Occlusion-Ordered Semantic Instance Segmentation

by

Soroosh Baselizadeh

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, 2023

Author's Declaration

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

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

Abstract

Conventional semantic 'instance' segmentation methods offer a segmentation mask for each object instance in an image along with its semantic class label. These methods excel in distinguishing instances, whether they belong to the same class or different classes, providing valuable information about the scene. However, these methods lack the ability to provide depth-related information, thus unable to capture the 3D geometry of the scene.

One option to derive 3D information about a scene is monocular depth estimation. It predicts the absolute distance from the camera to each pixel in an image. However, monocular depth estimation has limitations. It lacks semantic information about object classes. Furthermore, it is not precise enough to reliably detect instances or establish depth order for known instances.

Even a coarse 3D geometry, such as the relative depth or occlusion order of objects is useful to obtain rich 3D-informed scene analysis. Based on this, we address occlusion-ordered semantic instance segmentation (OOSIS), which augments standard semantic instance segmentation by incorporating a coarse 3D geometry of the scene. By leveraging occlusion as a strong depth cue, OOSIS estimates a partial relative depth ordering of instances based on their occlusion relations. OOSIS produces two outputs: instance masks and their classes, as well as the occlusion ordering of those predicted instances.

Existing works pre-date deep learning and rely on simple visual cues such as the y-coordinate of objects for occlusion ordering. This thesis introduces two deep learning-based approaches for OOSIS. The first approach, following a top-down strategy, determines pairwise occlusion order between instances obtained by a standard instance segmentation method. However, this approach lacks global occlusion ordering consistency, having undesired cyclic orderings. Our second approach is bottom-up. It simultaneously derives instances and their occlusion order by grouping pixels into instances and assigning occlusion order labels. This approach ensures a globally consistent occlusion ordering. As part of this approach, we develop a novel deep model that predicts the boundaries where occlusion occurs plus the orientation of occlusion at the boundary, indicating which side of it occludes the other. The output of this model is utilized to obtain instances and their corresponding ordering by our proposed discrete optimization formulation.

To assess the performance of OOSIS methods, we introduce a novel evaluation metric capable of simultaneously evaluating instance segmentation and occlusion ordering. In addition, we utilize standard metrics for evaluating the quality of instance masks. We also evaluate occlusion ordering consistency, and oriented occlusion boundaries. We conduct evaluations on KINS and COCOA datasets.

Acknowledgements

I would like to express my heartfelt gratitude to my incredible supervisors, Prof. Yuri Boykov and Olga Veksler, for their unwavering support and their wonderful kindness throughout my master's studies. Olga and Yuri have not only imparted their exceptional expertise but also created an environment that fostered my personal and academic growth. Their willingness to engage in great discussions and challenge my thinking has been invaluable. I have learned so much from these conversations, and I am grateful for the intellectual stimulation and the enthusiasm they brought to our research endeavors.

I would like to thank my thesis committee members, Prof. Hongyang Zhang and Prof. Krzysztof Czarnecki. Their valuable time and thoughtful deliberation in reading and evaluating my thesis are deeply appreciated.

I would also like to thank Tom Cauduro and Lori Paniak, members of CSCF for their great help in facilitating the use of my computational resources. Thank you for your unwavering availability and willingness to help, even during holidays.

I would also like to extend my heartfelt thanks to my dear friends Mohammad (MZi), Niousha, Alireza (Vez), Shadi, Hadi, Soheil, Mohammad (Khalaji), Ehsan, Navid, Mehrbod, and Mohammad Ali. Your unwavering support and friendship have made this journey joyful and memorable. Through the highs and lows, your presence has made the tough times easier and the successes more meaningful. The laughter, camaraderie, and shared experiences have made this chapter of my life truly special.

Lastly, I want to express my sincere gratitude to my beloved family - my father, mother, and brother, Sina. Your constant love, support, and encouragement have been the foundation of my strength and resilience. Your belief in me and unwavering support have given me the courage to pursue my dreams. I am incredibly fortunate to have such a loving and supportive family by my side, and I am deeply grateful for all that you have done for me.

Dedication

To my mother, father, and brother, for the endless love and support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Given an input image, the task of semantic instance segmentation is to generate masks for individual objects in an image, along with their corresponding class labels. Unlike semantic segmentation, which classifies each pixel into pre-defined categories, instance segmentation also aims to separate objects of the same class and distinguish them as separate instances. This technique provides more detailed and precise information about the objects present in an image. Figure 1.1 provides an example of semantic instance segmentation and compares it with semantic segmentation.

Figure 1.1: Comparison of semantic segmentation and semantic instance segmentation. On left, dark blue shows 'background' class, purple shows 'car' class and light blue shows 'bicycle' class. On right, light blue shows the background. Other colors are random for distinguishing different object instances. Note that in instance segmentation there is an object class predicted for each mask as well, such as 'car', 'pedestrian', or 'bicycle', which is not shown in the figure for simplicity.

Instance segmentation has a wide range of applications in various fields. It plays a crucial role in autonomous driving [24, 83], where it assists in understanding the environment

Figure 1.2: Monocular depth estimation example from [85]. Warmer colors show smaller depth.

by identifying and tracking different objects such as cars, and pedestrians. Further, instance segmentation finds utility in robotics [35, 119], video surveillance [106, 79], remote sensing imagery [15], aerial imagery [117], and biomedical imaging [20] where accurate object localization and distinction are essential for performing specific tasks.

Classic instance segmentation methods, such as [40, 14, 112] adopt a two-stage pipeline involving bounding box generation followed by pixel-wise segmentation within the boxes. While achieving high accuracy, these methods suffer from inefficiency. Recently, contour-based methods, including [128, 64, 28] have gained attention due to their efficiency and accuracy. Instead of working with instance masks, they deal with instance contours. They represent contours using a limited number of vertices (e.g., 128) and try to regress these vertices using deep networks. By treating instance segmentation as a contour regression task, they reduce computational complexity compared to mask-based methods. These methods have demonstrated high accuracy while being computationally efficient, showing promise for real-time instance segmentation [128].

Since an image is a 2D projection of a 3D scene, one may want to ask questions related to 3D geometry. However, answering depth-related questions from just semantic instance segmentation is not possible, at least without further processing/learning. For example, a sofa can be to the left of a person in the image but it could be either behind or in front of the person in the corresponding 3D scene (from the camera's viewpoint). Augmenting semantic instance segmentation with depth-related information and relations among the objects can provide a significantly richer understanding of the scene.

One way to extract 3D information about a scene is monocular depth estimation.

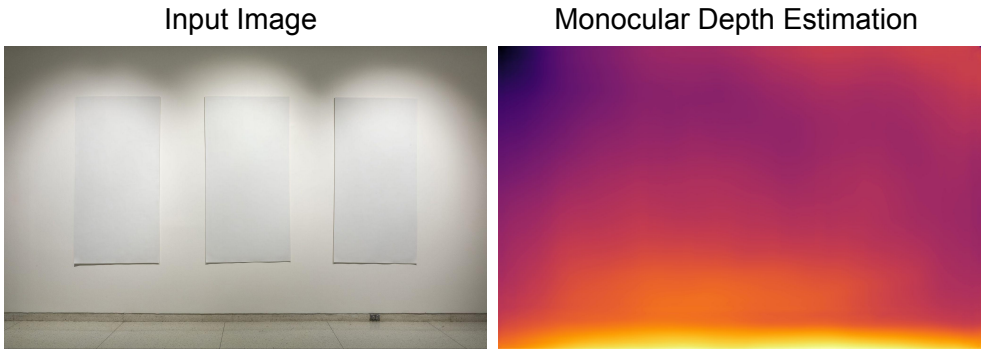


Figure 1.3: Monocular depth estimation has limitations. The pieces of cardboard on the wall are assigned almost the same depth as the wall behind them disabling the recovery of their relative depth ordering. The depth map is obtained using MiDaS [85] estimator. Warmer colors show smaller depth.

Monocular depth estimation receives a *single* input image and predicts the absolute distance from the camera for each pixel. Figure 1.2 shows an example of the input and output of a monocular depth estimator. Recent monocular depth estimation methods [25, 85] use deep learning architectures of encoder-decoder type, where the encoder extracts features and the decoder generates a depth map. Some methods [2, 1] incorporate skip connections or attention mechanisms for better performance. These models are trained on paired RGB images and ground-truth depth maps. During training, the models minimize the difference between predicted and ground-truth depth maps using loss functions. These methods have shown promising results in various applications of monocular depth estimation, such as navigation, autonomous driving, and virtual reality [134, 102, 107, 121].

Monocular depth estimates 3D geometry, but it lacks semantic information. Also, as we show in this thesis, it is not accurate enough neither to detect instances nor to establish depth order for known instances detected by other approaches. For example, consider a thin cardboard sheet leaning on a wall as in Figure 1.3. Since the resolution is limited, the cardboard and the wall are likely to get equal depth estimates. Yet the relative depth order of the cardboard is in front of the wall. In general, predicting the exact absolute depth of each pixel is a hard task. On the other hand, determining the *relative depth* order of objects is significantly easier, even for humans. Meanwhile, it can provide very useful 3D information about the scene.

Actually, tackling the hard task of absolute depth estimation is unnecessary for answering many interesting questions about the 3D geometry of scenes, e.g. if a pedestrian

is behind a car or in front of it in a driving scene. In fact, knowing the relative depth of objects gives a coarse 3D geometry of the scene that is useful in many applications. For example, understanding driving scenes [130], scene collaging [46], where one splits the image into several depth layers to understand complex scenes, and scene/video editing [122] where one can remove certain layers or add new layers from other scenes, can all benefit from such a coarse 3D geometry, i.e. relative depth ordering of objects. Interestingly, humans are highly skilled at ordering objects by depth in images and even line drawings [23]. Humans are known to rely on *occlusion* cues for deriving such orderings [75, 51, 43].

Occlusion refers to the phenomenon where one object obscures parts of another object in an image or a scene. Occlusion can occur when objects overlap in the image plane or when one object is positioned in front of another in a 3D scene. Understanding occlusion relations is important for amodal segmentation [83, 133], optical flow estimation [45, 116], and scene understanding [46, 104]. Occlusions are a strong depth ordering cue [43]. Given a pair of objects, humans easily determine which one occludes the other, and there are datasets [83, 133] with such pairwise annotations. Given a set of pairwise occlusion relations, we can recover the relative depth order between any pair of instances connected by a monotonic chain, i.e. each object in the chain occludes (or is occluded by) the next object. Hence, occlusion ordering, defined as establishing the correct order in which objects occlude each other from the camera's viewpoint, can provide a coarse 3D geometry of the scene. For example, in Fig. 1.5, right, thirteen objects on the right are in a known occlusion order. As discussed, such a coarse 3D geometry can answer many interesting question about the scene for different applications.

Motivated by the above, we address *Occlusion-Ordered Semantic Instance Segmentation* (OOSIS). The goal of OOSIS is to produce instance masks, their corresponding semantic classes, and their partial relative depth order based on occlusions. We can visualize occlusion order using either a graph, where nodes are instances and directed edges are inserted for known adjacent occlusions, or a relative depth map, where an instance is assigned a larger intensity than any other instance it occludes. Figure 1.4 shows an example of the two possible visualizations. Besides, Fig. 1.5 is an illustration of our input and output, which consists of instances with their corresponding classes, and their occlusion order, which is visualized as a relative depth map in Fig. 1.5.

OOSIS is useful for applications such as image captioning [52], question answering [67], and retrieval [50], where one is interested in a more detailed 3D description of the scene. Such descriptions are more insightful than a soup of orderless objects [130]. OOSIS is also useful for scene de-occlusion [127], which tries to find occlusion ordering and complete the invisible parts of the occluded objects. Scene/video editing [122] can also benefit from OOSIS as it helps to extract objects and organize them in ordered layers. In general,

OOSIS can also help scene understanding in other applications such as in autonomous driving [83], and medical imaging [21].

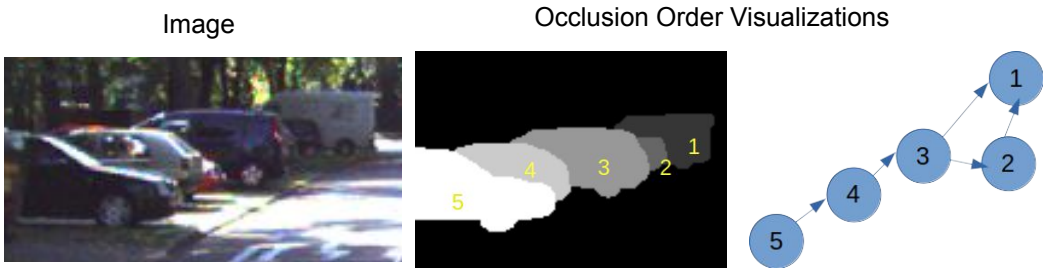


Figure 1.4: Occlusion order can be visualized either as a depth map or as a graph. In the depth map visualization, the background is black, and each instance is brighter than all of its occludees. In the graph visualization, each instance is a node and there is an edge from occluders to each of their immediate occludees.

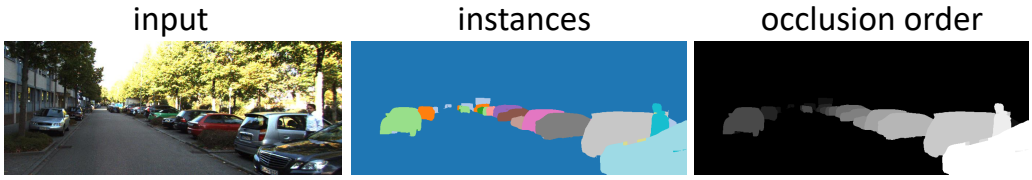


Figure 1.5: Illustration of occlusion-ordered semantic instance segmentation (OOSIS). Given an image, we output the instances with their corresponding classes, and their occlusion order, visualized as a relative depth map. The semantic classes of extracted instances is not shown here for simplicity.

Contributions

Our work is the first to address OOSIS in the context of deep learning. There are some works prior to deep learning [122, 46, 104], however, they are based on simple occlusion heuristics, such as size, or y-coordinate, see Chapter 3 for more details.

We initially show the failure of two naive deep learning-based methods, underscoring the importance of our later carefully designed approaches. In the first naive method, a deep model is directly trained to simultaneously predict both the relative depth and instance

labels. This model exhibits severe limitations in accurately extracting instance masks for objects within the scene and encounters difficulties in handling occlusion scenarios.

The second naive method employs two separate deep learning models. A deep model estimates the monocular depth map of the input image, and another deep model performs semantic segmentation. Using the obtained semantic segmentation, we identify the non-background pixels and apply clustering on their depth values, which were derived from the estimated depth map. This method also yields poor results due to the inherent inaccuracy of monocular depth estimation for precise detection of instances as discussed.

To accomplish the challenging task of OOSIS, we propose two effective approaches. Our first approach is top-down. It first applies standard semantic 'instance' segmentation [41, 128] and then estimates pairwise occlusion order between neighboring instances. The advantage of the top-down approach is that we can use state-of-the-art instance segmentation to get accurate masks. The disadvantage is that because occlusions are estimated in a pairwise fashion, they are not globally consistent. For example, we may have estimated that A occludes B, B occludes C, and C occludes A, leading to A occludes A. Occlusion cycles in ground truth are exceedingly rare, the overwhelming majority of cycles are from errors in pairwise occlusion estimates.

Our second approach is bottom-up, which means we group pixels into instances with proper occlusion ordering. It performs instance segmentation and occlusion ordering simultaneously by formulating it in CRF framework [10]. For this formulation, we need oriented occlusion boundaries, and we design a novel CNN model for this task, outperforming prior occlusion boundary methods [84]. The advantage of our bottom-up approach is that it produces globally consistent occlusion ordering. In addition, our bottom-up approach is also a novel method for standard instance segmentation, but it requires occlusion annotated ground truth.

To comprehensively assess the effectiveness of methods for OOSIS, we additionally devise a novel evaluation metric. This metric evaluates the quality of instance mask extraction and the performance of global occlusion ordering simultaneously, using both predicted and ground truth instance masks and occlusion graphs as inputs and producing Accuracy vs. Recall curves for each method.

We evaluate our approaches on KINS [83] and COCOA [133] datasets, using both standard metrics for instance masks, and our new metric for global occlusion order. In general, standard bottom-up approaches perform worse than the standard top-down ones for instances' mask quality. However, the accuracy of *our* bottom-up approach is similar or better, depending on the metric, than *our* top-down approach. Our bottom-up approach is also better at occlusion order because it recovers a globally consistent cycle-free order.

Thesis Organization

In the following, we first explain the necessary background for neural networks, deep learning, and discrete optimization in Chapter 2. Then, in Chapter 3, we discuss the related work in detail. Chapter 4 explains the naive methods in more detail and qualitatively demonstrates their poor performance. We elaborate on our two proposed effective approaches in Chapter 5, for our top-down approach, and Chapter 6, for our bottom-up approach. Chapter 7 presents various experiments and results for a comprehensive comparison of our approaches and baselines on the datasets and tasks discussed.

Chapter 2

Background

2.1 Neural Networks

In recent years, there has been a tremendous surge in the popularity of Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL). These technologies have gained widespread recognition across various domains, such as computer vision, natural language processing, and speech recognition. If one wants to clarify the relation between these topics, it can simply be described by DL $\subset$ ML $\subset$ AI. ML's main objective, and hence DL's, is to enable machines to *learn* from observed data. This process involves feeding a dataset, which is a set of observed historical data, into the system. The system is supposed to utilize analytical and statistical techniques to learn from the data so that after the learning phase, also called *training* phase, it can generate accurate predictions for new unseen inputs. ML and DL techniques aim to enhance the machine's ability to understand and interpret complex structures within data, i.e. its ability to learn.

In this work, we focus on a subset of ML, called *Supervised Learning* in which the algorithms rely on labeled data. It means that each sample in the training dataset has a corresponding label, which is the desired prediction for that input. For instance, if we want to classify different input images by the category of the object present in them, then a training sample would be an image plus a class label, such as 'cat', 'car', etc. By utilizing labeled data, we aim to train models that can accurately predict or classify future examples based on the knowledge gained from the provided labels. Supervised learning algorithms are versatile and can be applied to two main types of problems: classification and regression. *Classification* involves predicting "discrete" labels or categories, while *regression* deals with predicting "continuous" values, such as price.

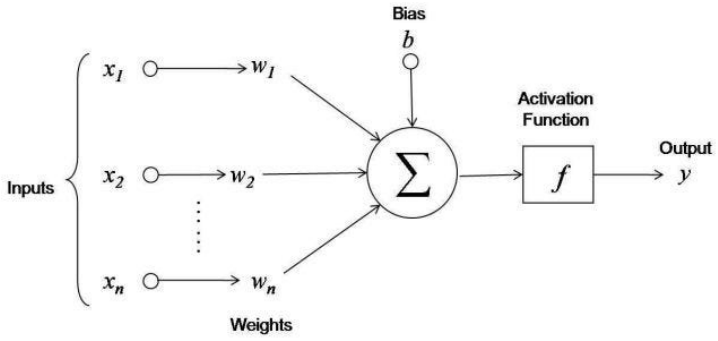


Figure 2.1: The internal structure of a neuron in a neural network. Image is from [6].

The following delves into the basics of machine learning, neural networks, and deep learning, establishing an introduction to the underlying concepts and principles.

2.1.1 Basics

Structure

A neural network is a computational model inspired by the structure and functioning of the human brain's neural networks [5]. A neural network receives an input and processes it to generate a final output for that input. An input can be different across different domains. For instance, if we work with images, our input would be the intensity values of all pixels of the image. Neural networks consist of interconnected nodes, called neurons that are organized in layers. Each neuron receives input signals, processes them, and produces an output signal. The basic components of a neural network include:

- **Neurons**: Neurons are the fundamental processing units within a neural network. They receive input signals, perform computations on them, and produce output signals. Figure 2.1 describes the internal structure of a neuron. For each neuron, the process of producing the output from the received input is as follows. Figure 2.1 depicts the procedure. The inputs, denoted as $x_1, x_2, \ldots, x_n$, are multiplied by their corresponding weights, represented as $w_1, w_2, \ldots, w_n$, respectively. The weighted inputs and the bias term b are then summed together. This summation is subsequently passed through an *activation function*, denoted as f, which generates the output of the neuron, y. Mathematically, this can be represented as $y = f(\sum_i (w_i x_i) + b)$.

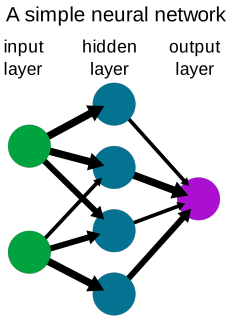


Figure 2.2: A visualization of a simple neural network with only one hidden layer. Image is from [76].

It can also be represented using vector operations. Consider $\mathbf{w}$ as a column vector where its i-th element is w_i. Likewise, suppose $\mathbf{x}$ is a column vector made from x_is. Then, $y = f(\mathbf{w}^T\mathbf{x} + b)$ [5]. Each neuron has its own weights, bias, and activation function.

- **Layers**: Neurons are organized into layers, typically categorized as the input layer, hidden layers, and output layer. The input layer receives the initial input data, while the output layer produces the final output or prediction. The hidden layers, located between the input and output layers, perform intermediate computations and extract features from the input data. The important thing about layers is that they are stacked so that the input of one layer is the output of the previous layer or a combination of the outputs of several previous layers. Hence, the layers are the same in the essence of their working structure and they process the original input further and further. Figure 2.2 depicts a simple neural network with one hidden layer. A neural network can have arbitrarily large numbers of hidden layers. Figure 2.3 shows a neural network with three hidden layers. When all neurons in a layer are connected to *all* neurons of the previous layer, the layer is called a Fully Connected Layer. A network consisting of fully connected layers is called a Fully Connected Network (FCN).

Now, the question is how neural networks learn. The answer lies in the training procedure explained below.

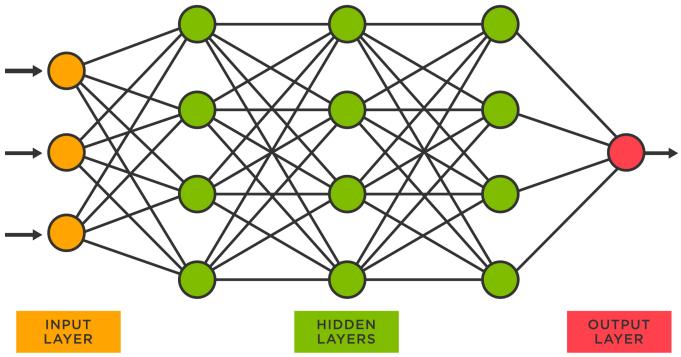


Figure 2.3: Visualization of a neural network with three hidden layers. The output of one layer is the next layer's input. Image is from [118].

Learning and Training

The learning process of neural networks involves updating the *parameters* of the network to improve its ability to make accurate predictions. The parameters include the described weights and biases for all neurons of all layers. It is obvious that the output of the network solely depends on the weights, biases, and activation functions described. If the activation functions themselves have any parameter, they will also be considered as a parameter that can be learned and updated during the training phase. Updating all weights and parameters of the network is achieved through a mechanism called *backpropagation* [91].

During training, the neural network takes the input data and produces output predictions. This step is called *forward propagation* as the information flows from the input layer to the output. The produced predictions are compared to the desired or true labels using a *loss* function, which quantifies the error or mismatch between the predicted and actual values. The goal is to minimize this error by adjusting the weights of the network. Backpropagation [91] is a technique used to determine how each weight in the network contributes to the overall error. The backpropagation algorithm distributes the error across the layers of the network based on the contribution of each weight. It applies the chain rule of calculus to compute the gradients efficiently by propagating the error gradients backward layer by layer. Once the gradients are computed, an optimization algorithm, such as *gradient descent* [16] or one of its variants, is used to update the weights. The update is performed by taking a step in the direction opposite to the gradient. A *learning rate* determines the size of the step, which controls the speed of convergence.

The weight updates gradually refine the network's performance over multiple iterations

or *epochs*. By repeatedly presenting the training data to the network (forward propagation), adjusting the weights through backpropagation and optimization, and fine-tuning the predictions, the network learns to generalize patterns and improve its accuracy on unseen data.

In the following, we elaborate on activation functions, loss functions, and optimizers such as gradient descent.

Activation Functions

An activation function in a neural network determines the activation level of a neuron, indicating whether it should be "fired" or activated based on its input. It serves as a threshold or decision-making mechanism for the neuron, determining the relevance or importance of its input in the prediction process. By applying simple mathematical operations, the activation function transforms the input signal into an output signal that represents the neuron's activation state. This activation or non-activation decision is crucial in the overall functioning and learning capabilities of the neural network, as it allows the network to model complex relationships and make predictions based on the importance of specific inputs. Also, activation functions play a crucial role in neural networks by introducing non-linearity to the network's computations [30, 29]. They determine the output of a neuron based on its input. Activation functions enable neural networks to make non-linear transformations on the input data.

The most common activation functions used in neural networks are as follows [29].

- **Sigmoid**: The sigmoid activation function, also known as the logistic function, maps the input to a value between 0 and 1. It has the mathematical form

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{2.1}$$

  The sigmoid function is widely used in binary classification problems because it squashes the input to a probability-like output, where values closer to 0 indicate the presence of one class, and values closer to 1 indicate the presence of another class. However, the sigmoid function suffers from vanishing gradients, limiting its effectiveness in hidden layers of neural networks. Vanishing gradients refer to a phenomenon that can occur during the training of deep neural networks, where the gradients calculated during backpropagation become extremely small as they propagate from the output layer to the earlier layers. This results in the weights of the

earlier layers being updated very slowly, or not being updated at all, impeding the learning process.

- **Softmax**: The softmax activation function is commonly used in multi-class classification problems. It takes a vector of inputs and produces a probability distribution over multiple classes, ensuring that the sum of the output probabilities is equal to 1. The softmax function is defined as

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, \tag{2.2}$$

  where x_i is the input to the i-th neuron and n is the total number of classes. Softmax is often used in the output layer of the network to produce normalized class probabilities for classification tasks.

- **Tanh (Hyperbolic Tangent)** : The hyperbolic tangent function, denoted as $tanh(x)$, is similar to the sigmoid function but maps the input to a range between -1 and 1. It is defined as

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{2.3}$$

  Tanh is useful when working with data that ranges from negative to positive values and is symmetric around zero. Like the sigmoid function, tanh suffers from vanishing gradients for extreme input values.

- **ReLU (Rectified Linear Unit [34])**: The ReLU activation function is defined as

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

  where x is the input to the neuron. ReLU sets negative inputs to zero and keeps positive inputs unchanged. This simple yet effective function introduces sparsity, accelerates training, and mitigates the vanishing gradient problem. ReLU is widely used in deep neural networks and has contributed to the success of many state-of-the-art architectures.

Loss Functions

In machine learning and optimization tasks, a loss function, also known as a cost function or objective function, is a mathematical function that quantifies the discrepancy or "loss" between the predicted outputs of a model and the actual or desired outputs. It serves as a measure of how well the model is performing on a given task.

The purpose of a loss function is to provide a numerical representation of the model's performance, allowing it to be optimized and improved through the process of training. By evaluating the loss, the model can adjust its internal parameters, such as weights and biases, to minimize the discrepancy between predictions and ground truth values.

The choice of a suitable loss function depends on the specific problem and the nature of the data. Different tasks, such as classification, regression, or sequence generation, often require different loss functions. Here are a few commonly used loss functions [48]:

- **Mean Squared Error (MSE)**: MSE is widely used for regression tasks. It calculates the average squared difference between the predicted and actual values. It penalizes larger errors more heavily due to the squared term. The formula for MSE is

$$MSE = \frac{1}{n}\sum_{i}(y_i - \hat{y}_i)^2, \tag{2.5}$$

  where y_i represents the actual target value, $\hat{y}_i$ represents the predicted value, and n is the total number of samples.

- **Binary Cross-Entropy** [90]: Binary cross-entropy is typically used for binary classification problems. It measures the dissimilarity between the predicted probabilities and the true binary labels, quantifying the information loss. Its formula is

$$BCE = -\frac{1}{n}\sum_{i}(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)), \tag{2.6}$$

  where y_i represents the true binary label (0 or 1), $\hat{y}_i$ represents the predicted probability, and n is the total number of samples.

- **Categorical Cross-Entropy** [90]: Categorical cross-entropy is suitable for multi-class classification problems. It calculates the average cross-entropy loss across all classes, comparing the predicted class probabilities with the true class labels. It can be formulated as

$$CE = -\frac{1}{n}\sum_{i}\sum_{j} y_{i,j}\log(\hat{y}_{i,j}), \tag{2.7}$$

  where $y_{i,j}$ represents the true class label (0 or 1) for sample i and class j, $\hat{y}_{i,j}$ represents the predicted probability for sample i and class j, and n is the total number of samples.

- **Weighted Binary Cross-Entropy** [97]: The weighted binary cross-entropy extends this by introducing weights to the loss calculation. The weights assigned to the positive class (w_{pos}) and the negative class (w_{neg}) allow for emphasizing the importance of correctly predicting the positive class or addressing class imbalance issues. It is formulated as

$$wBCE = -\frac{1}{n}\sum_{i}([w_{pos}y_i \log(\hat{y}_i) + w_{neg}(1-y_i)\log(1-\hat{y}_i)]), \tag{2.8}$$

  where y_i represents the true binary label (0 or 1), $\hat{y}_i$ represents the predicted probability, n is the total number of samples, w_{pos} is the weight assigned to the positive class (class 1), and w_{neg} is the weight assigned to the negative class (class 0). By multiplying each term in the loss calculation by the corresponding weight, the weighted binary cross-entropy loss function gives more significance to the positive class during training, allowing the model to focus on correctly predicting the positive class based on the assigned weight.

Optimizers

Optimizers are algorithms used in machine learning to update the parameters of a model during the training process. Their primary purpose is to minimize the loss function and optimize the model's performance. Optimizers determine how the model's parameters, such as weights and biases, are adjusted based on the gradients of the loss function with respect to those parameters.

The gradient of the loss function w.r.t the parameters represents the direction and magnitude of the steepest increase in the loss function. Optimizers utilize this information to iteratively update the model's parameters, gradually moving them in a direction that reduces the loss and improves the model's predictions. Figure 2.4 shows a simplified workflow of optimization based on gradients.

There are various types of optimizers available, each with its own update strategy. Some of the commonly used optimizers include:

- **Gradient Descent** [16]: The basic form of optimization, where the model's parameters are updated by taking steps proportional to the negative gradient of the loss function.
- **Stochastic Gradient Descent (SGD)** [87, 3]: A variation of gradient descent that randomly selects a subset of training samples (mini-batch) for each update step. This helps accelerate the training process and improve convergence.

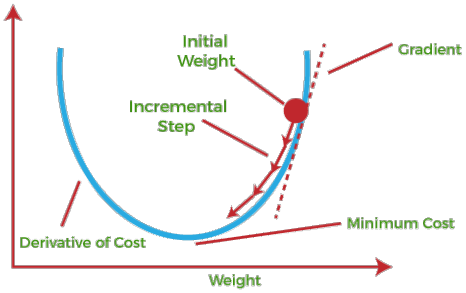


Figure 2.4: Optimizing based on gradient descent. Starting from the initial weights, the mode refines the weights using the gradients of the loss function w.r.t the weights to reach the optimal weight values. The learning rate determines the size of the steps. Image is from [39].

- **Adam (Adaptive Moment Estimation)** [55]: An adaptive learning rate optimization algorithm that computes individual adaptive learning rates for different model parameters. It combines the advantages of AdaGrad [31] and RMSprop [103] methods to provide efficient and effective optimization.

The choice of optimizer depends on the specific problem, dataset, and model architecture. Different optimizers have different behaviors and may converge at different rates. It is often recommended to experiment with different optimizers and learning rates to find the optimal combination for a given task.

The learning rate is a hyperparameter that determines the step size at which the model's parameters are updated during training. It controls the speed and magnitude of the adjustments made to the model's parameters in response to the computed gradients of the loss function.

The learning rate is a crucial parameter as it influences the convergence and stability of the training process [47]. If the learning rate is set too high, the updates to the parameters may be too large, causing the training process to oscillate or even diverge. On the other hand, if the learning rate is set too low, the updates may be too small, resulting in slow convergence and prolonged training time [13].

2.1.2 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) [61, 59] are a class of deep learning models specifically designed for analyzing visual data, such as images. CNNs have revolutionized the field

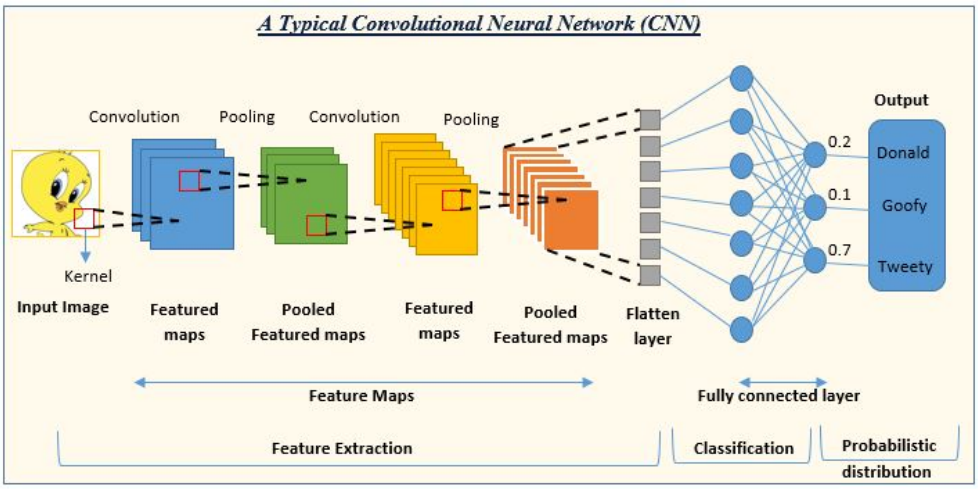


Figure 2.5: Visualization of a convolutional neural network (CNN). Image is from [93].

of computer vision by demonstrating exceptional performance in various tasks, including image classification [61, 59, 96, 42], object detection [86], and image segmentation [132, 88]. Figure 2.5 visualizes an example of a CNN. CNNs are basically different from Fully Connected Networks (FCNs), like the ones we have described so far, by using two layers, namely convolution and pooling layers. The following explains each of them.

Convolution Layers

Convolutional layers are fundamental building blocks of Convolutional Neural Networks (CNNs) and play a crucial role in capturing and extracting local features from input data, especially in the context of images. A convolutional layer applies a set of learnable filters (also known as kernels) [54] to the input data using the convolution operation.

The convolution operation involves sliding the filters across the input data and computing the element-wise multiplication between the filter and the corresponding local receptive field of the input [54]. Figure 2.6 shows an example of such an operation. This process results in a feature map, which represents the filtered response of each filter to different local regions of the input [54]. The filters in a convolutional layer are responsible for detecting specific patterns or features, such as edges, textures, or more complex structures, at different spatial locations [54].

Each filter in a convolutional layer is typically small in spatial size but extends across the full depth (number of channels) of the input data. This depth-wise connectivity allows

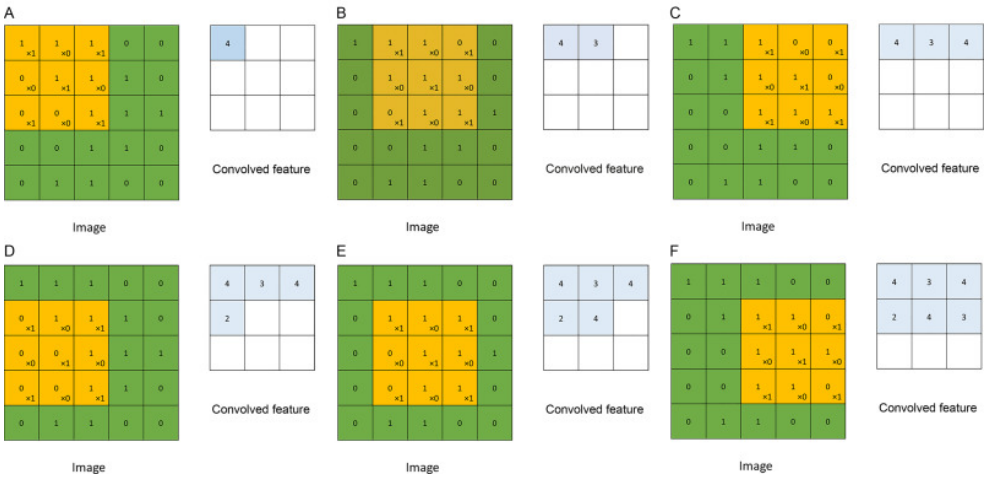


Figure 2.6: An example of a single convolution filter operating on an input image. In each step, the filter shifts by stride= 1 to the right or down to produce the result. The kernel size is 3×3 and there is no padding in this example. The values for the filter are written at the bottom right corner of each cell. Image is from [33].

the filters to capture both spatial and channel-wise information simultaneously. The filters are shared across the entire input, meaning the same set of weights is used at every spatial location, enabling the network to detect similar patterns regardless of their position in the input [54].

To further enhance the capability of capturing spatial information, convolutional layers often incorporate additional operations. These operations include:

- **Padding**: Padding involves adding extra border pixels to the input data to preserve the spatial dimensions of the input and avoid reducing the size of the feature map. Padding helps to maintain spatial resolution and prevent information loss at the borders of the input. Padding is usually employed by adding zeros [54].
- **Stride**: Stride determines the step size at which the filters are applied to the input during the convolution operation. A stride of 1 means the filters move one pixel at a time, preserving the spatial resolution. Larger stride values result in a smaller output size but can help reduce computational complexity [54].

Convolutional layers are typically stacked together to form deeper networks, allowing the model to learn hierarchical representations of increasing complexity [54]. The output

of one convolutional layer serves as the input to the next layer, enabling the network to capture high-level features by progressively combining low-level features.

CNNs differ from fully connected networks (FCNs) in their architectural design and their ability to exploit spatial hierarchies in data. Here are some key differences between CNNs and FCNs:

- **Local Connectivity**: CNNs take advantage of the local connectivity of neurons in the visual cortex. Each neuron in a convolutional layer is connected to only a small local region in the input image, allowing them to focus on local patterns and features. In contrast, FCNs connect each neuron to every neuron in the previous layer, lacking the notion of spatial locality [54].
- **Shared Weights**: CNNs utilize shared weights across different spatial locations in the input. The same set of weights (filters) is applied to different parts of the input, enabling the network to learn and detect patterns regardless of their position in the image. FCNs, on the other hand, have independent weights for each connection, resulting in a large number of parameters and less weight sharing [54].
- **Hierarchical Feature Extraction**: CNNs employ a hierarchical structure to capture increasingly complex features. Lower layers in a CNN learn simple features like edges and textures, while higher layers learn more abstract features like shapes and objects. This hierarchical feature extraction allows CNNs to represent and recognize complex visual patterns effectively. FCNs lack this explicit hierarchical structure [54].
- **Translation Invariance**: CNNs inherently possess translation invariance, meaning they can recognize patterns regardless of their position in the image. This property makes CNNs robust to small variations and enables them to generalize well to new, unseen images. FCNs, without the convolutional and pooling operations, lack this translation invariance [54].

Atrous Convolution

Atrous convolution, also known as dilated convolution [44, 92], is an operation that allows for the extraction of multi-scale features from images or feature maps. Different from standard convolutions, atrous convolution introduces controlled gaps or holes in the convolutional kernel. These gaps enable the convolutional operation to capture information from a wider area while maintaining the original resolution of the feature map and having the same number of parameters as a standard convolution kernel of the same size.

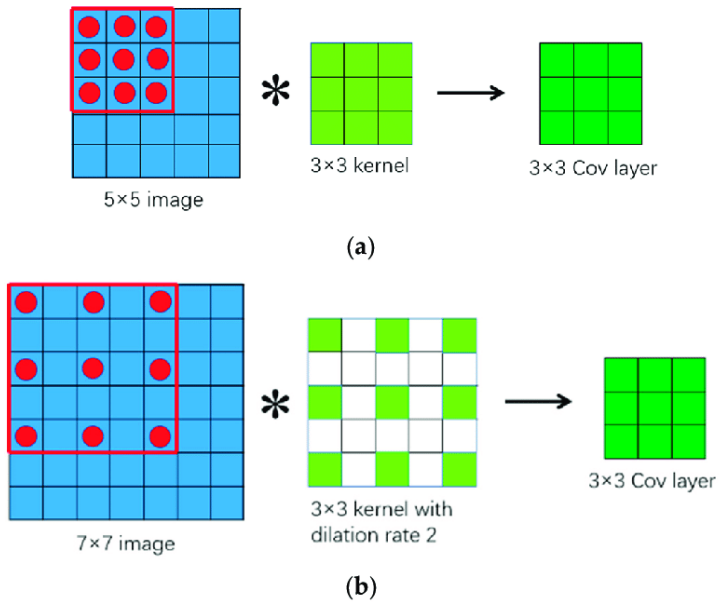


Figure 2.7: **a)** Standard $3x3$ convolution kernel. **b)** Atrous convolution kernel with the same size. The small green squares of the kernel are each multiplied by the corresponding red dots. As seen, the atrous kernel captures information from a larger area (the red square) by introducing holes between the kernel elements. Image is from [94].

Figure 2.7 shows an example of an atrous convolution kernel performing on an input image. As seen, though the size of the kernels is 3×3 for both the standard convolution and the atrous convolution, the latter is taking the information from a larger area into consideration by using holes.

By adjusting the dilation rate, which determines the spacing between the kernel elements, atrous convolution can effectively capture contextual information at different scales. This makes it particularly valuable in tasks such as image segmentation, where capturing both fine-grained details and global context is crucial for accurate and comprehensive analysis [132, 19].

Pooling Layers

Pooling is an essential operation in Convolutional Neural Networks (CNNs) that helps to reduce the spatial dimensionality of feature maps, extract dominant features, and introduce spatial invariance [38, 54]. Pooling is typically applied after convolutional layers

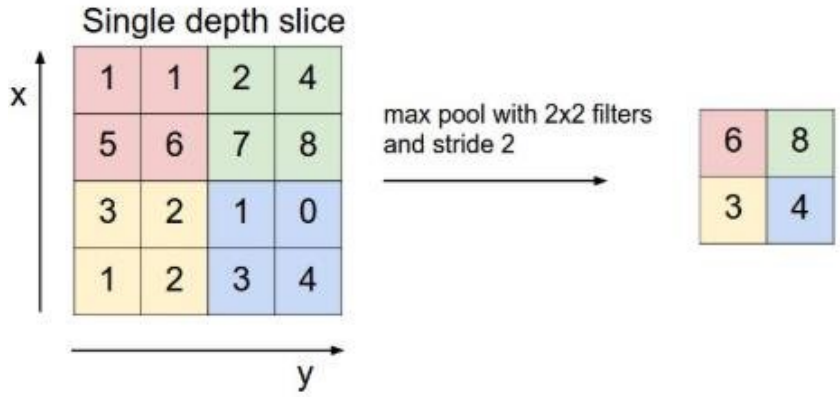


Figure 2.8: An example of a max pooling operation with a filter size of 2 and a stride of 2. Image is from [60].

to progressively downsample the feature maps while retaining the most relevant information [38, 54].

There are different types of pooling operations used in CNNs, with the most common one being max pooling. Max pooling divides the input feature map into non-overlapping rectangular regions (pools) and outputs the maximum value within each pool. Figure 2.8 depicts an example of such an operation. By selecting the maximum value, max pooling retains the most salient feature in each local region. Max pooling helps in extracting the most significant features while discarding irrelevant variations and reducing the spatial dimensionality of the feature maps.

The main advantages of pooling are [27]:

- **Dimensionality reduction**: Pooling reduces the spatial dimensionality of the feature maps, making subsequent layers computationally more efficient and reducing the risk of overfitting [27, 54].
- **Translation invariance**: Pooling introduces spatial invariance by selecting the most dominant feature within each local region. This allows the network to recognize features regardless of their precise location, enhancing the network's ability to generalize [27, 54].
- **Extraction of dominant features**: Pooling helps to extract the most important and distinctive features by selecting the maximum or average values. This can en-

hance the network's ability to capture the most relevant information for the given task [27, 54].

It is important to note that pooling is a form of lossy compression since it discards some information by downsampling. However, by retaining the most salient features, pooling contributes to the overall efficiency and effectiveness of CNNs in various computer vision tasks such as image classification, object detection, and segmentation.

2.2 Labeling Problems and Energy Optimization

Most explanations below are based on [110, 10]. A variety of computer vision tasks, such as semantic segmentation, disparity map estimation, and instance segmentation can be thought of as *labeling* problems. In a labeling problem, we have a set of *sites*, $\mathcal{P}$, and a set of *labels*, $\mathcal{L}$ [110]. Sites represent the image parts for which we want to predict a label. In this work, we consider dense labeling problems and we want to label all pixels in an image. Hence, our sites are all the pixels of our image. We refer to the sites as *pixels* for simplicity. The labels can be different based on the problem. For example, in semantic segmentation, the labels are different possible semantic classes, like 'car', 'pedestrian', etc. The labeling problem is to assign a label from $\mathcal{L}$ to each pixel $p \in \mathcal{P}$. We name the label assigned to pixel $p \in \mathcal{P}$ as x_p. A labeling, $\mathbf{x}$, is the vector of x_p for all different pixels. The optimization approach defines a systematic two-step framework to solve different labeling problems [110]. The steps can be described as follows:

- First, one should define an *objective* function. This function maps all possible solutions for the problem to a real number. Its goal is to measure the quality or *goodness* of a solution [110]. Designing such an objective function can be challenging as one must formulate various constraints a solution needs to satisfy in a way that a better solution receives a better score from the function. As a result, to formulate such a function, we need to first, extract the required constraints for the problem at hand. In many vision tasks, there are two main sets of constraints defined. *Data* constraints that reflect how much the solution matches with the observed data. *Prior* constraints reflect how much the solution agrees with the prior knowledge we have about a good solution for the problem [110]. Mostly, the objective functions are designed in a way that smaller values correspond to better solutions. Such objective functions are widely known as *energy functions* [110].

- Second step involves finding a solution that has the best goodness. More formally, one should minimize the defined energy function over the possible solutions. Optimizing the energy function is even harder than designing one [110]. Most of the energy functions are not convex, meaning they can have multiple local minima. Also, obviously, the set of all possible solutions is exponentially large for most vision tasks, such as semantic segmentation or instance segmentation. The computational demands are quite intense [110]. Hence, most methods aim for finding an approximate answer [110].

Below, we explain some energy functions and optimization algorithms.

2.2.1 Energy Functions

A wide variety of energy functions have the form of

$$E(\mathbf{x}) = \sum_{p \in \mathcal{P}} D_p(x_p) + \sum_{p,q \in \mathcal{N}} V_{p,q}(x_p, x_q), \tag{2.9}$$

where $\mathcal{N}$ is the set of interacting pairs of pixels. It can consist of the adjacent pixels, e.g. in a 4 or 8-neighborhood system, or it can be any other set [10]. In this formulation, the D terms are called the *unary* terms and the V terms are called the *pairwise* terms.

An example of pairwise terms is the famous Potts model [82]. It captures the behavior of neighboring pixels by promoting label similarity. It favors homogeneous regions and promotes spatial coherence in the labeling. In other words, it promotes *smoothness* in the labeling, penalizing frequent alternation among different labels [10, 110]. It is formulated as:

$$E(x) = \sum_{p,q \in \mathcal{N}} w_{p,q}[x_p \neq x_q], \tag{2.10}$$

where the [.] is the Iverson bracket, and $\mathcal{N}$ is the set of interacting pixels. $w_{p,q}$ is a pair-specific weight. In general, it can be a fixed constant for all pairs or a variable for each [10]. In this work, we have the same constant weights for all pairs. This energy penalizes a labeling $\mathbf{x}$ by $w_{p,q}$ for each adjacent pixel pair (p, q) if their labels are different. Hence, it favors smooth solutions. Obviously, if an energy function only has a single term forcing the *prior* knowledge, such as the smoothness prior as in the Potts model above, it will not find a good solution. For instance, it can output the same label for all pixels minimizing the Potts model although being undesired and not distinguishing different pixels. This underscores the need for other terms and in fact, the *data* terms in the energy function.

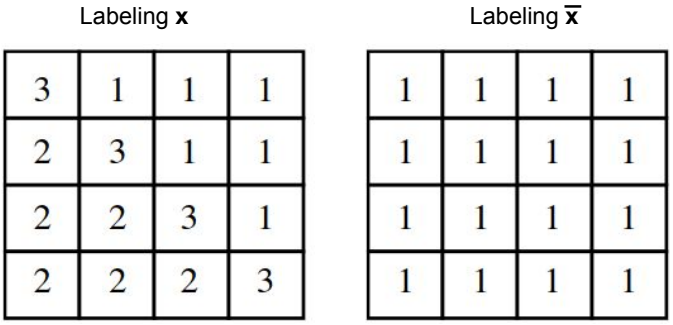


Figure 2.9: α-expansion move example, where $alph = 1$. Image is from [110]

Overall, most energy functions are a combination of different *unary* and *pairwise* terms each trying to capture the goodness of an input solution based on the observed *data*, or *prior* knowledge of the problem. For interested readers, please see [110, 10] for more examples and discussions.

2.2.2 Discrete Optimization Algorithms

As explained, after designing an energy function, one needs to minimize it over the possible solutions. Since computing the globally optimal solution is often NP-hard [10], there are a variety of methods opting for a local minimum [110]. When the labels are discrete, there are different ways to define a local minimum [110]. A natural way is by using the concept of *move* spaces [110, 80]. In fact, some of the most effective and efficient discrete optimization algorithms work based on defining *move* spaces and using graph cut algorithms [10, 110]. Below, we briefly describe *move*, *move space*, and two of the algorithms that we use in this thesis.

A *move* is a pair of labelings $(\mathbf{x}, \bar{\mathbf{x}}) \in \mathcal{X} \times \mathcal{X}$, where $\mathcal{X}$ is the set of all possible labelings. A move space, $\mathcal{M}$, is a set of some moves, $\mathcal{M} \subset \mathcal{X} \times \mathcal{X}$ [10, 110]. If a a move such as $(\mathbf{x}, \bar{\mathbf{x}})$ is in $\mathcal{M}$ then it is allowed. It means you can go from $\mathbf{x}$ to $\bar{\mathbf{x}}$. If a move is not in $\mathcal{M}$ then such transition is forbidden. Naturally, a local minimum with respect to $\mathcal{M}$ is defined as a labeling such as $\mathbf{x}^*$ where no further move $m \in \mathcal{M}$ from $\mathbf{x}^*$ exists that can reduce the energy.

Below, we describe two optimization algorithms, along with their corresponding move spaces, that are later used in this thesis.

- **Expansion Moves** [10] Let $\alpha \in \mathcal{L}$, then $(\mathbf{x}, \bar{\mathbf{x}})$ is an α-expansion move if there

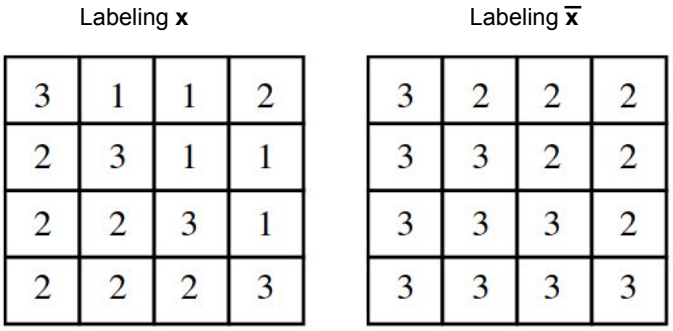


Figure 2.10: 1-jump move example. Image is from [110]

Algorithm 1 Optimizing α-expansion [10]

$E :=$ energy function, $\mathcal{L} :=$ set of all possible labels, *labelling* := labels for all pixels
$\mathbf{x} \leftarrow$ arbitrary labelling
$success \leftarrow 0$
while $success = 1$ **do**
    $success \leftarrow 0$
    for $\alpha \in \mathcal{L}$ **do**
        Find $\bar{\mathbf{x}} := argmin E(\bar{\mathbf{x}})$ among all $\bar{\mathbf{x}}$ within one α-expansion of $\mathbf{x}$
        if $E(\bar{\mathbf{x}}) < E(\mathbf{x})$ **then**
            $\mathbf{x} \leftarrow \bar{\mathbf{x}}$
            $success \leftarrow 1$
        end if
    end for
end while
return x

Algorithm 2 Optimizing 1-jump [110]

$E :=$ energy function, $\mathcal{L} :=$ set of all possible labels, *labelling* := labels for all pixels
$\mathbf{x} \leftarrow$ initialize with all pixels having the minimum label
$success \leftarrow 0$
while $success = 1$ **do**
  $success \leftarrow 0$
  Find $\bar{\mathbf{x}} := argmin E(\bar{\mathbf{x}})$ among all $\bar{\mathbf{x}}$ within one 1-jump of $\mathbf{x}$
  if $E(\bar{\mathbf{x}}) < E(\mathbf{x})$ **then**
    $\mathbf{x} \leftarrow \bar{\mathbf{x}}$
    $success \leftarrow 1$
  end if
end while
return x

exists $A \subset \mathcal{P}$ such that

$$\bar{x}_p = \alpha \qquad \text{for } p \in A \tag{2.11}$$
$$x_p = \bar{x}_p \qquad \text{for } p \notin A. \tag{2.12}$$

An example of an α-expansion move is shown in Figure 2.9. A labeling $\mathbf{x}$ is a local minimum with respect to expansion moves if no further α-expansion can be made for any α to decrease the energy function at that labeling. The work in [10] proposes an algorithm that finds a local minimum with respect to such moves. This algorithm works based on graph cuts. The overall algorithm is shown in Algorithm 1. In each iteration, it goes over all $\alpha \in \mathcal{L}$ and finds the optimal α-expansion move out of all possible ones to decrease the energy function. Then it performs that move and goes to the next label. It stops when no further moves are possible for reducing the energy. To find the optimal move within one α-expansion from the current labeling, the algorithm uses graph cuts. For further explanation on the graph-cut part see [10, 58], as explaining it is beyond the scope of this thesis.

- **Jump Moves [110]**: Consider when all labels are represented by integer values. A move $(\mathbf{x}, \bar{\mathbf{x}})$ is called an i-jump move if for any $p \in \mathcal{P}$, $\bar{x}_p - x_p = i \quad (i \geq 1)$ or $\bar{x}_p - x_p = 0$. An example of a 1-jump move is shown in Figure 2.10. Note that, in general, the jump moves do not require the labels to be integers or i to be positive. In this thesis, we only work with integer labels and 1-jumps, and hence, do not explain beyond. The optimization algorithm shown in Figure 2, finds a local minimum with respect to the 1-jump moves. In each iteration, it finds the optimal jump move and

performs it until no further move is possible. Again, finding the optimal move is based on graph cuts. For an explanation of how these graph-cut algorithms work, please see [110, 58].

Note that for any of these move-based algorithms to work using graph cuts, the energy functions need to be submodular [10, 58] with respect to that specific move. We will discuss submodularity for our energy functions later in Chapter 6.

Chapter 3

Related Work

3.1 Semantic Segmentation

Semantic segmentation is an image segmentation technique that assigns meaningful labels to each pixel based on a predefined set of classes. In this process, every pixel in an image is categorized into specific classes, such as "bicycle", or "background", providing an understanding of the scene. Figure 3.1 shows an example of semantic segmentation for an input image. There are three classes in the image, 'background', 'pedestrian', and 'car'. Semantic segmentation allows for identifying the classes of individual pixels.

Semantic segmentation is a versatile technique with a wide array of applications. It is employed in autonomous driving [12]. In medical imaging and diagnosis, it aids in precise delineation of structures and pathological regions [22, 69, 70]. Additionally, it finds use in facial segmentation, handwriting detection and analysis, agriculture, fashion, video surveillance, image editing, and more [17, 61, 72, 4, 68, 108, 71, 74].

Figure 3.1: An example of semantic segmentation for an input image. Each pixel is assigned a class label from the 'background', 'car', and 'pedestrian' classes.

Semantic segmentation, a challenging problem for machine vision despite its seemingly effortless execution by the human visual system, has been a subject of extensive research for decades. In recent years, deep learning and neural networks, particularly CNNs, have dominated the field of semantic segmentation [19, 132, 89]. These models gained popularity due to their ability to automatically extract features from large image datasets without the need for manual feature engineering.

To train a fully supervised semantic segmentation network, a dataset with pixel-level annotations is essential. Each pixel in the dataset is labeled with a class from a predefined set of classes. During training, the CNN learns to identify features and representations associated with each class label. Once trained, the CNN can process new images, accurately assigning class labels to each pixel. This training process, utilizing images and their corresponding pixel-level labels, is known as fully supervised semantic segmentation. By leveraging the power of CNNs and providing detailed annotations, significant progress has been made in achieving accurate and fine-grained semantic segmentation results. Our bottom-up approach in Chapter 6 relies on semantic segmentation for its first stage. We use PSPNet [132] as the base of our semantic segmentation model. Below, we explain PSPNet in detail.

3.1.1 PSPNet [132]

Pyramid Scene Parsing Network (PSPNet) is a deep learning architecture specifically designed for pixel-wise semantic segmentation tasks in computer vision.

The key idea behind PSPNet is to capture contextual information from different scales to improve segmentation accuracy. It leverages the concept of dilated convolutions [124], also known as atrous convolutions, to enlarge the receptive field of convolutional layers without increasing the number of parameters. This enables the network to capture both local and global contextual information.

PSPNet consists of two main components: the Pyramid Pooling Module and the Encoder-Decoder Structure. The Pyramid Pooling Module is responsible for capturing multi-scale contextual information. It takes the output feature maps from the last convolutional layer and performs pooling operations at different scales, using different bin sizes. This allows the network to capture contextual information at multiple scales and aggregate it into a fixed-size representation.

The Encoder-Decoder Structure in PSPNet combines the extracted contextual information with high-resolution feature maps to produce detailed segmentations. The encoder

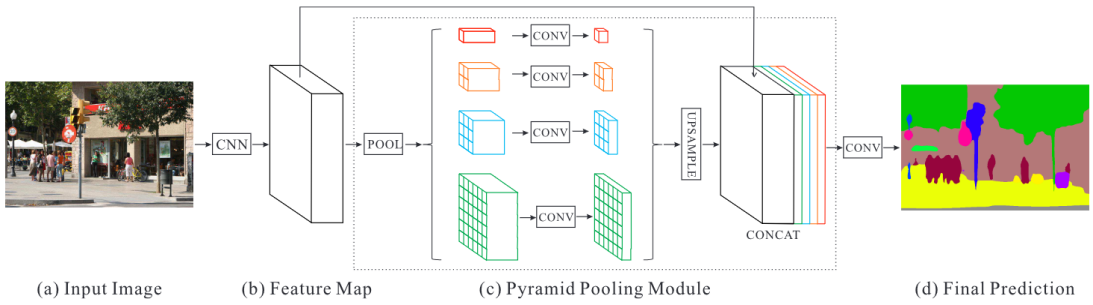


Figure 3.2: Overview of PSPNet. Given an input image (a), a CNN is applied and the outputs of the last convolutional layer are called as features (b), then a pyramid parsing module is applied to harvest different sub-region representations, followed by upsampling and concatenation layers to form the final feature representation, which carries both local and global context information in (c). Finally, the representation is fed into a convolution layer to get the final per-pixel prediction (d). Image and explanation are from [132].

part consists of convolutional and pooling layers that gradually reduce the spatial resolution while increasing the number of channels. The decoder part, on the other hand, uses up-sampling and skip connections to recover the spatial resolution and refine the segmentation output.

One notable aspect of PSPNet is its ability to handle images of arbitrary sizes during both training and inference. This is achieved by adopting a "sliding window" technique, where large images are divided into overlapping patches that are processed independently by the network. The final segmentation map is obtained by merging the predictions from all patches.

PSPNet has demonstrated impressive performance on various challenging scene parsing benchmarks, outperforming previous state-of-the-art methods. Its ability to capture multi-scale contextual information and effectively leverage dilated convolutions makes it particularly well-suited for detailed scene understanding tasks, such as object segmentation, scene labeling, and image parsing.

3.2 Semantic Instance Segmentation

In semantic "instance" segmentation, the goal is to generate a pixel-level mask for each object instance in an image, in addition to predicting its semantic class. For instance,

Figure 3.3: An example of semantic instance segmentation. For each instance, a mask is produced. Also, the semantic class of each instance is predicted, which is not shown in this figure for simplicity.

in Figure 3.3, all 'bicycle' and 'pedestrian' objects are identified by a separate mask and a predicted class. Instance segmentation enables models to understand the spatial extent and location of each object instance, distinguishing between different instances even if they belong to the same category. Instance segmentation is known as a keystone of many real-world computer vision applications, such as autonomous driving [24, 83], video surveillance [106, 79], and robotics [35, 119].

While semantic instance segmentation provides valuable scene information, it lacks depth-related details. In this thesis, we propose two approaches that not only provide instance segmentation-level information but also output a coarse 3D geometry of the scene. Our first approach, the top-down one, leverages conventional instance segmentation methods. Hence, we review the instance segmentation works and explain the two instance segmentation models employed in our experiments in more detail.

Generally, one can divide instance segmentation methods into two main groups, based on how they predict the final masks of object instances. Mask-based methods, such as [40, 14, 112, 8], predict a mask for each object, and during training, they optimize to produce better masks. On the other hand, contour-based methods focus on predicting and refining object contours. Contours represent the boundaries of object instances and can be represented using a smaller number of points compared to pixel-level masks. Consequently, contour-based methods often offer faster computation and more efficient processing.

3.2.1 Mask-based Instance Segmentation

Two-stage Methods: Two-stage instance segmentation methods involve a two-step process to identify and segment object instances within an image. In the first stage, these methods typically perform object detection to identify potential object regions within the

image. Object detection algorithms, such as Faster R-CNN [86] that use Region Proposal Networks (RPN) [86], are commonly used in this step. The object detection stage aims to generate a set of bounding box proposals that likely contain different object instances. In the second stage, the method refines the initial bounding box proposals and predicts the pixel-level masks for each object instance. This stage is responsible for accurately segmenting each object and assigning the appropriate class label. Examples of such methods include [40, 14, 112]. These methods are often more accurate but computationally intensive compared to single-stage methods [128].

One-stage Methods: Single-stage instance segmentation methods directly predict object masks in a single step, without the need for a separate object detection stage. These methods are designed to be simpler, faster, and more computationally efficient compared to their two-stage counterparts [128]. An example of such a method is YOLACT [8]. It combines object detection and mask prediction into a unified framework. YOLACT utilizes a set of predefined anchor boxes across multiple feature maps to detect objects at different scales and aspect ratios. The network simultaneously predicts object bounding boxes, mask coefficients, and class probabilities. Instead of directly generating binary object masks, YOLACT uses a linear combination of learned coefficients to produce high-quality instance segmentation masks. Other single-stage methods include [114, 115]. These methods greatly improve the speed. However, they sacrifice performance for that. As we are not focused on speed in this thesis, we utilize two-stage methods due to their superior performance.

Pixel-grouping Methods (Bottom-up): These methods do not rely on explicit object detection, region proposals, or anchor boxes. In contrast, they first find embeddings or features for each pixel of the image, then, group those pixels using clustering or other algorithms to form different object instances. Examples of such methods include [7, 77]. It is notable that *our* proposed "bottom-up" approach in Chapter 6 can not only be considered a method for occlusion-ordered semantic instance segmentation but for traditional instance segmentation as well, though it requires occlusion order annotation for training.

Mask R-CNN [40]

Mask R-CNN is a sophisticated and highly effective instance segmentation model that builds upon the Faster R-CNN [86] framework. The architecture of Mask R-CNN consists of five primary components, each playing a crucial role in achieving precise instance segmentation: 1- the backbone network, 2- the region proposal network (RPN), 3- the Region of Interest Align (RoIAlign) Layer, 4- the detection head, and 5- the mask head. Figure 3.4 visualizes the overal structure of Mask R-CNN model and the mentioned components.

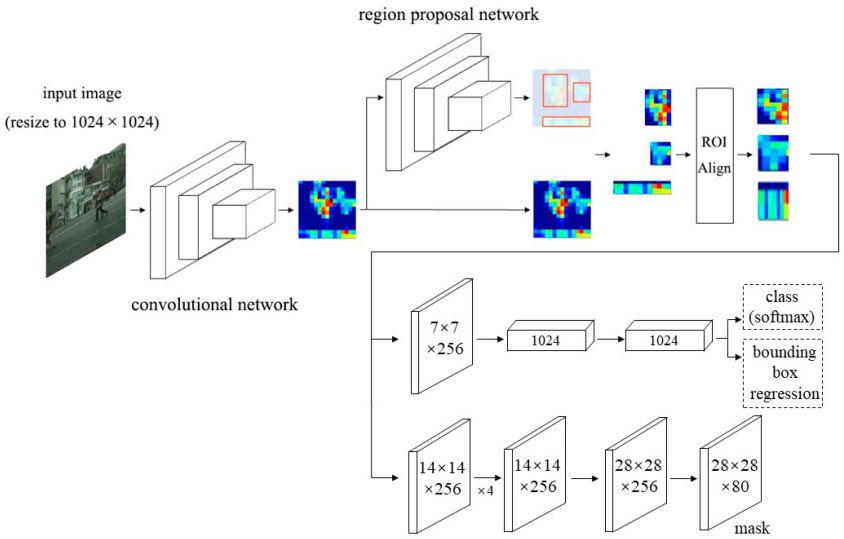


Figure 3.4: Mask R-CNN [40] instance segmentation model. Image is from [129]

The backbone network serves as the foundation of Mask R-CNN and is typically based on a deep CNN such as ResNet [42]. It processes the input image and extracts high-level feature maps that capture rich semantic information and spatial details.

The primary objective of the RPN is to propose potential bounding boxes that encompass objects, which are later refined and classified by subsequent stages of the network. It achieves this by analyzing features extracted by the backbone.

The RPN operates on a set of anchor boxes, which are predefined bounding boxes of different scales and aspect ratios that densely cover the image. These anchor boxes are generated across multiple positions on the feature map produced by the CNN backbone. For each anchor box, the RPN predicts two essential parameters: objectness scores and bounding box offsets. The objectness score indicates the probability of an anchor box containing an object of interest, while the bounding box offsets specify the necessary adjustments to align the anchor box with the object's precise location and size. To generate accurate region proposals, the RPN utilizes a combination of classification and regression losses during training. The classification loss measures the accuracy of the objectness scores, distinguishing between anchor boxes that contain objects and those that do not. The regression loss quantifies the discrepancy between the predicted bounding box offsets and the ground truth annotations. During inference, the RPN ranks the proposed region boxes based on their objectness scores. The boxes with high scores, typically surpassing a predefined threshold, are then further refined and filtered by the predicted bounding box

offsets, and non-maximum suppression, which only keeps the proposal with the highest score among highly overlapping ones.

Once the region proposals are obtained, the backbone features and proposals are passed through the Region of Interest (RoI) Align layer. This layer extracts a small feature map from the backbone features specific to the bounding box represented by each region proposal. Simply put, it splits the backbone's feature map for each proposed bounding box into a grid of equally spaced points and then uses bilinear interpolation to extract features at these points from the feature map.

The RoIAligned features are then passed to two prediction networks. The detection head and the mask head. The detection head employs fully connected layers to predict a semantic class for each proposal and represses the bounding box prediction. The mask head, however, predicts a binary mask within each proposal using a CNN that highlights the object in the region proposal.

During training, Mask R-CNN utilizes ground truth annotations to compute loss functions for both the semantic class prediction, bounding box regression, and mask prediction tasks. At inference time, given an input image, Mask R-CNN first passes it through the backbone network to extract feature maps. The RPN generates region proposals, which are then refined and pruned based on their objectness scores and overlaps. The remaining region proposals are fed into the detection and mask heads, which generate precise instance masks and classes for each object, resulting in highly accurate segmentation of object instances within the image.

3.2.2 Contour-based Instance Segmentation

Contour-based methods, including Curve GCN [64], Deep Snake [81], PolarMask [120], and LSNet [28], have recently gained attention and demonstrated promising results. These methods approach instance segmentation as a regression task, predicting the vertex coordinates of a contour represented by discrete vertices. By using a contour with a relatively small number of vertices (e.g., N = 128), these methods effectively capture the instance's shape [128]. In contrast to mask-based methods that process every pixel extensively, contour-based methods offer simplicity and require fewer computations [128]. The general framework of these methods is to initialize a first instance contour and deform it based on the features of the object's center to ideally match each instance's boundary.

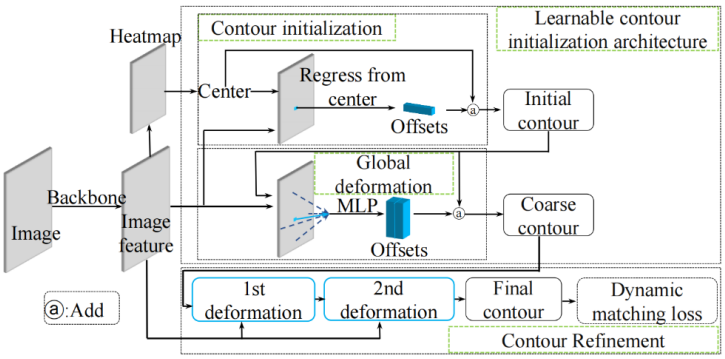


Figure 3.5: Overview of E2EC model. A learnable contour initialization architecture produces the coarse con- tour. Then, a contour refinement module produces the final contour with the supervision of DML, which is a loss function. Image and explanation are from [128]

E2EC [128]

Figure 3.5 shows the workflow of this method. An image is first passed to a backbone network to extract deep features. These features are then used to create a heatmap for each class, which highlights the points in the feature maps' space that potentially belong to each class. Note that the dimensions of feature maps is by far less than the input image. Hence, the computational cost of such a prediction is far less than a typical semantic segmentation. These heatmaps are then used, via employing non-maximum suppression, to find the candidate centers of instances.

E2EC's main contributions are the three following components. 1- Learnable Contour Initialization 2- Multi-direction Alignment 3- Dynamic Matching Loss [128].

In E2EC, unlike the prior works [120, 81] that manually design the initial contours, it is left to the network to learn the shape of the initial contour. This is done using two modules. First, the contour initialization module takes the features of the instance's center point as input and regresses the offsets for each vertex of the initial contour. These offsets represent the displacement of each vertex from the center point. By regressing the offsets, the module learns to generate an initial contour that is closer to the ground truth. Put differently, the model can arbitrarily select where to place the vertices of the contour instead of using a fixed pre-designed shape such as a circle around the center. The output of this stage is the *initial contour* as seen in Figure 3.5.

Second, the global contour deformation module refines the initial contour generated by the contour initialization module. It takes into account the features of all the vertices in

the initial contour, as well as the center point features. This global deformation allows the module to better adjust the initial contour and produce a more accurate representation of the instance boundary. This stage produces the *coarse contour* as in Figure 3.5.

The coarse contour is then passed to the contour refinement stage. To reduce learning difficulty, E2EC proposes a novel label sampling scheme called Multi-direction alignment (MDA). It fixes the directions of selected contour vertices with respect to the center point and uniformly samples between these fixed vertices to generate ground-truth vertices. MDA restricts vertex pairing and deformation paths, making learning easier while ensuring good performance. E2EC introduces a dynamic matching strategy for pairing predicted and ground-truth vertices instead of using fixed pairing. This strategy, along with the corresponding DML loss function, improves the quality of predicted boundary details by adjusting prediction points to the nearest ground-truth points and pulling them toward key label points. Explaining the details of this step is beyond the scope of this thesis. Interested readers are encouraged to see [128]

3.3 3D-Augmented Semantic Instance Segmentation

The most related to our work are [122, 46, 104]. They also propose to augment semantic instance segmentation by occlusion ordering. However, they predate deep learning and are based on simple hand-crafted strategies for occlusion order such as size, y-coordinate, and detection confidence. Our methods take advantage of deep learning both for instance segmentation and occlusion ordering. Below, we first explain three of these pre-deep learning methods. Then, we explain the only related method that leverages deep learning.

3.3.1 Pre-deep Learning

[122] introduces a layered model that addresses simultaneous instance segmentation and depth ordering. Their approach begins by applying detectors trained with support vector machines (SVMs) to obtain instance detections for each semantic class. They then propose a probabilistic model consisting of a shape model and a layer model. The shape model incorporates shape priors to ensure accurate segmentation masks, while the layer model enforces consistency in the depth ordering of the detections based on layering priors. The objective of the model is to optimize the segmentation masks and their depth ordering, such that overlaying the masks in the predicted order yields the best overall segmentation of the input image. The layer model incorporates simple heuristics to assess the quality of

the predicted depth ordering. Firstly, detections with higher detection scores are favored to have a lower depth order, as higher scores indicate fewer occluded parts in the object. Secondly, instances with lower bottom edges are encouraged to have a lower depth order. Lastly, larger objects are encouraged to have a lower depth order, as smaller objects are generally farther away. The optimization of the models is performed using the Expectation-Maximization (EM) [73] algorithm.

[46] proposes a scene parsing method inspired by the process of collaging used by digital artists to synthesize complex scenes. First, a dictionary of candidate object segments is retrieved based on their similarity to the given query image. These segments are then combined to form a "scene collage" that serves as an explanation for the query image. To make such a combination, the model assigns depth layers to each of the segments that can be warped and translated first. Finding the best collage is formulated as finding the best layered warped object segments from the dictionary that their combination is the most similar to the query image. The proposed method goes beyond pixel-level labeling and provides additional valuable information about the scene. This includes details such as the quantity of each object type present, the relationships and support among objects, and the ordinal depth of each object within the scene.

[104] initially employs a pre-deep learning scene parsing framework to generate candidate instance masks and an initial pixel labeling for the image. To select the most suitable object instances that align with the image and satisfy overlap constraints, an integer quadratic program is solved. These overlap constraints incorporate prior knowledge and statistical observations from the training set, such as cars overlapping the road or avoiding complete overlap between cars. By solving the program, a subset of object instances that best adheres to the overlap constraints is chosen. To determine the occlusion ordering, a graph is constructed with each selected object represented as a node. Edges are added between occluding pairs of objects, with weights determined based on training set statistics. These weights quantify the probability of an object from the first object's class occluding an object from the second object's class with the same overlap score. Edges with smaller weights are then removed until an acyclic graph is obtained. The topological sorting of the graph shows the occlusion ordering of the detected instances.

Obviously, these methods rely on simple cues, while our proposed methods employ deep learning that allows for richer feature learning and stronger performance.

3.3.2 Using Deep Learning

The only deep learning work augmenting instance segmentation by 3D geometry is [130]. It focuses on scenarios with only one semantic class, 'car'. Given an input image, this model outputs car instance masks and their depth ordering. For this, it divides the input image into overlapping patches at multiple resolutions. For each patch, a CNN predicts both the instance masks and depth ordering. The predicted orders are patch-based and they need to be combined into a global ordering for an image. To this end, they introduce a Markov random field (MRF) [37] framework to ensure a coherent output for the image. The MRF takes into account the predictions from the CNN for the overlapping patches and solves an energy minimization. The energy function specifically encourages connected components of the same y-coordiante to have the same depth ordering. This is a simple specific heuristic that works only for driving scenes. Such heuristic does not work on other scenes, e.g. for common object datasets like COCOA [133]. The energy function also encourages far pixels to have different labels, while closer ones are encouraged to agree on labeling. The energy is then optimized by QPBO [57] to obtain the final labeling.

Our work is different from theirs in several aspects. First, they use depth, not occlusion, ordering. This requires a training dataset with ground truth depth annotations, which is hard to obtain. Furthermore, they deal with only one semantic class. In their formulation, there is no consideration of different semantic classes for each object, e.g. 'car', 'pedestrian', etc. In our work, we design a general framework that involves detecting the correct semantic class for each predicted instance mask in addition to its occlusion ordering. Hence, we can deal with different semantic classes. Lastly, they propose one approach whereas we explore a variety of approaches with different properties.

3.4 Occlusion Order Prediction

Our top-down approach leverages pairwise occlusion order classifiers. Such classifiers [133, 62] are trained on ground-truth non-overlapping instance masks. In our top-down approach, we make use of these classifiers but have to deal with predicted and *overlapping* masks. Note that [133, 62] evaluate their pairwise order classifier, but on unseen yet *non-overlapping* ground truth data, which makes the task easier. In the following, we explain these classifiers separately.

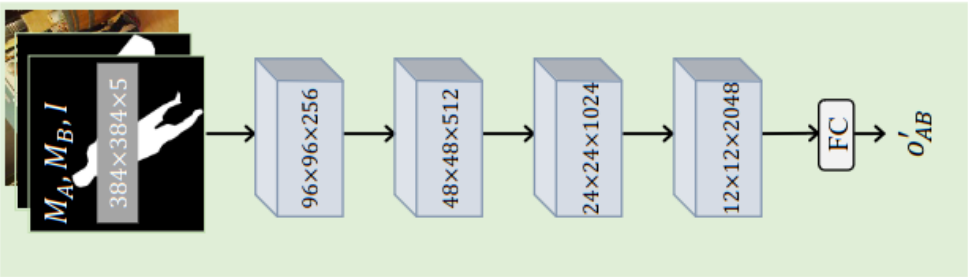


Figure 3.6: The overall architecture of InstaOrder. Image is from [62]

3.4.1 OrderNet [133]

This model takes as input two masks, one for each object instance A and B, plus the RGB image. It predicts a 3D output for each such input. The output for each sample is in form {A occludes B, B occludes A, no occlusion relation between the two}. The model uses a ResNet-50 [42] backbone and slightly changes it to account for the different number of input channels and the number of output classes.

3.4.2 InstaOrder [62]

This work also predicts the occlusion relation between two given instances. The input to this model is three parts. First, the image. Second, the mask of object instance A. Third, the mask of object instance B. Taking these three inputs, InstaOrder outputs a 2-dimensional output. The first component of the output is the probability that instance A occludes instance B. The second output is the probability that B occludes A. If none of the outputs are high, then it means there is no occlusion relation between the two instances. Otherwise, if only one of them is high, the interpretation is obvious. If both are high, it means that they are both occluding each other. Based on this, InstaOrder [62] takes bidirectional occlusion relation into consideration, whereas OrderNet [133] does not. However, such relations are extremely rare in many datasets. For example, KINS [83] and COCOA [133] datasets are entirely free of such relations.

Figure 3.6 visualizes the architecture of the InstaOrder. The mentioned inputs are stacked to make a $(H \times W \times 5)$ dimensional input. Note that masks are binary, and hence need one channel, while the RGB image has three channels. This input is processed through layers of the proposed CNN and finally, a fully connected layer predicts the output for the received inputs.

Note that both OrderNet and InstaOrder are 'naturally' trained on ground-truth masks of instances. It means that during training they optimize the model based on ground-truth masks and occlusion relations. In this thesis, our focus is on OOSIS where we want to extract the instance masks and their occlusion relations at the same time. Hence, these pairwise classifiers have to perform on predicted instance masks that can be overlapping. As they have never encountered such a case during training, they perform poorly in predicting pairwise occlusion relations for predicted masks and consequently, lead to weak performance on OOSIS. As part of our contribution, in this thesis, we identify this problem and propose a possible solution in our top-down approach.

3.5 Oriented Occlusion Boundary Prediction

Occlusion-oriented boundary models detect object boundaries and estimate the occlusion relations, i.e. which side of the boundary occludes the other side. Our bottom-up approach in Chapter 6 develops a method for oriented occlusion boundaries. There are two main groups of prior work on this task. One group treats the problem as a regression task to predict an orientation angle. These models disentangle the binary "boundary or not" prediction from the orientation prediction [113, 65, 111]. The other group of prior work treats the problem as a classification task and does not disentangle the boundary and orientation predictions [84]. Our work both treats the problem as classification and disentangles orientation and boundary prediction, outperforming the best prior work [84].

Below, we explain both approaches for occlusion-oriented boundary detection.

3.5.1 Regression-based Occlusion Oriented Boundary

Works like [113, 111] belong to this group. As stated, these methods try to regress an angle θ that determines the occlusion orientation. These models describe occlusion boundaries by orientation $\theta \in [-\pi, \pi)$ (the red arrows in Figure 3.7), which indicates occlusion relationship using the "left" rule where the left side of the arrows occludes the right side.

Figure 3.8 illustrates the architecture of [113]. This model consists of two branches. The first branch, known as the boundary branch, determines whether a pixel belongs to an occlusion boundary or not. It achieves this through binary prediction using a Sigmoid output layer. On the other hand, the second branch performs regression instead of classification. It generates a continuous value representing the angle θ for each pixel. The final

Figure 3.7: Occlusion boundaries are shown by orientation θ (the red arrows) using the 'left' rule, where the left side of the arrows occludes the right side [113]. Image is also from [113].

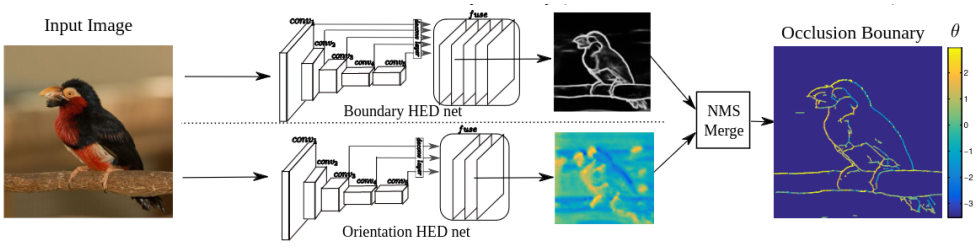


Figure 3.8: The overall architecture of a regression-based occlusion boundary model, [113]. Image is from [113].

prediction is obtained by applying non-maximum suppression to the boundary branch's output and element-wise multiplying it with the output of the orientation branch.

Now, consider $\theta_1 = \pi - \epsilon$ and $\theta_2 = -\pi + \epsilon$, where ϵ is a very small positive number. Obviously, these two angles describe a very similar occlusion orientation. However, algebraically, $|\theta_1 - \theta_2| = 2\pi - 2\epsilon$. This shows that typical regression losses cannot be used for such prediction. As a result, [113] designs a specific loss that can work with angles. Later work [84] shows classification-based works perform better using the good-behavior cross-entropy loss.

3.5.2 Classification-based Occlusion Oriented Boundary

[84] falls in this group of works. In this work, instead of making predictions for each pixel, the predictions are made for each pixel pair. This means for any two adjacent pixels, for example in a 4-neighborhood system, the model predicts if the first pixel occludes the other, or vice versa, or if there is no occlusion relation at all between the two pixels. Obviously, this can be represented as a classification task. It is sufficient to consider a 3-dimensional

prediction for each pixel pair, where each of its components corresponds to each of the described scenarios' probability.

This model, however, does not disentangle boundary and orientation prediction. For instance, if one wants to know if a pixel lies on an occlusion boundary or not, it is necessary to check all of the relations with its neighbors. In contrast, in the regression-based models, it could be easily checked by the output of the boundary branch.

Our boundary model, presented as a part of our bottom-up approach for OOSIS in Chapter 6 takes the best of both worlds. It is formulated as a classification model without the need for special losses for angles while disentangling boundary and orientation predictions and outperforms the state-of-the-art on the KINS [83] dataset.

3.6 Other Occlusion-related Work

Some works have focused on improving instance segmentation through occlusion-aware techniques [126, 53]. Our work is different in essence since our goal is to augment instance segmentation with occlusion ordering for the entire scene. Our outputs are both instance masks and their classes, and the occlusion relations among those detected objects. These models, however, just use occlusion information to improve the quality of produced instance masks. They do not output occlusion ordering for the scene. Furthermore, unlike [126], we do not need amodal annotations.

Chapter 4

Naive Approaches

Since OOSIS has not been approached post-deep learning, we first check if naive methods can effectively solve this task. We examine two such naive approaches below and show they perform poorly underlining the need for developing more carefully designed approaches that we propose in Chapters 5, and 6.

4.1 Direct CNN Training

First, we explore directly training CNN on occlusion order. We create the ground truth as follows. All background pixels are labeled with 0. All pixels in an instance are labeled with i if the maximum label of its occludees is $i - 1$. Then we train a pixel-level CNN to

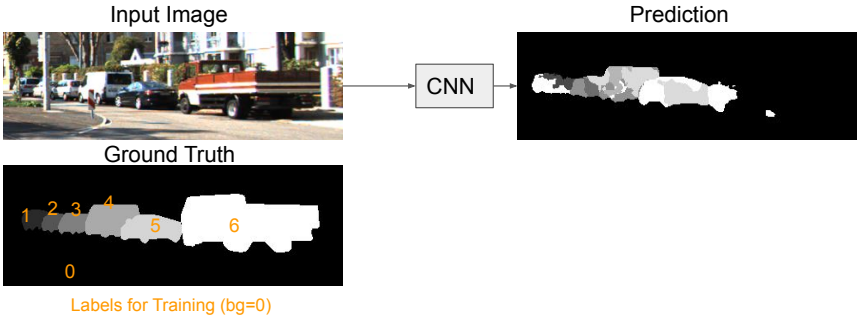


Figure 4.1: Visualization of the naive direct training of a CNN using instance and occlusion order, plus an example of how the ground truth is labeled.

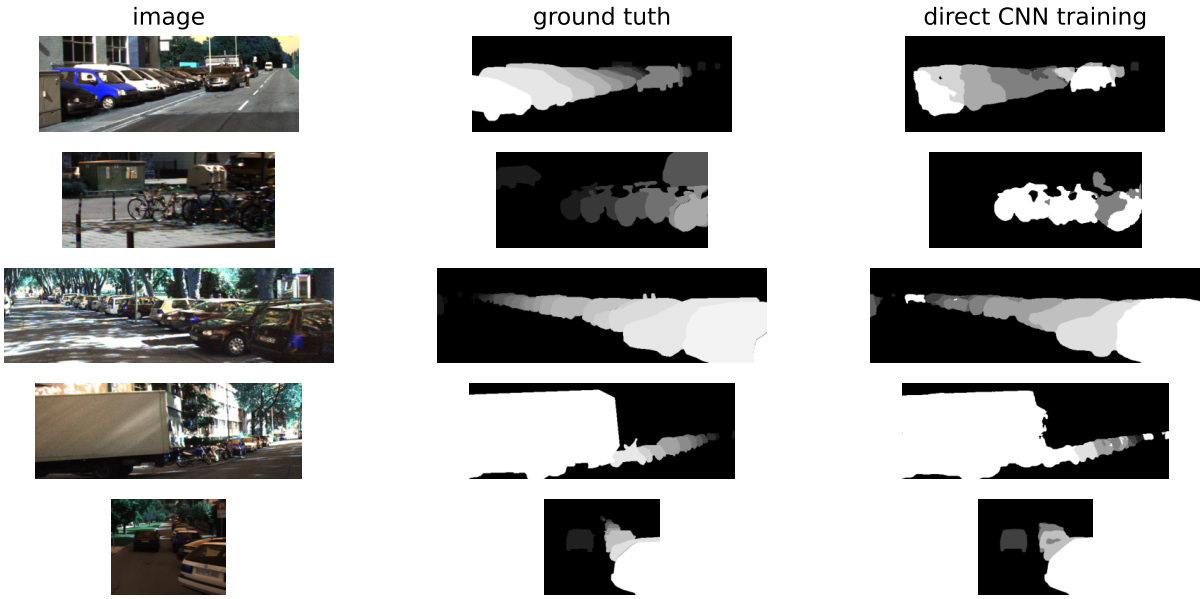


Figure 4.2: Relative depth maps for the second naive OOSIS method based on direct CNN training. This naive method performs poorly.

recover these labels using cross-entropy. We use PSPNet [132] for the CNN architecture. Figure 4.1 shows the workflow of this approach and an example of the ground truth labels.

The results of direct training are poor, see Figure 4.1 and more in Figure 4.2. This is not surprising, since OOSIS is a harder task than standard instance segmentation, for which there is no direct training approach. However, direct training for OOSIS is worth trying since the ground truth for OOSIS has extra (occlusion) information and occlusion-based labels are somewhat meaningful.

4.2 Clustering Monocular Depth Estimation

The second naive approach, illustrated in Figure 4.3, is as follows. We train a semantic segmentation CNN [132], and also apply a pre-trained monocular depth estimator [85]. We use PSPNet [132] for semantic segmentation and MiDaS (v21-384) [85] model for the depth estimator. Using the semantic segmentation model, we can find the non-background pixels. We intersect these pixels with the depth map. Then we use k-means to cluster the depth maps of the non-background pixels. The clusters give us the instances and the

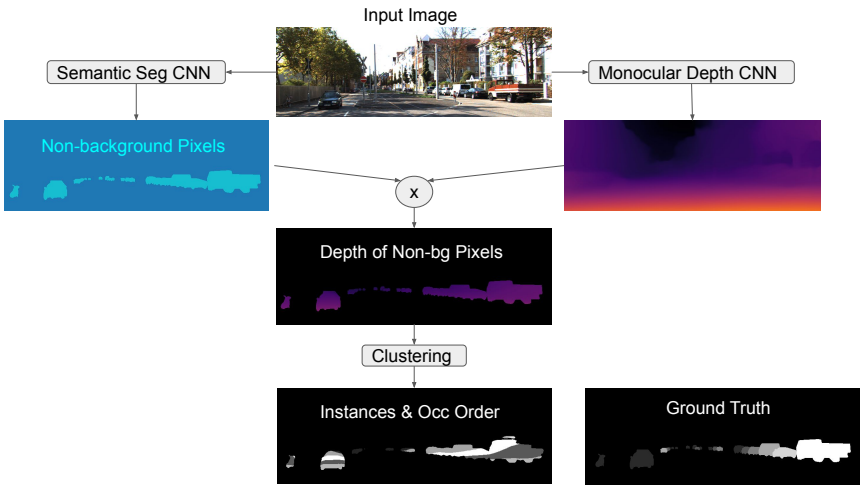


Figure 4.3: The workflow of the second naive approach where semantic segmentation and monocular depth estimation are done using a CNN. Then k-means clustering is performed on the depth estimation of the non-background pixels. The poor performance underscores the inadequacy of monocular depth estimation for recovering object-level relative depth relations.

average depth of each cluster is used for the occlusion ordering of adjacent clusters. We assign the dominant semantic class to these instances.

The results are poor, see Fig. 4.4. Monocular depth estimation has limited accuracy and is inadequate for instance segmentation. In addition, depth resolution may be insufficient for distinguishing between instances, as discussed in Chapter 1. Figure 4.4 clearly shows how the model is not only unable to extract instances but also to order them. The shortcomings of these simple methods show that OOSIS requires a specialized and tailored approach.

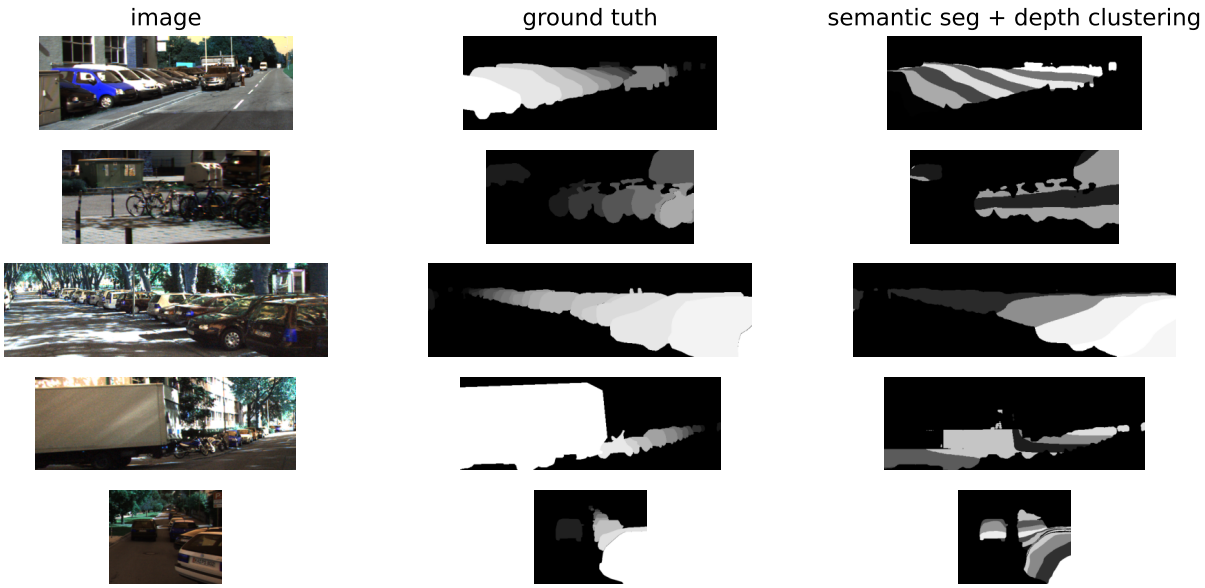


Figure 4.4: Relative depth maps for the second naive OOSIS method based on monocular depth clustering. This naive method performs poorly.

Chapter 5

Top-Down Occlusion-Ordered Semantic Instance Segmentation

The top-down approach consists of three stages, see Fig. 5.1. First, we use an off-the-shelf standard semantic instance segmentation to obtain instance masks and classes. For this step, we evaluate an established Mask-RCNN [41] and a state-of-the-art recent contour-based method E2EC [128]. We want to use the instance masks generated by such models and use pairwise occlusion ordering classifiers [133, 62] to detect the occlusion relations among the instances. However, directly using the produced instance masks does not work. We elaborate on this below.

The state-of-the-art semantic instance segmentation models, like Mask R-CNN and E2EC, can produce instance masks that are overlapping. See Figure 5.1 for some examples. It means that a pixel can belong to more than one instance. Hence, the labeling of pixels is *ambiguous*. In detection tasks, this is not usually a problem. However, we determined that it is essential to resolve the overlap between masks to obtain a good pairwise occlusion classification since the occlusion classifier is 'naturally' trained on ground truth masks, and they do not overlap, see experiments in Chapter 7. Thus the next step is mask overlap resolution. For removing overlap, we tested three approaches: assigning the overlap randomly, or to the instance with the larger confidence, or to the instance with the smaller confidence. We found that sorting instances by confidence and removing overlaps in the order of decreasing confidence works the best. In the future, CRF optimization [10] can be explored for this purpose.

As stated, the final step is to determine the occlusion order. A naive global occlusion ordering in Chapter 4 fails. A feasible alternative is to apply a pairwise occlusion

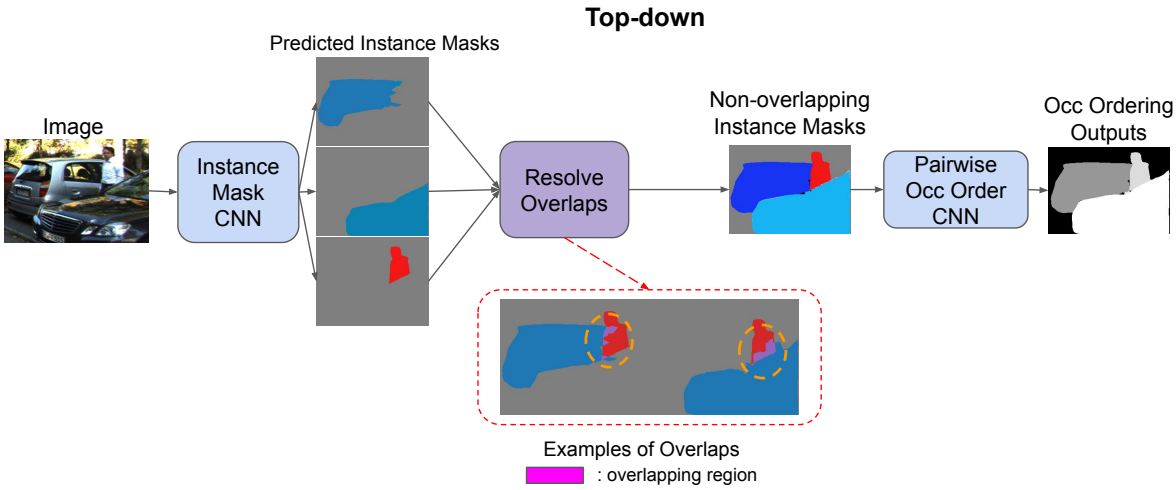


Figure 5.1: Overview of our top-down approach. The image is first given to a deep CNN to produce instance masks, and their semantic classes. The classes are not shown in the image for simplicity. The produced instance masks can be overlapping. Hence, they cannot be directly used by pairwise occlusion order classifiers. We develop a component that resolves such overlaps. The output of this component has non-overlapping instance masks and their corresponding semantic classes. These masks are then fed to a pairwise occlusion order classifier to predict the occlusion relation between each pair of instances. The result of the occlusion ordering is shown by a relative depth map where each instance is brighter than all of its occludees. The instance mask CNN and pairwise occlusion order classifiers are from prior works as discussed in Chapter 3.

order classifier, which takes as an input two neighboring instance masks and estimates which one is the *occluder*, and which one is the *occludee*. We tested *OrderNet* [133] and *InstaOrder* [62] for this step.

Occlusion ordering from the pairwise classifier is not necessarily globally consistent and may have cycles. Occlusion cycles in the ground truth are exceedingly rare, and the overwhelming majority of cycles are due to errors in pairwise predictions. In principle, we could break the cycles until the occlusion graph is cycle free. However, other than random, it is hard to come up with an efficient intelligent strategy. For example, one may want to remove the smallest set of edges which results in an acyclic ordering graph. However, this is an NP-hard 'feedback arc set' problem. We leave cycle removal as a future work. For visualization as a relative depth map, we must break cycles, which we do, randomly. The accuracy metrics in Chapter 7 are computed using the original set of ordering relations.

The advantage of the top-down approach is that it uses state-of-the-art methods for instance segmentation, resulting in accurate masks. The disadvantage is that the occlusion order is not globally consistent and, therefore, likely to contain cyclic errors.

Chapter 6

Bottom-Up Occlusion-Ordered Semantic Instance Segmentation

The bottom-up approach groups pixels into instances by assigning an occlusion order to each pixel. The approach is summarized in Figure 6.1 and consists of two stages. First, we design a novel method that simultaneously predicts semantic segmentation and oriented occlusion boundaries. Second, based on these predictions, we formulate CRF energy for labeling pixels with their occlusion order, simultaneously inferring instances and their occlusion order.

In the following, we first explain our designed deep CNN for joint semantic segmentation and oriented occlusion boundary prediction in Section 6.1. This explanation includes the model in Section 6.1.1, the training loss function in Section 6.1.2, and the architecture of the model in Section 6.1.3. Then, we elaborate on our energy function and CRF formulation producing the final results in Section 6.2. To accomplish this, we initially outline our proposed energy function and its optimization process using jump moves in Section 6.2.1. Subsequently, we provide a discussion on how we construct a submodular upper bound for our energy function to enable effective optimization by jump moves in Section 6.2.2. Lastly, we draw a comparison between jump moves and expansion moves in relation to our energy function, offering a justification for the utilization of jump moves in Section 6.2.3.

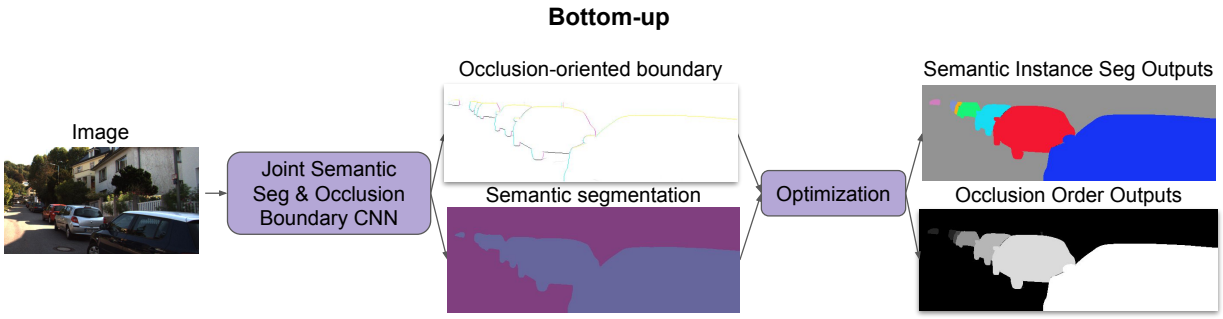


Figure 6.1: The workflow of our bottom-up approach. Our novel deep CNN predicts occlusion-oriented boundaries and semantic segmentation for an input image. Then, our novel optimization formulation uses the previous stage's outputs to generate the semantic instance masks and their occlusion ordering. The occlusion ordering is visualized by a relative depth map where each object is brighter than all of its occludes. In the semantic instance segmentation output, a class is predicted for each extracted instance mask. These classes are not shown in this figure for simplicity.

6.1 Joint Semantic Segmentation and Oriented Occlusion Boundary Model

Now, we introduce our deep model for jointly learning semantic segmentation and oriented occlusion boundaries. In Figure 6.2, we showcase the outputs of this model. The semantic segmentation map assigns each pixel with its respective semantic class, while an additional map highlights boundary pixels along with the occlusion orientation at those pixels, enabling us to infer the occluded neighbors for each pixel. For detecting occlusion oriented boundaries we both treat the problem as a classification task (instead of a regression one) and disentangle the boundary presence and orientation predictions. This way we take the best of both worlds, while prior work is just focused on either of them. For more clarification on how we are different from prior work, see Chapter 3.5.

6.1.1 Model

The orientation of occlusion at a boundary pixel can be represented by the normal vector of the boundary, pointing from the occluder to the occludee. Here we introduce three random variables for our model: $\mathbb{S}_p$, $\mathbb{B}_p$, and $\mathbb{O}_p$, corresponding to each pixel p. Variable $\mathbb{S}_p$ represents the semantic class of pixel p and variable $\mathbb{B}_p$ is a binary indicator showing

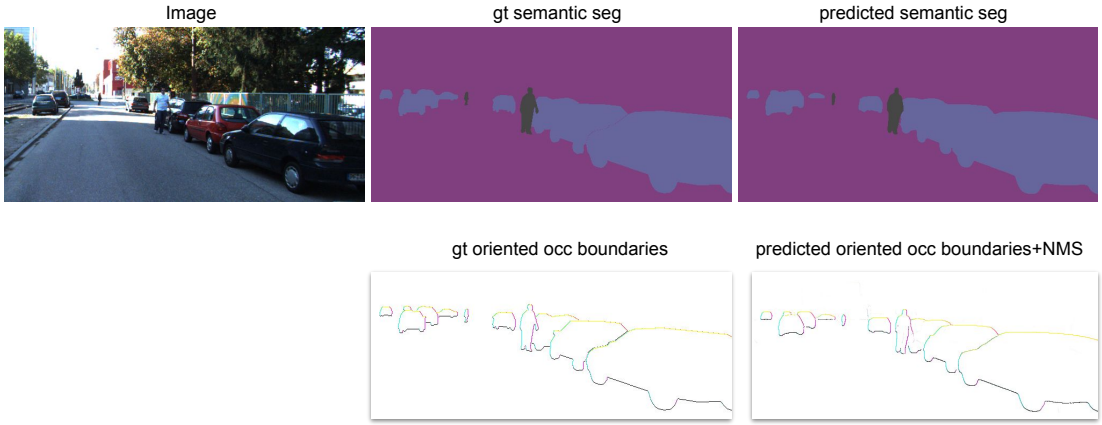


Figure 6.2: Illustration of our semantic segmentation and oriented occlusion boundaries for the bottom-up OOSIS. The images are: input, ground truth and our semantic segmentation, ground truth and our oriented occlusion boundaries after non-maximum suppression. The color scheme for the boundaries is: left-cyan, top-yellow, right-magenta, bottom-black. Best viewed zoomed in.

whether pixel p lies on an occlusion boundary. We also introduce an *oriented boundary* variable $\mathbb{O}_p \in \bar{D}$ where $\bar{D} = D \cup \{\emptyset\}$, see Figure 6.3. Variable $\mathbb{O}_p$ serves a dual purpose: besides indicating "no boundary" $\emptyset$, in case of the boundary it indicates a specific orientation of its normal, an outcome in the set of all possible orientations D, e.g. discretized bins of the whole 360-degree spectrum. Figure 6.3 shows the relationship between random variables $\mathbb{O}_p : \Omega \to \bar{D}$ and $\mathbb{B}_p : \Omega \to \{0, 1\}$. It also implies the following equations

$$\Pr(\mathbb{O}_p = \emptyset) \quad = \quad \Pr(\mathbb{B}_p = 0) \tag{6.1}$$

$$\Pr(\mathbb{O}_p = d) \quad = \quad \Pr(\mathbb{O}_p = d \,|\, \mathbb{B}_p = 1) \Pr(\mathbb{B}_p = 1) \qquad \forall d \in D \tag{6.2}$$

Our objective is twofold: we aim to predict $Pr(\mathbb{S}_p)$, which represents the probability distribution over each semantic class for pixel p, and $Pr(\mathbb{O}_p)$, which represents the distribution of the normal orientation for a boundary pixel and also represents the probability of it not being a boundary.

Figure 6.4 provides a simplified visual representation of our model, which we will now elaborate on. To obtain the two final predictions we seek, our PSPNet [132]-based model incorporates three interconnected heads:

1. Head "s": estimates of $Pr(\mathbb{S}_p)$ for each pixel p. This is similar to a standard semantic segmentation task.

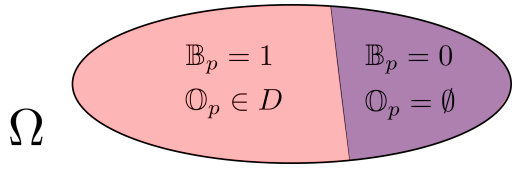


Figure 6.3: Oriented Occlusion Boundary $\mathbb{O}_p$ as a random variable. It is defined as a function $\mathbb{O}_p : \Omega \to \bar{D}$ over the probability space Ω of all elementary outcomes. Its range $\bar{D} := D \cup \{\emptyset\}$ includes all possible boundary directions D and a "no-boundary" outcome denoted by $\emptyset$. The diagram above shows the relationship between the boundary $\mathbb{B}_p$ and the oriented boundary $\mathbb{O}_p$ by illustrating the equivalence between the following pairs of events $\mathbb{B}_p = 1 \Leftrightarrow \mathbb{O}_p \in D$ and $\mathbb{B}_p = 0 \Leftrightarrow \mathbb{O}_p = \emptyset$.

2. Head "b": estimates $Pr(\mathbb{B}_p = 1)$, the probability that pixel p is an occlusion boundary.

3. Head "e": estimates the conditional distribution $Pr(\mathbb{O}_p = d|\mathbb{B}_p = 1)$, which is the distribution of the normal at p over $|D|$ bins, assuming p is a boundary.

Using the mentioned three heads, we produce the two mentioned final objectives as follows. First, $\Pr(\mathbb{S}_p)$ is estimated by the head 's' directly for each pixel, and no further processing is required. Second, for the oriented occlusion boundaries, we need to obtain *oriented boundary* prediction $o_p := Pr(\mathbb{O}_p)$, which is a distribution over $|D| + 1$ values. The basic probability relations in Eqs. (6.1),(6.2) imply a simple formal relation of this prediction to $b_p = \Pr(\mathbb{B}_p = 1)$ and to conditional orientation distribution $e_p = \Pr(\mathbb{O}_p = d|\mathbb{B}_p = 1)$ defined in the list of heads above. Indeed, separating $|D| + 1$ components of vector o_p into two parts: $|D|$ values corresponding to the probabilities $\Pr(\mathbb{O}_p = d)$ for $d \in D$ and an extra value corresponding to $\Pr(\mathbb{O}_p = \emptyset)$, equations (6.1),(6.2) give

$$o_p := [b_p e_p, 1 - b_p], \tag{6.3}$$

where $[,]$ is the concatenation of values.

6.1.2 Loss

We denote the ground truth for s_p, b_p, and e_p by S_p, B_p, and E_p, respectively. Additionally, as explained, we called the final output of our model "o". Let O_p indicate the ground truth for it.

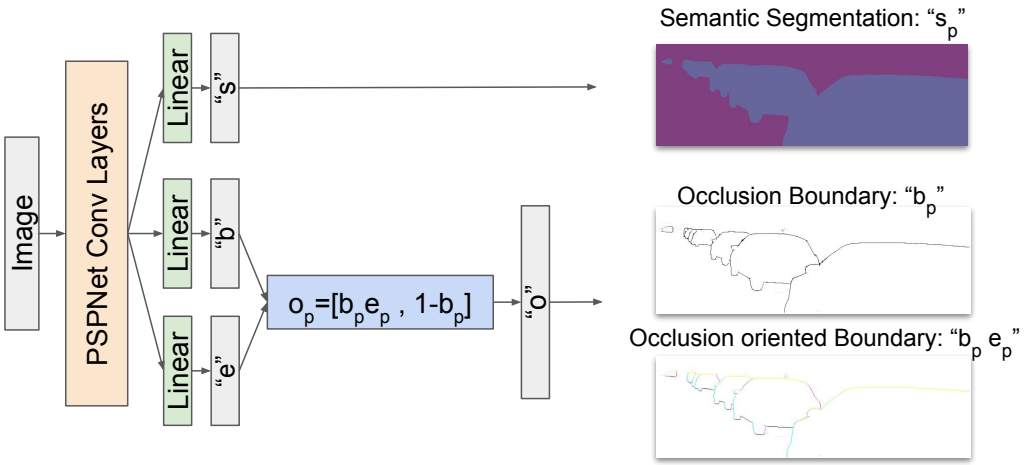


Figure 6.4: The overall structure of our joint semantic segmentation and occlusion oriented boundary model. Given an input, the model has three heads: s, b, e. s_p predicts the probability of each semantic class for pixel p. b_p estimates the probability that p is on an occlusion boundary. e_p estimates the probability of different possible orientations assuming p is a boundary pixel. The final output of occlusion oriented boundary o_p incorporates the probability that p is not a boundary $1 - b_p$ and the probability that p is a boundary and has any of possible orientations $b_p e_p$. The color scheme for the oriented boundaries is: left-cyan, top-yellow, right-magenta, bottom-black. Best viewed zoomed in.

Theorem 1. *Denoting cross-entropy by CE, we prove the following:*

$$\sum_p CE(O_p, o_p) = \sum_p CE(B_p, b_p) + B_p CE(E_p, e_p) \tag{6.4}$$

Proof. Following from what we explained for the random variable $\mathbb{O}_p$, and Figure 6.3, we have

$$\sum_p CE(O_p, o_p) = \sum_p -(1 - B_p)\ln(1 - b_p) + \sum_p \sum_{d \in D\ bins} -B_p E_p^d \ln b_p e_p^d \tag{6.5}$$

Now, consider only the last term:

$$\begin{aligned}
\sum_{d\in D\ bins} -B_p E_p^d \ln b_p e_p^d &= \sum_p B_p \sum_{d\in D\ bins} -E_p^d(\ln b_p + \ln e_p^d) \\
&= \sum_p B_p \sum_{d\in D\ bins} -E_p^d \ln b_p - E_p^d \ln e_p^d \\
&= \sum_p B_p \sum_{d\in D\ bins} -E_p^d \ln b_p + \sum_p B_p \sum_{d\in D\ bins} -E_p^d \ln e_p^d \\
&= \sum_p -B_p \ln b_p \sum_{d\in D\ bins} E_p^d + \sum_p B_p \sum_{d\in D\ bins} -E_p^d \ln e_p^d
\end{aligned} \tag{6.6}$$

Now, as we know E_p is a conditional distribution, and it sums to 1, we have:

$$\sum_{d\in D\ bins} -B_p E_p^d \ln b_p e_p^d = \sum_p -B_p \ln b_p + \sum_p B_p \sum_{d\in D\ bins} -E_p^d \ln e_p^d \tag{6.7}$$

Now, we substitute this back in Eq.6.5

$$\begin{aligned}
\sum_p CE(O_p, o_p) &= \sum_p -(1-B_p)\ln(1-b_p) + \sum_p -B_p \ln b_p + \\
&\quad \sum_p B_p \sum_{d\in D\ bins} -E_p^d \ln e_p^d \\
&= \sum_p CE(B_p, b_p) + \sum_p B_p CE(E_p, e_p) \\
&= \sum_p CE(B_p, b_p) + B_p CE(E_p, e_p)
\end{aligned}$$

□

This suggests that instead of applying cross-entropy (CE) loss directly on the predictions o, we can apply it separately on the predictions b and e. However, it is important to note that the cross-entropy loss for e should only be computed for the ground-truth boundaries. This distinction aligns with the fact that the ground truth O_p is defined for all pixels, whereas E_p is only defined for ground-truth boundaries.

To train our three classification heads s, b, and e, we use a cross-entropy loss defined above. Note that [113] also separate the boundary and its orientation, but they predict the angles by regression also trained on boundary points only. We use classification, which often works better, and provide a probabilistic derivation for the loss for the conditional

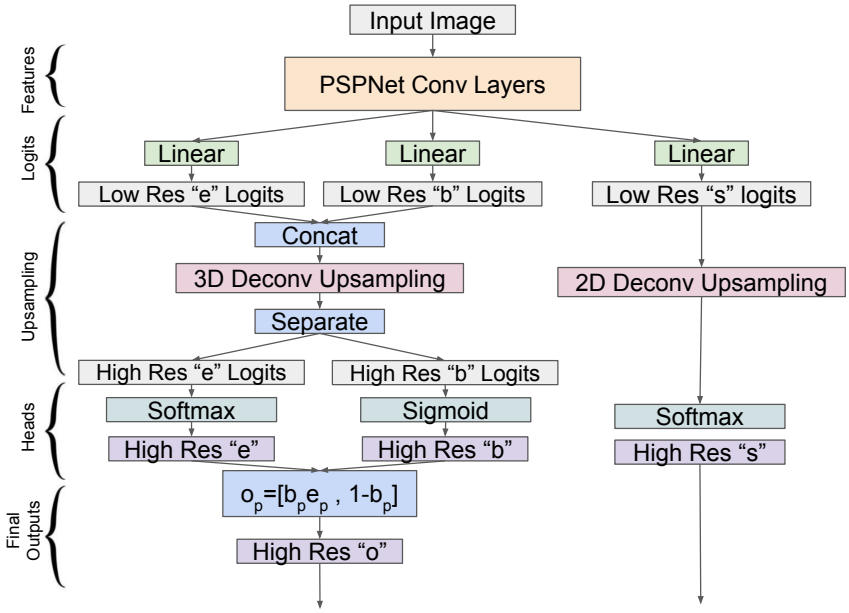


Figure 6.5: The detailed structure of our designed joint semantic segmentation and occlusion oriented boundary model.

head e. For the ‘b’ head, we replace the cross-entropy term with weighted cross-entropy loss, wCE, due to imbalanced ground truth where most pixels are non-boundaries. It has been shown to provide an improved performance in such cases. For the s head, we employ standard cross-entropy loss. The total loss is hence,

$$\mathcal{L} = \sum_{p} CE(S_p, s_p) + wCE(B_p, b_p) + B_p CE(E_p, e_p). \tag{6.8}$$

For simplicity, we use $D = 4$ orientation bins, which can be also interpreted as left, right, top, and bottom neighbor occlusions, analogous to [84]. For details on how we acquire the ground-truth occlusion orientations, see Chapter 7.12. For weighted CE, we use weight $= 0.9$ for positive samples, i.e. boundary samples, and wight $= 0.1$ for non-boundary samples.

6.1.3 Architecture

In accordance with Figure 6.5, we provide a detailed explanation of the model stages.

Features: The input image is processed by PSPNet [132] to extract deep features that capture relevant information from the image. **Logits:** Three separate linear layers are employed, each producing logits for the corresponding heads described earlier. It is important to note that as the output of PSPNet is of low resolution, we need to upsample the logits in the subsequent stage. **Upsampling:** For the "s" logits, we utilize 2D (Spatial) Deconvolution upsampling, also known as transposed convolution. The upsampling is applied separately for each channel (each semantic class), as denoted by the "2D" descriptor. We initialize the upsampling kernel with bilinear upsampling weights. On the other hand, for upsampling the "b" and "e" logits, which contain complementary information, we concatenate them and perform 3D Deconvolution upsampling. The "3D" nature of the kernel indicates that the channels (representing different orientation bins and "b") effectively utilize each other's information. Finally, we separate the upsampled "b" and "e" logits. **Heads:** Non-linearities are applied to the logits at this stage. Specifically, softmax is used for the "s" and "e" logits, while sigmoid activation is employed for the "b" logits. **Final Outputs:** They are produced as explained in Section 6.1.1 by: s_p estimating $\Pr(\mathbb{S}_p)$ and $o_p := [b_p e_p, 1 - b_p]$ estimating $\Pr(\mathbb{O}_p)$.

6.2 Occlusion Order-based Instance Grouping

6.2.1 Energy Function and Optimization

The input to this stage is the semantic segmentation and oriented occlusion boundaries from the previous stage, see Figure 6.2. We discard boundaries below a threshold of 0.1. Since we have only vertical and horizontal orientations, we store boundaries as a set of ordered pixel pairs, called the occlusion set: $\mathcal{O} = \{(p, q) \,|\, p \text{ occludes } q\}$.

We formulate the task of instance segmentation and occlusion ordering in CRF framework [10]. We assign a label x_p to each pixel p by formulating and minimizing an energy function over pixel-label assignments. The labels are non-negative integers and denote the occlusion order. If $x_p = 0$, then pixel p is the background. Positive x_p means that pixel p belongs to an instance. Given two neighboring pixels p, q, if $x_p > x_q$, then p occludes q. A connected component of pixels with the same label forms an instance. We can have many instances with an equal label, but they cannot be immediately adjacent spatially. This is a drawback of our formulation, but usually, instances that are immediately adjacent are involved in an occlusion relation and have distinct occlusion order which separates them, see, for example, the ground truth labels in Figure 7.1. To determine the instance class, we take the majority vote on the semantic segmentation limited to the instance pixels.

Let $\mathcal{P}$ be the set of pixels, and $\boldsymbol{x} = (x_p \mid p \in \mathcal{P})$ be a labeling vector. The energy is defined for $\boldsymbol{x}$ and consists of unary and pairwise terms. For each pixel p there is a unary term $u_p(x_p)$. It is small if label x_p is likely for p and large otherwise. We derive unary terms from semantic segmentation, even though it outputs a probability distribution over classes, not occlusion-order labels. However, the occlusion-order label 0 is identified with the background class and can be used for unary terms. Let σ_p be the probability for p to be the background according to semantic segmentation. We set

$$u_p(x_p) = (1 - \sigma_p) \cdot [x_p = 0] + \sigma_p \cdot [x_p \neq 0], \tag{6.9}$$

where $[\cdot]$ is Iverson bracket, equal to 1 if its argument is true and to 0 otherwise. If σ_p is large, the unary term for label 0 is small, encouraging p to be assigned to the background. If σ_p is small, then p is encouraged to be assigned to any label other than 0. There is no preference to any particular non-zero label as semantic segmentation is not informative about any label other than 0.

There are two types of pairwise terms: smoothness v and occlusion o. The smoothness terms v are modeled as in [10]. They encourage spatially coherent labeling by penalizing neighboring pixels that do not have the same label. Let p, q be neighbors. We define

$$v(x_p, x_q) = [x_p \neq x_q]. \tag{6.10}$$

The occlusion terms o model the occlusion set $\mathcal{O}$. Let $(p, q) \in \mathcal{O}$, i.e. p occludes q. We assign a prohibitive cost if the label of q is larger than that of p, and a negative cost if the label of p is larger

$$o(x_p, x_q) = c_\infty \cdot [x_p < x_q] - [x_p > x_q], \tag{6.11}$$

where c_∞ is prohibitively large. Since we are minimizing the energy, a negative cost is 'repulsive' [125], and we lower the energy if p gets assigned a larger label than q. Thus, occlusion terms encourage labeling boundaries, facilitating creation of instances, unlike the smoothness terms, which discourage boundaries. But labeling boundaries are encouraged *only* in places where we detect occlusion boundaries, which is important to maintain the overall spatial coherence of the labeling.

The best labeling $\boldsymbol{x}$ is found by minimizing the energy

$$E(\boldsymbol{x}) = \sum_{p \in \mathcal{P}} u_p(x_p) + \lambda_v \sum_{(p,q) \in \mathcal{N}} v(x_p, x_q) + \lambda_o \sum_{(p,q) \in \mathcal{O}} o(x_p, x_q), \tag{6.12}$$

where $\mathcal{N}$ is the set of pairs on an 4-connected grid. We set $\lambda_v = 20$ and $\lambda_o = 100$.

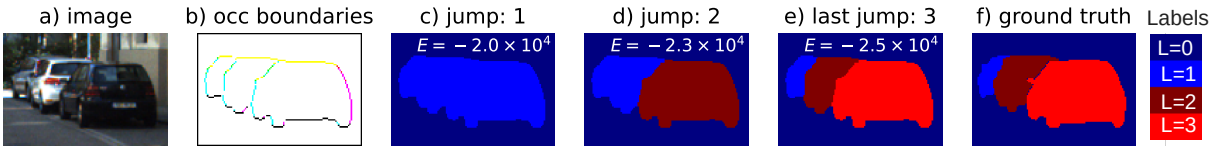


Figure 6.6: Jump move optimization. Warmer colors are larger labels. Energies for labelings are listed.

Figure 6.6 illustrates how our energy in Eq. (6.12) encourages instances along occlusion boundaries. The energy decreases when across occlusion boundaries, the occluder has a label larger than that of the occludee. Consider labelings (c, d, e). Labeling (a) has only one instance which groups all cars together. It does not take advantage of the occlusion boundaries between the cars. These 'missed' occlusion boundaries can be used to lower the energy. Labeling (d) has two instances and, therefore, is able to take advantage of some in-between car boundaries. Labeling (e) has three instances and takes advantage of most occlusion boundaries, leading to the lowest energy.

We minimize the energy in Eq (6.12) with the jump move algorithm [109], but see [99] for minimization methods review. Given a labeling $\boldsymbol{x}$, we say that $\boldsymbol{x}'$ is a jump move from $\boldsymbol{x}$ if whenever $x_p \neq x'_p$, then $x'_p = x_p + 1$, i.e. a jump move allows any pixel to increase its label by 1. There is an exponential number of jumps. An optimal jump move decreases the energy the most. It is computable with a graph-cut [56] if submodularity holds [9]. However, the smoothness terms s are nonsubmodular when $x_p \neq x_q$. We tried QPBO [57] for nonsubmoduar energies, but it did not work well. Instead, we replace nonsubmodular terms by a submodular upper bound, enabling graph-cut minimization. An optimal move is not guaranteed, but the energy is guaranteed to decrease, see Section 6.2.2.

The jump move algorithm works as follows. We initialize all pixels to label 0. Then we run a series of jump moves until the labeling stops changing. We say that $(p, q) \in \mathcal{O}$ is *activated* if $x_p > x_q$ so that $o(x_p, x_q)$ contributes -1 to the energy. The more terms in $\mathcal{O}$ are activated, the lower is the energy. Consider Figure 6.6. The first jump, Figure 6.6(c), switches to label 1 a segment of pixels which have a low background probability and are also correctly aligned to the oriented occlusion boundaries, creating one instance. With only one instance, many $(p, q) \in \mathcal{O}$ stay not activated. Therefore, the second jump move, in Figure 6.6(d), assigns pixels in the front-most car to label 2, activating more $(p, q) \in \mathcal{O}$ and decreasing the energy. The third jump move, Figure 6.6(e), increases by 1 the label of the frontal car, and the car behind it, and now most $(p, q) \in \mathcal{O}$ are activated. Further jump moves do not change the labeling. If we segment another instance, it would align mostly to $(p, q) \notin \mathcal{O}$, and we would have to pay smoothness costs $s(x_p, x_q)$ along most of

the segment boundary, increasing the energy.

6.2.2 Binary Energy for Jump Move and its Submodular Upper Bound

We now explain the binary energy for the jump move, why it is not submodular [9], and how we construct its submodular upper bound.

Given a labeling $\boldsymbol{x}$, a jump move either does not change the label of a pixel, or increases the label by 1. We pose the problem of finding the optimal jump move as binary energy minimization. Let $\boldsymbol{x}^c$ be the current labeling for which we wish to find the optimal jump move. We introduce a binary variable y_p for each pixel p, and collect all these variables into a vector $\boldsymbol{y}$. We define one-to-one correspondence between the set of jump moves from $\boldsymbol{x}^c$ and the set of all possible labelings of $\boldsymbol{y}$ as follows. For each labeling $\boldsymbol{y}$, let us denote the corresponding jump move as $m(\boldsymbol{x}^c; \boldsymbol{y})$, defined as

$$m(\boldsymbol{x}^c; \boldsymbol{y}) = \begin{cases} x_p^c & \text{if } y_p = 0 \\ x_p^c + 1 & \text{otherwise} \end{cases} \tag{6.13}$$

In words, label 0 is identified with a pixel keeping its label, and label 1 with pixel increasing its label by 1.

In the binary energy, let us use $\hat{u}, \hat{v}, \hat{o}$ to denote the unary, pairwise smoothness and occlusion terms, which are derived from the corresponding terms in the multi-label energy in Eq. (6.12).

We define the unary terms of binary energy as

$$\hat{u}(0) = u(x_p^c),\ \hat{u}(1) = u(x_p^c + 1). \tag{6.14}$$

Given a pixel pair $(p, q) \in \mathcal{N}$, we define the smoothness pairwise terms of the binary energy as

$$\hat{v}(0,0) = v(x_p^c, x_q^c),\ \hat{v}(0,1) = v(x_p^c, x_q^c{+}1),\ \hat{v}(1,0) = v(x_p^c{+}1, x_q^c),\ \hat{v}(1,1) = v(x_p^c{+}1, x_q^c{+}1). \tag{6.15}$$

Given a pixel pair $(p, q) \in \mathcal{O}$, we define the occlusion pairwise terms of the binary energy as

$$\hat{o}(0,0) = o(x_p^c, x_q^c),\ \hat{o}(0,1) = o(x_p^c, x_q^c{+}1),\ \hat{o}(1,0) = o(x_p^c{+}1, x_q^c),\ \hat{o}(1,1) = o(x_p^c{+}1, x_q^c{+}1). \tag{6.16}$$

We define the binary energy $\boldsymbol{y}$ as

$$E(\boldsymbol{y}) = \sum_{p \in \mathcal{P}} \hat{u}_p(y_p) + \lambda_v \sum_{(p,q) \in \mathcal{N}} \hat{v}(y_p, y_q) + \lambda_o \sum_{(p,q) \in \mathcal{O}} \hat{o}(y_p, y_q). \tag{6.17}$$

It is straightforward to check that $E(\boldsymbol{y}) = E(\mathbf{m}(\boldsymbol{x}^c, \boldsymbol{y}))$. Therefore the optimal jump move from $\boldsymbol{x}^c$ corresponds to the labeling $\boldsymbol{y}^*$ minimizing Eq.(6.17).

If a binary energy is submodular [9] then it can be optimized with a graph-cut [56]. For submodularity to hold, there are no conditions on the unary terms, but each pairwise term f must satisfy

$$f(0,0) + f(1,1) \leq f(0,1) + f(1,0). \tag{6.18}$$

It is straightforward but tedious to check that the occlusion pairwise terms $\hat{o}$ are submodular in all cases. Unfortunately, the smoothness term $\hat{v}$ is not submodular for any pair of pixels $(p,q) \in \mathcal{N}$ such that $x_p^c + 1 = x_q^c$. Indeed, in this case

$$\begin{aligned} \hat{v}(0,0) + \hat{v}(1,1) &= v(x_p^c, x_q^c) + v(x_p^c + 1, x_q^c + 1) = 1 + 1 && (6.19) \\ &> v(x_p^c, x_q^c + 1) + v(x_p^c + 1, x_q^c) = 1 + 0 && (6.20) \\ &= \hat{v}(0,1) + \hat{v}(1,0) && (6.21) \end{aligned}$$

To make optimization feasible with a graph-cut, we replace the energy in Eq. (6.17) with its submodular upper bound. For any $(p,q) \in \mathcal{N}$, if $x_p^c + 1 = x_q^c$, we replace $\hat{v}$ corresponding to this pair of pixels by a constant pairwise term c which always takes value 1 and therefore, is submodular. That is $c(a,b) = 1$ for any values of a, b. In practice, it is equivalent (the energies are equal up to a constant) to omitting the smoothness term $\hat{v}$ for any pixels $(p,q) \in \mathcal{N}$ s.t. $x_p^c + 1 = x_q^c$.

Intuitively, our approximation means the following. If in the current labeling $\boldsymbol{x}^c$, we have a neighboring pair of pixels whose label differs by exactly 1, then the jump move is not able to 'see' that it can achieve a lower-energy labeling by increasing the smaller label by 1. Minimizing an upper bound as opposed to the exact binary energy means that the optimal jump move is not guaranteed if there are neighboring pixel pairs in the current labeling whose labels differ by exactly 1. However the energy is guaranteed to decrease (or stay the same).

For details on how to construct the graph for submodular binary function minimization, see [56]. We use the graph-cut/max-flow algorithm of [11] to compute the minimum cut.

6.2.3 The Expansion vs. Jump move Algorithms

For minimizing CRF energies of the type of Eq. (6.12) the expansion algorithm [10] is frequently used. The expansion move has approximation guarantees and has been shown superior to other minimization algorithms for many energy types [98]. However, we found that the expansion algorithm does not work as well for our energy as the jump move, and now explain the reasons.

The expansion algorithm performs a sequence of α-expansion moves until convergence. Given the current labeling and some label α, an α-expansion move finds the subset of pixels to switch to label α s.t. the energy decreases by the largest amount.

Typically, one starts with a labeling where all pixels are assigned label 0, and then performs a series of cycles, where each cycle consists of one iteration over labels in $\{0, 1, 2, ..., l_{\max}\}$. Here l_{max} is the largest possible label. Notice that iterations are performed not necessarily in the consecutive order of labels. In general, the expansion moves for our energy are not submodular. However, it is straightforward to show that if we start with the labeling where each pixel is assigned 0, and then expand on labels $1, 2, ..., l_{\max}$, in that order, then each expansion is submodular, i.e. one cycle of the expansion algorithm is submodular if labels are in the increasing order. If we perform more than one cycle, then the expansions are not necessarily submodular.

Let $\boldsymbol{x}^0$ be a labeling where each pixel is assigned 0. Consider the example in Figure 6.6. Suppose we apply one cycle of expansions, in the increasing order, starting with $\boldsymbol{x}^0$. Let us refer to the car objects as car 1, 2, 3, in the order of their depth, with car 3 being the front-most. After the first expansion, i.e. expansion on label 1, we will get the same result as in Figure 6.6(c), with all cars labeled with 1. This is because if we start with $\boldsymbol{x}^0$, both the expansion move on label 1 and the jump move are submodular and can be optimized exactly, and, furthermore, the optimal jump and expansion moves coincide. The next expansion, i.e. expansion on label 2, will produce the labeling as in Figure 6.6(d) the same as the second jump move, where car 3 gets label 2, and cars 1 and 2 stay with label 1. Switching both cars 2 and 3 to label 2 would result in a worse energy (jump move would switch cars 2 and 3 to label 2 if that was lower energy than the energy in Figure 6.6(d)). Any further expansion moves will not change the labeling, unlike further jump moves. To get to the lower energy labeling in Figure 6.6(e)), we need to simultaneously switch car 2 to label 2 and switch car 3 to label 3. Jump move is able to do this, but the expansion algorithm cannot switch pixels to two different labels.

Chapter 7

Experiments

7.1 Datasets

We use KINS [83] and COCOA [133] datasets. KINS consists of 7,474 training and 7,517 test images, and 7 instance classes. It provides modal and amodal semantic instance masks with their occlusion order. We do not use amodal masks for any purpose. COCOA [133] is a dataset consisting of 5,073 images of natural scenes. It provides modal and amodal masks in addition to occlusion ordering. We do not use the amodal masks for any purpose. We train on the 'train' set and evaluate on the 'val' set for all models.

7.2 Implementation Details

For KINS: For E2EC, we use their official implementation and configurations, and we only train it on modal masks. Hence, the model is trained for 150 epochs. For Mask R-CNN, we use the implementation of [105]. For InstaOrder and OrderNet, we use the pre-trained models of [62]. For our bottom-up models, we use the PSPNet [132] from the implementation of [131]. We use weight=0.9 wor wCE in our deep model for semantic segmentation and occlusion boundary and train for 200 epochs using SGD and weight decay as in the implementation, we used a single GPU and a batch size of 8. We train using crop size of 225x225 and test on a resolution of 2048x615. The testing resolution is similar for all models. The backbone of our PSPNet is Resnet-50, similar to the backbone of Mask R-CNN that we used. P2ORM [84] is also used based on their official implementation and the settings were used as in their code or paper. For a fair comparison, we used the

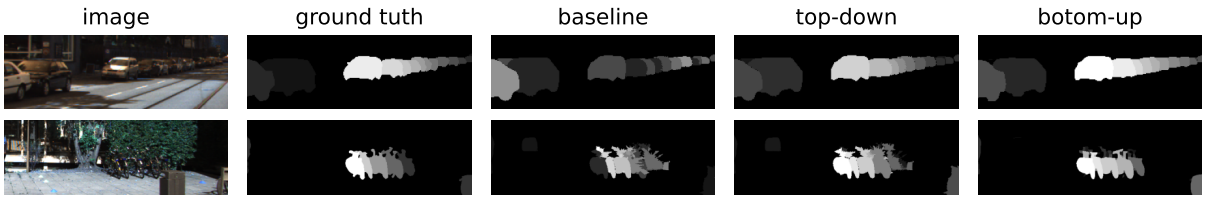


Figure 7.1: Relative depth map visualization of OOSIS for baseline and our approaches on KINS dataset. The best version of the baseline and our top-down approach is displayed.

4-neighborhood variant of theirs as we use a 4-neighborhood system in our deep model. For MiDaS [85], we used the official implementation with "midas_v21_384" model type.

For COCOA: For E2EC, we train for 50 epochs using Adam with parameters of their implementation for COCO and finetune for 10 epochs using SGD for maximum performance. For Mask R-CNN we use the implementation of [18] and use their configuration for COCO with the 1x training regiment. Again, for InstaOrder and OrderNet, we use the pre-trained models from [62]. Our setting for our deep model for semantic segmentation and occlusion boundary is the same as for KINS. The test resolution for all models is the same as the original image size. The backbones are the same as for KINS. MiDaS is the same as what we explained for KINS.

7.3 Baselines

We compare our methods against a baseline that utilizes existing techniques. Specifically, it involves using an instance segmentation method to generate instance predictions and then applying a pairwise occlusion ordering CNN directly on neighboring instances to obtain the occlusion order without any further processing or mask overlap resolution, as is the case with our top-down approach. The poor performance of these baselines underlines the role of the mask overlap resolution component we propose in the framework of the top-down approach in Chapter 5.

7.4 Confidence Score in our Bottom-up Approach

Bottom-up approaches lack the instance confidence score provided by detection-based instance segmentation methods like M-RCNN that use their deep network to predict it.

Some evaluation metrics require an instance confidence score for each instance prediction, as it will be discussed in Section 7.6. To address this, we follow the common practice of assigning the score based on simple heuristics [7]. Specifically, we assign a score to each instance by summing the predicted occlusion boundary probabilities on the border and inside of the instance, and subtracting the second from the first. This score rewards instances whose borders match the predicted boundaries and have no high-probability boundaries inside, indicating that they should not be further divided. It also gives higher scores to larger instances, and these tend to be more reliable. Using this score, we can evaluate our bottom-up approach, when a metric requires it.

7.5 Qualitative Results

Figure 7.1 and Figure 7.2 show visualizations of our approaches vs. the baselines and the ground truth for KINS. Figure 7.4 represents the output of our deep model for semantic segmentation and oriented occlusion boundary for KINS. Also, Figure 7.3 show visualizations of our approaches vs. the baselines and the ground truth for COCOA, and Figure 7.5 represents the output of our deep model for semantic segmentation and oriented occlusion boundary for COCOA.

7.6 Standard Mask Evaluation Metrics

Here, we explain and discuss the standard metrics used for assessing the quality of predicted instance masks by a method. These metrics are later used for the evaluation of our approaches.

7.6.1 Intersection over Union (IoU)

IoU is commonly used as an evaluation metric to measure the accuracy and quality of object localization or segmentation algorithms. In instance segmentation, IoU is used to assess the similarity between two masks, e.g. a predicted and a ground-truth mask. Given two masks a and b, the intersection-over-union (IoU) for them is defined as:

$$IoU(a, b) = \frac{a \cap b}{a \cup b}. \tag{7.1}$$

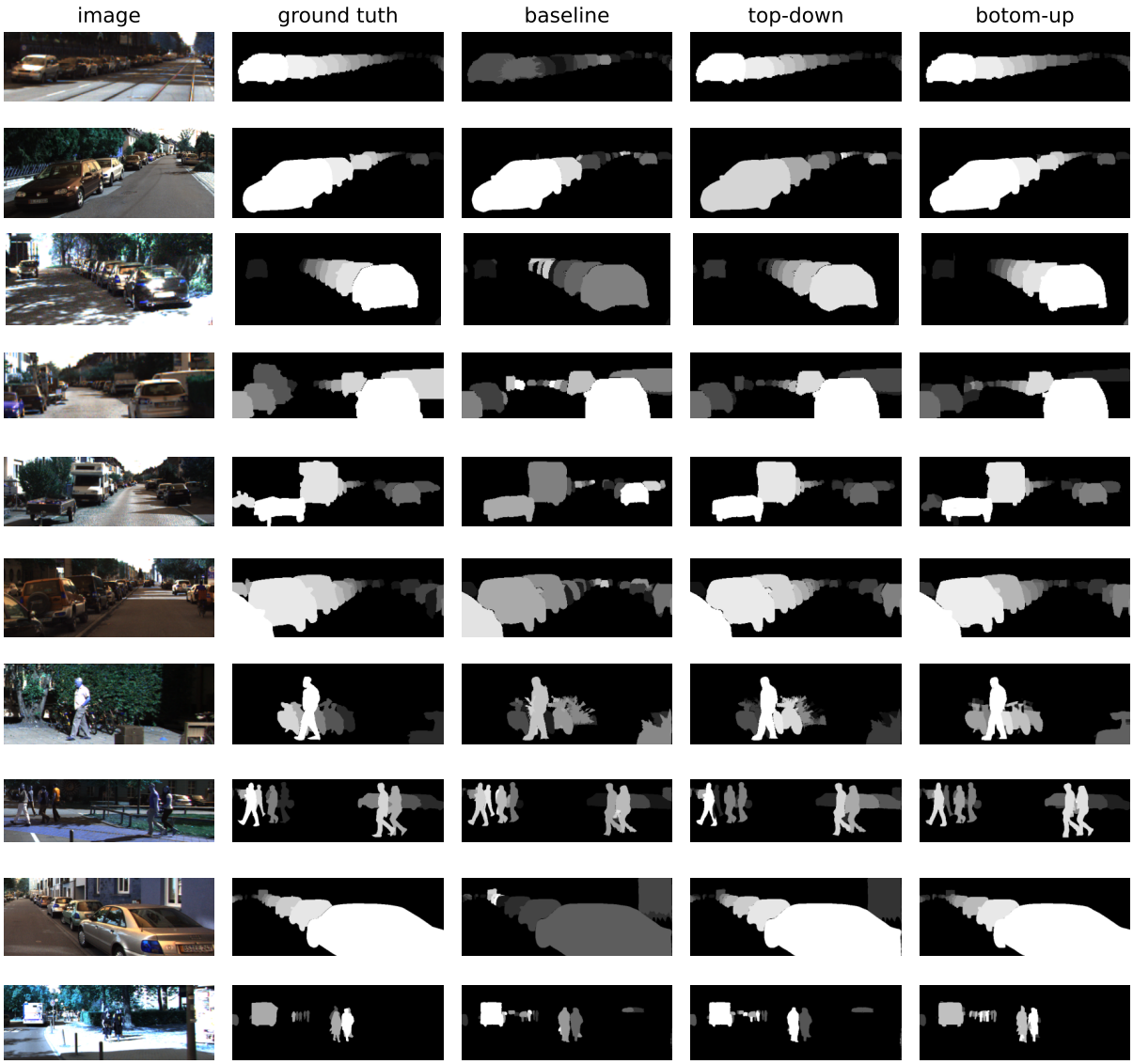


Figure 7.2: Relative depth map visualization of OOSIS for baseline and our approaches on KINS dataset. The best version of the baseline and our top-down approach is displayed.

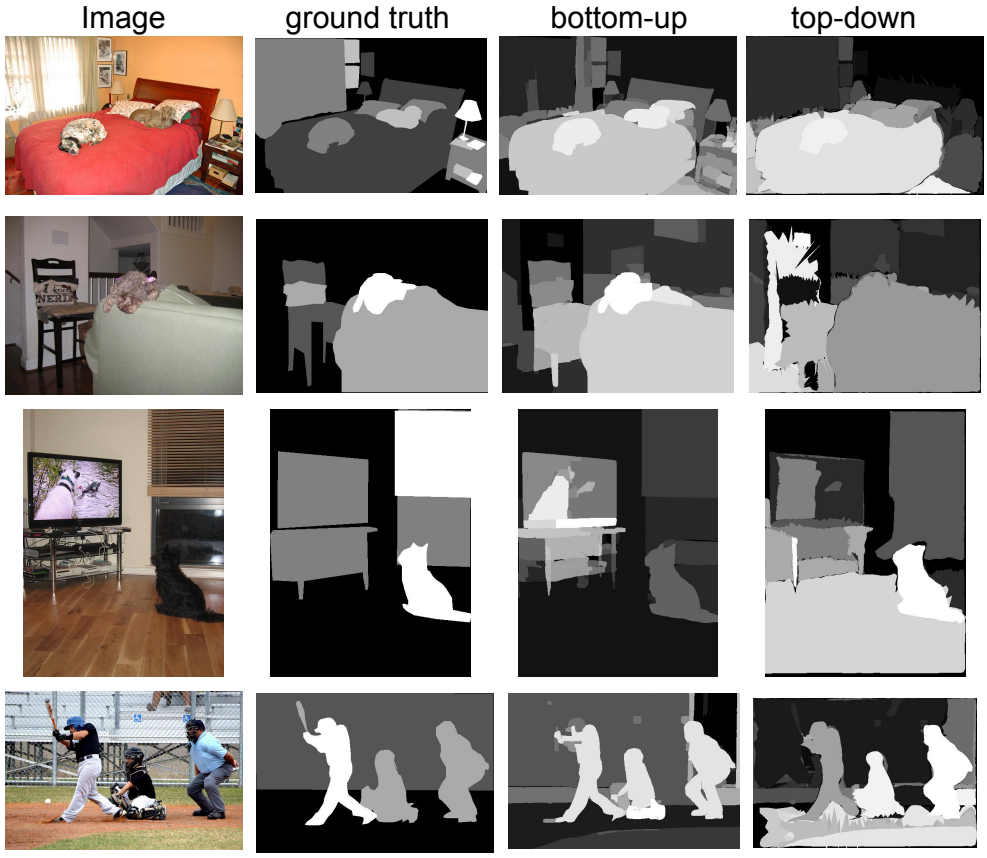


Figure 7.3: Relative depth map visualization of OOSIS for baseline and our approaches on COCOA dataset. The best version of the baseline and our top-down approach is displayed. COCOA is a harder dataset than KINS for OOSIS as common objects such as chairs, tables, etc are more complicated than objects in driving scenes.

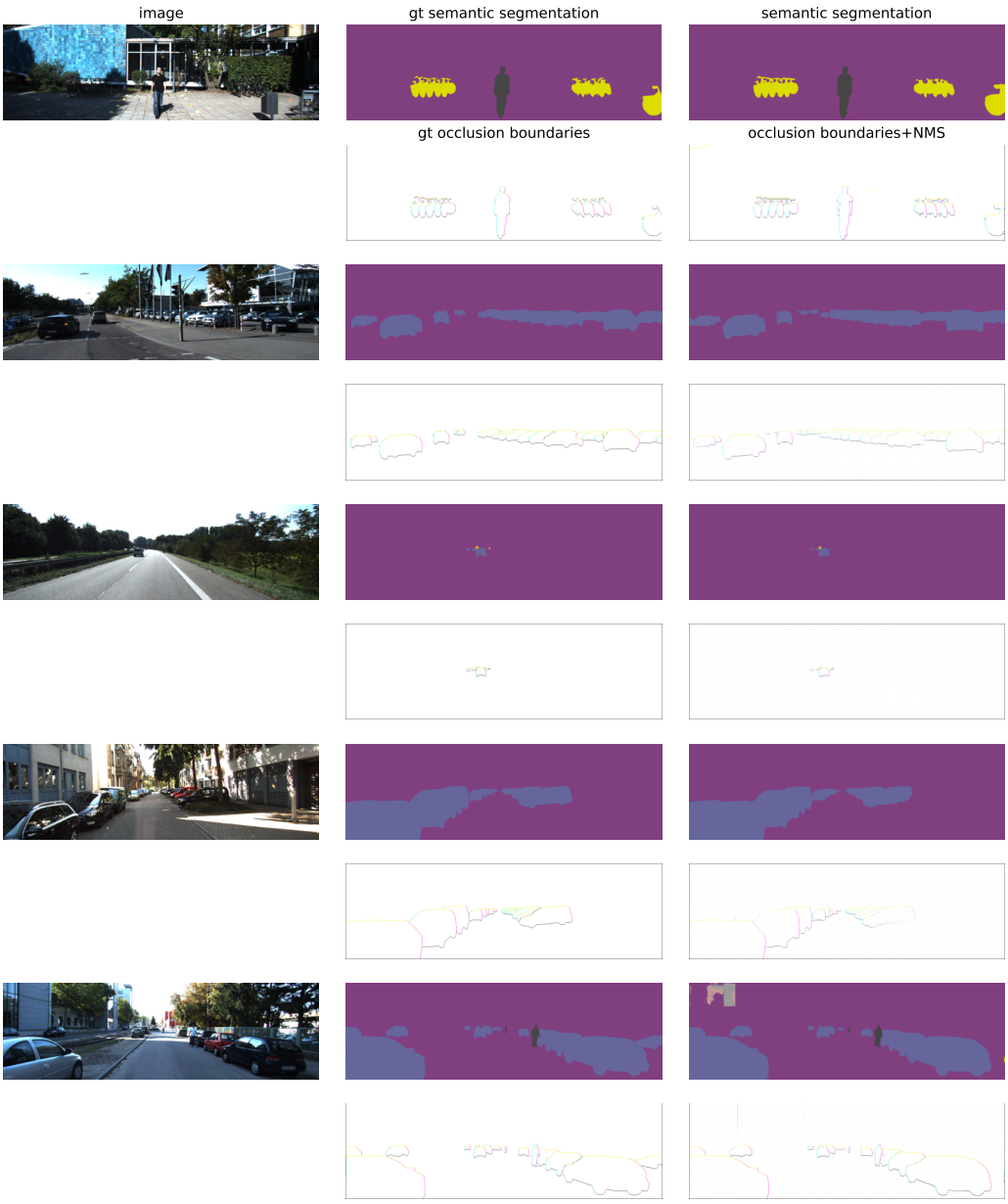


Figure 7.4: Illustration of our semantic segmentation and oriented occlusion boundaries for the bottom-up OOSIS. The images are: input, ground truth and our semantic segmentation, ground truth and our oriented occlusion boundaries after non-maximum suppression. The color scheme for the boundaries is: left-cyan, top-yellow, right-magenta, bottom-black. Best viewed zoomed in.

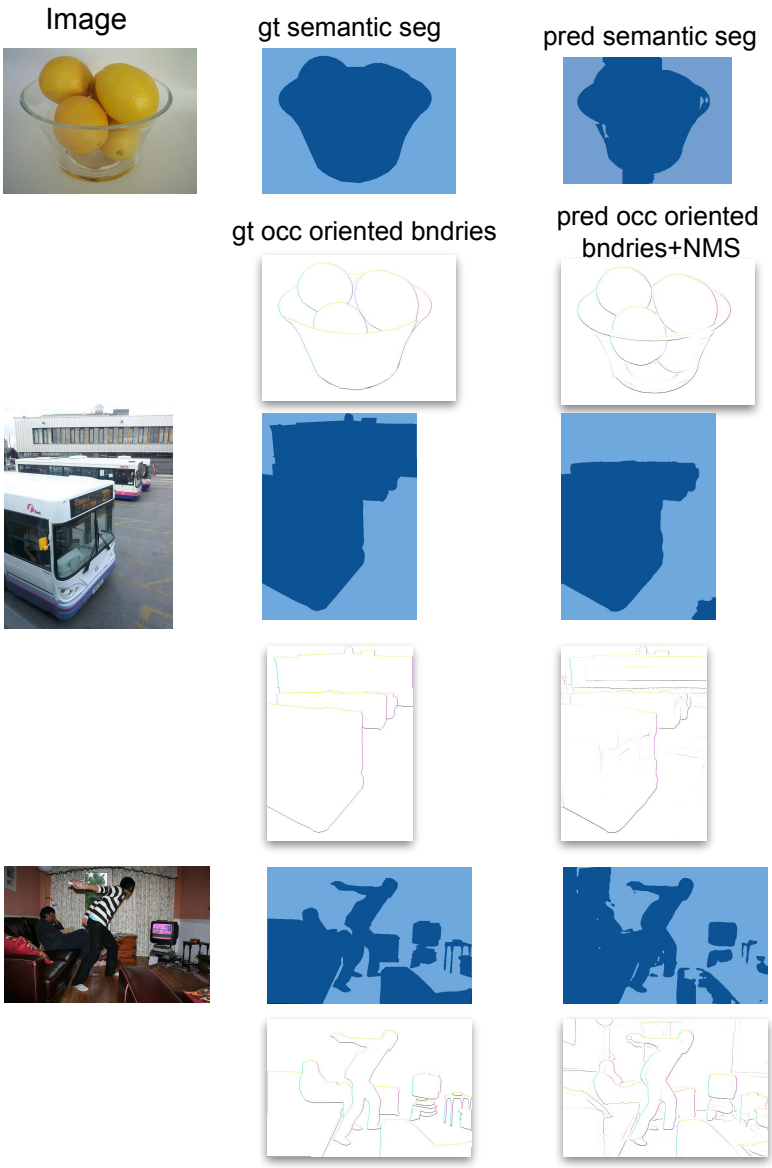


Figure 7.5: Illustration of our semantic segmentation and oriented occlusion boundaries for the bottom-up OOSIS. The images are: input, ground truth and our semantic segmentation, ground truth and our oriented occlusion boundaries after non-maximum suppression. The color scheme for the boundaries is: left-cyan, top-yellow, right-magenta, bottom-black. Best viewed zoomed in. Note that COCOA does not have several specific semantic classes but only a background/object class is determined.

This formula represents the ratio of the intersection of the masks to their union, where intersection refers to the overlapping area between the masks, and union represents the total area covered by both masks. IoU ranges from 0 to 1, where a score of 1 indicates a perfect overlap between the masks, while a score of 0 means no overlap at all.

7.6.2 mean Average Precision (mAP)

Among the metrics for instance segmentation, mean AP (mAP) [63, 83] is the most commonly used metric. The mAP is a score inspired by the detection problem. The mAP is calculated by finding Average Precision (AP) [32] for each class and then averaging over them.

For calculating AP for different classes, we first need to match the predicted instances and the ground-truth instances. To initiate the matching process, we start by identifying the predicted instance with the highest confidence score. This means that the instance segmentation algorithm must produce a confidence score for each predicted instance, to enable mAP computation. Then, we aim to find a corresponding ground-truth instance that shares the same semantic class and has the highest IoU value. This IoU serves as a measure of overlap between the predicted and ground-truth instances.

If the computed IoU surpasses a predefined minimum threshold, which we call t, we consider the instances as a match and move on to the next predicted instance. By adjusting t, we have the ability to control the number of matched instances.

To compute the Average Precision (AP) for each semantic class c in an instance segmentation task, several steps need to be followed. Firstly, Precision and Recall must be defined. Precision is the ratio of predicted instance masks with class label c that successfully match a ground truth mask of the same class. Recall, on the other hand, represents the ratio of ground truth masks with class c that are successfully matched by a prediction.

To calculate AP, the confidence score threshold for predicted instances can be adjusted. This allows us to exclude predicted masks below a certain confidence score from the precision and recall calculations. By varying this threshold from 0 to 1 and computing precision and recall at each step, we obtain a series of precision-recall pairs. These pairs can be used to construct a Precision-Recall curve. The area under the Precision-Recall curve corresponds to the average precision for class c. AP is calculated for each class, and the mean average precision (mAP) is obtained by averaging the AP values across all classes.

Now, remember that mAP also depends on t, the threshold of IoU in the matching process. Typically, different t values are used, and the mAP is calculated for each. Then,

the mean of such mAPs is reported. A common practice is to use ten t values from 0.5 to 0.95 with steps of 0.05. The mAP using this practice is called $mAP_{0.5:0.95}$.

There are multiple works that analyze the shortcomings of AP [26, 78, 130, 7, 49]. One problem with AP is that it does not penalize overlapping instances or a large number of instance predictions provided that the instances are ranked in the correct order [49]. Thus the unambiguous approaches, i.e. approaches that predict a single instance label per pixel, are at a disadvantage.

Our top-down and bottom-up approaches are unambiguous. However, even in such cases, the AP metric has a serious drawback [7]. AP metric depends on the confidence score assigned to instances. If we take the same instance segmentation and assign different confidence scores to the instances, AP scores will be different, even though the segmentation is exactly the same. Our top-down approach is based on instance detection methods and, therefore, has reasonable instance confidence scores estimated by CNN. However, for the bottom-up approach, we have to come up with ad-hoc confidence scores to employ the AP metric as we explained in Section 7.4. This is a non-appealing property of the AP metric, especially when applied to non-ambiguous instance segmentation, where the confidence scores are not required, as all the instances appear in the final instance segmentation result.

7.6.3 Weighted Coverage Score (WCS)

In [49] they propose metrics that are more appropriate for the approaches producing overlapping instances, i.e. ambiguous segmentation where a pixel can belong to more than one instance. However, since both our top-down and bottom-up approaches do not produce overlapping instances, we advocate the use of the Weighted Coverage Score metric [101, 95], which can be applied only to the unambiguous approaches, i.e. approaches which produce one instance (or background) per pixel.

Let $G = \{r_1, ..., r_k\}$ be the set of ground truth instances, and $S = \{s_1, ..., s_l\}$ be the set of segmented instances in an image. Given a pair of regions a, b, the intersection over union score is defined as

$$IoU(a, b) = \frac{a \cap b}{a \cup b} \tag{7.2}$$

The Weighted Coverage Score (WCS) metric is defined as

$$WCS(G, S) = \frac{1}{n} \sum_{i=1}^{k} |r_i| \max_{j=1,...,l} IoU(r_i, s_j), \tag{7.3}$$

where n is the number of pixels in the image and $|r_i|$ is the size of region r_i. Intuitively, WCS matches each ground truth instance with a segmented instance of the largest overlap and adds to the score the size of the ground truth region, weighted by the goodness of this overlap (where goodness is equal to the IoU score). The best value of WCS is 1, obtained when the set of segmented instances contains exactly the same segments as the ground truth. WCS metric does not depend on the confidence of a segmented instance, which is an intuitive property for the evaluation of unambiguous instance segmentation methods.

Note that using WCS metric makes sense only for unambiguous instance segmentation methods, i.e. each pixel is assigned to at most one instance. Otherwise, one can take S to be equal to the set of all possible segments, automatically achieving the best WCS score of 1.

7.7 Occlusion Boundary Model Ground Truth

Having the instance masks, one can easily get the boundary pixels. Hence, B_p is easy to produce. Regarding E_p, which is only defined for ground-truth boundaries, we determine the orientation distribution for each boundary pixel p. To achieve this, we examine whether pixel p occludes any of its left, right, top, or bottom neighbor pixels. We then set the corresponding components in the orientation vector to 1, indicating the presence of occlusion in those orientations. Finally, to ensure that the vector represents a valid probability distribution, we normalize it to have a sum of 1. For example, if pixel p only occludes its left neighbor, the corresponding E_p vector is $[1, 0, 0, 0]$. On the other hand, if pixel p occludes both its left and top neighbors, the E_p vector is $[0.5, 0, 0.5, 0]$, indicating that the normal's orientation lies between the two directions (i.e., 45 degrees to the top left).

7.8 Evaluation of Semantic Instance Masks

We evaluate the instance masks of our approaches using standard instance segmentation metrics. Occlusion ordering is not considered for this evaluation. We call an instance segmentation method unambiguous if each pixel is assigned to a single instance. Both of our approaches are unambiguous, whereas most instance segmentation methods are ambiguous, producing multiple overlapping masks [49]. For the top-down method, we use two well-established instance segmentation models, E2EC and M-RCNN, and different overlap resolution techniques, as explained in Chapter 5. We report metrics: $mAP_{0.5:0.95}$ [63, 83]

Table 7.1: Performance of our different approaches on semantic instance segmentation on the KINS dataset.

Model	Instance Detection	Overlap Resolution	Unambiguous Instance Segmentation	
			$mAP_{0.5:0.95}$ ↑	Weighted Coverage ↑
Top-down	M-RCNN	Random	11.1	66.7
		Low Confidence	7.3	61.2
		High Confidence	23.3	76.2
Top-down	E2EC	Random	14.2	68.1
		Low Confidence	10.3	63.2
		High Confidence	**24.1**	76.8
Bottom-up	Optimization	By Construction	21.6	**77.4**

and Weighted Coverage Score [95]. The former is widely-used for instance segmentation [63] [83]. The latter is more equitable when comparing bottom-up approaches vs. top-down ones, as it removes the need for predicting a mask confidence score [7]. However, it is only applicable when the predicted masks are unambiguous [7], as explained in Section 7.6.

Results on KINS: Tab.7.1 shows the results. Resolving the overlap by High Confidence yields superior results vs. other policies. E2EC is better than M-RCNN in all cases. Our bottom-up approach achieves the best Weighted Coverage Score (WCS), despite its lower mAP. This is due to the confidence score assignment influence, see [7, 49] and Section 7.6. While an improved scoring may increase mAP for bottom-up approaches, WCS is a more equitable comparison as explained in the previous paragraph. The results highlight the effectiveness of the High Confidence overlap resolution, and the strength of our bottom-up approach when compared with the recent powerful instance segmentation E2EC.

Results on COCOA

Table 7.2 presents the results obtained from evaluating our proposed approaches for instance segmentation. As in Table 7.1, for KINS, our top-down approach is assessed using two distinct instance segmentation models, namely Mask R-CNN [40] and E2EC [128]. We also examine the performance of each model when incorporating the three overlap resolution techniques proposed in Chapter 5. Additionally, the performance of our bottom-up approach is reported.

As outlined, all of our approaches produce unambiguous instance segmentation, i.e.

Table 7.2: Performance of our different approaches on semantic instance segmentation on the COCOA dataset.

Model	Instance Detection	Overlap Resolution	Unambiguous Instance Segmentation	
			$mAP_{0.5:0.95}$ ↑	Weighted Coverage ↑
Top-down	M-RCNN	Random	0.1	41.4
		Low Confidence	2.2	38.3
		High Confidence	15.2	56.5
Top-down	E2EC	Random	6.7	49.2
		Low Confidence	4.1	46.5
		High Confidence	**16.7**	57.7
Bottom-up	Optimization	By Construction	9.1	**61.2**

each pixel is associated with only one instance label (or the background). Notably, the results in Table 7.2 indicate superior performance of the top-down models employing E2EC compared to those utilizing M-RCNN. Moreover, overlap resolution by higher confidence consistently outperforms other techniques for handling overlaps. While the bottom-up approach demonstrates the highest Weighted Coverage metric, it exhibits a lower mAP. As explained previously and in Section 7.6, and [7], the Weighted Coverage score is more equitable when comparing bottom-up approaches with top-down ones, due to the reliance of mAP on the instance confidence scores and the lack of meaningful instance confidence scores in the bottom-up approach.

7.9 Evaluation of Occlusion Ordering Consistency

Consistent occlusion ordering is important for scene analysis. To assess the consistency of occlusion ordering, we measure the percentage of detected instances involved in cyclic occlusion ordering for each method.

Results on KINS: In Tab.7.3, we evaluate the baselines and our approaches on KINS dataset. The ground truth is cycle-free. As can be seen, the baselines perform poorly, having a large number of cycles. Our top-down approach with overlap resolution by higher confidence, significantly reduces cycles. The bottom-up approach is inherently cycle-free, providing further evidence of its efficacy.

Results on COCOA: Table 7.4 presents the results on COCOA dataset. It is worth

Table 7.3: % of predicted instances involved in cyclic occlusion ordering. (H) is mask overlap removal by higher confidence. The results are on the KINS dataset.

Model	% of Instances in Cycles ↓
Baseline: M-RCNN_InstaOrder	55.4
Baseline: M-RCNN_OrderNet	55.7
Baseline:E2EC_InstaOrder	46.3
Baseline:E2EC_OrderNet	44.5
Top-down:M-RCNN_(H)_InstaOrder	7.6
Top-down:M-RCNN_(H)_OrderNet	5.7
Top-down:E2EC_(H)_InstaOrder	9.1
Top-down:E2EC_(H)_OrderNet	6.4
Bottom-up	**- 0 -**

Table 7.4: % of predicted instances involved in cyclic occlusion ordering. (H) is mask overlap removal by higher confidence. The results are on the COCOA dataset.

Model	% of Instances in Cycles ↓
Baseline: M-RCNN_InstaOrder	72.4
Baseline: M-RCNN_OrderNet	82.9
Baseline:E2EC_InstaOrder	68.3
Baseline:E2EC_OrderNet	78.9
Top-down:M-RCNN_(H)_InstaOrder	27.5
Top-down:M-RCNN_(H)_OrderNet	46.7
Top-down:E2EC_(H)_InstaOrder	32.2
Top-down:E2EC_(H)_OrderNet	55.6
Bottom-up	**- 0 -**

noting that the ground truth for COCOA dataset is cycle-free, as it is for the KINS dataset. As shown in Table 7.4, the baselines, regardless of their instance segmentation model, exhibit a remarkably high cycle percentage, which indicates their inconsistent outputs. Our proposed top-down approaches mitigate the issue, reducing the occurrence of cyclic

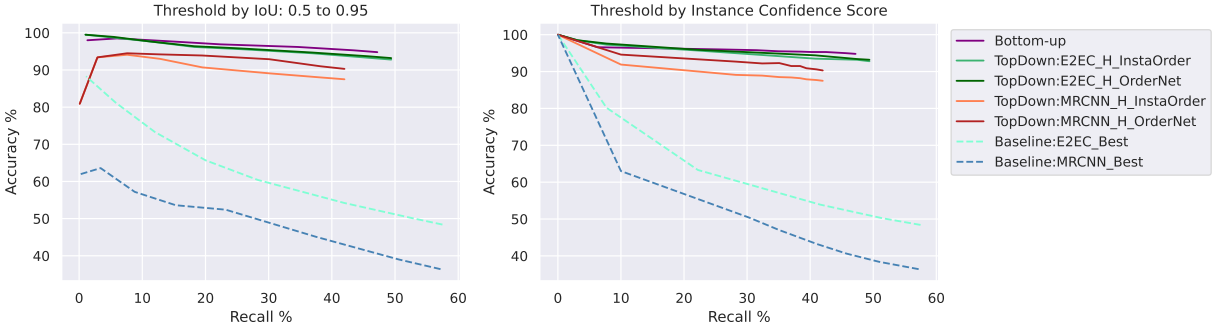
occlusion ordering by anywhere from 25% to 45%. Nonetheless, they still exhibit a notable number of instances with cyclic occlusion ordering. This is attributed to the inconsistent pairwise occlusion orderings generated by InstaOrder and OrderNet. COCOA dataset is considerably more difficult for pairwise occlusion order estimation (compare to KINS dataset, Table 7.3). In contrast, our bottom-up approach achieves a cycle-free occlusion ordering, consistent with the ground truth.

7.10 Evaluation of OOSIS using Our Metric (AOR Curves)

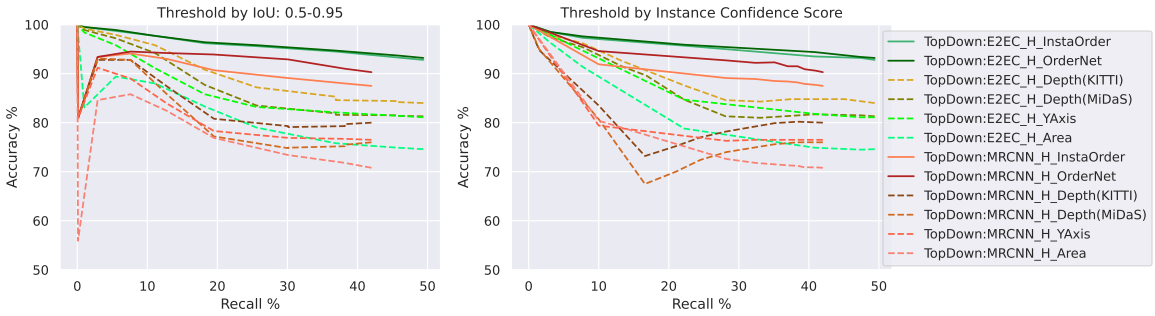
In an occlusion order graph, instances are nodes and directed edges connect occluders to their occludees. We construct occlusion graphs $\mathcal{G}$, based on ground truth, and $\hat{\mathcal{G}}$, based on the predicted instances and occlusion relations. To establish the correspondence between predicted instances and ground-truth instances, we follow a standard instance segmentation procedure [63] [83]. Specifically, we begin by selecting the predicted instance with the highest confidence and seek a matching ground-truth instance with the same semantic class and the highest Intersection-over-Union (IoU). If the IoU exceeds a minimum threshold, the instances are matched, and we proceed to the next predicted instance. By raising the IoU threshold, we can reduce the number of matched instances. Additionally, we can establish a minimum confidence score below which predicted instances are not considered for matching. Once we have a matching, we consider two nodes g_1 and g_2 in $\mathcal{G}$, where g_1 occludes g_2, i.e. endpoints of edges. If both g_1 and g_2 have a matched detection in $\hat{\mathcal{G}}$, we label the pair as "recovered," otherwise as "missed". We define a pair as "correctly ordered" if there is a directed path from the matched detected instance of g_1 to that of g_2 in $\hat{\mathcal{G}}$ and no path vice versa, i.e. one single order in the correct direction exists between the two.

We compute recall and accuracy, where recall is the ratio of recovered pairs to the total number of recovered and missed pairs, and accuracy is the ratio of correctly ordered pairs to the total number of recovered pairs. We plot the Accuracy vs. Recall (AOR) curve by varying thresholds in node matching, either based on IoU or confidence score. Our AOR is similar to what [113] use for occlusion boundaries. The curve evaluates mask and occlusion order simultaneously.

Results on KINS Fig.7.6a gives AOR curves of our methods and the baselines. Our bottom-up approach achieves the highest accuracy for similar recalls, outperforming the best top-down method by a margin of $\sim 2\%$ in accuracy for similar recalls above 30%. Our

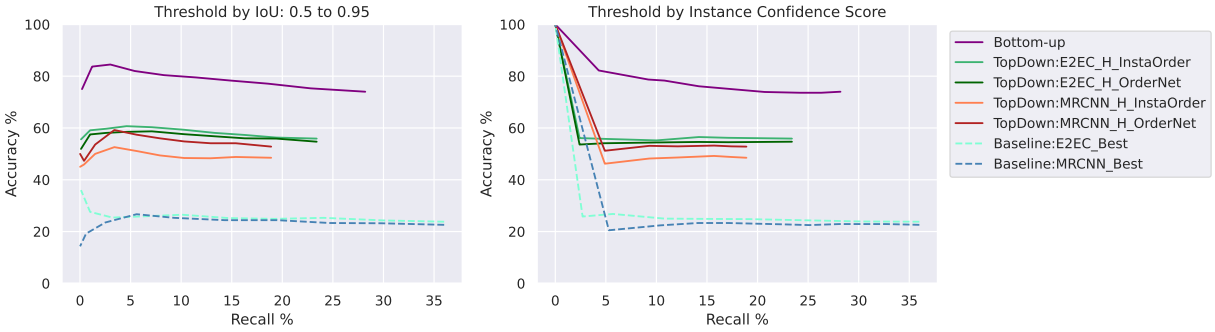


(a) Our approaches compared to baselines.

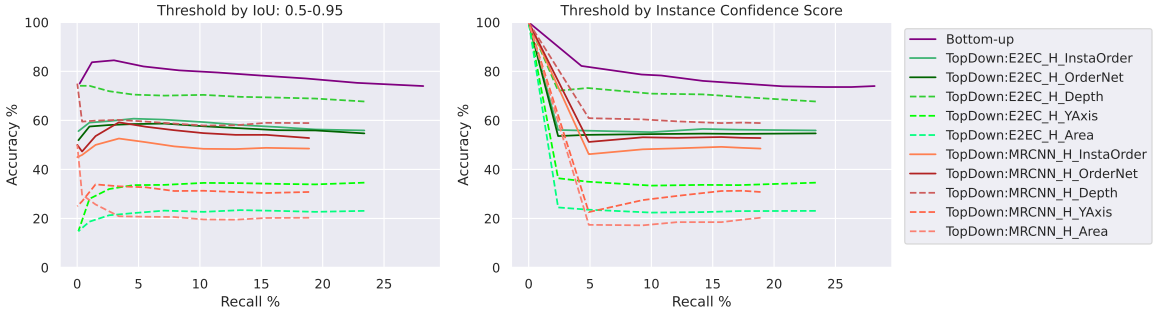


(b) Classifiers vs. simpler ideas for pairwise ordering in our top-down approach.

Figure 7.6: Left: AOR curves for IoU thresholds ranging from 0.5 to 0.95 in steps of 0.05 with confidence score threshold fixed at 0. Right: AOR curves for varying minimum confidence score, with six thresholds covering 0 to the maximum recall of each method with the minimum IoU threshold fixed to 0.5. (H) is overlap removal by higher confidence. A higher curve is better performance.



(a) Our approaches compared to baselines.



(b) Classifiers vs. simpler ideas for pairwise ordering in our top-down approach.

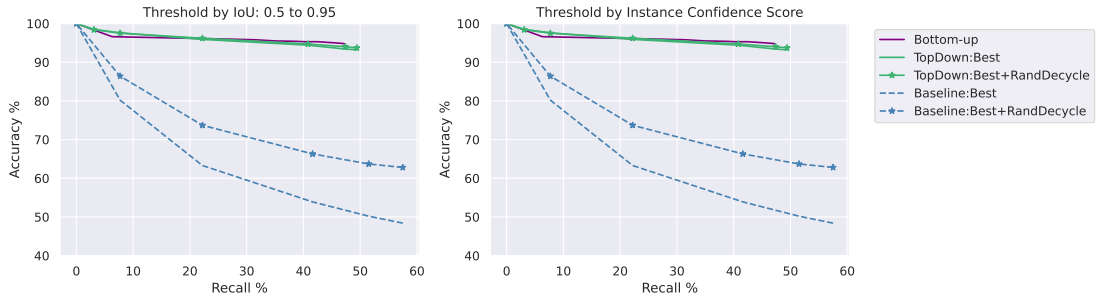
Figure 7.7: Left: AOR curves for IoU thresholds ranging from 0.5 to 0.95 in steps of 0.05 with confidence score threshold fixed at 0. Right: AOR curves for varying minimum confidence score, with six thresholds covering 0 to the maximum recall of each method with the minimum IoU threshold fixed to 0.5. (H) is overlap removal by higher confidence. A higher curve is better performance.

top-down approach with E2EC for instance segmentation performs as the next best alternative and has a slight $\sim 2\%$ advantage in recall over the bottom-up approach. Notably, while InstaOrder outperforms OrderNet in the pairwise occlusion ordering task [62], it performs slightly weaker than OrderNet when used in the top-down approach for OOSIS. Besides, our top-down method with M-RCNN performs worse than both our top-down approach with E2EC and our bottom-up one. This shows the importance of the chosen instance segmentation model on the top-down approach. Nevertheless, it still outperforms the baselines by a huge margin, stressing the inadequacy of simply combining prior works for OOSIS.

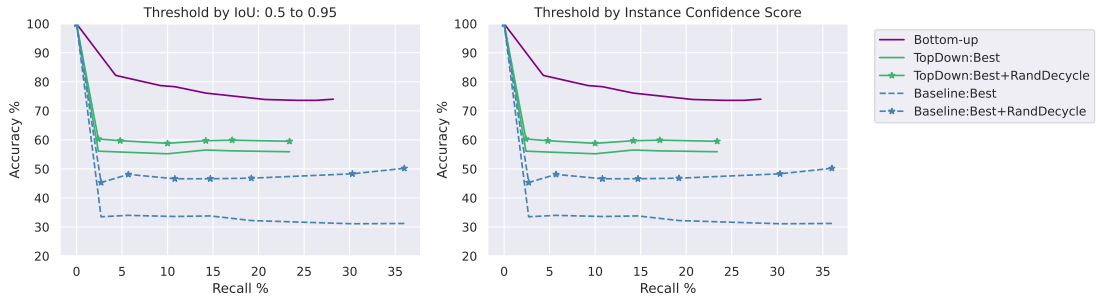
Fig.7.6b gives the AOR curves for our top-down approach with different pairwise occlusion ordering techniques including the discussed CNNs, InstaOrder and OrderNet, and other simpler ideas. We consider *depth*: the instance with a smaller mean depth estimate occludes the other, *area*: the bigger mask occludes the smaller, and *y-coordinate*: the mask with a lower center occludes the other. The depth is by monocular estimator MiDaS [85] or by HR-Depth [66], where the former is trained on a mixture of datasets and the latter is trained on KITTI [36], which is the mother dataset of KINS. The two CNN classifiers perform significantly better than the simple ideas, for both E2EC and M-RCNN. It is notable that *depth* works significantly weaker than using classifiers, which shows the inefficacy of monocular depth estimation for this task.

Results on COCOA: In this section, we conduct an evaluation of the baselines and our proposed approaches on the COCOA dataset, utilizing AOR curves. Figure 7.7a visualizes the efficacy of our bottom-up approach. Firstly, it achieves the highest accuracy across all recall values, surpassing both the baselines and top-down models. The performance gap amounts to approximately 50% compared to the baselines and around 20% compared to the best-performing top-down model. Secondly, in contrast to the KINS dataset, our bottom-up approach achieves higher recalls than the best top-down model, exhibiting a margin of approximately 6%. Among the top-down models, the one utilizing E2EC and InstaOrder demonstrates superior performance on COCOA. Importantly, all of our proposed approaches and models exhibit significantly improved performance compared to the baseline methods.

Fig. 7.7b also shows the AOR curves for comparing different pairwise occlusion ordering methods for our top-down approach. We employ the same simpler strategies we explained for the KINS dataset. Note that *depth* works poorly on the KINS dataset. This is because KINS images are driving scenes where objects can be very far making it significantly harder for depth estimation and increasing the chance that far objects will get similar/wrong depth estimates. In COCOA, images are mostly indoor scenes with objects being very closer to the camera helping the depth estimation to be more reliable. However, it is still weaker



(a) Comparison of Random Decycling on KINS Dataset.



(b) Comparison of Random Decycling on COCOA Dataset.

Figure 7.8: Left: AOR curves for IoU thresholds ranging from 0.5 to 0.95 in steps of 0.05. Right: AOR curves for varying minimum confidence score, with six thresholds covering 0 to the maximum recall of each method. (H) is overlap removal by higher confidence. A higher curve is better performance.

than the bottom-up approach for establishing the occlusion order.

7.11 Cycle Removal Experiment

The presence of cyclic occlusion orderings is highly undesirable, as discussed in Chapter 1, and Chapter 5. While our bottom-up approach guarantees a cycle-free occlusion ordering, the top-down approach does exhibit cyclic orderings, albeit to a lesser extent compared to the baselines. As mentioned in Chapter 5, it is hard to come up with an efficient intelligent strategy for cycle removal. For example, one may want to remove the smallest set of edges which results in an acyclic ordering graph. However, this is an NP-hard 'feedback arc set' problem. In this section, we investigate decycling by applying a post-hoc random cycle removal on the occlusion ordering graph of each method. The procedure is as follows. We

Table 7.5: The performance of our deep model for oriented occlusion boundary detection vs. the state-of-the-art, P2ORM. OIS, ODS, are F-measures based on the best threshold per image, or for the whole dataset, respectively. AP is average precision. All are based on POR curves for NMS-applied oriented occlusion boundaries on KINS.

Model	Oriented Occ Boundary		
	ODS↑	OIS↑	AP↑
P2ORM	77.8	80.6	79.4
Ours	**84.5**	**86**	**88.3**

find a cycle, remove a random edge from it and check if there are still any cycles left. We repeat this procedure until we get an acyclic graph.

Figures 7.8a and 7.8b present the performance results of this technique on the KINS and COCOA datasets, respectively. As observed, the bottom-up approach remains the top performer in terms of accuracy at similar recall values. Both the top-down approach and the baselines benefit from the random decycling technique, although the baselines still exhibit a significantly poorer performance compared to the top-down approach.

These findings underscore the importance of addressing cyclic occlusion orderings. It also demonstrates that random decycling improves the performance of both top-down and baseline methods, although they still fall short of the performance achieved by the bottom-up approach.

7.12 Evaluation of Oriented Occlusion Boundaries

In Chapter 6.1, we describe our oriented occlusion boundaries approach, which we then use in Chapter 6.2 for optimization. While optimization is crucial for high-quality globally consistent OOSIS, it heavily relies on the quality of the occlusion boundaries. We evaluate our performance against the state-of-the-art occlusion boundary method, P2ORM [84]. We use the standard oriented occlusion boundary metrics based on Precision vs. Recall curves (POR) [111]. Tab.7.5 shows our superior performance in all metrics by a margin, underscoring the effectiveness of our novel boundary model.

It is worth noting that P2ORM and previous works were originally developed for extracting all oriented occlusion edges in an image, while in our case, we only focus on the edges relevant to objects with classes of interest. As shown in Table 7.5, P2ORM performs weaker when trained and tested on this specific task.

Prior works [113, 111, 84] compare their methods using the angle of boundary using POR curves. Hence, to compare, we also convert our quantized predictions to an angle. To convert, we proceed as follows, keeping in mind that our orientation vector estimates are noisy. An orientation cannot be simultaneously to the left and to the right. Therefore, we take the maximum of the left and right components from "o". Similarly, orientation cannot be simultaneously to the top and bottom, therefore, we take the maximum of the top and bottom components from "o". We set the normal's angle by finding the arc tangent of the achieved vector of the two mentioned maximums and their directions. For example, if $o_p = [0.5, 0.1, 0.4, 0.0]$ the angle is $\alpha = arctan\frac{0.5}{0.4}$. Furthermore, we convert the angle of the normal to the angle of the boundary using the left rule [113], which involves adding or subtracting $\frac{\pi}{2}$.

7.13 Ablation Experiment on Oriented Occlusion Boundary Model

Here, we show the effect of the joint upsampling for heads "e" and "b". Tab.7.6, shows the joint upsampling improves the performance on AP metric of POR curves for oriented occlusion boundaries, and also on AP metric for just detecting boundaries, i.e. the "b" branch performance alone. This indicates the efficacy of sharing information in upsampling by the logits of "e" and "b".

Table 7.6: Ablation on the effect of joint upsampling of heads "b" and "e". All are for NMS-applied oriented occlusion boundaries on KINS.

Model	Oriented Occ Boundary			Boundary Detection
	ODS↑	OIS↑	AP↑	AP↑
Ours (Separate "b" and "e" Upsampling)	84.5	85.9	87.1	90.2
Ours (Joint "b" and "e" Upsampling)	84.5	**86**	**88.3**	**92.4**

As another ablation, we test different loss functions for the 'b' branch of the boundary model instead of the used Weighted Cross-Entropy. In particular, we evaluate Dice++ [123], Tversky++ [123], and also Combo [100] loss. For Dice++, we use $\gamma = 2$, and for Tversky++, we use $\gamma = 2, \alpha = 0.8, \beta = 0.2$. For Combo, we set $\alpha = 0.1$. Table 7.7 shows the performance using each of the losses in terms of Average Precision for occlusion boundary detection on the KINS dataset. The results show the effectiveness of Weighted CE. This

ablation is interesting because several prior works advocate complicated boundary losses such as the mentioned ones, while we show that simple cross-entropy, with a carefully chosen weight parameter, works better at least on the KINS dataset and for occlusion boundary detection.

Loss	Boundary Detection (AP)
Dice++	91.7
Tversky++	91.2
Combo	88.9
Weighted CE	**92.4**

Table 7.7: The performance of different loss functions for the 'b' branch of our boundary model in terms of boundary detection using AP metric on the KINS dataset.

7.14 Compute

All deep models, E2EC, Mask-RCNN, IntaOrder, OrderNet, and ours were trained and tested on a single GPU of NVIDIA RTX3090 using PyTorch. Optimization for our bottom-up was done on CPU using Python and Jupyter Notebook environment. Training our PSPNet-based deep model for joint semantic segmentation and occlusion boundary detection using the explained configurations specified in Section 7.2 took $\sim$ 12 hours for KINS and $\sim$ 4 hours for COCOA. For our bottom-up approach optimization, testing on a single image of KINS on CPU with no parallelization took $\sim$ 12 seconds. This was $\sim$ 13 seconds for COCOA. Training E2EC for KINS took $\sim$ 15 hours for KINS and $\sim$ 4 hours for COCOA. Testing each of InstaOrder or OrderNet on any of (instance segmentation, overlap removal technique) configuration pair took $\sim$ 1.5 hours. We did not have to train InstaOrder and OrderNet as we used pre-trained networks (on both KINS and COCOA datasets).

Chapter 8

Conclusion and Future Work

This thesis focused on occlusion-ordered semantic instance segmentation (OOSIS), which aims to address two key outputs: 1- segmentation masks and semantic classes for each object instance in an input image, and 2- the occlusion ordering among the objects. OOSIS goes beyond conventional semantic instance segmentation by incorporating a coarse 3D representation of the scene, achieved through the estimation of partial relative depth ordering.

Unlike prior works that relied on simple visual cues and pre-dated deep learning, we introduced two deep learning-based approaches to tackle OOSIS. The first approach follows a top-down strategy, where instance masks are initially extracted using standard instance segmentation methods. The next step involves resolving overlaps between different instance masks, aiming to accurately determine the occlusion relationship between neighboring objects, which is achieved by a CNN classifier designed to classify pairwise occlusion relations. While this approach benefits from the use of state-of-the-art instance segmentation methods, the pairwise nature of occlusion ordering leads to undesired cyclic orderings, thus lacking global ordering consistency.

The second approach takes a bottom-up approach. It involves grouping pixels into instances and assigning occlusion order labels using discrete energy optimization techniques. This approach aims to ensure global occlusion consistency, addressing the challenge of undesired cyclic orderings that may arise in the pairwise occlusion approach. Additionally, we have developed a novel occlusion-oriented boundary model, which surpasses the performance of previous methods in this domain.

In addition, we have proposed a novel evaluation metric, in Chapter 7 in the Section 7.10, specifically designed to assess the performance of OOSIS methods. This metric

takes into account both instance segmentation and occlusion ordering, enabling a comprehensive evaluation of OOSIS approaches. By simultaneously evaluating segmentation and ordering aspects, our evaluation metric provides a fair and effective means for future research in OOSIS to compare and analyze different methods. This metric serves as a valuable tool to benchmark and advance the field, fostering further improvements in both the segmentation and ordering capabilities of OOSIS methods.

An interesting direction for future research could involve the incorporation of Conditional Random Fields (CRF) formulation to effectively resolve the overlaps among predicted instance masks in the top-down approach. Such an approach would not only enhance the quality of the masks but also improve the accuracy of occlusion ordering. Besides, as we are utilizing deep learning for OOSIS for the first time, there are many other interesting directions for exploration. For instance, investigating the use of deep learning for a one-stage bottom-up approach instead of our current two-stage method would be an interesting direction for further exploration enabling a faster and more powerful OOSIS.

The main limitation of our work is that occlusions provide only a partial instance order. The relative depth order between two instances not connected by a monotone chain is unknown. Also, with neither of the two approaches a real-time OOSIS is possible. Future works can aim for such improvements as well.

References

[1] Ashutosh Agarwal and Chetan Arora. Depthformer: Multiscale vision transformer for monocular depth estimation with global local information fusion. In *2022 IEEE International Conference on Image Processing (ICIP)*, pages 3873–3877. IEEE, 2022.

[2] Ashutosh Agarwal and Chetan Arora. Attention attention everywhere: Monocular depth prediction with skip attention. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 5861–5870, 2023.

[3] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196, 1993.

[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] Varin Anand. Introduction to neural networks, Mar 2022.

[6] Arthur Arnx. First neural network for beginners explained (with code), Aug 2019.

[7] Min Bai and Raquel Urtasun. Deep watershed transform for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5221–5229, 2017.

[8] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact: Real-time instance segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9157–9166, 2019.

[9] Endre Boros and Peter L. Hammer. Pseudo-boolean optimization. *DAM*, page 2002, 2001.

[10] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, November 2001.

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

[12] 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.

[13] Jason Brownlee. Understand the impact of learning rate on neural network performance, Sep 2020.

[14] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: High quality object detection and instance segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 43(5):1483–1498, 2019.

[15] Osmar Luiz Ferreira de Carvalho, Osmar Abílio de Carvalho Júnior, Anesmar Olino de Albuquerque, Pablo Pozzobon de Bem, Cristiano Rosa Silva, Pedro Henrique Guimarães Ferreira, Rebeca dos Santos de Moura, Roberto Arnaldo Trancoso Gomes, Renato Fontes Guimarães, and Díbio Leandro Borges. Instance segmentation for large, multi-channel remote sensing imagery using mask-rcnn and a mosaicking approach. *Remote Sensing*, 13(1), 2021.

[16] Augustin Cauchy et al. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

[17] 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.

[18] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019.

[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] Long Chen, Martin Strauch, and Dorit Merhof. Instance segmentation of biomedical images with an object-aware embedding learned with local constraints. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 451–459. Springer, 2019.

[21] Long Chen, Yuli Wu, and Dorit Merhof. Instance segmentation of dense and overlapping objects via layering. *arXiv preprint arXiv:2210.03551*, 2022.

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

[23] Forrester Cole, Kevin Sanik, Doug DeCarlo, Adam Finkelstein, Thomas Funkhouser, Szymon Rusinkiewicz, and Manish Singh. How well do line drawings depict shape? In *ACM SIGGRAPH 2009 papers*, pages 1–9. 2009.

[24] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[25] Arun CS Kumar, Suchendra M Bhandarkar, and Mukta Prasad. Depthnet: A recurrent neural network architecture for monocular depth prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 283–291, 2018.

[26] Achal Dave, Piotr Dollár, Deva Ramanan, Alexander Kirillov, and Ross Girshick. Evaluating large-vocabulary object detectors: The devil is in the details. *arXiv preprint arXiv:2102.01066*, 2021.

[27] Deep Learning Specialization — deeplearning.ai. https://www.deeplearning.ai/courses/deep-learning-specialization/. [Accessed 13-Jul-2023].

[28] Kaiwen Duan, Lingxi Xie, Honggang Qi, Song Bai, Qingming Huang, and Qi Tian. Location-sensitive visual recognition with cross-iou loss. *arXiv preprint arXiv:2104.04899*, 2021.

[29] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 2022.

[30] Włodzisław Duch and Norbert Jankowski. Survey of neural transfer functions. *Neural computing surveys*, 2(1):163–212, 1999.

[31] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[32] 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.

[33] Ehsan Fathi and Babak Maleki Shoja. Deep neural networks for natural language processing. In *Handbook of Statistics*, volume 38, pages 229–316. Elsevier, 2018.

[34] Kunihiko Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4):322–333, 1969.

[35] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[36] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[37] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

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

[39] Gradient Descent in Machine Learning - Javatpoint — javatpoint.com. https://www.javatpoint.com/gradient-descent-in-machine-learning. [Accessed 13-Jul-2023].

[40] 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.

[41] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask r-cnn. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[43] Derek Hoiem, Andrew N Stein, Alexei A Efros, and Martial Hebert. Recovering occlusion boundaries from a single image. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE, 2007.

[44] Matthias Holschneider, Richard Kronland-Martinet, Jean Morlet, and Ph Tchamitchian. A real-time algorithm for signal analysis with the help of the wavelet transform. In *Wavelets: Time-Frequency Methods and Phase Space Proceedings of the International Conference, Marseille, France, December 14–18, 1987*, pages 286–297. Springer, 1990.

[45] Junhwa Hur and Stefan Roth. Iterative residual refinement for joint optical flow and occlusion estimation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5754–5763, 2019.

[46] Phillip Isola and Ce Liu. Scene collaging: Analysis and synthesis of natural images with semantic layers. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3048–3055, 2013.

[47] Robert A Jacobs. Increased rates of convergence through learning rate adaptation. *Neural networks*, 1(4):295–307, 1988.

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

[49] Rohit Jena, Lukas Zhornyak, Nehal Doiphode, Vivek Buch, James Gee, and Jianbo Shi. Beyond map: Re-evaluating and improving performance in instance segmentation with semantic sorting and contrastive flow. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023.

[50] Justin Johnson, Ranjay Krishna, Michael Stark, Li-Jia Li, David Shamma, Michael Bernstein, and Li Fei-Fei. Image retrieval using scene graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3668–3678, 2015.

[51] Gaetano Kanizsa, Paolo Legrenzi, and Paolo Bozzi. Organization in vision: Essays on gestalt perception. *(No Title)*, 1979.

[52] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015.

[53] Lei Ke, Yu-Wing Tai, and Chi-Keung Tang. Deep occlusion-aware instance segmentation with overlapping bilayers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4019–4028, 2021.

[54] Salman Khan, Hossein Rahmani, Syed Afaq Ali Shah, Mohammed Bennamoun, Gerard Medioni, and Sven Dickinson. *A guide to convolutional neural networks for computer vision*, volume 8. Springer, 2018.

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

[56] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *TPAMI*, pages 147–159, 2004.

[57] Vladimir Kolmogorov and Carsten Rother. Minimizing nonsubmodular functions with graph cuts-a review. *IEEE transactions on pattern analysis and machine intelligence*, 29(7):1274–1279, 2007.

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

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

[60] VS Lakkavaram, LVS Raghuveer, C Satish Kumar, G Sai Sri, and Shaik Habeeb. A review on practical diagnostic of tomato plant diseases. *Suraj Punj J Multidiscip Res*, 9:432–435, 2019.

[61] 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.

[62] Hyunmin Lee and Jaesik Park. Instance-wise occlusion and depth orders in natural scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21210–21221, 2022.

[63] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[64] Yiding Liu, Siyu Yang, Bin Li, Wengang Zhou, Jizheng Xu, Houqiang Li, and Yan Lu. Affinity derivation and graph merge for instance segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 686–703, 2018.

[65] Rui Lu, Feng Xue, Menghan Zhou, Anlong Ming, and Yu Zhou. Occlusion-shared and feature-separated network for occlusion relationship reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10343–10352, 2019.

[66] Xiaoyang Lyu, Liang Liu, Mengmeng Wang, Xin Kong, Lina Liu, Yong Liu, Xinxin Chen, and Yi Yuan. Hr-depth: High resolution self-supervised monocular depth estimation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 2294–2301, 2021.

[67] Mateusz Malinowski and Mario Fritz. A multi-world approach to question answering about real-world scenes based on uncertain input. *Advances in neural information processing systems*, 27, 2014.

[68] 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.

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

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

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

[72] 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.

[73] T.K. Moon. The expectation-maximization algorithm. *IEEE Signal Processing Magazine*, 13(6):47–60, 1996.

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

[75] Ken Nakayama. Biological image motion processing: a review. *Vision research*, 25(5):625–660, 1985.

[76] Neural network - Wikipedia — en.wikipedia.org. https://en.wikipedia.org/wiki/Neural_network. [Accessed 13-Jul-2023].

[77] Davy Neven, Bert De Brabandere, Marc Proesmans, and Luc Van Gool. Instance segmentation by jointly optimizing spatial embeddings and clustering bandwidth. In *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, pages 8837–8845, 2019.

[78] Kemal Oksuz, Baris Can Cam, Emre Akbas, and Sinan Kalkan. Localization recall precision (lrp): A new performance metric for object detection. In *Proceedings of the European conference on computer vision (ECCV)*, pages 504–519, 2018.

[79] Anton Osokin, Denis Sumin, and Vasily Lomakin. Os2d: One-stage one-shot object detection by matching anchor features. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XV 16*, pages 635–652. Springer, 2020.

[80] Ralph HJM Otten and Lukas PPP van Ginneken. *The annealing algorithm*, volume 72. Springer Science & Business Media, 2012.

[81] Sida Peng, Wen Jiang, Huaijin Pi, Xiuli Li, Hujun Bao, and Xiaowei Zhou. Deep snake for real-time instance segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8533–8542, 2020.

[82] Renfrey Burnard Potts. Some generalized order-disorder transformations. In *Mathematical proceedings of the cambridge philosophical society*, volume 48, pages 106–109. Cambridge University Press, 1952.

[83] Lu Qi, Li Jiang, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Amodal instance segmentation with kins dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[84] Xuchong Qiu, Yang Xiao, Chaohui Wang, and Renaud Marlet. Pixel-pair occlusion relationship map (p2orm): formulation, inference and application. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV 16*, pages 690–708. Springer, 2020.

[85] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(3), 2022.

[86] 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.

[87] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[88] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[89] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention*, pages 234–241, 2015.

[90] Reuven Y Rubinstein and Dirk P Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning*, volume 133. Springer, 2004.

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

[92] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[93] Saily Shah. Convolutional neural network: An overview, Mar 2022.

[94] Yifan Si, Dawei Gong, Yang Guo, Xinhua Zhu, Qiangsheng Huang, Julian Evans, Sailing He, and Yaoran Sun. An advanced spectral–spatial classification framework for hyperspectral imagery based on deeplab v3+. *Applied Sciences*, 11(12):5703, 2021.

[95] Nathan Silberman, David Sontag, and Rob Fergus. Instance segmentation of indoor scenes using a coverage loss. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 616–631. Springer, 2014.

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

[97] Carole H Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin, and M Jorge Cardoso. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support: Third International Workshop, DLMIA 2017, and 7th International Workshop, ML-CDS 2017, Held in Conjunction with MICCAI 2017, Québec City, QC, Canada, September 14, Proceedings 3*, pages 240–248. Springer, 2017.

[98] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. A comparative study of energy minimization methods for markov random fields. In *ECCV*, pages II: 16–29, 2006.

[99] Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, and Carsten Rother. A comparative study of energy minimization methods for mrfs. *TPAMI*, 2008.

[100] Saeid Asgari Taghanaki, Yefeng Zheng, S. Kevin Zhou, Bogdan Georgescu, Puneet Sharma, Daguang Xu, Dorin Comaniciu, and Ghassan Hamarneh. Combo loss: Handling input and output imbalance in multi-organ segmentation, 2021.

[101] Daniel Tarlow and Richard Zemel. Structured output learning with high order loss functions. In Neil D. Lawrence and Mark Girolami, editors, *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pages 1212–1220, La Palma, Canary Islands, 21–23 Apr 2012. PMLR.

[102] Guanzhong Tian, Liang Liu, JongHyok Ri, Yong Liu, and Yiran Sun. Objectfusion: An object detection and segmentation framework with rgb-d slam and convolutional neural networks. *Neurocomputing*, 345:3–14, 2019.

[103] Tijmen Tieleman and G Hinton. Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. *Technical report*, 2017.

[104] Joseph Tighe, Marc Niethammer, and Svetlana Lazebnik. Scene parsing with object instances and occlusion ordering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3748–3755, 2014.

[105] Minh Tran, Khoa Vo, Kashu Yamazaki, Arthur Fernandes, Michael Kidd, and Ngan Le. Aisformer: Amodal instance segmentation with transformer. *arXiv preprint arXiv:2210.06323*, 2022.

[106] Chien-Hao Tseng, Chia-Chien Hsieh, Dah-Jing Jwo, Jyh-Horng Wu, Ruey-Kai Sheu, and Lun-Chi Chen. Person retrieval in video surveillance using deep learning–based instance segmentation. *Journal of Sensors*, 2021:1–12, 2021.

[107] Julien Valentin, Adarsh Kowdle, Jonathan T Barron, Neal Wadhwa, Max Dzitsiuk, Michael Schoenberg, Vivek Verma, Ambrus Csaszar, Eric Turner, Ivan Dryanovski, et al. Depth from motion for smartphone ar. *ACM Transactions on Graphics (ToG)*, 37(6):1–19, 2018.

[108] 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.

[109] Olga Veksler. *Efficient Graph-Based Energy Minimization Meth. in Comp. Vis.* PhD thesis, 1999.

[110] Olga Veksler. *Efficient Graph-based Energy Minimization Methods in Computer Vision*. PhD thesis, Cornell University, July 1999. Available from www.neci.nj.nec.com/homepages/olga.

[111] Guoxia Wang, Xiaochuan Wang, Frederick WB Li, and Xiaohui Liang. Doobnet: Deep object occlusion boundary detection from an image. In *Computer Vision–ACCV 2018: 14th Asian Conference on Computer Vision, Perth, Australia, December 2–6, 2018, Revised Selected Papers, Part VI 14*, pages 686–702. Springer, 2019.

[112] Kaixin Wang, Jun Hao Liew, Yingtian Zou, Daquan Zhou, and Jiashi Feng. Panet: Few-shot image semantic segmentation with prototype alignment. In *proceedings of the IEEE/CVF international conference on computer vision*, pages 9197–9206, 2019.

[113] Peng Wang and Alan Yuille. Doc: Deep occlusion estimation from a single image. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 545–561. Springer, 2016.

[114] Xinlong Wang, Tao Kong, Chunhua Shen, Yuning Jiang, and Lei Li. Solo: Segmenting objects by locations. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVIII 16*, pages 649–665. Springer, 2020.

[115] Xinlong Wang, Rufeng Zhang, Tao Kong, Lei Li, and Chunhua Shen. Solov2: Dynamic and fast instance segmentation. *Advances in Neural information processing systems*, 33:17721–17732, 2020.

[116] Yang Wang, Yi Yang, Zhenheng Yang, Liang Zhao, Peng Wang, and Wei Xu. Occlusion aware unsupervised learning of optical flow. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4884–4893, 2018.

[117] Syed Waqas Zamir, Aditya Arora, Akshita Gupta, Salman Khan, Guolei Sun, Fahad Shahbaz Khan, Fan Zhu, Ling Shao, Gui-Song Xia, and Xiang Bai. isaid: A large-scale dataset for instance segmentation in aerial images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 28–37, 2019.

[118] What is a neural network? https://www.tibco.com/reference-center/what-is-a-neural-network. [Accessed 13-Jul-2023].

[119] Christopher Xie, Yu Xiang, Arsalan Mousavian, and Dieter Fox. Unseen object instance segmentation for robotic environments. *IEEE Transactions on Robotics*, 37(5):1343–1359, 2021.

[120] Enze Xie, Peize Sun, Xiaoge Song, Wenhai Wang, Xuebo Liu, Ding Liang, Chunhua Shen, and Ping Luo. Polarmask: Single shot instance segmentation with polar representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12193–12202, 2020.

[121] Xin Yang, Hongcheng Luo, Yuhao Wu, Yang Gao, Chunyuan Liao, and Kwang-Ting Cheng. Reactive obstacle avoidance of monocular quadrotors with online adapted depth prediction network. *Neurocomputing*, 325:142–158, 2019.

[122] Yi Yang, Sam Hallman, Deva Ramanan, and Charless C Fowlkes. Layered object models for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(9):1731–1743, 2011.

[123] Michael Yeung, Leonardo Rundo, Yang Nan, Evis Sala, Carola-Bibiane Schönlieb, and Guang Yang. Calibrating the dice loss to handle neural network overconfidence for biomedical image segmentation, 2022.

[124] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[125] Stella X. Yu and Jianbo Shi. Segmentation with pairwise attraction and repulsion. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 1, pages 52–58. IEEE, 2001.

[126] Xiaoding Yuan, Adam Kortylewski, Yihong Sun, and Alan Yuille. Robust instance segmentation through reasoning about multi-object occlusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11141–11150, 2021.

[127] Xiaohang Zhan, Xingang Pan, Bo Dai, Ziwei Liu, Dahua Lin, and Chen Change Loy. Self-supervised scene de-occlusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3784–3792, 2020.

[128] Tao Zhang, Shiqing Wei, and Shunping Ji. E2ec: An end-to-end contour-based method for high-quality high-speed instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4443–4452, 2022.

[129] Yiqing Zhang, Jun Chu, Lu Leng, and Jun Miao. Mask-refined r-cnn: A network for refining object details in instance segmentation. *Sensors*, 20(4), 2020.

[130] Ziyu Zhang, Alexander G Schwing, Sanja Fidler, and Raquel Urtasun. Monocular object instance segmentation and depth ordering with cnns. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2614–2622, 2015.

[131] Hengshuang Zhao. semseg. https://github.com/hszhao/semseg, 2019.

[132] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *CVPR*, 2017.

[133] Yan Zhu, Yuandong Tian, Dimitris Metaxas, and Piotr Dollár. Semantic amodal segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1464–1472, 2017.

[134] ZunShang Zhu, Ang Su, HaiBo Liu, Yang Shang, and QiFeng Yu. Vision navigation for aircrafts based on 3d reconstruction from real-time image sequences. *Science China Technological Sciences*, 58:1196–1208, 2015.