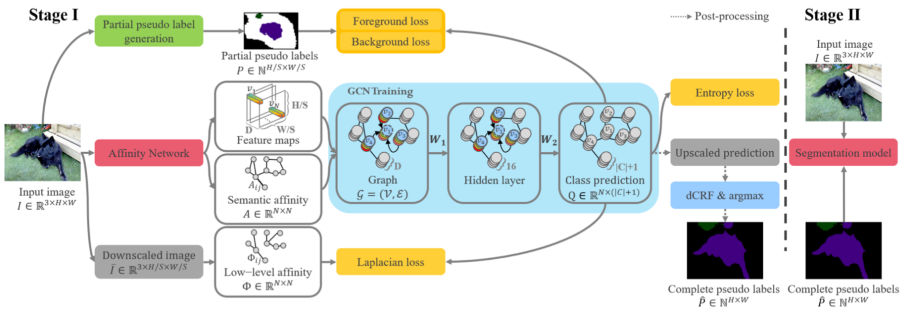

Stage I
Partial pseudo label generation
Partial pseudo labels $P \in \mathbb{N}^{H/S \times W/S}$
Foreground loss
Background loss
Post-processing
Stage II
Input image $I \in \mathbb{R}^{3 \times H \times W}$
GCN Training
$v_1$
$v_N$
H/S
D
W/S
Feature maps
Affinity Network
Input image $I \in \mathbb{R}^{3 \times H \times W}$
$A_{ij}$
Semantic affinity $A \in \mathbb{R}^{N \times N}$
Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
$W_1$
Hidden layer
16
$W_2$
Class prediction $Q \in \mathbb{R}^{N \times (|C|+1)}$
|C|+1
Entropy loss
Upscaled prediction
Segmentation model
dCRF & argmax
Downscaled image $\bar{I} \in \mathbb{R}^{3 \times H/S \times W/S}$
$\Phi_{ij}$
Low−level affinity $\Phi \in \mathbb{R}^{N \times N}$
Laplacian loss
Complete pseudo labels $\hat{P} \in \mathbb{N}^{H \times W}$
Complete pseudo labels $\hat{P} \in \mathbb{N}^{H \times W}$

Figure 2.42: Overview of the proposed 2-stage weakly-supervised image semantic segmentation framework with GCN-based feature propagation [89]

# Chapter 3

# Volumetric Loss

For weakly supervised semantic segmentation with volumetric annotation, we need a new loss function. Namely we need a loss function that encourages an object of a given class have size matching to that in the user annotation. The name "volumetric" instead of "size" is used for historical reasons. One of the first works to use constraints on size was [127], who used size constraints on 3D reconstructed volumes, therefore, the name volumetric is popular for size constraints. We are the first to introduce volumetric weak supervision for semantic segmentation.

In this chapter, we describe two types of volumetric losses that we designed, namely **Quadratic Loss** and **Outside Quadratic Loss**. Quadratic loss penalizes any deviation from the size provided by the user. Since user sizes are only approximate, our second volumetric loss, Outside Quadratic, has nonzero penalty only if the size estimated by CNN falls outside the bucket estimate provided by the user.

## 3.1 Introduction

To develop volumetric loss, we need an estimate of the size of each class in the output produced by a CNN. We can estimate the size of each class from the output of a CNN as follows. Suppose we have a total number of $c$ classes. Let $t$ be the CNN output, and assume this output has the same spatial resolution as the input image, as is usually the case in semantic segmentation. Then for each pixel, the output has as many channels as there are classes. Let $t_i^p$ be the CNN output corresponding to pixel $p$ in the input image and class $i$. Furthermore, we assume that the last CNN layer is softmax, so that $t_i^p \in [0, 1]$. A pixel $p$ is assigned to the class that maximizes $t_i^p$ over $i$.

Since each pixel $p$ is assigned to the class that maximize $t_i^p$ over $i$, to estimate the size of class $i$ in the network output $t$, we add $t_i^p$ over all image pixels $p$, namely let

$$\hat{t}_i = \sum_p t_i^p \tag{3.1}$$

Note that this is only an estimate of the size of class $i$, but the closer network outputs are to 0 and 1, the more precise this estimate gets. In practice CNN output often has low entropy, i.e., $t_i^p$ is indeed close to either 0 or 1.

Our volumetric loss functions are based on encouraging $\hat{t}_i$ to be close to the size that is provided by user annotation.

## 3.2 Quadratic Volumetric Loss

The first loss function, Quadratic Volumetric Loss, is based on Mean Square Error. Let $I$ be an input image, and let $size_i$ be the size of class $i$ in image $I$ as estimated by the user. This loss function penalizes any deviations from the size of the class $i$ estimated by the network, namely $\hat{t}_i$, see Eq. 3.1 from the size estimated by the user. Let $t$ be the output corresponding to image $I$. Then this loss for a single image is defined as:

$$L_{QVL}(t) = \sum_{i=1}^{c} (\hat{t}_i - size_i)^2, \tag{3.2}$$

where $size_i$ is the size of class $i$ for image $I$, as annotated by the user, and $\hat{t}_i$ as defined in Eq. 3.1. When we train, we average the loss in Eq. 3.2 over all images in the training dataset. Quadratic loss encourages each class to have size equal to the estimate provided by the user. Figure 3.1 shows the relationship between network prediction and Quadratic loss for a class with annotated size equal half of the image.

The advantage of Quadratic loss is that it has a smooth nonzero gradient almost everywhere, and thus might be easier to optimize with gradient descent. The disadvantage is that it encourages each class $i$ to have exactly the size as estimated by the user, namely $size_i$. However, we know that the user estimate is only approximate, and ideally, there should not be much (if any) penalty if class size is within some radius from the user estimate. Thus we develop another loss function, described in the next section.

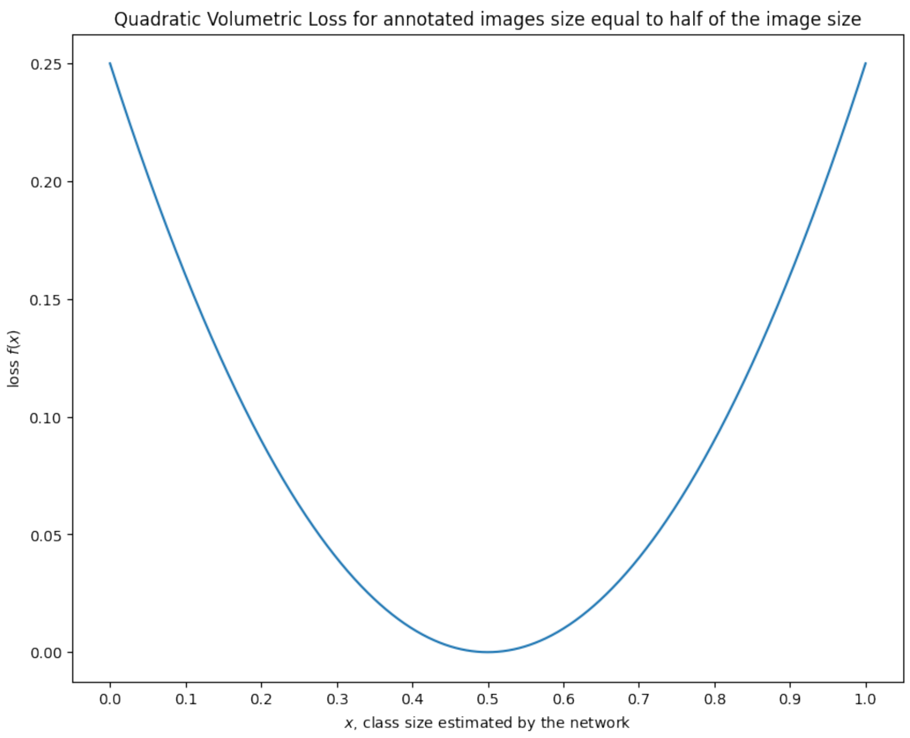


Figure 3.1: Plot of Quadratic loss when the network estimate of object size is $x$, for the case when the user annotates the class size to be half of the image size, i.e. 0.5

## 3.3 Outside Quadratic Volumetric Loss

In the second loss function, Outside Quadratic Volumetric Loss, the idea is to penalize the prediction of the network when the class sizes it estimates are outside of the size bucket provided by the user. Let $I$ be an input image, and let $size_i$ be the size of class $i$ in image $I$ as estimated by the user.

For each class $i$ in image $I$, let $a(i)$ be the lower end of the size bucket, and $b(i)$ be the upper end of size bucket, i.e. $a(i) \leq size_i \leq b(i)$. Let $t$ be the output corresponding to image $I$. If $\hat{t}_i$ is within $a(i)$ and $b(i)$, the corresponding loss will be 0. If it is less than $a(i)$, the penalty term will be dependent on the distance from $a(i)$, and if its greater than $b(i)$, the penalty term will be dependent on the distance from $b(i)$. We take the square of the difference between the network prediction and the end points if the prediction falls outside the end points and sum them up for each class. The loss for a single image is defined as:

$$f(x, a, b) = \begin{cases} (x-a)^2 & \text{if } x < a \\ (x-b)^2 & \text{if } x > b \\ 0 & \text{if } a \leq x \leq b \end{cases} \tag{3.3}$$

$$L_{OQVL}(t) = \sum_{i=1}^{c} f(\hat{t}_i, a(i), b(i)) \tag{3.4}$$

Similar to the previous loss, when we train, we average the loss in Eq. 3.4 over all images in the training dataset.

Outside quadratic loss considers the fact that the user estimates the object size only approximately. Thus there is no penalty if the size estimated by the network is not too far from the estimate (bucket) provided by the user. Figure 3.2 shows the relationship between network prediction and outside quadratic loss for the bucket with endpoints $a = 0.4$ and $b = 0.5$.

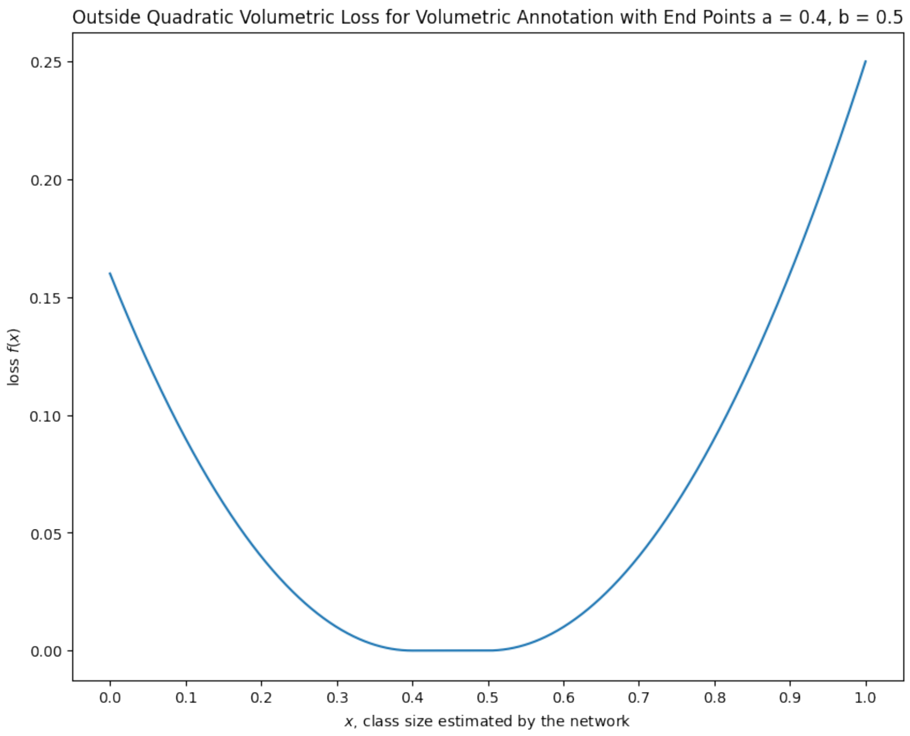


Figure 3.2: Plot of Outside Quadratic loss when the network estimate of object size is $x$, for endpoints $a = 0.4$ and $b = 0.5$. The loss is 0 between the end points.

# Chapter 4

# Volumetric Supervision

In this chapter, we show how weak volumetric supervision can be used for semantic image segmentation. We can take almost any method for weak supervision with image labels, and convert it to a method for volumetric supervision, using the volumetric loss functions introduced in chapter 3, assuming that a dataset with object sizes, in addition to image level tags, is available. In this chapter, we show how four recent methods [133, 6, 17, 22] for weakly supervised semantic segmentation with image level labels can be converted to weak volumetric supervision methods.

## 4.1 Joint Learning of Saliency Detection and Weakly Supervised Semantic Segmentation

The paper [133] proposes a unified multitask learning framework to jointly solve Weakly Supervised Semantic Segmentation and Saliency Detection using a single network. This method is called Saliency and Segmentation Network (SSNet) [133]. SSNet consists of two parts: Segmentation Network (SN) and Saliency Aggregation Module (SAM) [133]. For an input image, SN generates segmentation results and SAM predicts the saliency of each category.

We first describe [133] in section 4.1.1, and then describe how we incorporate volumetric loss in section 4.1.2.

### 4.1.1 Method Description

Saliency Detection (SD) identifies the important objects in an image [133]. Given the advances of CNNs this method has been heavily used for both saliency detection and semantic segmentation. Both these tasks involve generating pixel level masks, and thus have close connections. As the name suggests, this paper depends heavily on Saliency Maps.

#### Saliency Maps

Given an image, the Saliency Map of that image are the regions that gets noticed first. Saliency Segmentation is the process of identifying objects in an image as either being part of the background or the foreground [134]. Therefore the pixels in an image have two labels: background and foreground. Figure 4.1 shows the four images and their corresponding Saliency Segmentations. Black corresponds to background, and white corresponds to foreground [134]. It is a lot simpler and computationally cheaper to obtain a dataset of Saliency images compared to pixel level segmentation annotations. The first WSSS method [133] below heavily depends on Saliency Segmentation in creating a WSSS method.

Semantic segmentation and saliency detection both use CNN and both involve generating pixel level labels. Given a segmented output, the salient map can be derived by selecting objects as foreground, and the remaining pixel assumed to be background. The sliency map can be used to reduce the computational load of generating semantic segmentation of an image. A very common approach is to use CAMs to find the objects in an image, and SD to find the background.

Most WSSS methods use the results from pretrained SD models as a preprocessing step to create annotations for the WSSS models. However, these methods over look the connection between WSSS and SD, preventing them from fully exploiting segmentation cues of saliency annotations. This paper takes this interaction into account to create an end to end method to solve both WSSS and SD jointly [133].

Figure 4.2 shows the comparison between a WSSS method using only image level annotation versus the method proposed in the paper [133]. The method uses pixel level saliency information with image level category labels. Image level information help a CNN recognize semantic categories, but in terms of spatial distribution, the inference is coarse. Figure 4.2 (b) shows the output of using just image level annotation. As previously mentioned the proposed network SSNet is made up of two components, segmentation network (SN) and saliency aggregation module (SAM). Given an input image, SN generates the

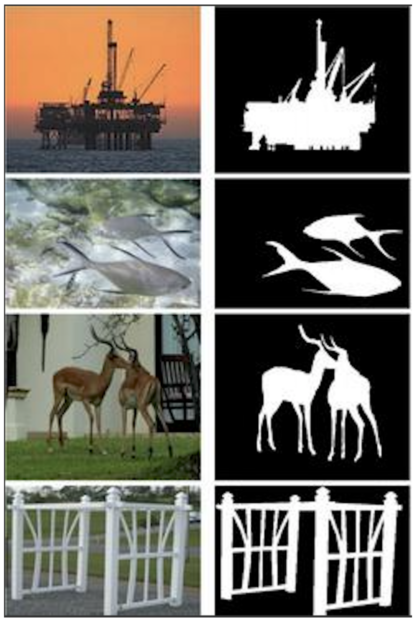


Figure 4.1: Saliency Segmentation. Black corresponds to background, and white corresponds to foreground

segmentation, shown in 4.2 (b). SAM predicts a saliency score for each semantic class and aggregates the segmentation map into a single channel saliency map according to the saliency score of each class, as shown in 4.2 (d). They both combine to give the result in 4.2 (c).

There are two variations of SSNet: SSNet-1, and SSNet-2 [133]. SSNet-1 is trained with pixel level saliency and image level annotation. SSNet-2 is trained with pixel level saliency, image level annotation and semantic segmentation results from SSNet-1. For the purposes of this thesis, we only focus on SSNet-1.

Figure 4.3 shows an overview of SSNet-1. Both the SN and the SAM are highlighted. The SN consists of a feature extractor to extract features from the input image and several convolution layers to predict segmentation results given the features. The SN follows the VGG-16 architecture, with 5 convolution layers. The fully connected classifier is removed and the convolution blocks are used as a feature extractor. To obtain larger feature maps, the down sampling operator is removed from the last two convolution blocks and dilated convolution is used to retain the original receptive field. The feature extractor generates feature maps of 1/8 the input image size. Thus the input images are resized to $256 * 256$, so the resulted feature maps are $32 * 32$ in size [133].

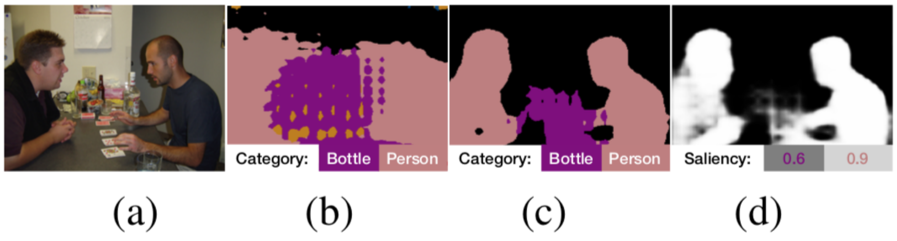


Figure 4.2: (a) input image. (b) segmentation results predicted by the model trained with only image-level labels. (c) segmentation results predicted by paper. (d) saliency map predicted paper [133]

A $1 \times 1$ convolution layer is used for generating segmentation results. The predicted C-channel segmentation map and one channel background map are 1/8 the input image size, in which C is the number of semantic classes. Each element of the segmentation map and background map is a value in [0, 1]. The values of all classes sum to 1 for each pixel. Then the segmentation results are upsampled by a deconvolution layer to the input image size.

SAM takes the $32*32$ outputs F of the feature extractor, and generates a C-dimensional vector $v$ with a $32*32$ convolution layer and a sigmoid function, of which each element $v_i$ is the saliency score of the $i$-th category. Then the saliency map $S$ is given by a weighted sum of the segmentation masks of all classes, where $H_i$ is the $i$-th channel of the segmentation results:

$$S = \sum_{i=1}^{C} v_i \cdot H_i \tag{4.1}$$

Training is done using two datasets [133]:

1. Saliency Dataset $D_s = (X^n, Y^n)_{n=1}^{N_s}$ where $X^n$ is the image and $Y^n$ is the ground truth, where each element is either 1 or 0 corresponding to salient object or background respectively
2. Segmentation Dataset $D_c = (X^n, t^n)_{n=1}^{N_c}$ where $X^n$ is the image and $t^n$ is the one hot encoding of the categories of the image.

As mentioned, for an input image, SN generates the segmentation results. The probability of each category can be derived by averaging the segmentation results over spatial

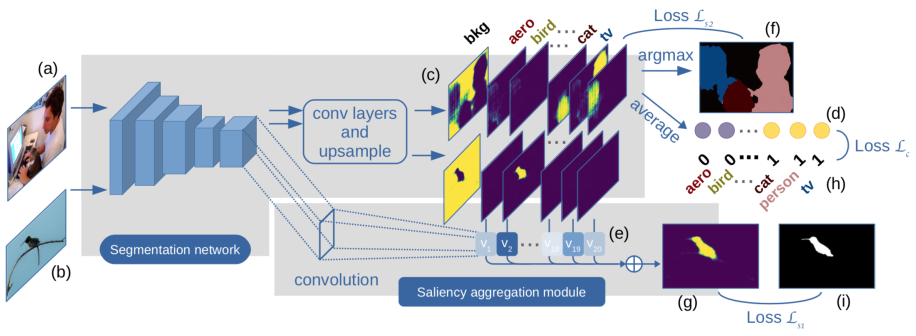


Figure 4.3: The proposed method trained with (a) images annotated with category labels and (b) images with saliency ground-truth. For an input image, the segmentation network generates a (c) segmentation results, of which the average over spatial locations indicates (d) the probability of each category. The saliency aggregation module predicts (e) saliency score of each category to aggregate segmentation masks of all categories into a (g) saliency map. In the first training stage, the network is trained with the (h) category labels and (i) saliency ground truth [133]

locations. This loss, denoted as $L_c$ is defined as:

$$L_c = -\frac{1}{N_c}\sum_{n=1}^{N_c}[\sum_{i=i}^{C} t_i^n \log \hat{t}_i^n + (1 - t_i^n)\log(1 - \hat{t}_i^n)], \tag{4.2}$$

in which:

- $t_i^n$ is the $i$-th element of $t^n$. $t_i^n$ is either 1 or 0, representing image $X^n$ containing the $t$-th category or not, respectively.
- $\hat{t}^n$ is the average over spatial positions of the segmentation maps $H^n$ of image $X^n$, of which each element $\hat{t}_i^n \, \epsilon \, [0, 1]$ represents the predicted probability of the $i$-th class objects existing in the image.

This equation allows to recognize semantic categories. Spatial information is discovered using pixel level saliency data. SAM generates saliency score of each category. A loss function is used so that the segmentation network has to precisely cut the recognized objects to make the derived saliency maps match the ground-truth. This loss, denoted as $L_s$ between the derived saliency maps and ground truth is defined as:

$$L_{s1} = -\frac{1}{N_s}\sum_{n=1}^{N_s}[\sum_{m} y_m^n \log s_m^n + (1 - y_m^n)\log(1 - s_m^n)], \tag{4.3}$$

in which:

- $y_m^n$ is the value of the $m$-th pixel of the saliency ground truth $Y^n$. $y_m^n \ \epsilon \ [0, 1]$
- $s_m^n$ is the value of the $m$-th pixel in the saliency map of image $Y^n$. $s_m^n \ \epsilon \ [0, 1]$

SSNet-1 is trained with the two losses defined above. A trained SSNet-1 is run on the classification dataset to obtain C + 1 channel segmentation results [133]. (C + 1 is the number of segmentation classes, and the background). The channels are cross multiplied with the one hot class label $t^n$ to suppress wrong predictions and refined CRF to enhance smoothness [133].

### 4.1.2 Adapting for Volumetric Supervision

The method in [133] is straightforward to adapt for volumetric supervision. The network used in [133], see figure 4.3, produces semantic segmentation output to which we can directly apply our volumetric loss functions developed in Chapter 3. In particular, the network produces output $t$, see the upper right branch in figure 4.3, where $t_i^p$ is the probability of pixel $p$ to be of class $i$. Thus, as described in chapter 3, we can approximate the size of class $i$ by averaging $t_i^p$ over all image pixels $p$, obtaining the size estimate $\hat{t}_i$.

The original method uses the sum of two losses $L_c$ and $L_{s1}$ in SSNet-1. These losses are used to optimize the model. To implement volumetric loss, at each iteration, we calculate the value of the volumetric loss function using $\hat{t}_i$ and the size of the objects using the methods described in chapter 3. The result is then added to the two original losses $L_c$ and $L_{s1}$. The method now uses this updated loss value to optimize the model.

Finally, we add an additional loss function which is not related to the size of an object. This component we add to the overall loss is for classes that are not present in the image, penalizing any pixel $p$ that is assigned to that class. We call this *Negative Loss*, as it penalizes any pixel which is assigned to some class not present in the image. This loss is defined as:

$$L_N(t) = \sum_{p=1}^{P}\sum_{i=1}^{C} y_i[-\log(1 - t_i^p + \epsilon)], \tag{4.4}$$

where epsilon is a small constant for numerical stability, (in practice we use $\epsilon = 1^{-6}$), the output $t_i^p$ is the prediction from the network run through softmax and $y_i$ is 0 if the class $i$ is present and 1 otherwise. This loss function is used in conjunction to one of the two volumetric loss functions described in chapter 3.

Notice the difference and similarities of our negative loss to the loss in Eq. 4.2. For classes not present in the image, the loss function in Eq. 4.2 penalizes any pixels assigned to that class. However, the penalty in Eq. 4.2 works on the image level. That is, for any class $i$ not present in the image, the network first averages predictions for that class in variable $\hat{t_i^n}$, and then calculates a penalty. Thus, it is still possible to predict a small number of pixels that may belong to the class not present in the image, since predictions are averaged over the whole image. Our negative loss is pixel precise, meaning any pixel assigned to the class not present in the image incurs a loss, without averaging predictions over the whole network first. Results of this method with Volumetric Supervision is shown in chapter 5.3.1.

## 4.2 Single Stage Semantic Segmentation from Image Labels

This paper [6] performs WSSS in a single stage by exploring three properties of a WSSS method: local consistency, semantic fidelity, and completeness [6]. Using these properties as guidelines, this paper develops a segmentation-based network model and a self-supervised training scheme to train for semantic masks from image level annotations in a single stage.

We first describe [6] in section 4.2.1, and then describe how we incorporate volumetric loss in section 4.2.2.

### 4.2.1 Method Description

Weakly supervised segmentation from image labels is more challenging compared to other weakly supervised methods such as bounding boxes and scribbles because of the complete lack of localization cues. Attention mechanisms, such Class Activation Maps (CAM) [137] are used as they offer a partial solution. CAMs localise the most important regions in the image using a pretrained classification network.

Figure 4.4 shows the three properties and short comings of most WSSS [6]:

1. Local Consistency. Two areas in close proximity with similar appearance maybe assigned different classes. Thus local consistency implies that neighbouring pixels with similar appearance share the same label.

2. Completeness. Attention maps like CAMs often do not cover the whole extent of the object. Completeness means that model identifies all visible class occurrences in the image

3. Semantic Fidelity. While the area of the attention maps dominates for the correct object class, parts of the map may still be mislabelled i.e., are semantically inaccurate. Semantic fidelity is exhibited by models producing segmentation masks that allow for reliable classification decisions

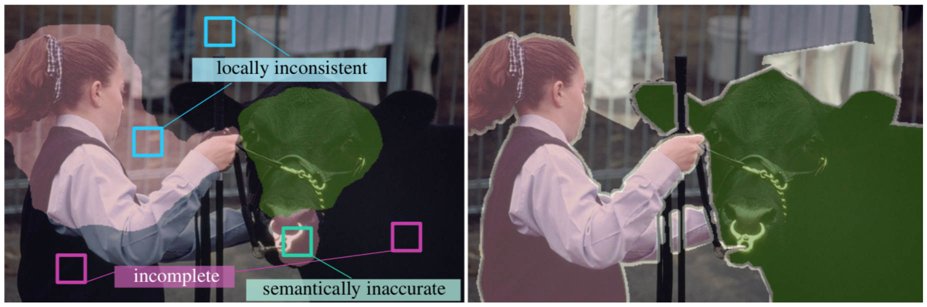


Figure 4.4: Typical shortcomings of attention maps (left) vs. pixel level annotations (right). Three defining properties of segmentation: local consistency, completeness and semantic fidelity [6]

Since classification requires only sufficient evidence, CAMs ensure neither completeness nor local consistency [6]. Using these properties, the paper proposes a model that significantly outperforms CAMs in terms of segmentation accuracy. First there is a normalized global weighted pooling, for computing classification scores and allowing concurring training for segmentation task. Secondly a Pixel adaptive mask refinement was implemented to make sure masks heed appearance cues. Finally, the paper introduces Stochastic Gate that counters the compounding effect of inaccuracies in the pseudo masks. Stochastic gate mixes feature representations with varying field sizes.

Figure 4.5 shows the model architecture. The method follows the established design of a fully convolutional segmentation with a softmax and skip layers. The figure also shows the three new additions [6]:

1. New class aggregation function
2. Pixel Adaptive mask refinement module
3. Stochastic gate

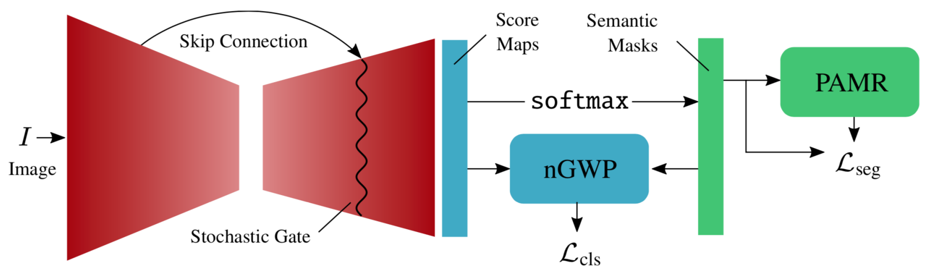


Figure 4.5: Model Overview [6]

The purpose of the class aggregation function is to provide semantic fidelity. This function leverages segmentation masks for classification decisions. To address this, the paper proposes normalized Global Weighted Pooling (nGWP) [6]. nGWP utilizes pixel level confidence predictions for relative weighting of the corresponding classification scores. Additionally focal mask penalty is incorporated into the classification scores to ensure completeness.

Pixel Adaptive Mask Refinement (PAMR) [6] is used to comply with local consistency, which revises the coarse mask predictions with respect to appearance cues.

Stochastic Gate (SG) [6] is the final component. Masks from PAMR might contain deviations from the ground truth and supervised learning can further compound these errors and overfit. SG combines deep feature representation responsible for this event with more robust but less expressive shallow features in a stochastic way.

### 4.2.2 Adapting for Volumetric Supervision

This method is also straightforward to adapt to volumetric supervision. The model in Figure 4.5 outputs per-pixel class prediction $x$. Their per-pixel class predictions are passed through a softmax layer, which normalizes prediction for each pixel over the possible classes to be between 0 and 1. This produces the normalized (between 0 and 1) class predictions $t$. Then we compute $\hat{t}_i$ for each class $i$ and apply volumetric loss functions as described in chapter 3.

The original method uses one loss $L_{cls}$ to optimize the model. To implement volumetric loss, at each epoch, we calculate the value of the volumetric loss function using the score maps and the size of objects using the methods described in chapter 3. The result is then added to the original loss $L_{cls}$. The method now uses the updated loss value to optimize the model.

Similar to the previous method, we add Negative Loss, Eq. 4.4 to the overall loss. Results of this method with Volumetric Supervision is shown in chapter 5.3.2.

## 4.3 Weakly Supervised Semantic Segmentation via Sub-category

CAMs [137] are most responsive to the most discriminative parts of an object. Chang et al. [17] focuses on enforcing the network to pay attention to other parts of an object and thus improving the quality of the response maps. This enforcement is done by performing clustering on image features to generate pseudo sub-categories labels within each annotated parent class, and construct a sub-category objective to assign the network to a more challenging task [17]. By iteratively clustering image features, the training process does not limit itself to the most discriminative object parts, hence improving the quality of the response maps [17].

We first describe [17] in section 4.3.1, and then describe how we incorporate volumetric loss in section 4.3.2.

### 4.3.1 Method Description

The two previous methods we discussed in sections 4.1 and 4.2 were single stage methods. The method of Change et al. consists of two main stages, first constructing pseudo ground truth, and then training a standard semantic network using the pseudo ground truth instead of ground truth.

This method [17] focuses on the first step, improving the quality of the constructed pseudo-ground truth. Currently CAMs [137] are used for initial localization of a class. However because CAMs are trained with classification, rather than localization objective, they tend to focus on the most discriminative part of the object instead of localizing the entire thing. This leads to poor accuracy for the pseudo-ground truth. Thus Chang et al. proposes an unsupervised method for the network to learn better representations by discovering subcategories [17]. They accomplish this in two steps:

1. Perform clustering on image features of the parent class, i.e., the actual classes present in the dataset
2. Use clustering from each image as pseudo labels to optimize the subcategory objective

The method starts with a baseline mode achieved using CAMs [137]. As previously mentioned, CAMs only highlight the important parts of an image, which is not sufficient for semantic segmentation. This issue is addressed using the subcategory learning to discover more object parts.

For each parent class, K subcategories are determined by using KMeans clustering [17]. Each image is assigned a pseudo label, which is the index of the subcategory. Then a subcategory object is created to jointly train the network. By iteratively upgrading the feature extractor, classifiers and subcategory pseudo labels, the enhanced feature representation leads to better response maps that cover more complete regions of the object [17].

The method uses CAM to generate an initial response. Chang et al. uses a typical network: first convolution layers for feature extraction, followed by global average pooling and finally a fully connected layer producing the output. For each parent class $P_c$ there are K subcategories $s_c^k$, where $k = \{1, 2, 3, ..., K\}$. Each image I has parent label $Y_p^c$ in $\{0, 1\}$ where $c$ corresponds to the list of classes. The subcategory for each category $c$ is denoted as $Y_s^{c,k}$ in $\{0, 1\}$. The parent class has a classifier $H_p$ and a classification loss $L_p$. The subcategory class has classifier $H_s$ with parameters $\theta_s$ and a standard multi classification loss $L_s$ with a larger label space $Y_s$. The objective is to learn $H_s$ while sharing the same features with $H_p$.

Since there is no ground truth label for subcategory objective loss, these are generated using unsupervised clustering. The clustering for each class $c$ is:

$$\min_{D \epsilon \mathbb{R}^{d*k}} \frac{1}{N^c} \sum \min_{Y_s^c} ||f - TY_s^c||_2^2, \tag{4.5}$$

where:

- $T$ is a $D * K$ centroid matrix
- $N^c$ is the number of images with class $c$
- $f$ is the extracted feature
- $Y_s^c$ for each image is the subcategory pseudo label for optimizing $L_s$

After getting the subcategory labels $Y_s$, the feature representations and the two classifiers are jointly optimized in the following equation:

$$\min_{\theta_p,\theta_s} \frac{1}{N^c} \sum_{i=1}^{N} L_p(H_p(f_i), Y_p) + \lambda L_s(H_s(f_i), Y_s), \tag{4.6}$$

where:

- $N$ Number of images
- $\lambda$ is the weight to balance the two functions

The parent classifier learns the feature space through supervised training, and the the subcategory provides more gradients to enhance feature representations. Figure 4.6 shows the entire model architecture. Resnet 38 is used as the backbone to generate the activation maps [17]. First the input image features are extracted. Then unsupervised clustering is performed on the features to get the subcategory pseudo labels $Y_s$. Next joint training is done to optimize both parent and subcategory classifier. Finally, iteratively performing unsupervised clustering and pseudo training the classification module, the optimized classification network is used to produce the final activation map.

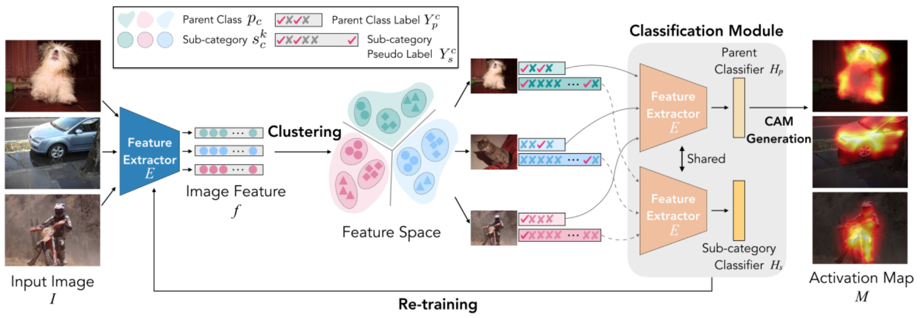


Figure 4.6: Model architecture [17]

## 4.3.2 Adapting for Volumetric Supervision

The model, shown in figure 4.6, shows two feature extractors outputting to the parent classifier, $H_p$ and the sub-category classifier $H_s$. The outputs of both these classifiers are used to calculate the respective losses $L_p$ and $L_s$. For the purposes of this thesis we only use the output of the parent classifier. Taking first the softmax and then the mean of the output of $H_p$ gives us the average over spatial positions of the segmentation map of each image. This is denoted as $\hat{t}_i$ which is used in the volumetric functions in chapter 3.

The original method uses the sum of the two losses $L_p$ and $L_s$ to optimize the model. To implement volumetric loss, at each epoch we calculate the value of the volumetric loss function using the output of the parent classifier $H_p$ and the size of objects using the methods described in chapter 3. The result is then added to the sum of the two original losses $L_p$ and $L_s$. The method now uses the updated loss value to optimize the model.

Similar to the previous methods, we add Negative Loss, Eq. 4.4 to the overall loss. As mentioned, this method has two stages, constructing pseudo ground truths, and then training a standard semantic network using the pseudo ground truth instead of ground truth. In this thesis, we only evaluate the first stage of the method in [17], namely the accuracy of the pseudo ground truth constructed. We do not proceed to the second stage, training on the pseudo-ground truth. This is because the more accurate is the pseudo ground truth, the more accurate the final results (those of the second stage) will be. Thus, it is sufficient to evaluate if we can obtain more accurate pseudo ground truths. Results of this method with Volumetric Supervision is shown in chapter 5.3.3.

## 4.4 Weakly Supervised Semantic Segmentation with Boundary Exploration

This paper [22] proposed a network called BENet to explore the idea of using object boundaries in WSSS. The approach is based on the idea of explicitly exploring object boundaries from training images to keep coincidence of segmentation and boundaries [22]. Boundary annotations are created by exploiting coarse localization maps obtained from CNN classifier, and annotations are used to train BENet which further excavates more object boundaries to provide constraints for segmentation [22]. The pseudo annotations generated are used to train an off the shelf segmentation model.

We first describe [22] in section 4.4.1, and then describe how we incorporate volumetric loss in section 4.4.2.

### 4.4.1 Method Description

Similar to the previous method [17], this method [22] consists of two main stages, first constructing pseudo ground truth, and then training a standard semantic network using the pseudo ground truths instead of ground truths.

Classification requires translational invariance. But semantic segmentation requires translational variance, that is it should be sensitive to the position of the class of interest in the image [22]. CAMs are proposed to overcome this difference by using a global average pooling [137]. However, CAMs activate the more important regions and thus obtain incomplete segmentation results. Most methods thus focus on getting better foreground regions, instead of focusing on an objects boundaries.

Object boundaries are essential to segmentation performance, but is difficult to detect without proper annotations. This paper proposes boundary exploration based segmentation (BES) [22]. BES is done in three steps. First a simple scheme is used to obtain a small amount of boundary labels by filtering localization maps. Second, BENet [22] trained by synthetic boundary labels is designed to explore more object boundaries. Finally, massive explored boundary information is used to provide constraints for localization maps propagation.

Figure 4.7 shows the model architecture of BENet. The figure highlights the two key components: Boundary Exploration and Attention pooling CAM.

First coarse localization maps are obtained through the attention-pooling CAM. Attention-pooling CAM is an improved CAM mechanism to obtain better initial object localization maps. Fully convolutional network (FCN) architecture of CAM is used to capture object's spatial information. The only modification is changing the calculation for obtaining classification scores by applying attention pooling which can dynamically assign different weights per pixels.

Boundary labels are synthesized to train BENet which is able to excavate more object boundaries [22]. Boundary exploration targets to predict precise boundary maps with the original training images as input. Boundary labels are manually synthesized from localization maps and used to train BENet to predict boundaries [22]. The boundary classification ability of BENet helps to explore massive boundaries which are then used for revising localization maps [22].

Finally, semantic segmentation is generated by applying predicted boundary information to revise localization maps [22].

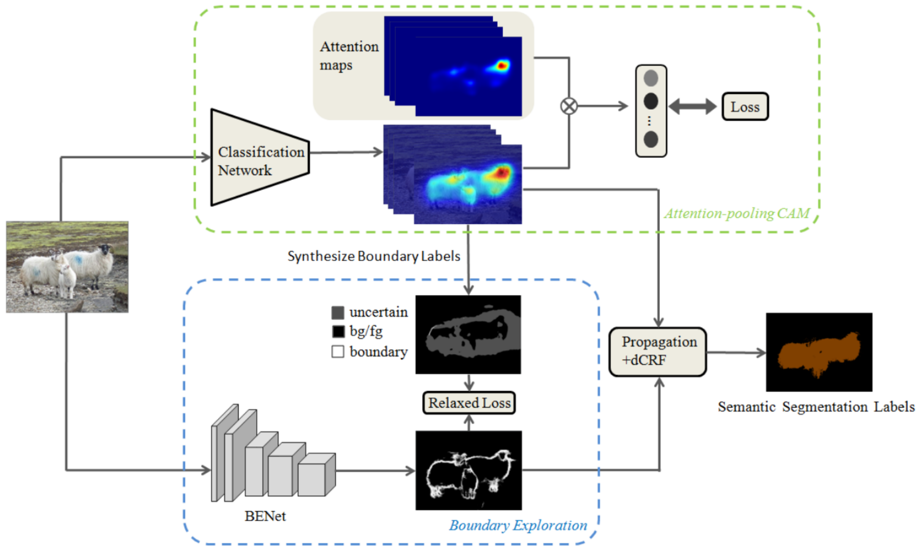


Figure 4.7: Model architecture[22]

### 4.4.2 Adapting for Volumetric Supervision

The output of BNet as shown in figure 4.7 is used to calculate the losses. Similar to the previous methods, taking first the softmax of the output and then the mean gives us the average over spatial positions of the segmentation map of each image. This average is denoted as $\hat{t}_i$ which is used in the volumetric functions in chapter 3.

The original method uses one loss $L_B$ to optimize the model. To implement volumetric loss, at each epoch we calculate the value of the volumetric loss function using the output of BNet and the size of objects using the methods described in chapter 3. The result is then added to the original loss $L_B$. The method now uses the updated loss value to optimize the model.

Similar to the previous methods, we add Negative Loss, Eq. 4.4 to the overall loss. Similar to the previous method [17], there are two stages, and in this thesis, we only evaluate the first stage of the method in [22], namely the accuracy of the pseudo ground truth constructed. We do not proceed to the second stage, training on the pseudo ground truth. This is because the more accurate the pseudo ground truth, the more accurate the final results (those of the second stage) will be. Thus, it is sufficient to evaluate if we can obtain more accurate pseudo ground truths. Results of this method with Volumetric Supervision is shown in chapter 5.3.4.

# Chapter 5

# Experimental Evaluation

In this section, we evaluate our volumetric loss supervision using the four methods discussed in Chapter 4. Since currently, there is no real dataset with volumetric annotations collected from the users, we first create a simulated dataset with volumetric annotation from the most commonly used benchmark for semantics segmentation, the Pascal VOC 2012 dataset [36, 35, 34]. The creation of the simulated dataset is outlined in section 5.1. Next, in section 5.2, we discuss our main evaluation measure, mean over intersection (mIOU). In section 5.3, we discuss the results for the Quadratic loss and Outside Quadratic loss. In section 5.4, we measure the performance of our methods assuming there is error in user annotation. Finally, in section 5.5, we develop a method for estimating class sizes directly from image-level annotations, without asking user to provide volumetric annotations.

## 5.1 Semantic Segmentation Dataset with Volumetric Annotation

We propose a new type of weakly supervised semantic segmentation, namely volumetric supervision. For volumetric supervision, one needs not only the object classes present in the image, but also a rough size of each object class. Currently, there is no real dataset with volumetric information for semantic segmentation available. Therefore, we simulate a dataset with volumetric information from an already existing semantic segmentation dataset with full ground truth available. We use the most common such dataset, namely the Pascal Visual Object Class (VOC) 2012 dataset [36, 35, 34]. We use the full ground truth just to construct a simulated volumetric annotations. We do not use full ground truth when training our weakly supervised methods.

### 5.1.1 Pascal VOC 2012 Dataset

The Pascal Visual Object Class (VOC) Challenge 2012 is an image recognition challenge that has run since 2005. The main goal is to recognize objects from a number of classes. The challenge is a supervised learning problem. There are twenty object classes. These classes are grouped into four categories [36, 35, 34]:

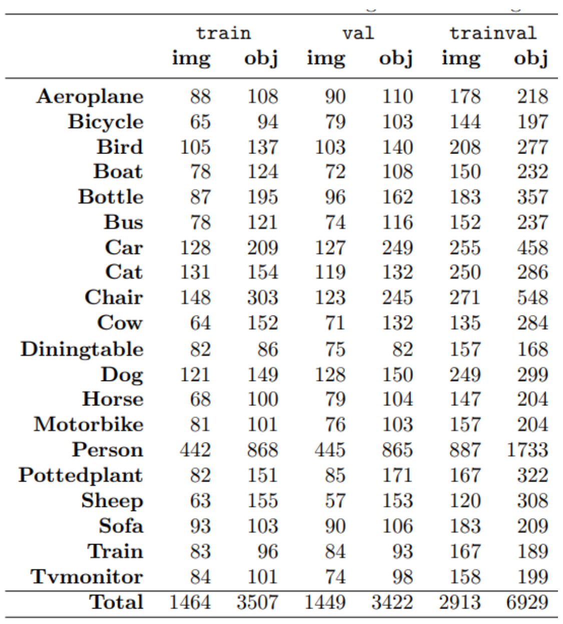
1. Person: person
2. Animal: bird, cat, cow, dog, horse, sheep
3. Vehicle: aeroplane, bicycle, boat, bus, car, motorbike, train
4. Indoor: bottle, chair, dining table, potted plant, sofa, tv/monitor

There are three main object recognition competitions: classification/detection, segmentation and action classification [36, 35, 34]. For this thesis we focus on segmentation.

Table 5.1 shows the break down of images for training and validation for Pascal VOC 2012 dataset [36, 35, 34]. The dataset contains 20 classes, plus one extra for background. Label 255 is also used as void label. Void labels are ignored during training and testing. Void pixels are pixels that are difficult to annotate correctly. There are 1464 training images with 3507 objects and 1449 validation images with 3422 objects [36, 35, 34]. Figure 5.2 shows an example image and its respective segmented ground truth.

### 5.1.2 Pascal VOC 2012 Augmented Dataset

In accordance with all the WSSS methods, we augment the dataset with additional images from the Semantic Boundaries Dataset (SBD) [46]. SBD was created for the task of predicting semantic contours rather than semantic segmentation. SBD has 11355 image annotations [46]. There are 8498 training images, and 2857 validation images [46]. These images are from the Pascal VOC 2011 dataset, for the 20 classes. Table 5.3 shows the breakdown of the number of objects in images per class for both training and validation for the Semantic Boundaries Dataset.



| | train | | val | | trainval | |
|---|---|---|---|---|---|---|
| | **img** | **obj** | **img** | **obj** | **img** | **obj** |
| **Aeroplane** | 88 | 108 | 90 | 110 | 178 | 218 |
| **Bicycle** | 65 | 94 | 79 | 103 | 144 | 197 |
| **Bird** | 105 | 137 | 103 | 140 | 208 | 277 |
| **Boat** | 78 | 124 | 72 | 108 | 150 | 232 |
| **Bottle** | 87 | 195 | 96 | 162 | 183 | 357 |
| **Bus** | 78 | 121 | 74 | 116 | 152 | 237 |
| **Car** | 128 | 209 | 127 | 249 | 255 | 458 |
| **Cat** | 131 | 154 | 119 | 132 | 250 | 286 |
| **Chair** | 148 | 303 | 123 | 245 | 271 | 548 |
| **Cow** | 64 | 152 | 71 | 132 | 135 | 284 |
| **Diningtable** | 82 | 86 | 75 | 82 | 157 | 168 |
| **Dog** | 121 | 149 | 128 | 150 | 249 | 299 |
| **Horse** | 68 | 100 | 79 | 104 | 147 | 204 |
| **Motorbike** | 81 | 101 | 76 | 103 | 157 | 204 |
| **Person** | 442 | 868 | 445 | 865 | 887 | 1733 |
| **Pottedplant** | 82 | 151 | 85 | 171 | 167 | 322 |
| **Sheep** | 63 | 155 | 57 | 153 | 120 | 308 |
| **Sofa** | 93 | 103 | 90 | 106 | 183 | 209 |
| **Train** | 83 | 96 | 84 | 93 | 167 | 189 |
| **Tvmonitor** | 84 | 101 | 74 | 98 | 158 | 199 |
| **Total** | 1464 | 3507 | 1449 | 3422 | 2913 | 6929 |

Figure 5.1: Table showing the breakdown of total images and images per class for segmentation task in the Pascal VOC 2012 dataset [36, 35, 34]

### 5.1.3 Train and Validation split

Following common practises in [133, 6, 17, 22], the original Pascal VOC training set is augmented with images from SBD. In total there are 10582 images used for training. Only image level labels are used. For validation, 1449 images from the Pascal VOC validation list are used. Volumetric annotation is created for each image for both training and validation sets.

### 5.1.4 Volumetric Annotations

We simulate volumetric user annotations as follows. First, for each image and for each class present in the image, we compute the number of pixels of that class present in the image, and normalize by the image size. Since we do not expect users to be able to give us precise object size, we then simulate imprecise user input by quantizing the size information into ten equally space buckets. Bucket $i$, for $1 \leq i \leq 10$ contains objects with size between $(i-1) * 10\%$ and $i * 10\%$ of the image size. If the object falls into bucket $i$, then its size is approximated by $i * 10\%$ of the image size.

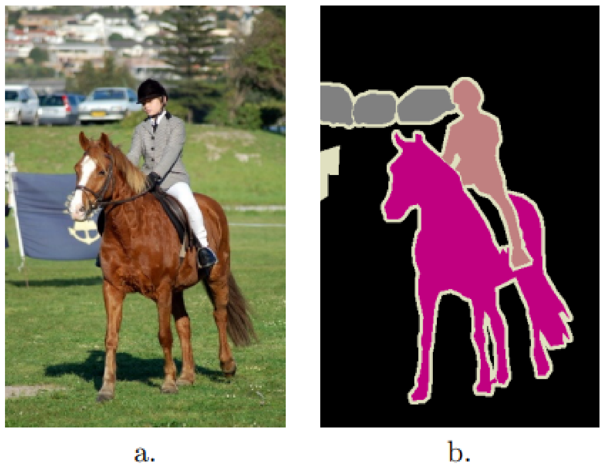
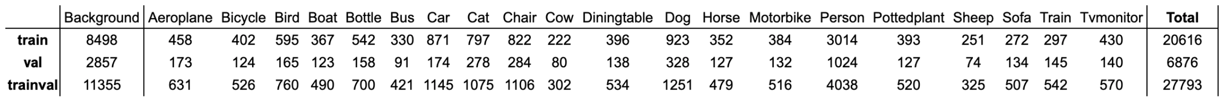


Figure 5.2: Segmentation dataset example. Image (a) is the training image. Image (b) is the segmentation ground truth showing the different classes: car, horse and person. Background is in black. The cream color represents 255 or void label [36, 35, 34]



| | Background | Aeroplane | Bicycle | Bird | Boat | Bottle | Bus | Car | Cat | Chair | Cow | Diningtable | Dog | Horse | Motorbike | Person | Pottedplant | Sheep | Sofa | Train | Tvmonitor | **Total** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **train** | 8498 | 458 | 402 | 595 | 367 | 542 | 330 | 871 | 797 | 822 | 222 | 396 | 923 | 352 | 384 | 3014 | 393 | 251 | 272 | 297 | 430 | 20616 |
| **val** | 2857 | 173 | 124 | 165 | 123 | 158 | 91 | 174 | 278 | 284 | 80 | 138 | 328 | 127 | 132 | 1024 | 127 | 74 | 134 | 145 | 140 | 6876 |
| **trainval** | 11355 | 631 | 526 | 760 | 490 | 700 | 421 | 1145 | 1075 | 1106 | 302 | 534 | 1251 | 479 | 516 | 4038 | 520 | 325 | 507 | 542 | 570 | 27793 |

Figure 5.3: Table showing the breakdown of objects in images per class for both training and validation for the Semantic Boundaries Dataset

## 5.2 mean Intersection over Union (mIoU)

Before moving to the results section, we introduce the metric used to measure the performance of the methods. The performance of a semantic segmentation models on the Pascal VOC 2012 dataset are measured by mean Intersection over Union (mIoU) [99]. mIoU is calculated by first getting the Intersection over Union (IoU) for each of the 20 classes at a pixel level [56]. The equation to calculate IoU is as follows:

$$IoU = \frac{T_p}{T_p + F_n + F_p} * 100\% = \frac{overlap}{union}, \tag{5.1}$$

where:

- $T_p$ stands for true positive. These are correctly predicted pixels that belong to the class

- $F_n$ stands for false negative. These are incorrectly predicted pixels that belong to the class
- $F_p$ stands for false positive. These are pixels that do not belong to the class but are predicted as in the class

Figure 5.4 shows the original image and the ground truth of segmentation result in the first two images respectively. The third image is the segmentation generated by the model. The difference between the ground truth and prediction is shown in the fourth image, which represents the three components of IoU. Yellow is $T_p$, Green is $F_n$ and Red is $F_p$ [51].

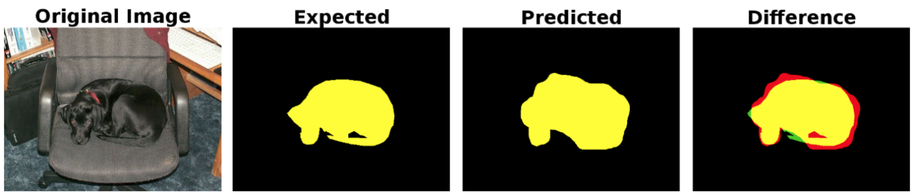


Figure 5.4: The input image, ground truth and model segmentation prediction. Difference between the ground truth and prediction is the last image. Yellow is $T_p$, Green is $F_n$ and Red is $F_p$ [51]

Figure 5.5 shows the visual results of IoU. When IoU is 0, it means the ground truth and prediction have no overlap. IoU of 1 means the ground truth and prediction perfectly overlap each other. IoU of $\frac{1}{7}$ means only a $7^{th}$ of the prediction matches the ground truth. Once we have the IoU of all the 20 classes, the mean of all these values are taken and reported as the mIoU. mIoU is a value from 0 to 100. The higher the value, the more accurate the segmentation.

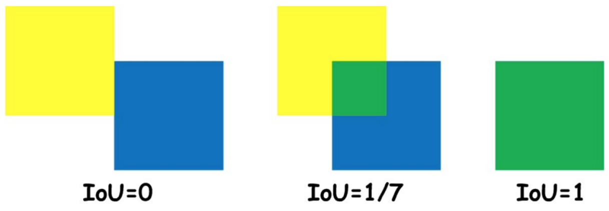


Figure 5.5: Visual representation of IoU in terms of ground truth and prediction [107]

## 5.3 Results for Quadratic loss and Outside Quadratic loss

In this section, we perform two experiments using the Volumetric Annotations described in section 5.1.4. We perform the tests using both **Quadratic Volumetric Loss** and **Outside Quadratic Volumetric Loss**. For Outside Quadratic Volumetric Loss, the endpoints $a$ and $b$ are the endpoints of the volumetric annotation buckets. The result for all four methods [133, 6, 17, 22] are shown below in terms of both mIoU and visually. For each test we also show the IoU of each individual classes.

### 5.3.1 Joint Saliency and Semantic Segmentation

The mIoU result of method [133] is 0.5420. Table 5.1 shows the mIoU values of using Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss with Volumetric Annotations on Joint Learning of Saliency Detection and Weakly Supervised Semantic Segmentation. For these two trials the individual class IoU as well as the original class IoU are shown in table 5.2. Figure 5.6 shows the visual results of using Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss with Volumetric Annotations. For an input image, we show its ground truth, the results without using Volumetric Loss and the results of QVL and OQVL.

| | mIoU |
|---|---|
| Original | 0.5420 |
| Quadratic Volumetric Loss | **0.6098** |
| Outside Quadratic Volumetric Loss | 0.5979 |

Table 5.1: mIoU results of using Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss with Volumetric Annotations on Joint Learning of Saliency Detection and Weakly Supervised Semantic Segmentation.

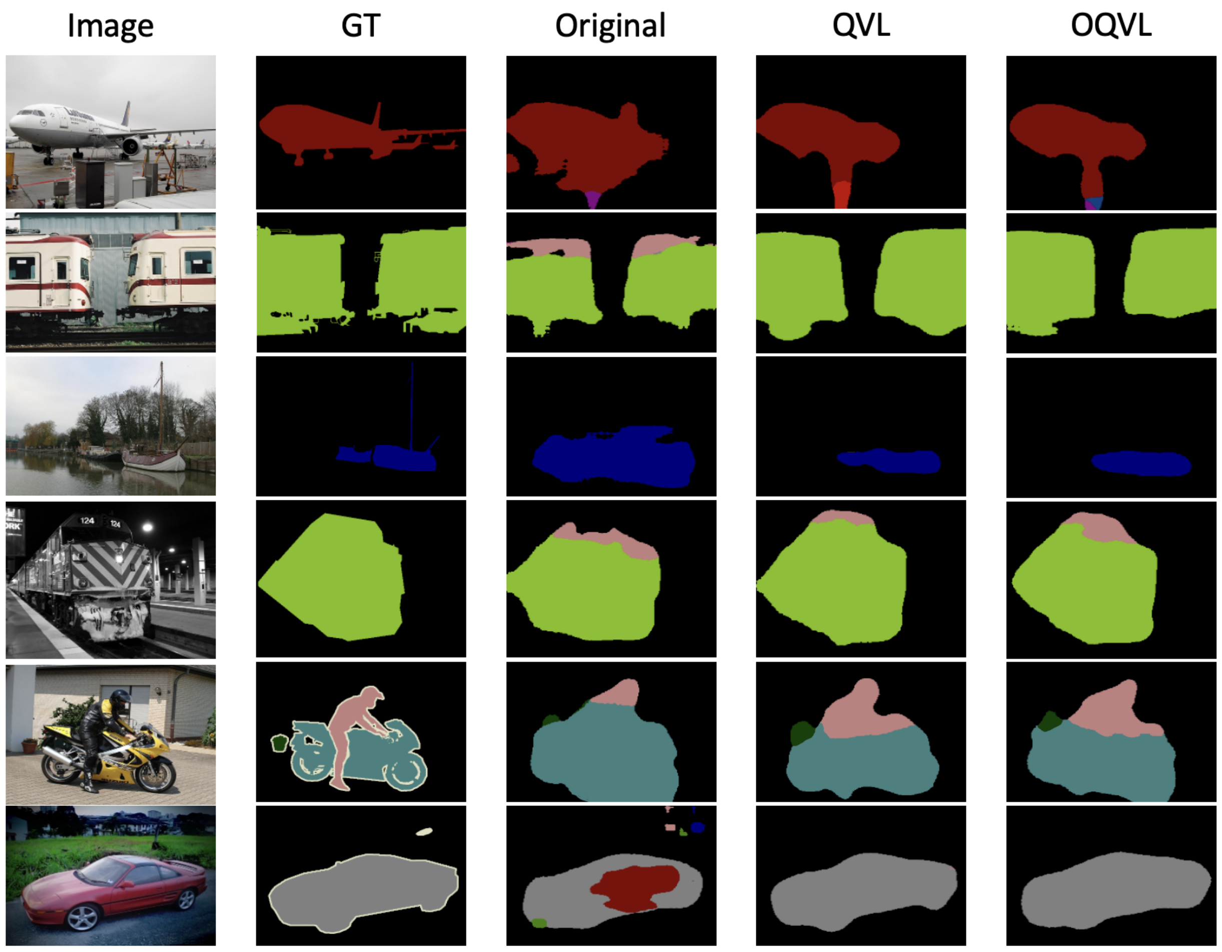


Figure 5.6: Visual results of using Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss with Volumetric Annotations on Joint Learning of Saliency Detection and Weakly Supervised Semantic Segmentation. For an input image, we show its ground truth, the results without using Volumetric Loss and the results of QVL and OQVL

| | IoU | | |
|---|---|---|---|
| **Class** | **Original** | **QVL** | **OQVL** |
| Background | 0.8866 | 0.9330 | 0.9211 |
| Aeroplane | 0.6660 | 0.7324 | 0.7205 |
| Bicycle | 0.3183 | 0.3674 | 0.3555 |
| Bird | 0.7019 | 0.7619 | 0.7500 |
| Boat | 0.5554 | 0.6302 | 0.6183 |
| Bottle | 0.6165 | 0.6590 | 0.6471 |
| Bus | 0.6885 | 0.7621 | 0.7502 |
| Car | 0.6123 | 0.6900 | 0.6781 |
| Cat | 0.7162 | 0.7860 | 0.7741 |
| Chair | 0.1306 | 0.1890 | 0.1771 |
| Cow | 0.5493 | 0.6474 | 0.6355 |
| Dinningtable | 0.2045 | 0.2572 | 0.2453 |
| Dog | 0.6706 | 0.7478 | 0.7359 |
| Horse | 0.5476 | 0.6308 | 0.6189 |
| Motorbike | 0.5851 | 0.6448 | 0.6329 |
| Person | 0.6732 | 0.7292 | 0.7173 |
| Pottedplant | 0.3419 | 0.4076 | 0.3957 |
| Sheep | 0.5872 | 0.6587 | 0.6468 |
| Sofa | 0.2671 | 0.3434 | 0.3315 |
| Train | 0.5682 | 0.6559 | 0.6440 |
| Tv/monitor | 0.4951 | 0.5730 | 0.5611 |
| **mIoU** | **0.5420** | **0.6098** | **0.5979** |

Table 5.2: IoU for each individual class for Original, Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss with Volumetric Annotations on Joint Learning of Saliency Detection and Weakly Supervised Semantic Segmentation

As we see from the results of the first experiment, introducing volumetric supervision with QVL has significantly increased performance, with an increase in mIoU of 6.78 %. Looking at the individual IoU of each of the class for the first experiment, we can see overall all the classes benefit from volumetric supervision, some more than others. Classes such as Cow, Train, Horse and Tv/monitor all show over 8% increase in mIoU, while others such as Bottle or Person show increases that are slightly less than the mean.

Using OQVL has resulted in a 5.59% increase in mIoU. This too has increased the performance of the method, although by not as big of a margin as the first loss. This observation is also evident in the IoU for each of the individual classes. The results from the first two experiments also show that Quadratic Volumetric Loss (QVL) performs slightly better than Outside Quadratic Volumetric Loss (OQVL).

This is somewhat against our expectations, as we expected Outside Quadratic loss to perform better, as it has no penalty if object class falls into the correct size bin. The reason might be that Quadratic loss may be easier to optimize, given that it is smooth and has nonzero gradient almost everywhere, unlike the Outside Quadratic loss, which has a large stretch of zero gradient.

Visually in figure 5.6, the results of Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss are quite similar. Compared to the original method, with volumetric supervision, there are much fewer pixels being assigned to classes not present in the image.

### 5.3.2 Single Stage Semantic Segmentation

The mIoU result of method [6] is 0.5818. Table 5.3 shows the mIoU values of using Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss with Volumetric Annotations on Single Stage Semantic Segmentation from Image Labels. For these two trials the individual class IoU as well as the original class IoU are shown in table 5.4. Figure 5.7 shows the visual results of using Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss with Volumetric Annotations. For an input image, we show its ground truth, the results without using Volumetric Loss and the results of QVL and OQVL.

| | mIoU |
|---|---|
| Original | 0.5818 |
| Quadratic Volumetric Loss | **0.6114** |
| Outside Quadratic Volumetric Loss | 0.6008 |

Table 5.3: mIoU results of using Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss with Volumetric Annotations on Single Stage Semantic Segmentation from Image Labels

As we see from the results of the first experiment, introducing volumetric supervision with QVL increases performance, although not as much as the previous method [133]. The first experiment resulted in an increase in mIoU of 2.96 %. Looking at the individual IoU of each of the class for the first experiment, we can see that not all the classes benefit from volumetric supervision. Classes such as Aeroplane, Bicycle, Potted plant and Sofa show great increase in mIoU, while others such as Bird, Tv/monitor, Train, show a decrease in individual IoU.

The second experiment using OQVL as the loss shows a slight decrease in mIou compared to the first one. It still has a 1.9% increase in mIoU over the original method.

Individual class IoU follow the same pattern as the first experiment. However, the overall results from the first two experiments match the pattern of the previous method and show that QVL performs slightly better than OQVL. Visually, in figure 5.7 we can see that the shape of objects tend to be more precise when the volumetric loss is used.

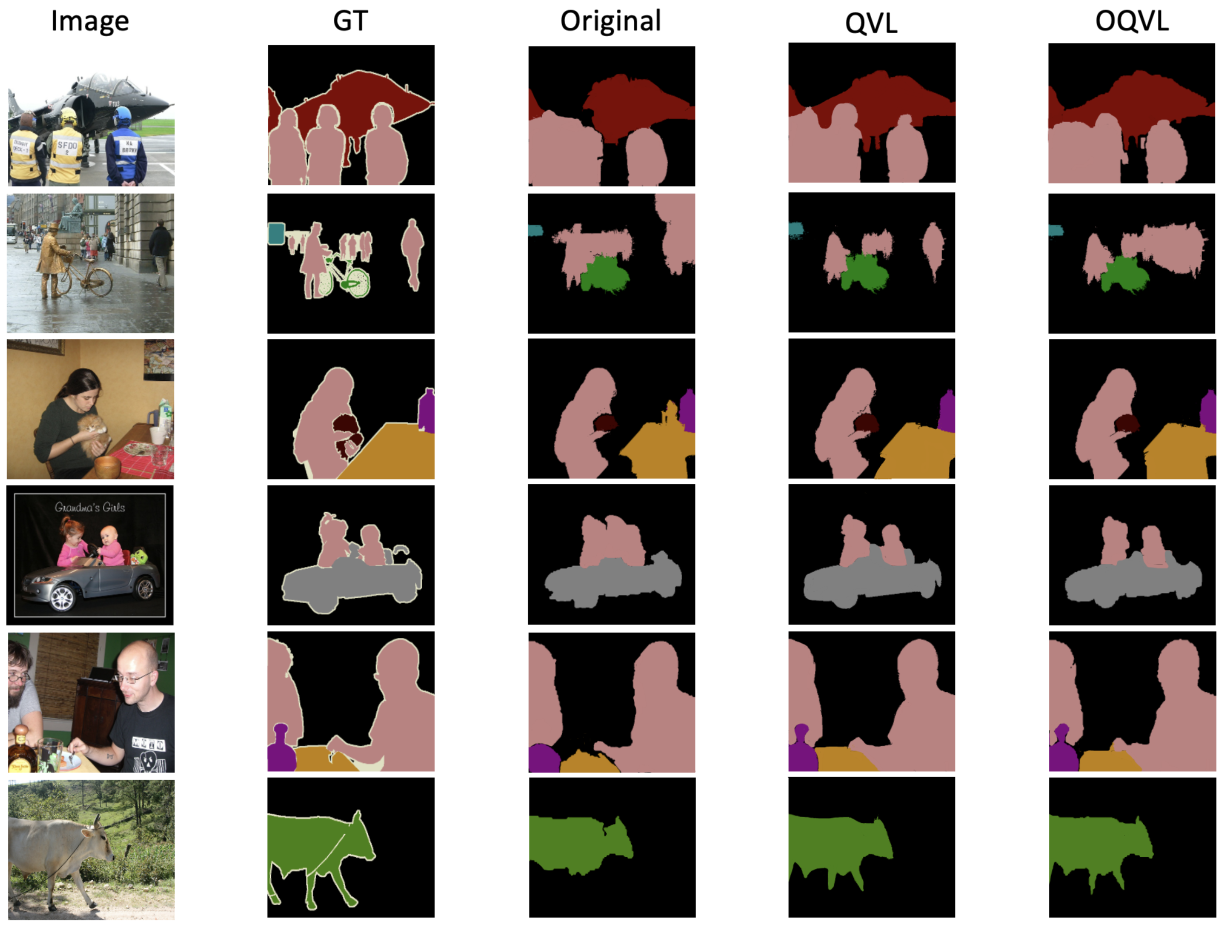


Figure 5.7: Visual results of using Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss with Volumetric Annotations on Single Stage Semantic Segmentation from Image Labels. For an input image, we show its ground truth, the results without using Volumetric Loss and the results of QVL and OQVL

| | IoU | | |
|---|---|---|---|
| **Class** | **Original** | **QVL** | **OQVL** |
| Background | 0.8621 | 0.8691 | 0.8685 |
| Aeroplane | 0.6162 | 0.7045 | 0.6935 |
| Bicycle | 0.3417 | 0.4892 | 0.4785 |
| Bird | 0.6829 | 0.6448 | 0.6335 |
| Boat | 0.4562 | 0.4990 | 0.4885 |
| Bottle | 0.6113 | 0.5993 | 0.5885 |
| Bus | 0.7078 | 0.7721 | 0.7615 |
| Car | 0.6363 | 0.6597 | 0.6485 |
| Cat | 0.7626 | 0.7762 | 0.7655 |
| Chair | 0.2748 | 0.3434 | 0.3325 |
| Cow | 0.6112 | 0.6698 | 0.6585 |
| Dinningtable | 0.2757 | 0.3290 | 0.3185 |
| Dog | 0.7383 | 0.7640 | 0.7495 |
| Horse | 0.6272 | 0.6762 | 0.6655 |
| Motorbike | 0.6785 | 0.6175 | 0.6065 |
| Person | 0.6848 | 0.6632 | 0.6525 |
| Pottedplant | 0.4221 | 0.5067 | 0.4955 |
| Sheep | 0.6713 | 0.7319 | 0.7205 |
| Sofa | 0.3444 | 0.4693 | 0.4585 |
| Train | 0.7011 | 0.6472 | 0.6365 |
| Tv/monitor | 0.5120 | 0.4066 | 0.3955 |
| **mIoU** | **0.5818** | **0.6114** | **0.6008** |

Table 5.4: IoU for each individual class for Original, Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss with Volumetric Annotations on Single Stage Semantic Segmentation from Image Labels

### 5.3.3 Semantic Segmentation via Sub-category

The mIoU result of method [17] is 0.4977. Table 5.5 shows the mIoU values of using Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss with Volumetric Annotations on Single Stage Semantic Segmentation from Image Labels. For these two trials the individual class IoU as well as the original class IoU are shown in table 5.6. Figure 5.8 shows the visual results of using Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss with Volumetric Annotations. For an input image, we show its ground truth, the results without using Volumetric Loss and the results of QVL and OQVL.

|  | mIoU |
|---|---|
| Original | 0.4977 |
| Quadratic Volumetric Loss | **0.5138** |
| Outside Quadratic Volumetric Loss | 0.5114 |

Table 5.5: mIoU results of using Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss with Volumetric Annotations on Weakly Supervised Semantic Segmentation via Sub-category Exploration

As we see from the results, introducing volumetric supervision with QVL slightly increases performance. The first experiment resulted in an increase in mIoU of 1.61%. Looking at the individual IoU of each of the class for the first experiment, we can see that all the classes benefit equally from volumetric supervision. Background, has had the biggest increase in IoU, with over 5% increase.

The second experiment using OQVL as the loss shows a slight decrease in mIoU compared to the first experiment. This loss still has a 1.37% increase in mIoU over the original method. Similar to the first experiment, all the classes benefit equally from the use of volumetric supervision. The overall results from the first two experiments match the pattern of previous methods and show QVL performs better than OQVL. Visually, from figure 5.8, we can see that our method labels pixels with classes not present in the image less often, compared to the original method.

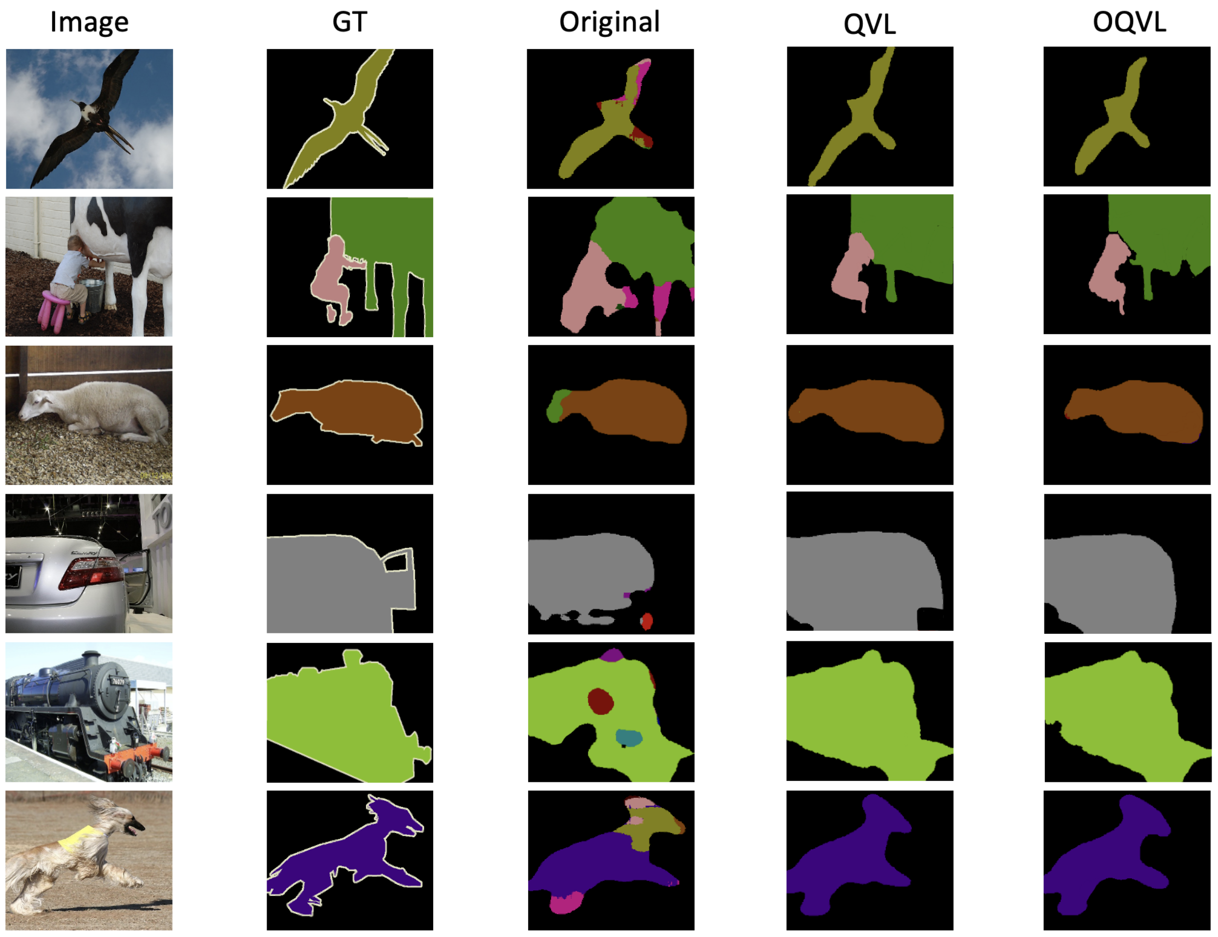


Figure 5.8: Visual results of using Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss with Volumetric Annotations on Weakly Supervised Semantic Segmentation via Sub-category Exploration. For an input image, we show its ground truth, the results without using Volumetric Loss and the results of QVL and OQVL

| | IoU | | |
|---|---|---|---|
| **Class** | **Original** | **QVL** | **OQVL** |
| Background | 0.7521 | 0.8158 | 0.8134 |
| Aeroplane | 0.3632 | 0.3769 | 0.3745 |
| Bicycle | 0.2954 | 0.3091 | 0.3067 |
| Bird | 0.3813 | 0.3950 | 0.3926 |
| Boat | 0.3253 | 0.3390 | 0.3366 |
| Bottle | 0.5421 | 0.5558 | 0.5534 |
| Bus | 0.6773 | 0.6910 | 0.6886 |
| Car | 0.5301 | 0.5438 | 0.5414 |
| Cat | 0.6231 | 0.6368 | 0.6344 |
| Chair | 0.2767 | 0.2904 | 0.2880 |
| Cow | 0.5023 | 0.5160 | 0.5136 |
| Dinningtable | 0.4761 | 0.4898 | 0.4874 |
| Dog | 0.5717 | 0.5854 | 0.5830 |
| Horse | 0.5712 | 0.5849 | 0.5825 |
| Motorbike | 0.5853 | 0.5990 | 0.5966 |
| Person | 0.5212 | 0.5349 | 0.5325 |
| Pottedplant | 0.4132 | 0.4269 | 0.4245 |
| Sheep | 0.5509 | 0.5646 | 0.5622 |
| Sofa | 0.4901 | 0.5038 | 0.5014 |
| Train | 0.5305 | 0.5442 | 0.5418 |
| Tv/monitor | 0.4721 | 0.4858 | 0.4834 |
| **mIoU** | **0.4977** | **0.5138** | **0.5114** |

Table 5.6: IoU for each individual class for Original, Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss with Volumetric Annotations on Weakly Supervised Semantic Segmentation via Sub-category Exploration

### 5.3.4 Semantic Segmentation with Boundary Exploration

The mIoU result of method [22] is 0.6572. Table 5.7 shows the mIoU values of using Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss with Volumetric Annotations on Single Stage Semantic Segmentation from Image Labels. For these two trials the individual class IoU as well as the original class IoU are shown in table 5.8. Figure 5.9 shows the visual results of using Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss with Volumetric Annotations. For an input image, we show its ground truth, the results without using Volumetric Loss and the results of QVL and OQVL.

|  | mIoU |
|---|---|
| Original | 0.6572 |
| Quadratic Volumetric Loss | **0.6760** |
| Outside Quadratic Volumetric Loss | 0.6678 |

Table 5.7: mIoU results of using Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss with Volumetric Annotations on Weakly Supervised Semantic Segmentation with Boundary Exploration

As we see from the results of the first experiment, volumetric supervision with QVL has increased performance. This loss resulted in an increase in mIoU of 1.88%. Looking at the individual IoU of each of the class for the first experiment, we can see that most classes benefit from volumetric supervision. Classes such as Bottle, Bus, Car and Potted plant saw a big increase in IoU, where as classes such as Chair, Cow and Sheep saw slight increases. Some classes such as Dining table and Sofa saw a decrease in performance.

The second experiment using OQVL as the loss shows a slight decrease in mIoU compared to the first one. This loss still results in a 1.06% increase in mIoU over the original method. Individual class IoU do not follow the same pattern as the previous experiment. Classes such as Boat, Cat, Dinning table, Sofa and Train performed better than the first experiment. However, the overall results from the first two experiments match the pattern of the previous methods and show that QVL performs slightly better than OQVL. Visually, from figure 5.9, we can see that our method produces better object shapes, compared to the original method.

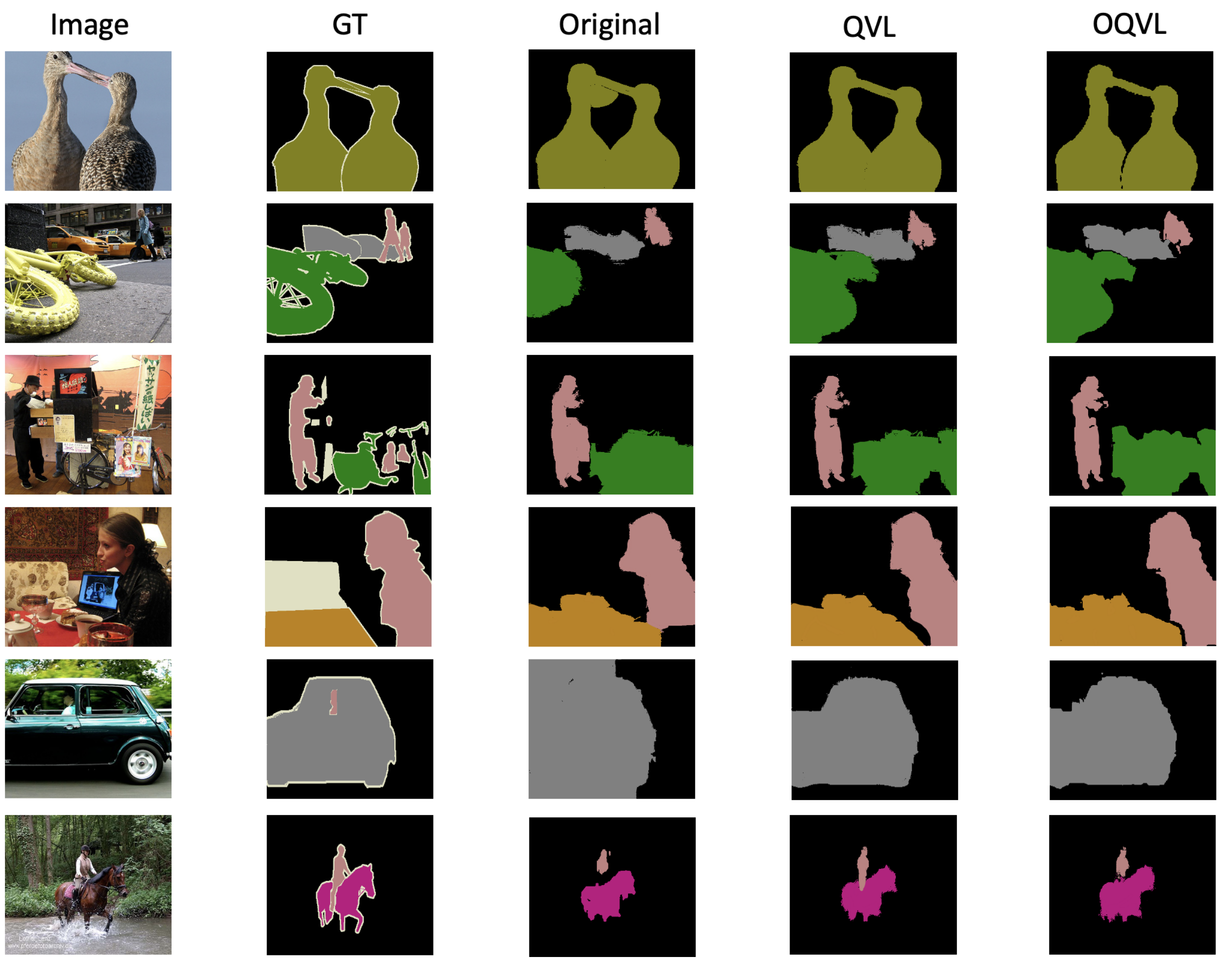


Figure 5.9: Visual results of using Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss with Volumetric Annotations on Weakly Supervised Semantic Segmentation with Boundary Exploration. For an input image, we show its ground truth, the results without using Volumetric Loss and the results of QVL and OQVL

|  | **IoU** |  |  |
|---|---|---|---|
| **Class** | **Original** | **QVL** | **OQVL** |
| Background | 0.8711 | 0.8831 | 0.8814 |
| Aeroplane | 0.6601 | 0.6811 | 0.6738 |
| Bicycle | 0.3285 | 0.3504 | 0.3554 |
| Bird | 0.8047 | 0.8180 | 0.8147 |
| Boat | 0.5113 | 0.5143 | 0.5669 |
| Bottle | 0.5970 | 0.6384 | 0.6058 |
| Bus | 0.7861 | 0.8400 | 0.8120 |
| Car | 0.6737 | 0.7228 | 0.7199 |
| Cat | 0.8210 | 0.8123 | 0.8290 |
| Chair | 0.3669 | 0.3712 | 0.3714 |
| Cow | 0.8226 | 0.8273 | 0.8182 |
| Dinningtable | 0.5142 | 0.4465 | 0.4735 |
| Dog | 0.7773 | 0.8063 | 0.7630 |
| Horse | 0.7927 | 0.8210 | 0.8003 |
| Motorbike | 0.7511 | 0.7841 | 0.7283 |
| Person | 0.6796 | 0.7084 | 0.6957 |
| Pottedplant | 0.4876 | 0.5595 | 0.5196 |
| Sheep | 0.7936 | 0.7947 | 0.7738 |
| Sofa | 0.6187 | 0.6182 | 0.6212 |
| Train | 0.6068 | 0.6263 | 0.6422 |
| Tv/monitor | 0.5374 | 0.5722 | 0.5586 |
| **mIoU** | **0.6572** | **0.6760** | **0.6678** |

Table 5.8: IoU for each individual class for Original, Quadratic Volumetric Loss and Outside Quadratic Volumetric Loss with Volumetric Annotations on Weakly Supervised Semantic Segmentation with Boundary Exploration

### Auxiliary Experiment

In this experiment we evaluate the effect of the negative loss in Eq. 4.4 on the performance of our method. We evaluate the results with and without Negative loss for the modified Joint Learning of Saliency Detection in section 4.1.2. For reference we use the results of Joint Learning of Saliency Detection and Weakly Supervised Semantic Segmentation [133]. The results are shown in table 5.9. Negative loss somewhat improves results, but as seen from the table, most of the progress is achieved by incorporating volumetric loss.

|  | **Original Result** | **Quadratic Volumetric Loss** |  | **Outside Quadratic Volumetric Loss** |  |
|---|---|---|---|---|---|
|  |  | No Negative Loss | Negative Loss | No Negative Loss | Negative Loss |
| **mIoU** | 0.5420 | 0.6010 | **0.6098** | 0.5854 | 0.5979 |

Table 5.9: **Auxiliary Experiment** to show the effectiveness of Negative Loss in Eq. 4.4

## 5.4 Results with noise in user annotation

In this section, we test the sensitivity of our approach to human error in annotating sizes of object classes. We repeat the experiments in section 5.3, but with some misslabeled data, as explained below in section 5.4.1. In total, four tests are performed, two for each loss functions.

### 5.4.1 Noise in Volumetric Annotations

To stimulate human error while annotating, we introduce noise in the Volumetric Annotations. Two variations of annotation noise are introduced. The first is called Neighbouring Bucket Error, while the second one is called Random Bucket Error.

For Neighbouring Bucket Error, we randomly place 5%, 10%, 15% and 20% of the classes in a bucket neighbouring the correct bucket placement. This simulates the situation where the user makes a size annotation error, but this error is actually not far from the correct size bucket.

For Random Bucket Error, we randomly place 5%, 10%, 15% and 20% of the objects in a randomly chosen bucket, excluding the correct bucket. This simulates an annotation error when the user mislabels object size randomly, i.e., the mislabeled size is equally likely to be any bucket, except the correct bucket.

### 5.4.2 Joint Saliency and Semantic Segmentation

For all the four tests, the mIoU are shown in table 5.10. The left side of the table show the results of placing 5%, 10%, 15% and 20% of the objects in a neighbouring bucket. For comparison, we also show the results of placing 0%, i.e., placing all objects in their correct bucket. These are the results of using QVL and OQVL from section 5.3.1. We also shows the results of the original method, without the use of volumetric supervision.

The right side of the table is the same as its left counterpart, with the only difference being that the objects are placed in a random bucket instead of its neighbouring bucket.

| **Neighbouring Bucket** | | | |
|---|---|---|---|
| **Noise Exp.** | | **mIoU** | **Original mIoU** |
| | 0% | 0.6098 | |
| | 5% | 0.5757 | |
| **QVL** | 10% | 0.5428 | 0.5420 |
| | 15% | 0.4507 | |
| | 20% | 0.3169 | |
| | 0% | 0.5979 | |
| | 5% | 0.5650 | |
| **OQVL** | 10% | 0.5222 | 0.5420 |
| | 15% | 0.4051 | |
| | 20% | 0.2550 | |

| **Random Bucket** | | | |
|---|---|---|---|
| **Noise Exp.** | | **mIoU** | **Original mIoU** |
| | 0% | 0.6098 | |
| | 5% | 0.5338 | |
| **QVL** | 10% | 0.4809 | 0.5420 |
| | 15% | 0.3888 | |
| | 20% | 0.2423 | |
| | 0% | 0.5979 | |
| | 5% | 0.5031 | |
| **OQVL** | 10% | 0.4603 | 0.5420 |
| | 15% | 0.3432 | |
| | 20% | 0.1804 | |

Table 5.10: mIoU results of the different noise experiments for Joint Learning of Saliency Detection and Weakly Supervised Semantic Segmentation

For the four noisy experiments, we use the same dataset, with the only difference being where and how many objects are misplaced. For the first experiment where we use QVL and the objects are placed in a neighbouring bucket, we can see that when up to 10% of objects are placed in the incorrect bucket, it still outperforms the original methods score. When we change the loss function to OQVL, we see that the model becomes more sensitive to incorrect object size annotations. The model gets outperformed by the original method when more than 5% of objects are misplaced. This aligns with our previous findings that QVL outperforms OQVL and that OQVL is more sensitive to incorrect volumetric annotations than QVL.

For the third and fourth experiments, we repeat the previous experiments, but this time the objects are misplaced randomly. We can see that regardless of the loss function used, even when 5% of objects are places in random buckets, the model suffers greatly. This shows that the method is highly sensitive when objects size annotations are randomly incorrect, and we should try to ensure that volumetric annotations are done as precisely as possible. Looking at the results of the individual scores for the two experiments, we can see that Random Bucket QVL mIoU scores are better than Random Bucket OQVL, which aligns with our previous results that QVL does better than OQVL.

### 5.4.3 Single Stage Semantic Segmentation

For all the four shift experiments the mIoU are shown in table 5.11. The left side of the table show the results of placing 5%, 10%, 15% and 20% of the objects in a neighbouring bucket. For comparison, we also show the results of placing 0%, i.e., placing all objects in their correct bucket. These are the results of using QVL and OQVL from section 5.3.2. We also show the results of the original method, without the use of volumetric supervision.

The right side of the table is the same as its left counterpart, with the only difference being that the objects are placed in a random bucket instead of its neighbouring bucket.

| **Neighbouring Bucket** | | | | **Random Bucket** | | | |
|---|---|---|---|---|---|---|---|
| **Noise Exp.** | | **mIoU** | **Original mIoU** | **Noise Exp.** | | **mIoU** | **Original mIoU** |
| | 0% | 0.6114 | | | 0% | 0.6114 | |
| | 5% | 0.5942 | | | 5% | 0.5836 | |
| **QVL** | 10% | 0.5444 | 0.5818 | **QVL** | 10% | 0.5238 | 0.5818 |
| | 15% | 0.4523 | | | 15% | 0.3917 | |
| | 20% | 0.3185 | | | 20% | 0.2439 | |
| | 0% | 0.6008 | | | 0% | 0.6008 | |
| | 5% | 0.5882 | | | 5% | 0.5773 | |
| **OQVL** | 10% | 0.5338 | 0.5818 | **OQVL** | 10% | 0.5132 | 0.5818 |
| | 15% | 0.4067 | | | 15% | 0.3461 | |
| | 20% | 0.2579 | | | 20% | 0.1833 | |

Table 5.11: mIoU results of the different noise experiments for Single Stage Semantic Segmentation from Image Labels

For the four noisy experiments, we use the same dataset, with the only difference being where and how many objects are misplaced. For the first and second experiments, where we use QVL and OQVL respectively, and with the objects are placed in a neighbouring bucket. For both we see that the model performs better than the original when up to 5% of object are placed in neighbouring buckets, with QVL performing better than OQVL. When 10% or more objects are misplaced, the results start to deteriorate.

For the third and fourth experiments, we repeat the previous experiments, but this time the objects are misplaced randomly. For the third noise experiment, where QVL is the loss used, the model outperforms the original when upto 5% of object are placed in random buckets. The fourth noise experiment where we use OQVL, the model suffers, even when 5% of objects are misplaced. These results show that the model is sensitive to size annotation mistakes when over 5% of objects are misplaced. This also confirms all the previous findings that QVL outperforms OQVL.

### 5.4.4 Semantic Segmentation via Sub-category

For all the four noisy experiments the mIoU are shown in table 5.12. The left side of the table show the results of placing 5%, 10%, 15% and 20% of the objects in a neighbouring bucket. For comparison, we also show the results of placing 0%, i.e., placing all objects in their correct bucket. These are the results of using QVL and OQVL from section 5.3.3. We also show the results of the original method, without the use of volumetric supervision.

The right side of the table is the same as its left counterpart, with the only difference being that the objects are placed in a random bucket instead of its neighbouring bucket.

| Neighbouring Bucket | | | | Random Bucket | | | |
|---|---|---|---|---|---|---|---|
| **Noise Exp.** | | **mIoU** | **Original mIoU** | **Noise Exp.** | | **mIoU** | **Original mIoU** |
| **QVL** | 0% | 0.5138 | 0.4977 | **QVL** | 0% | 0.5138 | 0.4977 |
| | 5% | 0.4995 | | | 5% | 0.4961 | |
| | 10% | 0.4854 | | | 10% | 0.4640 | |
| | 15% | 0.4444 | | | 15% | 0.3761 | |
| | 20% | 0.3798 | | | 20% | 0.2981 | |
| **OQVL** | 0% | 0.5114 | 0.4977 | **OQVL** | 0% | 0.5114 | 0.4977 |
| | 5% | 0.4985 | | | 5% | 0.4937 | |
| | 10% | 0.4692 | | | 10% | 0.4264 | |
| | 15% | 0.3920 | | | 15% | 0.3547 | |
| | 20% | 0.3274 | | | 20% | 0.2209 | |

Table 5.12: mIoU results of the different noise experiments for Weakly Supervised Semantic Segmentation via Sub-category Exploration

For the first and second noise experiments where we use QVL and OQVL respectively, and with the objects are placed in a neighbouring bucket, the scores align with the previous method. For both experiments we can see that the model outperforms the original when upto 5% of object are placed in neighbouring buckets, with QVL performing better than OQVL. When 10% or more objects are misplaced, the results start to deteriorate.

For the two experiments with random buckets, we repeat the previous experiments, but this time the objects are misplaced randomly. We can see that regardless of the loss function used, even when 5% of objects are places in random buckets, the results suffer. This shows that the method is sensitive when objects size annotations are randomly incorrect, and we should try to ensure that volumetric annotations are done as precisely as possible. Looking at the results of the individual scores for the four experiments, we can see that the experiments using QVL generates mIoU scores that are better than the experiment using OQVL, which aligns with our previous results that QVL performs better than OQVL.

### 5.4.5 Semantic Segmentation with Boundary Exploration

For all the four noisy experiments the mIoU are shown in table 5.13. The left side of the table show the results of placing 5%, 10%, 15% and 20% of the objects in a neighbouring bucket. For comparison, we also show the results of placing 0%, i.e. placing all objects in their correct bucket. These are the results of using QVL and OQVL from section 5.3.4. We also show the results of the original method, without the use of volumetric supervision.

The right side of the table is the same as its left counterpart, with the only difference being that the objects are placed in a random bucket instead of its neighbouring bucket.

| **Neighbouring Bucket** | | | | **Random Bucket** | | | |
|---|---|---|---|---|---|---|---|
| **Noise Exp.** | | **mIoU** | **Original mIoU** | **Noise Exp.** | | **mIoU** | **Original mIoU** |
| **QVL** | 0% | 0.6760 | 0.6572 | **QVL** | 0% | 0.6760 | 0.6572 |
| | 5% | 0.6649 | | | 5% | 0.6567 | |
| | 10% | 0.6539 | | | 10% | 0.6410 | |
| | 15% | 0.6217 | | | 15% | 0.5907 | |
| | 20% | 0.5704 | | | 20% | 0.5127 | |
| **OQVL** | 0% | 0.6678 | 0.6572 | **OQVL** | 0% | 0.6678 | 0.6572 |
| | 5% | 0.6583 | | | 5% | 0.6401 | |
| | 10% | 0.6457 | | | 10% | 0.6228 | |
| | 15% | 0.6135 | | | 15% | 0.5725 | |
| | 20% | 0.5622 | | | 20% | 0.4945 | |

Table 5.13: mIoU results of the different noise experiments for Weakly Supervised Semantic Segmentation with Boundary Exploration

For the first and second noise experiments where we use QVL and OQVL respectively, the objects are placed in a neighbouring bucket, the scores align with the previous method.

For both experiments we can see that the model performs better than the original when upto 5% of object are placed in neighbouring buckets, with QVL performing better than OQVL. When 10% or more objects are misplaced, the results start to deteriorate.

For the third and fourth experiments, we repeat the previous experiments, but this time the objects are misplaced randomly. We can see that regardless of the loss function used, even when 5% of objects are places in random buckets, the model performs worse than the original method without volumetric supervision. This shows that the method is sensitive when objects size annotations are randomly incorrect, and we should try to ensure that volumetric annotations are done as precisely as possible. Looking at the results of the individual scores for the four experiments, we can see the scores aligns with our previous results, showing that QVL does better than OQVL.

## 5.5 Estimating class sizes directly from image level annotations

In this section, we develop a method for estimating class sizes directly from image level annotations, without asking the user to provide volumetric annotations.

### 5.5.1 Estimating Class Sizes

Asking the user to provide volumetric annotations is more time consuming than asking the user to provide image level tags. In this section, we experiment with a method where we do not ask the user to provide size annotations, but try to estimate class sizes automatically.

The idea is as follows. We start with some initial estimate of class sizes, which may be be far from correct. Then we run semantic segmentation model training for a few epochs, and check, for every class and every image if the size went up or down from the original estimate. If the size went up, we increase the size estimate, and if the size went down, we decrease the size estimate. In this way, we can adaptively change the size of classes present in the image. Our intuition is that if the initial class size is underestimated, then after training, it should increase, and if it is overestimated, then after training, it should decrease.

To obtain the initial size estimates, we proceed as follows. Since the background typically is the largest class in an image, we estimate the size of the background as 50% of the image. The remaining 50% of the image we split evenly among all classes present in the image. This is a very rough estimate of class sizes, and may be far from correct. However, our goal is to see whether the tendency for the class is to increase or decrease in size, and change our estimate adaptively.

In total, four tests are performed. The first two tests use **Quadratic Volumetric Loss** and **Outside Quadratic Volumetric Loss** with initial size estimates without adaptation, i.e. the background stays at 50% and the sizes of other classes present in the image are split evenly to add up to the remaining 50%. These tests are called No Re-estimation, and are denoted as $NR_{QVL}$ and $NR_{OQVL}$ in the tables below.

The next pair of tests are the same as the first two, with the difference being the sizes of the objects in the training images are recalculated after every epoch. We start with 50% background and the classes present in images split evenly to add up to the remaining 50%. After the first epoch, the model is used to make predictions on the training images. Then the updated sizes from these images are used in the following epoch. These steps are repeated after every epoch to recalculate the size of objects in the training images. These tests are called with Re-estimation and are denoted as $R_{QVL}$ and $R_{OQVL}$ in the tables below.

### 5.5.2 Joint Saliency and Semantic Segmentation

The mIoU result of method [133] is 0.5420. Table 5.14 shows the mIoU values of the four different experiments performed on Joint Learning of Saliency Detection and Weakly Supervised Semantic Segmentation, compared to the original. For each of these tests, table 5.15 shows the individual class IoUs as well as the original class IoU.

| | mIoU |
|---|---|
| Original | 0.5420 |
| $NR_{QVL}$ | 0.5582 |
| $NR_{OQVL}$ | 0.5445 |
| $R_{QVL}$ | 0.5804 |
| $R_{OQVL}$ | 0.5702 |

Table 5.14: mIoU results of the different experiments for Joint Learning of Saliency Detection and Weakly Supervised Semantic Segmentation

For $NR_{QVL}$ and $NR_{OQVL}$, the model approximates the size of objects in an image without any user effort, by assuming 50% background and 50% foreground. We can immediately see a significant drop in mIoU. The experiments gives us a result of 55.82% and

54.45% which is a mere 1.62% and 0.25% increase respectively from the original method's score. These results also align with the previous findings, showing that QVL outperforms OQVL. The decrease of results also follow the same pattern in the individual class IoU.

The next two experiments $R_{QVL}$ and $R_{OQVL}$ are the same as the previous two experiments, with the only difference being that the size of objects in images are recalculated after every epoch. We initially start by assuming that background and foreground each cover 50% of the image. After an epoch the model outputs its segmentation results. We recalculate the exact sizes of the objects using these predictions and use the updated values for the next round of training. This process is repeated after every epoch. This size reevaluation after every epoch shows an increase in the results compared to the previous experiments. We get a 2.22% and 2.57% increase from the previous experiments respectively, and a 3.84% and 2.82% increase from the original methods mIoU. Similar to previous experiments, $R_{QVL}$, which uses QVL, outperforms $R_{OQVL}$ which uses OQVL.

While the results are worse than that with user provided size annotations, see table 5.14, these results are significantly better than the original method, so estimating object size using our approach is a viable technique for some semantic segmentation methods.

|  | **IoU** |  |  |  |  |
|---|---|---|---|---|---|
| **Class** | **Original** | $NR_{QVL}$ | $NR_{OQVL}$ | $R_{QVL}$ | $R_{OQVL}$ |
| Background | 0.8866 | 0.8898 | 0.8891 | 0.9036 | 0.8934 |
| Aeroplane | 0.6660 | 0.6725 | 0.6685 | 0.7030 | 0.6928 |
| Bicycle | 0.3183 | 0.3226 | 0.3208 | 0.3380 | 0.3278 |
| Bird | 0.7019 | 0.7216 | 0.7044 | 0.7325 | 0.7223 |
| Boat | 0.5554 | 0.5833 | 0.5579 | 0.6008 | 0.5906 |
| Bottle | 0.6165 | 0.6142 | 0.6190 | 0.6296 | 0.6194 |
| Bus | 0.6885 | 0.7076 | 0.6910 | 0.7327 | 0.7225 |
| Car | 0.6123 | 0.6427 | 0.6148 | 0.6606 | 0.6504 |
| Cat | 0.7162 | 0.7390 | 0.7187 | 0.7566 | 0.7464 |
| Chair | 0.1306 | 0.1426 | 0.1331 | 0.1596 | 0.1494 |
| Cow | 0.5493 | 0.5776 | 0.5518 | 0.6180 | 0.6078 |
| Dinningtable | 0.2045 | 0.2127 | 0.2070 | 0.2278 | 0.2176 |
| Dog | 0.6706 | 0.6923 | 0.6731 | 0.7184 | 0.7082 |
| Horse | 0.5476 | 0.5717 | 0.5501 | 0.6014 | 0.5912 |
| Motorbike | 0.5851 | 0.6049 | 0.5876 | 0.6154 | 0.6052 |
| Person | 0.6732 | 0.6823 | 0.6757 | 0.6998 | 0.6896 |
| Pottedplant | 0.3419 | 0.3410 | 0.3444 | 0.3782 | 0.3680 |
| Sheep | 0.5872 | 0.6082 | 0.5897 | 0.6293 | 0.6191 |
| Sofa | 0.2671 | 0.2838 | 0.2696 | 0.3140 | 0.3038 |
| Train | 0.5682 | 0.5914 | 0.5707 | 0.6265 | 0.6163 |
| Tv/monitor | 0.4951 | 0.5219 | 0.4976 | 0.5436 | 0.5334 |
| **mIoU** | **0.5420** | **0.5583** | **0.5445** | **0.5804** | **0.5702** |

Table 5.15: IoU for each individual class for Joint Learning of Saliency Detection and Weakly Supervised Semantic Segmentation

### 5.5.3 Single Stage Semantic Segmentation

The mIoU result of method [6] is 0.5818. Table 5.16 shows the mIoU values of the four different experiments performed on Single Stage Semantic Segmentation from Image Labels, compared to the original. For each of these tests, table 5.17 shows the individual class IoUs as well as the original class IoU.

| | mIoU |
|---|---|
| Original | 0.5818 |
| $NR_{QVL}$ | 0.5820 |
| $NR_{OQVL}$ | 0.5814 |
| $R_{QVL}$ | 0.5979 |
| $R_{OQVL}$ | 0.5856 |

Table 5.16: mIoU results of the different experiments for Single Stage Semantic Segmentation from Image Labels

For $NR_{QVL}$ and $NR_{OQVL}$, as we can see from the results, these two experiments do not improve the result very much. $NR_{QVL}$ only gives us an improvement of 0.02%, while $NR_{OQVL}$ actually performs worse that the original results. However looking at the individual IoU, we can see that the classes such as Bird, Tv/monitor, Train that saw a decrease in scores for $NR_{QVL}$ and $NR_{OQVL}$, perform much better in this setting. These two experiments also confirm that QVL outperforms OQVL, although by a small margin.

For $R_{QVL}$ and $R_{OQVL}$, where the size of objects are recalculated after every epoch, we can see that the results are an improvement over the previous two sets of experiments. $R_{QVL}$ gives an increase of 1.59% in mIoU over $NR_{QVL}$. $R_{OQVL}$ gives an increase of 0.42% over $NR_{OQVL}$. Comparing them to the results of the original method, we see an increase of 1.61% and 0.38% for $R_{QVL}$ and $R_{OQVL}$ respectively. These two experiments follow the same pattern as the previous method, showing that recalculating the size of objects after every epoch improves results and that QVL does better than OQVL. For this method, again, we get better results than the original method with our approach for size estimation, so it is a viable method to employ when user annotated sizes are not available.

| | **IoU** | | | | |
|---|---|---|---|---|---|
| **Class** | **Original** | $NR_{QVL}$ | $NR_{OQVL}$ | $R_{QVL}$ | $R_{OQVL}$ |
| Background | 0.8621 | 0.8622 | 0.8619 | 0.8782 | 0.8659 |
| Aeroplane | 0.6162 | 0.6162 | 0.6160 | 0.6323 | 0.6200 |
| Bicycle | 0.3417 | 0.3412 | 0.3415 | 0.3578 | 0.3455 |
| Bird | 0.6829 | 0.6822 | 0.6827 | 0.6990 | 0.6867 |
| Boat | 0.4562 | 0.4562 | 0.4560 | 0.4723 | 0.4600 |
| Bottle | 0.6113 | 0.6112 | 0.6111 | 0.6274 | 0.6151 |
| Bus | 0.7078 | 0.7072 | 0.7076 | 0.7239 | 0.7116 |
| Car | 0.6363 | 0.6382 | 0.6361 | 0.6524 | 0.6401 |
| Cat | 0.7626 | 0.7622 | 0.7624 | 0.7787 | 0.7664 |
| Chair | 0.2748 | 0.2742 | 0.2746 | 0.2909 | 0.2786 |
| Cow | 0.6112 | 0.6112 | 0.6110 | 0.6273 | 0.6150 |
| Dinningtable | 0.2757 | 0.2772 | 0.2755 | 0.2918 | 0.2795 |
| Dog | 0.7383 | 0.7382 | 0.7381 | 0.7544 | 0.7421 |
| Horse | 0.6272 | 0.6272 | 0.6270 | 0.6433 | 0.6310 |
| Motorbike | 0.6785 | 0.6782 | 0.6783 | 0.6946 | 0.6823 |
| Person | 0.6848 | 0.6842 | 0.6846 | 0.7009 | 0.6886 |
| Pottedplant | 0.4221 | 0.4222 | 0.4219 | 0.4382 | 0.4259 |
| Sheep | 0.6713 | 0.6762 | 0.6711 | 0.6874 | 0.6751 |
| Sofa | 0.3444 | 0.3442 | 0.3442 | 0.3605 | 0.3482 |
| Train | 0.7011 | 0.7012 | 0.7009 | 0.7172 | 0.7049 |
| Tv/monitor | 0.5120 | 0.5102 | 0.5118 | 0.5281 | 0.5158 |
| **mIoU** | **0.5818** | **0.5820** | **0.5814** | **0.5979** | **0.5856** |

Table 5.17: IoU for each individual class for Single Stage Semantic Segmentation from Image Labels

### 5.5.4 Semantic Segmentation via Sub-category

The mIoU result of method [17] is 0.4977. Table 5.18 shows the mIoU values of the four different experiments performed on Weakly Supervised Semantic Segmentation via Sub-category Exploration, compared to the original. For each of these tests, table 5.19 shows the individual class IoUs as well as the original class IoU.

|  | mIoU |
|---|---|
| Original | 0.4977 |
| $NR_{QVL}$ | 0.3429 |
| $NR_{OQVL}$ | 0.3381 |
| $R_{QVL}$ | 0.4651 |
| $R_{OQVL}$ | 0.4448 |

Table 5.18: mIoU results of the different experiments for Weakly Supervised Semantic Segmentation via Sub-category Exploration

$NR_{QVL}$ and $NR_{OQVL}$ assumes 50% background and 50% foreground. As we can see from the results, these experiments decrease mIoU compared to the original method. $NR_{OQVL}$ sees a decrease of 15.96% while $NR_{QVL}$ performs slightly better with a decrease of 15.48%.

$R_{QVL}$ and $R_{OQVL}$, where the sizes of objects are reevaluated, perform significantly better than $NR_{QVL}$ and $NR_{OQVL}$. But these still fall short when compared to the original methods scores. $R_{QVL}$ gives an improvement of 12.22% while $R_{OQVL}$ gives an improvement of 10.67%. These results show that recalculating the size of objects gives us a significant advantage and better results, as well as confirming all the previous methods findings that QVL performs better than OQVL. For this method, we get a decrease in performance, compared to the original method, using our method for estimating sizes.

|  | **IoU** |  |  |  |  |
|---|---|---|---|---|---|
| **Class** | **Original** | $NR_{QVL}$ | $NR_{OQVL}$ | $R_{QVL}$ | $R_{OQVL}$ |
| Background | 0.7521 | 0.7575 | 0.7527 | 0.8344 | 0.8141 |
| Aeroplane | 0.3632 | 0.3650 | 0.3602 | 0.6322 | 0.6119 |
| Bicycle | 0.2954 | 0.2319 | 0.2271 | 0.3819 | 0.3616 |
| Bird | 0.3813 | 0.3406 | 0.3358 | 0.3670 | 0.3467 |
| Boat | 0.3253 | 0.3127 | 0.3079 | 0.4292 | 0.4089 |
| Bottle | 0.5421 | 0.3123 | 0.3075 | 0.5320 | 0.5117 |
| Bus | 0.6773 | 0.4068 | 0.4020 | 0.5193 | 0.4990 |
| Car | 0.5301 | 0.3092 | 0.3044 | 0.3969 | 0.3766 |
| Cat | 0.6231 | 0.2716 | 0.2668 | 0.4080 | 0.3877 |
| Chair | 0.2767 | 0.1739 | 0.1691 | 0.4098 | 0.3895 |
| Cow | 0.5023 | 0.4797 | 0.4749 | 0.3878 | 0.3675 |
| Dinningtable | 0.4761 | 0.1856 | 0.1808 | 0.4488 | 0.4285 |
| Dog | 0.5717 | 0.3082 | 0.3034 | 0.4224 | 0.4021 |
| Horse | 0.5712 | 0.5056 | 0.5008 | 0.3718 | 0.3515 |
| Motorbike | 0.5853 | 0.3720 | 0.3672 | 0.4909 | 0.4706 |
| Person | 0.5212 | 0.1941 | 0.1893 | 0.4374 | 0.4171 |
| Pottedplant | 0.4132 | 0.3010 | 0.2962 | 0.4236 | 0.4033 |
| Sheep | 0.5509 | 0.3631 | 0.3583 | 0.4447 | 0.4244 |
| Sofa | 0.4901 | 0.3145 | 0.3097 | 0.3948 | 0.3745 |
| Train | 0.5305 | 0.4293 | 0.4245 | 0.6113 | 0.5910 |
| Tv/monitor | 0.4721 | 0.2659 | 0.2611 | 0.4233 | 0.4030 |
| **mIoU** | **0.4977** | **0.3429** | **0.3381** | **0.4651** | **0.4448** |

Table 5.19: IoU for each individual class for Weakly Supervised Semantic Segmentation via Sub-category Exploration

### 5.5.5 Semantic Segmentation with Boundary Exploration

The mIoU result of method [22] is 0.6572. Table 5.20 shows the mIoU values of the four different experiments performed on Weakly Supervised Semantic Segmentation with Boundary Exploration, compared to the original. For each of these tests, table 5.21 shows the individual class IoUs as well as the original class IoU.

|  | mIoU |
|---|---|
| Original | 0.6572 |
| $NR_{QVL}$ | 0.6602 |
| $NR_{OQVL}$ | 0.6598 |
| $R_{QVL}$ | 0.6606 |
| $R_{OQVL}$ | 0.6604 |

Table 5.20: mIoU results of the different experiments for Weakly Supervised Semantic Segmentation with Boundary Exploration

$NR_{QVL}$, $NR_{OQVL}$, $R_{QVL}$ and $R_{OQVL}$ all performed better than the original methods scores. All the experiments assume 50% background and 50% foreground. $R_{QVL}$ and $R_{OQVL}$, where the sizes of objects are reevaluated, perform better than $NR_{QVL}$ and $NR_{OQVL}$. For this method, we have a slight improvement in performance, compared to the original method, with our technique for class size estimation.

|  | IoU |  |  |  |  |
|---|---|---|---|---|---|
| **Class** | **Original** | $NR_{QVL}$ | $NR_{OQVL}$ | $R_{QVL}$ | $R_{OQVL}$ |
| Background | 0.8711 | 0.8769 | 0.8774 | 0.8767 | 0.8766 |
| Aeroplane | 0.6601 | 0.6421 | 0.6453 | 0.6470 | 0.6481 |
| Bicycle | 0.3285 | 0.3380 | 0.3408 | 0.3417 | 0.3409 |
| Bird | 0.8047 | 0.7701 | 0.7646 | 0.7637 | 0.7666 |
| Boat | 0.5113 | 0.4894 | 0.4955 | 0.5028 | 0.5113 |
| Bottle | 0.5970 | 0.6135 | 0.6152 | 0.6192 | 0.6210 |
| Bus | 0.7861 | 0.8180 | 0.8184 | 0.8156 | 0.8166 |
| Car | 0.6737 | 0.7047 | 0.7029 | 0.7009 | 0.7047 |
| Cat | 0.8210 | 0.8173 | 0.8165 | 0.8149 | 0.8144 |
| Chair | 0.3669 | 0.3588 | 0.3559 | 0.3549 | 0.3606 |
| Cow | 0.8226 | 0.8227 | 0.8203 | 0.8217 | 0.8221 |
| Dinningtable | 0.5142 | 0.4511 | 0.4356 | 0.4622 | 0.4491 |
| Dog | 0.7773 | 0.7666 | 0.7725 | 0.7709 | 0.7641 |
| Horse | 0.7927 | 0.8090 | 0.8087 | 0.8112 | 0.8095 |
| Motorbike | 0.7511 | 0.7730 | 0.7701 | 0.7690 | 0.7686 |
| Person | 0.6796 | 0.7009 | 0.7043 | 0.6971 | 0.7022 |
| Pottedplant | 0.4876 | 0.5368 | 0.5378 | 0.5327 | 0.5305 |
| Sheep | 0.7936 | 0.8015 | 0.7940 | 0.7919 | 0.7929 |
| Sofa | 0.6187 | 0.6267 | 0.6300 | 0.6301 | 0.6250 |
| Train | 0.6068 | 0.6273 | 0.6292 | 0.6262 | 0.6160 |
| Tv/monitor | 0.5374 | 0.5188 | 0.5199 | 0.5215 | 0.5266 |
| **mIoU** | **0.6572** | **0.6602** | **0.6598** | **0.6606** | **0.6604** |

Table 5.21: IoU for each individual class for Weakly Supervised Semantic Segmentation with Boundary Exploration

## 5.6 Overall Findings

Overall findings from the different experiments on the four methods demonstrate the following:

1. Volumetric Supervision does indeed improve results
2. Of the two loss functions used, Quadratic Volumetric Loss (QVL) performs better than Outside Quadratic Volumetric Loss (OQVL)
3. The model is sensitive to wrong annotations. When placing up to 5% objects in the neighbouring bucket, the model still performs better with volumetric annotations, but placing 5% or more objects in the wrong random bucket results in poorer performance

4. Using Volumetric Annotations with the ten buckets improves performance significantly than assuming 50% background and 50% foreground split amongst objects. Reevaluating the size of objects after every epoch helps improve results. Our method for adaptively estimating sizes of classes without needing user provided size annotation improved the results over the original methods in three out of four cases

# Chapter 6

# Summary and Conclusion

Fully supervised methods need pixel precise annotations for thousands of images. To reduce expensive annotation efforts, weak supervision for semantic segmentation is gaining popularity. In this thesis, we propose a new type of weak supervision for semantic segmentation: **Volumetric Supervision**. In addition to providing the image level labels, the user also provides approximate size of each object class present in the images. This type of annotation is still undemanding on the users time.

In order to incorporate volumetric information into weakly supervised segmentation, in this thesis we introduced several volumetric loss functions that penalize deviation from the object size annotated by the user. In order to test the effectiveness we incorporate our volumetric loss functions into four recently developed methods for weakly supervised segmentation with image level labels.

To evaluate our weak supervision, we outline several different experiments. We also created a simulated dataset that contains size information for the object classes. We also test the sensitivity of our approach to the possible mistakes in the size information dataset.

Our experimental evaluations shows that volumetric supervision gives a significant improvement over image level supervision. However it is sensitive to mistakes in the size information provided by the user.

Our simple method for estimating size of classes without relying on user size annotations worked in three out of four cases. Thus an interesting direction is improving our proposed method. For example, instead of replacing the size with a new estimate computed from the current CNN output, we could replace the size with a weighted combination of the previous size and currently estimated size, providing estimates that change more smoothly.

Another is the use in semi supervised semantic segmentation, where the dataset contains pixel precise labels as well as image level labels. For the images with pixel level labels, we provide exact size information, while using this thesis's methodology for the data with image level labels and train a model with the combination of the two.

# References

[1] Resnet, alexnet, vggnet, inception: Understanding various architectures of convolutional networks, Aug 2017.

[2] HS 13. Neural network: Introduction to neural network: Neural network for deep learning, Mar 2021.

[3] Jiwoon Ahn and Suha Kwak. Learning pixel-level semantic affinity with image-level supervision for weakly supervised semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4981–4990, 2018.

[4] Tanmay Anand, Soumendu Sinha, Murari Mandal, Vinay Chamola, and Fei Richard Yu. Agrisegnet: Deep aerial semantic segmentation framework for iot-assisted precision agriculture. *IEEE Sensors Journal*, 21(16):17581–17590, 2021.

[5] Seliverstova Angelina, L. Padma Suresh, and S. Veni. Image segmentation based on genetic algorithm for region growth and region merging. *2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET)*, pages 970–974, 2012.

[6] Nikita Araslanov and Stefan Roth. Single-stage semantic segmentation from image labels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[7] Bruno Artacho and Andreas Savakis. Waterfall atrous spatial pooling architecture for efficient semantic segmentation. *Sensors*, 19(24):5361, 2019.

[8] Gagana B. Class activation maps, Oct 2019.

[9] Nicolas Bacaër. Verhulst and the logistic equation (1838). 01 2011.

[10] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[11] Pragati Baheti. 12 types of neural networks activation functions: How to choose?, Jan 2022.

[12] Simran Bansari. Introduction to how cnns work, Feb 2019.

[13] Gaudenz Boesch. Vgg very deep convolutional networks (vggnet) - what you need to know, Dec 2021.

[14] 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.

[15] Patrick Brus. Class activation mapping, Jul 2020.

[16] 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.

[17] Yu-Ting Chang, Qiaosong Wang, Wei-Chih Hung, Robinson Piramuthu, Yi-Hsuan Tsai, and Ming-Hsuan Yang. Weakly-supervised semantic segmentation via subcategory exploration. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[18] 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.

[19] 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.

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

[21] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018.

[22] Liyi Chen, Weiwei Wu, Chenchen Fu, Xiao Han, and Yuntao Zhang. Weakly supervised semantic segmentation with boundary exploration. In *European Conference on Computer Vision*, pages 347–362. Springer, 2020.

[23] 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, 2010.

[24] Ankit Choraria. Class activation mapping in deep learning, Nov 2020.

[25] Keh-Shih Chuang, Hong-Long Tzeng, Sharon Chen, Jay Wu, and Tzong-Jer Chen. Fuzzy c-means clustering with spatial information for image segmentation. *Computerized Medical Imaging and Graphics*, 30(1):9–15, 2006.

[26] AI Club. Semantic segmentation using cnn's, Sep 2018.

[27] Jifeng Dai, Kaiming He, and Jian Sun. Boxsup: Exploiting bounding boxes to supervise convolutional networks for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1635–1643, 2015.

[28] Larry S Davis. A survey of edge detection techniques. *Computer graphics and image processing*, 4(3):248–270, 1975.

[29] Larry S Davis, Azriel Rosenfeld, and Joan S Weszka. Region extraction by averaging and thresholding. *IEEE transactions on systems, man, and cybernetics*, (3):383–388, 1975.

[30] 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.

[31] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

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

[33] Priya Dwivedi. Semantic segmentation popular architectures, Mar 2019.

[34] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[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.

[36] Mark Everingham and John Winn. The pascal visual object classes challenge 2012 (voc2012) development kit. *Pattern Analysis, Statistical Modelling and Computational Learning, Tech. Rep*, 8:5, 2011.

[37] 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.

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

[39] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004.

[40] Vincent Feng. An overview of resnet and its variants, Jul 2017.

[41] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[42] Jacob Gildenblat. Class activation maps in keras for visualizing where deep learning networks pay attention, Aug 2016.

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

[44] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning.* MIT press, 2016.

[45] John F Haddon. Generalised threshold selection for edge detection. *Pattern Recognition*, 21(3):195–203, 1988.

[46] Bharath Hariharan, Pablo Arbelaez, Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. Semantic contours from inverse detectors. In *International Conference on Computer Vision (ICCV)*, 2011.

[47] Muneeb Hassan. Vgg16 - convolutional network for classification and detection, Nov 2018.

[48] 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.

[49] 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.

[50] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[51] David Higham. How we use image semantic segmentation, Feb 2018.

[52] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[53] Zilong Huang, Xinggang Wang, Jiasi Wang, Wenyu Liu, and Jingdong Wang. Weakly-supervised semantic segmentation network with deep seeded region growing. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7014–7023, 2018.

[54] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*, 2017.

[55] Suhaas karthikeyan. How do convolutional neural networks work?, Sep 2021.

[56] Emmanuel Kendagor. How to implement image segmentation in ml, May 2021.

[57] Anna Khoreva, Rodrigo Benenson, Jan Hosang, Matthias Hein, and Bernt Schiele. Simple does it: Weakly supervised instance and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 876–885, 2017.

[58] Ralf Kohler. A segmentation system based on thresholding. *Computer Graphics and Image Processing*, 15(4):319–338, 1981.

[59] 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. Springer, 2016.

[60] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[61] Viveka Kulharia, Siddhartha Chandra, Amit Agrawal, Philip Torr, and Ambrish Tyagi. Box2seg: Attention weighted loss and discriminative feature learning for weakly supervised segmentation. In *European Conference on Computer Vision*, pages 290–308. Springer, 2020.

[62] Smriti Kumar and Deepak kumar Singh. Texture feature extraction to colorize gray images. *International Journal of Computer Applications*, 63:10–17, 2013.

[63] Malay Kumar Kundu and Sankar K Pal. Thresholding for edge detection using human psychovisual phenomena. *Pattern Recognition Letters*, 4(6):433–441, 1986.

[64] Vihar Kurama. A guide to alexnet, vgg16, and googlenet, Apr 2020.

[65] James Le. How to do semantic segmentation using deep learning, May 2021.

[66] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[67] Jungbeom Lee, Eunji Kim, Sungmin Lee, Jangho Lee, and Sungroh Yoon. Ficklenet: Weakly and semi-supervised semantic image segmentation using stochastic inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5267–5276, 2019.

[68] Qizhu Li, Anurag Arnab, and Philip HS Torr. Weakly-and semi-supervised panoptic segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 102–118, 2018.

[69] Di Lin, Jifeng Dai, Jiaya Jia, Kaiming He, and Jian Sun. Scribblesup: Scribble-supervised convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3159–3167, 2016.

[70] 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.

[71] D. Marr and Ellen Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London. Series B, Containing papers of a Biological character. Royal Society (Great Britain)*, 207:187–217, 02 1980.

[72] John Martinsson and Olof Mogren. Semantic segmentation of fashion images using feature pyramid networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[73] Anil Chandra Naidu Matcha. A 2021 guide to semantic segmentation, May 2021.

[74] G. Menegaz and R. 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 vol.2, 1996.

[75] D.J. Michael and A.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.

[76] Pierangelo Migliorati, Federico Pedersini, L. Sorcinelli, and Stefano Tubaro. Semantic segmentation applied to image interpolation in the case of camera panning and zooming. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '93, Minneapolis, Minnesota, USA, April 27-30, 1993*, pages 25–28. IEEE Computer Society, 1993.

[77] 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.

[78] Divyanshu Mishra. Demystifying convolutional neural networks using class activation maps., Sep 2019.

[79] Mayank Mishra. Convolutional neural networks, explained, Aug 2020.

[80] 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 (CVPR) Workshops*, June 2020.

[81] Seb NA. An introduction to neural network loss functions, Sep 2021.

[82] 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. *Advances in Neural Information Processing Systems*, 32, 2019.

[83] R. Nock and F. Nielsen. Statistical region merging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1452–1458, 2004.

[84] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1520–1528, 2015.

[85] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[86] Aleksander Obuchowski. Understanding neural networks 1: The concept of neurons, Apr 2020.

[87] Eai Fund Offical. History of image segmentation, Oct 2018.

[88] 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.

[89] Shun-Yi Pan, Cheng-You Lu, Shih-Po Lee, and Wen-Hsiao Peng. Weakly-supervised image semantic segmentation using graph convolutional networks. In *2021 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2021.

[90] 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 *Proceedings of the IEEE international conference on computer vision*, pages 1742–1750, 2015.

[91] Ravindra Parmar. Common loss functions in machine learning, Sep 2018.

[92] 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.

[93] Pedro O Pinheiro and Ronan Collobert. From image-level to pixel-level labeling with convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1713–1721, 2015.

[94] Bharath Raj. A simple guide to the versions of the inception network, Jul 2018.

[95] Sovit Ranjan Rath. Basic introduction to class activation maps in deep learning using pytorch, Jun 2021.

[96] Siddheswar Ray and Rose H. Turi. Determination of number of clusters in k-means clustering and application in colour segmentation. In *The 4th International Conference on Advances in Pattern Recognition and Digital Techniques*, pages 137–143, 1999.

[97] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

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

[99] Adrian Rosebrock. Intersection over union (iou) for object detection, Nov 2017.

[100] Datahacker rs. Cnn vgg 16 and vgg 19, Nov 2018.

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

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

[103] Sumit Saha. A comprehensive guide to convolutional neural networks, Dec 2018.

[104] Shipra Saxena. Alexnet architecture: Introduction to architecture of alexnet, Mar 2021.

[105] Ramprasaath R Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Why did you say that? *arXiv preprint arXiv:1611.07450*, 2016.

[106] N. Senthilkumaran and R. Rajesh. Image segmentation - a survey of soft computing approaches. In *2009 International Conference on Advances in Recent Technologies in Communication and Computing*, pages 844–846, 2009.

[107] Oleksii Sheremet. Intersection over union (iou) calculation for evaluating an image segmentation model, Jul 2020.

[108] Jianbo Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[109] Aparna Shibu. Cnn-a brief intro for beginners, Dec 2021.

[110] Ninad Shukla. Gradient weighted class activation map(grad-cam), Jul 2019.

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

[112] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3136–3145, 2019.

[113] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. *Advances in neural information processing systems*, 28, 2015.

[114] Siti Noraini Sulaiman and Nor Ashidi Mat Isa. Adaptive fuzzy-k-means clustering algorithm for image segmentation. *IEEE Transactions on Consumer Electronics*, 56(4):2661–2668, 2010.

[115] Guolei Sun, Wenguan Wang, Jifeng Dai, and Luc Van Gool. Mining cross-image semantics for weakly supervised semantic segmentation. In *European conference on computer vision*, pages 347–365. Springer, 2020.

[116] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[117] 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.

[118] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[119] Jun Tang. A color image segmentation algorithm based on region growing. In *2010 2nd international conference on computer engineering and technology*, volume 6, pages V6–634. IEEE, 2010.

[120] Meng Tang, Abdelaziz Djelouah, Federico Perazzi, Yuri Boykov, and Christopher Schroers. Normalized cut loss for weakly-supervised cnn segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1818–1827, 2018.

[121] Meng Tang, Federico Perazzi, Abdelaziz Djelouah, Ismail Ben Ayed, Christopher Schroers, and Yuri Boykov. On regularized losses for weakly-supervised cnn segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 507–522, 2018.

[122] Great Learning Team. Introduction to resnet or residual network, Sep 2020.

[123] Sik Ho Tsang. Review: Fcn - fully convolutional network (semantic segmentation), Oct 2018.

[124] Gustavo R Valiati and David Menotti. Detecting pedestrians with yolov3 and semantic segmentation infusion. In *2019 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 95–100. IEEE, 2019.

[125] Paul Vernaza and Manmohan Chandraker. Learning random-walk label propagation for weakly-supervised semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7158–7166, 2017.

[126] Vivian and Amy. Tutorial: Image segmentation, 2015.

[127] George Vogiatzis, Philip HS Torr, and Roberto Cipolla. Multi-view stereo via volumetric graph-cuts. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 391–398. IEEE, 2005.

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

[129] Chi-Feng Wang. The vanishing gradient problem, Jan 2019.

[130] M.A. Wani and B.G. Batchelor. Edge-region-based segmentation of range images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(3):314–319, 1994.

[131] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101–1113, 1993.

[132] Jia Xu, Alexander G Schwing, and Raquel Urtasun. Learning to segment under various forms of weak supervision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3781–3790, 2015.

[133] 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*, 2019.

[134] 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.

[135] 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.

[136] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):1943–1955, 2015.

[137] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.