Contextual Recurrent Level Set Networks and Recurrent Residual Networks for Semantic Labeling

Submitted in partial fulfillment of the requirements for
the degree of
Doctor of Philosophy
in the
Department of Electrical and Computer Engineering

Ngan Thi Hoang Le

B.S., Computer Science, National Science University, Vietnam
M.S., Computer Science, National Science University, Vietnam
M.S., Electrical and Computer Engineering, Carnegie Mellon University, USA

Carnegie Mellon University
Pittsburgh, PA

May, 2018

Acknowledgments

This thesis is the completion of my journey of Ph.D which was just like climbing a high peak step by step accompanied with encouragement, supporting, hardship, and frustration. When I found myself at top experiencing the feeling of fulfillment, I realized though only my name appears on the cover of this dissertation, a great many people including my family members, my friends, colleagues have contributed to accomplish this huge task. I would like to take this opportunity to thank all the people who have unconditionally supported and encouraged me throughout the years of my PhD program.

At this moment of accomplishment I am greatly indebted to my research guide, my advisor, Prof. Marios Savvides, for taking me as a PhD student and for all his support and guidance through the years. This work would not have been possible without his guidance and involvement, his support and encouragement

I am very grateful to Prof. Vijayakumar Bhagavatula, Prof. Arun Ross, and Dr. Saad Bedros, for taking the time to be members of my doctoral committee. Their feedback, along with that of Prof. Savvides, was of great value and helped me understand what I needed to focus on.

I greatly appreciate and acknowledge the support, friendships and collaboration received from my collaborators in the CyLab Biometrics Center: Khoa Luu, Chi Nhan Duong, Kha Gia Quach, Utsav Prabhu, Chandrasekhar Bhagavatula, Keshav Seshadri, Chenchen Zhu, Yutong Zheng, Ligong Han, Karanhaar Singh. I learned much from each of them through our work together and numerous inspiring conversations. They have taught me many lessons about not only work but also life.

In addition, I would like to thank my friends, my lab members Shreyas

Venugopalan, Ramzi Abiantun, Dipand Pal, Juefei Xu, Sasikanth Bendapudi, Raied Jadaany, Jingu Heo for their friendship through the years. Specially, I feel so much lucky to have Faten Mhiri to be one of the most supportive friend who have continuously encouraged, loved and cared me through the years.

I acknowledge the people who mean a lot to me, my parents Khoi Van Le and Hoan Thi Kim Hoang to whom I owe more than a PhD degree. I grant them for the selfless love, care and sacrifice they have done to shape my life and my family. Although they hardly understood what my research is, they were willing to support any decision I made. I would never be able to pay back the love and affection showered upon by my parents. My heart felt regard goes to my parents in law, specially my mother in law Hai Thi Pham for her love and moral support through the time of finishing the thesis.

I owe thanks to a very special person, my husband, Khoa Luu for his continued and unfailing love, support, sympathy and understanding during my pursuit of Ph.D degree that made the completion of thesis possible. He was always around when things became hard and I couldnt move in, he helped me to keep things in perspective. I greatly value his contribution and deeply appreciate his belief in me. I appreciate my babies, my little girl Sophia Le Luu and little boy Andrew Le Luu for abiding my ignorance and the patience they showed during my thesis writing. Words would never say how grateful I am to all of you. I consider myself the luckiest woman in the world to have such a lovely and caring family, standing beside me with their love and unconditional support.

This work has been partly funded by Carnegie Mellon University CyLab, National Institute of Justice grant no. 2013-IJ-CX-K005, US Naval Air Systems Command (NAVAIR) grant no. N68335-16-C-0177, NAVMAR Applied Sciences Corporation grant no. NASC-0040-CMU, and Carnegie Mellon Uni-

versity Software Engineering Institute CERT grant no. 6-599B2.

Abstract

Semantic labeling is becoming more and more popular among researchers in computer vision and machine learning. Many applications, such as autonomous driving, tracking, indoor navigation, augmented reality systems, semantic searching, medical imaging are on the rise, requiring more accurate and efficient segmentation mechanisms. In recent years, deep learning approaches based on Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have dramatically emerged as the dominant paradigm for solving many problems in computer vision and machine learning. The main focus of this thesis is to investigate robust approaches that can tackle the challenging semantic labeling tasks including semantic instance segmentation and scene understanding.

In the first approach, we convert the *classic variational Level Set* method to a learnable deep framework by proposing a novel definition of contour evolution named **Recurrent Level Set (RLS)**. The proposed RLS employs *Gated Recurrent Units* to solve the energy minimization of a variational Level Set functional. The curve deformation processes in RLS is formulated as a hidden state evolution procedure and is updated by minimizing an energy functional composed of fitting forces and contour length. We show that by sharing the convolutional features in a *fully end-to-end trainable framework*, RLS is able to be extended to **Contextual Recurrent Level Set (CRLS)** Networks to address semantic segmentation in the wild problem. The experimental results have shown that our proposed RLS improves both computational time and segmentation accuracy against the classic variational Level Set-based methods whereas the fully end-to-end system CRLS achieves competitive performance compared to the state-of-the-art semantic segmentation approaches on PAS-

CAL VOC 2012 and MS COCO 2014 databases.

The second proposed approach, **Contextual Recurrent Residual Networks (CRRN)**, inherits all the merits of sequence learning information and residual learning in order to simultaneously model *long-range contextual information* and learn *powerful visual representation* within a *single deep network*. Our proposed CRRN deep network consists of three parts corresponding to sequential input data, sequential output data and hidden state as in a recurrent network. Each unit in hidden state is designed as a combination of two components: a context-based component via sequence learning and a visual-based component via residual learning. That means, each hidden unit in our proposed CRRN simultaneously (1) learns long-range contextual dependencies via a context-based component. The relationship between the current unit and the previous units is performed as sequential information under an undirected cyclic graph (UCG) and (2) provides powerful encoded visual representation via residual component which contains blocks of convolution and/or batch normalization layers equipped with an identity skip connection. Furthermore, unlike previous scene labeling approaches [1, 2, 3], our method is not only able to exploit the long-range context and visual representation but also formed under a fully-end-to-end trainable system that effectively leads to the optimal model. In contrast to other existing deep learning networks which are based on pre-trained models, our fully-end-to-end CRRN is completely trained from scratch. The experiments are conducted on four challenging scene labeling datasets, i.e. SiftFlow, CamVid, Stanford background, and SUN datasets, and compared against various state-of-the-art scene labeling methods.

Index Terms

Pixel-level labeling, Semantic Instance Segmentation, Scene Labeling, Scene Parsing, Image Understanding, Level Set, Active Contour, Convolutional Neural Networks, Recurrent Neural Networks, Gated Recurrent Unit, Residual Network

Contents

List of Tables

List of Figures

List of Abbreviations

AC Active Contour

AP Average Precision

AR Average Recall

AUC Area Under Curve

BPTT Backpropagation Through Time

CLS Classic Level Set

CV Chan-Vese

CNNs Convolutional Neural Networks

CRF Conditional Random Fields

CRLS Context Recurrent Level Set

CRRN Contextual Recurrent Residual Networks

FCNs Fully Convolutional Networks

GAC Geodesic Active Contours

GRU Gated Recurrent Unit

IoU Intersection over Union

LS Level Set

LSTM Long Short Time Memory

MCG Multiscale Combinatorial Grouping

MLP Multilayer Perceptron

MRF Markov Random Fields

R-CNNs Region-based Convolutional Neural Networks

ReLU Rectified Linear Unit

ResNet Residual Networks

RLS Recurrent Level Set

RNNs Recurrent Neural Networks

RoI Region of Interest

RPNs Region Proposal Networks

SGD Stochastic Gradient Descent

UCG Undirected Cyclic Graph

Chapter 1

Introduction

One o the central goals of computer vision is object recognition and scene understanding. Humans are able to (1) possess a remarkable ability to parse an image simply by looking at them. (2) analyze an image and separate all the components present in it (3) recognize a new object that has never seen before based on observing a set of objects. The Holy Grail of a computer vision system is to able to perform as same tasks as humans are with the ability of across huge variations in pose, appearance, viewpoint, illumination, occlusion, etc. Obviously, this is a very difficult computational problem and involves in eventually making intelligent machines. An image understanding system can be roughly defined in four different subproblems i.e. image classification, object detection, semantic segmentation, semantic instance segmentation as shown in Fig. 1.1, according to its level of complexity.

1. **Image classification:** assign a label to an image. In this subproblem, all the objects presenting in a scene are given one label, independent of its location.
2. **Object detection:** predict the bounding box around an object as well as the class of each object. This subproblem is also known as object localization.

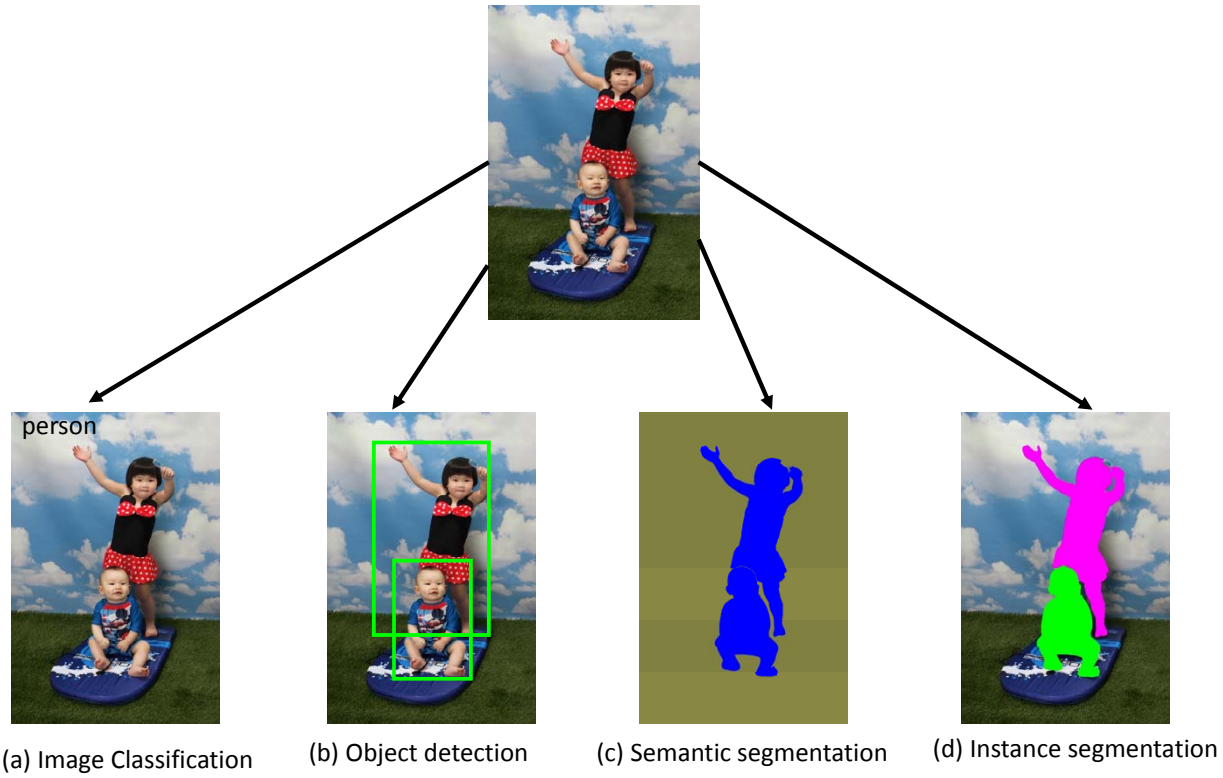


Figure 1.1: The four subproblems of image understanding (a) Image Classification, (b) Object Detection, (c) Semantic Segmentation, (d) Semantic Instance Segmentation

3. **Semantic segmentation**: predict the semantic class of the individual pixels in an image. However this subproblem unable to distinguish different instances of the same class.

4. **Instance segmentation:** label as well as provide pixel-level segmentation to each object instance in the image. This subproblem is the higher level of semantic segmentation. It can be considered as a combination of the second subproblem and third problem.

In general, object detection does not provide accurate pixel-level object segmentation and semantic segmentation ignores to distinguish different instances in the same class whereas instance segmentation labels all the pixels that belong to objects and assign each pixel to a given object instance. Clearly, the later problem is more challenging and considered to be the most challenging problem in object recognition and seen as a generalization

of the previous subproblem.

In the early days (1970s), computer vision could be broken up into three categories: (i) low-level vision, related to image processing such Laplacian, Laplacian of a Gaussian, Sobel edge detector, Canny edge detector [4], (ii) mid-level vision, leading to representations of surfaces utilizing such concepts as global motion, grouping, surfaces, and regions and (iii) high-level vision, corresponding to object understanding. Later, this category was almost replaced by feature-based learning approaches. Features evolved from edges and corners to linear filters such as Gaussian, Gabor and Haar wavelets. After that, histogram-based features were successfully developed and became an important feature-based learning with Scale-Invariant Feature Transformation (SIFT) [5] Histogram of Oriented Gradients (HOG) [6] and Speeded-Up Robust Features (SURF)[7]. Instead of using hand-crafted discriminative features, Le-Cun et al.[8, 9] introduced an important class of visual representation learning methods, called deep learning which is known as Convolutional Neural Networks (CNNs). This approach consists of a multiple level of representations which was obtained by convolving an input with receptive field. At the early days of 1990s, CNNs were not considered as important as feature-based learning in object recognition. In the last few years, however, this scenario drastically changed, mainly due to increasing amounts of available data, more powerful computing machines and some new algorithmic developments. One of the notable mention is the work done by Krizhevsky, et al. [10] which achieved impressive classification results on the challenging ImageNet dataset [11]. After that, CNN-based deep learning approaches became popular and have been pursued dramatacally in computer vision and machine learning. Image labeling and understanding including **semantic instance segmentation** and **scene labeling** is one of the key tasks which has played an important role in various areas of computer vision and machine learning.

Among numerous segmentation methods developed in last few decades, Active Contour (AC), or Deformable Models, based on variational models and partial differential equations (PDEs), can be considered as one of the most widely used approaches in med-

ical image segmentation. Among many AC-based approaches in the last few decades for image segmentation, variational LS methods [12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23] have obtained promising performance under some constraints, e.g. resolution, illumination, shape, noise, occlusions, etc. However, the segmentation accuracy in the LS methods dramatically drops when dealing with images collected in the wild, e.g. the PASCAL Visual Object Classes (VOC) Challenge [24], the Microsoft Common Objects in Context (MS COCO) [25] database, etc. Meanwhile, the recent advanced deep learning based segmentation approaches [26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45] have achieved the state-of-the-art performance on dealing with both multi-instance object segmentation and scene understanding on these databases.

The second problem in the family of semantic segmentation is known as scene labeling or scene parsing which refers to associating each pixel with one semantic class in a scene image. This task is very challenging as it implies solving jointly detection, segmentation and multi-label recognition problems. To address this issue, a large body of research has recently proposed different approaches mainly focusing on utilizing contextual information via graphical model [46, 47, 48, 49, 50, 51] or recurrent neural networks [1, 2, 3, 32, 49, 49, 52, 53, 54, 55, 56, 57, 58] or enriching visual representations via convolutional neural networks [31, 38, 53, 59, 60, 61].

1.1 Thesis Objective

The focus of this thesis is to develop different robust algorithms to tackle the pixel-level labeling problem, including semantic instance segmentation and scene labeling.

The first algorithm we introduce is aimed to bridge the two disconnected areas, i.e. connect the classic variational Level Set (LS) in image processing area to deep learning. The first proposed method is to answer the following challenging.

- LS is a well-known segmentation approach in image processing research. It is, how-

ever, limited when performing on images in the wild. *How to make LS method work well on images in the wild?*

- LS is a pure image segmentation process and there is no learning from available data while large-scale databases together machine learning (deep network approaches) have big advantages nowadays. *How to incorporate LS into deep learning to inherit the merits of both LS and deep learning?*
- Most deep learning-based semantic image segmentation methods perform the segmenting task using the softmax function. *Is it possible to replace softmax function by LS to get better outcome? If replacing softmax by LS, how can we reformulate LS to perform forward and backward pass in the deep learning framework?*

The second algorithm is to exploring all the merits of sequence learning information and residual learning in order to simultaneously model *long-range contextual information* and learn *powerful visual representation* within a *single deep network* to deal with scene labeling. The scene labeling approach is based on the observation that scene labeling problem in the real world needs both information of the context dependencies and visual representation. For example, powerful visual representation is capable to discriminate a road from a beach or sea from the sky; but it may not effective enough to tell a patch of sand belongs to the side of a road or to a beach and it is almost impossible to distinguish a pixel belonging to the sky from a pixel belonging to sea by only looking at a small patch around them. In such circumstance, pixels can not be labeled based only on short-range context i.e. a small region around them. Clearly, the context presented in the whole scene can show its advantage to describe them. Indeed, the roles of contextual information i.e. short-range and long-range context and powerful descriptive visual representation are equally important in the scene labeling problem.

1.2 Thesis Contributions

This thesis contains different contributions for pixel-level object understanding/labeling tasks, namely semantic instance segmentation and scene parsing. Semantic instance segmentation problem is presented by the first proposed approach, **Contextual Recurrent Level Set Networks for Instance Segmentation** whereas the scene labeling problem is introduced via the second proposed approach: **Deep Contextual Recurrent Residual Networks for Scene Labeling**:

1. **Contextual Recurrent Level Set Networks for Instance Segmentation:** To address the aforementioned questions in Section 1.1, this thesis first presents a novel look of LS methods under the viewpoint of deep learning approaches, i.e. Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs). The classic LS (CLS) segmentation formulation is now redefined as a recurrent learning framework, named **Recurrent Level Set (RLS) Networks**, due to its evolution properties, i.e. stability and the growth of oscillations in the models. In compare to CLS method [62], RLS is more robust and productive when dealing with images in the wild, due to the learning ability as shown in Fig. 1.2. The proposed RLS is designed as a trainable system thus it is easily extended to a fully end-to-end system, which we named **Contextual RLS (CRLS) Networks** to efficiently incorporate with other deep learning frameworks for handling the semantic segmentation in the wild. Compared to other ConvNet-based segmentation methods [26, 29, 30, 31], our proposed fully end-to-end trainable deep networks CRLS inherits all the merits of the LS model to enrich the object curvatures and the ConvNet model to encode powerful visual representation. Indeed, the proposed CRLS is able to learn both object contours via the LS energy minimization and visual representation via the sharing deep features as shown in some examples in Fig. 1.3.

2. **Deep Contextual Recurrent Residual Networks for Scene Labeling:** To effectively address the scene labeling problem, we propose a novel deep network named **Contextual Recurrent Residual Networks (CRRN)** that inherits all the merits of sequence learning information and residual learning in order to simultaneously model *long-range contextual information* and learn *powerful visual representation* within a *single deep network*. Our proposed CRRN deep network consists of three parts corresponding to sequential input data, sequential output data and hidden state. Each unit in hidden state is designed as a combination of two components: a context-based component via sequence learning and a visual-based component via residual learning. That means, each hidden unit in our proposed CRRN simultaneously (1) learns long-range contextual dependencies via context-based component. The relationship between the current unit and the previous units is performed as sequential information under an undirected cyclic graph (UCG) and (2) provides powerful encoded visual representation via residual component which contains blocks of convolution and/or batch normalization layers equipped with an identity skip connection. Furthermore, unlike previous scene labeling approaches [1, 2, 3], our method is not only able to exploit the long-range context and visual representation but is also formulated under a fully-end-to-end trainable system that effectively leads to the optimal model. In contrast to other existing deep learning network which are based on pre-trained models, our fully-end-to-end CRRN is completely trained from scratch.

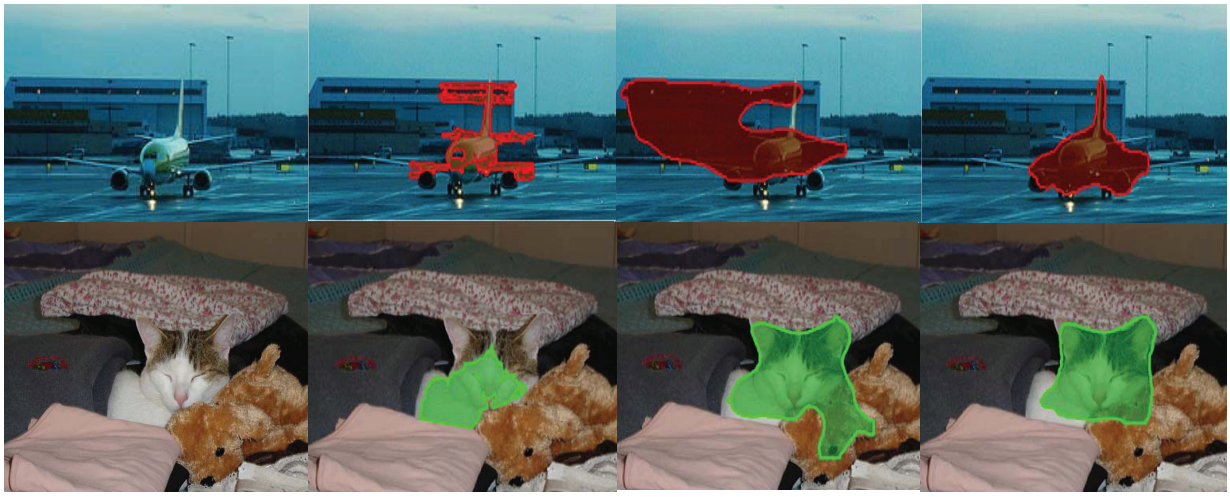


Figure 1.2: Examples of pixel-level labeling results on single object From left to right: the 1^{st} column is the input images, the 2^{nd} column is segmentation results from CLS [62], the 3^{rd} column is segmentation results from MNC [29] and the 4^{th} column is segmentation results from our **RLS**

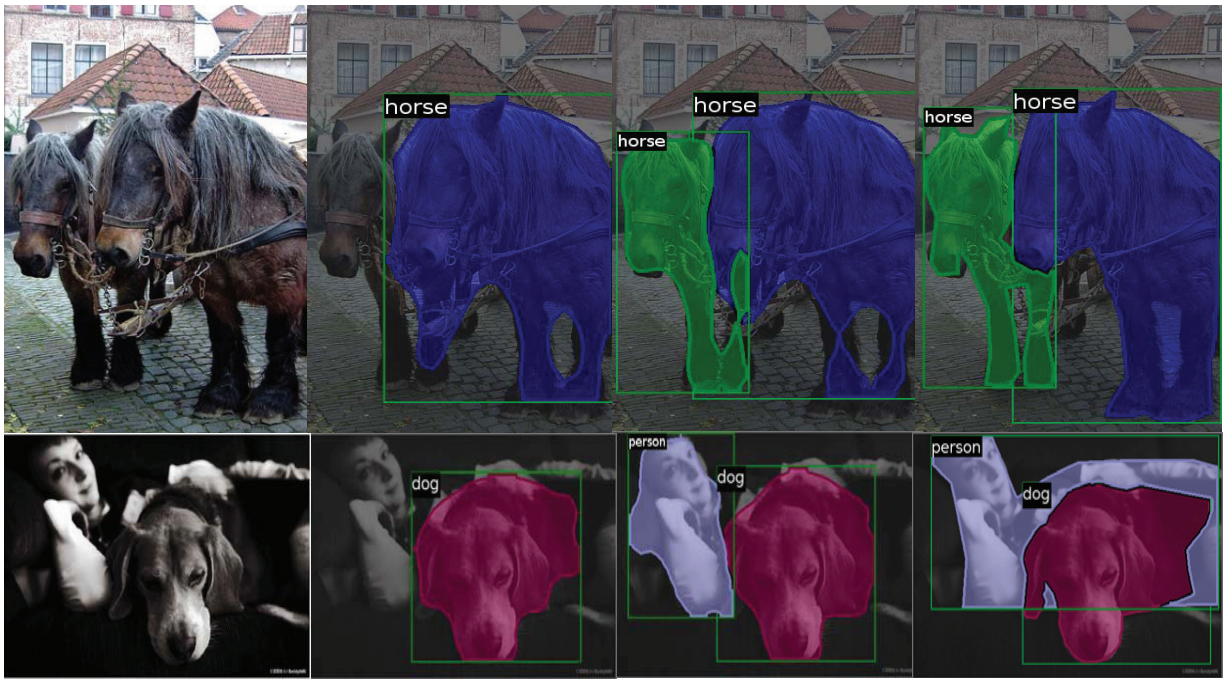


Figure 1.3: Examples of pixel-level labeling results on multiple objects From left to right: the 1^{st} column is the input images, the 2^{nd} column is segmentation results from MNC [29], the 3^{rd} column is segmentation results from our **CRLS** a nd 4^{th} column is ground truth images

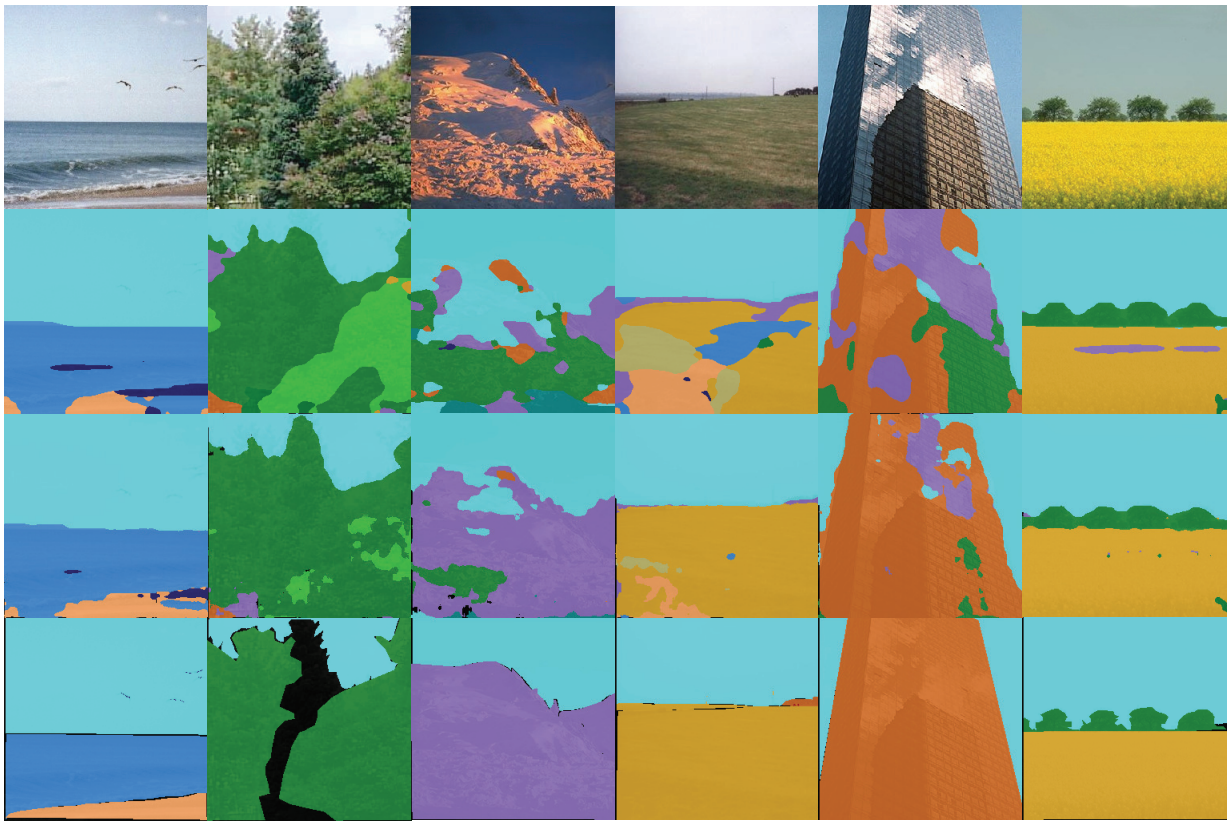


Figure 1.4: Examples of scene labeling results. From top to bottom: the input images, the segmentation results from [56], our CRRN segmentation results, and the ground truth

1.3 Thesis Outline

The rest of the thesis is organized as follows:

- **Chapter 2: Background:** In this chapter, we start with introducing the multi layer perceptron model and then present the basics of convolution neural networks (CNNs), recurrent neural networks (RNNs) and active contour (AC). The forward and backward propagations of the networks are formulated in this chapter. We provide a board survey of the recent advances in CNNs including he improvement of CNNs on different aspects such as convolutional layer, pooling layer, active function, loss function, regularization, optimization and application. Recurrent neural networks (RNNs) together the recent developments Gated Recurrent Units (GRUs) and Long short Time Memory (LSTM) are also introduced in this chapter. The

common problems in RNNs such as Backpropagation Through Time (BPTT) and vanishing problem are given later. AC with their variations are given in details after the introduction of CNNs and RNNs. The most remarkable network architectures including LeNet, AlexNet, ZFNet, VGG-16, GoogleNet, ResNet and DenseNet are summarized in this chapter. Since our proposed CRLS network for semantic instance segmentation makes use of region-based networks, we review the Region-based Convolutional Neural Network (R-CNN) family including Fast R-CNN, Faster R-CNN, Mask R-CNN and YOLO in the end of this chapter.

- **Chapter 3: Related Work:** In this chapter, we provide a brief review on different image segmentation techniques including both naive "pure image processing" approaches and learnable approaches. The chapter starts with various naive image segmentation approaches consisting of thresholding, edge-based, region-based, watershed, normalized-cut, and finally, active contour. In the next part of this chapter, we focus on deep-learning based semantic segmentation methods which have been applied to both semantic instance segmentation and scene labeling. Different approaches are introduced under three groups, i.e. graphical model approaches, CNNs-based approaches, and RNNs-based approaches. In the end of this chapter, we introduce the most common large-scale datasets that have been used to evaluate the performance of different semantic segmentation approaches.

- **Chapter 4: Contextual Recurrent Level Set (CRLS) Networks to Semantic Instance Segmentation:** In this chapter, we detail the proposed deep framework CRLS to solve the semantic instance segmentation problem. The chapter starts with analyzing the coherency between LS and RNNs and then we detail the proposed Recurrent Level Set (RLS) which inherits the merits of both LS and RNNs. The proposed RLS is designed under learnable framework to partition an image into foreground and background segments.We then explain how to extend the proposed

RLS to CRLS to solve the semantic instance segmentation. The chapter ends with CRLS inference and implementation details.

- **Chapter 5: Deep Contextual Recurrent Residual Networks (CRRN) for Scene Labeling:** In this chapter, we detail the proposed deep framework CRRN to solve the scene labeling problem. After we introduce the scene labeling problem and motivations, the proposed CRRN is detailed with model learning and inference and implement details.
- **Chapter 6: Experimental Results.** In this section, we describe the datasets, metrics that are used to evaluate the performance of each proposed approach. The comparison against the state-of-the-art on both semantic instance segmentation and scene labeling are reported in this chapter.
- **Chapter 7: Conclusions.** This section summarizes methodologies, contributions, and results and discusses for the future work.

Chapter 2

Background

This section starts with a basic description of the most common Artificial Neural Networks (ANNs) model, the Multilayer Neural Networks. We then introduce the Convolutional Neural Networks (CNNs) models, and Recurrent Neural Networks (RNNs), specialized ANN architecture particularly useful for computer vision problems. Then, we introduce Active Contour (AC) and a classic variational Level Set (LS) mechanism which have been used in image segmentation.

2.1 Multi-Layer Neural Network

2.1.1 Neuron

Let us consider the simplest possible neural network which is called "**neuron**" as illustrated in Fig. 2.1. A computational model of a single neuron is called a perceptron which consists of one or more inputs, a processor, and a single output.

In this example, the neuron is a computational unit that takes $\mathbf{x} = [x_1, x_2, x_3]$ as input, the intercept term $+1$ as bias $\mathbf{b}$, and the output $\mathbf{o}$. The gold of this simple network is to learn a function $f : \mathrm{R}^{\mathrm{N}} \rightarrow \mathrm{R}^{\mathrm{M}}$ where N is the number of dimensions for input $\mathbf{x}$ and M is

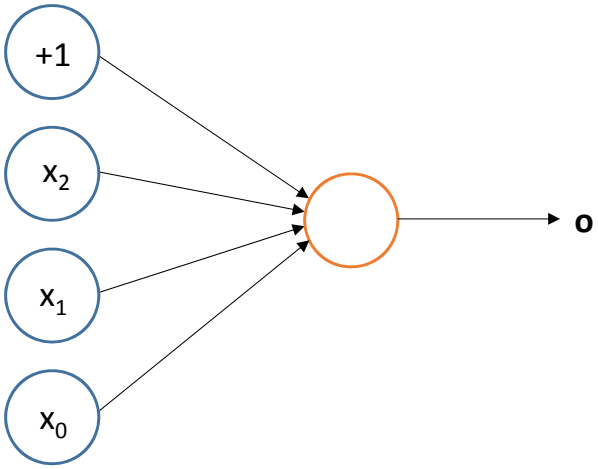


Figure 2.1: An example of one neuron which takes input $\mathbf{x} = [x_1, x_2, x_3]$, the intercept term $+1$ as bias, and the output $\mathbf{o}$.

the number of dimensions for output which is computed as $\mathbf{o} = f(\mathbf{W}, \mathbf{x})$. Mathematically, the output $\mathbf{o}$ of a one output neuron is defined as:

$$\mathbf{o} = f(\mathbf{x}, \theta) = \sigma\left(\sum_{i=1}^{N} w_i x_i + b\right) = \sigma(\mathbf{W}^T\mathbf{x} + b) \tag{2.1}$$

In this equation, σ is the point-wise non-linear activation function. The common non-linear activation function for hidden units are chosen as a hyperbolic tangent ($Tanh$) or logistic sigmoid as shown in Eq. 2.4. A different activation function, the rectified linear ($ReLU$) function, has been proved to be better in practice for deep neural networks. This activation function is different from $Sigmoid$ and ($Tanh$) because it is not bounded or continuously differentiable. Furthermore, when the network goes very deep, ReLU activations are popular as they reduce the likelihood of the gradient to vanish. The rectified linear activation ($ReLU$) function is given by Eq. 2.4. These functions are used because they are mathematically convenient and are close to linear near origin while saturating rather quickly when getting away from the origin. This allows neural networks to model well both strongly and mildly nonlinear mappings. Fig. 2.2 is the plot of $Sigmoid$, $Tanh$ and rectified linear ($ReLU$) functions.

$$Sigmoid(\mathbf{x}) = \frac{1}{1 + exp^{-x}} \tag{2.2}$$

$$Tanh(\mathbf{x}) = \frac{exp^{2x-1}}{exp^{2x+1}} \tag{2.3}$$

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

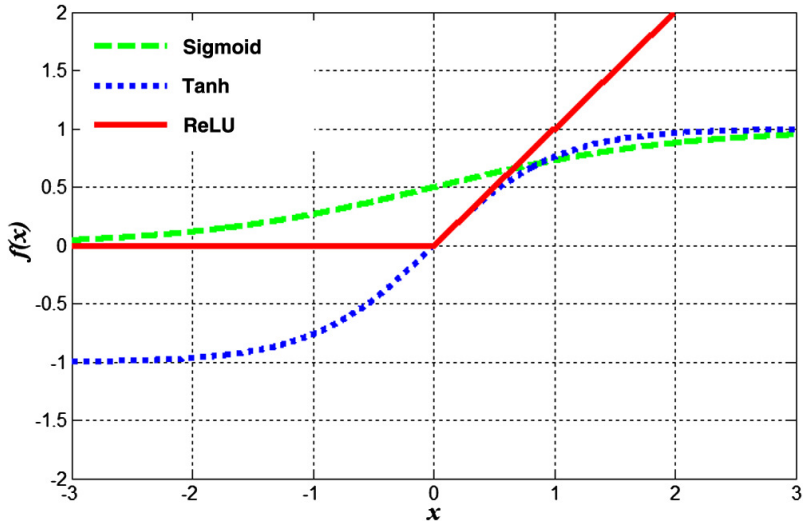


Figure 2.2: Plot of different active functions, i.e. *Sigmoid*, *Tanh* and rectified linear (*ReLU*) functions

Notably, the system becomes linear with matrix multiplications if removing the activation function. The *Tanh* activation function is actually a rescaled version of the *sigmoid*, and its output range is [1,1] instead of [0,1]. The rectified linear function is piece-wise linear and saturates at exactly 0 whenever the input is less than 0.

2.1.2 Neural Networks

A neural network is composed of many simple neurons, so that the output of a neuron can be the input to another. An special case of a neural networks is also called multi-layer perceptron network (MLP) and illustrated in Fig. 2.3.

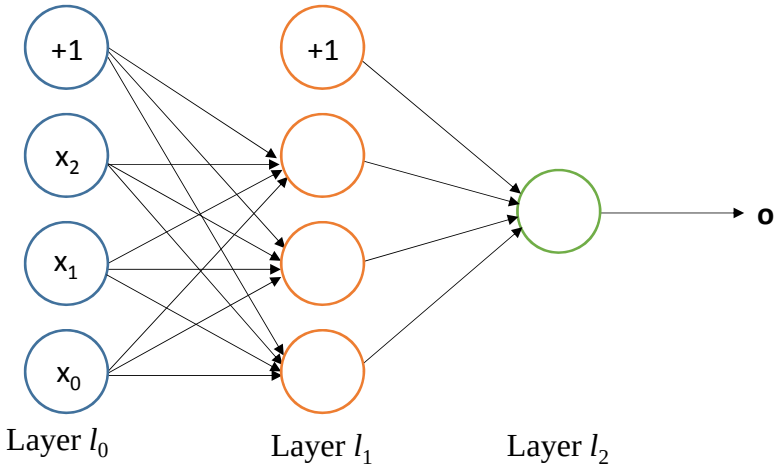


Figure 2.3: An example of multi-layer perceptron network (MLP)

A typical neural network is composed of one input layer, one output layer and many hidden layers. Each layer may contains many units. In this network, $\mathbf{x}$ is the input layer, $\mathbf{o}$ is the output layer. The middle layer is called hidden layer. In the Fig. 2.3, the neural network contains 3 units of input layers, 3 units of hidden layer, and 1 unit of output layer.

In general, we consider a neural network with L hidden layers of units, one layer of input units and one layer of output units. The number of input units is N, output units M, and units in hidden layer l is N^l. The weight of the j^{th} unit in layer l and the i^{th} unit in layer $l+1$ is denoted by w_{ij}^l. The activation of the i^{th} unit in layer l is $\mathbf{h}_i^l$. The input and output of the network are denoted as $\mathbf{x}(n)$, $\mathbf{o}(n)$, respectively, where n denotes training instance, not time. The forward propagation and backpropagation procedures are given as follows:

2.1.3 Forward propagation

Let us consider a neural network with $L+2$ layers, where the first layer is the input layer, the next $1, .., L$ layers are the corresponding hidden layers followed by the $L+1$ layer as the output layer.

The activation $\mathbf{h}$ is initialized by the input $\mathbf{x}$. Generally, the forward propagation is

expressed in the following Eq. 2.6.

$$\begin{aligned} \mathbf{h}^0 &= \mathbf{x} \\ \mathbf{h}^{l+1} &= \sigma_l(\mathbf{W}^l\mathbf{h}^l + \mathbf{b}^l), \forall l \in 0, 1, ...L-1 \\ \mathbf{o} &= \mathbf{h}^{L+1} \end{aligned} \tag{2.5}$$

where $\mathbf{W}_l, \mathbf{b}_l$ are trainable parameters corresponding to weight parameters and bias parameters for layer l, $\mathbf{x}$ is the input vector and corresponding to a vectorized image whereas $\mathbf{o}$ is the output of the network and it can be interpreted in different ways depending on the task.

In the example above in Fig. 2.3, we define $\mathbf{W}_0 \in \mathrm{R}^{3\times3}$ and $\mathbf{W}_1 \in \mathrm{R}^{1\times3}$. The input vector is considered as the initial hidden layer, i.e. $\mathbf{h}^0 = \mathbf{x}$.

$$\begin{aligned} h_0^0 &= x_0 \\ h_1^0 &= x_1 \\ h_2^0 &= x_2 \end{aligned} \tag{2.6}$$

The activation h_i^l of the unit i in layer l is computed as $\mathbf{h}^l = f(\mathbf{W}^{l-1}\mathbf{h}^{l-1})$.

$$\begin{aligned} h_0^1 &= f(w_{00}^0 h_0^0 + w_{01}^0 h_1^0 + w_{02}^0 h_2^0) \\ h_1^1 &= f(w_{10}^0 h_0^0 + w_{11}^0 h_1^0 + w_{12}^0 h_2^0) \\ h_2^1 &= f(w_{20}^0 h_0^0 + w_{21}^0 h_1^0 + w_{22}^0 h_2^0) \end{aligned} \tag{2.7}$$

The output layer is the last hidden layer and computed as:

$$\mathbf{o} = \mathbf{h}^1 = f(w_{00}^1 h_0^1 + w_{01}^1 h_1^1 + w_{02}^1 h_2^1) \tag{2.8}$$

Stochastic gradient descent (SGD) is a classical and one of the more popular optimization methods for training neural networks. It differs from the vanilla gradient descent, aka batch gradient descent, which computes the gradient of the cost function w.r.t. to the parameters θ for the entire training dataset, SGD performs a parameter update for each training example. Thus, SGD avoids redundant computations for large datasets and is much faster compared to the vanilla algorithm. It therefore can also be used to perform online learning. In SGD, the output of the network is compared to the expected output and an error is calculated. This error is then propagated back through the network, one layer at a time. The weights are updated according to the amount that they contributed to the error. This process is called backpropagation. The errors can be saved up across all of the training examples and the network can be updated at the end of this batch. This is called batch learning. Updating the network for the entire training dataset is called an epoch. A network may be trained for many epochs. When updating the network, some additional parameters terms such as momentum and learning rate decay can be defined and implemented that can improve weight updating to avoid getting stuck in local minima.

2.1.4 Backpropagation

Let consider a neural network with N_n training instances. The input and output of the network for each training instance n are denoted as $\mathbf{x}(n)$, $\mathbf{y}(n)$, where $\mathbf{x}(n) \in \mathrm{R}^N$, $\mathbf{x}(n) = (x_1(n), x_2(n), ..., x_N(n))$ and $\mathbf{y}(n) \in \mathrm{R}^M$ is the ground truth of $\mathbf{x}(n)$.

At the initial layer (layer $l = 0$): $\mathbf{x}^0(n) = (x_1^0(n), x_2^0(n), ..., x_N^0(n))$.

The activation units is computed (omitting the bias) $\mathbf{h}_i^{l+1}(n) = \sigma\left(\sum_{1 \le j \le N^l} w_{ij}^l x_j(n)\right)$.

In the output layer (layer $l = L + 1$), we obtain $\mathbf{o}(n) = (x_1^{L+1}(n), x_2^{L+1}(n), ..., x_M^{L+1}(n))$.

The objective of training is to find a set of network weights such that the summed

squared output error $\xi = \sum_{1 \leq n \leq N_n} ||\mathbf{y}(n) - \mathbf{o}(n)||^2 = \sum_{1 \leq n \leq N_n} \xi(n)$ is minimized. This is done by incrementally changing the weights along the direction of the error gradient w.r.t. weights:

$$\frac{\partial \xi}{\partial w_{ij}^l} = \sum_{1 \leq n \leq N_n} \frac{\partial \xi(n)}{\partial w_{ij}^l} \tag{2.9}$$

The new weights are updated using a small learning rate γ:

$$w_{ij}^l(new) = w_{ij}^l - \gamma \frac{\partial \xi}{\partial w_{ij}^l} \tag{2.10}$$

This formula is used in batch learning mode, where new weights are computed after presenting all training samples (epoch). In the beginning stage, the weights are initialized with random numbers.

The pseudo code of backpropagation algorithm is given in Alg. 1.

Algorithm 1 Backpropagation Algorithm

Input: current weight w_{ij}^l, training samples
Output: new weights
Computation steps:
Step 1 (forward): For each sample n, compute $\mathbf{h}_i^{l+1}(n) = f\left(\sum_{1 \leq j \leq N^l} w_{ij}^l x_j(n)\right)$.
Step 2 (backward): For each unit $\mathbf{h}_i^l$, where $l = L+1, L, ..., 1$, the error propagation term $\xi_i^m(n)$ is computed as:

- For the output layer: $\xi_i^{L+1}(n) = (\mathbf{y}_i(n) - \mathbf{o}_i(n)) \frac{\partial f(\mathbf{u})}{\partial \mathbf{u}}|_{\mathbf{u}=\mathbf{z}_i^{L+1}}$
- for hidden layers: $\xi_j^l(n) = \sum_{1 \leq i \leq N^{l+1}} \xi_i^{l+1} w_{ij}^l \frac{\partial f(\mathbf{u})}{\partial \mathbf{u}}|_{\mathbf{u}=\mathbf{z}_j^l}$
  , where $\mathbf{z}_i^l(n) = \sum_{1 \leq j \leq N^{l-1}} \mathbf{h}_j^{l-1}(n) w_{ij}^{l-1}$

Step 3: Adjust the connection weights according to $w_{ij}^l(new) = w_{ij}^l - \gamma \frac{\partial \xi}{\partial w_{ij}^l}$

2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [8, 9, 63, 64] are a special case of fully connected multi-layer perceptrons that implement weight sharing for processing data that has a

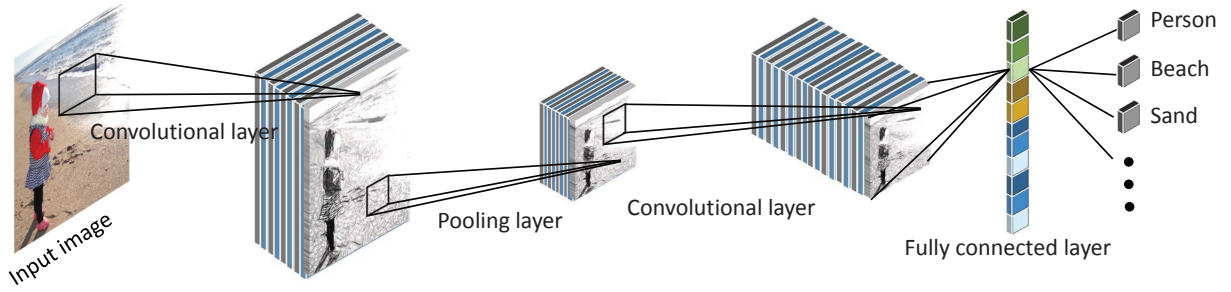


Figure 2.4: Architecture of a typical convolutional network for image classification containing three basic layers: convolution layer, pooling layer and fully connected layer

known, grid-like topology (e.g. images). CNNs use the spatial correlation of the signal to constrain the architecture in a more sensible way. Their architecture, somewhat inspired by the biological visual system, possesses two key properties that make them extremely useful for image applications: spatially shared weights and spatial pooling. These kind of networks learn features that are shift-invariant, i.e., filters that are useful across the entire image (due to the fact that image statistics are stationary). The pooling layers are responsible for reducing the sensitivity of the output to slight input shift and distortions. Since 2012, one of the most notable results in Deep Learning is the use of convolutional neural networks to obtain a remarkable improvement in object recognition for ImageNet classification challenge.

A typical convolutional network is composed of multiple stages, as shown in Fig. 2.4. The output of each stage is made of a set of 2D arrays called feature maps. Each feature map is the outcome of one convolutional (and an optional pooling) filter applied over the full image. A point-wise non-linear activation function is applied after each convolution. In its more general form, a convolutional network can be written as:

$$\begin{aligned} \mathbf{h}^0 =& \mathbf{x} \\ \mathbf{h}^l =& pool^l(\sigma_l(\mathbf{w}^l\mathbf{h}^{l-1} + \mathbf{b}^l)), \forall l \in 1, 2, ...L \\ \mathbf{o} =& \mathbf{h}^L = f(\mathbf{x}, \theta) \end{aligned} \tag{2.11}$$

where $\mathbf{w}^l, \mathbf{b}^l$ are trainable parameters as in MLPs at layer l. $\mathbf{x} \in \mathrm{R}^{c \times h \times w}$ is vectorized from an input image with c is color channels, h is the image height and w is the image width. $\mathbf{o} \in \mathrm{R}^{n \times h' \times w'}$ is vectorized from an array of dimension $h' \times w'$ of output vector (of dimension n). $pool^l$ is a (optional) pooling function at layer l.

The main difference between MLPs and CNNs lies in the parameter matrices $\mathbf{w}_l$. In MLPs, the matrices can take any general form, while in CNNs these matrices are constraints to be Toeplitz matrices. That is, the columns are circularly shifted versions of the signal of various shifts for each columns in order to implement spatial correlation operation using matrix algebra. Moreover, these matrices are very sparse since the kernel is usually much smaller than the input image. Therefore, each hidden unit array $\mathbf{h}_l$ can be expressed as a discrete-time convolution between kernels from $\mathbf{w}_l$ and the previous hidden unit $\mathbf{h}_{l-1}$ (transformed through a point-wise non-linearity and possibly pooled). There are numerous variants of CNNs architectures in the literature. However, their basic components are very similar which contains of convolutional layer, pooling layer, activation layer and fully connected layer.

2.2.1 Convolutional Layer

The convolutional layer aims to learn feature representations of the inputs and it is composed of several convolution kernels which are used to compute different feature maps. In a convolutional layer, each neuron of a feature map is connected to a region of neighbouring neurons in the previous layer. The new feature map can be obtained by first convolving the input with a kernel and then applying an element-wise nonlinear activation function on the convolved results. The feature maps are obtained by using a set of different kernels. The feature value at location (i, j) of the l^{th} layer in the k^{th} feature map is computed as:

$$o^l_{i,j,k} = w^l_k x^l_{i,j} + b^l_k \tag{2.12}$$

where $x_{i,j}^{l}$ is the input patch centered at location (i, j) of the l^{th} layer. w_k^l and b_k^l are learned weight matrix and bias matrix at the l^{th} layer in the k^{th}.

Tiled convolution [65] is a variation of traditional convolution that tiles and multiples feature maps to learn rotational and scale invariant features. Separate kernels are learned within the same layer, and the complex invariances can be learned implicitly by square root pooling over neighboring units. Recently *dilated CNNs* [40] have been introduced with one more hyper-parameter to the convolutional layer. By inserting zeros between filter elements, Dilated CNNs can increase the networks receptive field size and let the network cover more relevant information. This is very important for tasks which need a larger receptive field when making the prediction. *Transposed convolution* [66] can be seen as the backward pass of a corresponding traditional convolution. It is also known as deconvolution. A comparison between these different types of convolutions is given in Fig. 2.5.

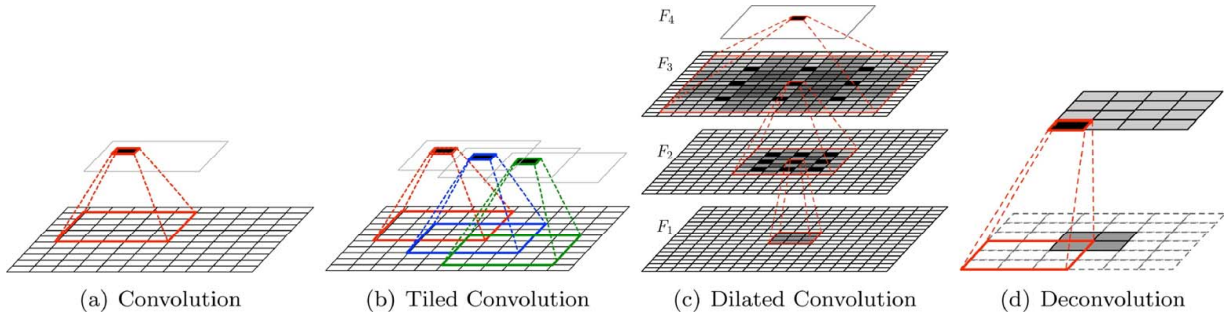


Figure 2.5: Comparison between different convolutions (a) Convolution, (b) Tiled Convolution, (c) Dilated Convolution, and (d) Transposed convolution.

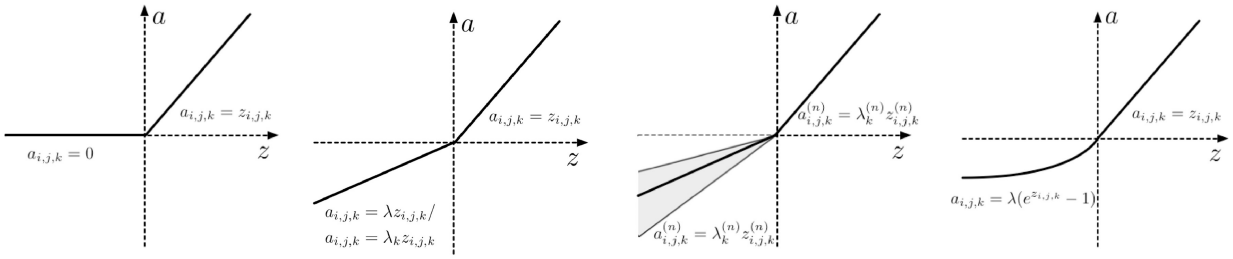
2.2.2 Activation Layer

The activation function introduces nonlinearities to CNNs. This layer controls how the signal flows from one layer to the next, emulating how neurons are fired in our brain. Let $\sigma(.)$ denote a nonlinear activation function. The activation value $a_{i,j,k}^{l}$ of convolutional

feature $z^l_{i,j,k}$ can be computed as:

$$a^l_{i,j,k} = \sigma(o^l_{i,j,k}) \tag{2.13}$$

Clearly, a proper activation function significantly can improve the performance of a CNN for a certain task. Going beyond the traditional *ReLU, Sigmod, Tanh* , there are some recent developments on activation functions. A potential disadvantage of ReLU unit is that it has zero gradient whenever the unit is not active. This may cause units that do not active initially to never become active as the gradient-based optimization will not adjust their weights. Also, it may slow down the training process due to the constant zero gradients. To alleviate this problem, [67] introduces *LReLU* which has a non-zero gradient over its entire domain, unlike the standard ReLU function. Rather than using a predefined parameter in ReLU, LReLU, [68] proposes the *Parametric Rectified Linear Unit* (PReLU) that generalizes the traditional rectified unit. PReLU improves model fitting with nearly zero extra computational cost and little overfitting risk. Furthermore, PReLU gives robust initialization which enables us to train extremely deep rectified models directly from scratch and to investigate deeper or wider network architectures. Inspirited by LReLU, [69] proposes *Randomized Leaky Rectified Linear Unit* (RReLU). In RReLU, the parameters of negative parts are randomly sampled from a uniform distribution in training, and then fixed in testing. To speed up learning of deep neural networks and achieve higher classification accuracies, [70] proposes *ELU*. Like ReLU, and its variations, ELU avoids the vanishing gradient problem by setting the positive part to identity. Different from ReLU, ELU has a negative part which is beneficial for fast learning. A comparison between activation functions are given in Fig. 2.6 whereas the mathematically equations are given in Table. 2.1.



(a) ReLU (b) LReLU/PReLU (c) RReLU (d) ELU

Figure 2.6: Comparison between different activation functions (a) ReLU , (b) LReLU/PReLU, (c)RReLU, (d)ELU

Table 2.1: Mathematically equations of different activation functions

Activation	Equation	Explanation
ReLU	$a_{i,j,k} = max(h_{i,j,k}, 0)$	$h_{i,j,k}$: input of the activation func at (i, j) on the k^{th}channel
LReLU	$a_{i,j,k} = max(h_{i,j,k}, 0) + \lambda min(h_{i,j,k}, 0)$	λ: predefined parameter in (0, 1)
PReLU	$a_{i,j,k} = max(h_{i,j,k}, 0) + \lambda_k min(h_{i,j,k}, 0)$	λ_k: learned parameter at k^{th} channel.
RReLU	$a^n_{i,j,k} = max(h^n_{i,j,k}, 0) + \lambda^n_k min(h^n_{i,j,k}, 0)$	$z^n_{i,j,k}$: input of activation func at (i, j) on the k^{th} channel of n^{th} sample
ELU	$a_{i,j,k} = max(h_{i,j,k}, 0) + min(\lambda(e^{h_{i,j,k}}, 0)$	λ: predefined parameter, it controls the saturation for negative input

2.2.3 Pooling layer

Pooling layer is also called sampling layer to smooth the input from the convolutional layer. The pooling layer helps to (1) reduce the sensitivity of the filters to noise and variations. Pooling operation is usually placed between two convolutional layers. Each feature map of a pooling layer is connected to its corresponding feature map of the preceding convolutional layer. Let $pool(.)$ denote the pooling function for the feature map $a^l_{,,k}$ at the l^{th} layer, the pooling layer can be expressed as follows:

$$h^l_{i,j,k} = pool(a^l_{i',j',k}) \forall i', j' \in W(i,j) \tag{2.14}$$

where $W(i, j)$ denotes the neighborhood around location.

To give sufficient conditions for the design of invertible neural network layers, [71] pro-

poses $L_p pooling$ which is done via a summary statistic over the outputs of groups of nodes. When p = 1, l_p pooling corresponds to average pooling, and the case $p = \inf$, it is max pooling. To replace the deterministic pooling operations with a stochastic procedure by randomly using the conventional max pooling and average pooling methods, [72] proposes *mixed pooling*. Mixed pooling has ability to address the over-fitting problem encountered in CNNs generation. *Stochastic pooling* [73] is a dropout-inspired pooling method for regularizing large convolutional neural networks. Instead of picking the maximum value within each pooling region as max pooling does, stochastic pooling randomly picks the activations according to a multinomial distribution, which ensures that the non-maximal activations of feature maps are also possible to be utilized. This stochastic pooling is hyper-parameter free and can be combined with other regularization approaches, such as dropout and data augmentation. *Spectral pooling* [74] performs dimensionality reduction by cropping the representation of input in frequency domain. As it is proven, Spectral pooling preserves considerably more information per parameter than other pooling strategies and enables flexibility in the choice of pooling output dimensionality. *Spatial pyramid pooling* (SPP) [75] is another pooling strategy to generate a fixed-length representation regardless of image size/scale. This type of pooling is robust to object deformations.

2.2.4 Loss function

It is important to choose an appropriate loss function for a specific task. *Softmax* loss is a commonly used loss function which is essentially a combination of multinomial logistic loss and softmax. Softmax turns the predictions into non-negative values and normalizes them to get a probability distribution over classes. Such probabilistic predictions are used to compute the multinomial logistic loss. Recently, work to improve Softmax is proposed by [76], called Large-Margin Softmax (L-Softmax). *L-Softmax* introduces an angular margin to the angle θ_{ij} between input feature vector x_i and the j^{th} column w_j of weight

Table 2.2: Mathematically equations of different loss functions

Loss	Equation	Explanation
Softmax	$p_j^i = \frac{e^{z_j^i}}{\sum_{k=1}^{K} e^{z_k^i}}$ $L = -\frac{1}{N}\left[\sum_{i=1}^{N}\sum_{j=1}^{K} logp_j^i\right]_{y^i=j}$	N: number of training samples K: number os classes
L-Softmax	$\phi(\theta_j) - (-1)^k cos(m\theta_j) - 2k$ $t_j = \|w_j\|\|a^i\|\phi(\theta_j)$ $L = \frac{t_j}{t_j + \sum_{k\neq j} t_k}$	$k \in [0, m-1]$ is an integer, m controls the margin among classes. m = 1: L-Softmax reduces to softmax
Contrastive	$d^{i,l} = \|\|h_p^{(i,l)} - h_q^{(i,l)}\|\|_2^2$ $L = \frac{1}{2N}[z(d^{(i,L)})$ $+(1-z)max((m - d^{(i,L)}), 0)]$	$h_p^{(i,l)}, h_q^{(i,l)}$ are 2 output of a pair of input $x_p^{(i)}, x_q^{(i)}$ at layer l^{th} m is a margin parameter $z = 1$ if $x_p^{(i)}, x_q^{(i)}$ are matching. Otherwise, $z = 0$
Triplet	$d_{a,p}^i = \|\|h_a^i - h_p^i\|\|_2^2$ $d_{a,q}^i = \|\|h_a^i - h_q^i\|\|_2^2$ $L = \frac{1}{N}\sum_{i=1}^{N} max(d_{a,p}^i - d_{a,q}^i + m, 0)$	x_a^i is an anchor instance of class i x_p^i is a positive instance x_q^i is a negative instance
Coupled Clusters	$d_{pp} = \|\|h_p^i - c_p\|\|_2^2$ $d_{qp} = \|\|h_q^* - c_p\|\|_2^2$ $c_p = \frac{1}{N_p}\sum_{i=1}^{N_p} h_p^i$ $L = \frac{1}{2N_p}\sum_{i=1}^{N_p} max(d_{pp} - d_{qp} + m, 0)$	N_p: number of samples per set h_q^*: the feature representation of x_q^* x_q^*: the nearest negative sample to c_p c_p: estimated center point

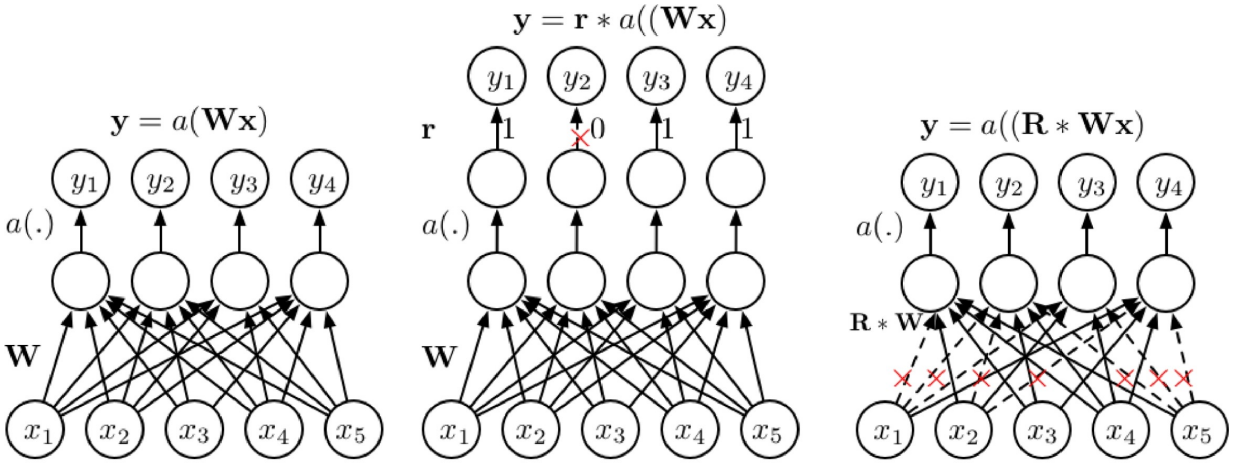
matrix. *Contrastive loss* [77] is commonly used to train Siamese network[78], which is a weakly-supervised scheme for learning a similarity measure from pairs of data instances labelled as matching or non-matching. Unlike Softmax, L-Softmax and Contrastive losses, *Triplet loss* [79] considers three samples per loss. Triplet loss and its variants have been widely used in various tasks, including re-identification, verification, and image retrieval. However, randomly selected anchor samples may judge falsely in some special cases which is then solved by [80] *Coupled Clusters* (CC) loss. In the coupled clusters loss function is defined over the positive set and the negative set instead of using the triplet units like Triplet loss.

Table 2.3: summary on regularization

Loss	Equation	Explanation
l_p-norm	$E = L + \lambda R$	R : regularization term λ: regularization strength
DropOut	$y = r * f(\mathbf{Wx})$	$\mathbf{x}$: input to fully-connected layers $\mathbf{W}$: weight matrix $\mathbf{r}$: binary vector , drawn from a Bernoulli $r_i = Bernoulli(p)$ $f(.)$: nonlinear activation function
DropConnect	$y = f(\mathbf{R} * \mathbf{Wx})$	$R_{ij} = Bernoulli(p)$

2.2.5 Regularization

Regularization aims to reduce the overfitting problem during training. l_p-*norm*, Dropout, and DropConnect are some common regularization techniques have been developed to solve this. l_p-norm is applied to modify the objective function by adding additional terms that penalize the model complexity. When $p = 2$, the l_2-norm regularization is referred to as weight decay. When $l < 1$, l_p regularization focuses more on sparsity effect of the weight but conducts to non-convex function. When $l >= 1$, l_p regularization is convex and it makes the optimization easier. Dropout regularization[81] has been proven to be very effective in reducing overfitting. Several methods [82, 83, 84, 85] have been proposed to improve Dropout. [82] proposes a fast *Dropout* method by sampling from a Gaussian approximation. [83] proposes an adaptive Dropout method, where the Dropout probability for each hidden variable is computed using a binary belief network that shares parameters with the deep network. [84] figures out that applying standard Dropout before 1×1 convolutional layer generally increases training time but does not prevent overfitting. Therefore, they propose SpatialDropout to across the entire feature map. *DropConnect* [85] a generalization of DropOut. Instead of setting a randomly selected subset of activations to zero. DropConnect sets a randomly selected subset of weights within the network to zero. Table 2.3 gives a summary on regularization whereas Fig. 2.7 illustrate the comparison between different regularization techniques.



(a) No-Drop (b) DropOut (c) DropConnect

Figure 2.7: Comparison between different techniques (a) l_p-norm, (b) DropOut, (c)DropConnect

2.2.6 Optimization

Data augmentation, weight initialization, stochastic gradient descent, batch normalization, shortcut connections are some common optimization techniques that will be discussed. Popular *data augmentation* methods are sampling [86], various photometric transformations [87], shiftting [88] and greedy strategy [89]. Crawling web to obtain more data in fine-grained recognition tasks is suggested by [90]. Since deep networks contains a large number of parameters and loss functions are non-convex, a proper *weight initialization* is one of the most important prerequisites. [86] suggests to use zero-mean Gaussian distribution with standard deviation 0.01 and the bias (2^{nd}, 4^{th}, 5^{th}, fully connected) to constant. [91] chooses a Gaussian distribution with zero mean and a variance of $0.5(N_{in} + N_{out})$, where N_{in} and N_{out} are the number of input and output neurons. Later, [92] shows that orthonormal matrix initialization works much better for linear networks than Gaussian initialization. *Stochastic gradient descent* [93](SGD) is the most popular technique used to update the parameter during backpropagation. In practice, SGD is updated using mini-batch as opposed to a single example. However, mini-batch SGD methods may not result in convergence. To solve the problem, momentum [93] is proposed to accumulate a velocity

vector in the relevant direction. Parallelized SGD methods [94] are proposed to improve SGD to make it more suitable for parallel, large-scale machine learning. Batch Normalization [95] is one of the most common data normalization and has many advantages such as faster learning and higher overall accuracy compared with global data normalization. Even it able to solve the problems of vanishing gradient and prevent deep neural networks from overfitting, it still have difficulties in optimizing the networks, worse generalization performance. *Shortcut connections* is then proposed in [96] to help gradients can be directly propagated to shallower units which help the networks (ResNet) much easier to be optimized than the original mapping function and more efficient to train very deep networks.

2.2.7 Fully connected layer

After several convolutional and pooling layers, there may be one or more fully connected layers which aim to perform high-level reasoning. This is the last layer in the neural network. The term fully connected implies that every neuron in the previous layer is connected to every neuron on the next layer. This mimics high level reasoning where all possible pathways from the input to output are considered (e.g. classifying the input image into various classes based on the training dataset). The fully connected layer uses Softmax as the activation function to ensure the sum of output probabilities from the fully connected layer is 1. Note that fully connected layer not always necessary as it can be replaced by a 1×1 convolution layer.

2.2.8 Applications

Herein, we introduce some recent works that apply CNNs to achieve state-of-the-art performance. *Object detection* has been a long-standing and important problem in computer vision [97]. Object proposal based methods attract a lot of interests and are widely studied

in the literature [98, 99]. The methods are to test whether a sampled window is a potential object or not, and further pass the output object proposals to more sophisticated detectors to determine whether they are background or foreground. R-CNN [99]uses Selective Search (SS) to extract around 2000 bottom-up region proposals that are likely to contain objects. Spatial pyramid pooling network (SPP net) [75], Fast-RCNN, Faster R-CNN [100] are improvements of R-CNNs. CNNs have been applied in *image classification* for a long time [101]. Compared to traditional methods, CNNs achieve better classification accuracy on large scale datasets [86]. With large number of classes, proposing a hierarchy of classifiers is a common strategy for image classification [102]. Fine-grained using object part information is another strategy for image classification [103]. *Visual tracking* is an another application that turns the CNNs model from a detector into a tracker [104]. [105] uses pre-trained CNNs to transfer the learned representation to target object. In this work, they use an online SVM to learn a target appearance discriminatively against background. Instead of learning one complicated and powerful CNN model, [106] use a relatively small number of filters in the CNNs within a framework equipped with a temporal adaptation mechanism. As an special case of image segmentation, *saliency detection* is another computer vision application that uses CNNs. [107] introduce a CNN-based algorithm which sequentially exploits local context and global context. In [108], the model is pre-trained on large-scale image classification dataset and then shared among different contextual levels for feature extraction. In additional to the previous applications, *pose estimation* [109, 110] is another interesting research that uses CNNs to estimate human-body pose. Instead of learning local body parts, [111, 112] learn full-body human pose estimation. In addition to still image pose estimation with CNNs, human pose estimation in videos have been studied [113]. *Action recognition* in both still images and in videos are special case of recognition and are challenging problems. [114] utilizes CNN-based representation of contextual information in which the most representative secondary region within a large number of object proposal regions together the contextual features are used to describe the primary region.

Action recognition in video sequences are reviewed in [115]. *Text detection and recognition* using CNNs is the next step of optical character recognition (OCR). [116, 117] focus on text detection which is the successor step of text recognition which is taken care by both CNNs and RNNs [118]. For word spotting, [119, 120] apply CNNs model to perform text detection. One of the main computer vision applications that inherit big benefit of CNNs is *semantic image parsing* which is then detailed in te next chapter. Going beyond still images and videos, *speech recognition* is also an important research field that have been improved by applying CNNs. An attention model with layer-wise context expansion in [121] archives good performance for speech recognition task. *Speech synthesis* [122] is an another task that effected by the growth of CNNs. In short, CNNs have made breakthroughs in many computer vision areas i.e image, video, speech and text. The summary on the works related to CNNs is given in Fig. 2.8.

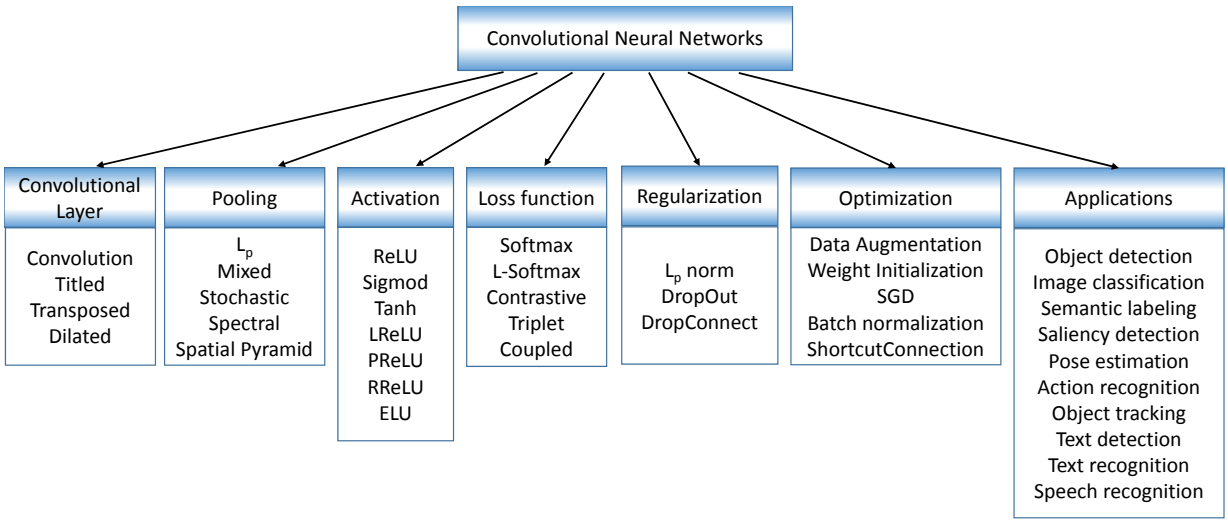


Figure 2.8: Summary on the works related to CNNs

To train a network we consider two main procedures, namely, forward propagation and backward propagation. For each procedure, two important layers are taking into account i.e. convolutional layer and pooling layer.

2.2.9 Forward Propagation

Convolutional layer

Suppose we have a image sized $N \times N$ which is followed by a convolutional layer with a filer w of size $n \times n$. The neural network learns by adjusting a set of weights w_{ij}^l. The convolution can be expressed as:

$$o_{ij}^l = \sum_{a=0}^{n-1} \sum_{b=0}^{n-1} w_{ab} h_{i+a,j+b}^{l-1} + \mathbf{b}^l \tag{2.15}$$

where $h_{i+a,j+b}^{l-1}$ is the output of the previous $(l-1)^{th}$ layer and the input of the l^{th} layer.

Activation layer

Let $\sigma(.)$ denote the activation function. The activation at layer l^{th} is computed as:

$$a_{ij}^l = \sigma(o_{ij}^l) \tag{2.16}$$

Pooling layer Let us assume max pooling as the pooling approach implemented in the pooling layer. The max-pooling layers are quite simple. For this layers, they simply take take a $k \times k$ region as an input and output is a single value, which is the maximum value in that region. The input layer is $N \times N$ is divided into $\frac{N}{k} \times \frac{N}{k}$ blocks of size $k \times k$, each block is reduced to a single value via the max function.

$$h_{ij}^l = pool^l(a_{ij}^l) \tag{2.17}$$

In order to tell how well the neural network is doing, we compute the error $\xi(y^L)$.

2.2.10 Backward Propagation

Convolutional Layer: Let assume we have some error E at the convolutional layer. By applying the chain rule, sum the contributions of all expressions in which the variable

occurs.

$$\frac{\partial E}{\partial w_{ab}} = \sum_{i=0}^{N-n}\sum_{j=0}^{N-n}\frac{\partial E}{\partial o_{ij}^{l}}\frac{\partial o_{ij}^{l}}{\partial w_{ab}} = \sum_{i=0}^{N-n}\sum_{j=0}^{N-n}\frac{\partial E}{\partial o_{ij}^{l}}h_{(i+a)(j+b)}^{l-1} \tag{2.18}$$

In order to compute the gradient, we need to know the values $\frac{\partial E}{\partial o_{ij}^{l}}$.

$$\frac{\partial E}{\partial o_{ij}^{l}} = \frac{\partial E}{\partial h_{ij}^{l}}\frac{\partial h_{ij}^{l}}{\partial o_{ij}^{l}} = \frac{\partial E}{\partial h_{ij}^{l}}\frac{\partial \sigma(o_{ij}^{l})}{\partial o_{ij}^{l}} = \frac{\partial E}{\partial h_{ij}^{l}}\sigma'(o_{ij}^{l}) \tag{2.19}$$

Since we already know the error at the current layer $\frac{\partial E}{\partial h_{ij}^{l}}$, we can very easily compute $\frac{\partial E}{\partial o_{ij}^{l}}$ at the current layer by just using the derivative of the activation function $\sigma'(o_{ij}^{l})$.

Therefore, the gradient component for each weight by applying the chain rule is.

$$\frac{\partial E}{\partial w_{ab}} = \sum_{i=0}^{N-n}\sum_{j=0}^{N-n}\frac{\partial E}{\partial h_{ij}^{l}}\sigma'(o_{ij}^{l})h_{(i+a)(j+b)}^{l-1} \tag{2.20}$$

After computing the weights for this convolutional layer, we need to propagate errors back to the previous layer.

$$\frac{\partial E}{\partial h_{ij}^{l-1}} = \sum_{a=0}^{n-1}\sum_{b=0}^{n-1}\frac{\partial E}{\partial o_{(i-a)(j-b)}^{l}}\frac{\partial o_{(i-a)(j-b)}^{l}}{\partial h_{ij}^{l}} = \sum_{a=0}^{n-1}\sum_{b=0}^{n-1}\frac{\partial E}{\partial o_{(i-a)(j-b)}^{l}}w_{ab} \tag{2.21}$$

This is similar convolution in the forward propagation. But the filter w is applied to $o_{(i-a)(j-b)}^{l}$ instead of $o_{(i+a)(j+b)}^{l}$.

Pooling Layer: In forward propagation, $k \times k$ (k=2) blocks can be reduced to a single value. Then, at the backwards propagation this single value acquires an error computed from the previous layer. This error is then just forwarded to the neuron where it came from. Since it only came from one neuron in the $k \times k$ block, the back-propagated errors from max-pooling layers are rather sparse. Therefore, the max-pooling layers do not actually do any learning themselves, they actually reduce the size.

2.3 Recurrent Neural Networks

2.3.1 Vanilla Recurrent Neural Networks:

The Recurrent Neural Networks (RNNs) is an extremely powerful sequence model and was introduced in the early 1990s [123]. A typical RNNs contains three parts, namely, sequential input data ($\mathbf{x}_t$), hidden state ($\mathbf{h}_t$) and sequential output data ($\mathbf{o}_t$) as shown in Fig. 2.9.

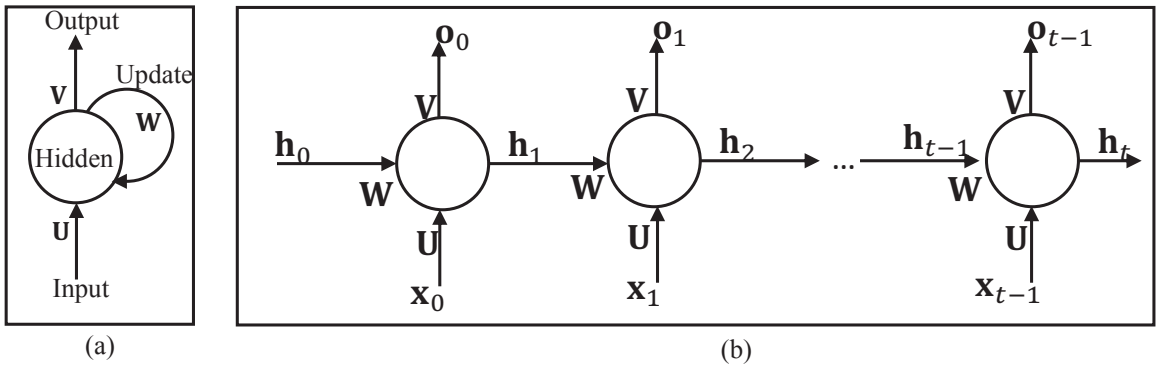


Figure 2.9: An RNNs and the unfolding in time of the computation involved in its forward computation.

RNNs make use of sequential information and perform the same task for every element of a sequence where the output is dependent on the previous computations. The activation of the hidden states at timestep t is computed as a function f of the current input symbol $\mathbf{x_t}$ and the previous hidden states $\mathbf{h_{t-1}}$. The output at time t is calculated as a function g of the current hidden state $\mathbf{h_t}$ as follows:

$$\begin{aligned} \mathbf{h}_t &= f(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1}) \\ \mathbf{o}_t &= g(\mathbf{V}\mathbf{h}_t) \end{aligned} \tag{2.22}$$

where $\mathbf{U}$ is the input-to-hidden weight matrix, $\mathbf{W}$ is the state-to-state recurrent weight matrix, $\mathbf{V}$ is the hidden-to-output weight matrix. f is usually a logistic sigmoid function or a hyperbolic tangent function and g is defined as a softmax function.

Most work on RNNs has made use of the method of backpropagation through time (BPTT) [124] to train the parameter set $(\mathbf{U}, \mathbf{V}, \mathbf{W})$ and propagate error backward through time. In classic backpropagation, the error or loss function is defined as:

$$E(\mathbf{o}, \mathbf{y}) = \sum_t ||\mathbf{o}_t - \mathbf{y}_t||^2 \tag{2.23}$$

where $\mathbf{o}_t$ is prediction and $\mathbf{y}_t$ is labeled groundtruth.

For a specific weight $\mathbf{W}$, the update rule for gradient descent is defined as $\mathbf{W}^{new} = \mathbf{W} - \gamma \frac{\partial E}{\partial \mathbf{W}}$, where γ is the learning rate. In RNNs model, the gradients of the error with respect to our parameters $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{W}$ are learned using Stochastic Gradient Descent (SGD) and chain rule of differentiation.

In practice, there are different ways to desgin a RNNs architecture. In a simple case, RNNs accepts a fixed-sized vector as input (e.g. an image) and produce a fixed-sized vector as output (e.g. probabilities of different classes) as given in the first model in Fig. 2.10. More flexible, RNNs can have various architecture as shown in Fig. 2.10. An example of RNNs architecture for sentiment analysis with an sentence as sequence input and score as fixed-sized output is illustrated in Fig. 2.11 whereas Fig. 2.12 shows an example of machine translation (English to slang English) using RNNs.

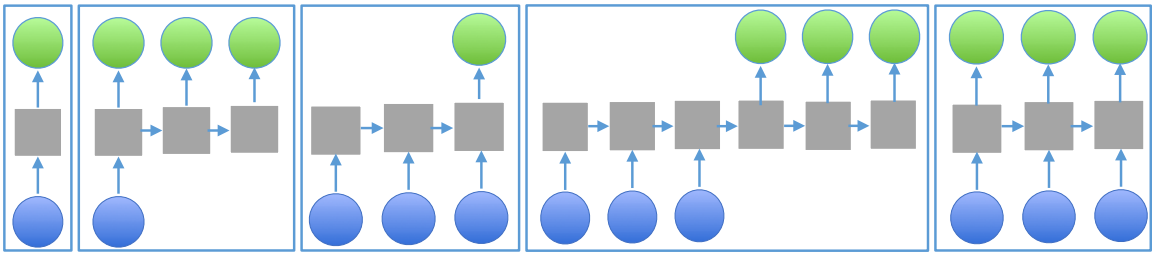


Figure 2.10: From left to right: fixed-sized input to fixed-sized output (e.g. image classification); fixed-sized input (e.g. image) and sequence output (e.g. a set of words in image captioning); sequence input (set of words in a sentence) and sequence output (e.g machine translation); Synced sequence input and sequence output (e.g label every frame in a video)

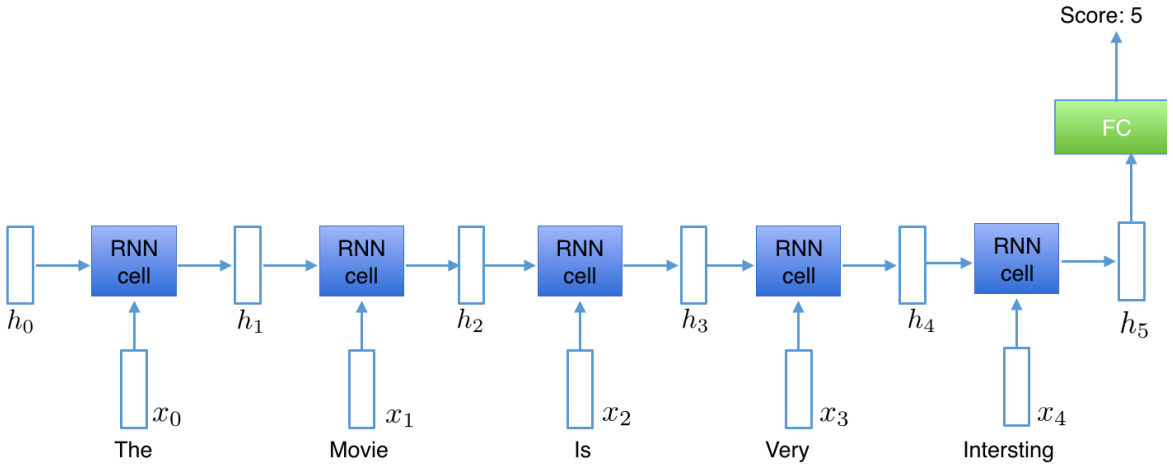


Figure 2.11: An example of RNNs architecture for sentiment analysis with an sentence as sequence input and score as fixed-sized output

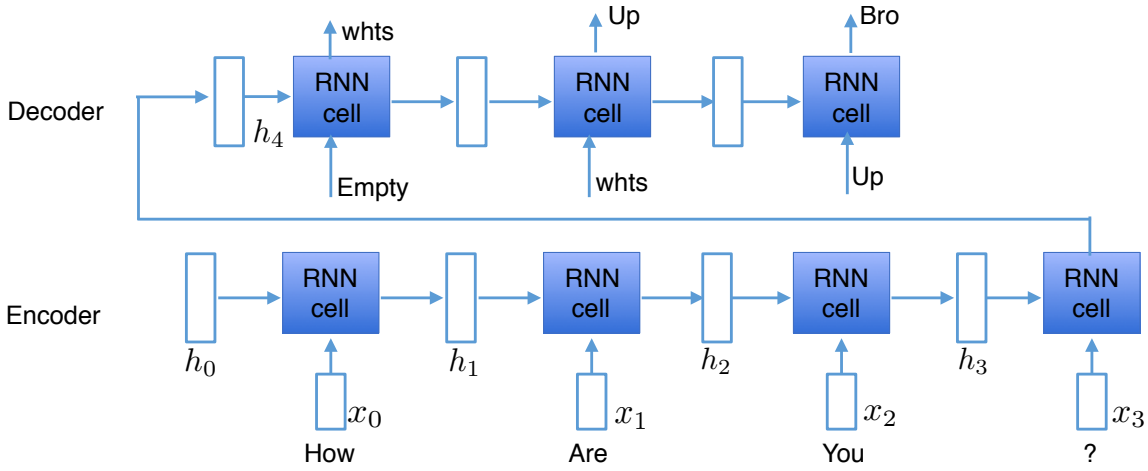


Figure 2.12: An example of RNNs architecture for machine translation with an sentence as sequence input and sentence as sequence output

2.3.2 Long Short Time Memory

The difficulty of training an RNNs to capture long-term dependencies has been studied in [125] and later in [126]. To address the issue of learning long-term dependencies, Hochreiter and Schmidhuber [127] proposed Long Short-Term Memory (LSTM), which is able to maintain a separate memory cell inside it that updates and exposes its content only when deemed necessary. Similar to RNNs, LSTMs are defined under a chain like structure, but

the repeating module has a different structure. Instead of having a single neural network layer (single tanh layer), LSTMS have four neural networks in each layer and they are interacting in a very special way.

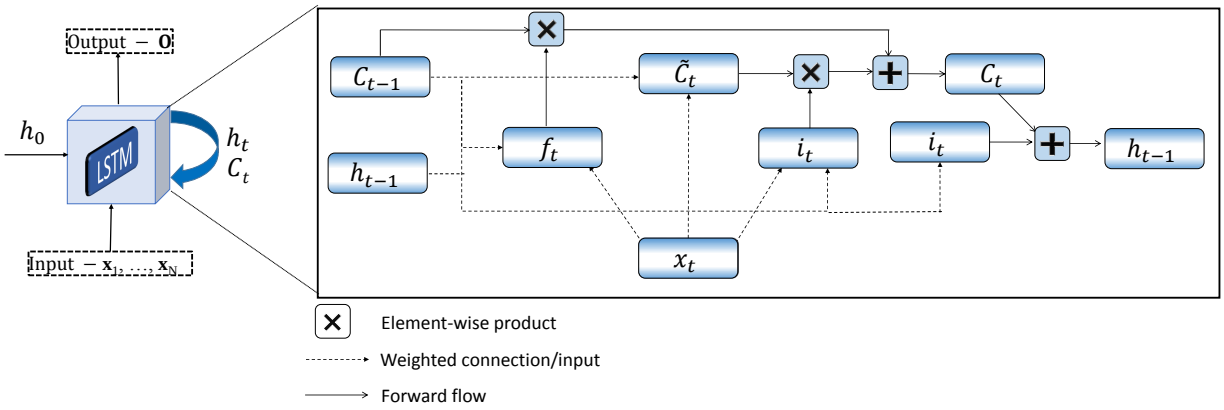


Figure 2.13: The unfolding LSTM in time of the computation involved in its forward computation

The key to LSTMs is the cell state ($\mathbf{C}_t$) which has the ability to remove or add information by optionally letting information through. Gates are composed out of a sigmoid layer and a pointwise multiplication operation. The unfolding LSTM in time of the computation involved in its forward computation is given in Fig. 2.13. This decision what information is thrown away from the cell state is made by a sigmoid layer called the forget gate layer. ($\mathbf{f}_t$).

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \tag{2.24}$$

Two gates: input gate layer(i_t) and candidate state ($\tilde{C}_t$) are used to decide what information is kept in the cell state.

$$\begin{aligned} \mathbf{f}_t &= \sigma(\mathbf{W}_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \\ \tilde{\mathbf{C}}_t &= tanh(\mathbf{W}_C \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C) \end{aligned} \tag{2.25}$$

The old state cell at time $t-1$ ($\mathbf{C}_{t-1}$) is update to the new state cell ($\mathbf{C}_t$) by two terms: forgetting the things we decided to forget earlier ($\mathbf{f}_t * \mathbf{C}_{t-1}$) and how much we decided to update each state value ($\mathbf{i}_t * \tilde{\mathbf{C}}_t$).

$$\mathbf{C}_t = \mathbf{f}_t * \mathbf{C}_{t-1} + \mathbf{i}_t * \tilde{\mathbf{C}}_t \tag{2.26}$$

Finally, a sigmoid gate is applied to decides what parts of the cell state ($\mathbf{C}_t$) is used as output.

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \tag{2.27}$$

The output of the sigmoid gate is then multiplied by a *tanh* to push the values to be between -1 and 1.

$$\mathbf{h}_t = \mathbf{o}_t * tanh(\mathbf{C}_t) \tag{2.28}$$

2.3.3 Gated Recurrent Unit

Recently, a Gated Recurrent Unit (GRU) was proposed by [128] to make each recurrent unit adaptively capture dependencies of different time scales. Like the LSTM unit, the GRU has gating units that modulate the flow of information inside the unit, but without having separate memory cells. That means the GRU fully exposes its memory content each timestep and balances between the previous memory content and the new memory content strictly using leaky integration, even though its adaptive time constant is controlled by the update gate. The state updating process is illustrated as in Fig. 2.14 (right).

Each GRU thus has a reset gate $\mathbf{r}_t$ and an update gate $\mathbf{z}_t$. At time timestep t, the activation state $\mathbf{h}_t$ of a GRU unit is a linear interpolation between the previous memory

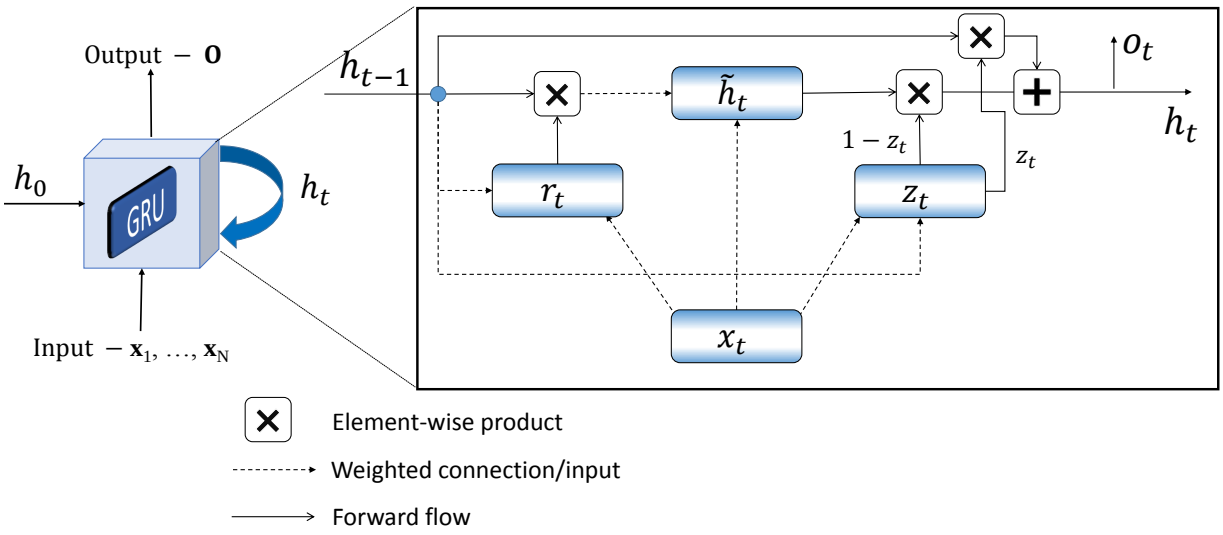


Figure 2.14: The unfolding GRU in time of the computation involved in its forward computation

content $\mathbf{h}_{t-1}$ and the new candidate memory content $\tilde{\mathbf{h}}_t$ as shown in Eq. 2.29.

$$\mathbf{h}_t = (1 - \mathbf{z}_t)\mathbf{h}_{t-1} + \mathbf{z}_t\tilde{\mathbf{h}}_t \tag{2.29}$$

In the above equation, the update gate $\mathbf{z}_t$ controls how much of the previous memory content is to be forgotten and how much of the new memory content is to be added. The update gate $\mathbf{z}_t$ is a linear sum between the previous hidden states $\mathbf{h}_{t-1}$ and the current input $\mathbf{x}_t$ as shown in Eq. 2.30.

$$\mathbf{z}_t = f(\mathbf{U}_z\mathbf{x}_t + \mathbf{W}_z\mathbf{h}_{t-1}) \tag{2.30}$$

where f is usually a logistic sigmoid function or a hyperbolic tangent function. The GRU, however, does not have any mechanism to control the degree to which its state is exposed, but exposes the whole state each time. The new candidate memory content $\tilde{\mathbf{h}}_t$ is computed similarly to that of the traditional recurrent unit as given in Eq. 2.31.

$$\tilde{\mathbf{h}}_t = tanh(\mathbf{U}_h\mathbf{x}_t + \mathbf{W}_h(\mathbf{h}_{t-1} \circ \mathbf{r}_t)) \tag{2.31}$$

where $\circ$ signifies an element-wise multiplication between previous state $\mathbf{h}_{t-1}$ and the reset gate, which is computed similarly to the update gate as in Eq. 2.32.

$$\mathbf{r}_t = f(\mathbf{U}_r \mathbf{x}_t + \mathbf{W}_r \mathbf{h}_{t-1}) \tag{2.32}$$

When $\mathbf{r}_t$ is close to 0 (off), the reset gate effectively makes the unit act as if it is reading the first symbol of an input sequence, allowing it to forget the previously computed state.

The most notable feature shared between these units is the additive component of their update from step $(t-1)^{th}$ to step t^{th}, which RNNs lacks. In RNNs, the activation or content of a unit is always replaced by a new value computed from the current input and the previous hidden state. On the other hand, GRU keeps the existing content and adds the new content on top of it. This addition creates shortcut paths that bypass multiple temporal steps. These shortcuts allow the error to be back-propagated easily without too quickly vanishing as a result of passing through multiple, bounded nonlinearities, thus reducing the difficulty due to vanishing gradients. Furthermore, the addition allows each unit to remember the existence of a specific feature in the input stream for a long series of steps. Any important feature, decided by GRU will not be overwritten but be maintained as it is.

Several variants of RNNs have been later introduced and successfully applied to wide variety of tasks, such as natural language processing [129], [130], [20], [131], speech recognition [132], [133], machine Translation [134], [128], [135], conversation modeling [136], [137], question answering [138], object recognition: [139], [140], [141], image captioning: [142], [143], [144].

2.3.4 Backpropagation Through Time (BPTT)

The feedforward backpropagation algorithm cannot be directly transferred to RNNs because the error backpropagation pass presupposes that the connections between units in-

duce a cycle-free ordering. The solution of the BPTT approach is to "unfold" the RNNs in time, by stacking identical copies of the RNNs, and redirecting connections within the network to obtain connections between subsequent copies. This unrolls to a feedforward network, which can be trained with the backpropagation algorithm.

The pseudo code of BPTT algorithm is given as in Alg.2.

Algorithm 2 BPTT Algorithm

Input: current weight w_{ij}^m, training time series
Output: new weights
Computation steps:
Step 1 (forward): For each sample n, compute $\mathbf{h}_i^{m+1}(n) = f\left(\sum_{1\leq j\leq N^m} w_{ij}^m \mathbf{x}_j(n)\right)$.
Step 2 (backward): For each time n $(1 \leq n \leq T)$, unit activation $\mathbf{h}_i(n)$ and output $o_i(n)$, compute the error propagation $\xi_i(n)$

- For the output layer: $\xi_j(T) = (\mathbf{y}_j(T) - \mathbf{o}_j(T))\frac{\partial f(\mathbf{u})}{\partial \mathbf{u}}|_{\mathbf{u}=\mathbf{z}_j^T}$
- For hidden layers $h_j(T)$ at time T: $\xi_j(T) = \sum_{1\leq i\leq L} \xi_i(T) w_{ij}^{out} \frac{\partial f(\mathbf{u})}{\partial \mathbf{u}}|_{u=\mathbf{z}_j(n)}$
- For the output of earlier layers: $\xi_j(n) = \left[(\mathbf{y}_j(n) - \mathbf{o}_j(n)) + \sum_{1\leq i\leq N} \xi_i(n+1) w_{ij}^{back}\right] \frac{\partial f(\mathbf{u})}{\partial \mathbf{u}}|\mathbf{u} = \mathbf{z}_j(n)$
- For hidden unit $h_j(n)$ of earlier layers: $\xi_i(n) = \left[\sum_{1\leq j\leq N} \xi_j(n+1) w_{ji} + \sum_{1\leq j\leq L} \xi_j(n) w_{ji}^{out}\right] \frac{\partial f(\mathbf{u})}{\partial \mathbf{u}}|_{\mathbf{u}=\mathbf{z}_j(n)}$ where $z_i(n)$ is the potential of the corresponding unit.

  Step 2 (update): The weight are updated as follows:
  - $w_{ij}(new) = w_{ij} + \gamma \sum_{1\leq n\leq T} \xi_i(n) \mathbf{h}_j(n-1)$
  - $w_{ij}^{in}(new) = w_{ij}^{in} + \gamma \sum_{1\leq n\leq T} \xi_i(n) \mathbf{x}_j(n)$
  - For input unit: $w_{ij}^{out}(new) = w_{ij}^{out}(new) + \gamma \sum_{1\leq n\leq T} \xi_i(n) \mathbf{x}_j(n)$
  - For hidden unit: $w_{ij}^{out}(new) = w_{ij}^{out}(new) + \gamma \sum_{1\leq n\leq T} \xi_i(n) \mathbf{h}_j(n)$
  - $w_{ij}^{back}(new) = w_{ij}^{back} + \gamma \sum_{1\leq n\leq T} \xi_i(n) \mathbf{y}_j(n-1)$

2.3.5 Training Recurrent Neural Networks(RNNs)

A generic RNNs, with input $\mathbf{x}_t$ and hidden state $\mathbf{h}_t$ for time step t, is given by:

$$\mathbf{h}_t = (\mathbf{U}\mathbf{x}_t + \mathbf{W}\sigma(h_{t-1})) + \mathbf{b} \tag{2.33}$$

where **U, W** are the input weight and recurrent weight matrices, respectively and **b** is the bias term. $\mathbf{h}_0$ is provided by the user, set to zero or learned, and σ is the activation function. A cost $\xi = \sum_{1 \leq t \leq T} \xi_t$, where ξ_t measures the performance of the network on some given task, where $\xi_t = L(\mathbf{h}_t)$. The parameters of the model are collected in θ for the general case.

One of the common approaches for computing the necessary gradients is backpropagation through time (BPTT), where the recurrent model is represented as a multi-layer and backpropagation is applied on the unrolled model as shown in Fig. 2.15.

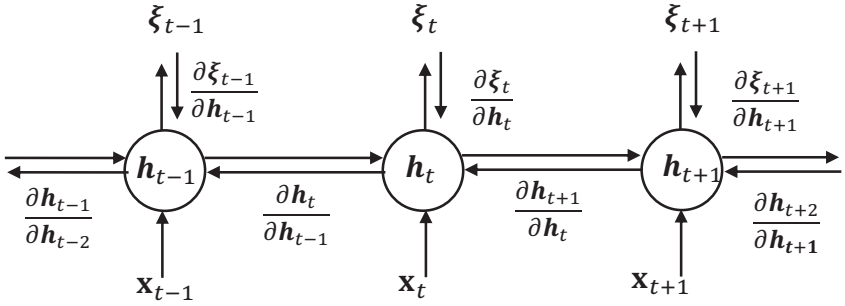


Figure 2.15: Unrolling RNNs in time. $\mathbf{h}_t$ is the hidden state of the network at time t, $\mathbf{x}_t$ is the input of the network at time t and by ξ_t is the error obtained from the output at time t

In this section, we diverge from the classical BPTT equations and rewrite the gradients in order to better highlight the exploding gradients problem.

$$\frac{\partial \xi}{\partial \theta} = \sum_{1 \leq t \leq T} \frac{\partial \xi_t}{\partial \theta} \tag{2.34}$$

$$\frac{\partial \xi_t}{\partial \theta} = \sum_{1 \leq k \leq t} \frac{\partial \xi_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \theta} \tag{2.35}$$

Any gradient component $\frac{\partial \xi_t}{\partial \theta}$ is a sum, whose terms refer to temporal components. Each component measures how θ at step k affects the cost at step $t > k$. $\frac{\partial \mathbf{h}_k}{\partial \theta}$ is immediate partial derivative of the state $\mathbf{h}_k$ with respect to θ.

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{k<i\leq t} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \prod_{k<i\leq t} \mathbf{W}^T diag(\sigma'(\mathbf{h}_{t-1})) \tag{2.36}$$

$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}$ transport the error "in time" from step t back to step k. The Eq. 2.36 provides the form of Jacobian matrix $\frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}}$ for the specific parametrization given in Eq. 2.33. In Eq. 2.36, *diag* converts a vector into a diagonal matrix and and σ' computes the derivative of σ in an element-wise fashion. $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}$ takes the form of a product of $t-k$ Jacobian matrices. Clearly, the product of $t-k$ real numbers (bases) can shrink to zero or explode to infinity. That causes the exploding gradients problem which refers to the large increase in the norm of the gradient during training or vanishing gradients problem which refers to the opposite behavior.

2.3.6 Exploding and Vanishing Gradients in Training RNN

As defined by Bengio, et al.[125], exploding gradients problem refers to the large increase in the norm of the gradient during training whereas vanishing gradients problem refers to the opposite behaviour, when long term components go exponentially fast to norm 0, making it impossible for the model to learn correlation between temporally distant events.

Set σ in Eq. 2.36 to the identity function and assume that t goes to infinity and let $l = t - k$. We have:

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = (\mathbf{W}^T)^l \tag{2.37}$$

Let $\mathbf{W}^T$ have n eigenvalues with order $|\lambda_1| > |\lambda_2| > ... > |\lambda_n|$ and the corresponding eigenvectors $\mathbf{q}_1$, $\mathbf{q}_2$, .., $\mathbf{q}_n$ which form a vector basic. We can now write the row vector $\frac{\xi_t}{\mathbf{h}_t}$ into this basis.

$$\frac{\xi_t}{\mathbf{h}_t} = \sum_{i=1}^{n} c_i \mathbf{q}_i^T \tag{2.38}$$

Furthermore, we have:

$$\mathbf{q}_i^T (\mathbf{W}^T)^l = \lambda_i^l \mathbf{q}_i^T \tag{2.39}$$

Thus,

$$\frac{\xi_t}{\mathbf{h}_t}\frac{\mathbf{h}_t}{\mathbf{h}_k} = c_j\lambda_j^l\mathbf{q}_j^T + \lambda_i^l\sum_{i=j+1}^{n} c_i\left(\frac{\lambda_i}{\lambda_j}\right)^l = c_j\lambda_j^l\mathbf{q}_j^T \tag{2.40}$$

j is the index such that $c_j \neq 0$ and $c'_j = 0$ for any $j' \leq j$.

Due to $\frac{|\lambda_i|}{|\lambda_j|} < 1$ if$(i > j)$, $lim_{l\to\infty}\left(\frac{|\lambda_i|}{|\lambda_j|}\right)^l = 0$. If $|\lambda_j| > 1$ then $\frac{\mathbf{h}_t}{\mathbf{h}_k}$ grows exponentially fast with l along the direction $\mathbf{q}_j$ which cause the exploding gradient problem. In the other hand, when $|\lambda_j| < 1$, the problem of vanishing gradient is leaded.

It is sufficient for the largest eigenvalue λ_1 of the recurrent weight matrix to be smaller than 1 for long term components to vanish (as $t \to \infty$) and necessary for it to be larger than 1 for gradients to explode.

The problem can be generalized for nonlinear function σ where the absolute values of σ' is bounded by a value γ_h and the abbsolute values of W is bounded by γ_W therefore, $||diag(\sigma'(\mathbf{h}_k))|| < \gamma_h$ and $||W|| < \gamma_W$. As given in Eq. 2.36, the Jacobian matrix $\frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k}$ is given by $\mathbf{W}^T diag(\sigma'(\mathbf{h}_k))$. Thus,

$$||\frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k}|| \leq ||\mathbf{W}^T||\ ||diag(\sigma'(\mathbf{h}_k))|| < \gamma_W\gamma_h \tag{2.41}$$

Therefore,

$$\frac{h_t}{h_k} = \prod_{i=k+1}^{t}\frac{h_i}{h_{i-1}} < (\gamma_W\gamma_h)^{(t-k)} \tag{2.42}$$

The exponential term $(\gamma_W\gamma_h)^{t-k}$ can easily become a very small or large number when $\gamma_W\gamma_h$ is much smaller or larger than 1 and $t-k$ is sufficiently large. During the training, once the gradient value grows extremely large, it causes an overflow (i.e. NaN) which is easily detectable at runtime; this issue is called the exploding gradient problem. When the gradient value goes to zero, however, it can go undetected while drastically reducing the learning quality of the model for far-away words in the corpus; this issue is called the vanishing gradient problem.

2.4 Active Contour

The key ideas behind the Active Contour (AC) for image segmentation is to start with an initial guess boundary represented in a form of closed curves i.e. contours C. The curve is then iteratively modified by applying shrink or expansion operations and moved by image-driven forces to the boundaries of the desired objects. The entire process is called contour evolution, denoted as $\frac{\partial C}{\partial t}$ where C is the object boundary (contour).

There are two main approaches in active contours: snakes and level sets. Snakes explicitly move predefined snake points based on an energy minimization scheme, while level set approaches move contours implicitly as a particular level of a function.

2.4.1 Classic Snakes

The first model of active contour, named classic snakes or explicit active contour, was proposed by Kass et al.[145]. In this approach, a contour parameterized by arc length s as $C(s)(x(s), y(s)) : 0 \leq s \leq 1$. An energy function $E(C)$ can be defined on the contour such as:

$$E(C) = \int_0^1 E_{int} + E_{ext} \tag{2.43}$$

where E_{int} and E_{ext} are the internal energy and external energy functions, respectively. The internal energy function determines the regularity, i.e. smooth shape, of the contour.

$$E_{int}(C(s)) = \alpha|C'(s)|^2 + \beta|C''(s)|^2 \tag{2.44}$$

Here α controls the tension of the contour, and β controls the rigidity of the contour while $C^{'}s$) makes the spline act like a membrane (like elasticity) and $C^{''}(s)$ makes it act like a thin-plate (like rigidity). The external energy term determines the criteria of contour

evolution depending on the image $\mathbf{I}(x, y)$, and can be defined as in Eq. 2.45.

$$E_{image} = w_{line}E_{line} + w_{edge}E_{edge} + w_{term}E_{term} \tag{2.45}$$

$$E_{line} = \mathbf{I}(x, y) \tag{2.46}$$

$$E_{edge} = -| \bigtriangledown \mathbf{I}(x, y)|^2 \tag{2.47}$$

$$E_{term} = \frac{\left(\phi_x^2\phi_{yy} - 2\phi_x\phi_y\phi_{xy} + \phi_y^2\phi_{xx}\right)}{| \bigtriangledown \phi|^3} \tag{2.48}$$

The first term is given in Eq. 2.46 and it depends on the sign of w_{line} which guides the snake towards the lightest or darkest nearby contour. The second term defined in Eq. 2.47 attracts the snake to large intensity gradients. The third term E_{term} attracts the snake toward termination of line segments and corners. E_{term} is defined in Eq. 2.48 using curvature of level lines. Fig. 2.16 shows an example of snake with 70 snakes points forming a contour around the moth. Each point moves towards the optimum coordinates, where the energy function converges to the minimum.

Figure 2.16: An example of classic snakes

The snake provide an accurate location of the edges only if the initial contour is given sufficiently near the desired edges. Moreover, snake cannot detect more than one boundary simultaneously because the snakes maintain the same topology during the evolution stage.

2.4.2 Level Set Method

Level set (LS) based or implicit active contour models have provided more flexibility and convenience for the implementation of active contours, thus, they have been used in a variety of image processing and computer vision tasks. The basic idea of the implicit active contour is to represent the initial curve C implicitly within a higher dimensional function, called the level set function $\phi(x, y) : \Omega \rightarrow R$, such as:

$$C = (x, y) : \phi(x, y) = 0, \forall (x, y) \in \Omega \tag{2.49}$$

where Ω denotes the entire image plane. Fig. 2.17 (left) shows the evolution of level set function $\phi(x, y)$, and Fig. 2.17 (right) shows the propagation of the corresponding contours C.

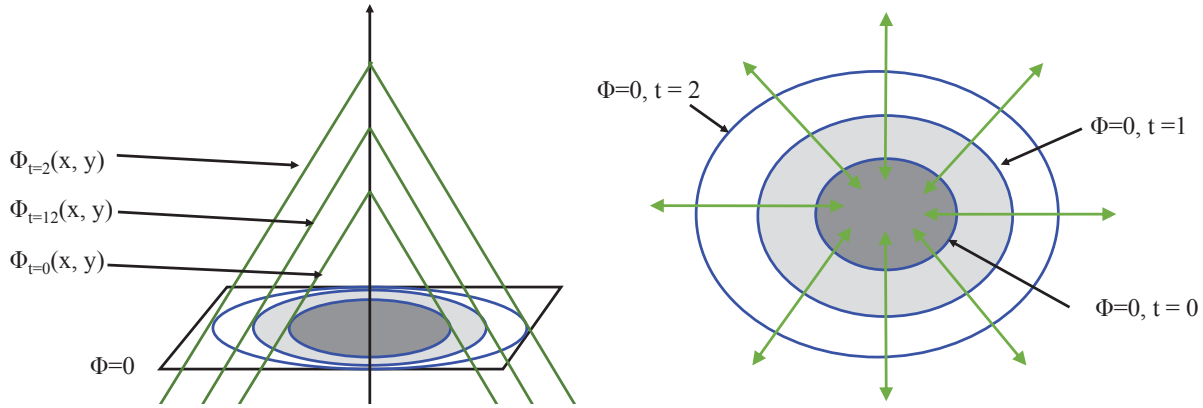


Figure 2.17: Level set evolution and the corresponding contour propagation: (a) topological view of level set $\phi(x, y)$ evolution, (b) the changes on the zero level set $C = (x, y) : \phi(x, y) = 0$

The evolution of the contour is equivalent to the evolution of the level set function, i.e. $\frac{\partial C}{\partial t} = \frac{\partial \phi(x,y)}{\partial t}$. One of the advantages of using the zero level set is that a contour can be

defined as the border between a positive area and a negative area, so the contours can be identified by signed distance function as follows:

$$\phi(x) = \begin{cases} d(x, C)) & \text{if } x \text{ is inside } C \\ 0 & \text{if } x \text{ is on } C \\ -d(x, C)) & \text{if } x \text{ is outside } C \end{cases} \tag{2.50}$$

where $d(x, C)$ denotes the distance from an arbitrary position to the curve.

The level set evolution can be written in the form as follows:

$$\frac{\partial \phi}{\partial t} + F\,|\nabla \phi| = 0 \tag{2.51}$$

where F is a speed function. In some particular cases, F is defined as mean curvature, $F = div\left(\frac{\nabla \phi}{||\nabla \phi||}\right)$.

An outstanding characteristic of level set methods is that contours can split or merge as the topology of the level set function changes. Therefore, level set methods can detect more than one boundary simultaneously, and multiple initial contours can be placed as shown in Fig. 2.18.

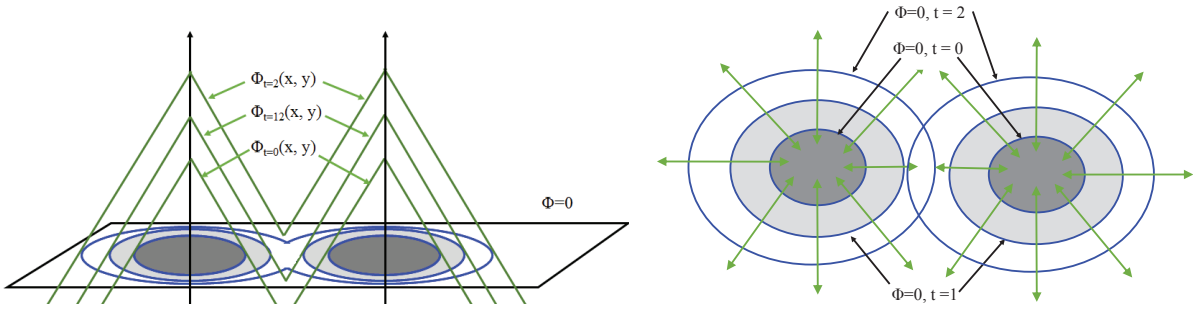


Figure 2.18: Topology of level set function changes in the evolution and the propagation of corresponding contours: (a) the topological view of level set $\phi(x, y)$ evolution, (b) the changes on the zero level set $C : \phi(x, y) = 0$

The computation is performed on the same dimension as the image plane Ω, therefore,

the computational cost of level set methods is high and the the convergence speed is quite slow.

The process of evolving of the curve C (contour evolution), denoted as $\frac{\partial C}{\partial t}$ and illustrate in Fig. 2.19. In this example, iterative stages of active contour evolution for the segmentation of objects (cells in this case) in images is shown in red. The underlying image are the level-set function is typically defined over the entire 2D image domain.

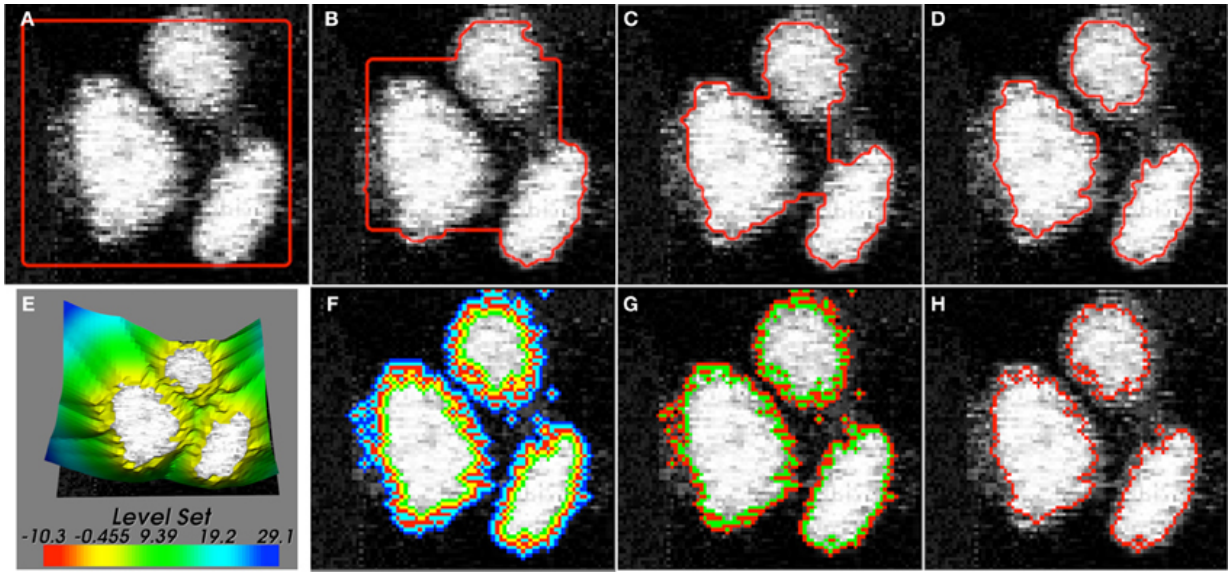


Figure 2.19: Example of active contour evolution for image segmentation

2.4.3 Edge-based Active Contours

Edge-based active contours are closely related to the edge-based segmentation. Most edge-based AC models consist of two components: regularity and edge detection. The first part determines the shape of contours whereas the second one attracts the contour towards the edges.

Geodesic active contour (GAC) model, one of the most popular methods among the edge-based active contour models, was proposed by Caselles et al.[146]. Given an initial

curve C_0, the curve evolution is given as:

$$\frac{\partial \phi(x,y)}{\partial t} = g(\mathbf{I}(x,y))(\kappa(\phi(x,y)) + F)| \bigtriangledown \phi(x,y)| \tag{2.52}$$

where g denotes a stopping function which is based on an edge indicator scalar function, i.e. $g(\mathbf{I}(x,y)) = \frac{1}{1+|\bigtriangledown \phi(x,y)|}$. The curvature κ maintains the regularity of the contours whereas the constant speed F keeps the contour evolving. In GAC model, the contours move in the normal direction with a speed of $\kappa(\phi(x,y)) + F$ and therefore stops on the edges.

It is easy to see that edge-based active contour models inherit some disadvantages of the edge-based segmentation methods, such as a reliance on the image gradient, omission of blurry boundaries and a sensitivity to local minima and noise. Moreover, edge-based active contour models have a few of their own disadvantages (compared to the region-based active contour models which will be discussed in the next section) that are due to the structure of the speed functions and the stopping functions. It is easy to see that the edge-based active contour models evolve the contour towards only one direction, either inside or outside because of the constant speed F. Thus, an initial contour should be placed completely inside or outside the ROI, so some level of a prior knowledge is still required. Later, Paragios [147] proposed Gradient vector flow fast geodesic active contour by replacing the edge detection with a gradient vector field.

2.4.4 Region-based Active Contours (Chan-Vese)

Under the scenarios of image segmentation, we consider an image in 2D space, Ω. The Level Set is to find the boundary C of an open set $\omega \in \Omega$, which is defined as: $C = \partial\omega$. In

VLS framework, the boundary C can be represented by the zero level set ϕ as follows:

$$\forall(x,y) \in \Omega \begin{cases} C = & \{(x,y) : \phi(x,y) = 0\} \\ inside(C)) & \{(x,y) : \phi(x,y) > 0\} \\ output(C)) & \{(x,y) : \phi(x,y) < 0\} \end{cases} \tag{2.53}$$

For image segmentation, Ω denotes the entire domain of an image $\mathbf{I}$. The zero LS function ϕ divides the region Ω into two regions: region inside ω (foreground), denoted as inside(C) and region outside ω (background) denoted as outside(C). The length of the contour C and the area inside the contour C are defined as follows:

$$\begin{aligned} Length(C) &= \int_\Omega |\nabla H(\phi(x,y))| dxdy = \int_\Omega \delta(\phi(x,y))|\nabla\phi(x,y)|dxdy \\ Area(C) &= \int_\Omega H(\phi(x,y))dxdy \end{aligned} \tag{2.54}$$

where, $H_\epsilon(\cdot)$ is a Heaviside function.

Typically, the LS-based segmentation methods start with an initial level set ϕ_0 and an given image $\mathbf{I}$. The LS updating process is performed via gradient descent by minimizing an energy function which defined based on the difference of image features, such as color and texture, between foreground and background. The fitting term in LS model is defined by the inside contour energy (E_1) and outside contour energy (E_2).

$$E = E_1 + E_2 = \int_{insideC} (I_{(x,y)} - c_1)^2 dxdy + \int_{outsideC} (I_{(x,y)} - c_2)^2 dxdy \tag{2.55}$$

where c_1 and c_2 are the average intensity inside and outside the contour C, respectively. Fig. 2.20 gives an example of all possible cases of the curve. It is easy to see that the energy E is minimized when the contour is right on the object boundary.

Most region-based active contour models consist of two components: regularity and energy minimization. The first part is to determine the smooth shape of contours whereas

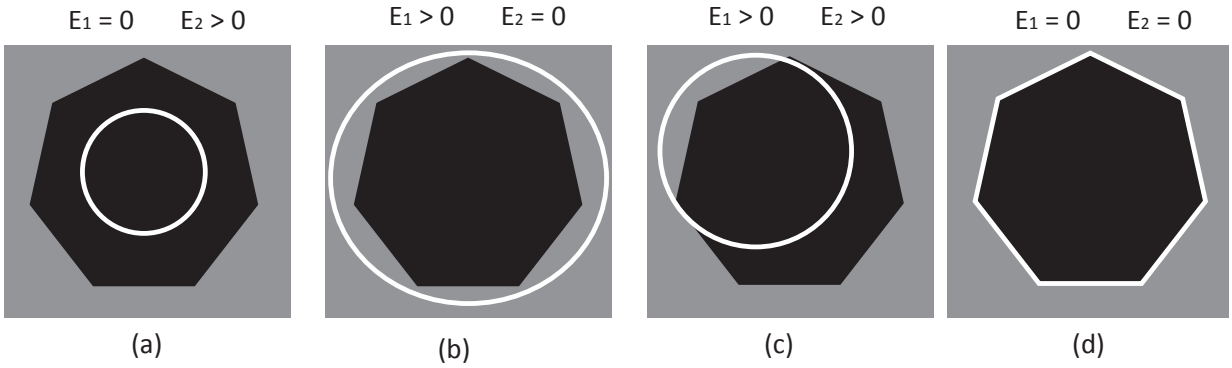


Figure 2.20: An illustration of energy inside the contour C (E_1)and outside and the contour C (E_2)

the second part searches for uniformity of a desired feature within a subset.

One of the most popular region based active contour models is proposed by Chan-Vese (CV) [62]. In this model the boundaries are not defined by gradients and the curve evolution is based on the general Mumford-Shah (MS) [148] formulation of image segmentation as shown in Eq. 2.56.

$$E = \int_{\Omega} |\mathbf{I} - u|^2 dxdy + \int_{\Omega/C} |\nabla u|^2 dxdy + \nu Length(C) \tag{2.56}$$

CV's model is an alternative form of MS's model which restricts the solution to piecewise constant intensities and it has successfully segmented an image into two regions, each having a distinct mean of pixel intensity by minimizing the following energy functional.

$$\begin{aligned} E(c_1, c_2, \phi) &= \mu \text{Area}(\omega_1) + \nu \text{Length}(C) \\ +\lambda_1 \int_{\omega_1} |\mathbf{I}(x,y) - c_1|^2 dxdy &+ \lambda_2 \int_{\omega_2} |\mathbf{I}(x,y) - c_2|^2 dxdy \end{aligned} \tag{2.57}$$

where c_1 and c_2 are two constants. The parameters $\mu, \nu, \lambda_1, \lambda_2$ are positive parameters and usually fixing $\lambda_1 = \lambda_2 = 1$ and $\mu = 0$. Thus, we can ignore the first term in Eq. 2.57.

Thus the energy functional is rewritten as follows:

$$\begin{aligned} E(c_1, c_2, \phi) = \mu \int_\Omega H(\phi(x,y))dxdy + \nu \int_\Omega \delta(\phi(x,y))|\nabla\phi(x,y)|dxdy \\ + \lambda_1 \int_{\omega_1} |\mathbf{I}(x,y) - c_1|^2 dxdy + \lambda_2 \int_{\omega_2} |\mathbf{I}(x,y) - c_2|^2 dxdy \end{aligned} \tag{2.58}$$

For numerical approximations, the δ function needs a regularizing term for smoothing. In most cases, the Heaviside function H and Dirac delta function δ are defined as in (2.59) and (2.60), respectively.

$$H_\epsilon(x) = \frac{1}{2}\left(1 + \frac{2}{\pi}\arctan\left(\frac{x}{\epsilon}\right)\right) \tag{2.59}$$

$$\delta_\epsilon(x) = H'(x) = \frac{1}{\pi}\frac{\epsilon}{\epsilon^2 + x^2} \tag{2.60}$$

As $\epsilon \to 0$, $\delta_\epsilon \to \delta$, and $H_\epsilon \to H$. Using Heaviside function H, the Eq. 2.58 becomes Eq. 2.61.

$$\begin{aligned} E(c_1, c_2, \phi) = \mu \int_\Omega H(\phi(x,y))dxdy + \nu \int_\Omega \delta(\phi(x,y))|\nabla\phi(x,y)|dxdy \\ + \lambda_1 \int_\Omega |\mathbf{I}(x,y) - c_1|^2 H(\phi(x,y))dxdy + \lambda_2 \int_\Omega |\mathbf{I}(x,y) - c_2|^2 (1 - H(\phi(x,y)))dxdy \end{aligned} \tag{2.61}$$

In the implementation, they choose $\epsilon = 1$. For fixed c_1 and c_2, gradient descent equation with respect to ϕ is:

$$\frac{\partial\phi(x,y)}{\partial t} = \delta_\epsilon(\phi(x,y)[\nu\kappa(\phi(x,y) - \mu - \lambda_1((\mathbf{I}(x,y) - c_1)^2 + \lambda_2((\mathbf{I}(x,y) - c_2)^2] \tag{2.62}$$

where δ_ϵ is a regularized form of Dirac delta function and c_1, c_2 are the mean of inside the contour ω_{in} and the mean of the outside of the contour ω_{out}, respectively. The curvature κ is given by:

$$\kappa(\phi(x,y)) = -div\left(\frac{\triangle\phi}{|\triangle\phi|}\right) = -\frac{\phi_{xx}\phi_y^2 - 2\phi_x\phi_y\phi_{xy} + \phi_{yy}\phi_x^2}{\left(\phi_x^2 + \phi_y^2\right)^{1.5}} \tag{2.63}$$

where $\partial_x\varphi_t, \partial_y\varphi_t$ and $\partial_{xx}\varphi_t, \partial_{yy}\varphi_t$ are the first and second derivatives of φ_t with respect to x and y directions. For fixed ϕ, gradient descent equation with respect to c_1 and c_2 are:

$$\begin{aligned} c_1 &= \frac{\sum_{x,y} \mathbf{I}(x,y) H(\phi(x,y))}{\sum_{x,y} H(\phi(x,y))} \\ c_2 &= \frac{\sum_{x,y} \mathbf{I}(x,y)(1 - H(\phi(x,y)))}{\sum_{x,y}(1 - H(\phi(x,y)))} \end{aligned} \tag{2.64}$$

herein, we use notation φ_t to indicate φ at the iteration t^{th} in order to distinguish the φ at different iterations. Under this formulation, the curve evolution is shown as a time series process which helps to give better visualization of reformulating LS. From this point, we redefine the curve updating in a time series form for the LS function φ_t as in Eq. (2.65).

$$\varphi_{t+1} = \varphi_t + \eta \frac{\partial \varphi_t}{\partial t} \tag{2.65}$$

The LS at time $t+1$ depends on the previous LS at time t and the curve evolution $\frac{\partial \varphi_t}{\partial t}$ with a learning rate η.

2.4.5 State-of-the-art Level Set Methods and Their Limitations

Li et al. [149] solved the problem of segmenting images with intensity inhomogeneity by using a local binary fitting energy. By minimizing the unbiased pixel-wise average misclassification probability, Wu et al. [150] formulated an active contour to segment an image without any prior information about the intensity distribution of regions. By realizing curve evolution via simple operations between two linked lists, Shi and Karl [151] achieved a fast level set algorithm for real-time tracking. Also, they incorporated the smoothness regularization with the use of a Gaussian filtering process and proposed the two-cycle fast (TCF) algorithm to speed up the level set evolution.

To overcome the limitation of CLS being binary-phase segmentation, Samson et al. [12] associated a LS function with each image region, and evolves these functions in a coupled

manner. Later, Brox & Weickert [13] performed hierarchical segmentation by iteratively splitting previously obtained regions using the CLS. To deal with reinitialization, DRLSE [15] is proposed with a new variational level set formulation in which the regularity of the LS function is intrinsically maintained during the LS evolution. Different from the aforementioned methods that work on global, Li et al. [16] focused on intensity inhomogeneity which often occurs in real-world images. In their approach, they derived a local intensity clustering property and defined a local clustering criterion function in a neighborhood of each point. Lucas [17] suggested using a single LS function to perform the LS evolution for multi-region segmentation. It requires managing multiple auxiliary LS functions when evolving the contour, so that no gaps/overlaps are created. Bae & Tai [14] proposed to divide an image into multiple regions by a single, piecewise constant LS function, obtained using either augmented Lagrangian optimization, or graph-cuts. Later, to deal with local minima, LSE [21] uses some form of local processing to tackle intra-region inhomogeneity, which makes such methods susceptible to local minima. Recently, Dubrovina et al. [152] have developed a multi-region segmentation with single LS function. However, Dubrobina et al's approach was developed for contour detection and needs good initialization, namely, it requires specify more initial regions than it is expected to be in the final segmentation. Furthermore, the algorithm requires that initial contours cover the image densely, specifically the initial contour has to pass through all different regions. In addition to multi region segmentation problem, optimization [19, 23, 131], and shape prior [153] have also been considered.

Generally, the level set model minimizes a certain energy function via gradient descent [154], making the segmentation results prone to getting stuck in local minima. To conquer this problem, Chan et al. [155] restated the traditional Mumford-Shah image segmentation model [148] as a convex minimization problem to obtain the global minimum . The above methods have obtained promising performance in segmenting high quality images. However, when attempts are made to segment images with heavy noise, this leads to poor

segmentation results. Existing methods assume that pixels in each region are independent when calculating the energy function. This underlying assumption makes the contour motion sensitive to noise. In addition, the implementation of level set methods is complex and time consuming, which limits their application to large scale image databases. To maintain numerical stability, the numerical scheme used in level set methods, such as the upwind scheme or finite difference scheme, must satisfy the Courant-Friedrichs-Lewy (CFL) condition [156], which limits the length of the time step in each iteration and wastes time.

Some limitations of variational level set approaches are observed as follows:

- They are unsupervised approaches and therefore require no learning properties from the training data. Thus, they have difficulty in dealing with noise and occlusions.
- There are many parameters which are chosen by empirical results.
- The are build off of gradient descent to implement the non-convex energy minimization and can get stuck in undesired local minima and thereby lead to erroneous segmentations.
- Most of the level set based approaches are not able to robustly segment images in the wild.
- They often give unpredictable segmentation results due to unsupervised behaviors.
- The accuracy of segmenting results strongly depends on the number of iterations which is usually set as a big number.

2.5 Common Deep Network Architectures

Many semantic segmentation approaches including semantic instance segmentation have made use of some popular deep networks architectures, i.e. LeNet AlexNet, VGG-16, GoogLeNet, and ResNet.

2.5.1 LeNet

The first successful applications of Convolutional Networks is developed by Yann LeCun [64]. Of these, the best known is the LeNet architecture. LeNet applies several banks to recognise hand-written numbers on checks (cheques) digitized in 32x32 pixel images. The ability to process higher resolution images requires larger and more convolutional layers, so this technique is constrained by the availability of computing resources. The LeNet architecture is given in Fig. 2.21.

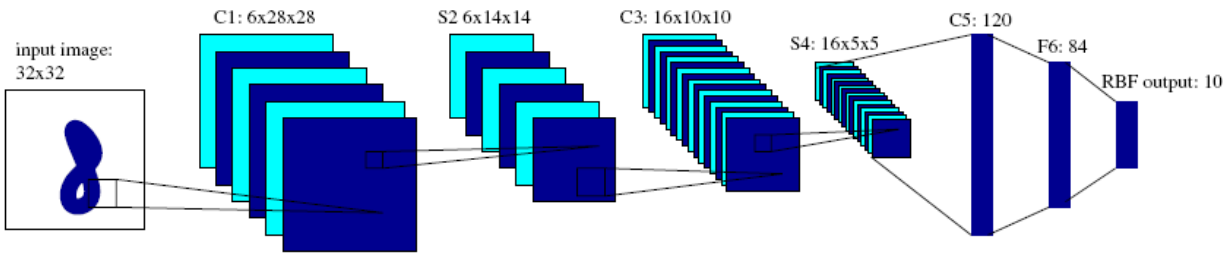


Figure 2.21: Architecture of LeNet

2.5.2 AlexNet

Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton create a large, deep convolutional neural network, called AlexNet [10] that is used to win the 2012 ILSVRC (ImageNet Large-Scale Visual Recognition Challenge). In 2012, AlexNet significantly outperforms all the prior competitors and wins the challenge by reducing the top-5 error to 15.3%. The second place top-5 error rate, which is not a CNN variation, is around 26.2%. Since then, CNNs became the standard direction of evolution to improve performance. The network has a very similar architecture as LeNet but is deeper, with more filters per layer, and with stacked convolutional layers. AlexNet is composed of 5 conv layers, max-pooling layers, dropout layers, and 3 fully connected layers as given in Fig. 2.22. The network they designed is used for classification with 1000 possible categories. AlexNet network is designed with following key contributions: (a) Train the network on ImageNet data,

which contains over 15 million annotated images from a total of over 22,000 categories. (b) Used ReLU for the nonlinearity functions. (c) Used data augmentation techniques that consisted of image translations, horizontal reflections, and patch extractions. (d) Implemented dropout layers in order to combat the problem of overfitting to the training data. (e) Trained the model using batch stochastic gradient descent, with specific values for momentum and weight decay. (f) Trained on two Nvidia Geforce GTX 580 GPUs for five to six days.

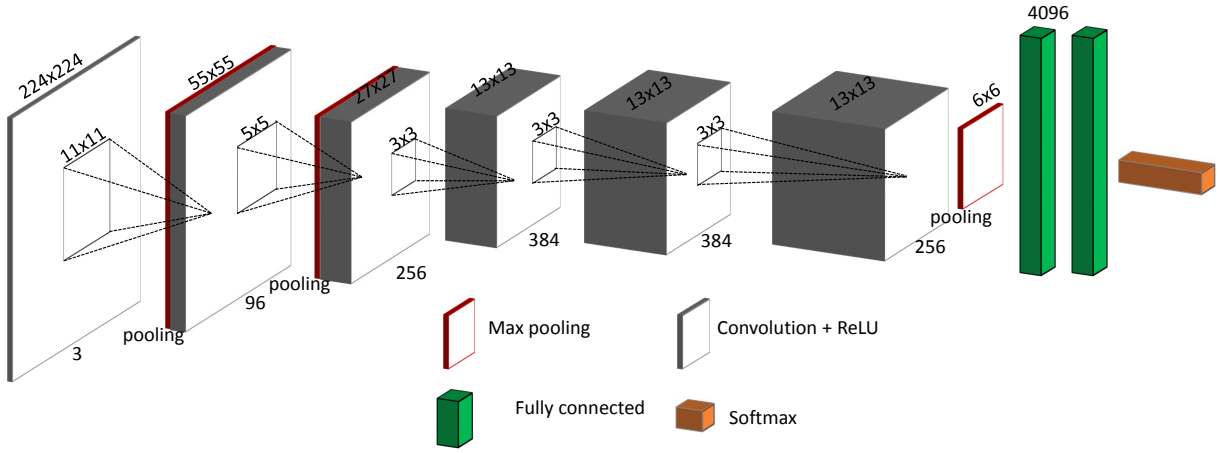


Figure 2.22: Architecture of AlexNet

2.5.3 ZFNet

The ILSVRC 2013 winner is a Convolutional Network from Matthew Zeiler and Rob Fergus, which achieves a top-5 error rate of 11.2% . It is known as the ZFNet [66]. It is an improvement of AlexNet by tweaking the hyperparameters, in particular by expanding the size of the middle convolutional layers and making the stride and filter size on the first layer smaller as shown in Fig. 2.23. In this work, the authors explain a lot of the intuition behind ConvNets and show how to visualize the filters and weights correctly. ZFNet can be summarized as follows: (a) similar architecture to AlexNet, except for a few minor modifications. (b) AlexNet is trained on 15 million images whereas ZFNet is trained on

only 1.3 million images (c) Instead of using 11x11 sized filters in the first layer, ZFNet used 7x7 sized filter and a decreased stride value. (d) The number of filters used is raised when the network grows. (e)Used ReLUs for their activation functions, cross-entropy loss for the error function, and trained using batch stochastic gradient descent. (f) Trained on a GTX 580 GPU for twelve days. (g) Developed a visualization technique named Deconvolutional Network.

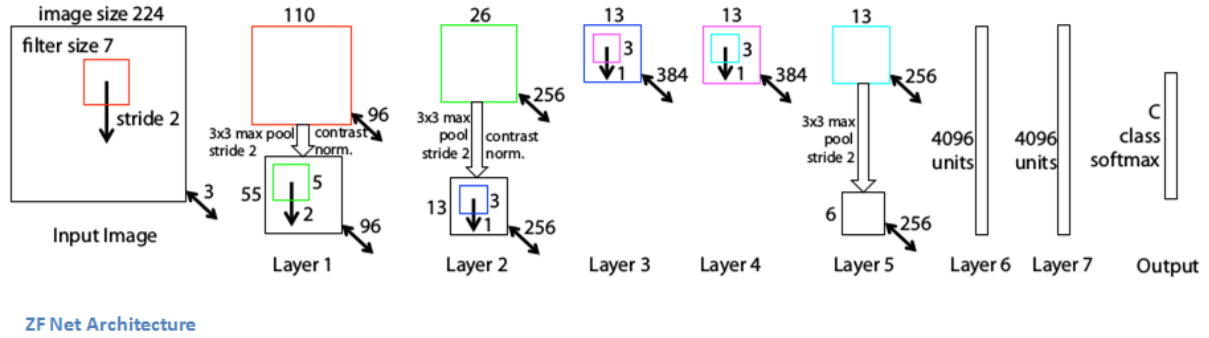


Figure 2.23: Architecture of ZFNet

2.5.4 VGG-16

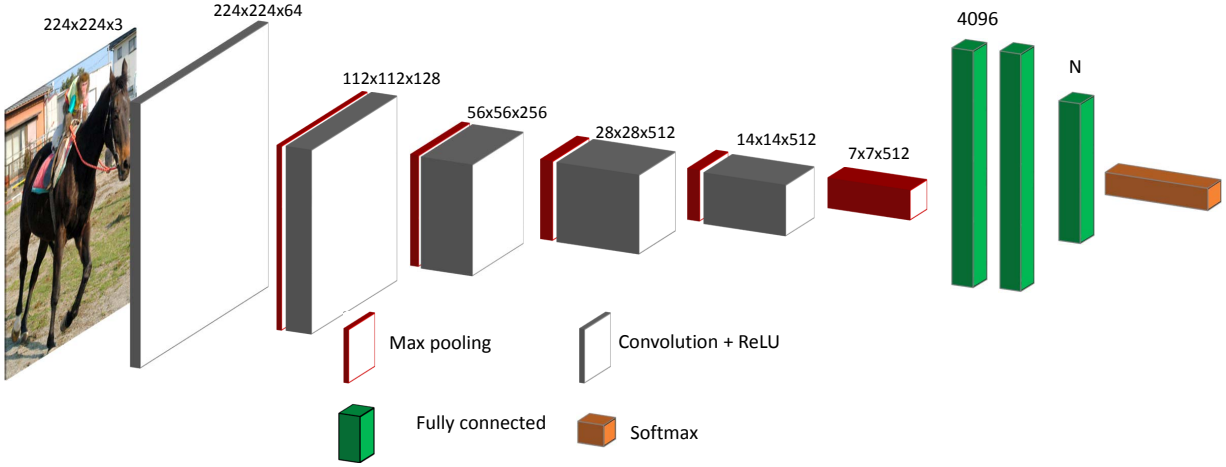


Figure 2.24: Architecture of VGG

VGG [157] model is developed by Simonyan and Zisserman in 2014 and archives 7.3% error rate on ILSVRC 2014. VGG-16 is not a winner on ILSVRC 2014 but it quickly

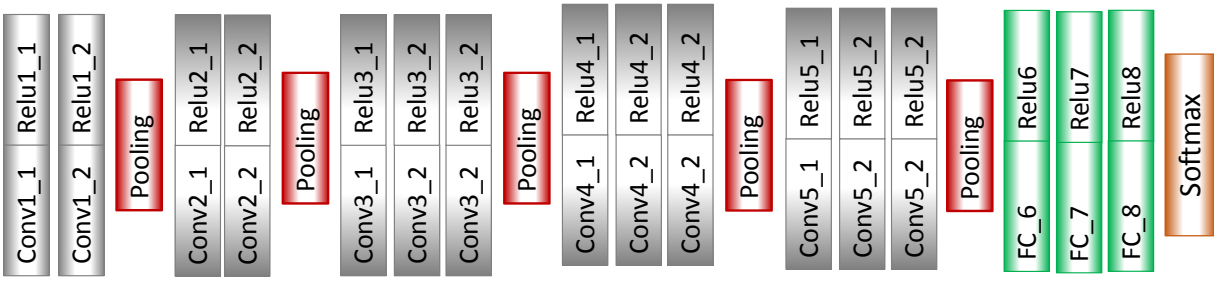


Figure 2.25: Simple visualization of 16 layers in VGG

became common because of simplicity and depth with 16 layer CNN that strictly used 3x3 filters with stride and pad of 1, along with 2x2 maxpooling layers with stride 2 as given in Fig. 2.24. It is currently one of the most preferred choice in the community for extracting features from images. However, VGGNet consists of 140 million parameters, which can be a bit challenging to handle. The simple visualization with 16 layers of VGG-16 is given in Fig. 2.25. The main contribution of VGG-16 can be summarized as follows: (a) use only 3x3 sized filters instead of 11x11 in AlexNet and 7x7 in ZFNet. (b) As a result of the conv and pool layers, the spatial size of the input volumes at each layer decrease. (c) The depth of the volumes increase due to the increased number of filters. (d) The number of filters doubles after each maxpool layer. (e) Used ReLU layers after each conv layer and trained with batch gradient descent.(f) Worked well on both image classification and localization tasks. (g) Built model with the Caffe toolbox. (h) Used scale jittering as one data augmentation technique during training. (i)Trained on 4 Nvidia Titan Black GPUs for two to three weeks.

2.5.5 GoogLeNet/Inception

GoogLeNet [158] is a 22 layer CNN and is the winner of ILSVRC 2014 with a top 5 error rate of 6.7%. This is one of the first CNN architectures that really strayed from the general approach of simply stacking conv and pooling layers on top of each other in a sequential structure as shown in Fig. 2.26. The network uses a CNN inspired by LeNet

but implements a novel element which is dubbed an inception module. The sequence of inception module in GoogleNet is illustrated more detailed in Fig. 2.27 which explains the first inception module in Fig. 2.26. GoogleNet can be summarized as follows: (a) It is a very deep networh with over 100 layer in 9 inception modules. (b) No use of fully connected layers help to save a huge number of parameters. (c) Uses 15x fewer parameters than AlexNet, reduced the number of parameters from 60 million (AlexNet) to 4 million. (d) During testing, multiple crops of the same image are created, fed into the network, and the softmax probabilities are averaged to give us the final solution. (e) Utilize R-CNN to detect object. (f) Trained on a few high-end GPUs within a week.

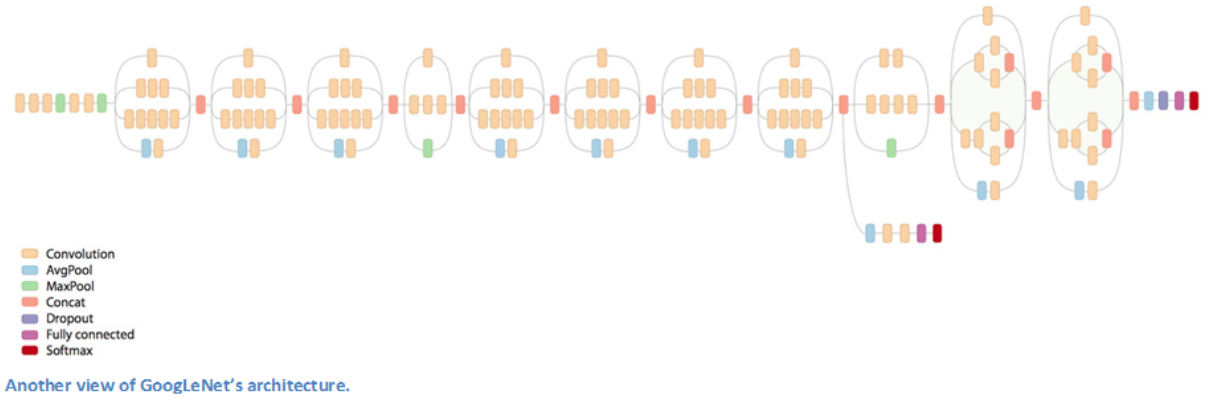


Figure 2.26: Architecture of GoogLeNet

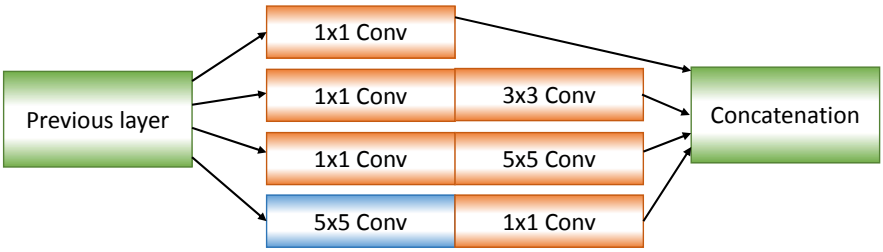


Figure 2.27: An explaination of inception module

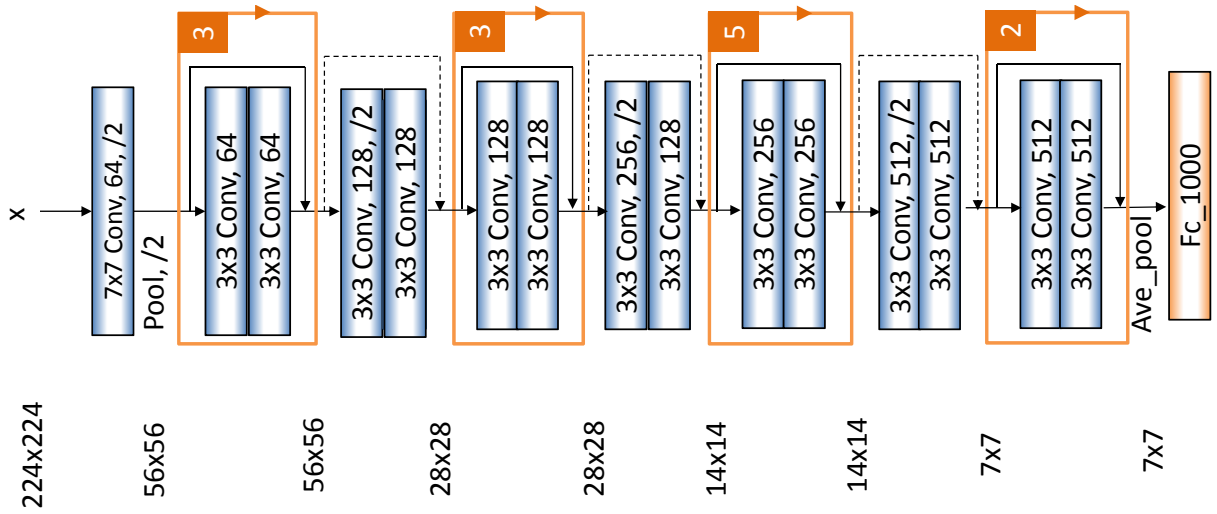


Figure 2.28: Architecture of ResNet-50

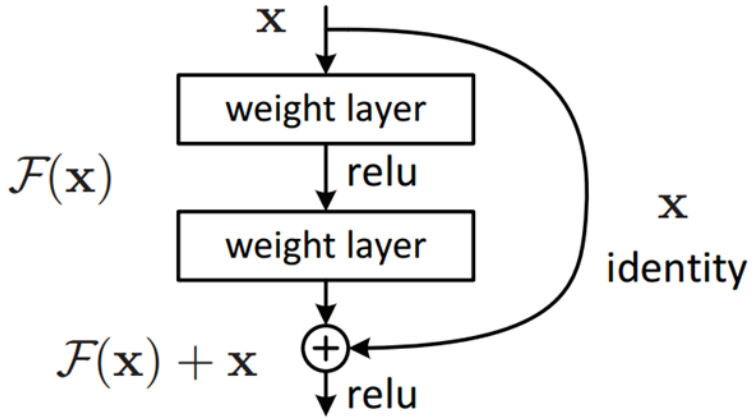


Figure 2.29: Explaination of a residual block

2.5.6 ResNet

Residual Neural Networks (ResNet) [96] by Kaiming He, et al. introduced skip connections and won ILSVRC 2015 with an notable low error rate error rate of 3.6% and beating human-level performance on this dataset. Such skip connections are also known as gated recurrent units and have a strong similarity to recent successful elements applied in RNNs. An example of ResNet-50 is given in Fig. 2.28. The idea behind a residual block is that the input still goes through conv-ReLU-conv series. The core idea of ResNet is introducing

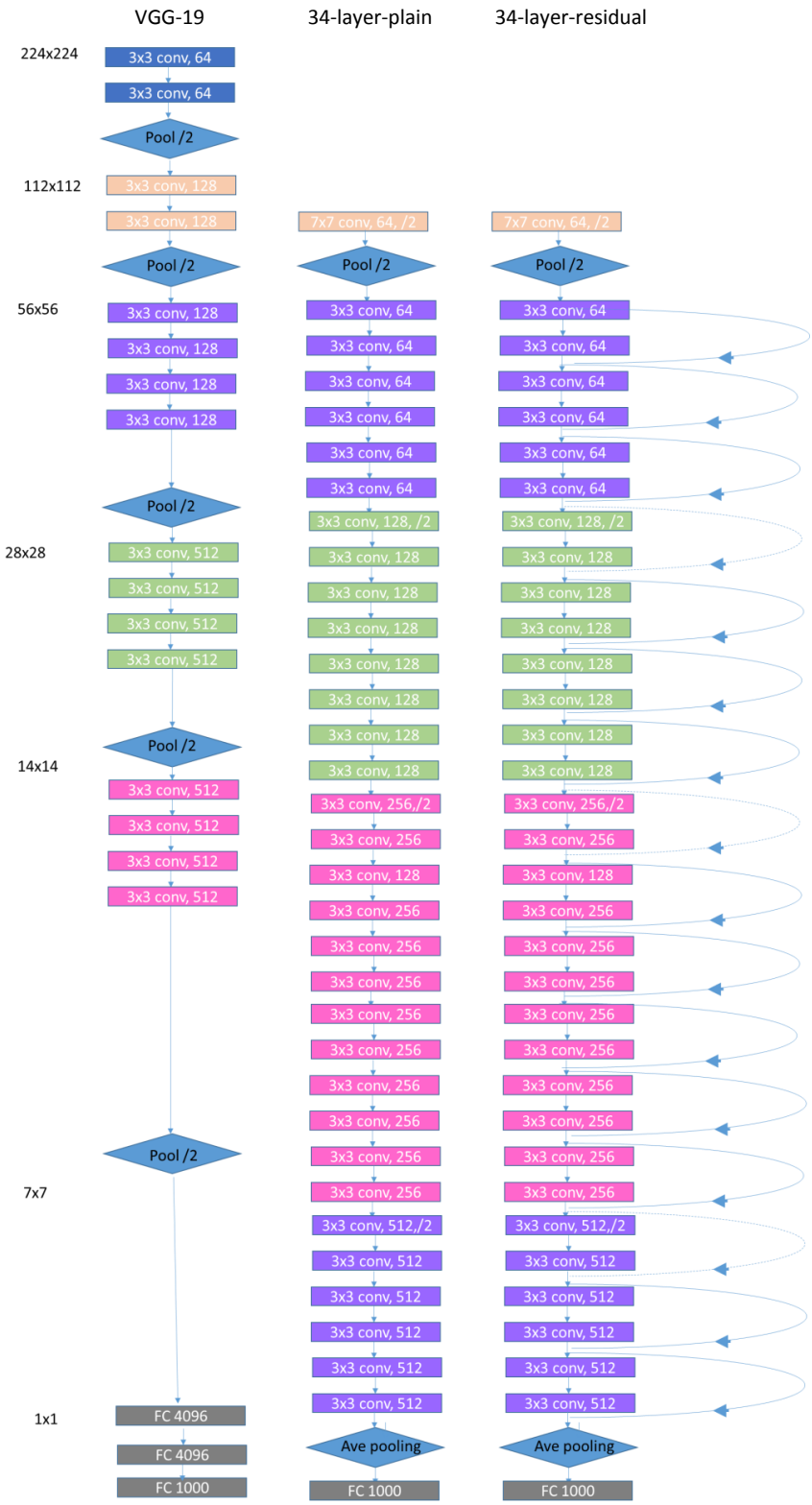


Figure 2.30: Comparison between Resnet and VGG-19, 34-layer-plain and 34-layer residual

a so-called identity shortcut connection that skips one or more layers, as shown in Fig. 2.29. The comparison between Resnet, VGG-19 and 34-layer plain networks is shown in Fig. 2.30.

2.5.7 DenseNet

Building upon the architecture of ResNet, DenseNet connects each layer to every other layer in a feed-forward fashion. The traditional convolution networks needs L connections for a L-layer network whereas DenseNet has $\frac{L(L+1)}{2}$ connections. DenseNet consists of multiple dense blocks, each of which consists of multiple layers. Each layer produces k features, where k is referred to as the growth rate of the network. Instead of drawing representational power from extremely deep or wide architectures, DenseNet exploits the potential of the network through feature re-use, yielding condensed models that are easy to train and with highly efficient parameters. In contrast to ResNet which combines features through summation before they are passed into a layer, DenseNet combines features by concatenating. An example of DenseNet with two blocks for image classification is given in Fig. 2.31. DenseNet layers are typically very narrow architectures e.g., 12 filters per layer, adding only a small set of feature-maps to the collective knowledge of the network and keep the remaining feature maps unchanged. DenseNet is composed of many components i.e. dense connectivity, composite function, transition layers, growth rate, bottleneck layers. Fig. 2.32 gives an illustration of DenseNet architecture on ImageNet for image classification. The l^{th} layer receives the feature-maps of all preceding layers, $\mathbf{x}_l = H_l([\mathbf{x}_0, \mathbf{x}_,...\mathbf{x}_{l-1}])$ as input. H_l is a composite function of three consecutive operations: BN, ReLU, 3×3 convolution. Transition layers used in our experiments consist of a batch normalization layer and an 1×1 convolutional layer followed by a 2×2 average pooling layer. If k is growth rate, H_l produces k feature maps, the input of l^{th} layer is $k_0 + k \times (l-1)$.

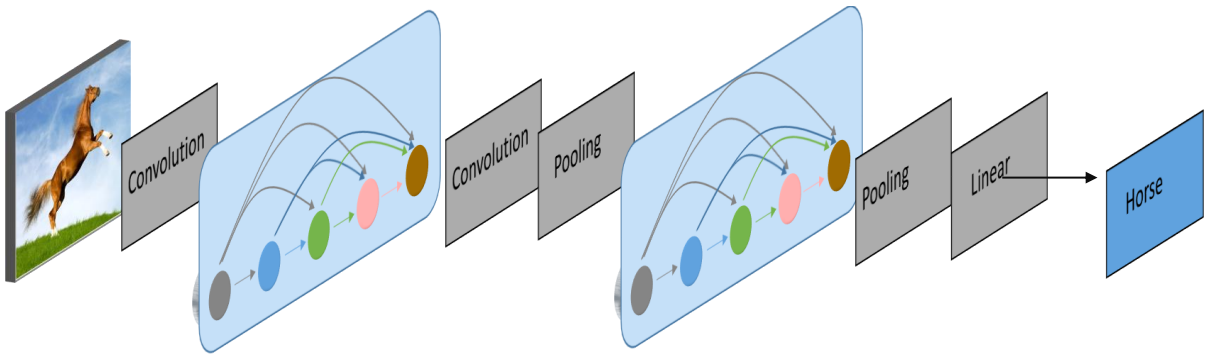


Figure 2.31: An example of DenseNet with 2 blocks for image classificaiton

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1\times1 \text{ conv} \\ 3\times3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Figure 2.32: Architecture of DenseNet on ImageNet for image classication

2.6 Region-based Convolutional Neural Networks

2.6.1 R-CNN

Region-based Convolutional Neural Networks (**R-CNN**) [30] is one of the important frameworks for the object detection and segmentation tasks. R-CNN, the first generation of family of region-based CNN, applies the high-capacity deep CNNs to classify given bottom-up region proposals. Selective search is used in R-CNN to generate 2,000 different regions that have the highest probability of containing an object. Then the CNNs are used as a feature

extractor and the system is further trained for object detection with Support Vector Machines (SVM). Finally, it performs bounding-box regression. The architecture of R-CNN is given if Fig. 2.33. The method achieves high accuracy but it is very time-consuming. R-CNN can be summarized as follows:

- The system can use the pre-trained model (AlexNet, VGG, Resnet) which is trained on image classification task.
- Generate a set of category-independent regions of interest by selective search ($\sim$2k candidates per image).
- Run the bounding boxes, which are warped to have a fixed size as required by CNNs, through a pre-trained network and finally an SVM to see what object the image in the box is.
- Run the boxes through a linear regression model to output tighter coordinates for the box once the object has been classified.
- The system requires a forward pass of the CNNs for every single region proposal for every single image.
- The system is trained by three different models separately - the CNNs to generate image features, the classifier that predicts the class, and the regression model to tighten the bounding boxes.
- At testing time, the detection process takes 47s per image using VGG-16 network.

It is easy to see that training a R-CNN model is expensive because the following problems:

- Running selective search to propose 2,000 regions of interest (RoIs) candidates for every image.
- Generating the CNN feature vector for every image region.
- The process is implemented by three models separately without much shared weights.

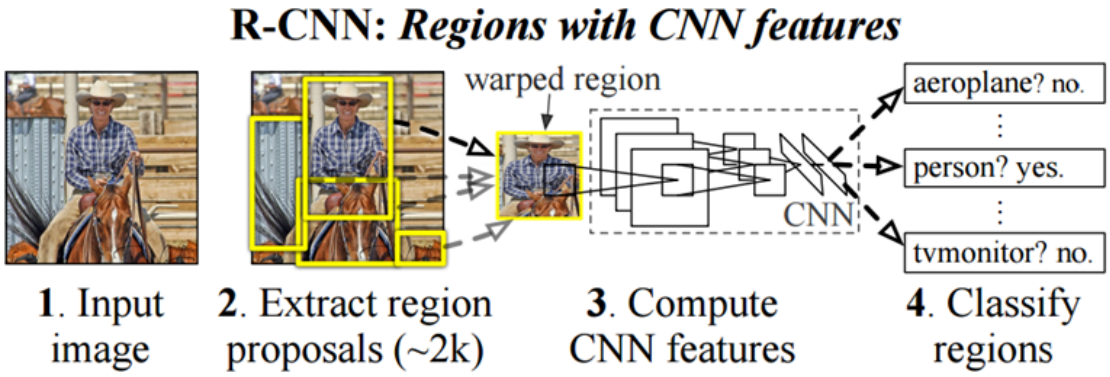


Figure 2.33: Flowchart of R-CNN

2.6.2 Fast R-CNN

[159] solves this problem by sharing the features between proposals and jointly training the CNNs, classifier, and bounding box regressor in a single model. The network is designed to only compute a feature map once per image in a fully convolutional style, and to use ROI pooling to dynamically sample features from the feature map for each object proposal. The network also adopts a multi-task loss, i.e. classification loss and bounding-box regression loss. Based on the two improvements, the framework is trained end-to-end. Fig. 2.34 gives an illustration of flowchart of fast R-CNN. The processing time for each image significantly reduced to 0.3s. Fast R-CNN accelerates the detection network using the RO I-pooling layer. In Fast-RCNN, SVM classifier is replaced by softmax layer. It also added a linear regression layer parallel to the softmax layer to output bounding box coordinates. However the region proposal step is designed out of the network and, hence, still remains a bottleneck, which results in a sub-optimal solution and exhibits a dependence on the external region proposal methods. The flowchart of Fast-RCNN, which contains many steps similar to R-CNN, can be summarized as follows:

- The system can be pre-trained on image classification task.
- Generate a set of category-independent RoIs by selective search ($\sim$ 2k candidates per

image).

- Sharing feature by replacing the last max pooling layer of the pre-trained CNNs with a RoI pooling layer. Using RoI pooling, the input region is divided into grids and then apply max-pooling in each grid. The RoI pooling layer outputs fixed-length feature vectors of region proposals.
- Replace the last fully connected layer and the last softmax layer (K classes) with a fully connected layer and softmax over K + 1 classes (same as in R-CNN, +1 is the background class).
- The system gives two outputs: softmax layer gives K + 1 classes and regression provide bounding boxes.
- The model is optimized for a loss combining two tasks of classification and localization.

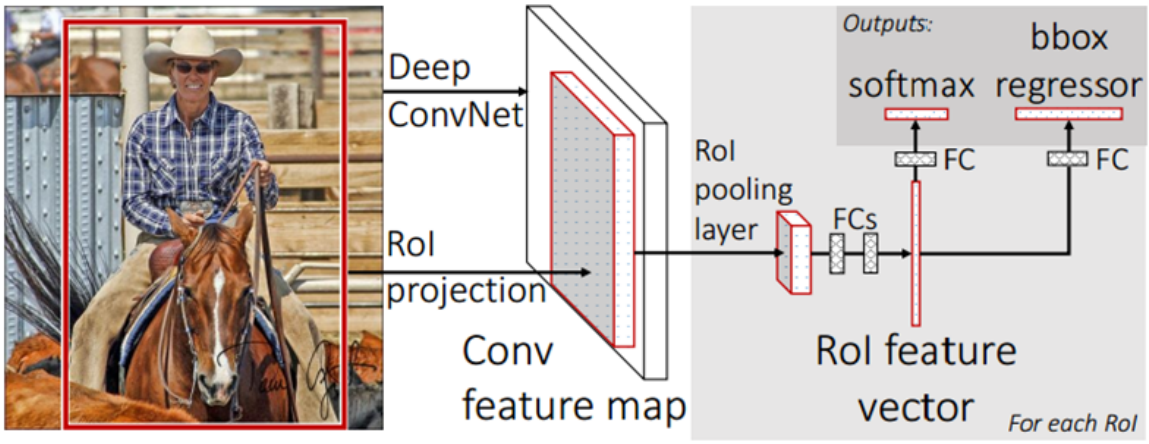


Figure 2.34: Flowchart of Fast-RCNN

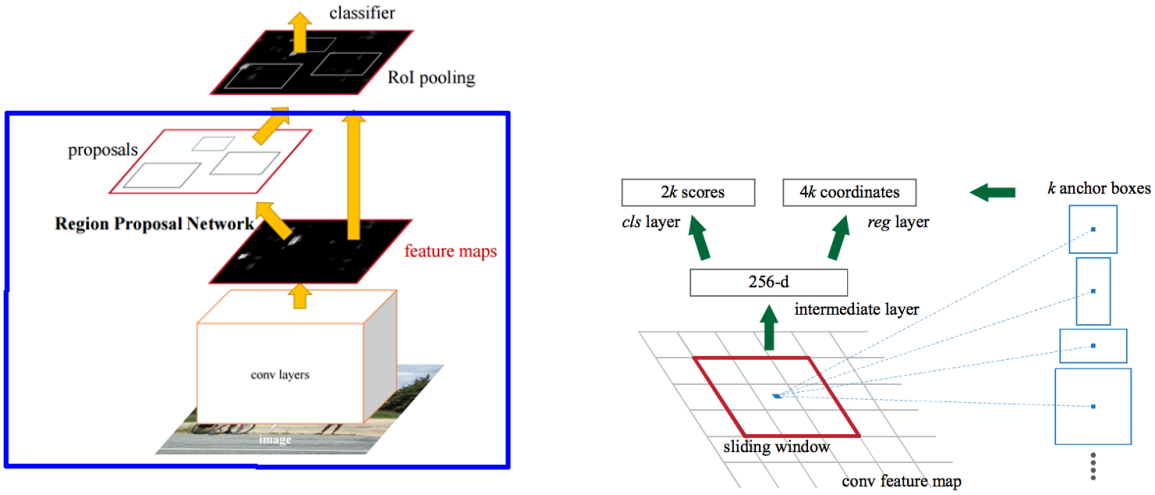
Clearly, Fast-RCNN is much faster than R-CNN in both training and testing time. However, the improvement is not dramatic because the region proposals are generated separately by another model and that is very expensive.

2.6.3 Faster R-CNN

[100] addresses the problem of generating RoI in fast R-CNN by introducing the Region Proposal Networks (RPNs) to replace selective search strategy. An RPNs is implemented in a fully convolutional style to predict the object bounding boxes and the objectness scores. In addition, the anchors, i.e. a set of predefined templates, are defined with different scales and ratios to achieve the translation invariance. The RPNs shares the full-image convolution features with the detection network. Fig. 2.35(a) gives an illustration of flowchart of faster R-CNN. In Fig. 2.35(a), RPN is inside the blue box. Therefore the whole system is able to complete both proposal generation and detection computation within 0.2s using very deep VGG-16 model [157]. With a smaller Zeiler-Fergus (ZF) model [66], it can reach the level of real-time processing. The flowchart of Faster R-CNN is given as follows:

- Pre-train CNNs network on image classification tasks.
- Slide a small window over the convolution feature map of the entire image. At the center of each sliding window, we predict multiple regions of various scales (1:1, 2:1, 1:2) and ratios (0.5, 1, 2), meaning there are 9 anchors at each sliding position.
- Fine-tune the RPNs end-to-end for the region proposal task by ignore all off-side region and keeping positive samples which have IoU > 0.7, while negative samples have IoU < 0.3.
- Train a Fast R-CNN object detection model using the proposals generated by the RPNs.
- Use the Fast R-CNN network to initialize RPN training. Herein, RPNs and the detection network have shared CNNs feature.
- Fine-tune the unique layers of Fast R-CNN.
- Similar Fast R-CNN, loss function combines the losses of classification and bounding

box regression.



(a) Flowchart of Faster-RCNN

(b) Region Proposal Networks (RPNs)

Figure 2.35: Flowchart of Faster-RCNN and Region Proposal Networks (RPNs)

2.6.4 Mask R-CNN

Mask R-CNN [160] extends Faster R-CNN [100] to pixel-level image segmentation by adding a branch for predicting segmentation masks on each Region of Interest (RoI), in parallel with the existing branch for classification and bounding box regression. The mask branch is a small FCN applied to each RoI, predicting a segmentation mask in a pixel-to pixel manner. Fig. 2.36 gives an illustration of flowchart of Mask R-CNN. Mask R-CNN is simple to implement and train given the Faster R-CNN framework, which facilitates a wide range of flexible architecture designs. Additionally, the mask branch only adds a small computational overhead, enabling a fast system and rapid experimentation. Since pixel-level segmentation requires much more fine-grained alignment than bounding boxes, mask R-CNN replaces the RoI pooling layer by RoIAlign layer. Therefore, RoI can be better and more precisely mapped to the regions of the original image. RoIAlign has a large impact: it improves mask accuracy by relative 10% to 50%, showing bigger gains

under stricter localization metrics. The multi-task loss function of Mask R-CNN combines the loss of classification, localization and segmentation mask.

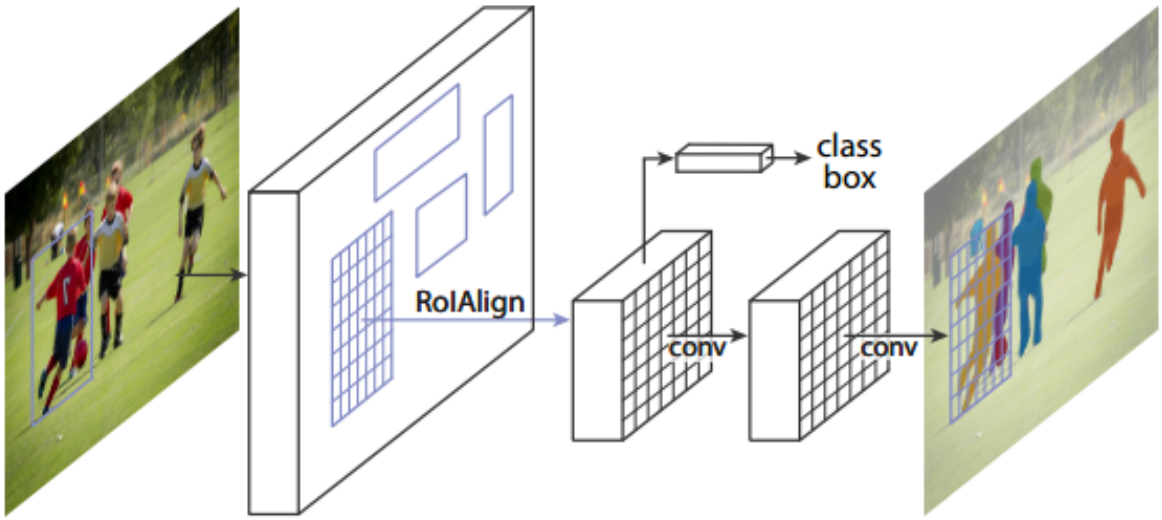


Figure 2.36: Flowchart of Mask R-CNN

Summary of models in the R-CNN family is illustrated in Fig. 2.37

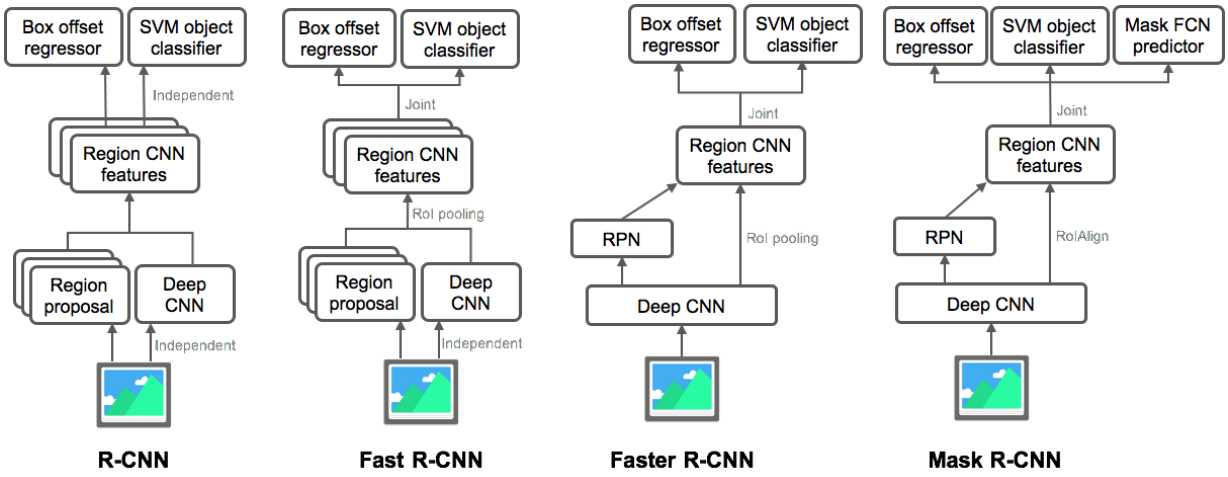


Figure 2.37: Summary of models in the R-CNN family

2.6.5 YOLO

Different from the aforementioned models in the R-CNN family which learn to solve a classification task, YOLO model (You Only Look Once)[161] treat the task of object recognition as a unified regression problem. The flowchart of YOLO model is shown in Fig. 2.38 with following steps:

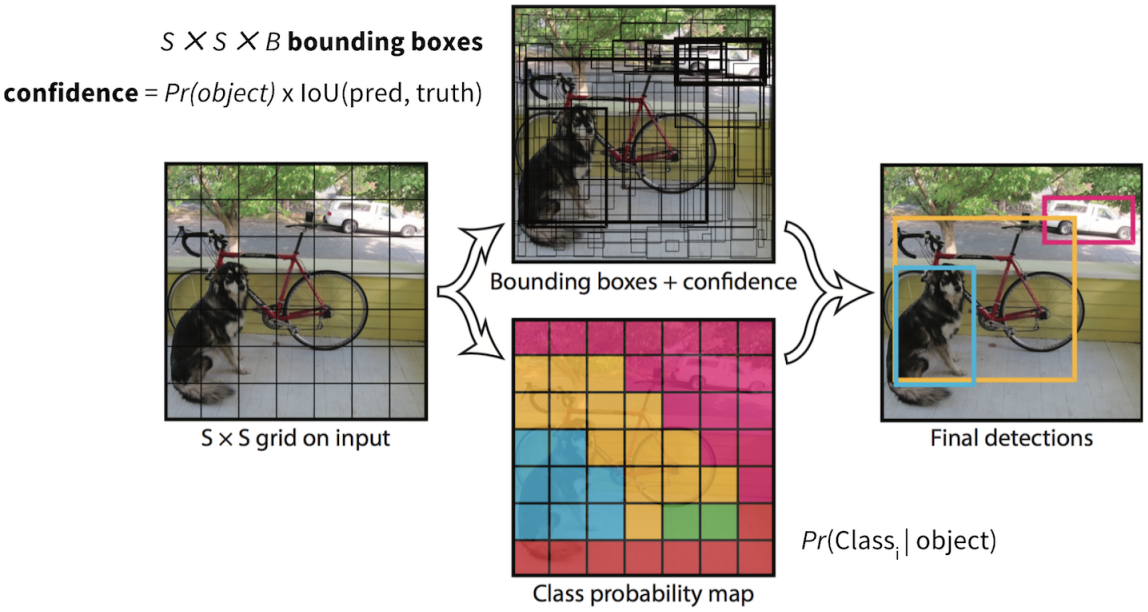


Figure 2.38: Flowchart of YOLO model

- Pre-train CNN network on image classification tasks.
- Split an image into a set of SxS cells (grids).
- Two information is extracted from each cell: (1) location of B bounding boxes and a confidence score which is probability of containing an object. (2) probability of object class conditioned on the existence of an object in the bounding box (the object belongs to the class C_i— containing an object). If the cell contains an object, it predicts a probability of this object belonging to one class C_i where $i = 1, 2, \cdots, K$.
- Each box corresponding to 4 location predictions, 1 confidence score, and K conditional probability scores for object classification.

- The final layer of the pre-trained CNNs is modified to output a prediction tensor of size $S \times S \times (5B + K)$.
- The YOLO is trained to minimize the sum of squared errors, with scale control parameters: control the loss from bounding box predictions and control the loss from confidence predictions for non-object boxes.

Chapter 3

Related Work

This chapter intends to provide a brief review of different image segmentation techniques. Image segmentation plays an important role in various areas of image processing, such as object detection and classification, action classification, scene understanding, and other visual information analysis processes. In general, image segmentation techniques are broadly divided into two main categories, namely, unsupervised approach with naive classical image segmentation methods and supervised approach with semantic image segmentation methods. Most of the classical approaches depend on filtering and statistical techniques such as thresholding, edge detection, boundary based, region-based, morphological, normalised graph cut, k-means approaches, etc. The other is learning-based approach which consists of both traditional learning methods such as linear discriminant analysis (LDA), support vector machine (SVM) and the recent developments on deep neural networks including CNNs-based and RNNs-based methods. Deep neural networks (DNNs)-based approach have become popular and productive for semantic image segmentation and image understanding with real-time application.

As definition, image segmentation is to separate an image into foreground and background whereas semantic segmentation does not only segment the predefined objects out of the background but also classifies the object category as shown in Fig. 3.1. However,

the semantic segmentation is unable to separate instances (specially overlapping) of the same category. Meaning that semantic segmentation cannot count the number of instances existing in the image as given in Fig. 3.1 (b) which can not tell how many horses. Semantic instance segmentation is to address this issue as shown in Fig. 3.1 (c).

In this chapter, we first review the unsupervised image segmentation category which focuses more on active contour techniques. We then give a literature on supervised image segmentation category which focuses more on DNNs-based approaches. In the end of this chapter, we introduce the most common large-scale datasets that have been used to evaluate the performance of different semantic segmentation approaches.

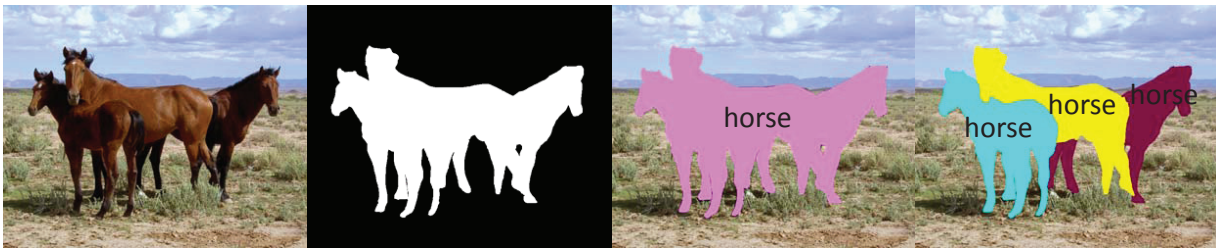


Figure 3.1: An example of the segmentation result (b) from an input image (a), semantic segmentation (c) and semantic instance segmentation (d)

.

Depend on application, i.e. texture segmentation, scene understanding, medical segmentation, there are various challenges when dealing with image segmentation.

Texture images: Traditionally, the combination of texture classification and image segmentation is used in order to segment the disjoint uniform regions based on textures. As given in Fig. 3.2, it is affected by various external aspects such as illumination, resolution, pose, etc. Moreover, it is difficult to generalize the system in order to find a pattern without the knowledge of domain.

Scene images: Scene labeling is an important task in the image understanding problem. In such natural scene images, low-level features, such as color or texture, extracted from a small window around pixels (short-range context) is not enough information to

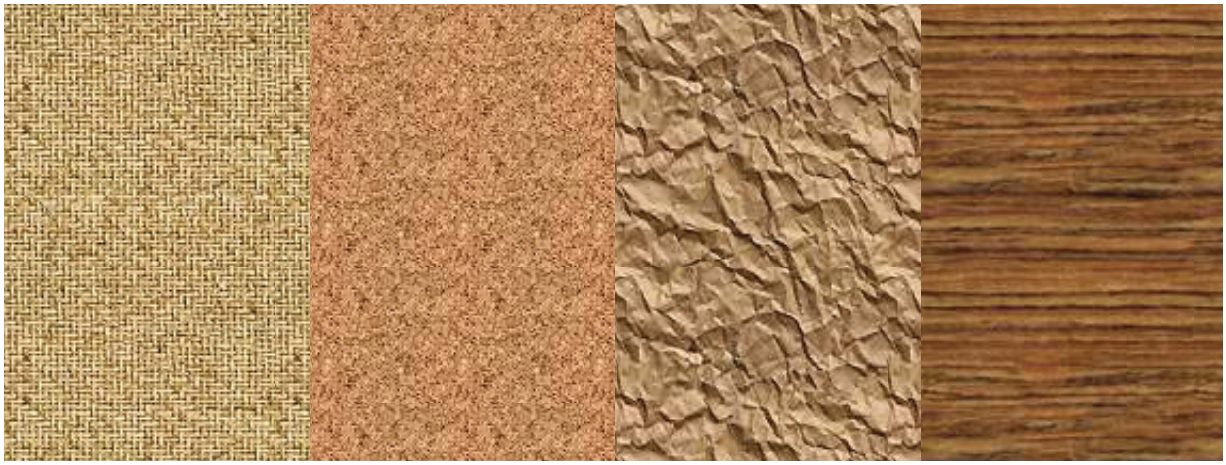


Figure 3.2: Difficulties in texture segmentation

Figure 3.3: Difficulties in scene segmentation

label pixels. For example, it is almost impossible to distinguish the "sea" from the "sky" or distinguish "tree" from "forest" using short-range context or low-level feature as given in Fig. 3.3.

Medical Images: Analyzing biomedical images is one of the most import subjects of study for biologists. Such images are usually low quality and contains a lot of noise. One of the common solutions used image segmentation algorithms is Level Set. However its effect depends on many conditions such as illumination, noise, and require knowledge of the domain. Available medical dataset is usually small and not big enough for deep learning approach which requires quite large number of training samples.

3.1 Naive Image Segmentation

Several classical unsupervised image segmentation methods have been reported in the literature [4].

3.1.1 Thresholding Technique

Thresholding technique is one of the important techniques for image segmentation in this category. In this technique, it is assumed that all pixel intensities of an image which are lying within a certain range belong to the same class. This technique deals with the pixel intensity of an image such as grayscale level, color level, texture level with the main goal is to determine the threshold value of an image using. The pixels whose intensity values exceed the threshold value are assigned in one segment and the remaining are assigned in zero segment. Histogram thresholding [162, 163] is also a useful technique in image segmentation. The histogram thresholding techniques are based on the similarity between grey levels. Image segmentation using entropy-based thresholding [164] is also common in this category. One of the oldest methods, the Otsus thresholding method [165] has been utilized to analyze the maximum separability of classes. In this method, an optimal threshold is selected by the discriminant criterion, namely, so as to maximize the separability of the resultant classes in gray levels. By utilizing only the zeroth- and the first-order cumulative moments of the gray-level histogram, Otsu's thresholding is very simple but has proven to be one of the most popular binarization approaches. Using cluster organization from the histogram of an image, [166] proposes a gray-level thresholding algorithm. Their method provide a new similarity measure based on inter-class variance of the clusters to be merged and the intra-class variance of the new merged cluster. Besides grayscale level thresholding, color thresholding by pixel classification in a hybrid colour space for color image segementation is also studied [167, 168]. [168] is first based on histogram thresholding to determine the consistent regions in the color image. Then, fuzzy c-means algorithm for

the betterment of the compactness of the segments. The main drawbacks of thresholding technique are that these cannot be efficiently applied for blurred images or for multiple image component segmentations or inhomogeneous object.

3.1.2 Edge-based Technique

Edge detection or boundary based technique is another common classical image segmentation techniques. In this technique, it is assumed that the pixel intensities change abruptly at the edges between two regions in the image. This approach is more akin to edge detection or boundary detection than to true image segmentation. Some common edge-based segmentation techniques are gradient, Laplacian, Laplacian of a Gaussian, Sobel edge detector, Canny edge detector [4]. However, most of the existing edge detectors provide discontinuous or over-detected edges while actual region boundaries should be closed curves. Thus, post-processing such as edge tracking, gap filling, smoothing and thinning are used in such techniques to obtain closed region boundaries. Multiscale edge detection technique is presented in [169] for segmenting natural images. This approach is quite efficient to eliminate the need of explicit scale selection and edge tracking methods. By combining an improved isotropic edge detector and a fast entropic thresholding technique, an automatic color image segmentation method is proposed by [170]. In this method, seeded region growing (SRG) is initialized by centroids between these adjacent edge regions. These seeds are then replaced by the centroids of the generated homogeneous image regions by incorporating the required additional pixels step by step. With complicated images where objects contain many edges/boundaries, the edge detection technique usually give over-segmentating results and cannot determinate the closed boundaries.

3.1.3 Region-based Technique

Region-based techniques looking for uniformity within a sub-region, based on a desired property, such as, intensity, color or texture have been widely used to solve image segmentation problems. One of the simplest implementations of this approach is region growing [171, 172] which merges pixels or small sub-regions into larger sub-regions. In region-growing or merging techniques, the input image is first fitted into a set of homogeneous primitive regions. Then, similar neighboring regions are merged according to a certain decision rule following by an iterative merging process. There are two main steps in this techniques, i.e. splitting and merging. The first step (splitting) is initialized an entire image as one region. Then, each region is partitioned into four regions (each region is one segment). The process is terminated when all regions of the image are homogeneous. In the next step of merging, all the similar neighboring region are merged to unify the segmentation results. Different from greedy techniques in [171, 172], [173] heuristic-based region merging is applied to over-segmented images. In this approach, the over segmentation is obtained by different segmentation techniques or even the same technique with different initial conditions. The merging process is guided using only information about the behavior of each pixel in the input segmentations. Seeded region growing (SRG) for image segmentation is introduced by [174] which is based on the conventional region growing postulate of similarity of pixels within region. SRG is controlled by a number of initial seeds without tuning the homogeneity parameters. By itself, however, it is not a self-contained process, as it also requires the input of a few control points in the image known as seeds. To overcome this limitation, [175] proposes an automatic seeded region growing algorithm to segment colour and multispectral images. In this method, histogram analysis is used to automatically generate the seeds and instance-based learning is used as distance criteria. However, selecting initial seeds is a crucial problem in region growing. Moreover, region growing or region competition adjacent sub-regions under criteria involving the uniformity of regions

or sharpness of boundaries. Strong criteria tend to produce over-segmented results, while weak criteria tend to produce poor segmentation results by over-merging the sub-regions with blurry boundaries. Region-based approaches are generally less sensitive to noise, and usually produce more reasonable segmentation results as they rely on global properties rather than local properties, but their implementation complexity and computational cost can often be quite large. Furthermore, most region-based segmentation methods do not explicitly represent the uncertainty in the estimated parameter values and, therefore, give incorrect segmentation when parameter estimates are poorly selected.

3.1.4 Watershed Technique

As definition, a watershed in geography is a ridge that divides areas drained by different river systems. If a pixel intensities are considered as a topological surface in which the pixel value at each pixel represents the height of the surface, then segmentation of the image is defined by the resulting basins. An image has many different heights to produce a practical segmentation, transformation of color intensity is applied in watershed. In particular, the gradient magnitude after being smoothed by preprocessing procedures is commonly used instead of the original image. To obtain better segmentation results, foreground and background markers are also commonly used in watershed. [176, 177] proposes a watershed segmentation method using gradient under marker-controlling. In this method, the foreground marker is placed in he blobs and background markers for the areas without blobs. The procedure contains six main steps: (a) Convert the color image to grayscale. (b) Compute the gradient magnitude from the grayscale image. (c) Mark the foreground objects. (d) Mark the background. (e) Modify the gradient magnitude image so that its regional minima occur at foreground and background marker pixels. (f) Apply the watershed transform to the modified gradient image. Steps (c) and (d) can be automatically done by mathematical morphology with erosion for opening and dilation for

reconstruction.

3.1.5 Normalized -Cut Technique

Normalized graph cuts (Ncut) a special case spectral clustering has become very popular over the years for addressing the problem of image segmentation. [178] uses normalised cut by solving an eigensystem of equations to propose a general image segmentation approach. This method aims at extracting the global impression of an image by treating the image segmentation as a graph partitioning problem. (Ncut) criterion is proposed to measure both the total dissimilarity between the different groups as well as the total similarity within the groups. Using local spectral histograms and graph partitioning methods to separate an image into texture and nontexture are proposed by [179]. Inherit the merits of both watershed and (Ncut) is found in [180]. By incorporating the advantages of the mean shift (MS) segmentation and Ncut partitioning methods, [181] proposes method requires low computational complexity and is therefore very feasible for real-time image segmentation processing. In this method, MS is applied as preprocessing to form segmented regions that preserve the desirable discontinuity characteristics of the image. The segmented regions are then represented by using the graph structures, and the Ncut method is applied to perform globally optimized clustering. This segmentation algorithm provides superior outcome as it is applied on the adjacent regions instead of image pixels. Based on the graph reduction method, [182] developed a cost function, named as ratio cut to segment color images efficiently. The cut ratio is defined as the ratio of the corresponding sums of two different weights of edges along the cut boundary and models the mean affinity between the segments separated by the boundary per unit boundary length. This cost function works efficiently with both region-based segmentation and pixel-based segmentation approaches.

3.1.6 Active Contour based Technique

The key idea behind the Active Contour (AC) for image segmentation is to start with an initial guess boundary represented in a form of closed curves i.e. contours C. The curve is then iteratively modified by applying shrink or expansion operations and moved by image-driven forces under given constraints to more accurately detect the object boundaries to the boundaries of the desired objects. The entire process is called contour evolution, denoted as $\frac{\partial C}{\partial t}$ where C is the object boundary (contour). The existing active contour models are generally categorized into three groups, depend on the kind of information used: *edge-based models* [145, 183, 184, 185], *global region-based models* [62, 148, 186, 187], *local region-based active contour models* [149], [188], [189], [190], [191], [192], [193]. The first category, *edge-based models*, utilizes image gradient as an additional constraint to stop the contours on the boundaries of the desired objects. For instance, geodesic active contour (GAC) model [183] constructs an gradient stop function to attract the AC to the object boundaries. Later, [184] proposes an novel AC model which is able to eliminate the expensive re-initialization procedure by penalizing the deviation of the level set function. In general, these kinds of models have the ability to handle only images with well-defined edge information. However, they are sensitive to image noise and weak boundaries. To overcome those problems, the second category, *global region-based models*, uses the statistical information inside and outside AC to guide the curve evolution. Clearly, global region-based models have several advantages over edge-based models such as less sensitivity to image noise and a higher capability to detect weak boundaries (even without boundaries) because they do not use the image gradient. Furthermore, global region-based models are robust to initial contours which means the initial contour can start anywhere in the image. One of the most successful region-based models is the piecewise constant model [62] with the assumption that each image region is statistically homogeneous. It is, however, limited to handle the image inhomogeneity problem which is then solved by the

third category using *localized image information as constraints*. One of the early works is proposed by Li et al.[188] which defines a local weighted K-means model in the level set formulation for both image segmentation and bias correction. The K-means clustering objective function is integrated over the entire domain and incorporated into a variational level set formulation. Later, Zhang et al.[189] presents a local image fitting (LIF) model in the LS formulation, which minimizes the differences between the fitted image and the original one. Wang et al.[193] proposes a local Gaussian distribution fitting (LGDF) model, an improvement of the local binary fitting (LBF) model [188], which models the local image intensities by Gaussian distributions with different means and variances.

Images in the wild are corrupted by large amount of problems such as strong noise, illumination, various resolution, texture, and intensity inhomogeneity simultaneously. State-of-the-art models (CV, LBF, LIF, LGDF, etc.) have failed to obtain high accurate segmentation. Recently, many improved AC models have been proposed via introducing more discriminative features [21], [194], [195]. Mukherjee and Acton [21] generalize the traditional Level Set model by representing the illumination of the regions of interest in a lower dimensional subspace using a set of pre-specified basis functions. Using both global energy term to characterize the fitting of global Gaussian distribution according to the intensities inside and outside the evolving curve, as well as a local energy term to characterize the fitting of local Gaussian distribution based on the local intensity information, Zhou et al. [194] define a unified fitting energy framework based on Gaussian probability distributions to obtain the maximum a posteriori probability (MAP) estimation. Lately, Ngo et al.[195] combines deep learning and level set for the automated segmentation of the left ventricle of the heart from cardiac cine magnetic resonance (MR) data. Multi-region segmentation has been recently taken care. Dubrovina et al. [152] have developed a multi-region segmentation with single LS function. However, Dubrobina et al's approach is developed for contour detection and needs good initialization, namely, it requires specify more initial regions than it is expected to be in the final segmentation. Furthermore, the algorithm

requires that initial contours cover the image densely, specifically the initial contour has to pass through all different regions. In addition to multi region segmentation problem, optimization [19, 23, 131], and shape prior [22], have also been considered.

Most of the aforementioned classical unsupervised image segmentation approaches require some priori knowledge regarding the image data distribution or appropriate parameters to be operated upon.

3.2 Semantic Image Segmentation

Semantic segmentation refers to associating one of the classes to each pixel in an image. This problem has played an important role in many research areas and has attracted numerous studies recently when deep learning have become ubiquitous in semantic segmentation with three main approaches, namely graphical model approaches, CNN-based approaches and RNN-based approaches. Semantic instance segmentation is considered the next step after semantic segmentation and at the same time the most challenging problem in comparison with the rest of low-level pixel segmentation techniques. Its main purpose is to represent objects of the same class splitted into different instances. The automation of this process is no straight forward, thus the number of instances is initially unknown and the evaluation of performed predictions is not pixel-wise such as in semantic segmentation. Consequently, this problem remains partially unsolved but the interest in this field is motivated by its potential applicability. Instance labeling provides us extra information for reasoning about occlusion situations, also counting the number of elements belonging to the same class and for detecting a particular object for grasping in robotics tasks, among many other applications.

3.2.1 Graphical model approaches

In the past, using traditional vision techniques, semantic segmentation was approached from a undirected graphical model paradigm utilizing Markov Random Fields (MRF) and Conditional Random Fields (CRF) [46, 48, 49, 196, 197]. Lately, He, et al. [46] proposes contextual feature incorporated into CRF which combines the outputs of several components i.e. image-label mapping, patterns within the label field, fine/coarse resolution patterns. The context information is then continue studied under an auto-context algorithm in [198]. The context information is learn via a discriminative probability maps which is then applied to vision tasks and 3D Brain image segmentation. In contrast to much previous works on structured prediction, [48] directly trains a hierarchical inference procedure inspired by the message passing mechanics of some approximate inference procedures in graphical models. Later, [196] proposes nonparametric approach for object recognition and scene parsing using label transfer and dense SIFT flow algorithm whereas as [197] utilized k-nearest neighbors in a retrieval dataset to classify superpixels.

One of the first studies to utilize contextual information within the MRF framework is proposed by [49]. Similar efforts have been made for using CRFs on unary and pairwise image features [199]. Parametric and non-parametric techniques are also combined to model global order dependencies to provide more information and context [1]. Higher order dependencies are also modeled using a fully connected graph [200, 201]. One of the recently works on graphical model is proposed by [32] which introduces a new form of convolutional neural network that combines the strengths of CNNs and Conditional Random Fields (CRFs)-based probabilistic graphical modelling to delineate visual objects. This approach formulates mean-field approximate inference for the Conditional Random Fields with Gaussian pairwise potentials as RNNs.

3.2.2 CNN-based approaches

In this section, we start review the CNN-based methods for semantic segmentation. Scene labeling is a particular case of semantic segmentation. Then we comprehend the CNN-based methods for semantic instance segmentation.

Semantic Segmentation

One of the first works on CNN-based semantic segmentation approaches is proposed by [59]. It uses a multiscale convolutional network trained from raw pixels to extract dense feature vectors that encode regions of multiple sizes centered on each pixel. This work consists of two components: multi-scale convolutional representation and graph-based classification. To simultaneously detect and segment all instances of a category in an image, [202] proposes SDS (Simultaneous Detection and Segmentation) approach which uses CNN to classify region proposals. Later, [31] proposes fully convolutional networks (FCNs) that take input of arbitrary size and produce correspondingly-sized output with efficient inference and learning. It adapts classification networks to fully convolutional networks and transfers learned representations to the segmentation task. It also defines a skip architecture that combines semantic information from a deep, coarse layer with appearance information from a shallow, fine layer to produce accurate and detailed segmentations. Many works later that make use of [31] are proposed later. Some works like [1, 39, 203, 204, 205, 206] archive remarkable segmentation performance. [203] combines the responses at the final layer with a fully connected CRF to overcome the poor localization problem of deep networks. [204] proposes a method that achieves competitive accuracy but only requires easily obtained bounding box annotation. The basic idea is to iterate between automatically generating region proposals and training convolutional networks. These two steps gradually recover segmentation masks for improving the networks, and vise versa. [205] studies the more challenging problem of learning CNNs for semantic image segmentation from either (1) weakly

annotated training data such as bounding boxes or image-level labels or (2) a combination of few strongly labeled and many weakly labeled images, sourced from one or multiple dataset. [206] introduces a purely feed-forward architecture mapping small image elements (superpixels) to rich feature representations extracted from a sequence of nested regions of increasing extent. These regions are obtained by zooming out from the superpixel all the way to scene-level resolution. This approach exploits statistical structure in the image and in the label space without setting up explicit structured prediction mechanisms, and thus avoids complex and expensive inference. In order to mitigates the limitations of the existing methods based on FCNs, [207] identifies detailed structures and handles objects in multiple scales natural by integrating deep deconvolution network and proposal-wise prediction. The deconvolution network is composed of deconvolution and unpooling layers, which identify pixelwise class labels and predict segmentation masks. To dealing with long range context, [1] introduce a global scene constraint to alleviate similar pixels they are indistinguishable from local context. They estimate the global potential in a non-parametric framework. For better global potential estimation, a large margin based CNN metric learning method is proposed. The final pixel class prediction is performed by integrating local and global beliefs. SegNet [39] consists of an encoder network, a corresponding decoder network followed by a pixel-wise classification layer. The encoder network is topologically identical to the 13 convolutional layers in the VGG16 network whereas the decoder is to map the low resolution encoder feature maps to full input resolution feature maps for pixel-wise classification. SegNet is developed upon the Caffe framework and evaluated on road scenes and SUN RGB-D indoor scene segmentation tasks. Dilation Network [40] develops a new CNNs module that is specifically designed for dense prediction. The presented module uses dilated convolutions to systematically aggregate multiscale contextual information without losing resolution. The architecture is based on the fact that dilated convolutions support exponential expansion of the receptive field without loss of resolution or coverage. Also, to solve the problem of dense prediction, DeepLab [208] uses atrous convolution to highlight

convolution with upsampled filters. Atrous convolution allows us to explicitly control the resolution at which feature responses are computed within deep CNNs. Furthermore, they propose atrous spatial pyramid pooling (ASPP) to robustly segment objects at multiple scales. To improve the localization of object boundaries, they combine methods from deep CNNs and probabilistic graphical models. Paying more attention on speed, CNN-based network for mobile is another important research. Efficient neural network (ENet) [209] is created specifically for tasks requiring low latency operation. ENet defines bottleneck module as a combination of a 1×1 projection that reduces the dimensionality, a main convolutional layer, and a 1×1 expansion. Batch Normalization and PReLU [68] are used between all convolutions. This network does not contain post-processing and bias term. Different from other networks that use pooling indices in the last upsampling module, ENet decoder max pooling is replaced with max unpooling, and padding is replaced with spatial convolution without bias. ENet requires low latency operation which is up to $18\times$ faster, requires $75\times$ less FLOPs, has $79\times$ less parameters, and provides similar or better accuracy to existing models.

Recently, [210] explores patch-patch context between image regions, and "patch-background" context. For learning from the patch-patch context, they formulate CRFs with CNN-based pairwise potential functions to capture semantic correlations between neighboring patches. Efficient piecewise training of the proposed deep structured model is then applied to avoid repeated expensive CRF inference for back propagation. For capturing the patch-background context, they use a network design with traditional multi-scale image input and sliding pyramid pooling.

Semantic Instance Segmentation

Semantic instance segmentation is considered the next step after semantic segmentation and at the same time it is the most challenging problem in comparison with the rest of low-level pixel segmentation techniques. Its main purpose is to represent objects of

the same class splitted into different instances. The automation of this process is no straight forward, thus the number of instances is initially unknown and the evaluation of performed predictions is not pixel-wise such as in semantic segmentation. Consequently, this problem remains partially unsolved but the interest in this field is motivated by its potential applicability. Instance labeling provides us extra information for reasoning about occlusion situations, also counting the number of elements belonging to the same class and for detecting a particular object for grasping in robotics tasks, among many other applications.

For this purpose, Hariharan et al. [26] proposed a Simultaneous Detection and Segmentation (SDS) method in order to improve performance over already existing works. Their pipeline uses, firstly, a bottom-up hierarchical image segmentation and object candidate generation process called Multi-scale Combinatorial Grouping (MCG) [211] to obtain region proposals. For each region, features are extracted by using an adapted version of the Region-CNN (R-CNN)[30], which is fine-tuned using bounding boxes provided by the MCG method instead of selective search and also alongside region foreground features. Then, each region proposal is classified by using a linear Support Vector Machine (SVM) on top of the CNN features. Finally, and for refinement purposes, Non-Maximum Suppression (NMS) is applied to the previous proposals. Later, Pinheiro et al. [212] present DeepMask model, an object proposal approach based on a single convolutional network. This model predicts a segmentation mask for an input patch and the likelihood of this patch for containing an object. The two tasks are learned jointly and computed by a single network, sharing most of the layers except last ones which are task- specific. In their system, the discriminative convolutional network is used to generate object proposals. Given an image patch, the training objective function includes two tasks, i.e. class-agnostic segmentation and likelihood of the patch being centered on a full object. Based on the DeepMask architecture as a starting point due to its effectiveness, the same authors presented a novel architecture for object instance segmentation implementing a top-down refinement process [213] and

achieving a better performance in terms of accuracy and speed. The goal of this process is to efficiently merge low-level features with high- level semantic information from upper network layers. The process consisted in different refinement modules stacked together (one module per pooling layer), with the purpose of inverting pooling effect by generating a new upsampled object encoding. Another approach, based on Fast R-CNN as a starting point and using DeepMask object proposals instead of Selective Search is presented by Zagoruyko et al [214]. This combined system called MultiPath classifier, improved performance over COCO dataset and supposed three modifications to Fast R-CNN: improving localization with an integral loss, provide context by using foveal regions and finally skip connections to give multi-scale features to the network. The system achieved a 66% improvement over the baseline Fast R-CNN. One of the notable networks which is proposed by [29] presents multi-task network cascades (MNC) model. MNC consists of three networks, respectively differentiating instances, estimating masks, and categorizing objects. These networks form a cascaded structure, and are designed to share their convolutional features. The model is designed for the nontrivial end-to-end training of this causal, cascaded structure. It inherits all the merits of FCNs for semantic segmentation [31] and instance mask proposal [29] [29] proposes Fully Convolutional Instance Segmentation (FCIS) for instance-aware semantic segmentation task. FCIS detects and segments the object instances jointly and simultaneously. By introducing the position-senstive inside/outside score maps, the underlying convolutional representation is fully shared between the two sub-tasks, as well as between all regions of interest. Mask R-CNN [215] is a conceptually simple, flexible, and general framework for object instance segmentation. This approach efficiently detects objects in an image while simultaneously generating a high-quality segmentation mask for each instance. Mask R-CNN extends Faster R-CNN by adding a branch for predicting an object mask in parallel with the existing branch for bounding box recognition. Recently, Hayder et al. [216] also addresses the problem of instance-level semantic segmentation that they cannot recover from errors in the object candidate generation process, such as too small or shifted

boxes. The authors introduce a novel object segment representation based on the distance transform of the object masks. They then designed an object mask network (OMN) with a new residual-deconvolution architecture that infers a representation and decodes it into the final binary object mask. This allows to predict masks that go beyond the scope of the bounding boxes and are thus robust to inaccurate object proposal boxes. However, this work does not tackle bounding box issue directly and the model is actually required a post-processing stage to obtain refined bounding box. By extending the pixel-level feature to the specially designed global pyramid pooling, [44] proposes pyramid scene parsing network (PSPNet) to improve the traditional dilated FCNs [40]. In PSPNet model, they develop an effective optimization strategy for deep ResNet based on deeply supervised loss. Given an input image, CNN is applied to get the feature map of the last convolutional layer then a pyramid parsing module is applied to harvest different sub-region representation. This is followed by upsampling and concatenation layers to form the final feature representation, which carries both local and global context information. Finally, the representation is fed into a convolution layer to get the final per-pixel prediction. The successful DenseNet, which has shown excellent results on image classification tasks, is extended to semantic segmentation by [217] by adding an upsampling path to recover the full input resolution. This network is built from dense blocks with two main paths, i.e. downsampling path with 2 Transitions Down (TD) and an upsampling path with 2 Transitions Up (TU). The network can be seen as a combination of DenseNet and FCNs.

3.2.3 RNN-based approaches

RNNs have been successfully applied to wide variety of tasks, including in speech processing [218], natural language processing [219] and lately in image processing [220]. [53] is one of the first works that uses the recurrent architecture to parse the scene with a smoother class annotation. [53] consists of a recurrent convolutional neural network which allows

to consider a large input context while limiting the capacity of the model. Contrary to most standard approaches, their method does not rely on any segmentation technique nor any task specific features. Later, [32] introduces a new form of convolutional neural network that combines the strengths of CNNs and Conditional Random Fields (CRFs)-based probabilistic graphical modelling to delineate visual objects. This approach formulates mean-field approximate inference for the Conditional Random Fields with Gaussian pairwise potentials as RNNs. Investigating two dimensional (2D) LSTM networks for natural scene images takes into account the complex spatial dependencies of label was proposed by [3]. The networks efficiently capture local and global contextual information over raw RGB values and adapt well for complex scene images. To learn the spatial dependencies between image regions to enhance the discriminative power of image representation, [221] proposes the convolutional recurrent neural network (C-RNN). The C-RNN is trained in an end-to-end manner from raw pixel images. CNN layers are firstly processed to generate middle level features. RNN layer is then learned to encode spatial dependencies. To exploit the local generic features extracted by CNNs and the capacity of RNNs to retrieve distant dependencies, [52] proposes a structured prediction architecture, called ReSeg which is based on the recently introduced ReNet [222] model for image classification. In this network, each ReNet layer is composed of four RNNs that sweep the image horizontally and vertically in both directions, encoding patches or activations, and providing relevant global information. Moreover, ReNet layers are stacked on top of pre-trained convolutional layers, benefiting from generic local features. Upsampling layers follow ReNet layers to recover the original image resolution in the final predictions. To handle the issue of exploiting the context information of an image , [2] explores multi-level contextual recurrent neural networks (ML-CRNNs) which encodes three kinds of contextual cues, i.e., local context, global context and image topic context in structural recurrent neural networks (RNNs). Applying CNNs to large images is computationally expensive because the amount of computation scales linearly with the number of image pixels, [223] presents a

RNN model that is capable of extracting information from an image or video by adaptively selecting a sequence of regions or locations and only processing the selected regions at high resolution Like CNNs, the proposed model has a degree of translation invariance built-in, but the amount of computation it performs can be controlled independently of the input image size. It is well known that contextual and multi-scale representations are important for accurate visual recognition, [224] presents the Inside-Outside Net (ION), an object detector that exploits information both inside and outside the region of interest. Contextual information outside the RoI is integrated using spatial RNNs. They use skip pooling to extract information inside the RoI at multiple scales and levels of abstraction. Combining local generic features extracted by CNNs and the capacity of RNNs to retrieve distant dependencies is proposed in ReSeg [225]. ReSeg modifies and extends the image classification ReNet to perform the more challenging task of semantic segmentation. Each ReSeg layer is composed of four RNNs that sweep the image horizontally and vertically in both directions, encoding patches or activations, and providing relevant global information. Moreover, ReSeg layers are stacked on top of pre-trained convolutional layers, benefiting from generic local features. Upsampling layers follow ReSeg layers to recover the original image resolution in the final predictions.

Summary of CNN-based semantic segmentation methods is given in Table .3.1 whereas the summary of RNN-based semantic segmentation is given in Table. 3.2.

Table 3.1: Summary of highlight CNN-based semantic segmentation methods

Method	Network	Summary
SDS [202]	R-CNN	Simmultaneous detection and segmentation 7 point boost (16% relative) over baseline 5 point boost (10% relative) over state-of-the art

FCNs [31]	VGG-16	Fully convolutional networks First end-to-end network recover the resolution PASCAL VOC: 62.2% mean IoU NYUDv2: 34% meanIoU SiFT Flow: 39.5% mean IoU
SegNet [39]	VGG-16	Encoder-Decoder Encoder is identical to VGG16 Decoder uses pooling indices to perform non-linear upsampling CamVid11: 60.1 meanIoU SUNRGB-D: 31.84% meanIoU
DilationNet [40]	VGG-16	Use dilated convolutions Aggregate multi-scale information without losing resolution PASCAL VOC 2012: 67.6% meanIoU
Bayesian SegNet [41]	VGG-16	Monte Carlo sampling with dropout to generate a posterior distribution Dropout is to form a probabilistic encoder-decode architecture CamVid11: 63.1% meanIoU SUN RGB-D: 30.7% meanIoU NYUv2: 32.4% meanIoU

DeepLab [42]	VGG-16 ResNet-101	Atrous convolution control the resolution ASPP to segment object at multiple scales Combine DCNN & probabilistic graphical model to improve localization PASCAL VOC2012: 79.9 % meanIoU Pascal Context (VGG-16) 37.6%-39.6%meanIoU Pascal Context (ResNet-101): 39.6%-45.7%meanIoU Pascal-Person (ResNEt-101): 58.9%-62.76% meanIoU Cityscapes: 70.4%meanIoU
DeepMask [43]	VGG-A	Generate proposals for segmentation MS-COCO: 0.126 AR@10 PASCAL 2007: 0.337 AR@10
UNet [226]	FCN	Reply on the strong use of data augmentation Particular designed for medical images Won cell tracking challenging 2015 Implemented on Caffe
ENet [209]	ENet bottleneck	Propose bottlenet model for efficiency Real time application NVIDIA TX1: 14.6 fps(640x360) NVIDIA Titan X: 135.4 fps(640x360) Cityscapes: 58.3% Class IoU, CamVid: 51.3% Class IoU SunRGB-D: 19.7% meanIoU

PSP [44]	FCN Resnet	Use Resnet50 as baseline ImageNet scene 2016: 57.2% (meanIOU, Pixel Acc) PASCAL VOC 2012: 82.6% accuracy Cityscapes: 78.4/80.2% Class meanIoU
FCIS [45]	FCN Resnet	Inherit the merits of FCNs for semantic instance segmentation Introduce position-senstive inside/outside PASCAL VOC 2012: 65.7% mAP^r@0.5 MS-COCO: 49.5mAP^r@0.5

Table 3.2: Summary of highlight RNN-based semantic segmentation methods

Method	Network	Summary
RCNN [227]	CNNs RNN	End-to-end trainable on raw pixel Low computational compplectity Run on CPU CNNs & RNN are trained separately Stanford Background:79.8 %/ 69.3% (Pixel/Class Acc) SiftFlow: 77.7%/29.8% (Pixel/Class Acc)
2D-LSTM [32]	LSTM	4 LSTM blocks scanning all directions of an image Low computational complexity (155K parameters) no pre- or post-processing is applied Stanford Background: 78.56%/68.79% (Pixel/Class Acc) SiftFlow: 70.11%/20.9%(Pixel/Class Acc)

ReSeg [52]	ReNet FCN GRU	Built on top of ReNet CNNs Vertical RNNs is applied first and then Horizontal RNNs CNNs & RNN are trained separately Weizmann Horses:91.6% meanIoU Oxford Flowers: 93.7% meanIoU CamVid: 58.8% meanIoU
DAG-RNN [56]	VGG-16 RNN	UCG is used to model the interactions Deconvolution is trained separated CamVid: 91.6%/78.1%(Global/Class Acc) Barcelona: 74.6%/24.6%(Global/Class Acc) SiftFlow: 85.3%/55.7%(Global/Class Acc)

3.3 Common datasets

In this section, we describe the most popular large-scale datasets currently in use for elvaluating the semantic segmentation systems. Table 3.3 shows a summarized view, gathering all their properties including number of classes, type, resolution and training/validation/testing splits. Based on the purpose, the datasets are divided into three categories corresponding to generic, outdoor and urban-driving.

3.3.1 Generic datasets

Semantic Boundaries Dataset (SBD)

SBD [228] is an extended version of the PASCAL VOC 2011. This dataset provides semantic segmentation ground truth for those images that are not labelled in VOC. It contains

annotations for 11,355 images from PASCAL VOC 2011. In additional to boundary, the dataset also provides ground truth for both category-level and instance-level. Since the images are obtained from the whole PASCAL VOC challenge, the training and validation splits diverge. In fact, SBD provides its own 11,355 images which splits into 8,498 images for training and 2,857 images for validation. Due to its increased amount of training data, this dataset is often used as a substitute for PASCAL VOC for deep learning.

PASCAL Visual Object Classes (VOC)

PASCAL VOC [24] consists of a ground-truth annotated dataset of images drawn from different versions, namely, PASCAL VOC 2005, PASCAL VOC 2006, PASCAL VOC 2007, PASCAL VOC 2008, PASCAL VOC 2009, PASCAL VOC 2010, PASCAL VOC 2011, PASCAL VOC 2012 for various purposes: classification, detection, segmentation, action classification, and person layout. In the scope of this thesis, PASCAL VOC 2012 is used for evaluating and named PASCAL VOC for short. There are 20 classes (+1 background) categorized into vehicles, household, animals, and other: aeroplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa, TV/monitor, bird, cat, cow, dog, horse, sheep, and person. Background is also considered if the pixel does not belong to any of those classes. The 2012 dataset contains images from 2008-2011 for which additional segmentations have been prepared. As in previous years, the assignment to training/test sets has been maintained. Between 2008 and 2012 the total number of images (objects) in the main dataset more than doubled from 4,332 images (12,684 objects) to 11,540 images (31,561 objects). The total number of images with segmentation has been increased from 2011. The dataset is divided into two subsets: training and validation with 1,464 images for training and 1,449 images for validation respectively. The test set is private for the challenge. This dataset is arguably the most popular for semantic segmentation so almost every remarkable method in the literature is being submitted to its performance evaluation server to validate against their private test set. Methods can be trained either using only

the dataset or either using additional information.

PASCAL Context

PASCAL Context [229] is an extension of the PASCAL VOC 2010 dataset for detection challenge. It goes beyond the original PASCAL semantic segmentation task by providing annotations for the whole scene. This dataset contains 10,103 images with pixel-wise labels for training and 9,637 images for testing with a total of 540 classes including the original 20 classes plus background from PASCAL VOC segmentation divided into three categories (objects, stuff, and hybrids). Since the dataset follows a power law distribution, there are many sparse classes. In practical, only the 59 most frequent are remarkable and is usually selected to conduct studies on this dataset, relabeling the rest of them as background.

PASCAL Part

PASCAL Part [230] is an extension of the PASCAL VOC 2010 dataset for detection challenge. This dataset goes beyond the original PASCAL VOC object detection task by providing segmentation masks for each body part of the object. For categories that do not have a consistent set of parts (e.g., boat), it provides the silhouette annotation. The original classes of PASCAL VOC are kept, but their parts are introduced, e.g., bicycle is now decomposed into back wheel, chain wheel, front wheel, handlebar, headlight, and saddle. It contains labels for all 10,103 images for training and validation images from PASCAL VOC as well as for the 9,637 testing images.

Microsoft Common Objects in Context (COCO)

MS COCO [25] is another image recognition, segmentation, and captioning large-scale dataset. It features various challenges, being the detection one the most relevant for this field since one of its parts is focused on segmentation. MS COCO is splitted into two equal parts: the first half of the dataset was released in 2014, the second half will be released

in 2015. The MS COCO 2014 contains 82,783 training, 40,504 validation, and 40,775 testing images. The 2014 dataset is annotated with 80 classes (+1 background). There are nearly 270k segmented people and a total of 886k segmented object instances in the MS COCO 2014 train+val data alone. The cumulative 2015 release will contain a total of 165,482 train, 81,208 val, and 81,434 test images. In particular, the test set is divided into four different subsets or splits: test-dev (20,000 images) for additional validation, debugging, test-standard (20,000 images) is the default test data for the competition and the one used to compare state-of-the-art methods, test challenge (20,000 images) is the split used for the challenge when submitting to the evaluation server, and test-reserve (20,000 images) is a split used to protect against possible overfitting in the challenge (if a method is suspected to have made too many submissions or trained on the test data, its results will be compared with the reserve split). Its popularity and importance has ramped up since its appearance thanks to its large scale. The new release MS COCO 2017 contains pixel-level segmentation of object belonging to 80 categories, keypoint annotations for person instances, stuff segmentations for 91 categories, and five image captions per image. The challenges in this dataset are (1) object detection with bounding boxes and segmentation masks, (2) joint detection and person keypoint estimation, and (3) stuff segmentation.

3.3.2 Urban - driving

Cityscape

Cityscape [231] is a large-scale database which focuses on semantic understanding of urban street scenes. It provides semantic, instance-wise, and dense pixel annotations for 30 classes grouped into 8 categories, i.e. flat surfaces (road, sidewalk, parking, rail track) , humans (person, rider), vehicles (car, truck, bus, on rails, motorcycle, bicycle, caravan, trailer), constructions (building, wall, fence, guard rail, bridge, tunnel), objects (pole, pole group, traffic sign, traffic light) , nature (vegetation, terrain), sky (sky), and void (ground,

dynamic, static). The dataset consist of around 5,000 fine annotated images and 20,000 coarse annotated ones. Data was captured in 50 cities during several months, daytimes, and good weather conditions. It was originally recorded as video so the frames were manually selected to have the following features: large number of dynamic objects, varying scene layout, and varying background.

CamVid

Camvid dataset [232] is a road/driving scene understanding database which was originally captured as five video sequences with a 960 $\times$ 720 resolution camera mounted on the dashboard of a car. Those sequences are sampled (four of them at 1 fps and one at 15 fps) adding up to 701 frames. This dataset has been originally designed for the problem of automated driving vehicle. This sequence depicts a moving driving scene in the city of Cambridge filmed from a moving car. It is a challenging dataset since, in addition to the car ego-motion, other cars, bicycles and pedestrians have their own motion and they often occlude one another. This data is manually annotated with 32 classes: void, building, wall, tree, vegetation, fence, sidewalk, parking block, column/pole, traffic cone, bridge, sign, miscellaneous text, traffic light, sky, tunnel, archway, road, road shoulder, lane markings (driving), lane markings (non-driving), animal, pedestrian, child, cart luggage, bicyclist, motorcycle, car, SUV/pickup/truck, truck/bus, train, and other moving object. In most of the research work, 11 common classes are used, i.e. building, tree, sky, car, sign, road, pedestrian, fence, pole, sidewalk, and bicyclist.

KITTI

KITTI dataset [233] is one of the most popular datasets for use in mobile robotics and autonomous driving. The tasks of interest are tereo, optical flow, visual odometry, 3D object detection and 3D tracking. It consists of hours of traffic scenarios recorded with a variety of sensor modalities, including high resolution RGB, grayscale stereo cameras, and

a 3D laser scanner. Despite its popularity, the dataset itself does not contain ground truth for semantic segmentation. However, various researchers have manually annotated parts of the dataset to fit their necessities. Up to 15 cars and 30 pedestrians are visible per image which is captured by driving around the mid-size city of Karlsruhe, in rural areas and on highways. Ros et al. [234] labeled 170 training images and 46 testing images (from the visual odometry challenge) with 11 classes: building, tree, sky, car, sign, road, pedestrian, fence, pole, sidewalk, and bicyclist.

3.3.3 Outdoor Scene Datasets

Stanford background

Stanford dataset [235] contains outdoor scene images imported from existing public datasets: LabelMe, MSRC, PASCAL VOC and Geometric Context. The dataset contains 715 images (size of 320 $\times$ 240 pixels) with at least one foreground object and having the horizon position within the image. The dataset is pixel-wise annotated (horizon location, pixel semantic class, pixel geometric class and image region) for evaluating methods for semantic scene understanding.

SiftFlow

Siftlow dataset [236] contains 2,688 fully annotated images which are a subset of the LabelMe database. Most of the images are based on 8 different outdoor scenes including streets, mountains, fields, beaches and buildings. Images are 256 $\times$ 256 belonging to one of the 33 semantic classes. Unlabeled pixels, or pixels labeled as a different semantic class are treated as unlabeled.

Table 3.3: Summary of the most popular large-scale datasets used for evaluating the performance of semantic segmentation

Datasets	Year	Classes	Type	Resolution	Training	Validation	Testing
SBD [228]	2011	21	Generic	Variable	8,498	2,857	N/A
PASCAL VOC 2012 [24]	2012	21	Generic	Variable	1,464	1,449	N/A
PASCAL-Context [229]	2014	540	Generic	Variable	10,103	N/A	9,637
PASCAL-Part [237]	2014	20	Generic	Variable	10,103	N/A	9,637
MS COCO [25]	2014	81	Generic	Variable	82,783	40,504	81,434
Cityscapes-FINE [231]	2015	30	Urban	2048x1024	2,975	500	1,525
Cityscapes-COARSE [231]	2015	30	Urban	2048x1024	22,973	500	N/A
Camvid [232]	2009	32	Urban	960x720	701	N/A	N/A
Camvid - Sturgess [238]	2009	11	Urban	960x720	367	100	233
KITTI-LAYOUT [233]	2012	3	Urban	Variable	323	N/A	N/A
KITTI-ROSS [234]	2015	11	Urban	Variable	170	N/A	46
Standford [235]	2009	8	Scene	320x240	725	N/A	N/A
Siftflow [236]	2011	33	Scene	256x256	2,688	N/A	N/A

Chapter 4

Contextual Recurrent Level Set (CRLS) Networks

4.1 Introduction

Variational Level Set (LS) has been a widely used method in medical segmentation. However, it is limited when dealing with multi-instance objects in the real world. In addition, its segmentation results are quite sensitive to initial settings and highly depend on the number of iterations. To address these issues and boost the classic variational LS methods to a new level of the learnable deep learning approaches, we propose a novel definition of contour evolution named *Recurrent Level Set (RLS)* to employ Gated Recurrent Units under the energy minimization of a variational LS functional. The curve deformation process in RLS is formed as a hidden state evolution procedure and updated by minimizing an energy functional composed of fitting forces and contour length. By sharing the convolutional features in a *fully end-to-end trainable framework*, we extend RLS to *Contextual RLS (CRLS)* to address semantic segmentation in the wild. The experimental results have shown that our proposed RLS improves both computational time and segmentation accuracy against the classic variational LS-based method whereas the fully end-to-end sys-

tem CRLS achieves competitive performance compared to the state-of-the-art semantic segmentation approaches.

To the best of our knowledge, this is the first study on formulating LS-based method under a learnable framework. The main contributions of our work can be summarized as follows:

- Bridging the gaps between two unconnected areas: the pure image processing variational LS methods and learnable deep neural networks approaches.
- Proposing a new formulation of curve evolution under an end-to-end deep learning framework by reforming the optimization process of CLS as a current network, named RLS.
- The proposed RLS framework is formed as a building block which is easily incorporated with other existing deep modules to provide a robust deep semantic segmentation, named CRLS.

In this chapter, we first introduce the proposed RLS which partitions an image into foreground and background segments. Before detailing the RLS learning and inference, we study the coherency between CLS and RNNs/GRU. We then describe how to extend RLS to CLRLS to solve the problem of semantic instance segmentation.

4.2 Recurrent Level Set (RLS)

4.2.1 Relationship between RNNs/GRUs and CLS

In order to reformulate CLS into a deep learning framework, we first study how CLS communicates with a RNNs/GRUs framework. For convenience, we first use the simple form of RNNs to study the correlation between deep learning and CLS. The unfolding in time of the computation involved in its forwards procedure of both RNNs (upper) and CLS (lower) is given in Fig. 4.1. It is easy to see that they both are time sequence process.

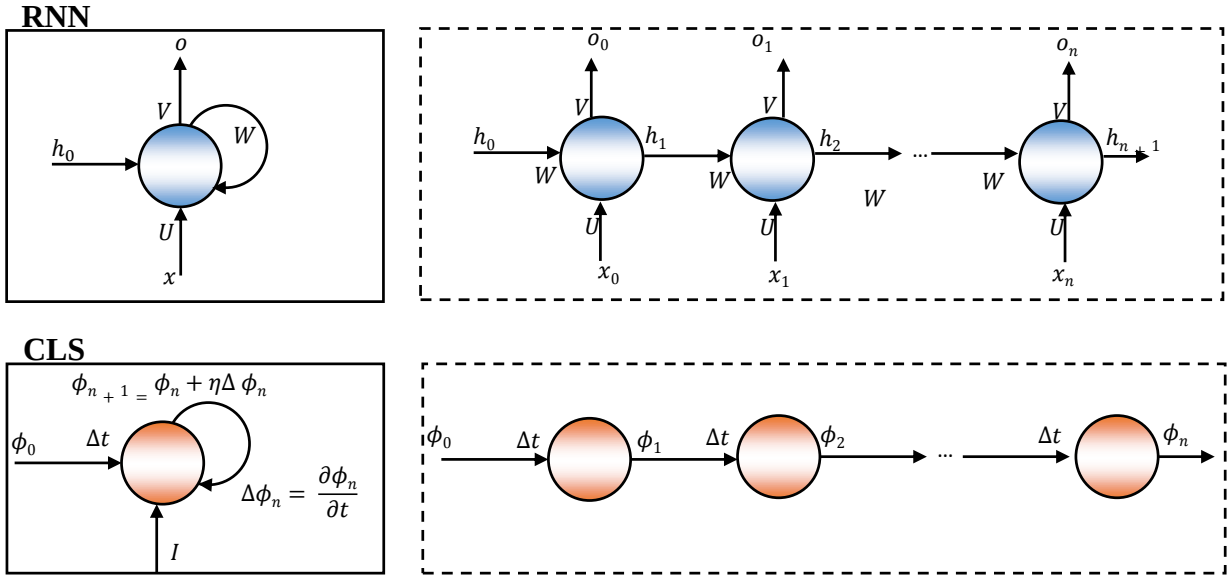


Figure 4.1: The unfolding in time of the computation involved in its forwards procedure of RNNs (upper) and CLS (lower)

However, the updating rule, input and output of each model are different.

To reformulate Level Set under a RNNs/GRUs deep framework, we need to solve the following problems:

- **Input:** How to generate a sequential data $\mathbf{x}_t$ from a single image $\mathbf{I}$ if we treat ϕ_t in CLS as hidden state h_t in RNNs with the same updating manner?
- **Output:** In the segmentation problem by CLS, the binary mask corresponding to foreground and background is computed by one forward process with many iterations whereas the RNNs is computed by forward procedure and learnt by backward procedure. How to compute the error and perform backward procedure in CLS framework?
- **Update Rule:** Does the update rule in CLS work in the same fashion as the forward procedure in RNNs/GRUs deep framework?

The aforementioned questions will be answered in our proposed RLS which is detailed in the following subsection.

Table 4.1: Comparison between CLS, GRUs and our proposed RLS

	Input	Update	Output
CLS	Image $\mathbf{I}$ Initial LS function ϕ_0	$\phi_{t+1} = \phi_t + \eta\frac{\partial \phi_t}{\partial t}$ $\frac{\partial \phi_t}{\partial t} = \delta_\epsilon(\phi_t[\nu\kappa(\phi_t - \mu$ $\lambda_1(\mathbf{I} - c_1)^2 + \lambda_2(\mathbf{I} - c_2)^2])$	ϕ_N
GRUs	Sequence $x_1, x_2, ...x_N$ Initial hidden state h_0	$\mathbf{z}_t = f(\mathbf{U}_z\mathbf{x}_t + \mathbf{W}_z\mathbf{h}_{t-1})$ $\mathbf{r}_t = f(\mathbf{U}_r\mathbf{x}_t + \mathbf{W}_r\mathbf{h}_{t-1})$ $\tilde{\mathbf{h}}_t = h(\mathbf{U}_h\mathbf{x}_t + \mathbf{W}_h(\mathbf{h}_{t-1} \circ \mathbf{r}_t))$ $\mathbf{h}_t = (1 - \mathbf{z}_t)\mathbf{h}_{t-1} + \mathbf{z}_t\tilde{\mathbf{h}}_t$ h: Tanh f: Sigmoid or Tanh	$\mathcal{G}(\mathbf{V}\mathbf{h}_N)$ $\mathcal{G}$: softmax
RLS	Image $\mathbf{I}$ Initial LS function ϕ_0	$\mathbf{x}_t = \kappa(\phi_{t-1} - \mathbf{U}_g(\mathbf{I} - c_1)^2 + \mathbf{W}_g(\mathbf{I} - c_2)^2)$ $\mathbf{z}_t = \sigma(\mathbf{U}_z\mathbf{x}_t + \mathbf{W}_z\phi_{t-1} + \mathbf{b}_z)$ $\mathbf{r}_t = \sigma(\mathbf{U}_r\mathbf{x}_t + \mathbf{W}_r\phi_{t-1} + \mathbf{b}_r)$ $\tilde{\mathbf{h}}_t = h(\mathbf{U}_y\mathbf{x}_t + \mathbf{W}_y(\phi_{t-1} \circ \mathbf{r}_t) + \mathbf{b}_y)$ $\phi_t = (1 - \mathbf{z}_t)\phi_{t-1} + \mathbf{z}_t\tilde{\mathbf{h}}t$ h: Tanh σ: Sigmoid	$\mathcal{G}(\mathbf{V}\phi_N + b_V)$ $\mathcal{G}$: softmax

4.2.2 Proposed Recurrent Level Set (RLS)

In this section, we take the classical variational LS (CLS) evolution introduced by [62] as an instance to demonstrate the idea of how to reformulate LS as an end-to-end trainable recurrent framework, named RLS. However, the proposed RLS can be applied to reform any LS approach once it successfully reforms CLS model because they share similar properties of curves moving over time. In our proposed RLS approach, the recurrent units work in the same fashion as GRUs.

As shown in Table 4.1, the first difficult part of reformulating CLS as recurrent network is data configuration. RNNs work on sequential data while both the input and the output of the CLS approach are single images. The critical question is *how to generate sequence data from a single image*. Notably, there are two inputs used in CLS, i.e. an input image $\mathbf{I}$ and an initial LS function ϕ_0, which is updated by Eq. (2.65). In our proposed RLS, we need to generate a sequence data $\mathbf{x}_t$ $(t = 1, \cdots, N)$ from single image $\mathbf{I}$. In order to achieve this goal, we define a function $g(\mathbf{I}, \phi_{t-1})$ as in Eq. (4.1).

$$\mathbf{x}_t = g(\mathbf{I}, \phi_{t-1}) = \phi_{t-1} + \eta \left[\kappa(\phi_{t-1}) - \mathbf{U}_g(\mathbf{I} - c_1)^2 + \mathbf{W}_g(\mathbf{I} - c_2)^2\right] \tag{4.1}$$

In Eq. (4.1), c_1 and c_2 are average values of inside and outside of the contour presented by the LS function ϕ_{t-1} and defined in Eq. (2.64). κ denotes the curvature and defined in Eq. (2.63). $\mathbf{U}_g$ and $\mathbf{W}_g$ are two matrices that control the force inside and outside of the contour.

Clearly, during the curve evolution, the input at iteration t^{th}, $\mathbf{x}_t$, is updated based on the input image $\mathbf{I}$ and the previous LS function ϕ_{t-1} which is in the same fashion as in LS and defined by Eq. (2.62). In our proposed RLS, $\mathbf{x}_t$ is considered to be input sequence whereas LS function ϕ_t is treated as hidden state. Notably, the initial LS function plays the role as initial hidden state. The relationship between $\mathbf{I}$, $\mathbf{x}_t$, and ϕ_t are given in Fig. 4.2.

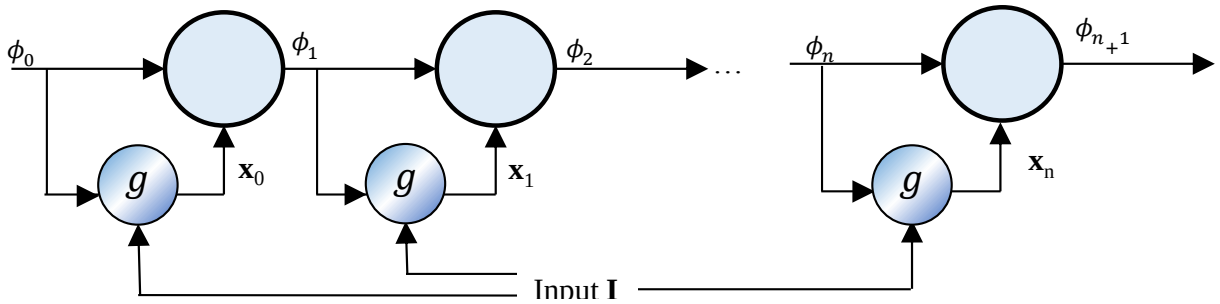


Figure 4.2: The visualization of generating sequence input data $\mathbf{x}_t$ and hidden state ϕ_t from the input image $\mathbf{I}$ and the initial LS function ϕ_0

So far, we have answered the question of generating input sequence from the single image $\mathbf{I}$. That means we use the same input as defined in LS problem, namely, the input image $\mathbf{I}$ and the initial ϕ_0. From the input, we are able to generate sequential data $\mathbf{x}_t$. The next important task is generating the hidden state ϕ_t from the input data $\mathbf{x}_t$ and the previous hidden state ϕ_{t-1}. Under the same intuition of proposing GRUs [128], the procedure of generating hidden state ϕ_t is based on the updated gate $\mathbf{z}_t$, the candidate

memory content $\tilde{\mathbf{h}}_t$ and the previous activation unit ϕ_{t-1} as the rule given in Eq. (4.2).

$$\phi_t = \mathbf{z}_t \tilde{\mathbf{h}}_t + (1 - \mathbf{z}_t)\phi_{t-1} \tag{4.2}$$

The update gate $\mathbf{z}_t$, which controls how much of the previous memory content is to be forgotten and how much of the new memory content is to be added is defined as in Eq. (4.3).

$$\mathbf{z}_t = \sigma(\mathbf{U}_z \mathbf{x}_t + \mathbf{W}_z \phi_{t-1} + \mathbf{b}_z) \tag{4.3}$$

where σ is a sigmoid function and $\mathbf{b}_z$ is the update bias. The RLS, however, does not have any mechanism to control the degree to which its state is exposed, but exposes the whole state each time. The new candidate memory content $\mathbf{y}_t$ is computed. as in Eq. (4.4).

$$\tilde{\mathbf{h}}_t = tanh(\mathbf{U}_{\tilde{h}} \mathbf{x}_t + \mathbf{W}_{\tilde{h}} (\phi_{t-1} \odot \mathbf{r}_t) + \mathbf{b}_{\tilde{h}}) \tag{4.4}$$

where $\odot$ denotes an element-wise multiplication, $\mathbf{b}_{\tilde{h}}$ is the hidden bias. The reset gate $\mathbf{r}_t$ is computed similarly to the update gate as in Eq. (4.5).

$$\mathbf{r}_t = \sigma(\mathbf{U}_r \mathbf{x}_t + \mathbf{W}_r \phi_{t-1} + \mathbf{b}_r) \tag{4.5}$$

where $\mathbf{b}_r$ is the reset bias. When $\mathbf{r}_t$ is close to 0 (off), the reset gate effectively makes the unit act as if it is reading the first symbol of an input sequence, allowing it to forget the previously computed state. The output $\mathbf{o}$ is computed from the current hidden states ϕ_t and then a softmax function is applied to obtain foreground/background segmentation $\hat{\mathbf{y}}$ given the input image as follows:

$$\begin{aligned}\hat{\mathbf{y}} &= \text{softmax}(\mathbf{o}) \\ \mathbf{o} &= \mathbf{V}\phi_t + \mathbf{b}_V\end{aligned} \tag{4.6}$$

where $\mathbf{V}$ is weighted matrix between hidden state and output. The proposed RLS model is trained in an end-to-end framework and its learning processes of the forward pass and the backward propagation are described as follows.

The proposed RLS in folded mode is given in Fig. 4.3 (left) where the input of the network is defined as the same as the CLS model, namely, an input image $\mathbf{I}$ and an initial LS function ϕ_0. The LS function is initialized in a similar fashion as in [62] via checkerboard function . In our proposed RLS, the curve evolution from ϕ_{t-1} at time $t-1$ to the next step ϕ_t at time t is designed in the same fashion as the hidden state in GRUs and is illustrated in Fig. 4.3 (right) where ϕ_t depends on both ϕ_{t-1} and the input $\mathbf{x}_t$. We have summarized CLS, GRUs and our proposed RLS in Table 4.1. It is easy to see that RLS shares the same input as CLS while updating procedure and output in the proposed RLS follows similar fashion as in GRUs. With such design, the next parts show how to train the proposed RLS.

4.2.3 RLS Learning

The Back Propagation Through Time (BPTT) method is used to train the parameter set $\theta = \{ \mathbf{U}_g, \mathbf{W}_g, \mathbf{U}_z, \mathbf{W}_z, \mathbf{U}_r, \mathbf{W}_r, \mathbf{U}_{\tilde{h}}, \mathbf{W}_{\tilde{h}}, \mathbf{V} \}$ and propagate error backward through time[1]. We apply RMS-prop [239] with momentum $\rho_m = 0.9$. This optimizer minimizes the following cross entropy loss function.

$$L_1(\hat{\mathbf{y}}, \mathbf{y}, \theta) = -\sum_{n=1}^{N}\sum_{k=1}^{K} \mathbf{y}_{nk} log(\hat{\mathbf{y}}_{nk}) \tag{4.7}$$

[1]The derivatives of all parameters are detailed in Appendix

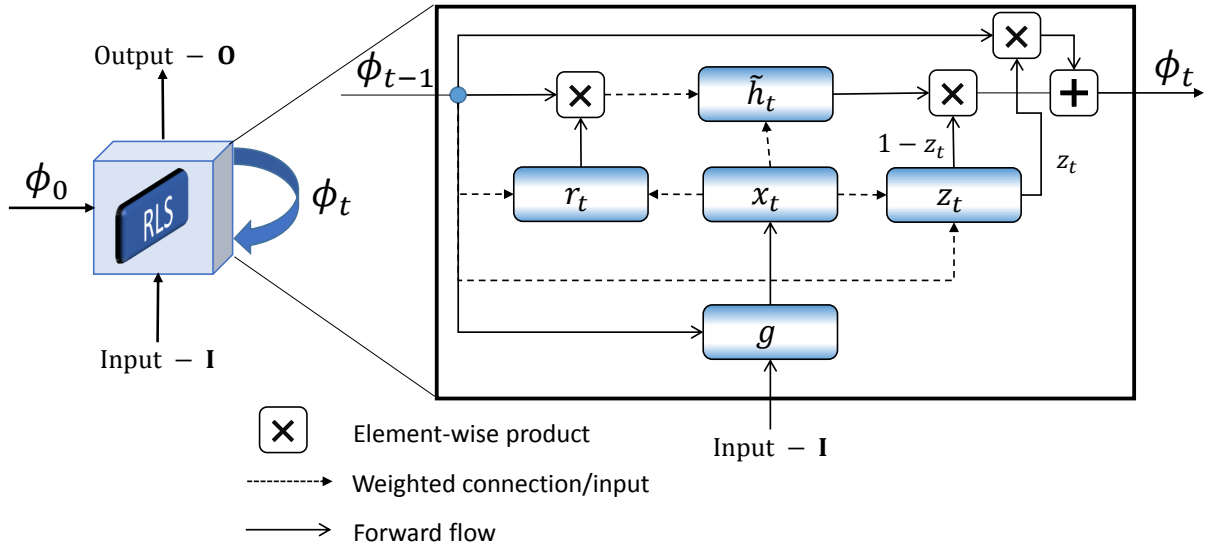


Figure 4.3: The proposed RLS network for curve updating process under the sequential evolution and its forward computation of curve evolution from time $t-1$ to time t.

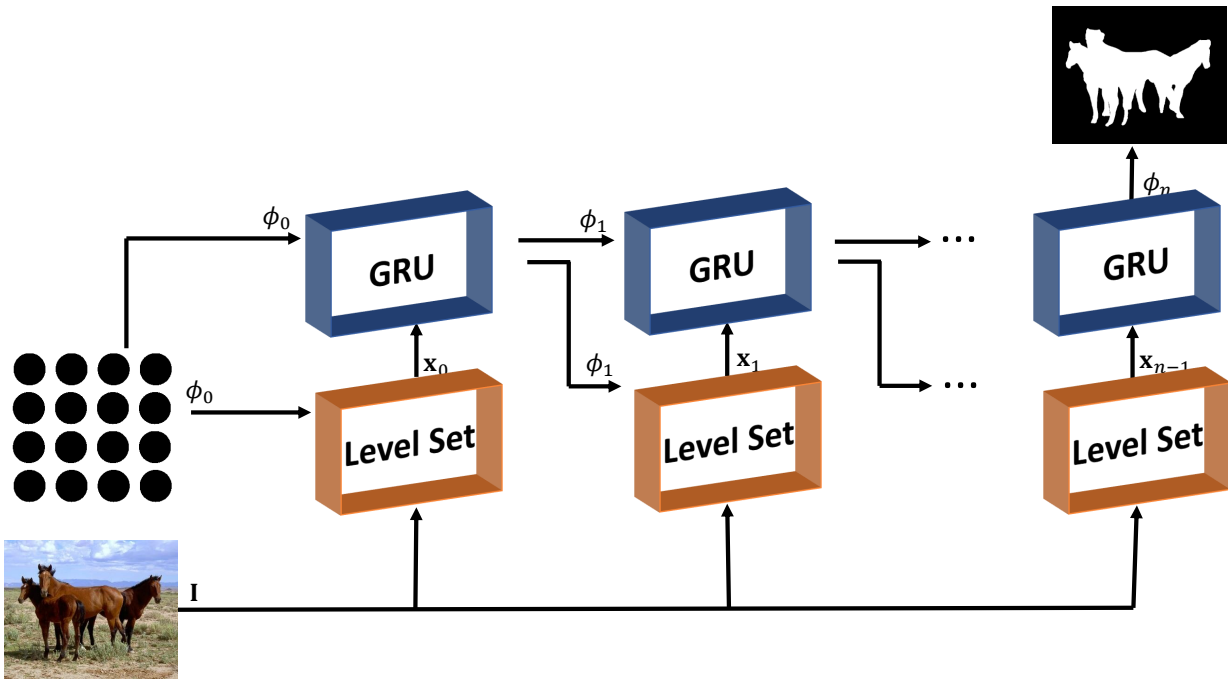


Figure 4.4: The unfolding of the proposed RLS network for curve updating process under recurrent fashion.

where N is the number of pixels, K is the number of classes ($K = 2$), since we only segment foreground/background. $\hat{\mathbf{y}}_{nk}$ is our predictions, and $\mathbf{y}_{nk}$ is the ground truth. The gradients

Algorithm 3 The proposed building block RLS

Input: Given an image $\mathbf{I}$, an initial level set function ϕ_0, time step N, learning rate η, initial parameters $\theta = (\mathbf{U}_z, \mathbf{W}_z, \mathbf{b}_z, \mathbf{U}_r, \mathbf{W}_r, \mathbf{b}_r\ \mathbf{U}_{\tilde{h}}, \mathbf{W}_{\tilde{h}}, \mathbf{b}_{\tilde{h}}, \mathbf{V}, \mathbf{b}_V)$
for each epoch **do**
  Set $\phi = \phi_0$
  for t = 1 : T **do**
    Generate RLS input $\mathbf{x}_t$: $\mathbf{x}_t \leftarrow g(\mathbf{I}, \phi_{t-1})$
    Compute update gate $\mathbf{z}_t$, reset gate $\mathbf{r}_t$ and intermediate hidden unit $\tilde{\mathbf{h}}_t$:
    $\mathbf{z}_t \leftarrow \sigma(\mathbf{U}_z\mathbf{x}_t + \mathbf{W}_z\phi_{t-1} + \mathbf{b}_z)$
    $\mathbf{r}_t \leftarrow \sigma(\mathbf{U}_r\mathbf{x}_t + \mathbf{W}_r\phi_{t-1} + \mathbf{b}_r)$
    $\tilde{\mathbf{h}}_t \leftarrow tanh(\mathbf{U}_{\tilde{h}}\mathbf{x}_t + \mathbf{W}_{\tilde{h}}\,(\phi_{t-1} \circ \mathbf{r}_t) + \mathbf{b}_{\tilde{h}})$
    Update the zero LS ϕ_t: $\phi_t \leftarrow (1 - \mathbf{z}_t)\tilde{\mathbf{h}}_t + \mathbf{z}_t\phi_{t-1}$
  end for
  Compute the loss function L: $L \leftarrow -\sum_n \mathbf{y}_n log\hat{\mathbf{y}}_n$
  Compute the derivate w.r.t. θ: $\nabla\theta \leftarrow \frac{\partial L}{\partial \theta}$
  Update θ: $\theta \leftarrow \theta + \eta\nabla\theta$
end for

of the error with respect to our parameters are learned using Stochastic Gradient Descent (SGD) and the chain rule of differentiation. Normalized and smoothed gradients of similar size for all weights are used such that even weights with small gradients get updated. This also helps to deal with vanishing gradients. We train our model using a decaying learning rate starting from $\eta = 10^{-3}$ and dividing by half every 200 epochs asymptotically toward $\eta = 10^5$ for 5000 epochs in total.

We summarize the proposed building block RLS in Algorithm 3. Unfolding in time of RLS under recurrent fashion is given in Fig. 4.4 where the output from GRUs component plays as the input of LS component and the output from LS component is the input of GRUs component.

4.3 Contextual Recurrent Level Sets (CRLS)

The proposed RLS described in the previous section performs the image segmentation task, i.e. dividing an image into two parts corresponding foreground and background segments.

Given a real-world image, RLS however performs neither the instance segmentation nor image understanding which are significant tasks in many computer vision application. In order solve this requirement, we introduce Contextual Recurrent Level Sets (CRLS) for semantic object segmentation which is an extension of our proposed RLS model to address the multi-instance object segmentation in the wild. The proposed CRLS is able to (1) localize objects existing in the given image; (2) segment the objects out of the background; (3) classify the objects in the image. The output of our CRLS is multiple values (each value is corresponding to one object class) instead of two values (foreground and background) as in RLS. The entire proposed CRLS modle is first introduced in Sec.4.3.1. Inference and training process are then described in Sec.4.4.

4.3.1 Model constructing

Our proposed CRLS inherits the merits of RLS and Faster-RCNN [100] for semantic segmentation which simultaneously performs three tasks which are addressed by three stages, i.e. detection, segmentation and classification in a fully end-to-end trainable framework as shown in Fig. 4.5. In our CRLS, the network takes an image of arbitrary size as the input, and outputs instance-aware semantic segmentation results. The network contains three components corresponding three stages of a semantic instance segmentation: object detection for proposing box-level, object segmentation is for mask-level and object classification is for catergorizing each instance . These three stages are designed to share convolutional features. Each stage involves a loss term and the loss of later stage relies on the output of an predecessor stage, so the three loss terms are not independent. We train the entire network end-to-end with a unified loss function.

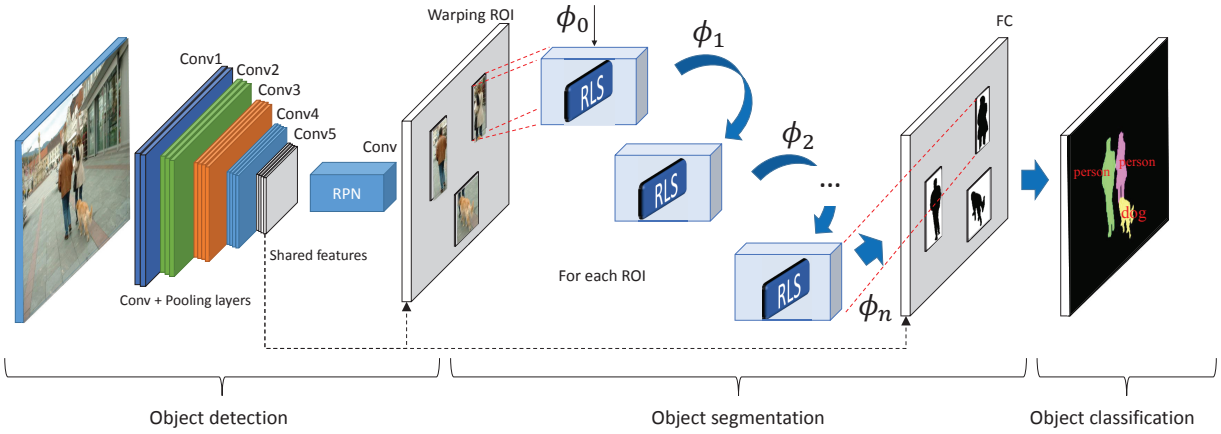


Figure 4.5: The flowchart of our proposed CRLS for semantic instance segmentation with three tasks which are addressed by three stages , i.e. object detection by Faster R-CNN [100], object segmentation by RLS and classification

Stage 1: Object detection

One of the most important approaches to the object detection and classification problems is the generations of Region-based Convolutional Neural Networks (R-CNN) family [30, 100, 159]. Aiming to design a *fully end-to-end trainable* framework, we adapt the Region Proposal Network (RPN) introduced in Faster R-CNN [100] to predict the object bounding boxes and the objectness scores. By sharing the convolutional features of a deep VGG-16 network [157], the whole system is able to perform both detection and segmentation computation efficiently.

As for the share convolutional features, we utilize a VGG-16 networks with 13 convolution layers where each convolution layer is followed by a ReLU layer but only four pooling layers are placed right after the convolution layer to reduce the spatial dimension. As shown in Fig. 4.6, the feature visualization is divided into 5 main components, i.e. *conv1*, *conv2*, *conv3*, *conv4* and *conv5*, by those pooling layers. Each component consists of 2 or 3 convolution layers.

In this stage, object detection, the network proposes object instances in the form of bounding boxes which are predicted with an objectness score. The network structure and loss function of this stage follow the work of Region Proposal Networks (RPNs) [100]. RPNs

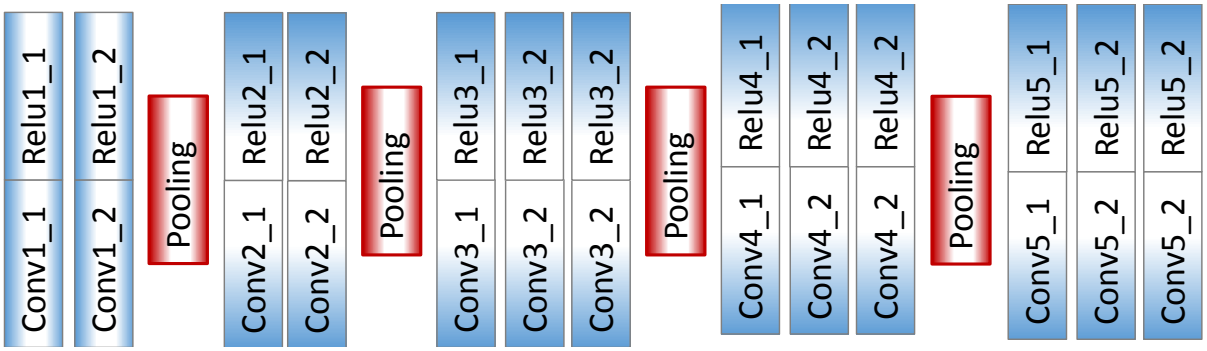


Figure 4.6: The VGG-16 based network for extracting share features

take an image of any size as the input and predicts bounding box locations and objectness scores in a fully-convolutional form. A 3×3 convolutional layer for reducing the dimension is applied on top of share convolutional features. The lower dimension features are fed into two sibling 1×1 convolutional layers: one is for a box-regression layer and the other is for box-classification layer. An illustration of RPNs is given in Fig. 4.7.

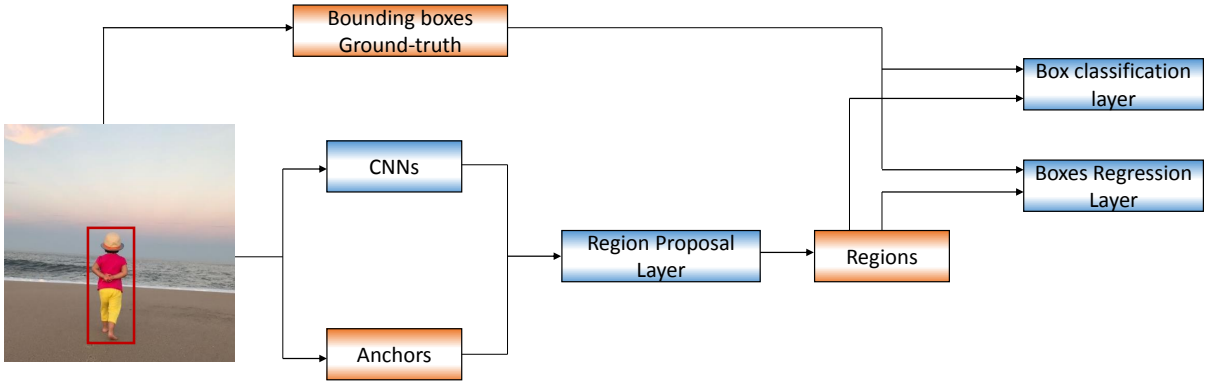


Figure 4.7: Architecture of Region Proposal Network (RPN)

From the share feature maps and at each sliding-window location, multiple region proposals are generated. Denote k is the maximum possible proposals for each location. Each box is presented by four values corresponding to locations (x, y w, h) and two scores corresponding the probability of object or not object. An anchor is centered at the sliding window and is associated with a scale and aspect ratio. We use three scales and three aspect ratios, yielding k = 9 anchors at each sliding position. For the share feature maps

of size W×H, there are k×W×H anchors. For the purpose of object detection, there are only two anchors considered:

- *Positive anchor*: an anchor with the highest Intersection-over-Union (IoU) overlap with a ground-truth box, or an anchor that has an IoU overlap higher than 0.7 with any ground-truth box.
- *Negative anchor*: an anchor with IoU ratio is lower than 0.3 for all ground-truth boxes.
- Anchors that are neither positive nor negative do not contribute to the training objective.

There are two sibling output layers: probability of classification and bounding box regression in the end of RPNs, therefore, the RPNs loss is based on two losses: boxes regression loss and object classification loss.

$$L_{RPN}(p^i, b^i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p^i, p^{i*}) + \lambda \frac{1}{N_{reg}} \sum_i p^{i*} L_{reg}(b^i, b^{i*}) \tag{4.8}$$

In this equation:

- i is the index of an anchor in a mini-batch.
- p^i is the predicted probability of anchor i being an object and p^i is computed by a softmax over the K outputs of a fully connected layer (K categories).
- N_{cls} and N_{reg} are the two normalization terms.
- λ is a balancing parameter.
- p^{i*} is groundtruth and it sets as 1 if the anchor is positive, and 0 if the anchor is negative.
- $b^i = (b^i_x, b^i_y, b^i_w, b^i_h)$ is a vector representing the four parameterized coordinates of the predicted bounding box.
- $b^{i*} = (b^{i*}_x, b^{i*}_y, b^{i*}_w, b^{i*}_h)$ is groundtruth bounding of positive box and is a vector repre-

senting the four parameterized coordinates. Let denote $(x^{i*}, y^{i*}, w^{i*}, h^{i*})$, (x^i, y^i, w^i, h^i), (x^a, y^a, w^a, h^a) are four coordinations (box center, width, and height) of groundtruth box, predicted box and anchor box, the parameterization of the four coordinates are as follows:

$$\begin{aligned} b_x^{i*} &= (x^{i*} - x^a)/w^a \quad b_x^i = (x^i - x^a)/w^a \\ b_y^{i*} &= (y^{i*} - y^a)/h^a \quad b_y^i = (y^i - y^a)/h^a \\ b_w^{i*} &= log(w^{i*}/w^a) \quad b_w^i = log(w^i/w^a) \\ b_h^{i*} &= log(h^{i*}/h^a) \quad b_h^i = log(h^i/h^a) \end{aligned} \tag{4.9}$$

- The bounding boxes regression is defined by $smooth_{l_1}$ as follows:

$$L_{reg} = \sum_{u \in x,y,w,h} smooth_{l_1}(b_u^i - b_u^{i*}) \tag{4.10}$$

$$smooth_{l_1}(x) = \begin{cases} \frac{1}{2}x^2 & if \quad |x| < 1 \\ |x| - 0.5 & otherwise \end{cases} \tag{4.11}$$

The classification loss (L_{cls}) is log loss over two classes (object vs. not object). The RPNs loss is defined as in Eq.4.12 where i is the ground truth class.

$$L_{cls}(p^i, p^{i*}) = -log(p^i). \tag{4.12}$$

To easy to follow, we denote the loss of this stage as:

$$L_{RPN} = L_{RPN}(B(\Theta)) \tag{4.13}$$

where Θ represents all network parameters to be optimized. B is the network output of this stage, representing a list of bounding boxes $B = \{b\}_i$. Each bounding box is presented by four coordinations (box center, width, and height) and predicted objectness probability $b_i = (b_x, b_y, b_w, b_h, p_i)_i$.

Stage 2: Object segmentation

The second stage takes the share features and the results from the first stage (predicted box) as inputs. The output of this stage is binary segmentation which contains foreground (obje ct) and background.

For each predicted box, we first extract a fixed-size ($m \times m$) from the deep feature map (conv5) via *RoI warping layer* which crops and warps a region on the feature map into the target fixed size by interpolation. For this stage, we set $m = 21$. RoI warping layer is actually a max pooling to convert features inside any valid region of interest into a small feature map. Each RoI is defined by a four-tuple (r, c, h, w) that specifies its top-left corner (r, c) and its height and width (h, w). RoI max pooling first partitions the feature map sized $h \times w$ into $m \times m$ grid of sub-windows of approximate size $\frac{h}{m} \times \frac{w}{m}$. The max pooling is then applied into each sub-window. As a result, RoI warping layer outputs a grid cell.

The fixed size extracted features are passed through the proposed RLS together with a randomly initial ϕ_0 to generate a sequence input data $\mathbf{x}_t$ based on Eq. (4.1). The curve evolution procedure is performed via LS updating process given in Eqs. (2.62) and (2.65). This task outputs a binary mask as given in Eq. (4.6) sized $m \times m$ and parameterized by an m^2 dimensional vector.

Given a set of predicted bounding box from the first stage, the loss term L_{SEG} of the second stage for foreground segmentation is given by:

$$L_{SEG} = L_{SEG}(S(\Theta)|B(\Theta)) \tag{4.14}$$

Here, S is the network designed by RLS and is presenting a list of segmentations $S = \{S\}_i$. Each segmentation S_i is an $m \times m$ mask.

Stage 3: Object classification via fully-connected network

The third stage takes the share feature, bounding box from the first stage, object segmentation from the second stage as inputs. The output of this stage is class score for each object instance.

For each predicted bounding box from the first stage, we first extract the feature by ROI pooling (f_{ROI}). The feature is then go through the proposed RLS and presented by a binary mask ($f_{SEG}(\Theta)$) from the second stage. The input of this stage (f_{MSK}) is the masked feature which depends on the segmentation results ($f_{SEG}(\Theta)$) and ROI feature (f_{ROI}) and computed by element-wise product of those, $f_{MSK}(\Theta) = f_{ROI}(\Theta) * f_{SEG}(\Theta$. To predict the category of a region, we first apply two fully-connected layers to masked feature f_{MSK}. As a result, we receive mask-based feature vector which is then concatenated with another box-based feature vector to build a joined feature vector. Notably, the box-based feature vector is computed by applying two fully-connected layers to ROI feature f_{ROI}. Finally, two fully-connected layers are attached to the joined feature and each gives class scores and refined bounding boxes using softmax classification of $(K+1)$ classes (including background). The entire procedure of Stage 3 is illustrated in Fig. 4.8.

Let C is the network output of this stage and representing a list of category predictions for all instances: $C = \{C\}_i$. The loss term L_{CLS} of the third stage is expressed in Eq. 4.15.

$$L_{CLS} = L_{CLS}(C(\Theta)|B(\Theta), S(\Theta)) \tag{4.15}$$

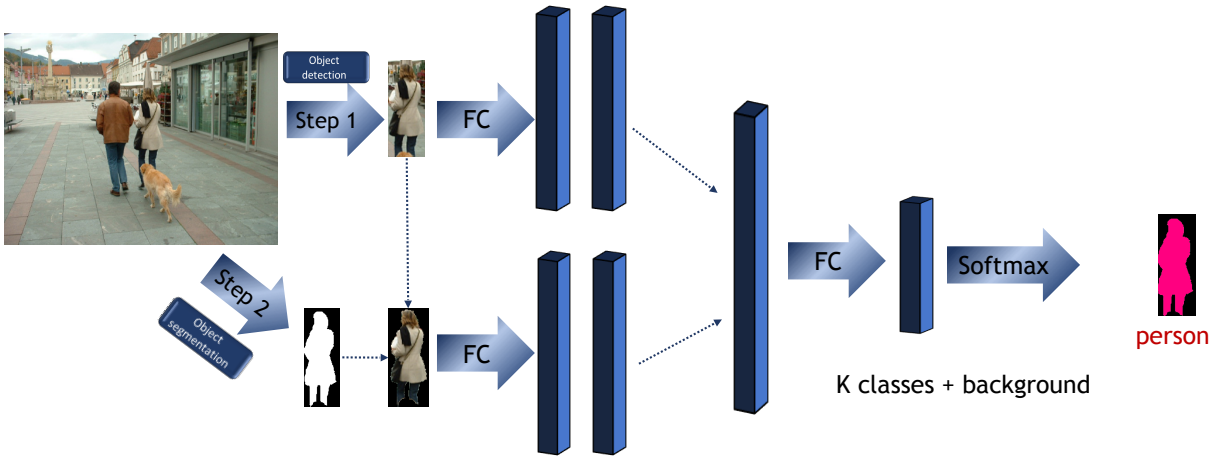


Figure 4.8: An illustration of Stage 3 for object classification using $f_{ROI}(\Theta) * f_{SEG}(\Theta)$ as inputs

The loss of entire proposed CRLS network is defined as in Eq. 4.17.

$$\begin{aligned} L(\Theta) &= L_{RPN} + L_{SEG} + L_{CLS} \\ &= L_{RPN}(B(\Theta)) + L_{SEG}(S(\Theta)|B(\Theta)) + L_{CLS}(C(\Theta)|B(\Theta), S(\Theta)) \end{aligned} \tag{4.16}$$

4.4 Inference

Given an image, the top-scored 300 ROIs are first chosen by RPNs proposed boxes. Non-maximum suppression (NMS) with an intersection-over-union (IoU) threshold 0.7 is used to filter out highly overlapping and redundant candidates. Then, for each ROIs, we apply our proposed RLS on top to get its foreground mask. From the ROIs and segmenting mask, the category score of each object instance is predicted via two fully-connected layers followed by a soft-max layer.

Besides softmax loss, which is to optimize when classifying a region is object or non-object in the first stage, there are three "dependent" losses in which the later loss depends on the predecessor loss. Three losses are asigned for each ROI, i.e. (1) smooth l_1 loss -

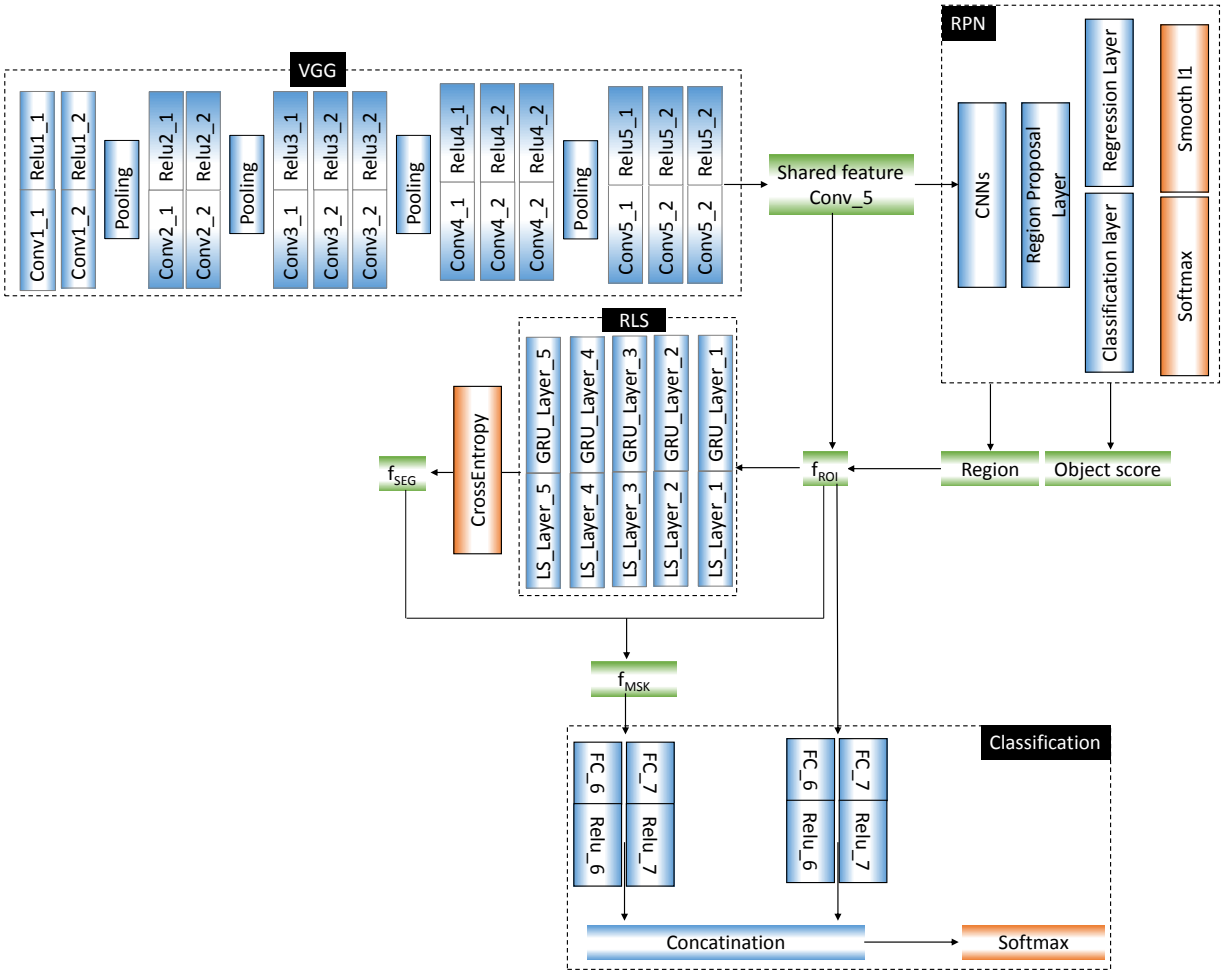


Figure 4.9: Architecture of the proposed CRLS for semantic instance segmentation with 3 stages corresponding to object detection (RPNs), object segmentation (RLS) and object classification

bounding box regression loss in the first state, (2) cross entropy loss - foreground segmentation loss in the second stage (3) softmax loss - instance classification loss in the third stage. The network architecture with the losses is given in Fig. 4.9. Among all the losses, computing the gradient w.r.t. predicted box positions and address the dependency on the bounding box $B(\Theta)$ is the most difficulty. Given a full size feature map $\mathcal{H}(\Theta)$, we crop a bounding box region (predicted box) $b_i(\Theta) = (b_x, b_y, b_w, b_h)$ and warp it to a fixed size by interpolation. The wrapped region ROI is presented as:

$$\mathcal{F}_{ROI}(\Theta) = \mathcal{I}(b_i(\Theta))\mathcal{F}(\Theta) \tag{4.17}$$

where $\mathcal{I}$ is cropping and warping operations. As for dimension, $\mathcal{F}(\Theta) \in \mathrm{R}^{\mathrm{N}}$ is a vector reshaped from the image, $N = W \times H$. The cropping and warping matrix $\mathcal{I} \in \mathrm{R}^{M} \times \mathrm{R}^{N}$. $\mathcal{F}_{ROI}(\Theta)$ is a target region sized $w \times h$ and $M = w \times h$. $\mathcal{I}(b_i(\Theta))$ presents transforming a that box $b_i(\Theta)$ from size of $b_w \times b_h$ into the size of $w \times h$. Let (x', y') and (x, y) are two points on target feature map $\mathcal{F}_{ROI}(\Theta)$ size of $w \times h$ and original map $\mathcal{F}(\Theta)$ size of $(W \times H)$, respectively. Using interpolation with (G) is the bilinear interpolation function, $\mathcal{I}(b_i(\Theta))$ is computed as follows:

$$\begin{aligned} \mathcal{I}(b_i(\Theta)) &= (G)(b_x + \frac{x'}{w} - x)(g)(b_y + \frac{y'}{h} - y) \\ g(z) &= max(0, 1 - |z|) \end{aligned} \tag{4.18}$$

4.5 Implementation Details

The proposed RLS approach is implemented using the Tensorflow system [240] whereas the extended CRLS semantic segmentation is implemented using Caffe environment [241]. Three functional sub-networks corresponding to three tasks, i.e. detection, segmentation and classification (shown in Fig. 4.5), are connected together to construct an end-to-end network. The first and the third sub-networks, i.e. object detection and object classification, are adopted from Faster R-CNN framework [100]. In the second sub-network, we re-implement the defined layers from RLS-Tensorflow to RLS-Caffe for both forward and backward operations[2] in Python. Since TensorFlow supports automatic differentiation capabilities, RLS-Tensorflow is therefore easier to implement than the new layers in RLS-Caffe.

During the training, the share features are obtained by 13 convolution layers initialized using the pre-trained VGG-16 model. Each convolution layer is always followed by a ReLU

[2]see the appendix for the derivatives in details

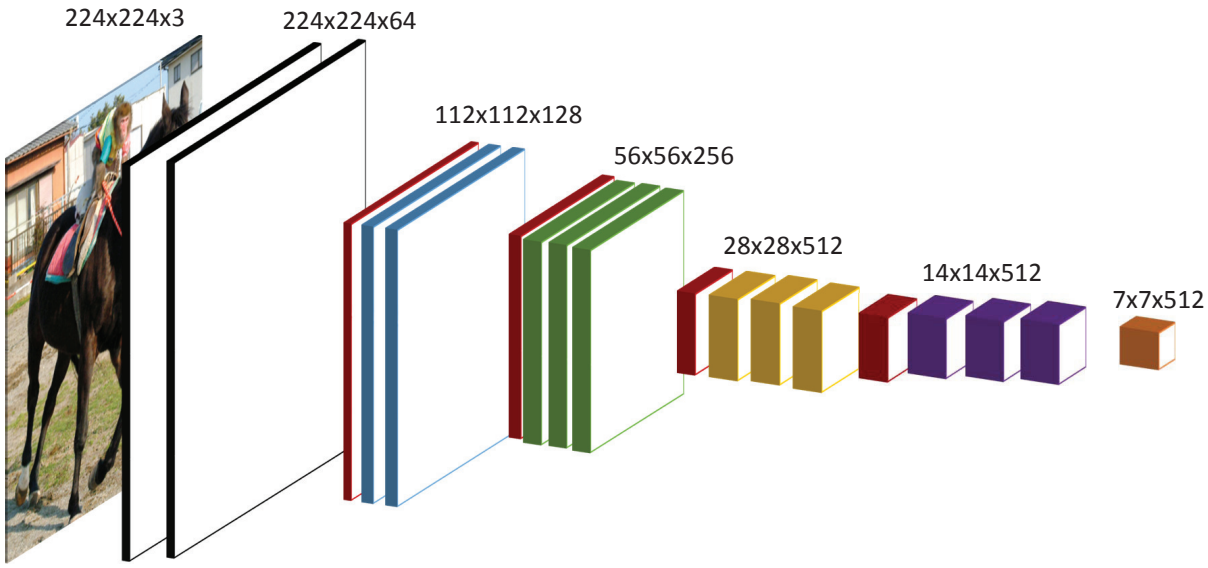


Figure 4.10: The architecture of the pre-trained model VGG-16 used in the proposed CRLS

layer but only 4 pooling layers are placed right after the convolution layer to reduce the spatial dimension. As shown in Fig. 4.10, the first section of CRLS is divided into 5 main components, i.e. conv1, conv2, conv3, conv4 and conv5, by those pooling layers. Each component consists of from 2 to 3 convolution layers.

In the first task, i.e. object detection, a 3×3 convolutional layer reduces feature dimensions and then two consecutive 1×1 convolutional layers predicts object's locations and object's presenting scores. The location regression is with reference to a series of pre-defined boxes, or anchors, at each location. Furthermore, we choose the two normalization terms (N_{cls}, N_{reg}) are chosen as $N_{cls} = 256$ and $N_{reg} = 2,400$. The balancing parameter λ is set as $\lambda = 10$. As for non-maximum suppression (NMS), which is used to reduce the number of boxes generated from the first stage ($\sim 10^4$ regressed boxes are produced from the first stage), the threshold of the Intersection-over-Union (IoU) ratio is chosen as 0.7. As a result, the top-ranked 300 boxes are kept for the second stage.

In the second task, i.e. object segmentation by the proposed RLS, we first extract a fixed-size (21×21) deep feature from an arbitrary box predicted using the object detection task. The proposed RLS takes the extracted feature as the input together with the

randomly initial ϕ_0 to generate a sequence input data $\mathbf{x}_t$ based on Eq. (4.1). The curve evolution procedure is performed via LS updating process given in Eq. (2.62). This task outputs a binary mask as given in Eq. (4.6) sized $m \times m$ and parameterized by an m^2 dimensional vector.

For each region candidate in the final task, i.e. object classification, a feature representation is firstly extracted by RoI-pooling from the shared convolutional features inside the bounding box region. Then it is masked by the segmenting mask prediction, which pays more attention on the foreground feature. Then the masked feature goes through two fully-connected layers, resulting a mask-based feature vector for classification.

The proposed CRLS semantic segmentation is implemented using Caffe environment [241]. Training images are resized to rescale their shorter side to 600 and use SGD optimization. On PASCAL VOC [24], we perform 32k and 8k iterations at learning rates of 0.001 and 0.0001, respectively. On the MSCOCO [25], we perform 180k and 20k iterations at learning rates of 0.001 and 0.0001, respectively.

Chapter 5

Contextual Recurrent Residual Networks (CRRN)

5.1 Introduction

Scene labeling is another case of pixel-level labeling and this problem has played an important role in many applications that are difficult, even for humans, given limited context. For example, whether a patch of dirt belongs to the side of a road or to a beach would be unclear until one looks at more of the image, thereby gaining context. Some studies aim to explicitly encode different kinds of context in order to facilitate labeling. However, most of the existing state-of-the-art methods mainly focus on either exploiting short-range contextual information [1, 2, 49, 199] or enriching visual representations [31, 53, 59, 60]. Thus, it is prone to misclassify visually similar pixels actually belonging to different classes. Indeed, the roles of contextual information and powerful descriptive visual representation are equally important in the scene labeling problem. To effectively address the scene labeling problem, the model has to be capable of learning the visual representation as well as modeling the contextual information.

Residual networks are able to learn effective visual representations but *unable to model*

context explicitly. Recently, deep residual networks (ResNets) [242] have emerged as a family of extremely deep architectures showing compelling accuracy and desirable convergence behaviors. They consist of blocks of convolutional and/or batch normalization layers equipped with an identity skip connection. The identity connection helps to address the vanishing gradient problem and allows the ResNets to robustly train using standard stochastic gradient descent despite very high model complexity. This enables ResNets to extract very rich representations of images that perform exceedingly well in image recognition and object detection challenges [215]. The extremely deep architectures in ResNets show compelling accuracy and robust convergence behaviors and achieve state-of-the-art performance on many challenging computer vision tasks on ImageNet [11], PASCAL VOC Challenge [243] and MS COCO [25] competitions. Nonetheless, they are feed forward models that do not explicitly encode contextual information and typically cannot be applied to sequence modeling problems.

On the other hand, *Recurrent neural networks (RNNs)* are effective in context modeling but *lack the ability to learning visual representation.* RNNs, and their variants, e.g. Long-Short Term Memories (LSTMs), Gated Recurrent Units (GRUs), etc., have become popular in the domains where sequence modeling is a natural problem, e.g. speech recognition, language processing, question answering, etc. Only recently, vision problems, e.g. scene labeling [1], object segmentation [52], have been reformulated as sequence learning, thereby allowing RNNs to be applied directly. Scene labeling, in particular, has seen the use of RNNs coupled with Directed Acyclic Graphs (DAGs) to model an image as a sequence [1, 2, 3]. There have also been a few studies that utilize RNNs to compute visual representations [223, 244]. However, these representations, e.g. vanilla RNNs and LSTMs, are clearly not as powerful and informative as the ones obtained through residual convolutional learning.

This paper presents a novel deep network named Contextual Recurrent Residual Networks (CRRN) to effectively model *contextual information* with rich *visual representation* within a *single deep network.* Our proposed CRRN deep networks consists of three parts

corresponding to sequential input data, sequential output data and hidden state which contains a set of units and each unit is designed as a combination of recurrent residual blocks along with vanilla RNNs components. Thus, each unit in the proposed network itself is able to simultaneously learn powerful visual representation while modeling long-range contextual dependencies. Furthermore, a powerful visual representation allows the model to have better descriptors of local image patches leading to richer contextual knowledge embeddings. For instance, a representation that discriminates well enough between a patch of sand on the side of a road versus on a beach will allow the RNNs component to model context better as compared to a representation that describes the two patches very similarly. Further, unlike previous approaches [1, 2, 3], our method is not only able to exploit the long-range context and visual representation but also formed under a fully-end-to-end trainable system that effectively leads to the optimal model.

The contributions of our work can be summarized as follows:

- We propose a fully end-to-end deep learning framework that is able to make use the advantages of both RNNs and ResNets in the recurrent form.
- In contrast to other existing deep learning network which are based on pre-trained models, our fully-end-to-end CRRN is completely trained from scratch.
- Our approach utilizes a more complex internal model involving residual blocks which allow for richer and more robust feature extraction during sequence learning.
- Each hidden unit in our proposed CRRN effectively learns contextual dependencies via recurrent component while simultaneously providing powerful encoded visual representation via residual component.
- Residual learning has so far been applied on raw visual data, our model instead applies residual learning technique to learn richer representations of the visual encoding provided by the recurrent component.

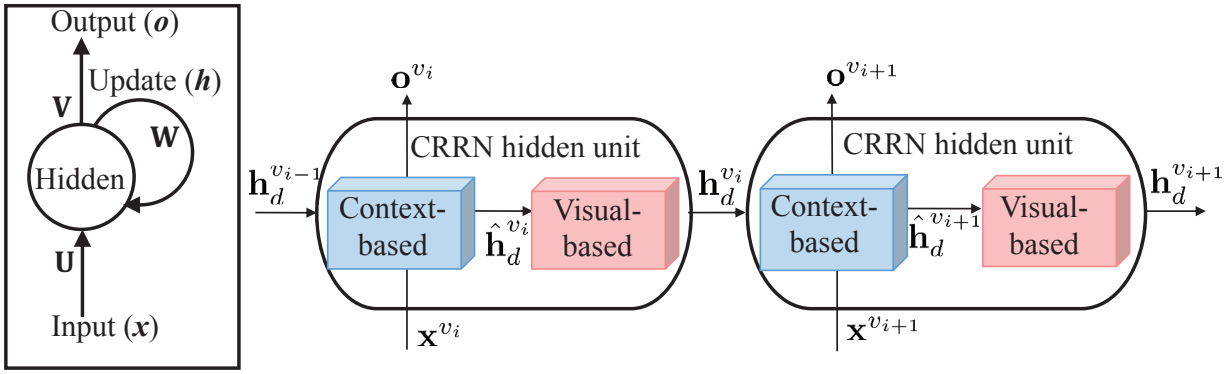


Figure 5.1: The proposed CRRN and the unfolding in time of the computation involved in its forward procedure.

5.2 The Proposed CRRN

Our proposed CRRN is designed as a composition of multiple CRRN units. Each of them consists of input, hidden, and output units. In the core of each CRRN hidden unit, two components, i.e. context-based and visual-based components, are employed to simultaneously handle two significant tasks. The former helps to handle the contextual knowledge embedding process while the latter tries to increase the robustness of visual representation extracted by the model. With this structure, one component can benefit from the other and provide more robust output as a result. On one hand, the powerful visual representation from the visual-based component allows the model to have better descriptors for each local patch and results in the better contextual knowledge embeddings. On the other hand, with better memory contextual dependencies from context-based component, highly discriminative descriptors can be extracted. Fig. 5.1 demonstrates the folding CRRN on the left and unfolding CRRN in time on the right.

Moreover, unlike previous approaches that are only able to capture the short-range context presented in small local input patches, our CRRN aims at modeling larger-range context by utilizing a graph structure over an image. As a result, the long-range contextual dependencies among different regions of an image can be efficiently modeled and, therefore, increasing the capability of the entire model. In particular, given an input image, it is first

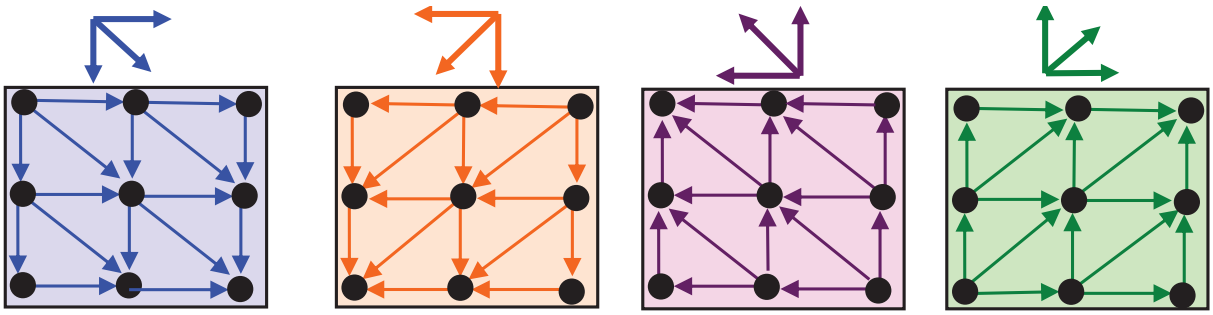


Figure 5.2: Decomposition of UCG into four DAGs. From left to right: southeast, southwest, northwest and northeast, where • denotes as a vertex

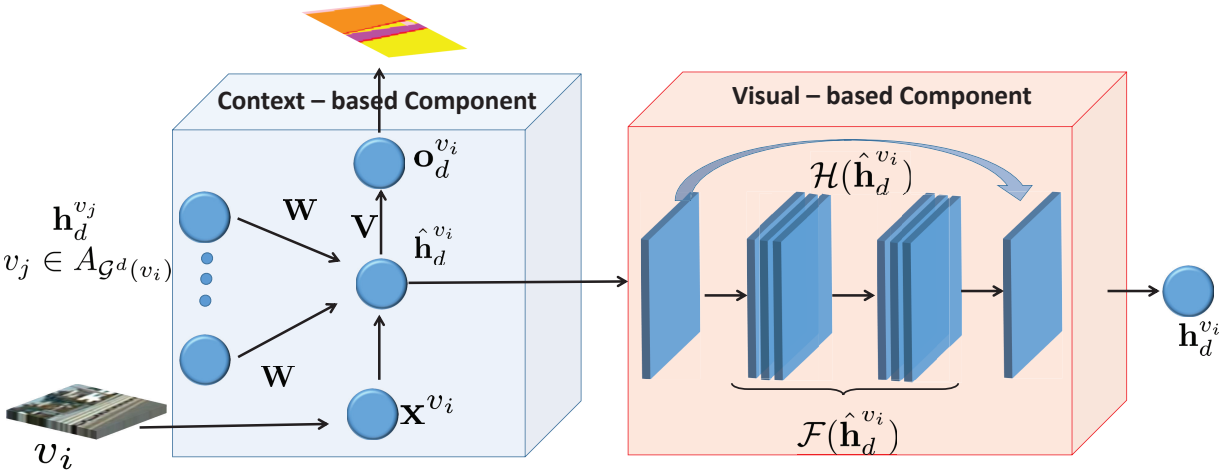


Figure 5.3: The forward procedure of CRRN at vertex $\mathbf{v}_i$ of one CRRN unit with two components corresponding to context information modeling and visual representation learning

divided into N non-overlapping blocks and their interactions are represented as an undirected cyclic graph (UCG). However, an UCG is unable to unroll as the forward-backward style deep model [2, 56] due to its loopy structures. Therefore, to address this issue, we first decompose the UCG into four directed acyclic graphs (DAGs) along southeast, southwest, northwest and northeast directions as given in Fig. 5.2. Then the contextual dependencies presented in each DAG are modeled by our CRRN. Finally, these information is combined to produce the final prediction.

Formally, an input image $\mathbf{I}$ is first divided into N non-overlapping blocks $\{v_i\}_{i=1,2,..,N}$. Each block is then considered as a vertex in four DAGs $\mathcal{G}^1$, $\mathcal{G}^2$, $\mathcal{G}^3$, $\mathcal{G}^4$ corresponding to the four directions. Each DAG is formed as $\mathcal{G}^d = \{\mathcal{V}, \mathcal{E}\}|_{d=1}^4$, where $\mathcal{V} = \{v_i\}_{i=1,2,..,N}$, and

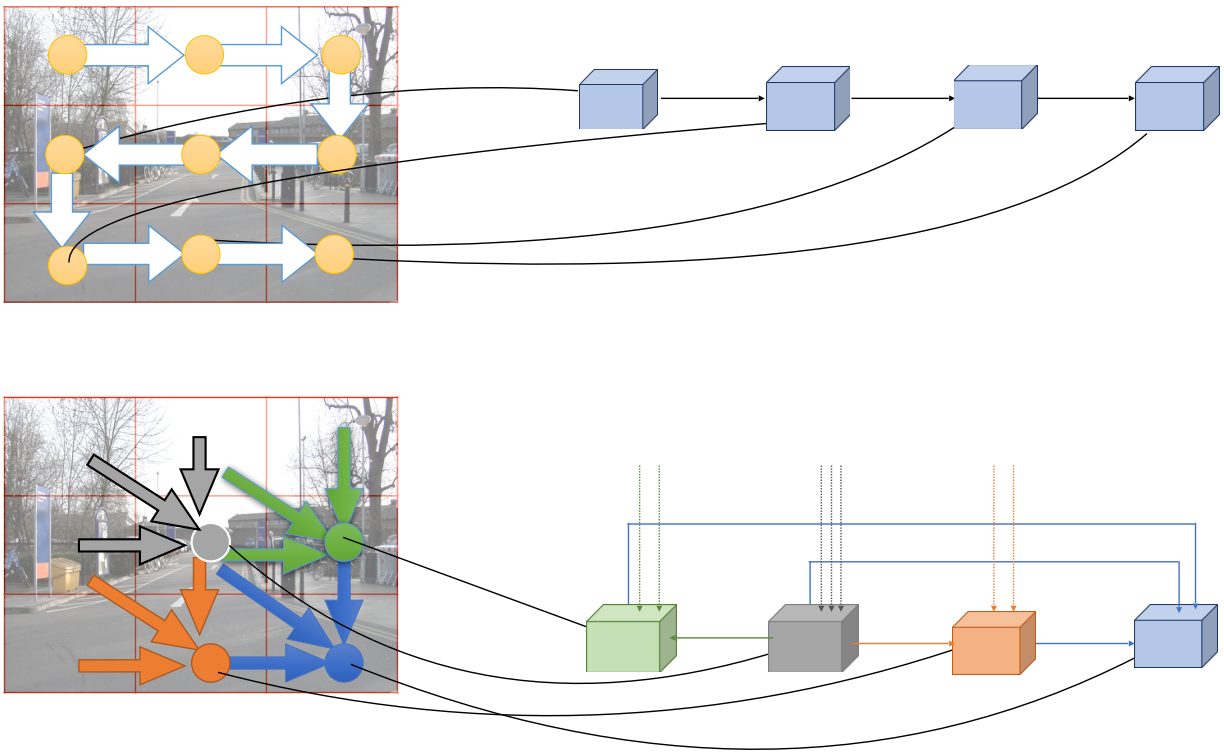


Figure 5.4: The relationship between a vertex and its predecessors in two scenarios: upper is snake scan and lower is northeast DAG scan

$\mathcal{E} = \{e_{ij}\}$ is the edge set where each edge e_{ij} represents the relationship between vertices v_i and v_j. For each direction $\mathcal{G}^d$, let $\mathcal{A}_{\mathcal{G}^d(v_i)}$ and $\mathcal{S}_{\mathcal{G}^d(v_i)}$ be the predecessor and successor sets of v_i. Depending on the relationships between vertices in $\mathcal{G}^d$, all vertices are organized into a sequence and fed into the CRRN structure for modeling. Fig.5.4 shows the relationship between a vertex and its predecessors in two scenarios of snake scan (upper figure) and northeast DAG scan (lower figure). It is easily to see that the snake scan strategy produces a network which is similar to the vanilla RNNs whereas the DAG strategy gives a more complicated network where each node in the network receives the input from three sources as in Fig.5.4.

As illustrated in Fig. 5.3, the i-th CRRN unit takes an input of $\mathbf{x}^{v_i}$ and the hidden state of all vertices in the predecessor set $\mathcal{A}_{G^d(v_i)}$. The input $\mathbf{x}^{v_i}$ can be computed as $\mathbf{x}^{v_i} = \text{vec}(v_i)$, where vec $(\cdot)$ is the vectorization operator. The following will detail the proposed CRRN

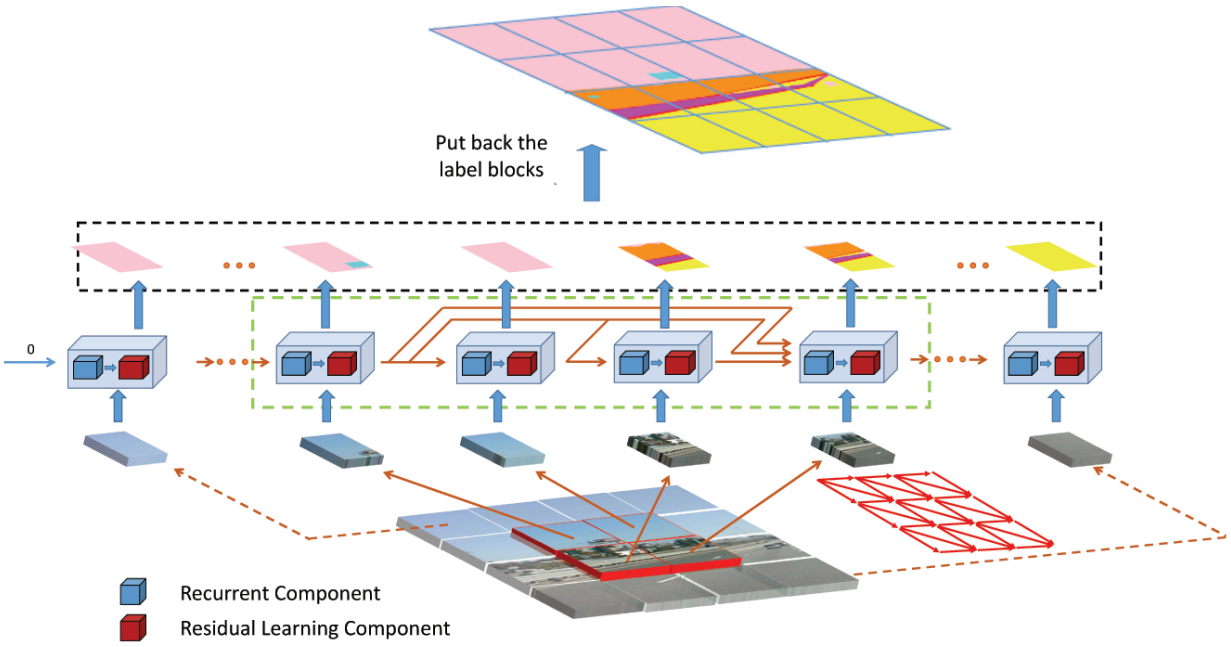


Figure 5.5: An illustration of our proposed CRRN architecture at one direction (southeast)

architecture with two main tasks: (1) the contextual information embedding via sequence learning, and (2) the visual representation learning.

5.2.1 Contextual information embedding

Starting with one DAG $\mathcal{G}^d$ at vertex v_i, the contextual relationship between the vertex v_i and its predecessor set $\mathcal{A}_{G^d(v_i)}$ is expressed as a non-linear function over the current input x^{v_i} and the summation of hidden layers of all its predecessors in the context-based component as follows:

$$\begin{aligned} \hat{\mathbf{h}}_d^{v_i} &= f_{\text{CONTEXT}}(\mathbf{x}^{v_i}, \mathcal{A}_{\mathcal{G}^d(v_i)}; \theta_1) \\ &= \phi(\mathbf{U}\mathbf{x}^{v_i} + \sum_{v_j \in \mathcal{A}_{\mathcal{G}^d(v_i)}} \mathbf{W}\mathbf{h}_d^{v_j} + \mathbf{b}) \end{aligned} \tag{5.1}$$

where $\mathbf{h}_d^{v_j}$ is the hidden state of vertex v_j belonging to predecessor set $\mathcal{A}_{\mathcal{G}^d(v_i)}$ of vertex v_i in the DAG $\mathcal{G}^d$, $\hat{\mathbf{h}}_d^{v_i}$ is the intermediate hidden state of vertex v_i and $\theta_1 = \{\mathbf{U}, \mathbf{W}, \mathbf{V}, \mathbf{b}\}$ is

the parameters of context-based component representing the connection weights of input-to-hidden, predecessor-to-hidden and hidden-to-output; and the hidden bias, respectively. $\phi(\cdot)$ is the activation function, i.e. ReLU function.

5.2.2 Visual representation learning

To further extract powerful visual representation as well as address the vanishing problem during modeling, we employ the visual-based component with residual learning technique. Given the intermediate hidden state $\hat{\mathbf{h}}_d^{v_i}$ of the vertex v_i provided by the context-based component, the representation $\mathbf{h}_d^{v_i}$ of each vertex v_i is computed as in Eqn. (5.2).

$$\begin{aligned} \mathbf{h}_d^{v_i} &= f_{\text{VISUAL}}(\hat{\mathbf{h}}_d^{v_i}; \theta_2) \\ &= \phi(\mathcal{F}(\hat{\mathbf{h}}_d^{v_i}) + \mathcal{H}(\hat{\mathbf{h}}_d^{v_i})) \end{aligned} \tag{5.2}$$

where $\mathcal{F}$ denotes a residual function consisting of a two-layer convolutional network in residual-network style, i.e equipped with an residual learning connection as shown in Fig. 5.3. This network is a stack of two convolutional layers, i.e. alternating convolution, batch normalization and ReLU operations. Meanwhile, $\mathcal{H}$ is defined as an identity mapping, where $\mathcal{H}(\hat{\mathbf{h}}_d^{v_i}) = \hat{\mathbf{h}}_d^{v_i}$. θ_2 represents the parameters of the two-layer convolutional network.

The final output from the four DAGs are then combined using the following Eqn.5.3.

$$\begin{aligned} \mathbf{o}^{v_i} &= f(\mathbf{x}^{v_i}, \mathcal{A}_{\mathcal{G}(v_i)}; \theta_1, \theta_2) \\ &= \sum_{d=1}^{4} \mathbf{V}\hat{\mathbf{h}}_d^{v_i} + \mathbf{b}_o \end{aligned} \tag{5.3}$$

where $\mathbf{V}$ is the hidden-to-output weight matrix, $\mathbf{b}_o$ is output bias. The CRRN is then optimized to minimize the negative log-likelihood over the training data as follows:

$$\theta_1^*, \theta_2^* = \arg\min_{\theta_1, \theta_2} L(\theta_1, \theta_1) \tag{5.4}$$

$$
\begin{aligned}
L(\theta_1, \theta_1) &= -\frac{1}{n} \sum_{v_i \in \mathcal{G}} \sum_j \log p(l_j^{v_i} | \mathbf{x}^{v_i}; \theta_1, \theta_2) \\
p(l_j^{v_i} | \mathbf{x}^{v_i}; \theta_1, \theta_2) &= \frac{e^{f_{l_j^{v_i}}(\mathbf{x}^{v_i}, \mathcal{A}_{\mathcal{G}(v_i)}; \theta_1, \theta_2)}}{\sum_{c=1}^{C} e^{f_c(\mathbf{x}^{v_i}, \mathcal{A}_{\mathcal{G}(v_i)}; \theta_1, \theta_2)}}
\end{aligned}
\tag{5.5}
$$

where C is the number of classes; n is the number of images; $l_j^{v_i}$ is the correct label of j-th pixel in block v_i; and $f_{l_j^{v_i}}(\mathbf{x}^{v_i}, \mathcal{A}_{\mathcal{G}(v_i)}; \theta_1, \theta_2) = o_j^{v_i}$.

5.2.3 Model learning

The optimal parameters can be obtained with the Stochastic Gradient Descent (SGD) algorithm given by:

$$
\theta_1 \leftarrow \theta_1 - \lambda \frac{\partial L}{\partial \theta_1}; \theta_2 \leftarrow \theta_2 - \lambda \frac{\partial L}{\partial \theta_2} \tag{5.6}
$$

where λ denotes the learning rate.

The derivatives are computed in the backward pass procedure is processed in the reverse order of forward propagation sequence as illustrated in Fig.5.6. Instead of looking at predecessor $\mathcal{A}_{\mathcal{G}^d(v_i)}$ in the forward pass, we are now taking a look at successor $\mathcal{S}_{\mathcal{G}^d(v_i)}$.

It is clear that the backpropagation error at the intermediate hidden layer $d\hat{\mathbf{h}}_d^{v_i}$ comes from two sources: one from output $\frac{\partial \mathbf{o}_d^{v_i}}{\partial \hat{\mathbf{h}}_d^{v_i}}$ and the other one from hidden layer $\frac{\partial \mathbf{h}_d^{v_i}}{\partial \hat{\mathbf{h}}_d^{v_i}}$ whereas the backpropagation error at hidden layer $d\mathbf{h}_d^{v_i}$ comes from its successor set $\mathcal{S}_{\mathcal{G}^d(v_i)}$. i.e, $\sum_{v_k \in \mathcal{S}_{\mathcal{G}^d(v_i)}} \frac{\partial \mathbf{o}_d^{v_k}}{\partial \hat{\mathbf{h}}_d^{v_k}} \frac{\partial \hat{\mathbf{h}}_d^{v_k}}{\partial \mathbf{h}_d^{v_i}}$.

To make it simple and general, we would like to express the parameters in a form without subscript $_d$. For example we will use $\mathbf{h}^{v_i}$ instead of $\mathbf{h}_d^{v_i}$. The derivative with

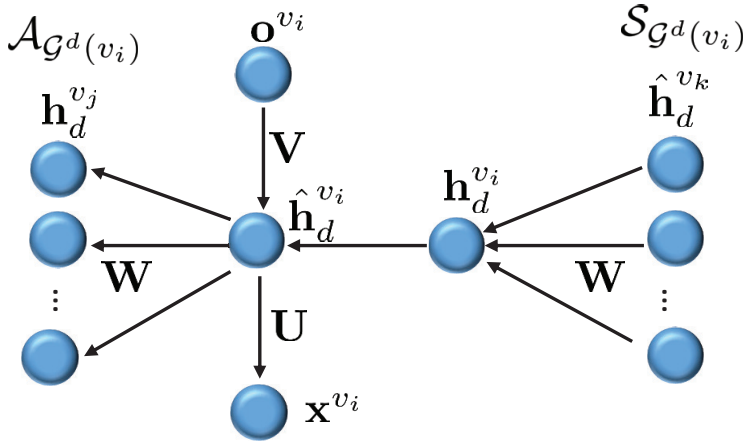


Figure 5.6: The backward procedure of CRRN at vertex $\mathbf{v}_i$

respect to the model parameters $\{\mathbf{o}^{v_i}, \hat{\mathbf{h}}^{v_i}, \mathbf{h}^{v_i}, \mathbf{V}, \mathbf{U}, \mathbf{W}\}$ can be computed as follows:

$$
\begin{aligned}
d\mathbf{o}^{v_i} &= \frac{\partial L}{\partial \mathbf{o}^{v_i}} \\
d\hat{\mathbf{h}}^{v_i} &= \frac{\partial L}{\partial \mathbf{o}^{v_i}} \frac{\partial \mathbf{o}^{v_i}}{\partial \hat{\mathbf{h}}^{v_i}} + \left(\frac{\partial \mathbf{h}^{v_i}}{\partial \hat{\mathbf{h}}^{v_i}}\right)^T d\mathbf{h}^{v_i} = \mathbf{V}^T d\mathbf{o}^{v_i} + \left(\frac{\partial \mathbf{h}^{v_i}}{\partial \hat{\mathbf{h}}^{v_i}}\right)^T d\mathbf{h}^{v_i} \\
d\mathbf{h}^{v_i} &= \sum_{v_k \in \mathcal{S}_{\mathcal{G}^d(v_i)}} \mathbf{W}^T \frac{\partial L}{\partial \hat{\mathbf{h}}^{v_k}} \frac{\partial \hat{\mathbf{h}}^{v_k}}{\partial \mathbf{h}^{v_i}} \circ \phi'(\hat{\mathbf{h}}^{v_i}) = \sum_{v_k \in \mathcal{S}_{\mathcal{G}^d(v_i)}} \mathbf{W}^T d\hat{\mathbf{h}}^{v_k} \circ \phi'(\hat{\mathbf{h}}^{v_i}) \\
\nabla \mathbf{V} &= \frac{\partial L}{\partial \mathbf{o}^{v_i}} \frac{\partial \mathbf{o}^{v_i}}{\partial \mathbf{V}} = d\mathbf{o}^{v_i} \left(\hat{\mathbf{h}}^{v_i}\right)^T \\
\nabla \mathbf{U} &= \frac{\partial L}{\partial \mathbf{o}^{v_i}} \frac{\partial \mathbf{o}^{v_i}}{\partial \hat{\mathbf{h}}^{v_i}} \frac{\partial \hat{\mathbf{h}}^{v_i}}{\partial \mathbf{U}} = d\hat{\mathbf{h}}^{v_i} \circ \phi'(\hat{\mathbf{h}}^{v_i}) \left(\mathbf{x}^{v_i}\right)^T \\
\nabla \mathbf{W} &= \frac{\partial L}{\partial \mathbf{o}^{v_i}} \frac{\partial \mathbf{o}^{v_i}}{\partial \hat{\mathbf{h}}^{v_i}} \frac{\partial \hat{\mathbf{h}}^{v_i}}{\partial \mathbf{W}} = \sum_{v_j \in \mathcal{S}_{\mathcal{G}}^d(v_i)} d\hat{\mathbf{h}}^{v_j} \circ \phi'(\hat{\mathbf{h}}^{v_j}) \left(\hat{\mathbf{h}}^{v_i}\right)^T
\end{aligned}
\tag{5.7}
$$

where $\circ$ represents the Hadamard product.

5.3 Inference

Given a testing image $\mathbf{I}$, we first divide $\mathbf{I}$ into N blocks v_i, $i = 1...N$. These blocks are then fed into four DAGs. The inference process of each pixel j in block v_i can be performed by

finding the class label that maximizes the conditional probability given by:

$$l_j^{v_i*} = \arg\max_c p(c|\mathbf{x}^{v_i}; \theta_1, \theta_2) \tag{5.8}$$

Finally, the prediction maps of all v_i are concatenated based on the location of block v_i in $\mathbf{I}$ for the final prediction map. Fig. 5.5 illustrates the inference process of CRRN at one direction (southeast) as an example.

5.4 Implementation Details

In this section, the implementation of our proposed model is discussed in details. The model is implemented in TensorFlow environment and runs in a machine of Core i7-6700 @3.4GHz CPU, 64.00 GB RAM and a single NVIDIA GTX Titan X GPU. In order to train the CRRN, each image in the training data is first divided into 64 ($= 8 \times 8$) blocks. Next, these blocks are reorganized into four sequences based on their relationships in the four DAGs. i.e. $\mathcal{G}_1$, $\mathcal{G}_2$, $\mathcal{G}_3$, $\mathcal{G}_4$. Each block is vectorized and used as the input for the CRRN unit. Our CRRN architecture is then employed to simultaneously model the contextual dependencies between blocks in different directions as well as extract their compact and rich representation. which simultaneously performs two tasks: model the contextual dependencies between this block and its neighbor in different directions and extract their compact and rich visual representation at this block. The dimensionality of the hidden layer is set to 256 in the recurrent component and reshaped to 16×16 before going through the residual learning component. From our preliminary results, hidden layers with smaller number of hidden units are not powerful enough to capture both contextual information and visual representation in four DAGs while larger number may increase the computational cost without too much performance improvement. The optimal parameter values of the whole network are obtained via a stochastic gradient descent procedure with

forward and backward passes. The learning rate is initialized to be 10^{-3}, and decays with the rate of 0.95 after 30 epochs.

Chapter 6

Experimental Results

Corresponding two proposed algorithms, i.e. CRLS Networks for semantic instance segmentation and CRRN for scene labeling, two experiments have been conducted and evaluated in this chapter. All the experiments are performed on a system of Core i7 @3.4GHz CPU, 64.00 GB RAM with a single NVIDIA GTX Titan X GPU.

6.1 Experimental Results of CRLS

In this section, we conduct two experiments corresponding to the object segmentation by the proposed RLS method and the semantic instance segmentation by the proposed CRLS system on PASCAL VOC [24] and COCO [25] datasets.

6.1.1 Datasets

Synthetic dataset The synthetic and medical dataset containing 720 images are artificially created from images reported in [62, 188]. There are actually about 20 images reported in [62, 188]. To avoid over-fitting problem, We have applied with different kinds of degradation (i.e., add various noise, blurring) and various affine transformations(i.e., rotation, translation, scale, and flip) into the images to generate the dataset of 720 images.

The degradation is added to the image to make sure the proposed RLS is able to run on both intensity homogeneity and inhomogeneity.

In this dataset, we use 360 images which were artificially generated from first 10 images for training and the rest (360 images) for testing. Meanwhile, the real images are collected from Weizmann [245] database which contains 100 natural images with 1 object in the background. From this dataset, we generate 4,700 images augmented using different kinds of noise and various affine operations. We used 2,350 images for training and 2,350 images for testing.

PASCAL VOC The PASCAL VOC 2012 [24] are a commonly used database for evaluating semantic segmentation. The PASCAL VOC 2012 training and validation set has 11,540 images containing 27,450 bounding box annotated objects and 6,929 segmentations in 20 categories and one background category. The data is divided into 5,717 images for training and 5,823 images for testing(val). The twenty object classes that have been selected are: person, bird, cat, cow, dog, horse, sheep, aeroplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa, tv/monitor.

MSCOCO [25] The MS COCO training set contains about 80,000 images for training and 40,000 images for validation. This dataset consists of $\sim 500,000$ annotated objects of 80 classes and one background class. COCO is a more challenging dataset as it contains objects in a wide range of scales from small ($< 32^2$) to large ($> 96^2$) objects.

6.1.2 Metrics

For the semantic segmentation task, we use a standard metric, i.e. Mean Average Precision (mAP), which was reported in PASCAL VOC 2012 to evaluate the performance. For a given task and class, the precision/recall curve is computed from a methods ranked output. Recall is defined as the proportion of all positive examples ranked above a given rank. Precision is the proportion of all examples above that rank which are from the positive

class. We use the Average Precision (AP) to summarize the shape of the precision/recall curve, and is defined as the mean precision at a set of eleven equally spaced recall levels [0, 0.1, ..., 1]:

$$AP = \frac{1}{11} \sum_{r \in 0,0.1,...,1.0} p_{interp}(r) \tag{6.1}$$

The precision at each recall level r is interpolated by taking the maximum precision measured for a method for which the corresponding recall exceeds r:

$$p_{interp}(r) = max_{\tilde{r}:\tilde{r}>r} p(\tilde{r}) \tag{6.2}$$

where $p(\tilde{r})$ is the measured precision at recall r. The intention in interpolating the precision/recall curve in this way is to reduce the impact of the wiggles in the precision/recall curve, caused by small variations in the ranking of examples. It should be noted that to obtain a high score, a method must have precision at all levels of recall this penalises methods which retrieve only a subset of examples with high precision (e.g. side views of cars).

6.1.3 Experiment 1 - Object Segmentation by RLS

This section compares our object segmentation RLS against Chan-Vese's LS [62], DRLSE [15], Li's method. [16], L2S [21] and a simple Neural Network with two fully-connected layers followed by a ReLU layer in between and we name it F-ConNet. The experiments are validated on both synthetic images, medical images and natural images collected in the wild. The input images are resized to 64×64, thus, the number of units in fully-connected layers and hidden cell of GRU is 4,096.

Fig. 6.1 shows some segmentation results using the baseline CLS model [62], DRLSE [15] as for global approach, Li et al [16, 21], L2S [21] as for the inhomogeneity approach,

F-ConNet as a baseline deep learning approach and our proposed RLS on the synthetic, medical and natural databases. The best results from CLS's method, DRLSE, Li's method are L2S are given in the second, third, fourth, fifth rows respectively. The sixth row shows our RLS segmentation results. The last row shows the ground truth. In each instance, the segmentation result is given as a black/white image. The average F-measure on the real image test set obtained by CLS, DRLSE, Li's, L2S, F-ConNet and our proposed RLS are reported as in Tables 6.1 with two separated groundtruth versions (GT1 and GT2) provided by two different people on Weizmann database [245].

In terms of speed, CLS method consumes 13.5 seconds while Li et al's and DRLSE approaches have similar time consuming of 20.4 seconds and 23.5 seconds on average to process one image with original size. L2S consumes less time (10.2 seconds) than the others CLS, Li's and DRLSE whereas the proposed RLS takes 0.008 seconds and F-ConNet takes 0.001 seconds on average testing time to segment an object. RLS achieves the best segmentation performance in this experiment on both groundtruth annotated by two different people.

Table 6.1: Average F-measure (FM) and testing time obtained by CV's model, **DRLSE**[15], **Li et al.**[16],**L2S** [21], **F-ConNet** and our proposed **RLS** with two different ground truth (GT1 and GT2) across Weizmann database

Methods	**FM (GT1)**	**FM (GT2)**	**Testing Time**
CV [62]	88.51	87.51	13.5(s)
DRLSE[15]	80.93	71.76	23.5 (s)
Li et al.[16]	79.65	63.87	20.4 (s)
L2S [21]	81.36	71.03	10.2 (s)
F-ConNet	93.30	93.26	**0.001** (s)
RLS	**99.16**	**99.17**	0.008 (s)

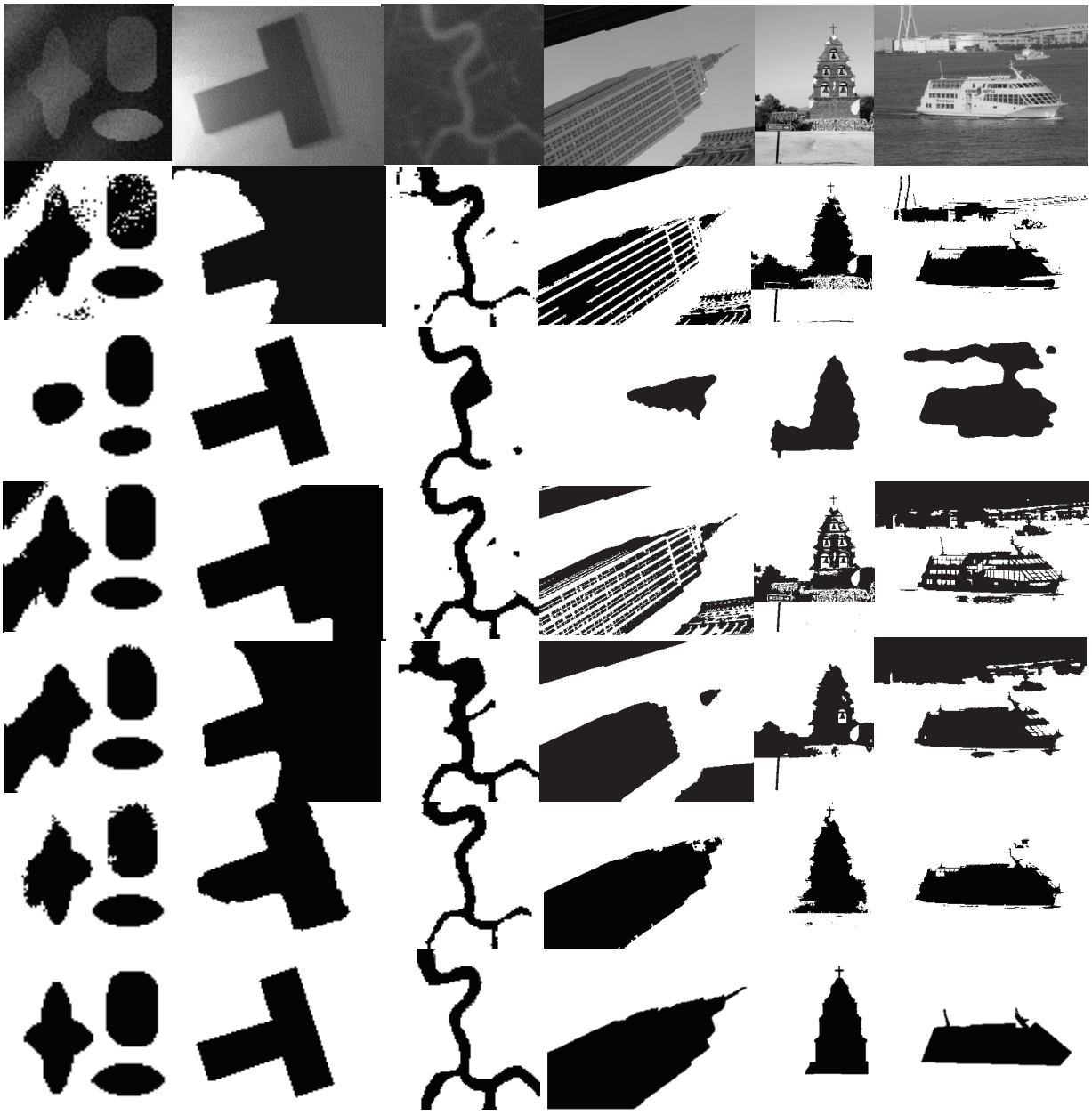


Figure 6.1: Object segmentation on medical, synthetic images and from Weizmann database. $1st$ row: input images, 2^{nd} row: segmentation by CLS [62], 3^{rd} row: segmentation by DRLSE [15], 4^{th} row: segmentation by Li et al.[16], 5^{th} row: segmentation by L2S [21], 6^{th} row: segmentation our RLS, 7^{th} row: groundtruth. The best results for [62], [15], [16], [21]

6.1.4 Experiment 2 - Semantic Instance Segmentation by CRLS

We demonstrate our proposed CRLS approach on PASCAL VOC 2012. We follow the same protocols used in recent papers [26, 27, 28, 29] for evaluating semantic segmentation.

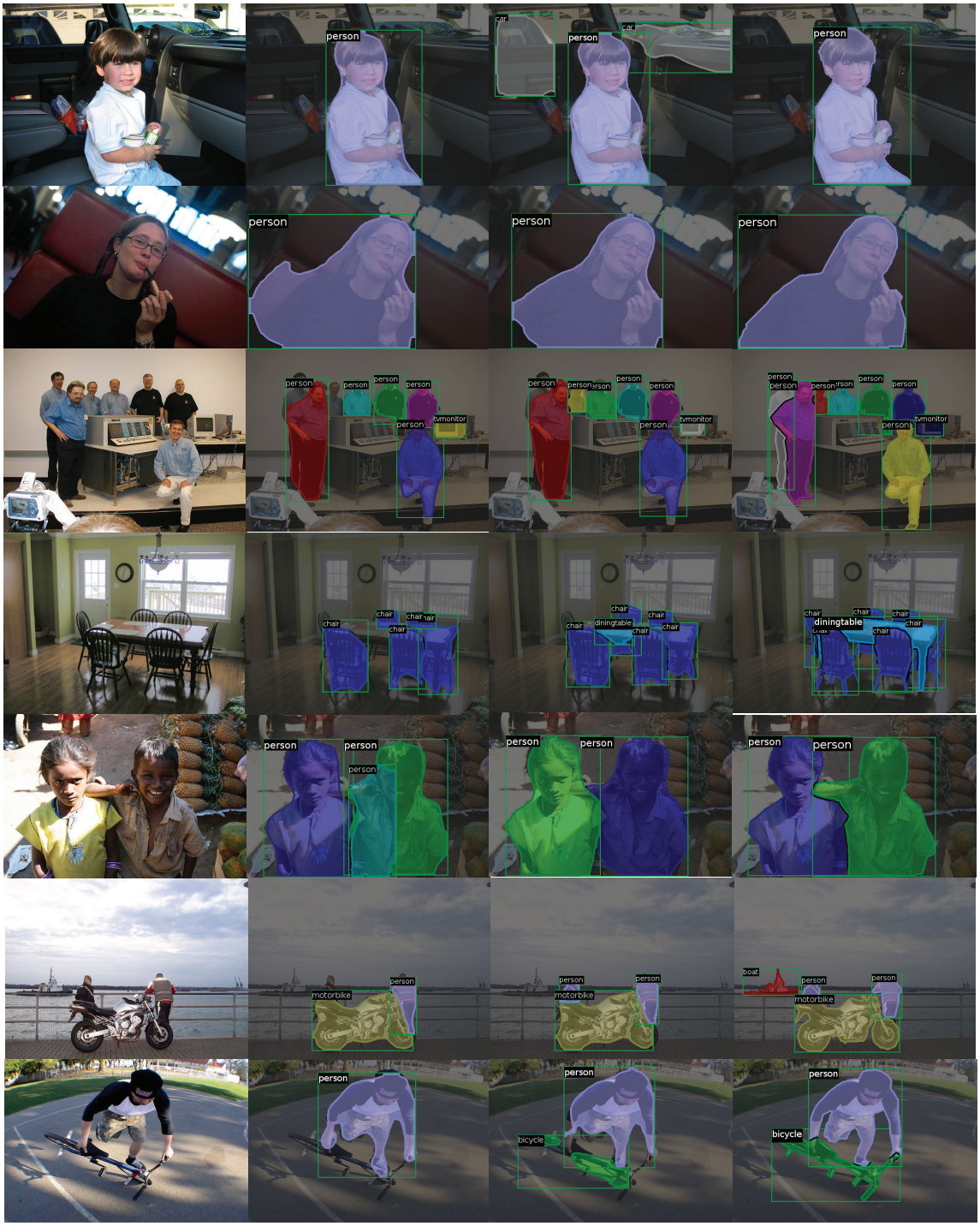


Figure 6.2: Some examples of semantic segmentation on PASCAL VOC 2012 database. The input image (1^{st} column), MNC [29] (2^{nd} column), our semantic segmentation CRLS (3^{rd} column) and the ground truth (4^{th} column). (Best viewed in color)

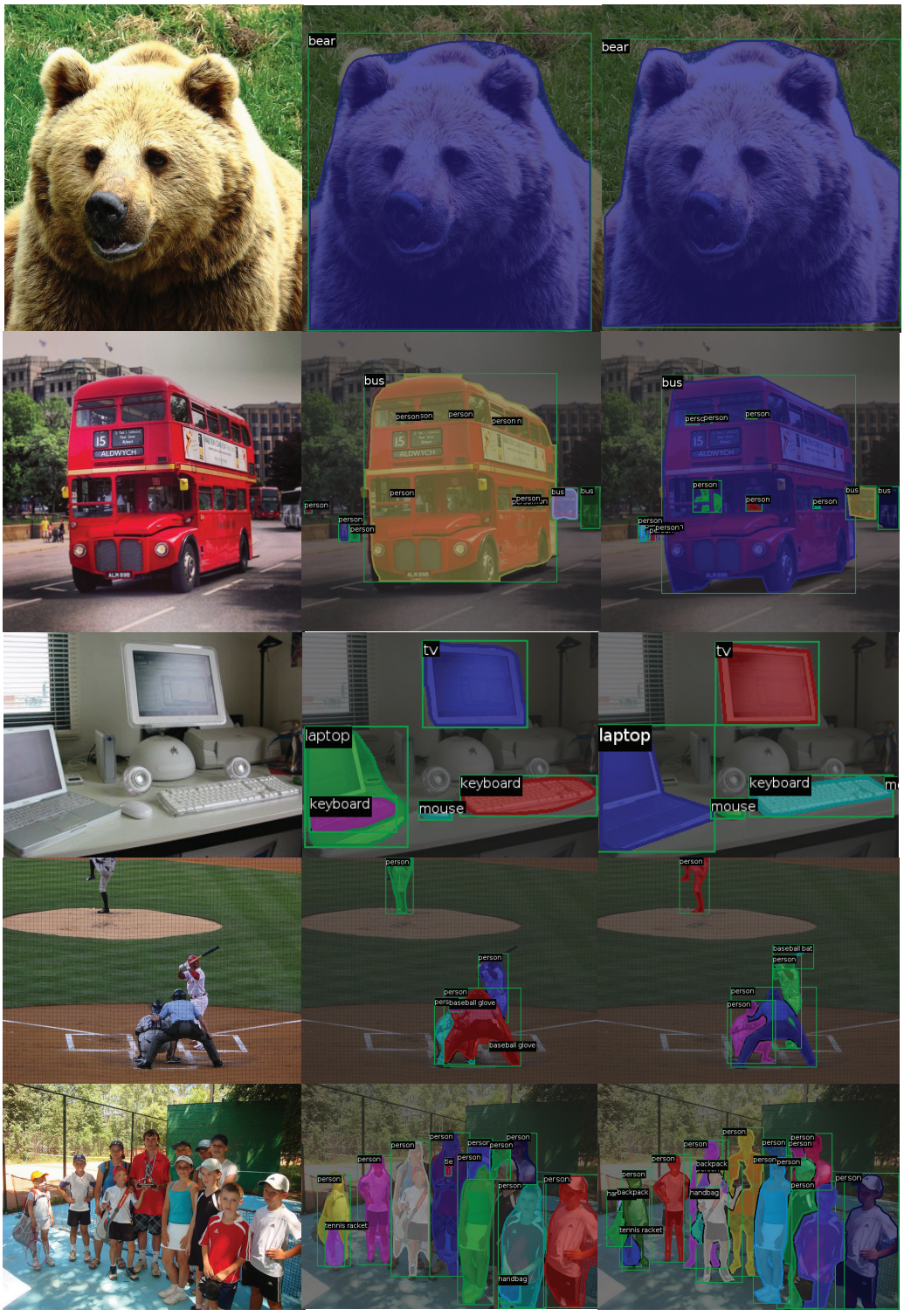


Figure 6.3: Some examples of semantic segmentation on MS COCO database on validation set. The input image (1^{st} column), our semantic segmentation CRLS (2^{nd} column) and the ground truth (3^{rd} column). (Best viewed in color)

Table 6.2: Quantitative results and comparisons against existing CNN-based semantic segmentation methods (**SDS** (AlexNet)[26], **Hypercolumn** [27], **CFM** [28], **MNC** [29]) on the PASCAL VOC 2012 validation set.

Methods	**mAP^r@.5**	**mAP^r@.7**	**Time (s)**
SDS (AlexNet)	49.7%	25.3%	48
Hypercolumn	60.0%	40.4%	>80
CFM	60.7%	39.6%	32
MNC	63.5%	41.5%	**0.36**
CRLS (Ours)	**66.7%**	**44.6%**	0.54

Table 6.3: Quantitative results and comparisons against existing CNN-based semantic segmentation method **MNC**[29] on the MS COCO 2014 database

Methods	**mAP^r@[.5:.95]**	**mAP^r@.5**
MNC	19.5%	39.7%
CRLS (Ours)	**20.5%**	**40.1%**

The models are trained on the PASCAL VOC 2012 training set, and evaluated on the validation set. The end-to-end CRLS network is trained using the ImageNet pre-trained VGG-16 model. Results are reported on the metrics commonly used in recent semantic object segmentation papers [26, 27, 28, 29]. We compute the mean average precision (mAP^r) [29] to show the segmentation accuracy. It is measured by Intersection over Union (IoU) which indicates the intersection-over-union between the predicted and ground-truth pixels, averaged over all the classes. In PASCAL VOC, we evaluate mAP^r with IoU at 0.5 and 0.7. In table 6.2, we compare our proposed CRLS with existing CNN-based semantic segmentation methods including SDS [26], Hypercolumn [27], CFM [28] and MNC [29]. All the results of those methods are quoted from paper [29]. Using the same testing protocol, our CRLS achieves higher mAP^r at both 0.5 and 0.7 than previous methods (about 3%). In addition to high segmentation accuracy, the experimental results also show that our proposed CRLS gives very efficient testing time (0.54 second per image). Some examples of multi-instance object segmentation by our proposed CRLS on PASCAL VOC

2012 database are shown in Fig. 6.2.

Our proposed approach is trained on the MS COCO 2014 80k training images and evaluated on 20k images in the test set (*test-dev*). We measure the performance of our method on two standard metrics which are the mean average precision (mAP^r) using IoU between 0.5 & 0.95 and mAP^r using IoU at 0.5 (as PASCAL VOC metrics) as shown in Table 6.3. Our CRLS achieves better results than the previous method (MNC) on the COCO dataset (noted that we only compare with their VGG-16 network results).

6.1.5 Experiment 3 - Multi-instace object segmentation by proposed CRLS

We have added some more semantic segmentating results on various cases i.e. overlapping objects, single-instance object, multi-instance object from PASCAL VOC and MS COCO databases. More illustrations for the comparison between MNC performance [29] and ours is also showed here. In this section, we also provide in details the segmentation performance (average precision %) on each category at IoU = 0.5 on PASCAL VOC 2012. The performance of CRLS on MS COCO database is also showed in details with different measures, i.e. average precision, average precision across scales, average recall, average recall across scales.

Overlapping objects:

When dealing with overlapping object, mislabeling is a common problem in MNC [29] as shown in the second row of Fig.6.4 while our CRLS can per2.0form very well even when only a small portion of object is given as illustrated in the third row of Fig.6.4. As shown in the first three images, we can see that our CRLS is able to segment and recognize a object even there is only small portion (chair, dinning table, bottle) existed. Notably, the partial objects are not well labeled in the groundtruth.

Single-instance object:

Figure 6.4: Some examples of overlapping objects segmentation by MNC [29] (2^{nd} column) and our CRLS (3^{rd} column) on PASCAL VOC 2012 with the ground truth (4^{th} column) database. (Best viewed in color)

Figure 6.5: Some examples of single-instance objects segmentation by MNC [29] (2^{nd} column) and our CRLS (3^{rd} column) on PASCAL VOC 2012 database with the ground truth (4^{th} column) database. (Best viewed in color)

Figure 6.6: Some examples of multiple-instance objects segmentation by MNC [29] (2^{nd} column) and our CRLS (3^{rd} column) on PASCAL VOC 2012 database with the ground truth (4^{th} column) database. (Best viewed in color)

Figure 6.7: Some examples of multiple-instance objects segmentation by MNC [29] (2^{nd} column) and our CRLS (3^{rd} column) on MS COCO database validation set. Each row has four images: the input image (first column), MNC segmentation results (second column), our semantic segmentation results (third column) and the ground truth (last column), respectively. (Best viewed in color)

When dealing with single-instance object, finding a shape boundary is another challenging in MNC [29] beside miss-labeling problem. As we can see from the second row of Fig. 6.5, MNC cannot segment and recognize the bird in the first image or segment and recognize the cat twice (one is classified as a cat and one is classified as a dog) in the last image. While our CRLS is able to segment and recognize the object with shape boundary, determining good boundary is still a big challenging in MNC approach as given in the second row (MNC) and third row (CRLS).

Multi-instance object: The semantic segmentation becomes more challenging when dealing with multi-instance object. In this scenario, many instances are captured with a small portion or covered by another instance (overlapping problem). It is easy to see that miss-labeling and mistakenly labeling are usual problems in MNC [29] as given in the second row of Fig. 6.6. Notably, in this example, the colors may vary and different between segmenting results and groundtruth because the instances are ordered differently.

The performance of CRLS on PASCAL VOC 2012 validation set with mean Average Precision (mAP) (%) per each class is given in Fig. 6.8 whereas the performance of CRLS on MS COCO test-dev set with Average Precision and Average Recall is given in Fig. 6.9.

The metrics used for COCO dataset are defined as follows. Average Precision (AP):

- mAP: AP at IoU=.50:.05:.95 (primary challenge metric)
- AP@.50 : AP at IoU=.50 (PASCAL VOC metric)
- AP@.75 : AP at IoU=.75 (strict metric)

AP Across Scales:

- AP^S: AP for small objects: area $< 32^2$
- AP^M: AP for medium objects: $32^2 <$ area $< 96^2$
- AP^L: AP for large objects: area $> 96^2$

Average Recall (AR):

- AR^1: AR given 1 detection per image

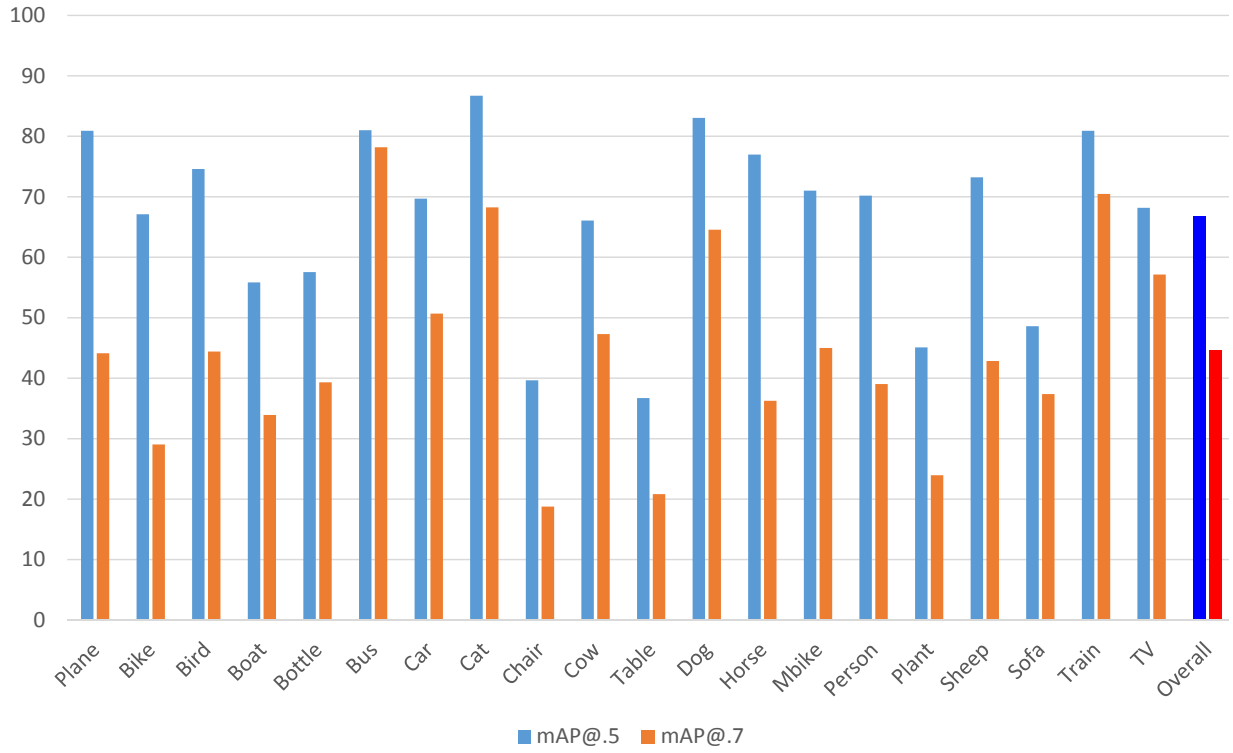


Figure 6.8: The performance of CRLS on PASCAL VOC 2012 validation set with mean Average Precision (mAP) (%) per each class

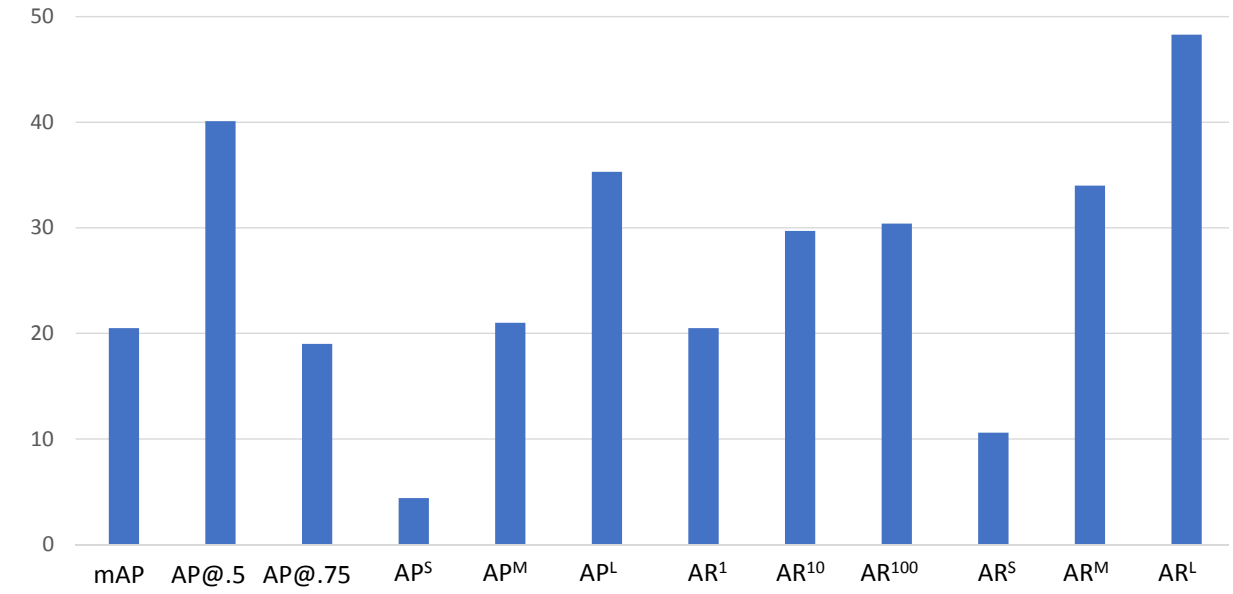


Figure 6.9: The performance of CRLS on MS COCO 2015 test-dev set with Average precision and Average recall

- AR^{10}: AR given 10 detections per image
- AR^{100}: AR given 100 detections per image

AR Across Scales:

- AR^{S}: AR for small objects: area $< 32^2$
- AR^{M}: AR for medium objects: $32^2 <$ area $< 96^2$
- AR^{L}: AR for large objects: area $> 96^2$

6.2 Experimental results of CRRN

We evaluate the efficiency of our proposed CRRN on four challenging and popular scene image labeling datasets, i.e. SiftFlow [246], CamVid [247], Stanford Background [235] and SUN [248].

6.2.1 Datasets

SiftFlow The SiftFlow dataset [246] contains 2,688 images captured from 8 typical outdoor scenes, i.e. coast, forest, highway, inside city, mountain, open country, street, tall building. Each image is 256×256 and labeled with 33 semantic classes (ignore the background). To run the experiment, the dataset is separated into 2 subsets corresponding to 2,488 images for training and the rest for testing as in [56].

CamVid The Camvid dataset [247] is a road scene dataset which contains 701 high-resolution images of 4 driving videos captured at both day and dusk modes [232]. Each image is 960x720 and is labeled with 32 semantic classes. We follow the usual split protocol as given in [249] with 468 images for training and the rest for testing.

Stanford Background The Stanford background dataset [235] contains 715 images annotated with 8 semantic classes. The dataset is randomly partitioned into 80% (572 images) for training and the rest (143 images) for testing with 5-fold cross validation.

SUN The Sun dataset [248] contains 16,873 images annotated with 3,819 semantic object categories. However, the numbers of images per category are highly unbalanced. Therefore, we only choose the images that has more than 60% of their pixels are labeled for evaluation. The chosen dataset with these categories contains 6,433 images and is randomly partitioned into two subsets corresponding to 5,798 images for training and the rest for testing.

6.2.2 Metrics

As for scene labeling, it has become common practice to report results using two metrics, namely, per-pixel accuracy (PA) and average per-class accuracy (CA). The first metric, i.e. PA is defined as the fraction of the number of pixels classified rightly over the number of pixels to be classified in total whereas the latter metric, i.e. CA, is defined as the average of per-pixel accuracy of all the classes existing in the dataset.

$$\begin{aligned} PA =& \sum_i n_{ii} / \sum_i t_i \\ CA =& (1/C) \sum_i (n_{ii}/t_i) \end{aligned} \tag{6.3}$$

n_{ij} is the number of pixels of class i that were predicted to be class j, C is total number of classes and $t_i = \sum_j n_{ij}$ is the total number of pixels of class i.

6.2.3 Experiment - Scene Labeling by CRRN

In order to make a fair comparison between our proposed CRRN and other state-of-the-art models, we divide the models into two groups: the ones trained using the benchmark datasets only; and the ones fine-tuned from other models which are trained over the large-scale datasets (i.e. ImageNet) or made used of deep network (i.e. VGG-verydeep-16) as their feature extractor. *Our CRRN falls into the first group where no pre-trained model is*

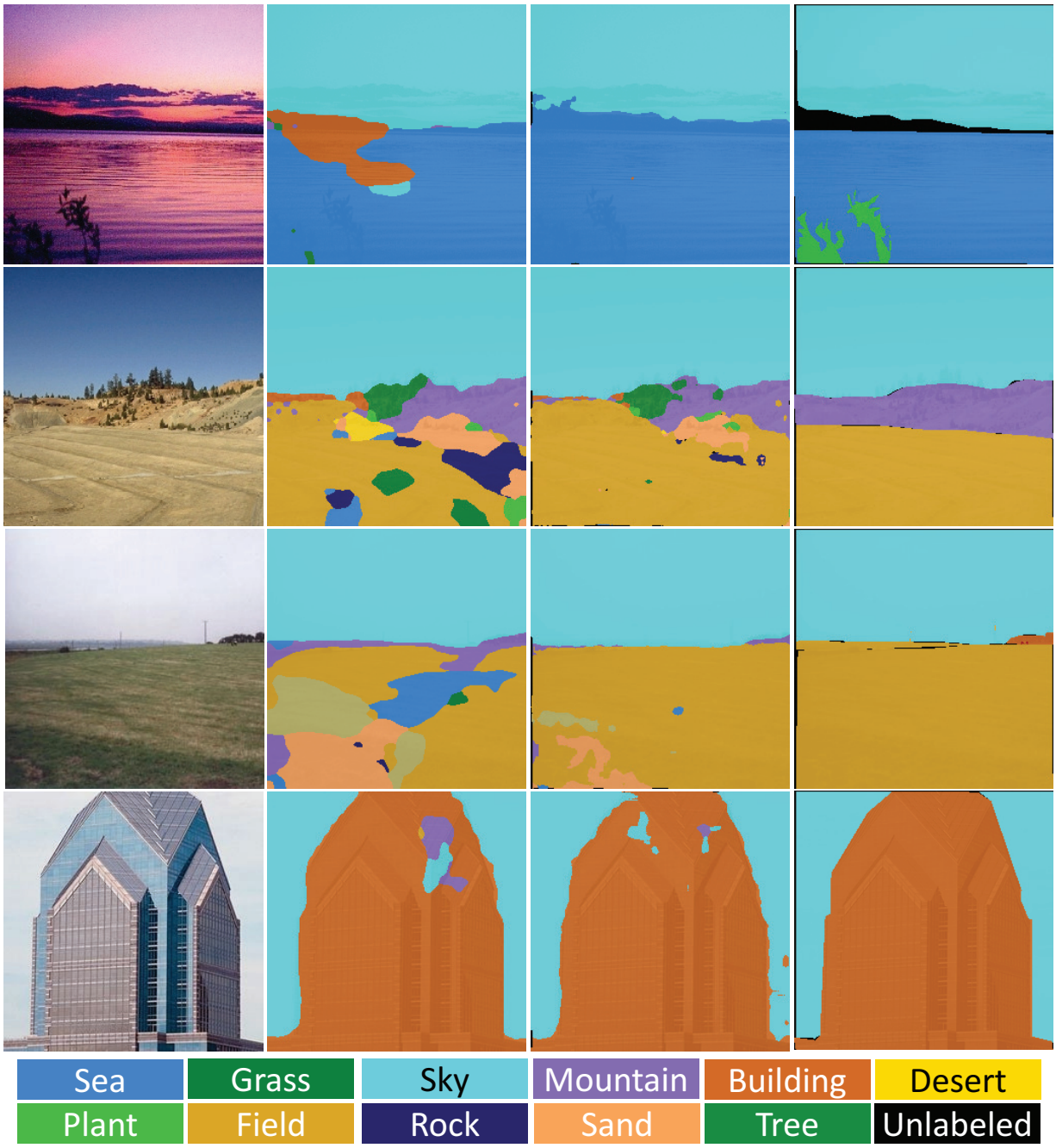


Figure 6.10: Comparison between our CRRN and CNN-65-DAG-RNN[56] in term of contextual dependencies learning. In each row, there are four images: input image (first column), the prediction labeling map of [56] (second column), the contextual labeling map extracted from our CRRN (third column) and the ground truth labels. **(Best viewed in color)**

used.

Comparing to the methods in the first group, the quantitative results of our approach

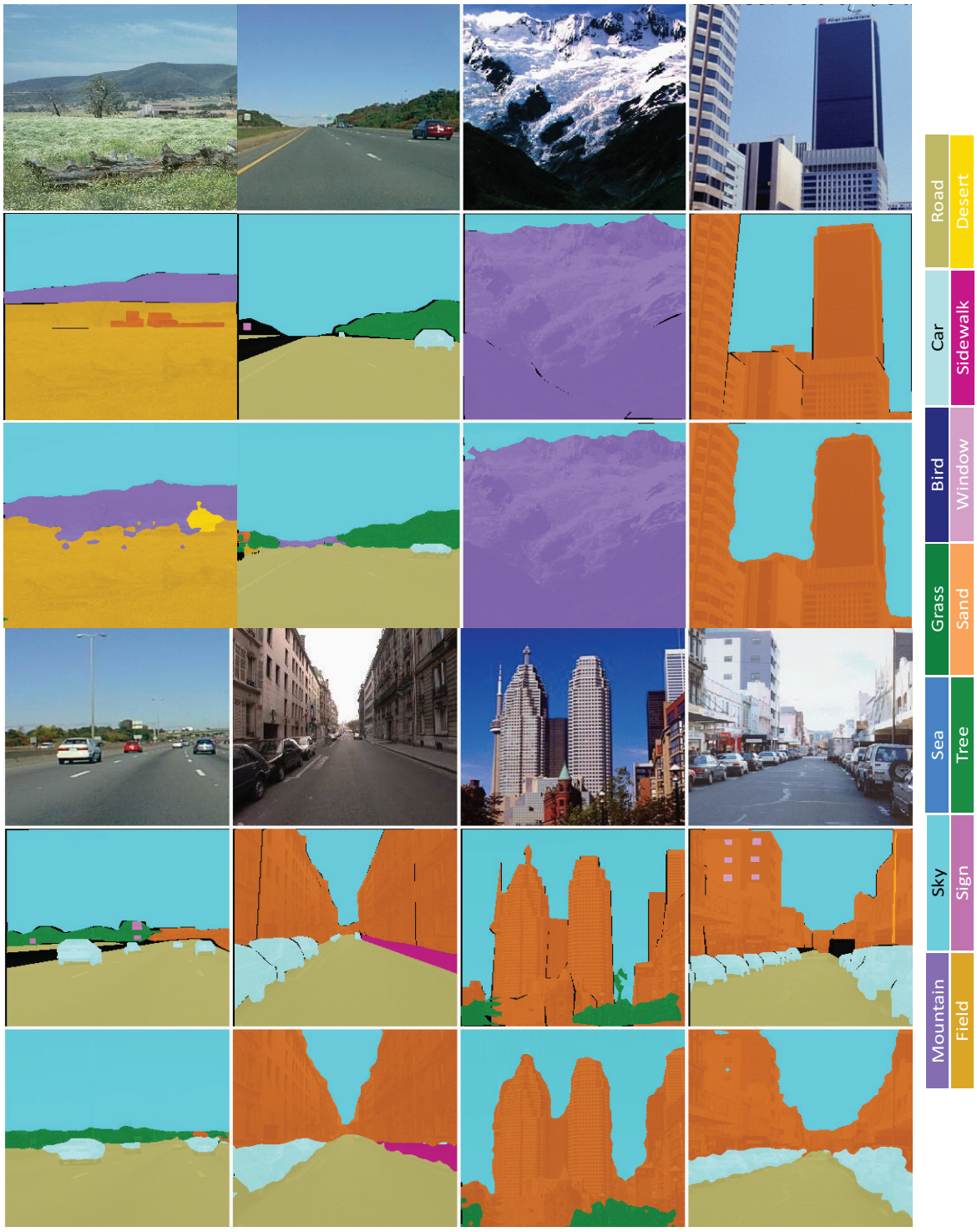


Figure 6.11: Examples of labeling results on SiftFlow dataset. Each column has six images: the input image (first and fourth rows), its ground truth labels (second and fifth rows) and our CRRN prediction labels (third and sixth rows), respectively. **(Best viewed in color)**

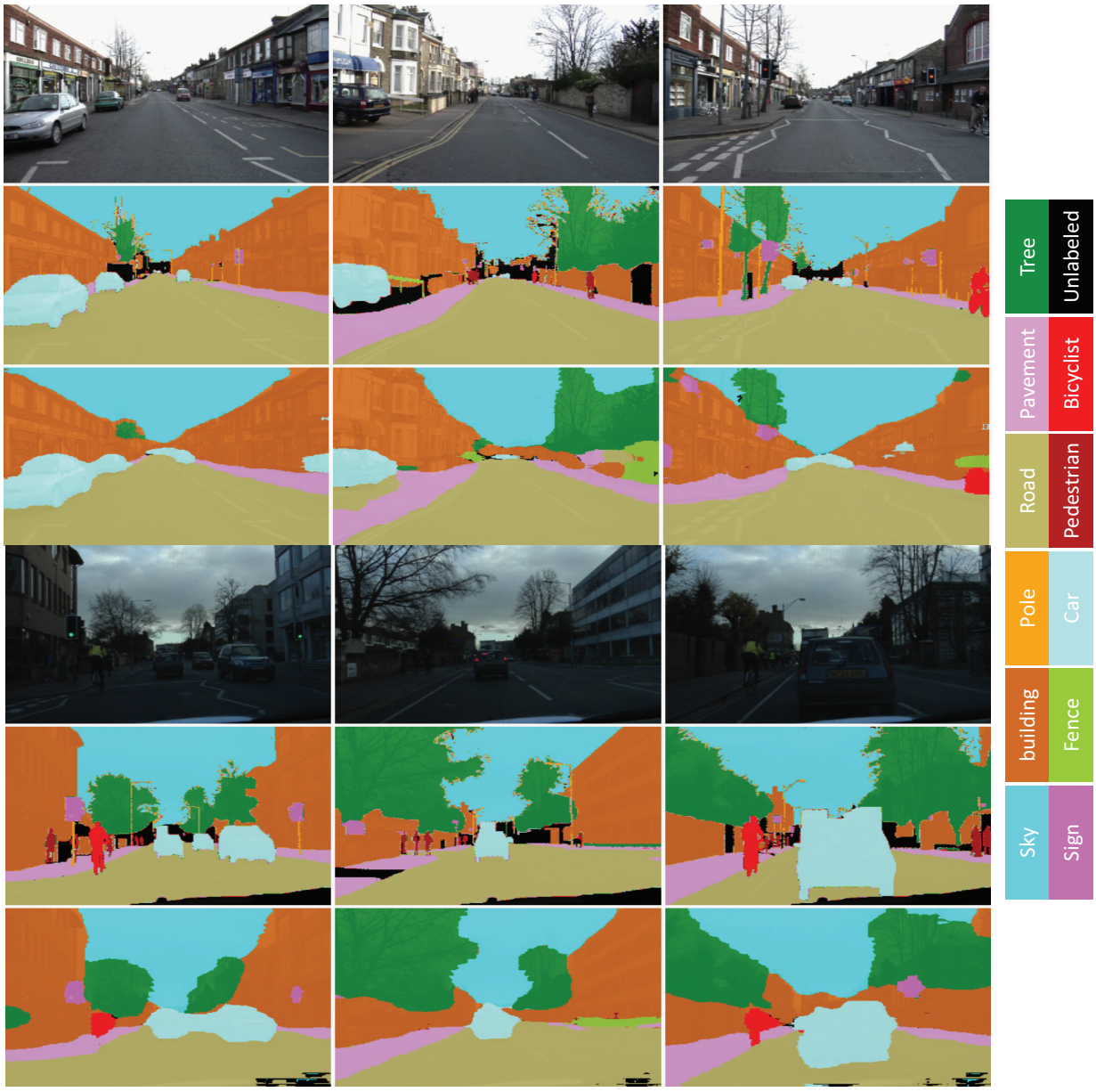


Figure 6.12: Examples of labeling results on CamVid dataset. Each column has six images: the input image (first and fourth rows), its ground truth labels (second and fifth rows) and our CRRN prediction labels (third and sixth rows), respectively. **(Best viewed in color)**

with three benchmark datasets namely, Siftflow, Camvid, Stanford and SUN are reported in Tables 6.4, 6.5, 6.6 and 6.7, respectively. The empirical results on three datasets show that our performance on both PA and CA scores are higher than state-of-the-art methods on larger dataset while giving quite competitive results on the smaller dataset. On larger Siftflow when we have big enough data for training, our proposed CRNN outperforms all

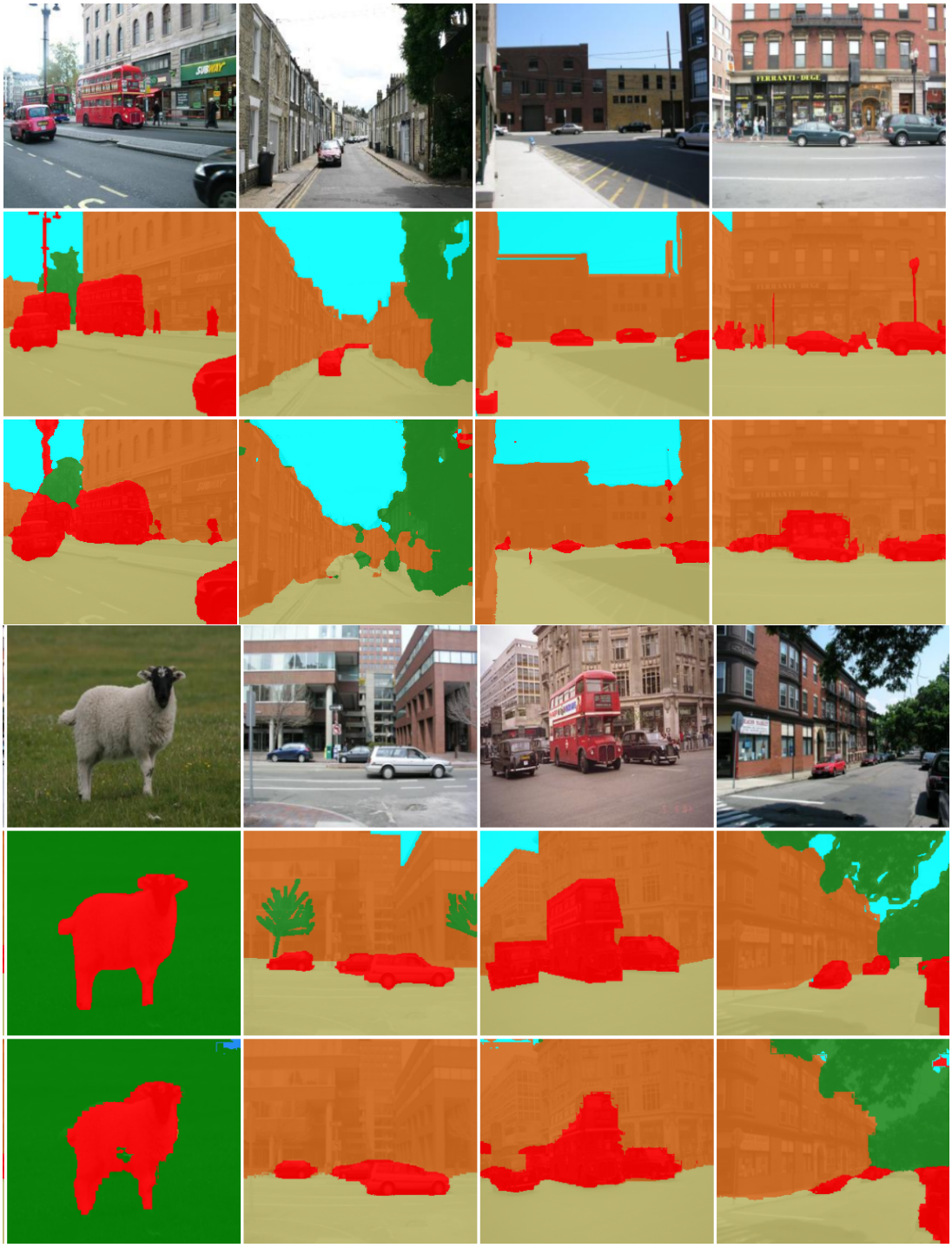


Figure 6.13: Examples of labeling results on Stanford dataset. Each column has six images: the input image (first and fourth rows), its ground truth labels (second and fifth rows) and our CRRN prediction labels (third and sixth rows), respectively. **(Best viewed in color)**

Figure 6.14: Examples of labeling results on SUN dataset. Each column has six images: the input image (first and fourth rows), its ground truth labels (second and fifth rows) and our CRRN prediction labels (third and sixth rows), respectively. **(Best viewed in color)**

Table 6.4: Quantitative results and comparisons against *non-fine-tuned models* **2D-LSTM** [3], **RCNN$_3$1/1 resolution (^{3})** [53], **Multi-scale Convnet** [59], **Multi-CNN - rCPN** [54], **Scene Graph Structure** [55], **CNN-65-DAG-RNN** [56],**Context&Attention** [49], **Recurrent Convolutional Neural Networks (RCNN)**[53], **Sample&Filter + MRF**, [57], **Multi-scale RCNN** [58] on Siftflow dataset.

METHOD	PA	CA
2D-LSTM	70.1%	20.9%
RCNN$_3$1/1 resolution (^{3})	77.7%	29.8%
Multi-scale Convnet	78.5%	29.6%
Multi-CNN - rCPN	79.6%	33.6%
Scene Graph Structure	80.6%	45.8%
CNN-65-DAG-RNN	81.1%	48.1%
Context&Attention	79.8%	48.7%
Sample&Filter + MRF	83.1%	44.3%
Multi-scale RCNN	83.5%	35.8%
Our CRRN	**84.7%**	**61.0%**

Table 6.5: Quantitative results and comparisons against *non-fine-tuned models* **Neural Decision Forest**s[250], **SVM+MRF**[249] on CamVid dataset.

METHOD	PA	CA
Neural Decision Forests	82.1%	56.1%
SVM+MRF	83.9%	**62.5%**
Our CRRN	**84.4%**	54.8%

other models in terms of both PA and CA scores. Our CA is 61.0% and PA is 84.0% compared to the next highest CA score of 48.1% [56] and PA score of 83.5% [58] as shown in Tables 6.4. On the small dataset, take Camvid as an instance, when we just have about 468 images for training, our PA is still about 0.5% higher than the state-of-the-art method [249]. On larger scale dataset such as SUN, our CRRN also achieves 1.41% higher than FCNN [31] in term of PA score.

Figure 6.10 illustrates the advantages of our CRRN in term of modeling the contextual

Table 6.6: Quantitative results and comparisons against *non-fine-tuned models* **2D-LSTM** [3], **Recurrent CNN** [53], **Multi-scale Convnet**(the best) [59], **Multi-scale RCNN** [58], **Associative Hierarchical Random Fields**[251] **Scene Graph Structure** [55], Hierarchical Random Fields [251] on Stanford-background Dataset

METHOD	PA	CA
2D-LSTM	78.6%	68.8%
$RCNN_3$1/1 resolution (^{3})	80.2%	69.9%
Multi-scale Convnet	81.4%	76.0%
Multi-scale RCNN	83.1%	74.8%
Associative Hierarchical Random Fields	80.9%	70.4%
Scene Graph Structure	84.6%	**77.3%**
Hierarchical Random Fields	80.9%	70.4%
Our CRRN	**85.23%**	75.2%

Table 6.7: Quantitative results and comparisons against *non-fine-tuned models* **CNN-65-DAG-RNN** [56], **FCNN** [31] on SUN dataset.

METHOD	PA	CA
CNN-65-DAG-RNN	71.51%	54.57%
FCNN	77.2%	**62.03**%
Our CRRN	**78.61**%	59.9%

Table 6.8: Quantitative results and comparisons against **CNN - Global Context** [1], **FCNN** [31], **VGG-conv5-DAG-RNN** [56] with *fine-tuned models* on Siftflow dataset.

METHOD	PA	CA
CNN - Global Context	80.1%	39.7%
FCNN	85.2%	51.7%
VGG-conv5-DAG-RNN	**85.3%**	55.7%
Our CRRN	84.7%	**61.0%**

dependency presented in the image. Comparing to [56], besides the local consistency between neighborhood regions, their semantic coherence is better enhanced in our CRRN. For example, after training, our CRRN can capture the contextual knowledge such as the

'building' is not likely to appear in the 'sea' (i.e. the first case) or the 'desert' is not usually covered by the 'field' (i.e. the second case). As a result, while [56] still has the problem of misclassification in its predictions, smoother and better labeling maps can be produced by our model. Our qualitative results on the datasets are further demonstrated in Figures 6.11, 6.12 , 6.13 and 6.14. As visualization, there exist many false positive cases where pixels are classified as unlabeled while they truly belongs to labeled classes. In such cases, our CRRN performs well to category the false positive pixels. Thank to the ability of simultaneously learning rich visual representation and modeling the context information, our model has better memory of contextual dependency and gives higher discriminative descriptors for each local patch

Furthermore, we also compare our CRRN with other fine-tuned methods on SiftFlow as shown in Table 6.8. In this group, most existing fine-tune methods [1, 31, 56] are trained on large-scale data and make use of powerful feature extractor while our CRRN is a non fine-tune model and trained on Siftflow only. From these results, one can see that our CRRN model archives the best CA score (61.0%) compared to the state-of-the-art (55.7%) [56] while giving a competitive PA score. We believe that given enough data, our CRRN performance can be boosted and is competitive to these models. We leave this as future work.

6.2.4 Analysis on false cases

In the following sections, we further study the scenarios where the segmenting results receive low pixel accuracy and low class accuracy corresponding to two cases of false positive where the groundtruth is mistakenly labeled as unlabeled and confusing where two labels are the same meaning

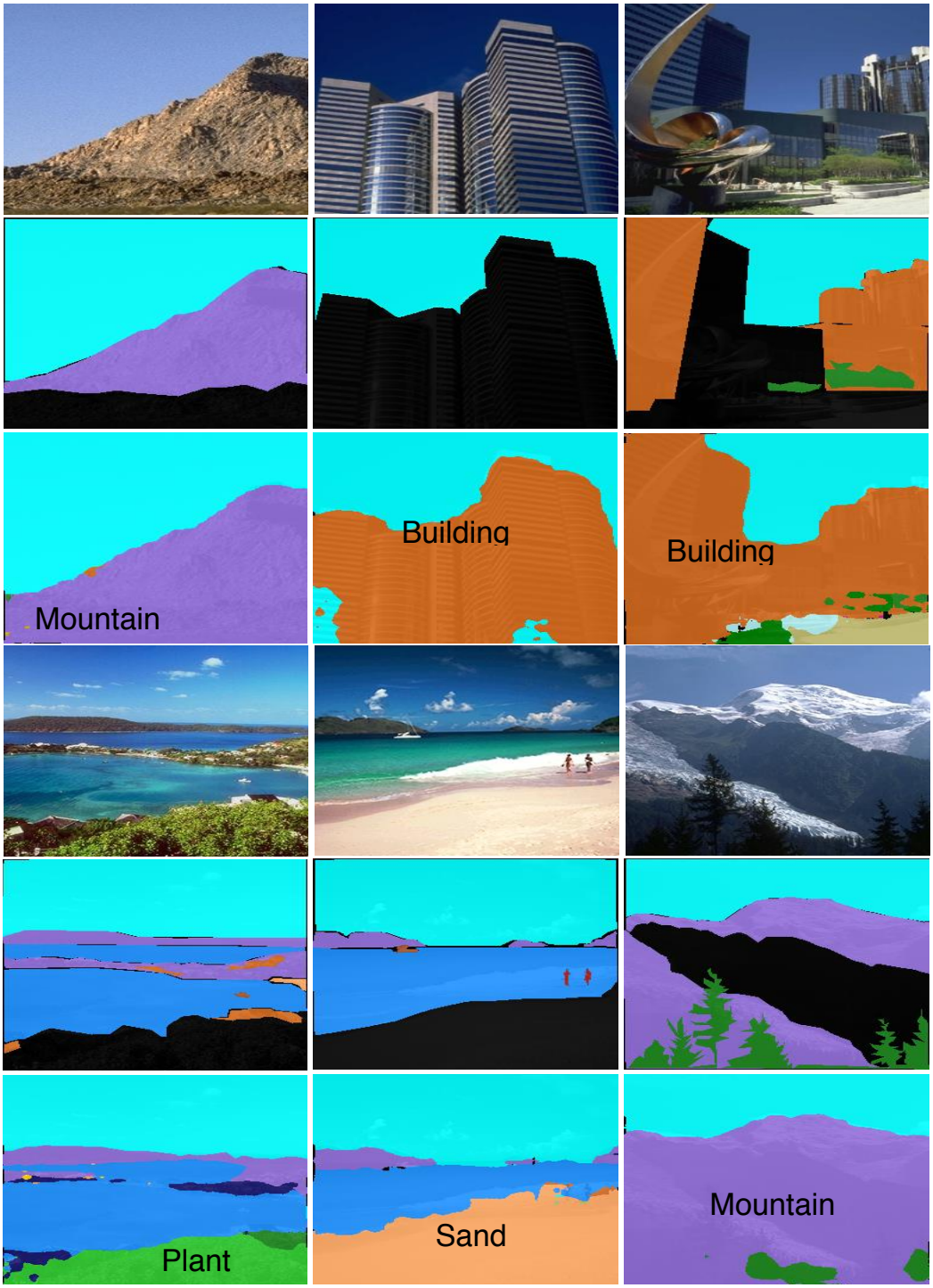


Figure 6.15: Examples of false positive where the ground truth is mistakenly labeled as unlabeled. Images are from Siftflow. For each column: the input image (first row and fourth), its ground truth labels (second row and fifth row) and our CRRN prediction labels (third row and sixth row), respectively. **(Best viewed in color)**

False Positive

The experiments conducted on the benchmark datasets have shown a case where the ground truth is mistakenly classified as unlabeled class. Some examples of the false positive case are given in Fig. 6.15 where images are from SiftFlow. As demonstrated in the second row of Fig. 6.15, many classes i.e. mountain, building, tree, sand are marked as unlabeled class. However, our CRRN is able to segment and classify them into correct labels as given in the third row. In this scenario of false positive, the accuracy scores are low but it does not mean that our segmenting results are incorrect.

Ambiguous Context

In addition to false positive, we are studying another circumstance where classes are ambiguously labeled. Take the first image in Fig. 6.16 as an instance. The "grass field" is labeled as "field" in the ground truth whereas it is classified as "grass" by our CRRN. In the second and the third images, "tree" and "plant" are ambiguously labeled. In such case, it is acceptable to recognize it as either "tree" or "plant". In the next three images, the "mountain" and "tree" are confusing because tree are growing on the mountain. Therefore, labeling this class as either "mountain" or "tree" is correct.

The study on ambiguous context scenario illustrates the advantage of our CRNN in term of capturing the contextual dependency. We found that even the pixel accuracy and class accuracy are low, the segmenting results are really feasible.

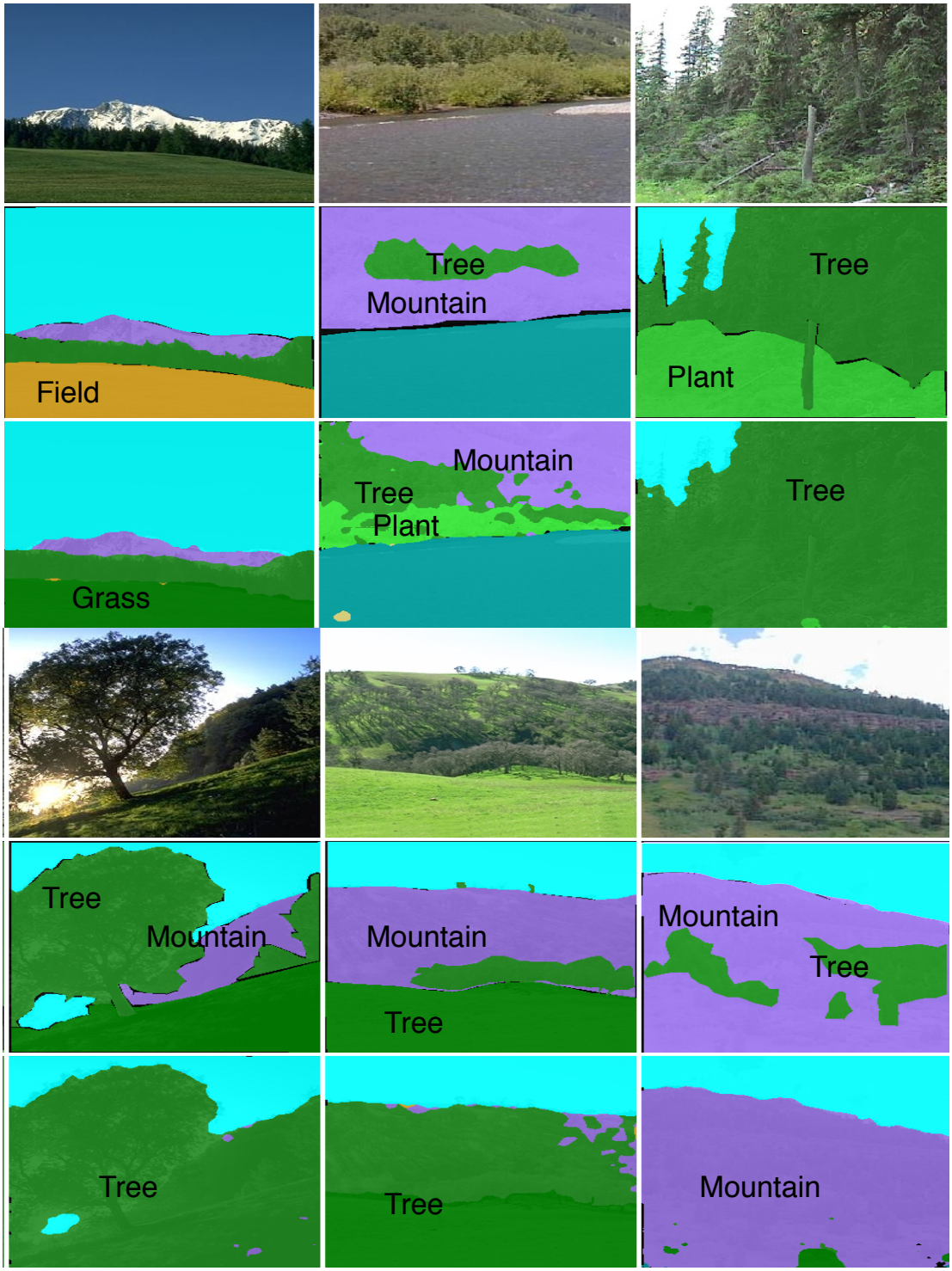


Figure 6.16: Examples of context confusing. For each column: the input image (first row and fourth), its ground truth labels (second row and fifth row) and our CRRN prediction labels (third row and sixth row), respectively. **(Best viewed in color)**

Chapter 7

Concluding Remarks and Future Work

7.1 Concluding Remarks

This thesis proposed two deep learning-based approaches for pixel-level segmentation. The first approach, Contextual Recurrent Level Set (CRLS) Networks for semantic instance segmentation, has built the bridge between two unconnected areas, namely, the pure image processing based variational LS methods and the learnable deep learning approaches. A novel contour evolution RLS approach has been proposed by employing GRUs under the energy minimization of a variational LS functional. The learnable RLS model is designed as a building block that enable RLS easily incorporate into other deep learning components. Using shared convolutional features, RLS is extended to contextual RLS (CRLS) which combines detection, segmentation and classification tasks within a unified deep learning framework to address multi-instance object segmentation in the wild. As we have seen that successfully reformulating the optimization process of classic LS as a recurrent neural network is highly beneficial. In addition to solving the LS-based segmentation problem, it is worthwhile to attempt doing the same for other iterative problems. It means our work not

only boosts the classic LS approaches to new level of deep learning but also provides a novel view of point for iterative problems that not yet explored. The experimental results show that the proposed RLS method outperforms the baseline approaches such as Chan-Vese, DRLSE, Li's, L2S and the F-ConNet methods on object segmentation task. Furthermore, the experiments on PASCAL VOC and MS COCO concludes that the proposed CRLS gives competitive performance on semantic instance segmentation in the real world images. To the best of our knowledge, our proposed fully end-to-end RLS method and CRLS system are the first Level Set based methods able to deal with real-world challenging databases, i.e. PASCAL VOC and MS COCO, with highly competitive results. Our work potentially provides a vehicle for further studies in level set and iterative problems in future.

The second approach, Contextual Recurrent Residual Networks (CRRN), is able to simultaneously model the long-range context dependencies and learn rich visual representation. The proposed CRRN is designed as a fully end-to-end deep learning framework and is able to make use the advantages of both sequence modeling and residual learning techniques. Our CRRN contains three parts corresponding to sequential input data, sequential output data and hidden CRRN unit. Each hidden CRRN unit has two main components: context-based component to model the context dependencies and visual-based component to learn the visual representation. Our proposed end-to-end CRRN is completely trained from scratch, without using any pre-trained models in contrast to most existing methods usually fine-tuned from the state-of-the-art pre-trained models, e.g. VGG-16, ResNet, etc. The experiments are conducted on four challenging scene labeling datasets, i.e. SiftFlow, CamVid, Stanford background and SUN datasets, and compared against various state-of-the-art scene labeling methods.

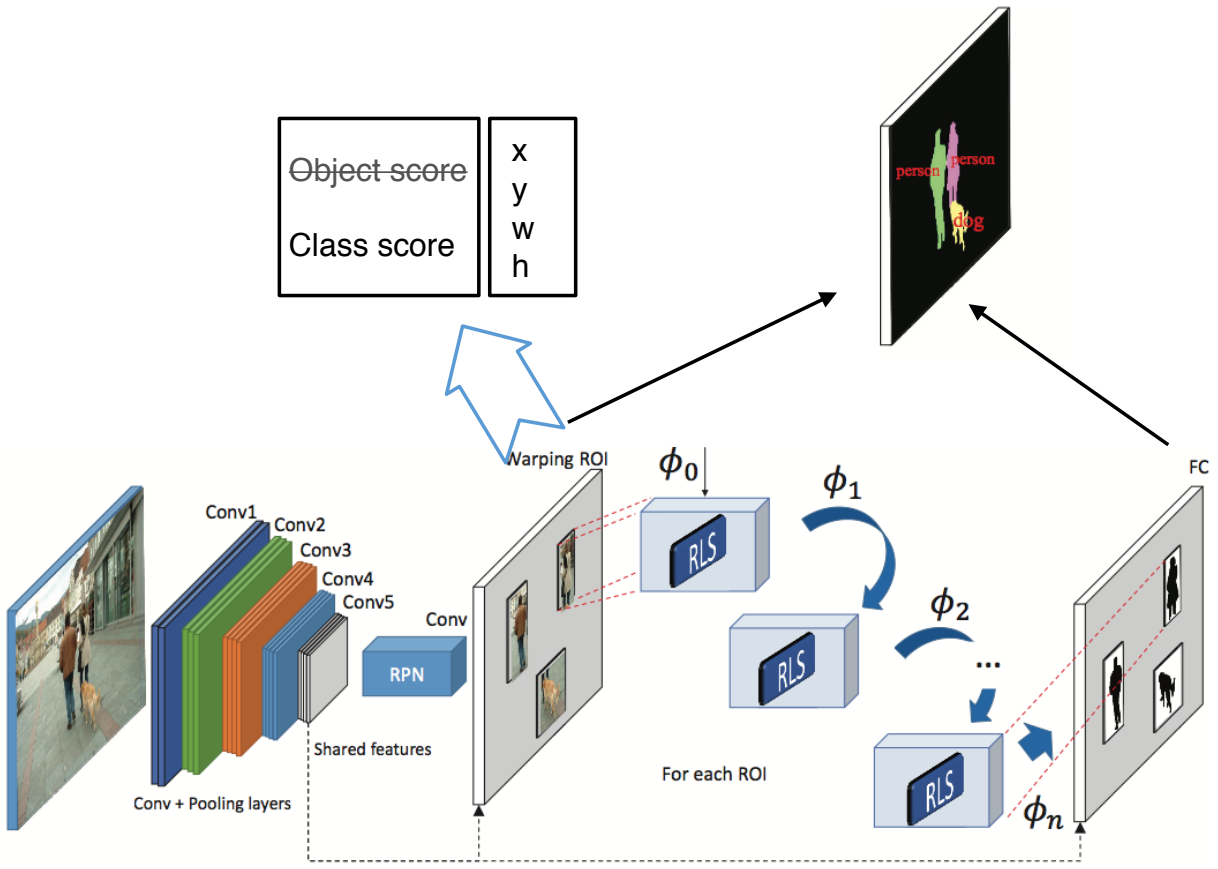


Figure 7.1: An improved version of CRLS Networks

7.2 Future Work

The proposed CRLS has archived good performance on PASCAL VOC 2012 and MS COCO 2014 databases. However, CRLS has some limitations which will be improved in the future work.

- The size of region proposal is fixed as 21×21 and this limitation effects to the segmentation accuracy and the detection as well. Remove the ROI warping step is doable to boost up the performance.
- There are two classification procedures which perform similar tasks, namely object detection procedure in the first stage to compute the objectness score and the image classification in the third stage to compute the class score. The object classification

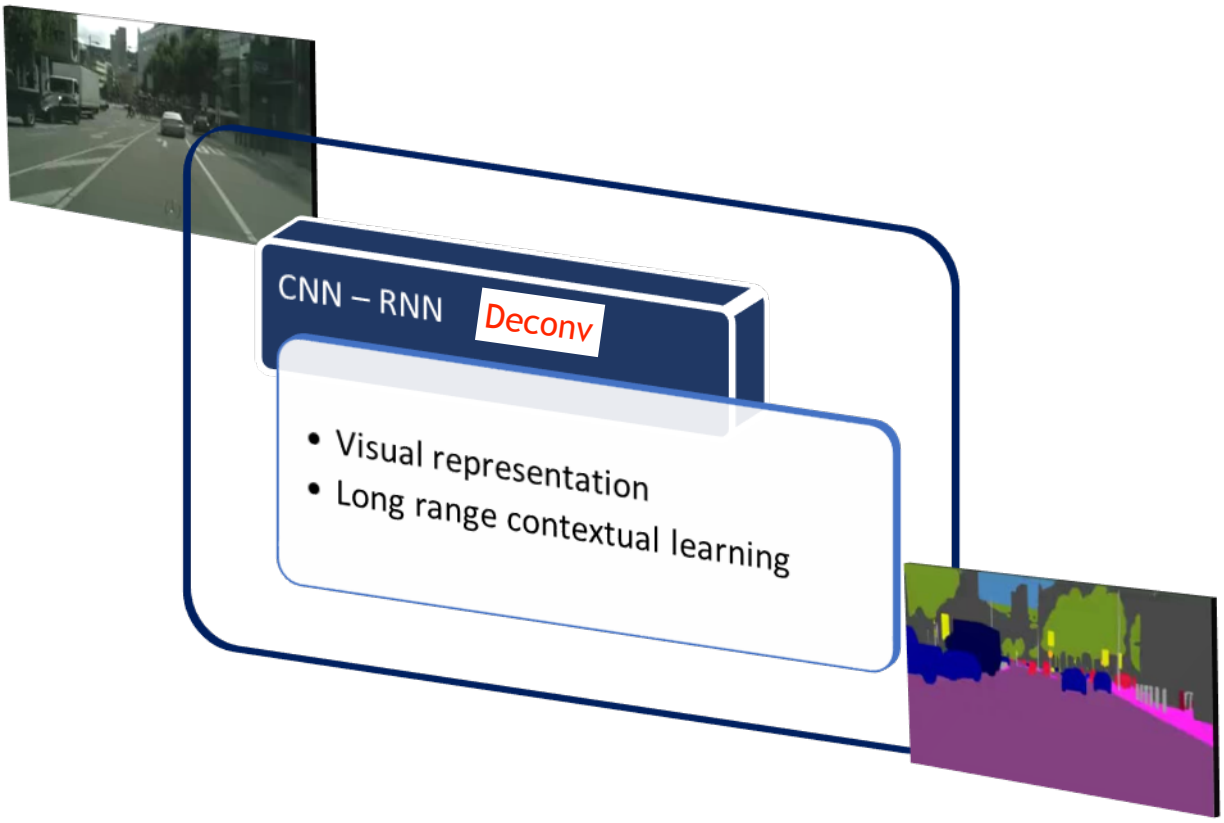


Figure 7.2: An improved version of CRRN

is basically a special case of the mage classification. Therefore, it is possible to combine these two tasks into one step to reduce the time consuming. The simple suggested solution is computing class score instead of objectness score. Therefore, we can remove the third stage and combine it into the first stage.

- There is no transposed convolution used in CRLS therefore the segmentation results on boundary is less accurate. The final feature map should be deconvolutioned by the transposed convolution to improve the performance, specially on the object boundary.
- The entire system is implemented on Caffe framework which has some limitations about the size and memory. Network design has a difficulty when setting the number of iterations once being implemented in Caffe. This drawback is completely solve if the CRLS is reimplemented in other environments like MXNET or Pytorch.

The improved version of CRLS for handling the above limitations is illustrated in Fig. 7.1.

Furthermore, we are also looking for the problem of segmenting the unseen objects as shown in Fig. 7.3 and Fig. 7.4. There are two scenarios related to this problem:

- Segment an unseen object which shares similar texture with the objects in the training dataset as shown in Fig. 7.3 where the tiger's texture is similar to zebra's texture and leopard's texture is similar to giraffe's texture. In such scenario, the proposed CRLS is able segment the object but it is unable to correctly classify because the label class doesnt exist in the training.
- Segment an unseen object which does not share similar texture with any object in the training dataset as shown in Fig. 7.4. In such case, our algorithm is able to neither segment nor classify because it does not learn such type of texture during the training procedure.

Figure 7.3: The first example of segmenting results on unseen objects which share similar texture with the trained objects. The first row: original images. The second rows: segmenting results by CRLS

While being designed as an effective approach for scene labeling, CRRN has the following disadvantages. An improved version of the proposed CRRNs is illustrated in Fig. 7.2.

- In the current version of CRRN, the upsampling is applied separately after CRNN. An suggested solution is incorporating a transposed convolution in the end of each CRRN

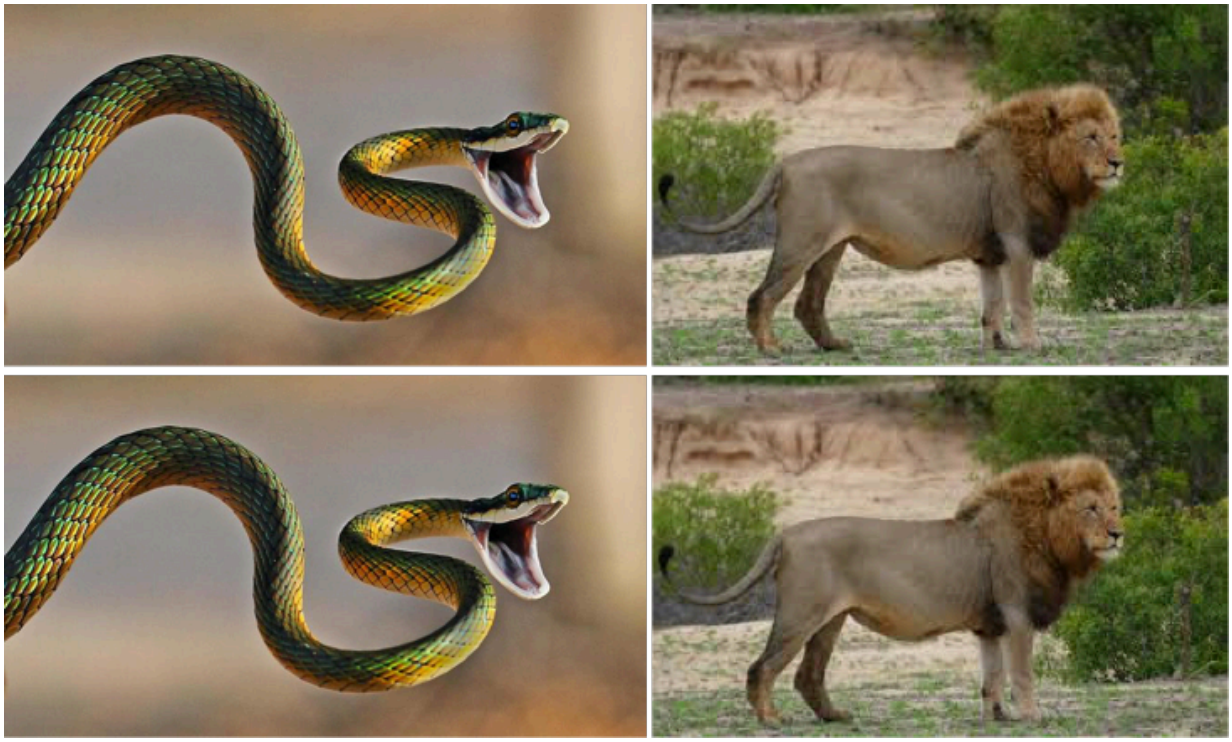


Figure 7.4: The second example of segmenting results on unseen objects which do not share similar texture with any object in the training dataset. The first row: original images. The second rows: segmenting results by CRLS

hidden unit, namely each CRRN hidden unit contains three components (context based, visual-based and upsampling-based) instead of two components.

- In CRRN, the input image is partitioned into non-overlapping blocks. The size of each block as well as the size of the input image affect to network size (memory requirement) and pixel labeling accuracy. Given an image, the smaller size of block gives the better performance and requires bigger memory.
- According to the experiment, the scene labeling accuracy is affected by confusing labeling where a same pixel is labeled with many synonym class names. In such circumstance, hierarchical model may be a good solution for scene labeling problem.

In this approach, we further study the cross databases problem where the algorithm is trained on one city and is tested on another unseen city as shown in Fig. 7.5 and Fig. 7.6. There are two scenarios related to this problem:

- The unseen city contains objects that are similar with the objects in the training set as given in Fig. 7.5. For example, the model is trained on the images from Pittsburgh street view and is tested on the images from New Orleans. Clearly, New Orleans is very similar to Pittsburgh in regards to city layout and they both contains similar objects such as car, traffic light, street, tree. Thus, the proposed CRRN is able to segment and label most of the objects in the unseen city.
- The unseen city contains objects and city layout that are different from the objects in the training set as given in Fig. 7.6. In the training set, there is no scooter/bicycle or there is no auto rickshaw as shown in the testing set. In such case, CRRN is able to label the objects (like people) that exist in the training set, but it is unable to label the scooter or bicycle. In the left image of Fig. 7.6, depend on the view, the auto rickshaws sometimes look similar a car, thus, CRRN give the wrong label (label it as a car).

These experimental results indicate that there is a future work to be perform in the area of domain adaptation and network fine tuning to enable these network to generalize across different datasets and city layout.

Figure 7.5: The first examples of cross databases problem where the model is trained on SUN database and tested on CamVid database. First row: images from CamVid database. Second row: Scene labeling results by our CRRN

Figure 7.6: The second examples of cross databases problem where the model is trained on SUN database and tested on images from Internet. First row: images from Internet. Second row: Scene labeling results by our CRRN

Appendix A

Appendix

Most of the works related to CLS proposed Chan Vese [62] have made use the final equations of curve evolution but none of researches provides the detailed derivative. In order to build CLS under a deep framework, it is necessary to know all the derivatives in details. Therefore, we first bring the curve evolution in CLS and its derivative in A.1. We then show the derivatives our proposed Recurrent Level Set (RLS) in Sections 4.2.2 with forward pass and backward pass procedures. We have added some more semantic segmentating results on various cases i.e. overlapping objects, single-instance object, multi-instance object from PASCAL VOC and MS COCO databases. More illustrations for the comparison between MNC performance [29] and ours is also showed here.

In this section, we also provide in details the segmentation performance (average precision %) on each category at IoU = 0.5 on PASCAL VOC 2012. The performance of CRLS on MS COCO database is also showed in details with different measures, i.e. average precision, average precision across scales, average recall, average recall across scales.

A.1 Derivation of Level Set

The value of constants c_1 and c_2 can be derived by setting the derivative of E w.r.t. c_1 and c_2, respectively: The energy functional defined as follows:

$$\begin{aligned} E(c_1, c_2, \phi) &= \mu \int_{\Omega} H(\phi(x,y))dxdy \\ &+ \nu \int_{\Omega} \delta(\phi(x,y))|\nabla\phi(x,y)|dxdy \\ &+ \lambda_1 \int_{\Omega} |\mathbf{I}(x,y) - c_1|^2 H(\phi(x,y))dxdy \\ &+ \lambda_2 \int_{\Omega} |\mathbf{I}(x,y) - c_2|^2 (1 - H(\phi(x,y)))dxdy \end{aligned} \tag{A.1}$$

Derivative of E w.r.t. c_1:

$$\begin{aligned} &\frac{\partial E}{\partial c_1} = \int_\Omega 2\lambda_1(\mathbf{I}(x,y) - c_1)H(\phi(x,y))dxdy = 0 \\ &\Rightarrow c_1 = \frac{\int_\Omega \mathbf{I}(x,y)H(\phi(x,y))dxdy}{\int_\Omega H(\phi(x,y))dxdy}, \end{aligned} \tag{A.2}$$

Derivative of E w.r.t. c_2:

$$\begin{aligned} &\frac{\partial E}{\partial c_2} = \int_\Omega 2\lambda_2(\mathbf{I}(x,y) - c_2)(1 - H(\phi(x,y)))dxdy = 0 \\ &\Rightarrow c_2 = \frac{\int_\Omega \mathbf{I}(x,y)(1 - H(\phi(x,y)))dxdy}{\int_\Omega (1 - H(\phi(x,y)))dxdy}. \end{aligned} \tag{A.3}$$

The optimal level set function ϕ can be computed by solving the associated Euler-Lagrange equation. Here, keeping c_1 and c_2 fixed,

$$E = \int_\Omega f(\phi, \nabla\phi)dxdy, \tag{A.4}$$

then the Euler-Lagrange equation is given by

$$\frac{\partial f}{\partial \phi} - \nabla \cdot \frac{\partial f}{\partial \nabla \phi} = 0. \tag{A.5}$$

We compute each term in the above equation as follows:

$$\begin{aligned} \frac{\partial f}{\partial \phi} &= \frac{\partial}{\partial \phi}\left(\nu\delta(\phi)|\nabla\phi| + \mu H(\phi) + \lambda_1|\mathbf{I} - c_1|^2 H(\phi)\right. \\ &\quad \left. +\lambda_2|\mathbf{I} - c_2|^2(1 - H(\phi))\right) \\ &= \nu|\nabla\phi|\frac{\partial\delta(\phi)}{\partial\phi} + \mu\delta(\phi) + \lambda_1|\mathbf{I} - c_1|^2\delta(\phi) \\ &\quad - \lambda_2|\mathbf{I} - c_2|^2\delta(\phi) \end{aligned} \tag{A.6}$$

in which, the first term vanishes since we care about the zero level set ($\phi = 0$).

$$
\begin{aligned}
|\nabla\phi| &= \sqrt{\phi_x^2 + \phi_y^2} \\
\frac{\partial}{\partial\phi_x}|\nabla\phi| &= \frac{2\phi_x}{2\sqrt{\phi_x^2 + \phi_y^2}} = \frac{\phi_x}{|\nabla\phi|} \\
\frac{\partial}{\partial\phi_y}|\nabla\phi| &= \frac{2\phi_y}{2\sqrt{\phi_x^2 + \phi_y^2}} = \frac{\phi_y}{|\nabla\phi|},
\end{aligned}
$$

$$
\begin{aligned}
\Rightarrow \frac{\partial}{\partial\nabla\phi}|\nabla\phi| &= \frac{\nabla\phi}{|\nabla\phi|} \\
\frac{\partial f}{\partial\nabla\phi} &= \nu\delta(\phi)\frac{\nabla\phi}{|\nabla\phi|}.
\end{aligned}
$$

Thus,

$$
\begin{aligned}
&\frac{\partial f}{\partial\phi} - \nabla\cdot\frac{\partial f}{\partial\nabla\phi} = \\
&-\delta(\phi)\left\{\nu\mathrm{div}\left(\frac{\nabla\phi}{|\nabla\phi|}\right) - \mu - \lambda_1|\mathbf{I} - c_1|^2 + \lambda_2|\mathbf{I} - c_2|^2\right\} = 0.
\end{aligned}
$$

The above equation is valid when ϕ is the optimal solution. Parameterizing the descent direction by an artificial time $t \geq 0$, we can formulate an iterative update equation for ϕ:

$$
\frac{\partial\phi}{\partial t} = \delta(\phi)\left\{\nu\mathrm{div}\frac{\nabla\phi}{|\nabla\phi|} - \mu - \lambda_1|\mathbf{I} - c_1|^2 + \lambda_2|\mathbf{I} - c_2|^2\right\}.
$$

Note that when the time derivative vanishes, ϕ will stop updating.

A.2 Derivatives of Recurrent Level Sets (RLS) forward/backward

A.2.1 Forward Pass Procedure

The forward pass procedure of RLS is calculated by the following equations:

$$\mathbf{z}_t = \sigma(f_z) = \sigma(\mathbf{U}_z \mathbf{x}_t + \mathbf{W}_z \phi_{t-1} + \mathbf{b}_z)$$

$$\mathbf{r}_t = \sigma(f_r) = \sigma(\mathbf{U}_r \mathbf{x}_t + \mathbf{W}_r \phi_{t-1} + \mathbf{b}_r)$$

$$\tilde{\mathbf{h}}_t = tanh(f_o) = tanh(\mathbf{U}_\phi \mathbf{x}_t + \mathbf{W}_o (\phi_{t-1} \odot \mathbf{r}_t) + \mathbf{b}_o)$$

$$\phi_t = (1 - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t + \mathbf{z}_t \odot \phi_{t-1}$$

$$\mathbf{O}_t = \mathbf{V}\phi_t + \mathbf{b}$$

$$\hat{\mathbf{y}}_\mathbf{t} = \text{softmax}(\mathbf{O}_t)$$

$$L = -\sum \mathbf{y}_t log \hat{\mathbf{y}}_t$$

where $\mathbf{z}_t, \mathbf{r}_t, \tilde{\mathbf{h}}_t, \phi_t, \mathbf{O}_t, \hat{\mathbf{y}}_\mathbf{t}$, are updated gate, reset gate, candidate memory gate, updated hidden state, intermediate output, output. σ is a sigmoid function and L is the loss function.

A.2.2 Backward Pass Procedure

$$\text{Derivative w.r.t. } \mathbf{O}_t : \Delta\mathbf{y} = \frac{\partial L}{\partial \mathbf{O}_t} = \hat{\mathbf{y}}_t - \mathbf{y}$$

$$\text{Derivative w.r.t. } \mathbf{b} : \frac{\partial L}{\partial \mathbf{b}} = \Delta\mathbf{y}$$

$$\text{Derivative w.r.t. } \mathbf{V} : \frac{\partial L}{\partial \mathbf{V}} = \Delta\mathbf{y}\phi_t$$

$$\text{Derivative w.r.t. } f_o : \nabla f_o = \frac{\partial L}{\partial f_o} = \frac{\partial L}{\partial \mathbf{O}_t}\frac{\partial \mathbf{O}_t}{\partial \phi_t}\frac{\partial \phi_t}{\partial \tilde{\mathbf{h}}_t}\frac{\partial \tilde{\mathbf{h}}_t}{\partial f_o}$$

$$= V^T\Delta\mathbf{y}(1-\mathbf{z}_t)(1-\mathbf{o}_t)\mathbf{o}_t$$

$$\text{Derivative w.r.t. } f_r : \nabla\ f_r = \frac{\partial L}{\partial \mathbf{r}_t} = \frac{\partial L}{\partial \mathbf{O}_t}\frac{\partial \mathbf{O}_t}{\partial \phi_t}\frac{\partial \phi_t}{\partial \tilde{\mathbf{h}}_t}\frac{\partial \tilde{\mathbf{h}}_t}{\partial \mathbf{r}_t}\frac{\partial \mathbf{r}_t}{\partial f_r}$$

$$= V^T\Delta\mathbf{y}(1-\mathbf{z}_t)(1-\mathbf{o}_t)\mathbf{o}_t\mathbf{W}_\phi^T\phi_{t-1}\mathbf{r}_t(1-\mathbf{r}_t)$$

$$\text{Derivative w.r.t. } f_z : \nabla f_z = \frac{\partial L}{\partial \mathbf{z}_t} = \frac{\partial L}{\partial \mathbf{O}_t}\frac{\partial \mathbf{O}_t}{\partial \phi_t}\frac{\partial \phi_t}{\partial \mathbf{z}_t}\frac{\partial \mathbf{z}_t}{\partial f_r}$$

$$= V^T\Delta\mathbf{y}(\phi_{t-1}-\mathbf{o}_t)\mathbf{z}_t(1-\mathbf{z}_t)$$

Derivatives w.r.t. $\mathbf{U}_\phi, \mathbf{W}_\phi, \mathbf{b}_\phi$

$$\text{Derivative w.r.t. } \mathbf{U}_o : \frac{\partial L}{\partial \mathbf{U}_o} = \frac{\partial L}{\partial \mathbf{O}_t}\frac{\partial \mathbf{O}_t}{\partial \phi_t}\frac{\partial \phi_t}{\partial \tilde{\mathbf{h}}_t}\frac{\partial \tilde{\mathbf{h}}_t}{\partial \mathbf{U}_o} = \nabla f_o\mathbf{x}_t$$

$$\text{Derivative w.r.t. } \mathbf{W}_o : \frac{\partial L}{\partial \mathbf{W}_o} = \frac{\partial L}{\partial \mathbf{O}_t}\frac{\partial \mathbf{O}_t}{\partial \phi_t}\frac{\partial \phi_t}{\partial \tilde{\mathbf{h}}_t}\frac{\partial \tilde{\mathbf{h}}_t}{\partial \mathbf{W}_o} = \nabla f_o\phi_{t-1}\odot\mathbf{r}_t$$

$$\text{Derivative w.r.t. } \mathbf{b}_o : \frac{\partial L}{\partial \mathbf{b}_o} = \frac{\partial L}{\partial \mathbf{O}_t}\frac{\partial \mathbf{O}_t}{\partial \phi_t}\frac{\partial \phi_t}{\partial \tilde{\mathbf{h}}_t}\frac{\partial \tilde{\mathbf{h}}_t}{\partial \mathbf{b}_o} = \nabla f_o$$

Derivatives w.r.t. $\mathbf{U}_r, \mathbf{W}_r, \mathbf{b}_r$

$$\text{Derivative w.r.t. } \mathbf{U}_r : \frac{\partial L}{\partial \mathbf{U}_r} = \frac{\partial L}{\partial \mathbf{O}_t}\frac{\partial \mathbf{O}_t}{\partial \phi_t}\frac{\partial \phi_t}{\partial \tilde{\mathbf{h}}_t}\frac{\partial \tilde{\mathbf{h}}_t}{\partial \mathbf{r}_t}\frac{\partial \mathbf{r}_t}{\partial \mathbf{U}_r} = \nabla f_r\mathbf{x}_t$$

$$\text{Derivative w.r.t. } \mathbf{W}_r : \frac{\partial L}{\partial \mathbf{W}_r} = \frac{\partial L}{\partial \mathbf{O}_t}\frac{\partial \mathbf{O}_t}{\partial \phi_t}\frac{\partial \phi_t}{\partial \tilde{\mathbf{h}}_t}\frac{\partial \tilde{\mathbf{h}}_t}{\partial \mathbf{r}_t}\frac{\partial \mathbf{r}_t}{\partial \mathbf{W}_r} = \nabla f_r\phi_{t-1}$$

$$\text{Derivative w.r.t. } \mathbf{b}_r : \frac{\partial L}{\partial \mathbf{b}_r} = \frac{\partial L}{\partial \mathbf{O}_t}\frac{\partial \mathbf{O}_t}{\partial \phi_t}\frac{\partial \phi_t}{\partial \tilde{\mathbf{h}}_t}\frac{\partial \tilde{\mathbf{h}}_t}{\partial \mathbf{r}_t}\frac{\partial \mathbf{r}_t}{\partial \mathbf{b}_r} = \nabla f_r$$

Derivatives w.r.t. $\mathbf{U}_z, \mathbf{W}_z, \mathbf{b}_z$

$$\text{Derivative w.r.t. } \mathbf{U}_z : \frac{\partial L}{\partial \mathbf{U}_z} = \frac{\partial L}{\partial \mathbf{O}_t}\frac{\partial \mathbf{O}_t}{\partial \phi_t}\frac{\partial \phi_t}{\partial \mathbf{z}_t}\frac{\partial \mathbf{z}_t}{\partial \mathbf{U}_z} = \nabla f_z \mathbf{x}_t$$

$$\text{Derivative w.r.t. } \mathbf{W}_z : \frac{\partial L}{\partial \mathbf{W}_z} = \frac{\partial L}{\partial \mathbf{O}_t}\frac{\partial \mathbf{O}_t}{\partial \phi_t}\frac{\partial \phi_t}{\partial \mathbf{z}_t}\frac{\partial \mathbf{z}_t}{\partial \mathbf{W}_z} = \nabla f_z \phi_{t-1}$$

$$\text{Derivative w.r.t. } \mathbf{W}_z : \frac{\partial L}{\partial \mathbf{W}_z} = \frac{\partial L}{\partial \mathbf{O}_t}\frac{\partial \mathbf{O}_t}{\partial \phi_t}\frac{\partial \phi_t}{\partial \mathbf{z}_t}\frac{\partial \mathbf{z}_t}{\partial \mathbf{W}_z} = \nabla f_z \phi_{t-1}$$

$$\text{Derivative w.r.t. } \mathbf{b}_z : \frac{\partial L}{\partial \mathbf{b}_z} = \frac{\partial L}{\partial \mathbf{O}_t}\frac{\partial \mathbf{O}_t}{\partial \phi_t}\frac{\partial \phi_t}{\partial \mathbf{z}_t}\frac{\partial \mathbf{z}_t}{\partial \mathbf{b}_z} = \nabla f_z$$

Derivatives w.r.t. $\phi_{t-1}, \mathbf{x}_t$

$$\begin{aligned}
&\text{Derivative w.r.t. } \phi_{t-1} : \frac{\partial L}{\partial \phi_{t-1}} = \frac{\partial L}{\partial \mathbf{O}_t}\frac{\partial \mathbf{O}_t}{\partial \phi_t}\frac{\partial \phi_t}{\partial \tilde{\mathbf{h}}_t}\frac{\partial \tilde{\mathbf{h}}_t}{\partial f_o}\frac{\partial f_o}{\partial \phi_{t-1}} \\
&+ \frac{\partial L}{\partial \mathbf{O}_t}\frac{\partial \mathbf{O}_t}{\partial \phi_t}\frac{\partial \phi_t}{\partial \mathbf{z}_t}\frac{\partial \mathbf{z}_t}{\partial f_z}\frac{\partial f_z}{\partial \phi_{t-1}} \\
&= \nabla f_o(\mathbf{W}_o \odot \mathbf{r}_t + \frac{\partial \mathbf{r}_t}{\partial f_r}\frac{\partial f_r}{\partial \phi_{t-1}}) + \nabla f_z \mathbf{W}_z \\
&= \nabla f_o \mathbf{W}_o \odot \mathbf{r}_t + \nabla f_r \mathbf{W}_r + \nabla f_z \mathbf{W}_z \\
&\text{Derivative w.r.t. } \mathbf{x}_t : \frac{\partial L}{\partial \mathbf{x}_t} = \frac{\partial L}{\partial \mathbf{O}_t}\frac{\partial \mathbf{O}_t}{\partial \phi_t}\frac{\partial \phi_t}{\partial \tilde{\mathbf{h}}_t}\frac{\partial \tilde{\mathbf{h}}_t}{\partial f_o}\frac{\partial f_o}{\partial \mathbf{x}_t} \\
&+ \frac{\partial L}{\partial \mathbf{O}_t}\frac{\partial \mathbf{O}_t}{\partial \phi_t}\frac{\partial \phi_t}{\partial \mathbf{z}_t}\frac{\partial \mathbf{z}_t}{\partial f_z}\frac{\partial f_z}{\partial \mathbf{x}_t} \\
&= \nabla f_o \mathbf{U}_o \odot \mathbf{r}_t + \nabla f_z \mathbf{U}_z + \nabla f_r \mathbf{U}_r
\end{aligned}$$

Bibliography

[1] B. Shuai, G. Wang, Z. Zuo, B. Wang, and L. Zhao, "Integrating parametric and non-parametric models for scene labeling," in *CVPR*, 2015, pp. 4249–4258. (document), 1, 2, 3.2.1, 3.2.2, 5.1, 6.8, 6.2.3

[2] H. Fan, X. Mei, D. Prokhorov, and H. Ling, "Multi-level contextual rnns with attention model for scene labeling," *arXiv preprint arXiv:1607.02537*, 2016. (document), 1, 2, 3.2.3, 5.1, 5.2

[3] W. Byeon, T. M. Breuel, F. Raue, and M. Liwicki, "Scene labeling with lstm recurrent neural networks," in *CVPR*, 2015, pp. 3547–3555. (document), 1, 2, 3.2.3, 5.1, 6.4, 6.6

[4] R. Gonzales and R.Woods, "Digital image processing," Addison-Wesley Publishing, 1st ed, Tech. Rep., 1993. 1, 3.1, 3.1.2

[5] D. G.Lowe, "Distinctive image features from scale-invariant keypoints," in *ICCV*, 2004, pp. 91–110. 1

[6] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893. 1

[7] H. Bay, A. Ess, T. Tuytelaars, and L. Gool, "Speeded-up robust features (surf)," *CVIU*, vol. 110, 2008. 1

[8] Y. LeCun, D. Touresky, G. Hinton, and T. Sejnowski, "A theoretical framework for

back-propagation," in *Proceedings of the 1988 connectionist models summer school*. CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988, pp. 21–28. 1, 2.2

[9] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. Springer, 1998, pp. 9–50. 1, 2.2

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105. 1, 2.5.2

[11] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, and et al., "Imagenet: large scale visual recognition challenge," 2014. 1, 5.1

[12] C. Samson, L. Blanc-Feraud, G. Aubert, and J. Zerubia, "A level set model for image classification," *International Journal Computer Vision*, vol. 40, no. 3, pp. 187–197, 2000. 1, 2.4.5

[13] T. Brox and J. Weickert, "Level set segmentation with multiple regions," *IEEE Transactions on Image Processing*, vol. 15, no. 10, p. 32133218, 2006. 1, 2.4.5

[14] E. Bae and X.-C. Tai, "Graph cut optimization for the piecewise constant level set method applied to multiphase image segmentation," 2009, pp. 1–13. 1, 2.4.5

[15] C. Li, C. Xu, C. Gui, and M. D. Fox, "Distance regularized level set evolution and its application to image segmentation," *IEEE Trans. Image Process. (TIP)*, vol. 19, no. 12, pp. 3243–3254, 2010. 1, 2.4.5, 6.1.3, 6.1, 6.1

[16] C. Li, R. Huang, Z. Ding, C. Gatenby, D. N. Metaxas, and J. C. Gore, "A level set method for image segmentation in the presence of intensity inhomogene ities with application to mri," *IEEE Transactions on Image Processing (TIP)*, vol. 20, no. 7, pp. 2007–2016, 2011. 1, 2.4.5, 6.1.3, 6.1, 6.1

[17] B. Lucas, M. Kazhdan, and R. Taylor, "Multi-object spring level sets (muscle),"

2012, pp. 495–503. 1, 2.4.5

[18] T. H. N. Le, K. Luu, , and M. Savvides, "Sparcles: Dynamic l1 sparse classifiers with level sets for robust beard/moustache detection and segmentation," *IEEE Trans. on Image Processing (TIP)*, vol. 22, no. 8, pp. 3097–3107, 2013. 1

[19] Q. Huang, X. Bai, Y. Li, L. Jin, and X. Li, "Optimized graph-based segmentation for ultrasound images," *Neurocomputing*, vol. 129, no. Complete, pp. 216–224, 2014. 1, 2.4.5, 3.1.6

[20] J. Li, M. Luong, and D. Jurafsky, "A hierarchical neural autoencoder for paragraphs and documents," *CoRR*, vol. abs/1506.01057, 2015. 1, 2.3.3

[21] S. Mukherjee and S. Acton, "Region based segmentation in presence of intensity inhomogeneity using legendre polynomials," *IEEE Signal Processing Letters*, vol. 22, no. 3, pp. 298–302, 2015. 1, 2.4.5, 3.1.6, 6.1.3, 6.1, 6.1

[22] T. H. N. Le, , and Savvides, "A novel shape constrained feature-based active contour (sc-fac) model for lips/mouth segmentationin the wild," *Pattern Recognition*, vol. 54, pp. 23–33, 2016. 1, 3.1.6

[23] J. Shen, Y. Du, and X. Li, "Interactive segmentation using constrained laplacian optimization," *IEEE Trans. Circuits Syst. Video Techn.*, vol. 24, no. 7, pp. 1088–1100, 2014. 1, 2.4.5, 3.1.6

[24] 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*, vol. 88, no. 2, pp. 303–338, Jun. 2010. 1, 3.3.1, 3.3, 4.5, 6.1, 6.1.1

[25] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. 1, 3.3.1, 3.3, 4.5, 5.1, 6.1, 6.1.1

[26] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Simultaneous detection and

segmentation," in *European Conference on Computer Vision*. Springer, 2014, pp. 297–312. 1, 1, 3.2.2, 6.1.4, 6.2

[27] B. Hariharan, P. Arbelaez, R. Girshick, and J. Malik, "Hyper- columns for object segmentation and fine-grained localization," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 1, 6.1.4, 6.2

[28] J. Dai, K. He, and J. Sun, "Convolutional feature masking for joint object and stuff segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 1, 6.1.4, 6.2

[29] ——, "Instance-fully convolutional instance-aware semantic segmentation via multi-task network cascades." 1, 1, 1.2, 1.3, 3.2.2, 6.1.4, 6.2, 6.2, 6.3, 6.1.5, 6.4, 6.5, 6.6, 6.7, A

[30] R. Girshick, J. Donahue, and J. M. T. Darrell, "Region-based convolutional networks for accurate object detection and semantic segmentation," *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 38, pp. 142–158, 2015. 1, 1, 2.6.1, 3.2.2, 4.3.1

[31] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 1, 1, 3.2.2, 3.2.2, 3.1, 5.1, 6.2.3, 6.7, 6.8, 6.2.3

[32] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. Torr, "Conditional random fields as recurrent neural networks," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2015. 1, 3.2.1, 3.2.3, 3.2

[33] T. N. Le, C. Zhu, Y. Zheng, K. Luu, and M. Savvides, "Robust hand detection in vehicles," in *International Conference on Pattern Recognition (ICPR)*, 2016, pp. 573–578. 1

[34] T. N. Le, Y. Zheng, C. Zhu, K. Luu, and M. Savvides, "Multiple scale faster-rcnn approach to drivers cell-phone usage and hands on steering wheel detection," in *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, June 2016, pp. 46–53. 1

[35] K. Luu, C. C. Zhu, C. Bhagavatula, T. N. Le, and M. Savvide, "A deep learning approach to joint face detection and segmentation," *Kawulok M., Celebi M., Smolka B. (eds) Advances in Face Detection and Facial Image Analysis. Springer, Cham*, pp. 1–12, 2016. 1

[36] Y. Zheng, C. Zhu, K. Luu, C. Bhagavatula, T. N. Le, and M. Savvides, "Towards a deep learning framework for unconstrained face detection," in *IEEE 8th Intl. Conference on Biometrics: Theory, Applications and Systems (BTAS)*, 2016, pp. 1–8. 1

[37] T. N. Le, K. Luu, C. Zhu, and M. Savvides, "Semi self-training beard/moustache detection and segmentation simultaneously," *Image and Vision Computing*, vol. 58, pp. 214–223, 2017. 1

[38] T. N. Le, C. Zhu, Y. Zheng, K. Luu, and M. Savvides, "Deepsafedrive: A grammar-aware driver parsing approach to driver behavioral situational awareness (db-saw)," *Pattern Recognition*, vol. 66, pp. 229–238, 2017. 1

[39] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017. 1, 3.2.2, 3.1

[40] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015. 1, 2.2.1, 3.2.2, 3.2.2, 3.1

[41] A. Kendall, V. Badrinarayanan, and R. Cipolla, "Bayesian segnet: Model uncertainty

in deep convolutional encoder-decoder architectures for scene understanding," *arXiv preprint arXiv:1511.02680*, 2015. 1, 3.1

[42] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2018. 1, 3.1

[43] P. O. Pinheiro, R. Collobert, and P. Dollár, "Learning to segment object candidates," in *NIPS*, 2015. 1, 3.1

[44] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2881–2890. 1, 3.2.2, 3.1

[45] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei, "Fully convolutional instance-aware semantic segmentation," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2359–2367. 1, 3.1

[46] X. He, R. Zemel, and M. Carreira-Perpinan, "Multiscale conditional random fields for image labeling," in *CVPR 2004*, vol. 2, 2004, pp. II–695–II–702. 1, 3.2.1

[47] Z. Tu and X. Bai, "Auto-context and its application to high-level vision tasks and 3d brain image segmentation," *TPAMI*, vol. 32, no. 10, pp. 1744–1757, 2010. 1

[48] D. Munoz, J. A. Bagnell, and M. Hebert, "Stacked hierarchical labeling," in *ECCV*, 2010, pp. 57–70. 1, 3.2.1

[49] J. Yang, B. Price, S. Cohen, and M.-H. Yang, "Context driven scene parsing with attention to rare classes," in *CVPR*, 2014, pp. 3294–3301. 1, 3.2.1, 5.1, 6.4

[50] S. Bu, P. Han, Z. Liu, and JunweiHan, "Scene parsing using inference embedded deep networks," *Pattern Recognition*, vol. 56, pp. 188–198, 2016. 1

[51] Q. Zhou, B. Zheng, W. Zhu, and L. J. Latecki, "Multi-scale context for scene labeling

via flexible segmentation graph," *Pattern Recognition*, vol. 59, pp. 312–324, 2016. 1

[52] F. Visin, K. Kastner, A. C. Courville, Y. Bengio, M. Matteucci, and K. Cho, "Reseg: A recurrent neural network for object segmentation," *CoRR*, vol. abs/1511.07053, 2015. 1, 3.2.3, 3.2, 5.1

[53] P. H. Pinheiro and R. Collobert, "Recurrent convolutional neural networks for scene labeling," in *ICML*, 2014, pp. 82–90. 1, 3.2.3, 5.1, 6.4, 6.6

[54] A. Sharma, O. Tuzel, and M. Liu, "Recursive context propagation network for semantic scene labeling," in *NIPS*, 2014, pp. 2447–2455. 1, 6.4

[55] N. Souly and M. Shah, "Scene labeling using sparse precision matrix," in *CVPR*, June 2016. 1, 6.4, 6.6

[56] B. Shuai, Z. Zuo, G. Wang, and B. Wang, "Dag-recurrent neural networks for scene labeling," in *CVPR*, 2016. 1, 1.4, 3.2, 5.2, 6.2.1, 6.10, 6.4, 6.2.3, 6.7, 6.8, 6.2.3

[57] M. Najafi, S. Taghavi Namin, M. Salzmann, and L. Petersson, "Sample and filter: Nonparametric scene parsing via efficient filtering," in *CVPR*, June 2016. 1, 6.4

[58] M. Liang, X. Hu, and B. Zhang, "Convolutional neural networks with intra-layer recurrent connections for scene labeling," in *NIPS*, 2015, pp. 937–945. 1, 6.4, 6.2.3, 6.6

[59] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *TPAMI*, vol. 35, no. 8, pp. 1915–1929, 2013. 1, 3.2.2, 5.1, 6.4, 6.6

[60] Z. Shuai, J. Sadeep, R.-P. Bernardino, V. Vibhav, S. Zhizhong, D. Dalong, H. Chang, and T. P. HS, "Conditional random fields as recurrent neural networks," in *CVPR*, 2015, pp. 1529–1537. 1, 5.1

[61] Y. Duan, F. Liu, L. Jiao, P. Zhao, and LuZhang, "Sar image segmentation based on convolutional-wavelet neural network and markov random field," *Pattern Recognition*, vol. 64, pp. 255–267, 2017. 1

[62] T. F. Chan and L. A. Vese, "Active contours without edges," *TIP*, vol. 10, no. 2, pp. 266–277, Feb. 2001. 1, 1.2, 2.4.4, 3.1.6, 4.2.2, 4.2.2, 6.1.1, 6.1.3, 6.1, 6.1, A

[63] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, 1998, pp. 2278–2324. 2.2

[64] Y. L. Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson, "Advances in neural information processing systems 2," 1990, ch. Handwritten Digit Recognition with a Back-propagation Network, pp. 396–404. 2.2, 2.5.1

[65] J. Ngiam, Z. Chen, D. Chia, P. W. Koh, Q. V. Le, and A. Y. Ng, "Tiled convolutional neural networks," in *Advances in neural information processing systems*, 2010, pp. 1279–1287. 2.2.1

[66] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *ECCV*, 2014, pp. 818–833. 2.2.1, 2.5.3, 2.6.3

[67] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1, 2013, p. 3. 2.2.2

[68] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034. 2.2.2, 3.2.2

[69] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015. 2.2.2

[70] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015. 2.2.2

[71] J. Bruna, A. Szlam, and Y. LeCun, "Signal recovery from pooling representations," *arXiv preprint arXiv:1311.4025*, 2013. 2.2.3

[72] D. Yu, H. Wang, P. Chen, and Z. Wei, "Mixed pooling for convolutional neural networks," in *International Conference on Rough Sets and Knowledge Technology*. Springer, 2014, pp. 364–375. 2.2.3

[73] M. D. Zeiler and R. Fergus, "Stochastic pooling for regularization of deep convolutional neural networks," *arXiv preprint arXiv:1301.3557*, 2013. 2.2.3

[74] O. Rippel, J. Snoek, and R. P. Adams, "Spectral representations for convolutional neural networks," in *Advances in neural information processing systems*, 2015, pp. 2449–2457. 2.2.3

[75] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015. 2.2.3, 2.2.8

[76] W. Liu, Y. Wen, Z. Yu, and M. Yang, "Large-margin softmax loss for convolutional neural networks." in *ICML*, 2016, pp. 507–516. 2.2.4

[77] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 539–546. 2.2.4

[78] G. Koch, "Siamese neural networks for one-shot image recognition," 2015. 2.2.4

[79] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823. 2.2.4

[80] H. Liu, Y. Tian, Y. Yang, L. Pang, and T. Huang, "Deep relative distance learning: Tell the difference between similar vehicles," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2167–2175. 2.2.4

[81] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov,

"Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012. 2.2.5

[82] S. Wang and C. Manning, "Fast dropout training," in *international conference on machine learning*, 2013, pp. 118–126. 2.2.5

[83] J. Ba and B. Frey, "Adaptive dropout for training deep neural networks," in *Advances in Neural Information Processing Systems*, 2013, pp. 3084–3092. 2.2.5

[84] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, "Efficient object localization using convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 648–656. 2.2.5

[85] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *International Conference on Machine Learning*, 2013, pp. 1058–1066. 2.2.5

[86] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015. 2.2.6, 2.2.8

[87] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2650–2658. 2.2.6

[88] J. Salamon and J. P. Bello, "Deep convolutional neural networks and data augmentation for environmental sound classification," *IEEE Signal Processing Letters*, vol. 24, no. 3, pp. 279–283, 2017. 2.2.6

[89] M. Paulin, J. Revaud, Z. Harchaoui, F. Perronnin, and C. Schmid, "Transformation pursuit for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on.* IEEE, 2014, pp. 3646–3653. 2.2.6

[90] Z. Xu, S. Huang, Y. Zhang, and D. Tao, "Augmenting strong supervision using web data for fine-grained categorization," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2524–2532. 2.2.6

[91] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256. 2.2.6

[92] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," *arXiv preprint arXiv:1312.6120*, 2013. 2.2.6

[93] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999. 2.2.6

[94] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Advances in neural information processing systems*, 2010, pp. 2595–2603. 2.2.6

[95] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015. 2.2.6

[96] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778. 2.2.6, 2.5.6

[97] D. T. Nguyen, W. Li, and P. O. Ogunbona, "Human detection from images and videos: A survey," *Pattern Recognition*, vol. 51, pp. 148–175, 2016. 2.2.8

[98] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *International journal of computer vision*, vol. 104, no. 2, pp. 154–171, 2013. 2.2.8

[99] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for

accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587. 2.2.8

[100] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. 2.2.8, 2.6.3, 2.6.4, 4.3.1, 4.5, 4.3.1, 4.3.1, 4.5

[101] M. Egmont-Petersen, D. de Ridder, and H. Handels, "Image processing with neural networksa review," *Pattern recognition*, vol. 35, no. 10, pp. 2279–2301, 2002. 2.2.8

[102] A.-M. Tousch, S. Herbin, and J.-Y. Audibert, "Semantic hierarchies for image annotation: A survey," *Pattern Recognition*, vol. 45, no. 1, pp. 333–345, 2012. 2.2.8

[103] G.-S. Xie, X.-Y. Zhang, W. Yang, M. Xu, S. Yan, and C.-L. Liu, "Lg-cnn: From local parts to global discrimination for fine-grained recognition," *Pattern Recognition*, vol. 71, pp. 118–131, 2017. 2.2.8

[104] J. Fan, W. Xu, Y. Wu, and Y. Gong, "Human tracking using convolutional neural networks," *IEEE Transactions on Neural Networks*, vol. 21, no. 10, pp. 1610–1623, 2010. 2.2.8

[105] S. Hong, T. You, S. Kwak, and B. Han, "Online tracking by learning discriminative saliency map with convolutional neural network," in *International Conference on Machine Learning*, 2015, pp. 597–606. 2.2.8

[106] H. Li, Y. Li, and F. Porikli, "Deeptrack: Learning discriminative feature representations online for robust visual tracking," *IEEE Transactions on Image Processing*, vol. 25, no. 4, pp. 1834–1848, 2016. 2.2.8

[107] L. Wang, H. Lu, X. Ruan, and M.-H. Yang, "Deep networks for saliency detection via local estimation and global search," in *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE, 2015, pp. 3183–3192. 2.2.8

[108] G. Li and Y. Yu, "Visual saliency based on multiscale deep features," *arXiv preprint*

arXiv:1503.08663, 2015. 2.2.8

[109] M. Patacchiola and A. Cangelosi, "Head pose estimation in the wild using convolutional neural networks and adaptive gradient methods," *Pattern Recognition*, vol. 71, pp. 132–143, 2017. 2.2.8

[110] A. Toshev and C. Szegedy, "Deeppose: Human pose estimation via deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1653–1660. 2.2.8

[111] A. Jain, J. Tompson, M. Andriluka, G. W. Taylor, and C. Bregler, "Learning human pose estimation features with convolutional networks," *arXiv preprint arXiv:1312.7302*, 2013. 2.2.8

[112] J. J. Tompson, A. Jain, Y. LeCun, and C. Bregler, "Joint training of a convolutional network and a graphical model for human pose estimation," in *Advances in neural information processing systems*, 2014, pp. 1799–1807. 2.2.8

[113] A. Jain, J. Tompson, Y. LeCun, and C. Bregler, "Modeep: A deep learning framework using motion features for human pose estimation," in *Asian conference on computer vision*. Springer, 2014, pp. 302–315. 2.2.8

[114] G. Gkioxari, R. Girshick, and J. Malik, "Contextual action recognition with r* cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1080–1088. 2.2.8

[115] J. Zhang, W. Li, P. O. Ogunbona, P. Wang, and C. Tang, "Rgb-d-based action recognition datasets: A survey," *Pattern Recognition*, vol. 60, pp. 86–105, 2016. 2.2.8

[116] M. Delakis and C. Garcia, "text detection with convolutional neural networks." in *VISAPP (2)*, 2008, pp. 290–294. 2.2.8

[117] H. Xu and F. Su, "Robust seed localization and growing with deep convolutional

features for scene text detection," in *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*. ACM, 2015, pp. 387–394. 2.2.8

[118] P. He, W. Huang, Y. Qiao, C. C. Loy, and X. Tang, "Reading scene text in deep convolutional sequences." in *AAAI*, vol. 16, 2016, pp. 3501–3508. 2.2.8

[119] T. Wang, D. J. Wu, A. Coates, and A. Y. Ng, "End-to-end text recognition with convolutional neural networks," in *Pattern Recognition (ICPR), 2012 21st International Conference on*. IEEE, 2012, pp. 3304–3308. 2.2.8

[120] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Deep features for text spotting," in *European conference on computer vision*. Springer, 2014, pp. 512–528. 2.2.8

[121] D. Yu, W. Xiong, J. Droppo, A. Stolcke, G. Ye, J. Li, and G. Zweig, "Deep convolutional neural networks with layer-wise context expansion and attention." in *Interspeech*, 2016, pp. 17–21. 2.2.8

[122] L.-H. Chen, T. Raitio, C. Valentini-Botinhao, J. Yamagishi, and Z.-H. Ling, "Dnn-based stochastic postfilter for hmm-based speech synthesis." in *INTERSPEECH*, 2014, pp. 1954–1958. 2.2.8

[123] M. I. Jordan, "Artificial neural networks," J. Diederich, Ed., 1990, ch. Attractor Dynamics and Parallelism in a Connectionist Sequential Machine, pp. 112–127. 2.3.1

[124] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Neurocomputing: Foundations of research," J. A. Anderson and E. Rosenfeld, Eds. Cambridge, MA, USA: MIT Press, 1988, ch. Learning Representations by Back-propagating Errors, pp. 696–699. 2.3.1

[125] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994. 2.3.2, 2.3.6

[126] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural

nets and problem solutions," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 6, no. 2, pp. 107–116, 1998. 2.3.2

[127] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997. 2.3.2

[128] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. 2.3.3, 2.3.3, 4.2.2

[129] T. Mikolov, S. Kombrink, L. Burget, J. Cernocký, and S. Khudanpur, "Extensions of recurrent neural network language model," in *ICASSP*, 2011, pp. 5528–5531. 2.3.3

[130] T. Mikolov, "Statistical language models based on neural networks," PhD thesis, Brno University of Technology, 2012. 2.3.3

[131] K. Zhang, Q. Liu, H. Song, and X. Li, "A variational approach to simultaneous image segmentation and bias correction." *IEEE Trans. Cybernetics*, vol. 45, no. 8, pp. 1426–1437, 2015. 2.3.3, 2.4.5, 3.1.6

[132] A. Graves, A. Mohamed, and G. E. Hinton, "Speech recognition with deep recurrent neural networks," *CoRR*, vol. abs/1303.5778, 2013. 2.3.3

[133] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," *CoRR*, vol. abs/1506.07503, 2015. 2.3.3

[134] N. Kalchbrenner and P. Blunsom, "Recurrent continuous translation models." Association for Computational Linguistics, October 2013. 2.3.3

[135] T. Luong, I. Sutskever, Q. V. Le, O. Vinyals, and W. Zaremba, "Addressing the rare word problem in neural machine translation," *CoRR*, vol. abs/1410.8206, 2014. 2.3.3

[136] O. Vinyals and Q. V. Le, "A neural conversational model," *CoRR*, vol. abs/1506.05869, 2015. 2.3.3

[137] R. Lowe, N. Pow, I. Serban, and J. Pineau, "The ubuntu dialogue corpus: A

large dataset for research in unstructured multi-turn dialogue systems," *CoRR*, vol. abs/1506.08909, 2015. 2.3.3

[138] F. Hill, A. Bordes, S. Chopra, and J. Weston, "The goldilocks principle: Reading children's books with explicit memory representations," *CoRR*, vol. abs/1511.02301, 2015. 2.3.3

[139] M. Liang and X. Hu, "Recurrent convolutional neural network for object recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 2.3.3

[140] W. Byeon, T. M. Breuel, F. Raue, and M. Liwicki, "Scene labeling with lstm recurrent neural networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 2.3.3

[141] S. Bell, C. L. Zitnick, K. Bala, and R. B. Girshick, "Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks," *CoRR*, vol. abs/1512.04143, 2015. 2.3.3

[142] J. Mao, W. Xu, Y. Yang, J. Wang, and A. L. Yuille, "Deep captioning with multimodal recurrent neural networks (m-rnn)," *CoRR*, vol. abs/1412.6632, 2014. 2.3.3

[143] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," *CoRR*, vol. abs/1411.4389, 2014. 2.3.3

[144] A. Karpathy and F. Li, "Deep visual-semantic alignments for generating image descriptions," *CoRR*, vol. abs/1412.2306, 2014. 2.3.3

[145] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *IJCV*, vol. 1, no. 4, pp. 321–331, 1988. 2.4.1, 3.1.6

[146] V. Caselles, F. Catté, T. Coll, and F. Dibos, "A geometric model for active contours in image processing," *Numerische Mathematik*, vol. 66, no. 1, pp. 1–31, Dec. 1993.

2.4.3

[147] O. M.-G. N. Paragios and V. Ramesh, "Gradient vector flow fast geometric active contours," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 3, pp. 402–407, 2004. 2.4.3

[148] D. Mumford and J. Shah, "Optimal Approximation by Piecewise Smooth Functions and Associated Variational Problems," *Communications on Pure and Applied Mathematics*, vol. 42, no. 5, pp. 577–685, 1989. 2.4.4, 2.4.5, 3.1.6

[149] C. Li, C. Kao, J. Gore, and Z. Ding, "Implicit active contours driven by local binary fitting energy," in *CVPR*, 2007, pp. 1–7. 2.4.5, 3.1.6

[150] H. Wu, V. V. Appia, and A. J. Yezzi, "Numerical conditioning problems and solutions for nonparametric i.i.d. statistical active contours." *TPAMI*, vol. 35, no. 6, pp. 1298–1311, 2013. 2.4.5

[151] Y. Shi and W. C. Karl, "Real-time tracking using level sets," vol. 2, June 2005, pp. 34–41 vol. 2. 2.4.5

[152] A. Dubrovina, G. Rosman, and R. Kimmel, "Multi-region active contours with a single level set function," *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 37, pp. 1585–1601, 2015. 2.4.5, 3.1.6

[153] T. H. N. Le and M. Savvides, "A novel shape constrained feature-based active contour model for lips/mouth segmentation in the wild," *Pattern Recognition*, vol. 54, pp. 23–33, 2016. 2.4.5

[154] H. Zhou, X. Li, G. Schaefer, M. E. Celebi, and P. C. Miller, *Computer Vision and Image Understanding*, vol. 117, no. 9, pp. 1004–1016, 2013. 2.4.5

[155] T. F. Chan, S. Esedoglu, and M. Nikolova, "Algorithms for finding global minimizers of image segmentation and denoising models," Siam Journal on Applied Mathematics, Tech. Rep., 2006. 2.4.5

[156] J. Weickert, B. M. T. H. Romeny, and M. A. Viergever, "Efficient and reliable schemes for nonlinear diffusion filtering," *TIP*, vol. 7, pp. 398–410, 1998. 2.4.5

[157] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014. 2.5.4, 2.6.3, 4.3.1

[158] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich *et al.*, "Going deeper with convolutions." Cvpr, 2015. 2.5.5

[159] R. Girshick, "Fast r-cnn," in *ICCV*, 2015, pp. 1440–1448. 2.6.2, 4.3.1

[160] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. 2.6.4

[161] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788. 2.6.5

[162] A. Brink, "Minimum spatial entropy threshold selection," *IEE Proceedings-Vision, Image and Signal Processing*, vol. 142, no. 3, pp. 128–132, 1995. 3.1.1

[163] O. J. Tobias and R. Seara, "Image segmentation by histogram thresholding using fuzzy sets," *IEEE transactions on Image Processing*, vol. 11, no. 12, pp. 1457–1465, 2002. 3.1.1

[164] T. H. N. Le, T. D. Bui, and C. Y. Suen, "Ternary entropy-based binarization of degraded document images using morphological operators," in *Document Analysis and Recognition (ICDAR), 2011 International Conference on.* IEEE, 2011, pp. 114–118. 3.1.1

[165] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE transactions on systems, man, and cybernetics*, vol. 9, no. 1, pp. 62–66, 1979. 3.1.1

[166] A. Z. Arifin and A. Asano, "Image segmentation by histogram thresholding using hierarchical cluster analysis," *Pattern recognition letters*, vol. 27, no. 13, pp. 1515–

1521, 2006. 3.1.1

[167] N. Vandenbroucke, L. Macaire, and J.-G. Postaire, "Color image segmentation by pixel classification in an adapted hybrid color space. application to soccer image analysis," *Computer Vision and Image Understanding*, vol. 90, no. 2, pp. 190–216, 2003. 3.1.1

[168] K. S. Tan and N. A. M. Isa, "Color image segmentation using histogram thresholding–fuzzy c-means hybrid approach," *Pattern Recognition*, vol. 44, no. 1, pp. 1–15, 2011. 3.1.1

[169] B. Sumengen and B. Manjunath, "Multi-scale edge detection and image segmentation," in *Signal Processing Conference, 2005 13th European*. IEEE, 2005, pp. 1–4. 3.1.2

[170] J. Fan, D. K. Yau, A. K. Elmagarmid, and W. G. Aref, "Automatic image segmentation by integrating color-edge extraction and seeded region growing," *IEEE transactions on image processing*, vol. 10, no. 10, pp. 1454–1466, 2001. 3.1.2

[171] C. Brice and C. Fennema, "Computer method in image analysis, chapter. scene analysis using regions," Los Angeles: IEEE Computer Society, Tech. Rep., 1977. 3.1.3

[172] S. Hojjatoleslami and J. Kittler, "Region growing: a new approach," *IEEE Transactions on Image processing*, vol. 7, no. 7, pp. 1079–1084, 1998. 3.1.3

[173] J. C. Pichel, D. E. Singh, and F. F. Rivera, "Image segmentation based on merging of sub-optimal segmentations," *Pattern recognition letters*, vol. 27, no. 10, pp. 1105–1116, 2006. 3.1.3

[174] R. Adams and L. Bischof, "Seeded region growing," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 16, no. 6, pp. 641–647, 1994. 3.1.3

[175] O. Gómez, J. A. González, and E. F. Morales, "Image segmentation using automatic

seeded region growing and instance-based learning," in *Iberoamerican Congress on Pattern Recognition*. Springer, 2007, pp. 192–201. 3.1.3

[176] F. Meyer, "Topographic distance and watershed lines," *Signal processing*, vol. 38, no. 1, pp. 113–125, 1994. 3.1.4

[177] J. M. Gauch, "Image segmentation and analysis via multiscale gradient watershed hierarchies," *IEEE transactions on image processing*, vol. 8, no. 1, pp. 69–79, 1999. 3.1.4

[178] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 8, pp. 888–905, 2000. 3.1.5

[179] X. Liu and D. Wang, "Image and texture segmentation using local spectral histograms," *IEEE Transactions on Image Processing*, vol. 15, no. 10, pp. 3066–3077, 2006. 3.1.5

[180] W. Yang, L. Guo, T. Zhao, and G. Xiao, "Improving watersheds image segmentation method with graph theory," in *Industrial Electronics and Applications, 2007. ICIEA 2007. 2nd IEEE Conference on*. IEEE, 2007, pp. 2550–2553. 3.1.5

[181] W. Tao, H. Jin, and Y. Zhang, "Color image segmentation based on mean shift and normalized cuts," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 37, no. 5, pp. 1382–1389, 2007. 3.1.5

[182] S. Wang and J. M. Siskind, "Image segmentation with ratio cut," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 6, pp. 675–690, 2003. 3.1.5

[183] V. Caselles, R. Kimmel, and G. Sapiro, "Geodesic active contours," *International Journal of Computer Vision (International Journal of Computer Vision (IJCV)*, vol. 22, no. 1, pp. 61–79, Feb. 1997. 3.1.6

[184] C. Li, C. Xu, C. Gui, and M. Fox, "Level set evolution without re-initialization: a new variational formulation," in *Proceedings of the IEEE Conference on Computer*

Vision and Pattern Recognition (CVPR), 2005, pp. 430–436. 3.1.6

[185] N. Paragios and R. Deriche, "Geodesic active regions and level set methods for supervised texture segmentation," *International Journal of Computer Vision (International Journal of Computer Vision (IJCV)*, vol. 46, pp. 223–247, 2002. 3.1.6

[186] J. Lie, M. Lysaker, and X. Tai, "A binary level set model and some application to mumfordshah image segmentation," *IEEE Trans. Image Process. (TIP)*, pp. 1171–1181, 2010. 3.1.6

[187] L. A. Vese and T. F. Chan, "A multiphase level set framework for image segmentation using the mumford and shah model," *IJCV*, vol. 50, no. 3, Dec. 2002. 3.1.6

[188] C. Li, C. yen Kao, J. C. Gore, and Z. Ding, "Minimization of region-scalable fitting energy for image segmentation," *TIP*, 2008. 3.1.6, 6.1.1

[189] K. Zhang, H. Song, and L. Zhang, "Active contours driven by local image fitting energy," *Pattern Recognition (PR)*, vol. 43, no. 4, pp. 1199–1206, 2010. 3.1.6

[190] L. Wang and C. Pan, "Robust level set image segmentation via a local correntropy-based k-means clustering," *Pattern Recognition (PR)*, vol. 47, no. 5, pp. 1917–1925, 2014. 3.1.6

[191] Y. Han, W. Wang, and X. Feng, "A new fast multiphase image segmentation algorithm based on non-convex regularizer," *Pattern Recognition (PR)*, vol. 45, no. 1, pp. 363–372, 2012. 3.1.6

[192] S. Liu and Y. Peng, "A local region-based chan-vese model for image segmentation," *Pattern Recognition (PR)*, vol. 45, no. 7, pp. 2769–2779, 2012. 3.1.6

[193] Y. Wang, S. Xiang, C. Pan, L. Wang, and G. Meng, "Level set evolution with locally linear classification for image segmentation," *Pattern Recognition (PR)*, vol. 46, no. 6, pp. 1734–1746, 2013. 3.1.6

[194] S. Zhou, J. Wang, S. Zhang, Y. Liang, and Y. Gong, "Active contour model based

on local and global intensity information for medical image segmentation," *Neurocomputing*, vol. 186, pp. 107–118, 2016. 3.1.6

[195] T. A. Ngo, Z. Lu, and G. Carneiro, "Combining deep learning and level set for the automated segmentation of the left ventricle of the heart from cardiac cine magnetic resonance," *Pattern Recognition (PR)*, vol. 35, pp. 159–171, 2017. 3.1.6

[196] C. Liu, J. Yuen, and A. Torralba, *Nonparametric Scene Parsing via Label Transfer*, 2011, vol. 33, no. 12, pp. 2368–2382. 3.2.1

[197] J. Tighe and S. Lazebnik, "Superparsing: Scalable nonparametriv image parsing with superpixels," in *ECCV*, 2010, pp. 352–365. 3.2.1

[198] Z. Tu, "Auto-context and its application to high-level vision tasks," in *CVPR*, 2008, pp. 1–8. 3.2.1

[199] J. Shotton, J. Winn, C. Rother, and A. Criminisi, "Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation," in *ECCV*, 2006, pp. 1–15. 3.2.1, 5.1

[200] Y. Zhang and T. Chen, "Efficient inference for fully-connected crfs with stationarity," in *CVPR*, 2012, pp. 582–589. 3.2.1

[201] A. Roy and S. Todorovic, "Scene labeling using beam search under mutex constraints," in *CVPR*, 2014, pp. 1178–1185. 3.2.1

[202] B. Hariharan, P. Arbelaez, R. Girshick, and J. Malik, "Simultaneous detection and segmentation," in *ECCV*, 2014, pp. 297–312. 3.2.2, 3.1

[203] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," in *ICLR*, 2015, p. abc. 3.2.2

[204] J. Dai, K. He, and J. Sun, "Boxsup: Exploiting bounding boxes to supervise convolutional networks for semantic segmentation," in *ICCV*, 2015. 3.2.2

[205] G. Papandreou, L. C. Chen, K. Murphy, and A. L. Yuille, "Weakly- and semi-supervised learning of a dcnn for semantic image segmentation," in *ICCV*, 2015. 3.2.2

[206] M. Mostajabi, P. Yadollahpour, and G. Shakhnarovich, "Feedforward semantic segmentation with zoom-out features," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 3376–3385. 3.2.2

[207] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015, pp. 1520–1528. 3.2.2

[208] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2018. 3.2.2

[209] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "Enet: A deep neural network architecture for real-time semantic segmentation," *arXiv preprint arXiv:1606.02147*, 2016. 3.2.2, 3.1

[210] G. Lin, C. Shen, A. van den Hengel, and I. Reid, "Efficient piecewise training of deep structured models for semantic segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 3194–3203. 3.2.2

[211] P. A. ez, J. Pont-Tuset, J. T. Barron, F. Marques, and J. Malik, "Multiscale combinatorial grouping," in *CVPR*, 2014, pp. 328–335. 3.2.2

[212] P. O. Pinheiro, T.-Y. Lin, R. Collobert, and P. Dollar, "Learning to refine object segments," in *ECCV*, 2016, pp. 75–91. 3.2.2

[213] P. O. Pinheiro, T.-Y. Lin, R. Collobert, and P. Dollár, "Learning to refine object segments," in *ECCV*, 2016. 3.2.2

[214] S. Zagoruyko, A. Lerer, T. Lin, P. H. O. Pinheiro, S. Gross, S. Chintala, and P. Dollár, "A multipath network for object detection," *CoRR*, vol. abs/1604.02135, 2016. 3.2.2

[215] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *ECCV*, 2016. 3.2.2, 5.1

[216] Z. Hayder, X. He, and M. Salzmann, "Boundary-aware instance segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 3.2.2

[217] S. Jégou, M. Drozdzal, D. Vazquez, A. Romero, and Y. Bengio, "The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*. IEEE, 2017, pp. 1175–1183. 3.2.2

[218] T. Robinson, "An application of recurrent nets to phone probability estimation," *IEEE Transactions on Neural Networks*, vol. 5, pp. 298–305, 1994. 3.2.3

[219] N. Stoianov, J. Nerbonne, and H. Bouma, "Modelling the phonotactic structure of natural language words with simple recurrent networks," in *Computational Linguistics in The Netherlands*, 1998, pp. 77–95. 3.2.3

[220] A. Graves and J. Schmidhuber, "Offline handwriting recognition with multidimensional recurrent neural networks," in *Advances in Neural Information Processing Systems 21*, 2009, pp. 545–552. 3.2.3

[221] Z. Zuo, B. Shuai, G. Wang, X. Liu, X. Wang, B. Wang, and Y. Chen, "Convolutional recurrent neural networks: Learning spatial dependencies for image representation," in *CVPRW*, 2015, pp. 18–26. 3.2.3

[222] F. Visin, K. Kastner, K. Cho, M. Matteucci, A. C. Courville, and Y. Bengio, "Renet: A recurrent neural network based alternative to convolutional networks," *CoRR*, vol. abs/1505.00393, 2015. 3.2.3

[223] V. Mnih, N. Heess, A. Graves *et al.*, "Recurrent models of visual attention," in *NIPS*, 2014, pp. 2204–2212. 3.2.3, 5.1

[224] S. Bell, C. L. Zitnick, K. Bala, and R. Girshick, "Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks," in *CVPR*, 2016. 3.2.3

[225] F. Visin, A. Romero, K. Cho, M. Matteucci, M. Ciccone, K. Kastner, Y. Bengio, and A. Courville, "Reseg: A recurrent neural network-based model for semantic segmentation," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2016 IEEE Conference on*. IEEE, 2016, pp. 426–433. 3.2.3

[226] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241. 3.1

[227] P. H. O. Pinheiro and R. Collobert, "Recurrent convolutional neural networks for scene labeling," in *ICML*, 2014. 3.2

[228] S. Bell, P. Upchurch, N. Snavely, and K. Bala, "Material recognition in the wild with the materials in context database (supplemental material)," in *Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015. 3.3.1, 3.3

[229] R. Mottaghi, X. Chen, X. Liu, N.-G. Cho, S.-W. Lee, S. Fidler, R. Urtasun, and A. Yuille, "The role of context for object detection and semantic segmentation in the wild," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 891–898. 3.3.1, 3.3

[230] X. Chen, R. Mottaghi, X. Liu, S. Fidler, R. Urtasun, and A. Yuille, "Detect what you can: Detecting and representing objects using holistic models and body parts," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 3.3.1

[231] M. Cordts, M. Omran, S. Ramos, T. Scharwächter, M. Enzweiler, R. Benenson,

U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset," in *CVPR Workshop on the Future of Datasets in Vision*, vol. 1, no. 2, 2015, p. 3. 3.3.2, 3.3

[232] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognition Letters*, vol. 30, no. 2, pp. 88–97, 2009. 3.3.2, 3.3, 6.2.1

[233] J. M. Alvarez, T. Gevers, Y. LeCun, and A. M. Lopez, "Road scene segmentation from a single image," in *European Conference on Computer Vision*. Springer, 2012, pp. 376–389. 3.3.2, 3.3

[234] G. Ros, S. Ramos, M. Granados, A. Bakhtiary, D. Vazquez, and A. M. Lopez, "Vision-based offline-online perception paradigm for autonomous driving," in *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*. IEEE, 2015, pp. 231–238. 3.3.2, 3.3

[235] S. Gould, R. Fulton, and D. Koller, "Decomposing a scene into geometric and semantically consistent regions," in *ICCV*, 2009, pp. 1–8. 3.3.3, 3.3, 6.2, 6.2.1

[236] C. Liu, J. Yuen, and A. Torralba, "Nonparametric scene parsing: Label transfer via dense scene alignment," in *CVPR*, 2009. 3.3.3, 3.3

[237] X. Chen, R. Mottaghi, X. Liu, S. Fidler, R. Urtasun, and A. Yuille, "Detect what you can: Detecting and representing objects using holistic models and body parts," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1971–1978. 3.3

[238] P. Sturgess, K. Alahari, L. Ladicky, and P. H. Torr, "Combining appearance and structure from motion features for road scene understanding," in *BMVC 2012-23rd British Machine Vision Conference*. BMVA, 2009. 3.3

[239] Y. Dauphin, H. De Vries, J. Chung, and Y. Bengio, "Equilibrated adaptive learning rates for non-convex optimization," pp. 1504–1512, 2015. 4.2.3

[240] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, and etc., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/ 4.5

[241] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *ACM Intl. Conf. on Multimedia*, 2014, pp. 675–678. 4.5, 4.5

[242] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016. 5.1

[243] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *ICCV*, vol. 111, no. 1, pp. 98–136, 2015. 5.1

[244] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *CVPR*, 2015. 5.1

[245] S. Alpert, M. Galun, R. Basri, and A. Brandt, "Image segmentation by probabilistic bottom-up aggregation and cue integration," in *CVPR*, June 2007. 6.1.1, 6.1.3

[246] C. Liu, J. Yuen, and A. Torralba, "Nonparametric scene parsing: Label transfer via dense scene alignment," in *CVPR*, 2009, pp. 1972–1979. 6.2, 6.2.1

[247] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla, "Segmentation and recognition using structure from motion point clouds," in *ECCV*, 2008, pp. 44–57. 6.2, 6.2.1

[248] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba, "Sun database: Large-scale scene recognition from abbey to zoo," in *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*. IEEE, 2010, pp. 3485–3492. 6.2, 6.2.1

[249] J. Tighe and S. Lazebnik, "Finding things: Image parsing with regions and per-

exemplar detectors," in *CVPR*, 2013, pp. 3001–3008. 6.2.1, 6.5, 6.2.3

[250] S. Rota Bulo and P. Kontschieder, "Neural decision forests for semantic image labelling," in *CVPR*, 2014, pp. 81–88. 6.5

[251] L. Ladick, C. Russell, P. Kohli, and P. H. S. Torr, "Associative hierarchical random fields," *TPAMI*, vol. 36, no. 6, pp. 1056–1077, 2014. 6.6