Convolutional Neural Network in Pattern Recognition

Xi Mo

B.S. Automation, Beihang University, Beijing, China, 2011

M.S. Physics, Beihang University, Beijing, China, 2013

Submitted to the graduate degree program in Department of People who read Abstracts and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Committee members

Dr. Cuncong Zhong, Chair

Dr. Bo Luo, Member

Dr. Taejoon Kim, Member

Dr. Fengjun Li, Member

Dr. Huazhen Fang, Member

Date defended: 5/4/2022

The Dissertation Committee for Xi Mo certifies
that this is the approved version of the following dissertation :

Convolutional Neural Network in Pattern Recognition

Dr. Cuncong Zhong, Chair

Date approved: 5/11/2022

Abstract

Since convolutional neural network (CNN) was first implemented by Yann LeCun et al. in 1989, CNN and its variants have been widely implemented to numerous topics of pattern recognition, and have been considered as the most crucial techniques in the field of artificial intelligence and computer vision. This dissertation not only demonstrates the implementation aspect of CNN, but also lays emphasis on the methodology of neural network (NN) based classifier.

As known to many, one general pipeline of NN-based classifier can be recognized as three stages: pre-processing, inference by models, and post-processing. To demonstrate the importance of pre-processing techniques, this dissertation presents how to model actual problems in medical pattern recognition and image processing by introducing conceptual abstraction and fuzzification. In particular, a transformer on the basis of self-attention mechanism, namely beat-rhythm transformer, greatly benefits from correct R-peak detection results and conceptual fuzzification.

Recently proposed self-attention mechanism has been proven to be the top performer in the fields of computer vision and natural language processing. In spite of the pleasant accuracy and precision it has gained, it usually consumes huge computational resources to perform self-attention. Therefore, realtime global attention network is proposed to make a better trade-off between efficiency and performance for the task of image segmentation. To illustrate more on the stage of inference, we also propose models to detect polyps via Faster R-CNN - one of the most popular CNN-based 2D detectors, as well as a 3D object detection pipeline for regressing 3D bounding boxes from LiDAR points and stereo image pairs powered by CNN.

The goal for post-processing stage is to refine artifacts inferred by models. For the semantic segmentation task, the dilated continuous random field is proposed to be better fitted to CNN-based models than the widely implemented fully-connected continuous random field. Proposed approaches can be further integrated into a reinforcement learning architecture for robotics.

Acknowledgements

I would like to thank all people who has supported me through my academic years as a PhD student. Appreciation for Dr. Wang, Guanghui who advised me through my first three years of study at KU. This dissertation cannot be finished without his support and numerous advice that guided me through the dark days both in my research and daily life. Special thanks to Dr. Zhong, Cuncong, who supported my study during my last two and half years at KU. I also want to thank all my valuable committee members for being on my advisor committee and helping me out at each step of my PhD program.

There are other faculties and staffs from the School of Engineering, Dr. John Gibbons, Ms. Pamala C. Shadoin, Ms. Joy K. Grisafe-Gross, Ms. Marianne A. Reed, who have made my study at KU an extraordinary, impressive experience. Also, I would like to thank my colleagues and leaders during my internships at ABB Robotics and CytoChip Inc., for their passion and inspirations that motivated me in exploring the capabilities of implementing deep learning and computer vision techniques in real-world operations.

Last but not the least, I would like to thank my wife Rui Li, my parents, and my cat Bugs Bunny. Their love, company, and support have made my tough days less tough and good days much brighter.

Contents

1 Introduction to Artificial Neural Networks **1**

1.1 Model the Biological Neurons 1

1.2 Feed-forward Network 2

1.2.1 Perceptron 2

1.2.1.1 Perceptron Model 3

1.2.1.2 Gradient Back-Propagation. 5

1.2.2 Radial-Basis Function Network 10

1.3 Recurrent Neural Network 12

1.3.1 Hopfield Nerual Network 12

1.3.2 Boltzmann Machine 14

1.3.2.1 Restrict Boltzmann Machine 14

1.3.2.2 Deep Belief Network 15

1.3.3 Long/Short-Term Memory 15

1.3.3.1 Vanilla RNN Inference 16

1.3.3.2 State Updating in LSTM 16

1.4 Introduction to Convolutional Neural Network 17

1.4.1 Milestones 17

1.4.2 From Time-Series to Image Filtering 19

1.4.2.1 Cross-Correlation and Convolution 19

1.4.2.2 2D Discrete Convolution 20

1.4.3 Vanilla Convolutional Network 20

1.5 Hybrid Neural Network 21

2 Methodology of Convolutional Neural Network **23**

2.1 Convolutional Layer and Deconvolutional Layer 23

2.1.1 Activation Function 23

2.1.2 Deconvolution or Transposed Convolution? 24

2.1.2.1 Deconvolutional Layer 24

2.1.2.2 Transposed Convolutional Layer 25

2.2 Gradients Back-propagation with Activation Functions 25

2.2.1 Model and Constraints 26

2.2.2 Gradients to the Last Hidden Layer 26

2.2.2.1 Forward Pass 26

2.2.2.2 Gradients Back-Propagation 28

2.2.3 Another Perspective to Convolution 31

2.2.4 Gradients Between Hidden Layers 32

2.3 Loss Function 34

2.3.1 KL-Divergence 34

2.3.1.1 Properties of Entropy 34

2.3.1.2 Relative Entropy 35

2.3.2 Cross-Entropy Loss 36

2.3.3 Focal Loss 36

2.3.4 Ranking Loss 37

2.4 Pooling Layer 38

2.4.0.1 Max Pooling and Avergae Pooling 38

2.4.0.2 Global Pooling 39

2.5 Normalization Layer 39

2.5.1 Batch Normalization 39

2.5.2 Layer Normalization 41

2.5.3 Instance Normalization 41

2.5.4 Group Normalization . . . 41
2.5.5 More Normalization Layers . . . 41
2.6 Training, Test and Validation . . . 42
2.6.1 Pre-Processing . . . 42
2.6.1.1 Normalization . . . 42
2.6.1.2 Decoupling . . . 42
2.6.1.3 Enhancement . . . 43
2.6.2 Training . . . 43
2.6.2.1 Regularization . . . 44
2.6.2.2 Dropout layer . . . 46
2.6.2.3 Optimizers . . . 46
2.6.3 Test . . . 48
2.6.4 Validation . . . 48
2.7 Pre-Processing: Beat-Rhythm Transformer . . . 49
2.7.0.1 Overview . . . 49
2.7.0.2 Concepts and Related Works . . . 49
2.7.0.3 Methodology . . . 52
2.7.0.4 Experiments . . . 59
2.7.0.5 Conclusion . . . 59

3 Pre-Processing: Stereo Frustums for 3D Object Detection 61

3.1 Introduction . . . 61
3.2 Related Works . . . 64
3.2.1 Monocular Pipeline for 3D Detection . . . 64
3.2.2 Stereo-Based Methods . . . 65
3.3 Stereo Frustums Pipeline . . . 67
3.3.1 Dense Epipolar Constraints . . . 67
3.3.2 Pipeline For Stereo Frustums . . . 71

3.3.3 Matching Algorithms . . . 72
3.4 Experiments . . . 73
3.4.1 Experiments Setup . . . 74
3.4.2 Quantitative Evaluation on Validation Set . . . 76
3.4.2.1 Overview on Comparisions . . . 77
3.4.2.2 Comparision with Pseudo-LiDAR . . . 77
3.4.3 Quantitative Evaluation on Test Set . . . 80
3.4.4 Runtime . . . 82
3.4.5 Qualitative Results . . . 83
3.5 Conclusion . . . 84

4 Inference: 2D Object Detection Using Faster R-CNN 85

4.1 Introduction . . . 85
4.2 Related Works . . . 88
4.3 Architecture of Faster R-CNN . . . 89
4.3.1 Backbone Structure . . . 89
4.3.2 RPN and Head Networks . . . 90
4.3.3 Loss Function . . . 91
4.4 Implementation Details . . . 92
4.4.1 Data Preparation . . . 92
4.4.2 Training . . . 94
4.4.3 Validation . . . 95
4.5 Experiments . . . 96
4.5.1 Detection . . . 96
4.5.2 Localization . . . 97
4.5.3 Fine-Tuning vs from Scratch . . . 97
4.6 Conclusion . . . 99

5 Inference: Self-Attention Mechanism for Semantic Segmentation 100

5.1 Introduction . . . 100

5.2 Motivation . . . 102

5.3 Related Works . . . 103

5.3.1 Predict Affordance Maps . . . 103

5.3.2 Self-Attention Modules in Semantic Segmentation . . . 104

5.4 Methodology . . . 105

5.4.1 Hierarchical Inference Architecture . . . 106

5.4.2 Realtime Global Attention . . . 106

5.4.3 Rethink Densely-Stacked Bottlenecks . . . 108

5.4.4 MGRID metric for Evaluation . . . 108

5.5 Experiments . . . 110

5.5.1 Configuration . . . 110

5.5.2 Train and Test . . . 110

5.5.3 Ablation Study . . . 111

5.5.4 Compare to State-of-the-Arts . . . 115

5.6 Conclusion and Future Works . . . 116

6 Post-Processing: Dilated Continuous Random Field for Semantic Segmentation 117

6.1 Introduction . . . 117

6.2 Continuous Random Field (CRF) . . . 120

6.2.1 The General Form . . . 120

6.2.2 DenseCRF for Semantic Segmentation . . . 121

6.3 Related Works . . . 122

6.3.1 Conventional CRF . . . 122

6.3.2 Refinement for Neural Networks . . . 123

6.3.3 CRF as Neural Networks . . . 123

6.4 Methodology . . . 125

6.4.1 Dilated CRF . . . 125
6.4.2 Architecture . . . 126
6.4.3 Training . . . 126
6.5 Experiments . . . 128
6.5.1 Implementation Details . . . 128
6.5.2 Compare to other CRFs . . . 130
6.6 Conclusion and Future Works . . . 131

7 More Future Works: Knowledge Repository 133
7.1 Introduction to Reinforcement Learning . . . 133
7.2 Technique Route . . . 134
7.2.1 Environment . . . 135
7.2.2 Observations . . . 135
7.2.3 Inference Engine . . . 135
7.2.4 Knowledge Repository . . . 135
7.2.5 The Agent . . . 136

8 Summary of Contribution and Discussion 137
8.1 Summary of Contribution . . . 137
8.2 Discussion . . . 139

A Source Codes for BRTransformer 164

B Hammersley-Clifford Theorem 188
B.1 Markov Random Field (MRF) . . . 188
B.2 Hammersley-Clifford Theorem . . . 189

C Copyright Statements 193

List of Figures

1.1 Sketch of a biological neuron. 2

1.2 Perceptron architecture and its multi-layered form. 3

1.3 Sigmoid and hyperbolic tangent activation functions. 5

1.4 Perceptron architecture and its multi-layered form. 11

1.5 Topology of a discrete Hopfield neural network with 6 binary neurons. 13

1.6 Vanilla recurrent neural network and long/short-term memory cell. 15

1.7 LeNet-5 (92) architecture for handwriting recognition. 18

1.8 One vanilla convolutional layer. 20

2.1 A $(l+1)$-layers convolutional neural network framework. 25

2.2 An example of one convolutional layer with a 3-set kernel activation. 26

2.3 Sliding kernel (left) and fixed kernel (right) at meantime. 31

2.4 Visualization of batch norm, layer norm, instance norm and group norm. 40

2.5 Generalization ability in training, and L1-, L2- regularization terms. 44

2.6 Optimizers comparison (83) on MNIST (32) dataset. 45

2.7 ECG comparison on normal type (137) and AF-events (126). 50

2.8 BRTransformer work flow, transformer image from (196). 52

2.9 Sample re-balance procedure before feeding to BRTransformer. 53

2.10 Local-Global Filtering Peak Detectors. 54

2.11 LGFPD results of same local window size, and its best result on test signals. 54

2.12 Positional embedding of conceptualized and fuzzified batch of samples. 56

2.13 Scaled dot-product attention and multi-head attention (196). 57

3.1 Illustration of stereo frustums. 64

3.2 Siamese pipeline for 3D object detection. 67
3.3 Coordinate systems of the KITTI data-collection vehicle (46) 68
3.4 P-R curves of 3D object detection results. 75
3.5 Runtime comparison between matching modules and original FPNv2. 80
3.6 Qualitative evaluation on validation set using Mask R-CNN detector. 81

4.1 Features of Faster R-CNN detector. 87
4.2 Structure for polyp detection. 88
4.3 Failure modes for Faster R-CNN. 98

5.1 Hierarchical inference architecture of 5-scales RGANet5. 101
5.2 Forward pass of RGANet5. 102
5.3 ESS-3 and realtime GAM. 105
5.4 MGRID metric. 110
5.5 Qualitative results on test-set. 113

6.1 Message-passing sketches for DenseCRF and DilatedCRF. 119
6.2 DilatedCRF pipeline. 121
6.3 DSConv module. 122
6.4 Training Performance w.r.t. epochs. 124
6.5 Visualization of affordance maps. 130

7.1 Robotic system powered by knowledge repository for reinforcement learning. . . . 134

List of Tables

2.1 BRTransformer performance on validation set. 59

3.1 3D object detection on car category using two 2D detectors. 76
3.2 BEV results on validation set. 77
3.3 BEV results on validation set. 78
3.4 3D detection results on validation set. 78
3.5 3D detection results on validation set. 78
3.6 3D detection results on validation set. 79
3.7 BEV detection results on validation set. 79
3.8 3D detection APs(%) on test set(Geiger et al.). 82
3.9 3D detection APs(%) on test set (Geiger et al.). 82
3.10 BEV detection APs(%) on test set (Geiger et al.). 83
3.11 BEV detection APs(%) on test set (Geiger et al.). 83

4.1 Validation metrics for polyp detection. 92
4.2 Validation metrics for polyp localization. 93
4.3 Fine-tuned detection results for 300 proposals on accuracy, precision and recall. . . 94
4.4 Fine-tuned detection results for 300 proposals on F1-/F2-scores and RT. 94
4.5 Fine-tuned localization results for 300 proposals and comparison (part. 1). 95
4.6 Fine-tuned localization results for 300 proposals and comparison (part. 2). 95
4.7 Performance comparison. 96

5.1 Ablation study on GAM and CELoss w/ adaptive weights (AW). 112
5.2 Comparison with large semantic segmentation models in model size. 113

5.3 Comparison with large semantic segmentation models in metrics. 114
5.4 Comparison with light-weighted semantic segmentation models in model size. . . . 114
5.5 Comparison with light-weighted semantic segmentation models in metrics. 115

6.1 DilatedCRF evaluation on test set (part. 1), all metrics are formatted as mean $\pm$ std. 127
6.2 DilatedCRF evaluation on test set (part. 2), all metrics are formatted as mean $\pm$ std. 127
6.3 Performance of ConvCRF and DenseCRF on test set. 129

Chapter 1

Introduction to Artificial Neural Networks

Abstract

Artificial neural network (ANN) has been developed to simulate groups of biological neurons both in their organization form and reactions to external stimuli. ANNs are considered as one of the most crucial techniques to bestow intelligence on machines. Based on the neuron types, ANNs can be categorized as conventional feed-forward network (FFN), recurrent neural network (RNN) including Boltzmmann machine, Hopfield neural network, long/short-term memory (LSTM), special FFN - convolutional neural network and hybrid neural network (HNN). Though there are huge varieties of ANN, this chapter focuses on introducing several popular neural networks in those categories except the convolutional neural network (CNN), for which relevant techniques and basic concepts are demonstrated in Chapter 2.

1.1 Model the Biological Neurons

For decades, neurologists have striven to acquire knowledge on the biomechanism of brain and nervous system. One intuitive motivation of this research is to get a thorough comprehension upon the neural mechanism that the brain learns, memorizes, and infers in response to outer stimuli. Meanwhile, computer scientists and computational mathematicians aim to establish explainable neural network models on the nervous system according to the knowledge of biological nervous system. These models make artificial intelligence feasible, but are still under development nowadays. In

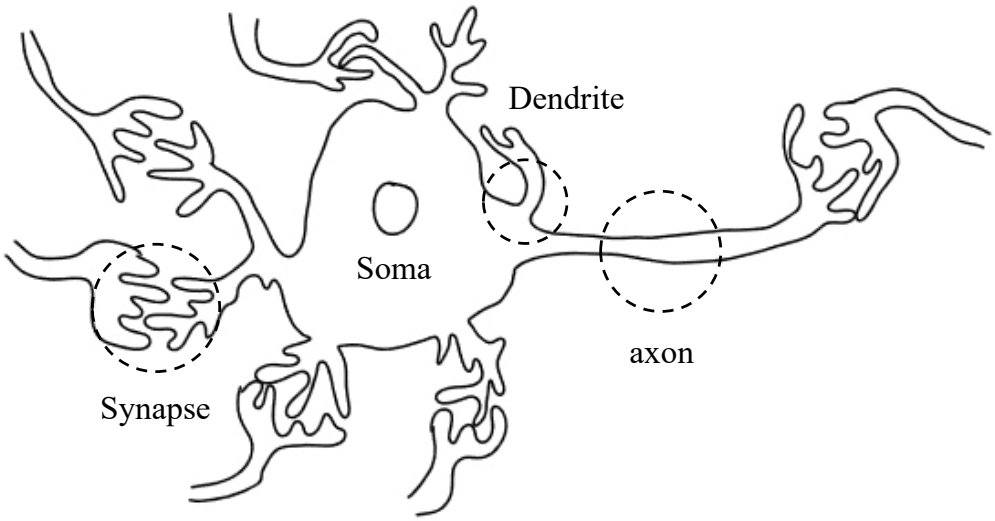


Figure 1.1: Sketch of a biological neuron.

1949, Donald Hebb published his book *The Orgnization of Behavior: A Neuropsychological Theory* and Hebb's rule (60) quickly became popular among academia. The rule states that neurons that fire together are wired together. If using a weight to quantify the relevance between two neurons, then it increases when both neurons are activated simultaneously, while reduces when they activate separately. Hebb's rule represents the very beginning of quantifying stimuli via weights between neurons. However, it does not cover the structural modeling of a simplified biological neuron. As depicted in Fig. 1.1, brain nervous system consists of numerous neurons: ~0.1 million neurons in fruit fly's brain, and 1.4 - 1.6 billion in human's brain. One neuron has multiple dendrites and one axon, external stimuli is carried between neurons via synapse, which release a chemical, namely neurotransmitter. The whole picture is much more complex, since the neurons are connected densely more than the style shown in Fig. 1.1.

1.2 Feed-forward Network

1.2.1 Perceptron

FFN is a category of neural networks that stimuli is unidirectionally transmitted starting at the input layer, and terminated at the output layer. In 1958, Frank Rosenblatt from Coneel Aeronautical

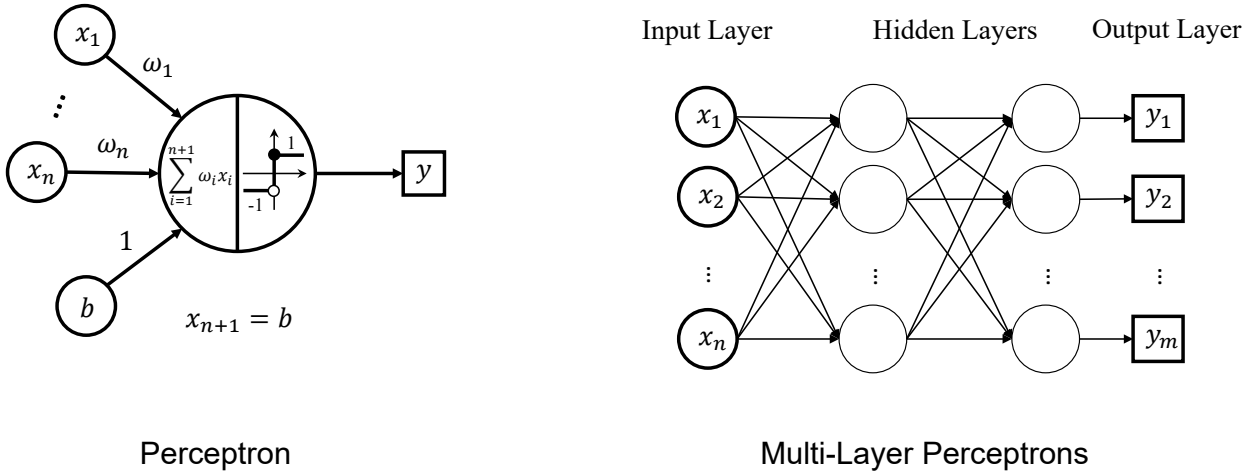


Figure 1.2: Perceptron architecture and its multi-layered form.

Laboratory invented the perceptron algorithm (156), which immediately attracted the attention of academia, and being recognized as the prototype of FFN. Perceptron is the mathematic abstract of one biology neuron that simplifies the dendrites, axon, and stimuli transmission.

1.2.1.1 Perceptron Model

As shown in Fig. 1.2, perceptron is an artificial neuron for binary classification. Its feed-forward stream can be denoted by a mathematic model: Given input vector $X = [x_1, ..., x_{n+1}]^T \in \mathbb{R}^{n+1}$, where $x_{n+1} = b$ is the bias; corresponding weights vector $W = [\omega_1, ..., \omega_n, 1]^T \in \mathbb{R}^{n+1}$, where ω_{n+1} is the weight to the bias b; activation function $f(\cdot)$. The output scalar y is then formulated as

$$z = \sum_{i=1}^{n+1} \omega_i x_i = W^T X, \;\; y = f(z) = \begin{cases} 1, & z \geq 0 \\ -1, & z < 0 \end{cases}. \tag{1.1}$$

Primitive Perceptron Learning Algorithm Given a train set $\mathscr{T} := \{s_i\}, 1 \leq i \leq N$; sample $s_i := (X_i, \hat{y}_i)$, where input vector $X_i \in \mathbb{R}^{n+1}$, and the label variable $\hat{y}_i \in \{-1, 1\}$ indicates the category X_i belongs to; learning rate $0 < \eta \leq 1$. Let $\mathscr{E} := \{X_i \mid y_i \neq \hat{y}_i, \; \forall 1 \leq i \leq N\}$, which denotes the set of

all incorrectly predicted samples, Rosenblatt's criterion function $\Gamma(W)$ is defined as

$$\Gamma(W) = -\sum_{X_i \in \mathscr{E}} \hat{y}_i W^T X_i, \tag{1.2}$$

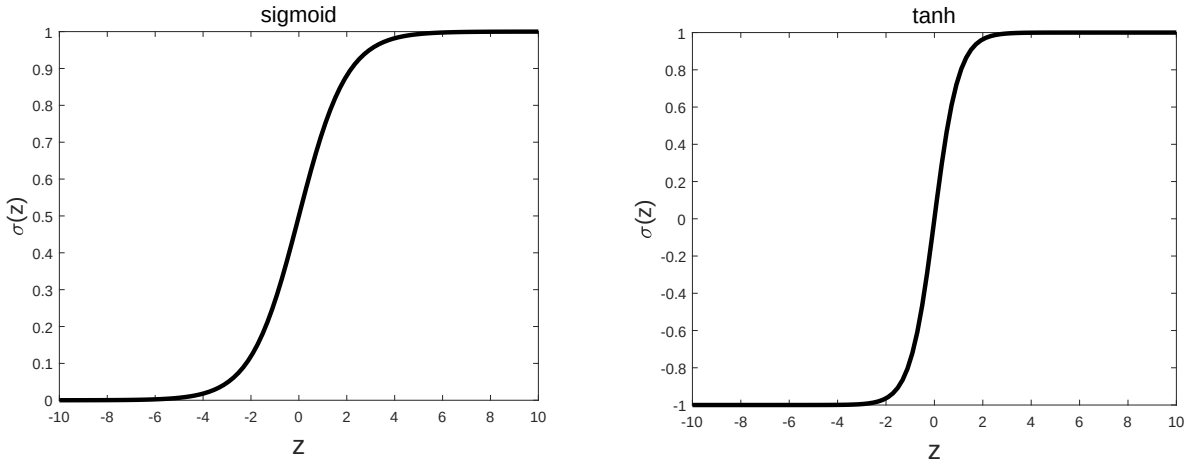
$\Gamma(W)$ is non-negative since $\hat{y}_i W^T X_i \leq 0$ for any misclassified sample. The target for learning is to learn the optimized W^* by minimizing $\Gamma(W)$: $W^* = \min_W \Gamma(W)$. Alg. 1 shows the steps of primitive perception in learning W from $\mathscr{T}$ without gradient back-propagation since the activation function f is discontinuous. Train set is considered as linear separable if $\exists \gamma \in R$ s.t. $\forall s_i \in \mathscr{T}$, $\hat{y}_i W^T X_i > \gamma$, Novikoff proved that, if the training set $\mathscr{T}$ is linear separable, then the primitive perceptron algorithm converges within finite iterations.

Algorithm 1: Primitive perceptron learning algorithm.

```
Initialize W, set total iterations N_s and threshold ε;
while Γ(W) > ε and current iteration ≤ N_s do
    for each X_i do
        if ŷ_i W^T X_i ≤ 0 then
            W := W + ηX_i;
    current iteration increased by 1;
return W
```

Multi-Layer Perceptrons The term 'Perceptron' can be either interpreted as the singe-layer perceptrons network or the single perceptron depicted as Fig. 1.2. One perceptron cannot find a hyper plane in nonlinear-separable sample space (128). However, multi-layer perceptrons (MLP) overcome this defect and become the main stream research (159) on neural networks till the 21st century. As illustrated in Fig. 1.2, MLP consists of one input layer, multiple hidden layers of perceptrons[1], and one output layer. Note that the nodes are densely-connected such that MLP is also named as the fully-connected layers. The input and output layers are not perceptrons but abstract layers. Hidden layers are perceptrons whose states are not observable to users, this is the main reason why MLP is known as a 'black box' for users.

[1] In contrast to MLP, Perceptron has only one hidden layer.



(a) Sigmoid activation function.

(b) Hyperbolic tangent activation function.

Figure 1.3: Sigmoid and hyperbolic tangent activation functions.

1.2.1.2 Gradient Back-Propagation.

Modern MLP is more powerful than primitive MLP due to the gradient back-propagation (BP) training algorithm and larger amount of activation functions. The primitive MLP[2] only implements two kinds of activation functions:

$$f(z) := sigmoid(z) = \sigma(z) = \frac{1}{1+e^{-z}},\ f(z) := tanh(z) = \sigma(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}, \tag{1.3}$$

where variable z is defined the same as Equ. 1.1. Two activation functions are continuous such that both are differential (see Fig. 1.3), i.e., for $sigmoid(z)$, $0 < \sigma(z) < 1,\ \sigma'(z) = (1-\sigma(z))\sigma(z), \forall z \in \mathbb{R}$. It's obvious that $\sigma'(z)$ achieves its maximum 0.25 at $z = 0.5$; for the hyperbolic tangent activation function $tanh(z)$, $-1 < \sigma(z) < 1,\ \sigma'(z) = 4/(2+e^{2z}+e^{-2z}), \forall z \in \mathbb{R}$ with a maximum 1.0 at $z = 0$. BP algorithm can be denoted as updating weights for all perceptrons leveraged on the partial derivatives of outputs with respect to inputs.

[2]The MLP implemented in Rumelhart et al.'s work.

MLP Learning Algorithms. Given the training set $\mathscr{T}$ and the MLP model shown in Fig. 1.2, loss function[3] $L(W)$ is proposed to evaluate the similarity between the distribution of predicted output $Y = [y_1, ..., y_m]$ and its corresponding groundtruth $\hat{Y} = [\hat{y}_1, ..., \hat{y}_m]$. Mean squared error (MSE, L2-norm) and mean absolute error (MAE, L1-norm) are the earliest criterion metrics applied to loss function:

$$L(W) := MSE(Y) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2,\ L(W) := MAE(Y) = \frac{1}{m}\sum_{i=1}^{m}|y_i - \hat{y}_i|. \tag{1.4}$$

In addition to the notations mentioned above, the variables related to layers and the order of perceptron should be made explicitly. Define the number of hidden layers as n_h, the output of k-th hidden layer as $Y_k = \left[y_1^k, ..., y_{n_k}^k\right]$, $0 \leq k \leq n_h$, where y_i^k signifies the i-th output of perceptron from the k-th layer. It should be noted that $Y_{n_h} = Y$ and $n_{n_h} = m$, $Y_0 = X$ and $n_0 = n + 1$ according to both definitions. The weights matrix W_k of the k-th layer collects the weights of all perceptrons as $W_k = \left[W_1^k, ..., W_{n_k}^k\right]$, $W_i^k \in \mathbb{R}^{n_{k-1} \times 1}$ and $k \geq 1$.

Algorithm 2: MLP learning algorithm - standard SGD optimizer

```
Initialize W = {W_1, ..., W_{n_h}} by non-zero numbers. set total iterations N_s;
while current iteration ≤ N_s do
    for each X_i do
        One forward pass. From 1st hidden layer to the n_h-th hidden layer, recursively
          calculate Y_1, ..., Y_{n_h - 1} and Y_{n_h} = Y using Y_i = W_i^T Y_{i-1}, i ≥ 1;
        Calculate loss L(W) using Eqn. 1.4;
        Update all weights inversely from the n_h-th hidden layer back to the 1st hidden
          layer using W_i := W_i - η ∂L(W)/∂W_i;
    current iteration increased by 1;
return W
```

The fundamental learning algorithm, namely standard stochastic gradient descent (SGD) learning algorithm is demonstrated as Alg. 2. Standard SGD algorithm updates W after one sample is fed to MLP, while for the batch SGD algorithm (refer to Alg. 3), the training set is grouped into

[3] Y is derived from the mapping from known sample space and unknown parametric space, thus can be regarded as a function with respect to the parameter W.

Algorithm 3: MLP learning algorithm - batch SGD optimizer

Initialize $W = \{W_1, ..., W_{n_h}\}$ by non-zero numbers. set total iterations N_s;
while *current iteration* $\leq N_s$ **do**
  for *every batch of* N_b *samples* **do**
    N_b forward passes. From 1st hidden layer to the n_h-th hidden layer, recursively calculate $Y_1, ..., Y_{n_h-1}$ and $Y_{n_h} = Y$ for N_b samples using $Y_i = W_i^T Y_{i-1}, i \geq 1$;
    Calculate N_b losses $L(W) = \{L_1(W), ..., L_{N_b}(W)\}$ using Eqn. 1.4;
    Update all weights inversely from the n_h-th hidden layer back to the 1st hidden layer using $W_i := W_i - \frac{\eta}{N_b} \sum_{i=1}^{N_b} \frac{\partial L_i(W)}{\partial W_i}$;
  current iteration increased by 1;
return W

batches with each contains N_b samples, the weights are then updated per batch. The most crucial step of BP algorithms is to calculate the gradients $\frac{\partial L(W)}{\partial W_i}$.

Chain Rule for BP Algorithms. Gradients $\frac{\partial L(W)}{\partial W_i}$ can be calculated via chain rule (formulated as Corollary 1.2.0.3) and partial derivatives with respect to vectors and matrix. Two useful rules for the partial derivatives can be described as Corollary 1.2.0.1 and Corollary 1.2.0.2.

Corollary 1.2.0.1. *Given linear transformation* $Y = W^T X, Y \in \mathbb{R}^{m\times 1}, W \in \mathbb{R}^{n\times m}, X \in \mathbb{R}^{n\times 1}$*, then* $\frac{\partial Y}{\partial X} = W^T$ *in the Jacobian layout.*

Proof. According to the condition $Y = W^T X$, $Y_i = \sum_j W_{(j,i)} X_j \Rightarrow \frac{\partial Y_i}{\partial X_j} = W_{(j,i)}$, the Jacobian matrix

$$\frac{\partial Y}{\partial X} = \begin{pmatrix} \frac{\partial Y_1}{\partial X_1} & \cdots & \frac{\partial Y_1}{\partial X_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial Y_m}{\partial X_1} & \cdots & \frac{\partial Y_m}{\partial X_n} \end{pmatrix} = \begin{pmatrix} W_{(1,1)} & \cdots & W_{(n,1)} \\ \vdots & \ddots & \vdots \\ W_{(m,1)} & \cdots & W_{(n,m)} \end{pmatrix} = W^T.$$

□

Corollary 1.2.0.2. *Given linear transformation* $Y = W^T X, Y \in \mathbb{R}^{m\times 1}, W \in \mathbb{R}^{n\times m}, X \in \mathbb{R}^{n\times 1}$*, then* $\frac{\partial Y}{\partial W} = [X, \cdots, X]^T$*, which has the dimensions* $m \times n$ *in the Jacobian layout.*

Proof. According to the condition $Y = W^T X$, $Y_i = \sum_j W_{(j,i)} X_j$, and $\frac{\partial Y}{\partial W} = \sum_{i=1}^{m} \frac{\partial Y_i}{\partial W}$. $\frac{\partial Y_i}{\partial W}$ is performed for every component of Y, and only i-th component of Y yields a non-zero derivative X_j

with respect to $W_{(j,i)}$, otherwise the derivative is 0, i.e,

$$\frac{\partial Y_i}{\partial W} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ X_1 & X_2 & \cdots & X_n \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}_{m \times n},$$

where the i-th row equals to X^T, therefore $\frac{\partial Y}{\partial W} = \sum_{i=1}^{m} \frac{\partial Y_i}{\partial W} = [X, \cdots, X]^T \in \mathbb{R}^{m \times n}$. □

Corollary 1.2.0.3 (Chain rule for BP Algorithms)**.**

$$\frac{\partial L(W)}{\partial W_k} = \frac{\partial L(W)}{\partial Y_{n_h}} \bullet \frac{\partial Y_{n_h}}{\partial Y_{n_h-1}} \bullet \cdots \bullet \frac{\partial Y_{k+1}}{\partial Y_k} \bullet \frac{\partial Y_k}{\partial W_k}, \quad 1 \leq k \leq n_h. \tag{1.5}$$

The forwarding of perceptrons in k-th hidden layer can also be modeled by Eqn. 1.3, that $Y_k = \sigma(Z_k) = \sigma(W_k^T Y_{k-1})$. According to Corollary 1.2.0.3, the loss δ_k back-propagated to the k-th hidden layer, and the gradients propagated from Y_k to W_k can be denoted as

$$\delta_k = \frac{\partial L(W)}{\partial Y_{n_h}} \bullet \frac{\partial Y_{n_h}}{\partial Y_{n_h-1}} \bullet \cdots \bullet \frac{\partial Y_{k+1}}{\partial Y_k}, \quad \frac{\partial Y_k}{\partial W_k} = \frac{\partial Y_k}{\partial Z_k} \cdot \frac{\partial Z_k}{\partial W_k}. \tag{1.6}$$

The chain rule can now be written in a gradients/loss back-propagation form:

$$\frac{\partial L(W)}{\partial W_k} = \delta_k \bullet \frac{\partial Y_k}{\partial Z_k} \bullet \frac{\partial Z_k}{\partial W_k}. \tag{1.7}$$

Computational Form of Chain Rule. Eqn. 1.7 represents the back-propagation process of gradients, whereas its form cannot be directly applied to implementations. To further calculate the

values with Corollary 1.2.0.1 and 1.2.0.2, components of Eqn. 1.7 can be reformulated as

$$\begin{aligned}
\delta_k &= \frac{\partial L(W)}{\partial Y_{n_h}} \cdot \frac{\partial Y_{n_h}}{\partial Y_{n_h-1}} \cdot \cdots \cdot \frac{\partial Y_{k+1}}{\partial Y_k} = \delta_{n_h} \cdot \prod_{i=n_h}^{k+1} \frac{\partial Y_i}{\partial Y_{i-1}} \\
&= \delta_{n_h} \cdot \prod_{i=n_h}^{k+1} \sum_{j=1}^{n_i} \frac{\partial y_j^i}{\partial Z_i} \cdot \frac{\partial (W_i^T Y_{i-1})}{\partial Y_{i-1}} = \delta_{n_h} \cdot \prod_{i=n_h}^{k+1} \sum_{j=1}^{n_i} \frac{\partial y_j^i}{\partial Z_i} \cdot W_i^T, \\
\frac{\partial Y_k}{\partial Z_k} \cdot \frac{\partial Z_k}{\partial W_k} &= \sum_{j=1}^{n_k} \frac{\partial y_j^k}{\partial Z_k} \cdot \frac{\partial (W_k^T Y_{k-1})}{\partial W_k} = \sum_{j=1}^{n_k} \frac{\partial y_j^k}{\partial Z_k} \cdot [Y_{k-1}, \cdots, Y_{k-1}]^T_{n_k \times n_{(k-1)}}
\end{aligned} \tag{1.8}$$

Eqn. 1.8 indicates one important point of view that if a variable is relevant to multiple sources, the summation of gradients from all these sources equals the gradient propagated to that variable. There are derivatives $\frac{\partial y_j^i}{\partial Z_i}, k \leq i \leq n_k$, that remain to be quantified. Given the fact that y_j^i is the j-th component of vector $\sigma(Z_i)$ according to Eqn. 1.3, thus $\frac{\partial y_j^i}{\partial Z_i} = \Big[0, \cdots, 0, \sigma'(y_j^i), 0, \cdots, 0\Big]$, where only j-th component is non-zero. Thus, $\frac{\partial y_j^i}{\partial Z_i}$ acts as a j-th column selector for matrices W_i^T and $[Y_{k-1}, \cdots, Y_{k-1}]$ in Eqn. 1.8. As a result, it can be rewritten as Eqn. 1.9:

$$\sum_{j=1}^{n_i} \frac{\partial y_j^i}{\partial Z_i} \cdot W_i^T = \begin{pmatrix} \sigma'(y_1^i) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma'(y_{n_i}^i) \end{pmatrix}_{n_i \times n_i} \cdot W_i^T := A_i W_i^T. \tag{1.9}$$

Likewise, we can reformulate Eqn. 1.8 as

$$\begin{aligned}
\delta_k &= \delta_{n_h} \cdot \prod_{i=n_k}^{k+1} \sum_{j=1}^{n_i} \frac{\partial y_j^i}{\partial Z_i} \cdot W_i^T = \delta_{n_h} \cdot \prod_{i=n_k}^{k+1} A_i W_i^T, \\
\frac{\partial Y_k}{\partial Z_k} \cdot \frac{\partial Z_k}{\partial W_k} &= \sum_{j=1}^{n_i} \frac{\partial y_j^k}{\partial Z_k} \cdot [Y_{k-1}, \cdots, Y_{k-1}]^T_{n_k \times n_{(k-1)}} = A_k [Y_{k-1}, \cdots, Y_{k-1}]^T
\end{aligned} \tag{1.10}$$

Thereafter, the final computational form of chain rule is described as Corollary 1.2.0.4.

Corollary 1.2.0.4 (Computational form of chain rule)**.**

$$\frac{\partial L(W)}{\partial W_k} = \delta_{n_h} \left(\prod_{i=n_k}^{k+1} A_i W_i^T \right) A_k [Y_{k-1}, \cdots, Y_{k-1}]^T. \tag{1.11}$$

Gradient Vanishing Problem Sigmoid and *tanh* activation function suffers from gradient vanishing problem especially when n_h is large enough. In this part, the gradient vanishing problem is addressed via computational form as illustrated in Corollary 1.2.0.4. It is obvious that the activation functions affect the gradients via matrix $A_i,\ k \leq i \leq n_h$. Assume the loss is low enough such that $\frac{\partial L(W)}{\partial W_k} \to 0$, and it indicates δ_{n_h}, W_i^T, Y_k are all very low in their induced norms. In this case, for instance, $\|A_i W i^T\| \leq \|A_i\| \cdot \|W_i^T\|, \forall k \leq i \leq n_h$, and refer to Eqn. 1.3, the ranges of sigmoid and *tanh* activation functions are $(0, 0.25]$ and $(0, 1.0]$ respectively. Therefore, $\|A_i W_i^T\| \leq \|W_i^T\|, \forall k \leq i \leq n_h$, and after multiplying A_i for $n_h - k + 1$ times, the gradients may diminish towards zeros.

Another issue for sigmoid and *tanh* activation functions is the early saturation issue. This issue also reflects the gradient vanishing problem. Refer to Fig. 1.3, the gradients of both functions drop to nearly 0 when $|z| > 4$ for sigmoid, and $|z| > 2$ for *tanh*, which are liable to step into the saturation regions in which the weights will not be updated. The gradient vanishing problem restricts the depth of primitive MLP until novel activation functions such as *ReLU* are proposed for deeper neural networks. These novel activation functions are discussed in Chapter 2.

1.2.2 Radial-Basis Function Network

Another typical FFN is Radial-Basis Function (RBF) Network (17; 108). The historical motivations of RBF network are to approximate arbitrary continuous functions and to solve the classical XOR problem that Perceptron cannot realize XOR logic[4] since Perceptron cannot partition non linear-separable sample space. RBF network implements a radial-basis kernel $\phi(\cdot)$ to project the distance vector between one neuron center and one sample to an infinite dimensional space, i.e,

$$\Phi(\|X - C_i\|) = \exp(-\frac{\|X - C_i\|^2}{2\sigma_i^2}), \tag{1.12}$$

[4]But for MLP the problem is solved.

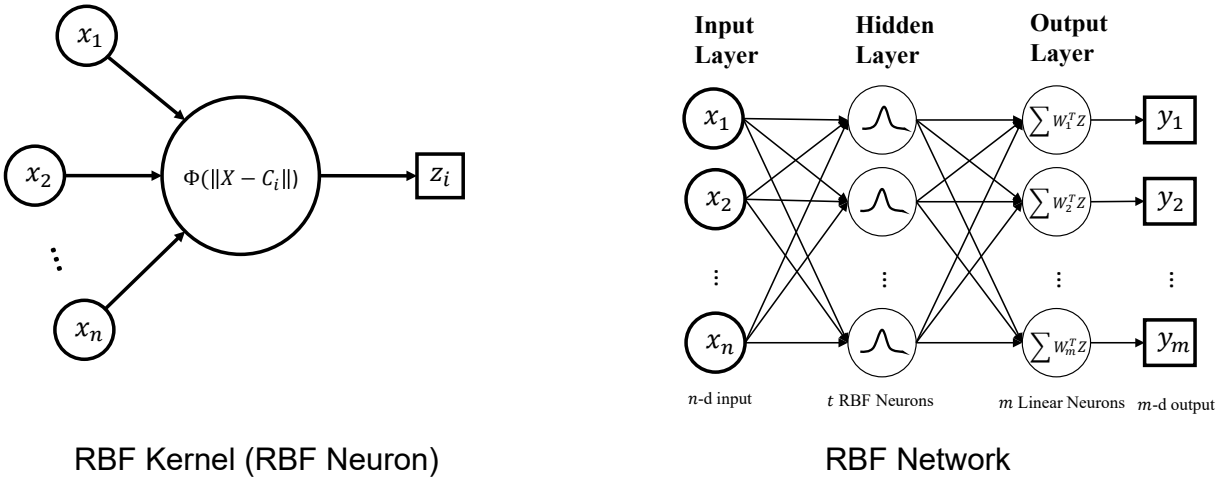


Figure 1.4: Perceptron architecture and its multi-layered form.

where $X \in \mathbb{R}^{n\times 1}$ signifies the input of a sample from the training set $\mathscr{T}$ with an annotation $\hat{Y} \in \mathbb{R}^{m\times 1}$, $C_i \in \mathbb{R}^{n\times 1}$ is the center of i-th neuron, and σ_i signifies the radius of i-th neuron. Distinguished from Perceptron, RBF network (see Fig. 1.4) is composed of three featured layers: the input layer that collects input vectors, the hidden layer that computes distance vectors then projects them via RBF kernel, and the output layer that aggregates intermediate outputs of neurons by weights.

As depicted in Fig. 1.4, one feed-forward pass of RBF network firstly collects an input sample $X = [x_1, \cdots, x_n]^T$ and feed it to t RBF neurons respectively. The intermediate outputs of hidden layer $Z = [z_1, \cdots, z_t]^T$ are then transmitted to the output layer consists of m linear neurons, which behaves similar to the perceptron without activation function. The final output $Y = [y_1, \cdots, y_m]^T$ can be aggregated into the loss function as shown in Eqn. 1.4.

To learn parameters from $\mathscr{T}$, the weights in output layer can be learnt via loss back-propagation. Centers matrix $C = [C_1, \cdots, C_t]$ and radius $\sigma_i,\ 1 \leq i \leq t$ must be preset before training since the necessary initialization of centers and radius can not be learnt via gradient back-propagation, e.g.,

$$\frac{\partial \Phi}{\partial \sigma_i} = \exp(-\frac{\|X - C_i\|^2}{2\sigma_i^2}) \cdot \frac{\|X - C_i\|^2}{\sigma_i^3},$$

whose initial values of C_i and σ_i must be preset. This requirement brings about another challenge of

how to properly set the initial values. There are two categories of approaches to achieve this goal. One category of approaches aims at randomly assigning values as for perceptrons, and the other one predicts more reasonable structure information via clustering, e.g., K-Means algorithm (117) for assigning centers and radius while it costs additional computational resources to cluster samples. The later yields more precise results since random assignment often leads to false centers and radius such that the RBF network is more liable to fall into local optimum.

1.3 Recurrent Neural Network

One intuitive way to understand RNN is that RNN is a category of neural networks that each neuron can take its current output as part of its next input. RNN can vary in organization form and neuron composition. The latest RNN techniques have set the records for multiple NLP tasks such as voice recognition, machine translation, speech generation etc. This section does not intend to go through details in proofs and theorems due to their weak relevance to CNN, but to present a conceptual introduction to the fundamental mechanism.

1.3.1 Hopfield Nerual Network

Hopfield network (65) is widely known as the first RNN model in history. Its topology simulates more connections than the Perceptron among biological neurons. One most distinguish feature of Hopfield network is its ability to recurrently update hidden states of neurons by time, which is also termed as associate memory. Hopfield network can be further categorized as continuous Hopfield neural network (CHNN, 1984) and discrete Hopfield neural network (DHNN, 1990). The main differences between CHNN and DHNN are the form of energy functions and the way energy functions are optimized. Moreover, classical DHNN aims at modeling the biological process of associating, that the outputs are deduced by input samples without global optimization, while CHNN aims at seeking the global optimum of energy function, which is regarded as a solver for varieties of dynamics optimization problems.

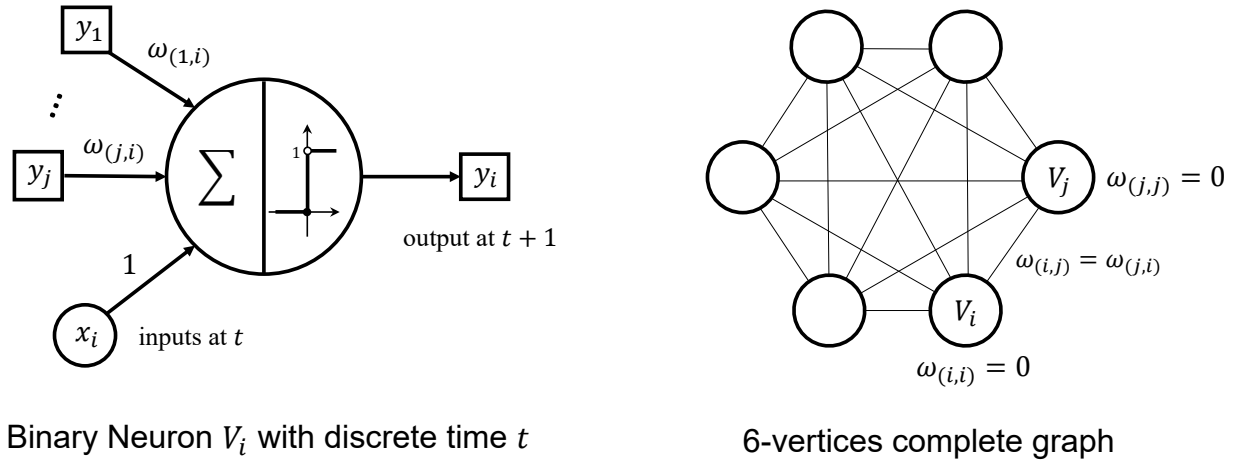


Figure 1.5: Topology of a discrete Hopfield neural network with 6 binary neurons.

Topology of Hopfield Neural Network The topology of DHNN and CHNN are the same compete graph $\mathscr{G} = \langle \mathscr{V}, \mathscr{E} \rangle$. $\mathscr{V} = \{V_i | 1 \leq i \leq N\}$ is the set of vertices, and $\mathscr{E}$ the set of edges that connects vertices. The graph is weighted and undirected, Fig. 1.5 depicts an example of $N = 6$ complete graph of DHNN.

Each vertice of $\mathscr{G}$ indicates a binary neuron, the i-th neuron receives inputs from all other neurons as well as an direct input $x_i(t)$ at discrete time t. Similar to the activation function of a perceptron, the next output $y_i(t+1) = f\left(\sum_{k=1,k\neq i}^{N} \omega_{(k,i)} y_k(t) + x_i(t)\right)$, where sign activation function f can be valued as $[0,1]$ or $[-1,1]$. $y_i(t)$ is also termed as activity of i-th neuron or the strength of synaptic input form i-th neuron to any other neuron.

Hebb's rule (60) and Storkey's rule (182) have been implemented to train a DHNN. The updates of weights are performed locally since both learning algorithms only take neurons associated to similar patterns or neurons nearby into consideration. As Hopfield networks intends to memorizes and associate states of neurons, the unsupervised training process does not require ground-truth annotations but the input patterns only. The convergence of state transition among neurons has been proved to be a minimum of Lyapunov function (8), an important stability corollary under the context of state space (225) and Lyapunov stability in control theories. The feed-back topology and time-series logic indicate that the Hopfield network is also a non-linear control system whose

state space at timestamp $t+1$ is depending on its previous state space at t.

1.3.2 Boltzmann Machine

Perceptron and RBF network etc. are all trained with samples and ground-truth, which is also known as the supervised learning. Boltzmann machine (3) is an unsupervised neural network with the same topology as Hopfield network as shown in Fig. 1.5. Distinguished from the complete graph of Hopfield network, the graph of Boltzmann machine is termed as probability undirected graph that merely depicts the dependence of hidden vertices $\mathscr{H} = \{h_i | 1 \leq i \leq N_h\}$ and visible vertices $\mathscr{V} = \{v_i | 1 \leq i \leq N_v\}$. Also, as Boltzmann machine does not have any exterior outputs such that gradient back-propagation algorithms cannot be directly implemented to Boltzmann machine. However, by minimizing the same energy function as of DHNN towards anticipated probability distribution of states, Boltzmann machine can be trained via simulated annealing (139) algorithm.

1.3.2.1 Restrict Boltzmann Machine

Smolensky named one variant of Boltzmann machine as Harmonium in 1985, later in 2006, Hinton & Salakhutdinov proposed restrict Boltzmann machine (RBM), and invented the fast contrastive divergence learning algorithm for its training. RBM simplifies the dense connection of visible and invisible neurons that RBM is composed by a two-layer structure of which the first layer is named after visible layer and the other is named after invisible layer. All neurons in visible layer are connected to those in invisible layers, while there are no connections between any two neurons in the same layer. The training process of RBM reveals the modern form of learning - feed-forward, with back-propagation via Gibbs sampling (48) and loss constructed by contrasting target distributions to the predicted distribution. Later in 2009, Salakhutdinov & Larochelle proposed an deeper RBM, namely deep Boltzmann machine (DBM) that stacks multiple RBMs. Despite the innovative idea for the modern neural network structure, DBM and RBM cannot be efficiently implemented for large-scale real-world problems.

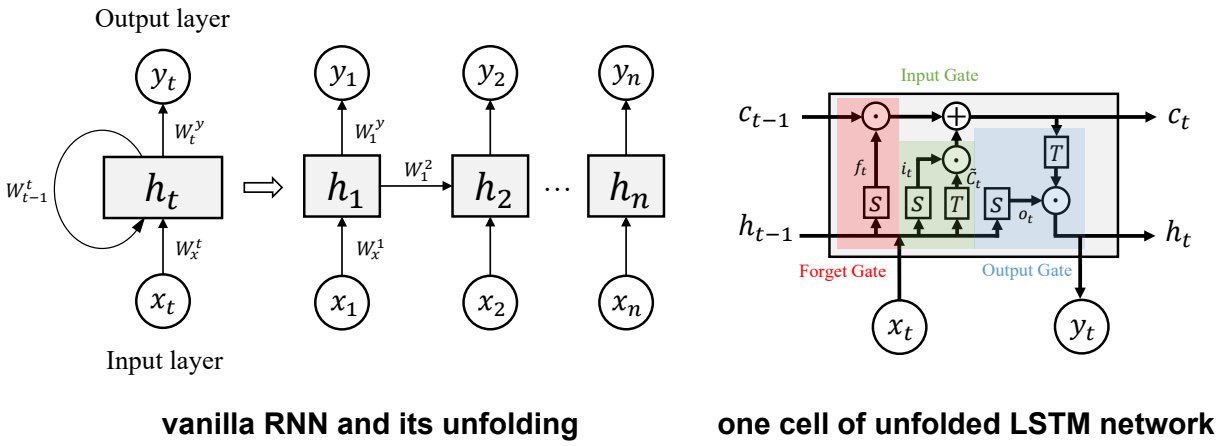


Figure 1.6: Vanilla recurrent neural network and long/short-term memory cell.

1.3.2.2 Deep Belief Network

The widely-known gradient back-propagation algorithm (see Alg. 2) is originated from Hinton's work on deep belief network (DBF) that describes a category of deep neural networks with hidden units (latent variables), whose neurons are disconnected in the same layer, whereas connected between layers. The gradient back-propagation updates weight $\omega_{ij}(t)$ at the timestamp $t+1$ by adding the increments derived from the partial derivative of a loss function in regarding to $\omega_{ij}(t)$. This training strategy is firstly implemented to RBM, it then becomes the modern training strategy for DBF, such as MLP and autoencoder (91).

1.3.3 Long/Short-Term Memory

In the field of classifying and predicting on time-series data, states of long distance recurrent neurons of vanilla RNN (39; 80) solver are overwhelmed by nearby neurons. Long/Short-term memory network (LSTM) is proposed by Hochreiter & Schmidhuber to strengthen the impact of long-term relationships. Moreover, LSTM alleviates the vanilla RNN's well-known long-term gradients vanishing (or exploding) problem by leveraging on the gated structure of neurons. LSTM sets records for the prediction tasks based on time-series data, whose long and short term relationships between moments are particularly important.

The structural differences between vanilla RNN and LSTM are shown in Fig. 1.6. Unfolded RNN has one hidden layer to receive time-series data. The hidden state h_t at timestamp t only transmits to the neighboring neuron. There are weights matrices signifying the transitions W_x^t from input x_t to the hidden state at moment t, W_{t-1}^t from hidden state at moment $t-1$ to the hidden state at moment t, and W_t^y from the hidden state at moment t to output y_t. It shoud be noted that, as there are lots of connections in LSTM, the weights matrices are not annotated in the figure.

1.3.3.1 Vanilla RNN Inference

Using a hyperbolic tangent activation function $tanh$, and assume the x_t and h_t are column vectors with the same size[5], the state update can be formulated as

$$y_t := f(x_t, h_t) := W_t^y \cdot h_t, \;\; h_t := tanh(W_x^t x_t + W_{t-1}^t h_{t-1}).$$

Three weights matrices are the trainable parameters of vanilla RNN model that memories the state of vanilla, which are trained via ground-truth and gradient back-propagation algorithm as MLP. Gradient vanishing problem occurs when the sequence is long enough, that the gradients propagate to very early inputs are recurrently multiplied to small values till diminishing, or it may propagate to exploding values that disable the entire network.

1.3.3.2 State Updating in LSTM

Besides the $tanh$ (T in Fig. 1.6) activation that output values ranged from -1 to 1, LSTM also implements sigmoid activation function *sigmoid* (S in Fig. 1.6). As depicted in Fig. 1.6, the output for forget gate is $O_f = c_{t-1} \odot f_t$, where c_{t-1} is the cell state at moment $t-1$, $\odot$ signifies the Hadamard product, and $f_t = sigmoid(W_x^f x_t + W_f h_{t-1}) \in (0,1)$. It is obvious that if $f_t \to 1$, then $O_f \approx c_{t-1}$ indicating almost nothing is forgot; if $f_t \to 0$, then $O_f \approx 0$ indicating everything is almost forgot. For the input gate, state $i_t = sigmoid(W_x^i x_t + W_i h_{t-1})$, and new information $\tilde{c}_t = tanh(W_x^c x_t + W_c h_{t-1})$, then the cell update $c_t = O_i + O_f$, where $O_i = i_t \odot \tilde{c}_t$. Since each element of

[5]The same assumption for the variables c, h, x, y etc. except the weight matrix W.

$\tilde{c}_t$ can be negative, positive or zero, O_i signifies a residual to update the previous cell state c_t. The output gate has two tasks outputting the cell state c_t and hidden state h_t to the moment at $t+1$, as well as formulating hidden state h_t as the cell output y_t for training. The hidden state is formulated as $h_t = o_t \odot tanh(c_t)$, where $o_t = sigmoid(W_x^o x_t + W_o h_{t-1})$. h_t can be viewed as part of cell state c_t being exposed to the exterior, while hiding the other parts of cell state. Cho et al. proposed a variant of LSTM, which is known as the gated recurrent unit (GRU) that exposes the entire cell state to the exterior. Both LSTM and GRU outperform vanilla RNN in the speech translation task.

1.4 Introduction to Convolutional Neural Network

MLP and RNN try to mimic the brain neurons in different topologies, while CNN aims at simulating animals' visual cortices. Different from other categories of artificial neural networks introduced so far, CNN has a shorter history of development due to insufficient knowledge upon human's cone cells and the difficulties in training. Though CNN belongs to the category of FFN, it has gained huge popularity in the field of computer vision and pattern recognition on both 1D and multi-dimensional signals.

1.4.1 Milestones

Understanding the Visual Receptive Field The modeling of animals' receptive field (72; 73) indicates facts that one visual neuron is activated only to a region of visual space and visual cells are sensitive to the orientations of straight edges within one receptive field, while being insensitive to spacial locations of the edges. The idea that visual neurons can be individually activated by local visual stimuli, along with the spacial invariant property establishes the foundations of CNN.

Neocognitron The first prototype of CNN is named after 'neocognitron' (44), which is inspired by Hubel & Wiesel's works. In this work, Fukushima & Miyake construct a layer-by-layer computational model to model the visual cortex. Neocognitron is a hierarchy model that takes single circular-shaped receptive field of an image as its input, then connects to its higher-level recep-

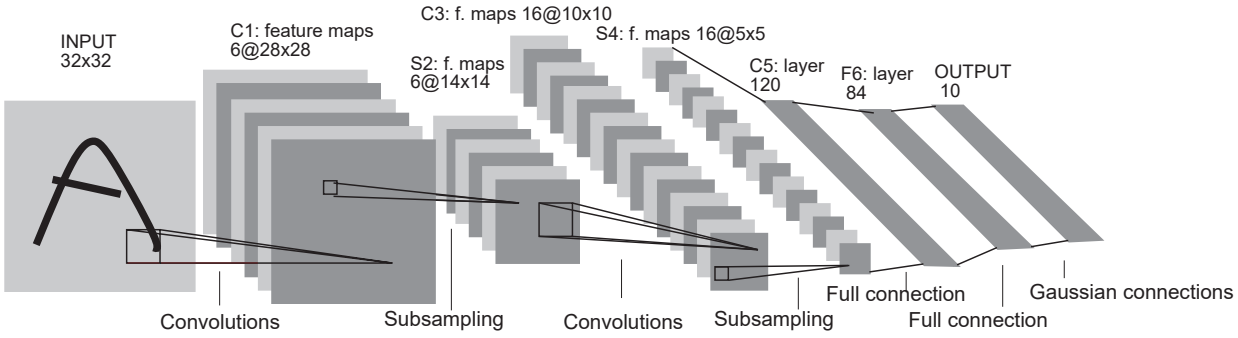


Figure 1.7: LeNet-5 (92) architecture for handwriting recognition.

tive fields in the following layers. The design of neocognitron that receptive fields are sparsely correlated evolves to the modern convolutional neural network architecture.

Achieve Shifting Invariance Waibel et al. introduced the time delay neural network (TDNN) whose motivation is to automatically classify 1-D phoneme in speech recognition. This work firstly achieved shift-invariance via removing uncorrelated dependencies in gradients back-propagation training and convolution. Hampshire & Waibel expanded the vanilla TDNN to 2-D case that can process images, which inspires the principles for modern CNN. However, their proposed neural networks are trained via Bayesian maximum-a-posteriori (MAP) learning (127) other than the gradients back-propagation.

Modern Convolutional Neural Network In 1989, LeCun et al. proposed a widely-known 5-layered CNN (see Fig. 1.7), namely LeNet-5, that solves the training problem for 2D image input using gradients back-propagation. Training of LeNet-5 is achieved by minimizing a global loss function using gradient-based approaches. Another essential contribution of this work is the training for multi-layer CNN. Each layer of LeNet can be interpreted as a graph transformer (GT) module that gets graph/graphs input and output a graph. Cascade GT modules construct a hierarchical GT network (GTN) that communicates states and gradients in the form of directed graphs between modules. It is also noteworthy that its gradients back-prorogation learning makes three approximations to drop the off-diagonal gradients, ensures that second derivatives are non-negative,

and calculates an average of gradients on a small subset instead of the entire training set. As for the architecture of LeNet-5, its backbone network consists of 4 convolutional layers, and its head network consists of 2 fully-connected layers and 1 RBF layer. This backbone-head architecture features the modern architecture of CNN.

1.4.2 From Time-Series to Image Filtering

1.4.2.1 Cross-Correlation and Convolution

Concepts of cross-correlation and convolution are derived from the time-series analysis of signal processing (134). Given two continuous time-series 1D signals $x(t)$ and $w(t)$ in the real number field, the cross-correlation between two signals is defined as

$$(x \circ w)(\tau) = \int_{-\infty}^{+\infty} x(t)w(t+\tau)dt, \tag{1.13}$$

that is, given an offset $\tau \in \mathbb{R}$, $w(t)$ is firstly shifted τ moments leftwards if $\tau > 0$, or rightwards if $\tau < 0$, then multiplies $x(t)$ across the time domain, followed by the integration. The convolution between $x(t)$ and $w(t)$ is defined as

$$\begin{aligned} (x * w)(t) &= \int_{-\infty}^{+\infty} x(\tau)w(t-\tau)d\tau, \ \ or \\ (x * w)(\tau) &= \int_{-\infty}^{+\infty} x(t)w(-t+\tau)dt. \end{aligned} \tag{1.14}$$

Relate Eqn. 1.14 to Eqn. 1.13, at the moment τ, convolution firstly flips the signal $w(t)$ to $w(-t)$ then performs cross-correlation with respect to the offset τ. For the discrete times-series signals $x[t]$ and $w[t]$, the cross-correlation and convolution are defined as

$$(x \circ w)(\tau) = \sum_{t=-\infty}^{+\infty} x[t]w[t+\tau], \qquad (x * w)(t) = \sum_{\tau=-\infty}^{+\infty} x[\tau]w[t-\tau]. \tag{1.15}$$

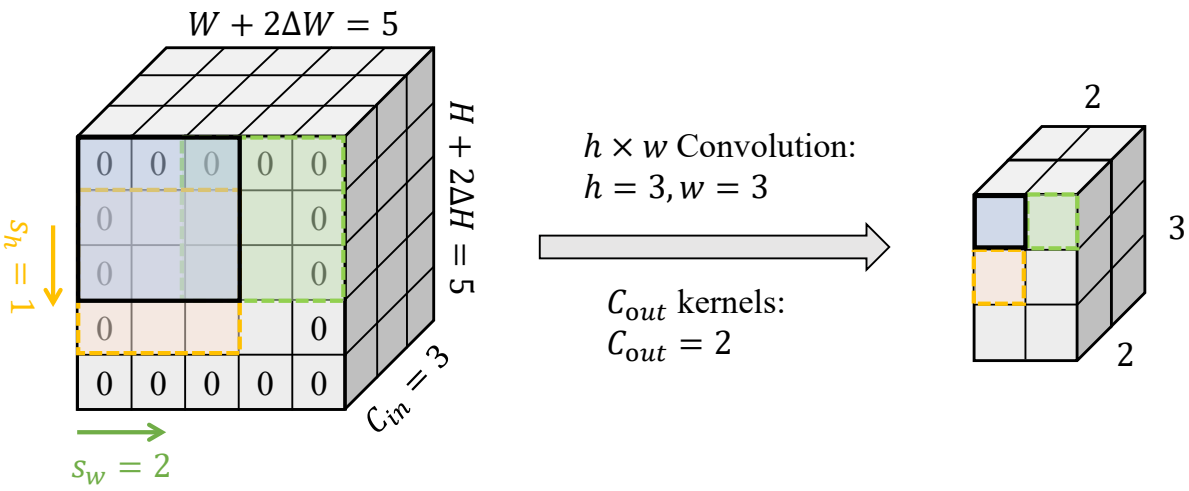


Figure 1.8: One vanilla convolutional layer.

1.4.2.2 2D Discrete Convolution

An image can be viewed as the discrete form of 2D signals whose integral coordinates (i, j) of pixels indicate the moments along height-wise and width-wise dimensions. The 1D-case of Eqn. 1.14 can be promoted to 2D-case (or higher dimension), i.e.,

$$(x * w)(i, j) = \sum_{\tau_1=-\infty}^{+\infty} \sum_{\tau_2=-\infty}^{+\infty} x[\tau_1, \tau_2] w[i - \tau_1, j - \tau_2]. \tag{1.16}$$

Eqn. 1.16 demonstrates the convolution in a row-wise style. The convolution can also be performed column-wise by exchanging the variables τ_1 and τ_2. Here $x(t)$ signifies an input image, and $w(t)$ is known as the filtering kernel for a image filter. Similar to the 1D case, the filtering kernel is flipped as $w(-i, -j)$ first at the moment (τ_1, τ_2), which can be described as rotation by 180°. Also, 0s can be assigned to indexes outside the boundaries of a fixed-sized filtering kernel.

1.4.3 Vanilla Convolutional Network

The convolutional layers of LeNet (92) are widely implemented in state-of-the art CNN-based approaches. A 32×32 Input image is down-sampled by convolutions to smaller feature volumes with multi-channel feature maps. In 1990, Yamaguchi et al. implemented max-pooling as a down-

sampling technique for TDNN, which becomes popular to CNN implementations. To illustrate the changes in feature sizes (see Fig. 1.8), assume the size of input feature map[6] to a vanilla convolution layer is $H \times W \times C_{in}$, the zero-padding length is ΔH and ΔW such that the size of padded image is $W_{in} \times H_{in} = (H + 2\Delta H) \times (W + 2\Delta W)$ and the filter-size of convolutional kernel is $h \times w$. Assume there are C_{out} sets of convolutional kernels in the vanilla convolutional layer, theses kernels independently filter the input feature volume w/o mutual connections between any two kernels[7]. Moreover, assume the kernel slides across the feature plane with height-wise stride s_h and width-wise stride s_w. Eqn. 1.17 illustrates the calculation of sizes to the output feature volume.

$$H_{out} = \lfloor \frac{H + 2\Delta H - h}{s_h} \rfloor + 1, \qquad W_{out} = \lfloor \frac{W + 2\Delta W - w}{s_w} \rfloor + 1. \tag{1.17}$$

It should be noted that $w \leq W_{in}$ and $h \leq H_{in}$, the depth (or channels) of output feature volume is equivalent to C_{out}. CNN is trained via BP learning, different from the GTN-based gradients back-propagation (92). A numerical analysis for fully-convolutional neural network will be demonstrated in Chapter 2.

1.5 Hybrid Neural Network

Biological neurons can either be simulated via computational models such as CNN, or they can be simulated via physical simulation, or being simulated using both. A representative category of HNN is Spiking neural network (SNN, 116). SNN simulates the bioelectrical signals of brain cells, that each spike neuron is considered as a response function w.r.t. continuous timing. When a spiking neuron fires, the output a strong pulse, otherwise the output response is weak. This process can be modeled as feeding analogy signal to SNN, and then get boolean outputs, which can be implemented to logic devices (202). SNN has been extended to a deeper network (138), but not yet competitive against non-spiking neural network in image recognition tasks. Recently, novel HNNs

[6] An color image can be regarded as a feature volume with three RGB channels, while a greyscale can be regarded as a one-channel feature map.

[7] How convolutional kernels filter input feature volume will be illustrated in Chapter 2.

have been proposed to construct a physical neural network (210) or link to real biological nerve system such as the brain of a fruit fly (100).

Chapter 2

Methodology of Convolutional Neural Network

Abstract

This chapter is an overview of the fundamental theories and techniques of modern CNN. The basics of a convolutional layer with activation function and deconvolutional layer are introduced in Section. 2.1. BP learning algorithms for CNN without leveraging on graph models are discussed in Section 2.2. In terms of constructing a convolutional neural network, several crucial techniques are introduced, including loss functions (Section. 2.3), pooling (Section. 2.4), and normalization (Section. 2.5). Finally, the implementation procedures are illustrated in Section. 2.6, along with the techniques for training such as regularization, dropout layer, and optimizers. As the development of modern CNN evolves very fast, there will be more and more novel techniques to boost the performance of modern CNN.

2.1 Convolutional Layer and Deconvolutional Layer

2.1.1 Activation Function

In convention, as well as imitating the behavior of visual neurons, each CNN layer is followed by an activation layer according to MLP and RNN designs. This layer processes all features from a feature volume x and outputs a feature volume y exactly the same size as x. To avoid the classical gradient diminishing problem for multi-layer CNN using sigmoid or *tanh* activation functions, rectified linear unit (*ReLU*) (129) activation function with the form $y = ReLU(x) = \max(0,x)$ becomes popular for the deep CNN network. At beginning, *ReLU* is implemented for RBM, later

becomes popular for CNN for its simplicity and easy implementation. Its non-linearity ensures that multi-layer CNN acts different from a single-layer CNN. The fact that when input $x > 0$, the derivative of $ReLU(x)$ is always 1 such that it populates gradients without loss, and this feature of *ReLU* is particularly suitable for ultra-deep CNN designs. However, it implies another problem that if the gradients are large enough in layers, the gradient back-propagated to shallow hidden layers would be extremely large, that is, the problem of gradient exploding.

One may wonder what's the best activation function layer for an given task using CNN. Unfortunately, the performance of activation functions tends to be unpredictable, and it largely depends on the input data and the model itself. There are varieties of activation functions since the proposal of *ReLU*, for instance, leaky rectified linear unit (*Leaky ReLU*, 115) that alleviates the 'dead neuron' problem caused by the negative inputs to *ReLU*, Gaussian error linear unit (*GELU*, 192) that smooths the rigid *ReLU* function, and *ReLU*6 (68) that adds a constant upper-bound 6 to *ReLU* when $x \geq 6$ to control the output etc. To be consistent with the notations in Chapter 1, the activation function layer is represented by $\sigma(x)$.

2.1.2 Deconvolution or Transposed Convolution?

2.1.2.1 Deconvolutional Layer

Deconvolution is a classical problem in signal processing. The task of deconvolution is to inverse the effect caused by a convolutional kernel, which is also called image restoration (22). Wiener filtering is widely implemented to remove known types of noises in images such as white noise and Gaussian blurring. Also, deconvolution can be estimated using inverse Fourier transform from frequency domain to spacial domain. The basic perspective of deconvolution is that, it reconstruct the original image. This idea is further implemented as a deconvolutional layer (228; 227), but not in the sense of classical image filtering. In this work, the authors implement a pipeline that has the potential to be fitted to a CNN by minimizing the reconstruction error of input image. One deconvolutional layer introduces sparse pooling and unpooling techniques, and multiple deconvolutional layers can be stacked to a network, which is then trained using gradients back-propagation. The

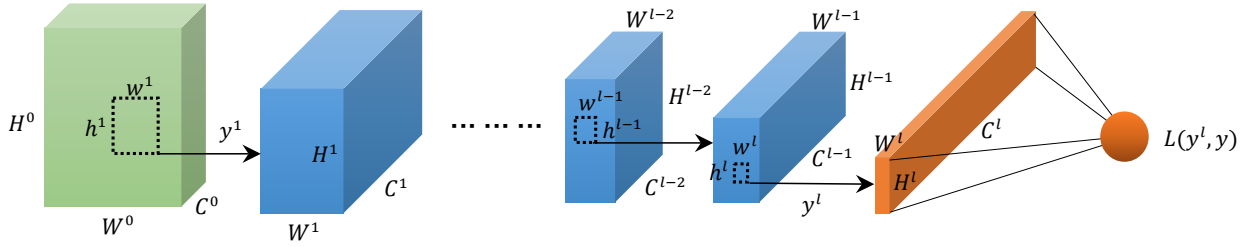


Figure 2.1: A $(l+1)$-layers convolutional neural network framework.

name 'deconvolutional layer' becomes widely known among computer vision community (173) although it is not mathematically strict in definition.

2.1.2.2 Transposed Convolutional Layer

Dumoulin & Visin argued that the 'deconvolutional layer' has already been defined as the 'inverse of convolution' such that it cannot be redefined, and they defined the 'transposed convolution' instead. Shi et al. showed that the transposed convolution layer can achieve the same result as of the fractional convolution layer (147). Transposed convolution is not the convolution with a transposed kernel, but firstly padding zeros to the input feature, then performing vanilla convolution using a fixed-sized convolutional kernel. More details of the calculations can be found in the guide (38) for more details in the calculations. It should be noted that deconvolutional layer is used for up-sampling the feature map to desired sizes. This indicates the capability of CNN in learning missing information in low-resolution images by minimizing the errors between reconstructed feature map and its groundtruth.

2.2 Gradients Back-propagation with Activation Functions

In this section, we firstly establish a simple convolutional neural network to illustrate the variables, and the relevance between inputs and outputs. Then the gradients back-propagation algorithms for hidden layers is demonstrated in mathematical notations to explore the possibilities to train the built CNN from scratch. As there are varieties of convolutional networks, this section only focuses

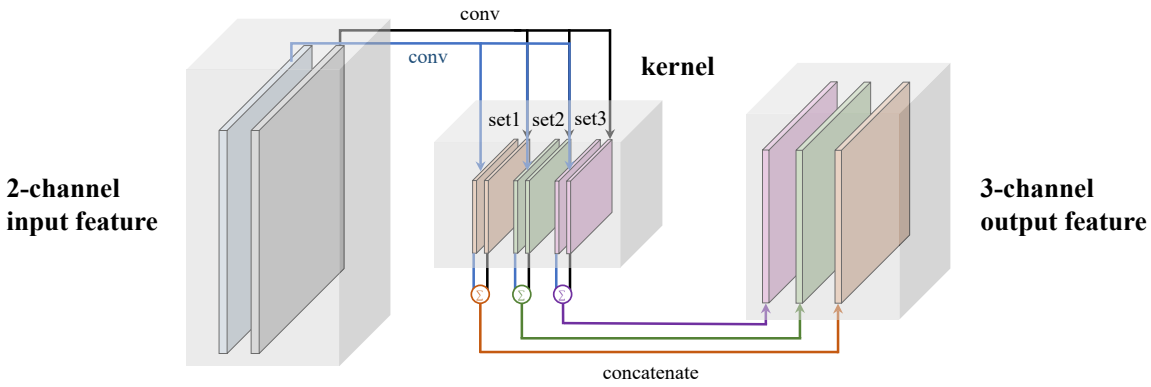


Figure 2.2: An example of one convolutional layer with a 3-set kernel activation.

on the vanilla CNN described in Chapter 1.

2.2.1 Model and Constraints

Lets consider a $(l+1)$-layers CNN model as shown in Fig. 2.1 with each hidden convolutional layer followed by an activation layer[1]. The l blue blocks signify all $l'-th,\ 1 \leq l' \leq l$ feature volumes, each feature volume is the output of one hidden convolutional layer. The orange blocks indicate the $l'-th$ $(l'=l+1)$ output layer that computes loss w.r.t. groundtruth as a scalar $L(y^l, y)$. Assume all activation function layers are leveraged on the same activation function $\sigma(\cdot)$, i.e., output of $l'-th$ hidden layer $y^{l'} = \sigma(u^{l'})$ where $u^{l'}$ is the intermediate results input to its following activation layer. y in the loss function signifies the groundtruth.

2.2.2 Gradients to the Last Hidden Layer

2.2.2.1 Forward Pass

For the $l-th$ hidden layer, we know that, $y^l = \sigma(u^l),\ u^l = \bigcup_{k=1}^{C^l} O_1^l[k]$, which indicates that the intermediate output u^l of the last hidden layer is the channel-wise concatenation of C^l filtering results by C^l sets (or groups) of convolutional kernels with the same size of $h^l \times w^l$. Concatenated in this way, feature volume u^l has C^l channels. Moreover, $O_1^l[k] = \sum_{s=1}^{C^{l-1}} O_2^l[s,k]$, which signifies

[1]The activation function layer is not annotated Fig. 2.1 since it can be combined into a vanilla convolutional layer

that the output of $k-th$ set of convolutional kernels (refer to Fig. 2.2) is the element-wise addition of the outputs O_2^l using C^{l-1} filters[2] in that set. O_2^l is formulated as

$$O_2^l[s,k] = \omega_{sk}^l[h^l, w^l] \circledast y_{sk}^{l-1}[H^{l-1}, W^{l-1}] + b_{sk}^l[H^l, W^l], \tag{2.1}$$

where b_{sk}^l is the bias to each kernel with a size of $H^l \times W^l$, '$\circledast$' operator signifies the convolution (refer to Eqn. 1.16) without flipping. Organize the variables and Eqn. 2.1, Eqn. 2.2 denotes the relation between convolution and y^l:

$$y^l = \sigma(u^l) = \sigma\left(\bigcup_{k=1}^{C^l}\sum_{s=1}^{C^{l-1}}\left(\omega_{sk}^l[h^l, w^l] \circledast y_{sk}^{l-1}[H^{l-1}, W^{l-1}] + b_{sk}^l[H^l, W^l]\right)\right). \tag{2.2}$$

What we are interested in is the computational form of O_2^l. Refer to Eqn. 1.16, firstly, set a feature coordinate system whose origin locates at the very top-left corner, the Y-axis stretches along width-wise direction, while the X-axis stretches along height-wise direction. If both h^l and w^l are odd numbers[3], we let $m = \frac{h^l-1}{2},\ n = \frac{w^l-1}{2}$. Also, if the local coordinate system of a convolutional kernel takes its center as the local origin, and its increasing directions of axis are consistent with feature coordinate system, then the weights of the convolutional kernel can be traversed via local coordinates. In addition, (p,q) can also denote the offsets relative to center $(0,0)$. With these notations, it is obvious that, during the convolution process, the image of kernel center in feature coordinate system is $(m+iSH_{sk}^l, n+jSW_{sk}^l)$, where SH_{sk}^l and SW_{sk}^l signify the strides along X- and Y- axis, i,j are integers within the ranges $[0, H^l-1],\ [0, W^l-1]$. Now we can formulate the convolution[4] as

$$\omega_{sk}^l \circledast y_{sk}^{l-1} = \bigcup_{i=0}^{H^l-1}\bigcup_{j=0}^{W^l-1}\left[\sum_{p=-m}^{m}\sum_{q=-n}^{n}\omega_{sk}^l(p,q) \bullet y_{sk}^{l-1}(p+m+iSH_{sk}^l, q+n+jSW_{sk}^l)\right]. \tag{2.3}$$

[2]Each convolutional filter convolves one channel of feature map of the input $H^{l-1} \times W^{l-1} \times C^{l-1}$ feature volume. Therefore, there are totally C^{l-1} convolutional kernels in each set.

[3]Which is natural since odd kernel sizes are usually implemented, otherwise using the halves of h^l and w^l.

[4]This only performs the cross-correlation step of Eqn. 1.16 since we assume the kernel ω_{sk}^l has been transposed beforehand. The $\bigcup$ operator also signifies height-wise or width-wise concatenation here.

If h^l and w^l are both even numbers, let $m = \frac{h^l}{2},\ n = \frac{w^l}{2}$. In this case, the center of convolutional kernel is ambiguous since local center (0,0) doesn't exist. However, we can still leverage on the relative positions for calculations. Rethink the Eqn. 2.3, if we move the origin of local coordinate system to the top-left corner of kernel, the convolution can be formulated as

$$\omega_{sk}^l \circledast y_{sk}^{l-1} = \bigcup_{i=0}^{H^l-1} \bigcup_{j=0}^{W^l-1} \left[\sum_{p=0}^{h^l-1} \sum_{q=0}^{w^l-1} \omega_{sk}^l(p,q) \cdot y_{sk}^{l-1}(p+iSH_{sk}^l, q+jSW_{sk}^l) \right]. \tag{2.4}$$

Eqn. 2.4 can be applied to both the cases when $h^l,\ w^l$ are odd or even. Take Eqn. 2.4 into the expression of O_2^l (Eqn. 2.1), this output can be formulated as

$$O_2^l[s,k] = \bigcup_{i=0}^{H^l-1} \bigcup_{j=0}^{W^l-1} \left[\sum_{p=0}^{h^l-1} \sum_{q=0}^{w^l-1} \omega_{sk}^l(p,q) \cdot y_{sk}^{l-1}(p+iSH_{sk}^l, q+jSW_{sk}^l) \right] + b_{sk}^l[H^l, W^l]. \tag{2.5}$$

After the modeling of one convolutional layer (with an activation function layer), the next step is to calculated the gradients back-propagation in that layer.

2.2.2.2 Gradients Back-Propagation

Similar to MLP, we can also apply BP algorithm (refer to Alg. 2) for updating the weights of ω_{sk}^l iteratively by

$$\omega_{sk}^l = \omega_{sk}^l - \eta \Delta \omega_{sk}^l, \tag{2.6}$$

where η is the learning rate, and $\Delta\omega_{sk}^l$ indicates the gradients back-propagated from loss function to the kernel. The key is then to compute the mathematic form of $\Delta\omega_{sk}^l$. According to the chain rule, we know

$$\Delta\omega_{sk}^l(p,q) = \sum \frac{\partial L}{\partial \omega_{sk}^l(p,q)} = \sum \left[\frac{\partial L}{\partial y^l} \odot \frac{\partial y^l}{\partial u^l} \odot \frac{\partial u^l}{\partial O_1^l[k]} \odot \frac{\partial O_1^l[k]}{\partial O_2^l[s,k]} \odot \frac{\partial O_2^l[s,k]}{\partial \omega_{sk}^l(p,q)} \right], \tag{2.7}$$

where the summation operator denotes the gradients propagated to $w_{sk}^l(p,q)$ equals to the total contribution of all related terms, and the Hadamard product '$\odot$' indicates the contribution is only

relates to the corresponding position between feature volumes.

Consider each term of Eqn. 2.7, $\frac{\partial L}{\partial y^l}$ has a size of $H^l \times W^l \times C^l$, $\frac{\partial y^l}{\partial u^l}$ has a size of $H^l \times W^l \times C^l$. $\frac{\partial u^l}{\partial O_1^l[k]}$ only related to $k-th$ output of O_1^l, which has a size of $H^l \times W^l \times C^l$ that all channels except $k-th$ channel are filled with zeros, while the $k-th$ channel of feature map is filled with ones. This indicates the transition of errors from the loss to the $k-th$ feature map. $\frac{\partial O_1^l[k]}{\partial O_2^l[s,k]}$ indicates the error transition from $k-th$ feature map to the filtering output of $s-th$ kernel, which has a valid size of $H^l \times W^l \times 1$, and being filled with ones. Thus, we can expand it to the same $H^l \times W^l \times C^l$ gradients volume as of $\frac{\partial u^l}{\partial O_1^l[k]}$. The last term $\frac{\partial O_2^l[s,k]}{\partial \omega_{sk}^l(p,q)}$ is directly relevant to ω_{sk}^l. In the following part, we are going to deduce a computational form of this derivative.

According to Eqn. 2.5, it is obvious that

$$\frac{\partial O_2^l[s,k]}{\partial \omega_{sk}^l(p,q)} = \bigcup_{i=0}^{H^l-1} \bigcup_{j=0}^{W^l-1} y_{sk}^{l-1}\left(p+iSH_{sk}^l, q+jSW_{sk}^l\right), \tag{2.8}$$

in which $0 \le p \le h^l - 1,\ 0 \le q \le w^l - 1$, and the size of its gradient matrix is $H^l \times W^l \times 1$. Eqn. 2.8 can also be extended to $H^l \times W^l \times C^l$ by padding zeros. Therefore, Eqn. 2.7 can be expressed as

$$\Delta\omega_{sk}^l(p,q) = \sum \left[\frac{\partial L}{\partial y^l} \odot \sigma'(u^l)\Big|_{sk} \odot \bigcup_{i=0}^{H^l-1} \bigcup_{j=0}^{W^l-1} y_{sk}^{l-1}\left(p+iSH_{sk}^l, q+jSW_{sk}^l\right) \right]_{H^l \times W^l}. \tag{2.9}$$

Let $\delta_{sk}^l = \frac{\partial L}{\partial y^l} \odot \sigma'(u^l)\Big|_{sk}$ denote the gradient matrix transmitted to the $s-th$ filtering output,

$$\begin{aligned}
\Delta\omega_{sk}^l(p,q) &= \sum \left[\delta_{sk}^l \odot \bigcup_{i=0}^{H^l-1} \bigcup_{j=0}^{W^l-1} y_{sk}^{l-1}\left(p+iSH_{sk}^l, q+jSW_{sk}^l\right) \right]_{H^l \times W^l} \\
&= \sum \left[\bigcup_{i=0}^{H^l-1} \bigcup_{j=0}^{W^l-1} \left[\delta_{sk}^l(i,j) \cdot y_{sk}^{l-1}\left(p+iSH_{sk}^l, q+jSW_{sk}^l\right) \right] \right]_{H^l \times W^l} \\
&= \sum_{i=0}^{H^l-1} \sum_{j=0}^{W^l-1} \left[\delta_{sk}^l(i,j) \cdot y_{sk}^{l-1}\left(p+iSH_{sk}^l, q+jSW_{sk}^l\right) \right],
\end{aligned} \tag{2.10}$$

The gradient computation for the entire kernel is then formulated as

$$\Delta\omega_{sk}^{l}[h^{l},w^{l}]=\bigcup_{p=0}^{h^{l}-1}\bigcup_{q=0}^{w^{l}-1}\sum_{i=0}^{H^{l}-1}\sum_{j=0}^{W^{l}-1}\left[\delta_{sk}^{l}(i,j)\cdot y_{sk}^{l-1}\left(p+iSH_{sk}^{l},q+jSW_{sk}^{l}\right)\right]. \tag{2.11}$$

Eqn. 2.10 is the computational form for $\Delta\omega_{sk}^{l}(p,q)$, it can be inferred from the conclusion that each position of kernel is related to multiple positions depending on how the kernel moves across a feature map. It is the underlying reason for single convolutional kernel to be sparsely connected to a feature map. This also expands the receptive field that is explicitly considered as the entire region of the convolutional kernel. Take a 2-D cross-correlation into consideration, Eqn. 2.11 can be further simplified as Corollary. 2.2.0.1:

Corollary 2.2.0.1 (Strided Convolution)**.** *Given a 2D feature map $y\in\mathbb{R}^{H\times W}$, 2D convolutional kernel $\omega\in\mathbb{R}^{h\times w}$, the gradients matrix $\delta\in\mathbb{R}^{H\times W}$ back propagated to y. The gradients $\Delta\omega$ back-propagated to ω can be defined as a strided convolution*

$$\Delta\omega=\delta\circledcirc y=\bigcup_{p=0}^{h-1}\bigcup_{q=0}^{w-1}\sum_{i=0}^{H-1}\sum_{j=0}^{W-1}\left[\delta(i,j)\cdot y\left(p+is_{h},q+js_{w}\right)\right], \tag{2.12}$$

where s_h and s_w are strides along height-wise and width-wise directions respectively. The Constraints of models, local and feature coordinate systems, operators are consistent with the definitions in Section. 2.2.2.

Noted that strided convolution is not the same 2D convolution (refer to Eqn. 1.16) in the context of signal processing due to the strides and non-transposed kernel. It's easier to compute the gradients to bias matrix $b_{sk}^{l}[H^{l},W^{l}]$, which is a $H^{l}\times W^{l}$ matrix equals to δ_{sk}^{l} according to Eqn. 2.2.

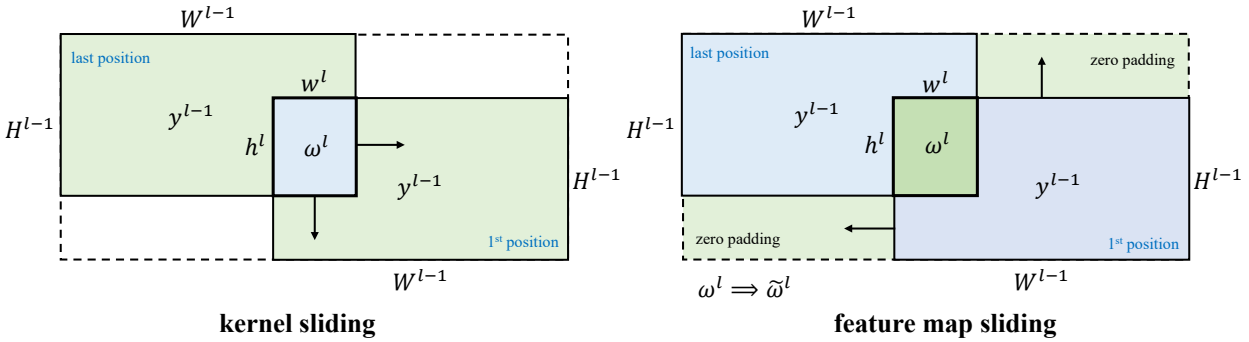


Figure 2.3: Sliding kernel (left) and fixed kernel (right) at meantime.
The figures only show the case when height-wise and width-wise strides (SH^l, SW^l) are divisible by $(H^{l-1}-h^l, W^{l-1}-w^l)$ respectively.

2.2.3 Another Perspective to Convolution

Convolution can be viewed in another perspective via relative motion. To illustrate this concept, reconsider the cross-correlation term in Eqn. 2.4 as

$$(\omega^l_{sk} \circ y^{l-1}_{sk})(i,j) = \sum_{p=0}^{h^l-1} \sum_{q=0}^{w^l-1} \omega^l_{sk}(p,q) \cdot y^{l-1}_{sk}(p+iSH^l_{sk}, q+jSW^l_{sk}). \tag{2.13}$$

(i,j) indicates the $i-th$ height-wise position and $j-th$ width-wise position of sliding kernel w^l_{sk} on fixed feature map y^{l-1}_{sk}. The range of i,j depends on the sizes of both:

$$0 \le i \le \lfloor \frac{H^{l-1}-h^l}{SH^l_{sk}} \rfloor = H^l - 1, \qquad 0 \le j \le \lfloor \frac{W^{l-1}-w^l}{SW^l_{sk}} \rfloor = W^l - 1.$$

This is to say, as shown in Fig. 2.3 (left image), feature map y^l_{sk} is fixed and the convolutional kernel slides from the initial position (i,j) (the lower-right y^{l-1} in left image), and ends at the last position as depicted as the upper-left y^{l-1}. This process can also be viewed as an reversed sliding of y^{l-1} along padded ω^l_{sk} (denoted as ω^l in the figure). For the former case, we deduce the Eqn. 2.8, but it is applicable to calculate the gradients w.r.t. y^{l-1}_{sk} in Eqn. 2.13. However, the latter,

or the equivalent form can be described as:

$$(y_{sk}^{l-1} \circ \omega_{sk}^{l})(i,j) = \sum_{p=0}^{H^{l-1}-1} \sum_{q=0}^{W^{l-1}-1} y_{sk}^{l-1}(p,q) \cdot \tilde{\omega}_{sk}^{l}(p + \tilde{h}_0^l - iSH_{sk}^l, q + \tilde{w}_0^l - jSW_{sk}^l), \tag{2.14}$$

in which $\tilde{\omega}_{sk}^{l}$ indicates the entire padded map (Fig. 2.3, right image), the coordinates are computed by transition. The first element of original ω_{sk}^{l} in feature coordinate system of $\tilde{\omega}_{sk}^{l}$ is $(\tilde{h}_0^l, \tilde{w}_0^l) = ((H^l - 1) \cdot SH_{sk}^l, (W^l - 1) \cdot SW_{sk}^l)$, and the 'large kernel' moves leftwards and upwards towards the origin of feature coordinate system with strides SH_{sk}^l and SW_{sk}^l.

2.2.4 Gradients Between Hidden Layers

In this section, the gradients back-propagation between hidden layers are presented in mathematical notations. The constraints are the same as those in section. 2.2.2. Assume we've known the gradients volume $\delta^l = \left\{ \delta_k^l | 1 \leq k \leq C^l \right\}$ back-propagated to the $l-th$ layer, and we know that δ_k^l is propagated to δ_{sk}^l with the same valid values whereas different in sizes. If we can deduce the relationship between δ_{sk}^l and δ_{sk}^{l-1}, then the gradients transmitted to the $(l-1)-th$ layer become more explicit. Recall the definition $\delta_{sk}^l = \frac{\partial L}{\partial y^l} \odot \sigma'(u^l)\Big|_{sk}$, likewise, $\delta_{sk}^{l-1} = \frac{\partial L}{\partial y^{l-1}} \odot \sigma'(u^{l-1})\Big|_{sk}$. Apply chain rule to δ_{sk}^l, that

$$\begin{aligned} \delta_{sk}^{l-1} &= \frac{\partial L}{\partial y^l} \odot \frac{\partial y^l}{\partial u^l} \odot \frac{\partial u^l}{\partial O_1^l[k]} \odot \frac{\partial O_1^l[k]}{\partial O_2^l[s,k]} \odot \frac{\partial O_2^l[s,k]}{\partial y^{l-1}} \odot \sigma'(u^{l-1})\Big|_{sk}, \\ &= \delta_{sk}^l \odot \frac{\partial O_2^l[s,k]}{\partial y_{sk}^{l-1}} \odot \sigma'(u_k^{l-1}) \end{aligned} \tag{2.15}$$

$O_2^l[s,k]$ is calculated according to Eqn. 2.1, 2.4 and 2.14, and

$$O_2^l[s,k] = \omega_{sk}^l \circledast y_{sk}^{l-1} + b_{sk}^l = \bigcup_{i=0}^{H^l-1} \bigcup_{j=0}^{W^l-1} (y_{sk}^{l-1} \circ \omega_{sk}^l)(i,j) + b_{sk}^l. \tag{2.16}$$

It is obvious that we can utilize Eqn. 2.14 and 2.16 to compute the term

$$\frac{\partial O_2^l[s,k]}{\partial y_{sk}^{l-1}(p,q)} = \bigcup_{i=0}^{H^l-1} \bigcup_{j=0}^{W^l-1} \tilde{\omega}_{sk}^l(p+\tilde{h}_0^l - iSH_{sk}^l, q+\tilde{w}_0^l - jSW_{sk}^l). \tag{2.17}$$

According to Eqn. 2.17, the sizes of both δ_{sk}^l and O_2^l are $H^l \times W^l \times 1$, which is not consistent with the size of y_{sk}^l and the following term $\sigma'(u_k^{l-1})$, as well as the output δ_{sk}^{l-1}. Thus, there is a shape transformation during the calculation phase of $\delta_{sk}^l \odot \frac{\partial O_2^l[s,k]}{\partial y_{sk}^{l-1}}$. First, calculate $\delta_{sk}^l \odot \frac{\partial O_2^l[s,k]}{\partial y_{sk}^{l-1}(p,q)}$ that indicates the propagation from δ_{sk}^l to $y_{sk}^l(p,q)$, next, sum up all contributions to form the overall contribution to the element $y_{sk}^l(p,q)$, the last step is to construct the overall contribution map from δ_{sk}^l to y_{sk}. With Eqn. 2.17, this procedure is formulated as:

$$\begin{aligned}
\delta_{sk}^l \odot \frac{\partial O_2^l[s,k]}{\partial y_{sk}^{l-1}} &= \bigcup_{p=0}^{H^{l-1}-1} \bigcup_{q=0}^{W^{l-1}-1} \sum \left[\bigcup_{i=0}^{H^l-1} \bigcup_{j=0}^{W^l-1} \delta_{sk}^l(i,j) \cdot \tilde{\omega}_{sk}^l(p+\tilde{h}_0^l - iSH_{sk}^l, q+\tilde{w}_0^l - jSW_{sk}^l) \right] \\
&= \bigcup_{p=0}^{H^{l-1}-1} \bigcup_{q=0}^{W^{l-1}-1} \left[\sum_{i=0}^{H^l-1} \sum_{j=0}^{W^l-1} \delta_{sk}^l(i,j) \cdot \tilde{\omega}_{sk}^l(p+\tilde{h}_0^l - iSH_{sk}^l, q+\tilde{w}_0^l - jSW_{sk}^l) \right] \\
&:= \delta_{sk}^l \ominus \tilde{\omega}_{sk}^l.
\end{aligned} \tag{2.18}$$

The defined operator '$\ominus$' indicates an inverted strided convolution, which leads to the corollary of inverted strided convolution on below. Corollary. 2.2.0.2 can be promoted to shallower layers, which facilitates the computation of weights updating according to corollary. 2.2.0.1.

Corollary 2.2.0.2 (Inverted Strided Convolution)**.** *Given constraints of models, local and feature coordinate systems, operators are consistent with the definitions in section. 2.2.2 and 2.2.3. The relation of values between gradient matrices δ_{sk}^l and δ_{sk}^{l-1} can be formulated using the inverted stride convolution defined in section. 2.2.4 as*

$$\delta_{sk}^{l-1} = \delta_{sk}^l \ominus \tilde{\omega}_{sk}^l \odot \sigma'(u_k^{l-1}). \tag{2.19}$$

2.3 Loss Function

Loss function is a general concept to measure the error between target and outcome, thus can be various in its form and highly-customized. The target of loss function[5] for CNN is to minimize the pixel-wise loss between groundtruth and predicted feature map. Loss function can be implemented to binary (0-1) and multi-class classification problem. The most commonly implemented loss functions include but not limited to MSE, MAE (both are introduced in section. 1.2 for MLP), Kullback-Leibler divergence (KL-Divergence, 88), cross-entropy loss, focal loss (103), Huber loss (74), ranking loss (24) or alike contrastive loss(201), triplet loss (166), and margin loss (157) for metric learning.

2.3.1 KL-Divergence

KL-Divergence is also named after 'relative entropy'. The term entropy[6] signifies the uncertainty of a random variable defined in information theory. Assume a discrete random variable X with limited possible values, the probabilistic distribution is defined as $P(X = x_i) = p_i,\ 0 \leq p_i \leq 1$, and $1 \leq i \leq n$, then the entropy of X is defined as

$$H(X) = \begin{cases} -\sum_{i=1}^{n} p_i \log p_i, & p_i \neq 0 \\ 0, & p_i = 0 \end{cases}, \tag{2.20}$$

due to the irrelevance to the exact values of X, $H(X)$ merely depends on the distribution of X, thus in the following part of this section, we let $H(p)$ denotes $H(X)$.

2.3.1.1 Properties of Entropy

The first property of entropy is the range of $H(p)$: $0 \leq H(p) \leq \log n$.

[5] It can also be named after the term 'error function', or 'cost'.

[6] In physics, entropy is the measure for thermal energy and molecular randomness, which has a similar idea, but different definition from the entropy in informatics.

Proof. $\forall 0 \leq p_i \leq 1$, $-\log p_i \geq 0$, this indicates $H(p) \geq 0$. Also, according to Jensen's inequality, $H(p) = \sum_{i=1}^{n} p_i \log \frac{1}{p_i} \leq \log\left(\sum_{i=1}^{n} p_i \cdot \frac{1}{p_i}\right) = \log n$. Therefore, $0 \leq H(p) \leq \log n$. □

Binary entropy indicates X can either be 0 or 1, the distribution of X is written as $P(X=1) = p$, $P(X=0) = 1-p$, $0 \leq p \leq 1$. Then when $p = 0.5$, binary entropy achieves its global maximum if its base $a > 1$, and its global minimum if the base $0 < a < 1$.

Proof. For binary entropy, $H(p) = -p\log p - (1-p)\log(1-p)$. If $p = 0$, $H(p) = 0$, for the rest of $0 < p \leq 1$, $H(p)$ is continuous and differentiable. $H'(p) = \log(1-p) - \log p = \log \frac{1-p}{p}$. Let $H'(p) = 0$, then $p = 0.5$. Next, discuss if this point is a global maximum or minimum. We know the second derivative $H''(p) = \frac{1}{p(1-p)} \ln \frac{1}{a} < 0$ if $a > 1$, and $H''(p) > 0$ if $0 < a < 1$, therefore this property is proven. □

The third property is related to the second property and it describes how entropy measures uncertainty. As the base $a > 1$ is commonly implemented for binary entropy, that when $p = 0.5$ binary entropy reaches its global maximum. This indicates the randomness of X achieves its maximum, while for $p = 0$ or $p = 1$, the randomness drops to 0.

2.3.1.2 Relative Entropy

Given 1D true probability distributions $p(x)$ and another 1D predicted probability distribution $q(x)$, where $x \in \mathscr{X}$, $\mathscr{X}$ is the discrete probability space. KL-Divergence is a measure for evaluating the similarity between two distributions. This relative entropy is defined mathematically as

$$D_{KL}(p \parallel q) = \sum_{x \in \mathscr{X}} p(x) \log \frac{p(x)}{q(x)}. \tag{2.21}$$

Relative entropy has two important properties. The first one is $D_{KL}(p \parallel q) \geq 0$, and it can be proved via Jensen's inequality or Gibbs' inequality. Two proofs are presented on below.

Proof. (using Jensen's inequality) We know that log function is convex, so as to its linear combi-

nations, thus the conditions are satisfied for Jensen's inequality. We have

$$-D_{KL}(p \parallel q) = \sum_{x \in \mathscr{X}} p(x) \log \frac{q(x)}{p(x)} \leq \log \sum_{x \in \mathscr{X}} \left(p(x) \cdot \frac{q(x)}{p(x)} \right) = \log \left(\sum_{x \in \mathscr{X}} p(x) \right) = 0,$$

such that $D_{KL}(p \parallel q) \geq 0$. □

Proof. (using Gibbs' inequality) Gibbs' inequality states that, assume $P = \{p_1, \cdots, p_n\}$ is a discrete probability distribution, then for any other discrete probability distribution $Q = \{q_1, \cdots, p_n\}$, $-\sum_{i=1}^{n} p_i \log p_i \leq -\sum_{i=1}^{n} p_i \log q_i$, this would directly deduce the corollary $D_{KL}(p \parallel q) \geq 0$. □

The other important feature is that, $D_{KL}(p \parallel q)$ may not equal to $D_{KL}(q \parallel p)$.

2.3.2 Cross-Entropy Loss

Cross-entropy is defined as the summation of relative entropy and the entropy of true distribution:

$$H(p,q) = H(p) + D_{KL}(p \parallel q), \tag{2.22}$$

it is obvious from this definition that when p and q are the same distributions, $D_{KL}(p \parallel q) = 0$ such that $H(p,q)$ hits its minimum $H(p)$, which is the constant entropy of true distribution once p is defined. Therefore, minimizing cross-entropy is equivalent to minimize KL-Divergence between two distributions. Moreover, we can deduce the computational form of cross-entropy loss from Eqn. 2.22 that

$$CELoss(p,q) = -\sum_{x \in \mathscr{X}} p(x) \log q(x). \tag{2.23}$$

2.3.3 Focal Loss

Lin et al. put forward the concept of focal loss for 2D object detection task, which contributes to the competitive performance in their model than larger models. Focal loss aims at alleviating the class imbalance problem using cross-entropy loss. Consider a binary classification problem that $BCELoss(p,q) = -p_0 \log q_0 - p_1 \log q_1$.

If we implement one-hot encoding of groundtruth, e.g., for a 3-classes classification problem, the category zero is represented using a vector $[1,0,0]^T$, likewise, category one and two are represented as $[0,1,0]^T$ and $[0,0,1]^T$. The advantage of one-hot encoding is that the category a pixel belongs to is described as a simple probability distribution. In this case, the binary cross-entropy loss would be

$$BCELoss(p,q) = \begin{cases} -\log q_1 & , \arg\max(p) = 1 \\ -\log(1-q_1) & , \arg\max(p) = 0 \end{cases}, \tag{2.24}$$

the focal loss for Eqn. 2.24 is defined as

$$FLoss(p,q) = \begin{cases} -\alpha(1-q_1)^{\gamma}\log q_1 & , \arg\max(p) = 1 \\ -(1-\alpha)q_1^{\gamma}\log(1-q_1) & , \arg\max(p) = 0 \end{cases}, \tag{2.25}$$

where $0 < \alpha < 1,\ \gamma > 0$ are hyper-parameters that regulate the importance between positive and negative samples, hard (False Negatives) samples and easy (True Negative) samples, respectively.

For the multi-class classification problem (more than 2 categories) and one-hot encoding, however, (103) didn't show the multi-class focal loss. In our work (122), the following focal loss for multi-class classification is tested.

$$FLoss(p,q) = -\alpha_i(1-q_i)^{\gamma}\log q_i,\ i = \arg\max(p),\ \text{ and } \sum_i \alpha_i = 1. \tag{2.26}$$

2.3.4 Ranking Loss

Retrieval using deep learning techniques is known as the metric learning (87), which mainly leverages on a category of loss functions, i.e., the ranking loss. Distinguished from cross-entropy loss that evaluate the similarity between two probability distributions, ranking loss aims at calculating the relative distances between input images or vectors. There are various names for ranking loss in different scenarios, such as contrastive loss, margin loss, hinge loss for support vector machine (SVM, 29) etc. For instance, the triplet loss (166) is one of the most popular ranking loss, which

can be defined as the sum of all Euclidean distances:

$$TLoss = \sum max(0,\ \| E(A) - E(P) \|^2 - \| E(A) - E(N) \|^2 + \beta),$$

where $E(\cdot)$ signifies an embedding function that projects input to another regulated feature space, A signifies an anchor input, P the positive input of the same class as of the anchor, N the negative input belongs a different class. β is a margin between positive and negative inputs.

2.4 Pooling Layer

2.4.0.1 Max Pooling and Avergae Pooling

A category of crucial down-sampling techniques is known as pooling. Max-pooling is the earliest pooling technique introduced for the CNN architecture. Max-pooling process is the same to convolution except the cross-correlation stage, that max-pooling simply calculate the maximum value on the feature map within the region of 'sliding convolutional kernel' as the pooling result. In this way, the output of max-pooling has the same number of channels as its input, and the sizes calculated as the convolution (refer to Eqn. 1.17). Max-pooling can be formulated using notations in Section. 2.2 as the following $l-th$ layer after the feature volume y^{l-1} with a kernel ω^l_{mp},

$$max\_pooling(\omega^l_{mp}, y^{l-1}_k) = \bigcup_{i=0}^{H^l-1} \bigcup_{j=0}^{W^l-1} \left[\max_{p=0}^{h^l-1} \max_{q=0}^{w^l-1} y^{l-1}_k (p + iSH^l_{sk}, q + jSW^l_{sk}) \right]. \tag{2.27}$$

Similarly, when calculating the average over the region, average-pooling is formulated as:

$$avg\_pooling(\omega^l_{ap}, y^{l-1}_k) = \bigcup_{i=0}^{H^l-1} \bigcup_{j=0}^{W^l-1} \left[\frac{1}{h^l \bullet w^l} \sum_{p=0}^{h^l-1} \sum_{q=0}^{w^l-1} y^{l-1}_k (p + iSH^l_{sk}, q + jSW^l_{sk}) \right]. \tag{2.28}$$

Gradients back-propagated in max-pooling layer can be regarded as directly populating the gradients from the output to all activated positions. This process is sparse according to the $H^l \times W^l$ activated positions of total $H^{l-1} \times W^{l-1}$ positions. One significant defect of max-pooling is related

to this sparsity, that strong responses are overwhelming. While for the average-pooling, which can be viewed as a average filtering with a filter filled with $\frac{1}{h^l \cdot w^l}$, the back-propagated averaged gradients are related to more regions as a convolutional kernel. However, average-pooling tends to weaken strong responses, thus bringing more noise than max-pooling.

2.4.0.2 Global Pooling

Global max-pooling and global average-pooling (102) are more aggressive pooling approaches than the techniques mentioned above. These two techniques perform pooling over the entire feature map y_k^{l-1}. Usually, global pooling layer is set as the layer following the last hidden convolutional layer (e.g., after the $l-th$ layer). Compare to a fully-connected layer, global pooling layer has no trainable parameters, and being much simpler than the linear projection.

2.5 Normalization Layer

Normalization layer contributes greatly to the stability of training a neural network by reducing the internal covariate shift (76). The shift occurs in every intermediate feature volume, that the network tends to greatly change the distribution in each layer. However, we want to output a conditional probability distribution that is the same as the groundtruth. This is how the shift contradicts with our target. What normalization in neural networks does is to enforce the same variance and mean to the outputs of intermediate layers, which is similar to the widely-implemented whitening during pre-processing. For CNN, this technique is especially crucial. Normalization of feature volume can be performed along channels, or layer-wise, or simply element-wise. The following part of this section introduces several normalization techniques.

2.5.1 Batch Normalization

An example is shown in Fig. 2.4, given a mini-batch x consisted of 3 feature volumes x_1 (green), x_2 (blue), and x_3 (orange). Batch norm (76) has three steps. The first step is to calculate the batch

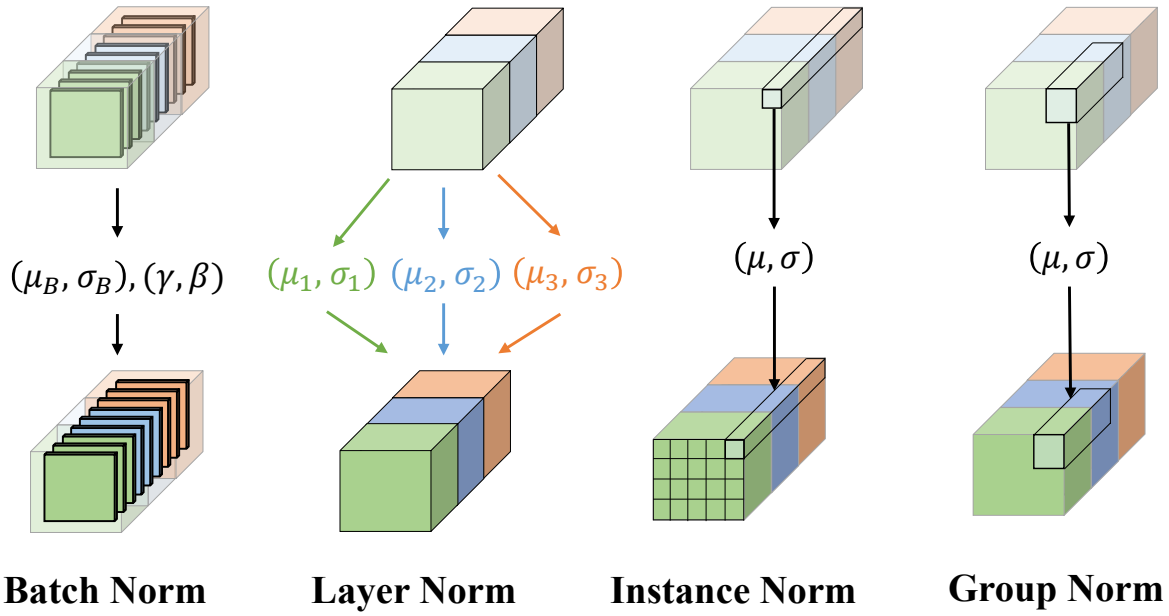


Figure 2.4: Visualization of batch norm, layer norm, instance norm and group norm.

mean and variance μ_B and σ_B as

$$\mu_B = \frac{1}{3}\sum_{i=1}^{3} x_i, \qquad \sigma_B = \frac{1}{3}\sum_{i=1}^{3}(x_i - \mu_B)^2.$$

Step 2 is to normalize the entire mini-batch with with the batch mean and variance as $\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$. The last step is to scale and shift normalized features as $y = \gamma \cdot \hat{x} + \beta$. ε is a small float to avoid zero denominator, scalars γ and β are trainable parameters. The authors argued that the scale and shift step slacks the rigid normalization by bringing in tuneable adjustments to variance and mean, which may enhance the performance. On the other hand, this also brings in uncertainties against normalization, which requires to be further studied. Another potential issue related to batch norm is related to the testing stage, that if the testing batchsize is different from the batchsize set for training, the trained scale and shift may result in unexpected outcomes. In real-world implementations, the training for scale and shift can be turned off manually.

2.5.2 Layer Normalization

As shown in Fig. 2.4, layer normalization (7) firstly calculates the mean and variance from each feature volume without the scale and shift, then normalizes each feature volume with the calculated mean and variance. The authors present its superior convergence speed in training against batch normalization, and shows that layer normalization benefits the training of RNN in batches greatly.

2.5.3 Instance Normalization

To improve the performance of CNN for the image generator in a generative adversarial network (GAN, 53), instance normalization (195) has the possibility to dramatically outperform batch normalization. As shown in Fig. 2.4, instance normalization normalizes all features in one channel across the entire mini-batch using their mean and variance to preserve the feature styles.

2.5.4 Group Normalization

Group normalization (211) firstly divides feature positions into groups, then normalizes each group with its mean and variance. Also the normalized features can be scaled and shifted. Group normalization is a flexible technique. It can be transformed into batch normalization, layer normalization, or instance normalization by setting the shape of a group.

2.5.5 More Normalization Layers

Besides the four normalization techniques mentioned above, there are also weight normalization (163) for regulating weights during training, batch-instance normalization (130) that allows the complement of batch normalization and instance normalization, switchable normalization (110), and cosine normalization (109) etc.

2.6 Training, Test and Validation

2.6.1 Pre-Processing

When preparing a dataset for training a CNN, the first and foremost step is to modify the dateset as desired to boost the generalization ability of the model. There are many widely-implemented techniques for pre-processing, such as normalization, decoupling, purging, enhancement etc.

2.6.1.1 Normalization

The normalization (136) mentioned here is similar to the normalization layer. It enforces the distribution with given mean and variance upon samples such that the samples are subject to identical distribution, which may boost the performance[7]. Another advantage of this step is that the learnable parameters of batch normalization layers could learn this distribution, which may stabilize training. However, mean and variance need to be manually set, and improper mean and variance may lead to severe gradient exploding or vanishing problems.

Uniformization is the most popular technique in normalization for neural networks. As the data are various in their values, it should be rescaled within a certain range to enhance the robustness of neural networks. Although rescaling to range $[0,1]$ fits most cases, the resolution becomes much lower for the data with a broader value range such that useful signals are dominated by noisy spikes. In this case, normalization and data purging are always necessary before any uniformization.

2.6.1.2 Decoupling

The purpose of feature decoupling is to map raw feature to separable space for the neural networks to learn more reasonable hyper-planes that better partition the sample space to semantic categories. Decoupling is widely-implemented to machine learning tasks, such as the word embedding (98), principle component analysis (PCA, 2), and singular value decomposition (181) etc.

Raw samples may appear to be duplicated or low in quality thereby need to be purged (10).

[7] If being lucky enough, the unknown distribution of samples may be regulated to $\mu \approx 0,\ \sigma \approx 1$.

This step is particularly important for supervised learning or small datasets, i.e, the quality of annotations often dominates the performance, and incorrect annotations are intuitively misleading for the neural networks. Therefore, they should be purged for better training performance. However, the low-quality samples can sometimes enhance the performance since they prevent the neural network from overfitting. In reality, it is almost impossible to be assure that there exist no low-quality samples even for a small dataset. The circumstances become worse for large datasets. The role of data purging remains to be further explored.

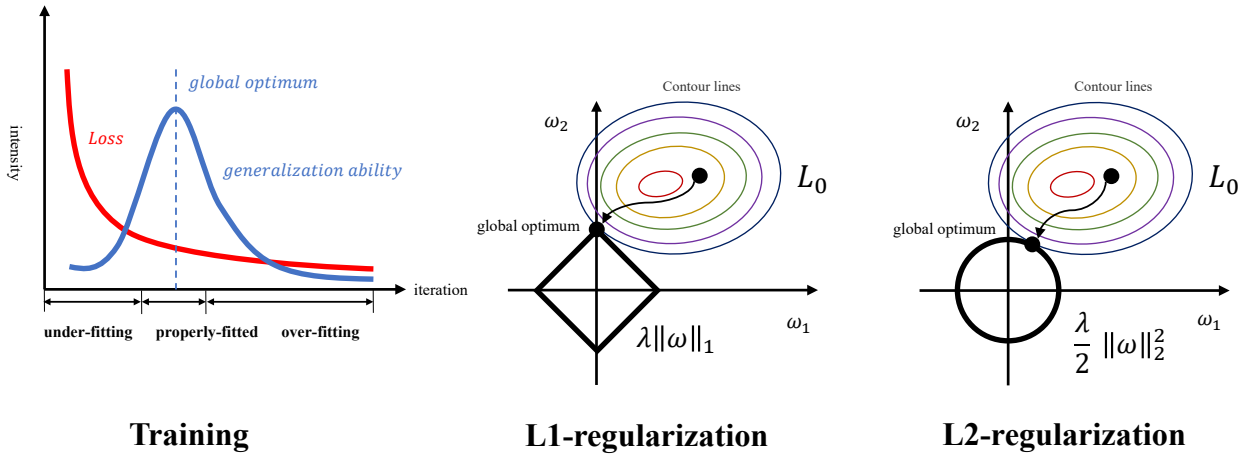
2.6.1.3 Enhancement

Data enhancement (176), or data augmentation, has become the most successful technique deployed for CNN. It is especially useful when the dataset is small, such that the samples cannot represent the sample space[8]. The aims for data enhancement are to balance number of samples in different categories, to enlarge dataset by bringing in synthetic samples, and to improve robustness of network via affine transformations such as random rotation, random shift and random flipping, image processing such as changing the colour contract or hue, image brightness or saturation, Gaussian blurring, and adding pepper or white noise etc. There are also special pre-processing techniques for different tasks depending on specific tasks. For example, for image retrieval, lower image resolution by blurring or reduce contextual information via masking, random cropping for generate training samples for high-resolution images, randomly shuffle training samples in order to prevent overfitting, which is also an underlying regularization approach.

2.6.2 Training

With pre-processed samples fed to a CNN-based model, one or more forward pass is firstly performed using initialized trainable parameters, then these parameters are updated from the loss function back to the input layer via gradient propagation. This progress forms one iteration during

[8]Fewer samples indicate sparser sampling, distribution retrieval of sample space is severely ill-posed due to the sparse sampling.



Training **L1-regularization** **L2-regularization**

Figure 2.5: Generalization ability in training, and L1-, L2- regularization terms.

training. The weights updating during one iteration can be regulated via regularization approaches or L1- and L2-terms, as well as optimized by optimizers. The purposes of implementing the techniques during training is to reach the global optimum of the convex parametric space.

2.6.2.1 Regularization

The generalization ability of one neural network model measures how good the model could classify novel samples (50; 91). Generalization ability is evaluated using the terms under-fitting, properly-fitted, and over-fitting (refer to Fig. 2.5). LeCun et al. defined the regularization as the the modification of learning algorithm so as to maximize its generalization ability. It should be noted that maximizing generalization ability is not achieved via finding the minimum of loss function in training (see Fig. 2.5), and the data enhancement mentioned above belongs to one category of regularization. L1-norm regularization and L2-norm regularization are the most popular approaches[9] by adding a penalty term to the loss function as L1-penalty or L2-penalty, i.e.,

$$L(\omega) = L_0(\omega) + \lambda\|\omega\|_1, \qquad L(\omega) = L_0(\omega) + \frac{\lambda}{2}\|\omega\|_2^2, \tag{2.29}$$

[9]L1- and L2- regularization are also termed as the Lasso regression and Ridge regression respectively in the topic of linear regression (16).

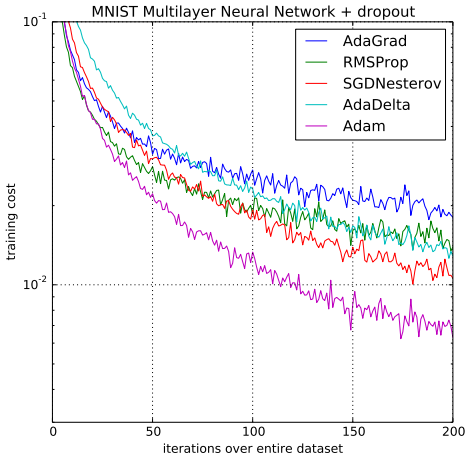


Figure 2.6: Optimizers comparison (83) on MNIST (32) dataset.

where ω is the parameters (all kernel weights for CNN) of the model, λ is a manually-set coefficient for scaling. Fig. 2.5 illustrates the 2D case that $\omega = \{\omega_1, \omega_2\}$, $\|\omega\|_1 = |\omega_1| + |\omega_2|$ and $\|\omega\|_2 = \sqrt{\omega_1^2 + \omega_2^2}$. As depicted in Fig. 2.5, the contour lines indicate positions where the original loss function L_0 has the same values in its bird's eye view, and the L1-regularization term has a squared boundary, the L2-regularization term has a circular boundary. An ideal training of ω starts from the initial point (between the contours) and ends at the global optimal point that signifies the first time L_0 intersects with the L1- and L2-penalty boundaries LeCun et al.. It has been proven that optimization of neural networks belongs to the category of convex optimization (16), thus there exists the global optimum that can be reached via gradients decent algorithms.

It can be inferred from Fig. 2.5 that L1-regularization may result in sparse parametric space since the path of L_0 is more liable to reach tips of the L1-penalty contour than the edges, while for L2-penalty term, its smoother contour is less likely to result in zero weights. As a result, L1-penalty is preferred in feature engineering, with which the dimension of parameters space is reduced, and L2-penalty is more popular in neural networks since this sparsity may result in dead neurons. Regularization effectively prevents the training from overfitting due to the penalties on parameters. However, it doesn't guarantee the convergence towards global optimum.

2.6.2.2 Dropout layer

Another widely-known regularization approach to prevent the training of CNN from over-fitting is the implementation of dropout layers (31; 180). During training, one dropout layer multiplies a gated variable to each intermediate output of the previous layer. The variable has a probability p to generate 1, otherwise 0, which is subject to Bernoulli distribution. This is to say, the dropout layer temporarily removes neurons from its previous layer with a probability of p. In the test phase, dropout layers are ignored, but the weights of the layer before a dropout layer are scaled by p to generate the expected outputs at training time.

2.6.2.3 Optimizers

One of the most important techniques for neural networks is the rules to update trainable weights, which at meantime, serve as effective regularization approaches. These rules are termed after optimizers. In Chapter 1, the SGD (Alg. 2 and 3) optimizer has been introduced. Brief introductions to other popular optimizers are presented in the following part of this section.

Adam optimizer (83) gains its popularity among CNN-based neural networks due to its simplicity and fast convergence (refer to Fig. 2.6). This optimizer involves two other optimizers: root-mean-square propagation (RMSprop) optimizer and SGD with momentum optimizer. The SGD with momentum optimizer (160; 186) is presented below. Supposed m_t denotes the $t-th$ moment ($t \geq 0$) of weights gradient $\Delta\omega_t = \frac{\partial L}{\partial \omega_t}$, then

$$m_{t+1} = \beta_1 m_t + (1-\beta_1)\Delta\omega_t, \qquad m_0 = 0, \tag{2.30}$$

substitute original $\Delta\omega_t$ (as Eqn. 2.6) by m_t, the SGD optimizer update ω_t with a learning rate η as

$$\omega_{t+1} = \omega_t - \eta m_t. \tag{2.31}$$

RMSprop improves the AdaGrad (37) optimizer. AdaGrad introduces adaptive learning rate to

replace the process of manually-tuning[10] the learning rate during training as

$$\omega_{t+1} = \omega_t - \frac{\eta}{\sqrt{G_{t+1}} + \varepsilon} \Delta\omega_t, \; G_{t+1} = G_t + \|\Delta\omega_t\|_2^2, \tag{2.32}$$

where $G_0 = 0$, ε is a small constant to ensure that the denominator is non-zero. Note that G_t also penalizes large gradients despite G_t may increase by time till infinity. RMSprop constructs a different adaptive learning rate term:

$$v_{t+1} = \beta_2 v_t + (1-\beta_2)\|\Delta\omega_t\|_2^2, \qquad \omega_{t+1} = \omega_t - \frac{\eta}{\sqrt{v_{t+1}} + \varepsilon} \Delta\omega_t, \tag{2.33}$$

where $v_0 = 0$, β_2 is a forgetting factor between [0,1]. Larger β_2 results in memorizing the old gradients and forgetting the new gradients, which is similar the behavior of β_1. Adam optimizer combines Eqn. 2.30 and Eqn. 2.33, it is updated during iterations as Alg. 4.

Algorithm 4: Adam Optimizer.

Initialize $\beta_1 = 0.9, \beta_2 = 0.999, m_0 = 0, v_0 = 0, t = 0$, total iterations N_s,ε,η;
while $t < N_s$ **do**
  Calculate $\Delta\omega_t = \frac{\partial L}{\partial \omega_t}$;
  $m_{t+1} := \beta_1 m_t + (1-\beta_1)\Delta\omega_t$;
  $v_{t+1} := \beta_2 v_t + (1-\beta_2)\|\Delta\omega_t\|_2^2$;
  Calculate normalized states: $\hat{m}_t = \frac{m_{t+1}}{1-\beta_1^{t+1}}$, $\hat{v}_t = \frac{v_{t+1}}{1-\beta_2^{t+1}}$
  Update weights: $\omega_{t+1} = \omega_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$
  $t := t + 1$;
return W

Other optimizers including ADADELTA (226), Adam with weight decay (AdamW, 107), and SGD with Nesterov momentum (131) etc. Note that the state-of-the-art optimizers cannot guarantee achieving the global optimum, but merely provide more feasible solutions to reach the global optimum, which remains to be a crucial research topic in the field of deep learning.

[10]For first iterations, set several larger learning rates, set smaller learning rates later to ensure fast convergence. Modulating learning rate can also be achieved via implementing an exponential function.

2.6.3 Test

Test a neural network is similar with training the neural network for one iteration[11]. Compare to the training phase, test doesn't need the loss function and any gradient computation. The only things test phase needs are the test samples and the model including weights and operations other than the gradients. It should be noted that the behavior of batch normalization layer and dropout layer are different form those during training. Also, as test requires a batchsize, if the batchsize is different from that of training, the original batch updating rules may not be suitable for test batches. However, during my test, the results were not greatly affected. Another thing is to accelerate the inference time during test. Single-precision or integer data type is much faster than double-precision type, but this reduction will result in drop in the performance due to the loss of precision.

Selection of test samples matters the most. If the features of test samples are coupled with training samples, or merely part of the training samples, the test result is not accountable. On the other hand, if the test samples are sampled from a different sample space than that of training samples, the test may fail due to its novelty. To avoid this dilemma as much as possible, a dataset should be manually divided into, at least, training-set and test-set. The novel test samples are collected in similar conditions as the training samples.

2.6.4 Validation

Validation stage is always accompanied after training or during the process of training. The goal of validation is to evaluate the effects of training which is similar with testing. In some cases, validation stage can be skipped if the performance of training has already been verified. Validation during training leverages on evaluation metrics to verify the network by using a portion of training samples[12]. Though these metrics cannot prove the generalization ability of the network, we can still utilize them for judging if the training evolves into the over-fitting stage since the value of loss function are not intuitive sometimes, or we can at least predict the performance of the network in

[11] Usually, the entire training procedure is divided into epochs, and each epoch is composed of several iterations.

[12] If the inference is fast, or the dataset is small, all training samples can be used for validation.

case training an unpleasant model may take a long period of time.

Validation after training suits the circumstance of large dataset when validation during training is inapplicable. K-fold cross-validation (45; 91) is the most commonly implemented approach. K-fold cross-validation constructs training-set and validation-set at the same time by splitting the entire training-set into K groups, and then takes one group as the validation-set, the others as a new training-set. For each group, it generates a pair of new training-set and validation-set in this way, validates results on the validation set, then discards the model and record all scores on metrics. Finally, it summarizes K-groups of scores and usually takes the mean values as the overall validation performance.

2.7 Pre-Processing: Beat-Rhythm Transformer

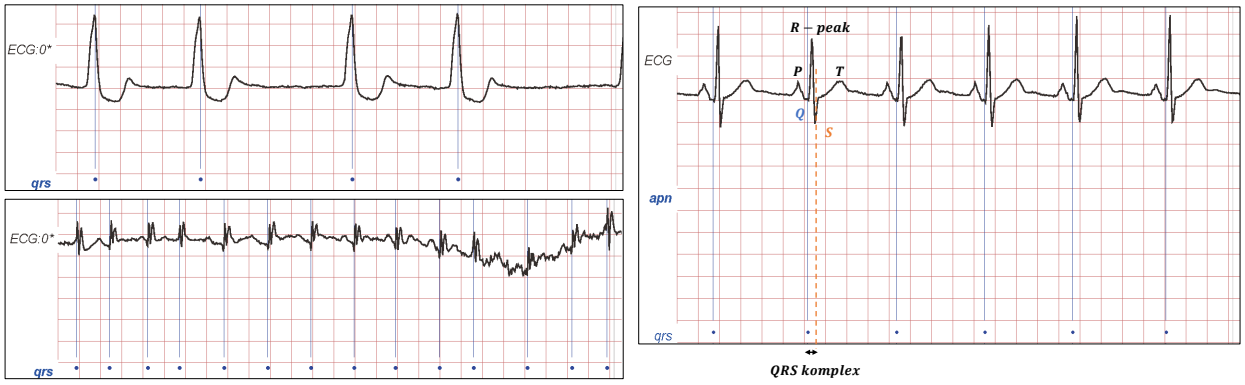
2.7.0.1 Overview

In this section, a beat-rhythm transformer[13](BRTransformer) is proposed as an example to illustrate the necessity of pre-processing. The BRTransformer is also one of the very first works that proposed self-attention mechanism (196) that inspires the designs of self-attention modules in CNN. This section first introduces the necessity of the Atrial Fibrillation (AF) detection, then demonstrates pre-processing techniques to detect R-peaks and how conceptual abstraction and fuzzification could simplify this real-world problem.

2.7.0.2 Concepts and Related Works

Paroxysmal Atrial Fibrillation (PAF) Event Atrial fibrillation (AF) is the most common sustained arrhythmia (19). In the United States, the percentage of Medicare fee-for-service beneficiaries with AF in 2010 was reported as 2% for those $<$65 years of age and 9% for those $\leq$ 65 years of age (78). AF events are characterized by disorganized atrial electrical activity and contraction. The incidence of AF in patients with acute coronary syndrome ranges from 10% to 21% and in-

[13]Source codes are presented in Appendix.A



different kinds of AF events **normal ECG**

Figure 2.7: ECG comparison on normal type (137) and AF-events (126).

creases with patient age and severity of myocardial infarction. In the medicare population, AF is associated with increased in-hospital mortality rate (25.3% with AF versus 16.0% without AF), 30-day mortality rate (29.3% versus 19.1%), and 1-year mortality rate (48.3% versus 32.7%). With multivariate adjustment, AF remains an independent predictor of death (79). Clinical symptoms of AF events (refer to Fig. 2.7) include irregularly rhythm (R-R intervals), no P-waves absence of an isoelectric baseline, variable ventricular rate, QRS komplexes that are usually lower than 120ms, and possible fibrillatory waves that are fine or coarse, etc.

According to the presentation and duration of AF, AF events can be classified as first episode, recurrent AF, PAF (Also known as intermittent AF), persistent AF, long-standing persistent AF, and permanent AF. Though AF is the most frequent arrhythmia for doctors to diagnose by electrocardiogram (ECG), PAF often remains unrecognized (1) due to its unpredictable paroxysmal symptom that occurs suddenly and then stops within 7 days. Sometimes, it may last for less than 24 hours (40). Moreover, it's unrealistic to personally monitor PAF events for days without interruption.

In this example, we explore the potentials of automatically detecting AF events regarding to the rhythm symptom. As one of the most significant symptoms of PAF, the heat-beat rhythm can be represented via intervals between consecutive R-peaks (see Fig. 2.7). Therefore, accurate R-peak detection algorithm is required to construct rhythmic features for detecting PAF events from

recorded raw ECG samples.

R-peak detection ECG records the electrical signal form the cardioid to examine cardiac conditions. A typical cardiograph may sample the electrical signal at 1kHz, and down-sample to 500Hz, 250Hz and 200Hz in excellent concordance (89). The schematic diagram of norm heat-beat is shown in Fig. 2.7 that, three waves, namely P, R, T, characterize one heart-beat, while the Q, S valley and R-peak of R wave indicate the most salient feature of the heart-beat. Among all the P, R, T peaks, R-peak is the most distinctive feature to be detected (121; 120; 42; 161; 150; 183; 30; 148). Distances between R-peaks represent the rhythmic feature of heart-beats. In clinical practice, it usually takes hours to record an raw ECG sample that consists of more than 720,000 sampled signals (200Hz). Thus, real-time peak detection in the pre-processing stage is necessary, so as to its following real-time implementations that predict AF events via positions of detected R-peaks.

PAF event classification The common approach to detect PAF event is the diagnosis by doctors' expertise. Only a few approaches are proposed to solve the problem by algorithms for classification, especially via implementation of machine learning techniques (185). As the state-of-the-art performer for language translation, Transformer (196) converts sentences to desired translations in an end-to-end fashion. We believe the sequence-to-sequence translating mechanism of Transformer can be accommodated to the task of PAF event detection.

BRTransformer Transformer is often viewed as the fourth category of neural networks for NLP tasks apart from RNN (26), CNN, and graph neural network (GNN, 165). Typical Transformer (196) consists of an encoder and a decoder networks, along with the final linear layer for modulating the predicted word vectors. Input word vectors of sentences are positionally encoded and fed to the encoder networks. The encoder then performs multi-head self-attention on the word vectors, and feed forward feature vectors to the decoder network that has the same architecture as the encoder network.

The overall process of transformer in NLP is sequence-to-sequence. However, it cannot be

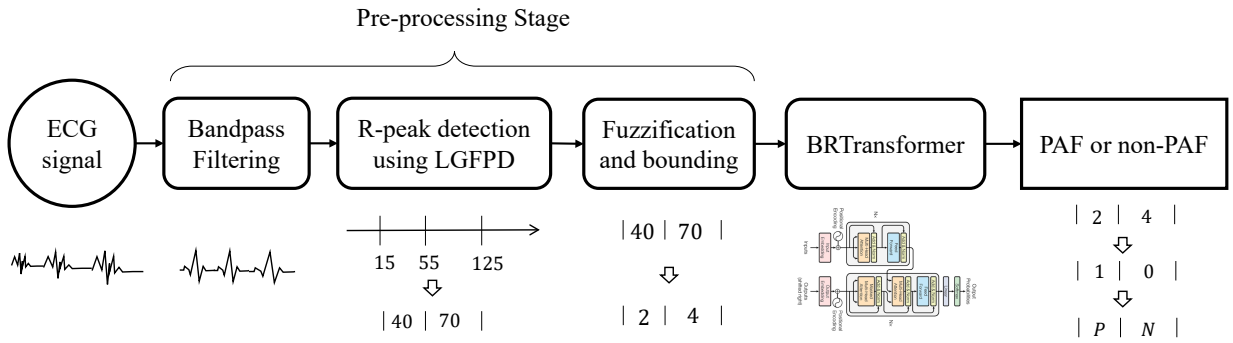


Figure 2.8: BRTransformer work flow, transformer image from (196).

directly applied to the AF event detection task. Therefore, we propose to reformulate the detection task as a classification task using fuzzy subset of membership quantification (118) and rhythm embedding (98). A fuzzy subset of membership indicates a category that one variable belongs to, e.g., when we describe the air temperature, there's no theoretical boundary of judging if the temperature is high, median, or low, whereas we can still quantify the air temperature to fuzzy subsets of membership 'low', 'median', 'high' based on the real-life experiences.

Rhythms of heart-beats can also be categorized into fuzzy subsets of membership. With the quantification of rhythms, we construct a word embedding of rhythms, i.e., the rhythm embedding, which reformulates the detection task to classify rhythms into PAF and non-PAF events in a sequence-to-sequence fashion.

2.7.0.3 Methodology

Pipeline The pipeline of proposed BRTransformer is demonstrated in Fig. 2.8. As there are multi-channel ECG signals, we only leverage on the raw signals whose R-peaks can be detected. Raw ECG signal has noises both in high frequency and low frequency. Therefore, we implemented the Butterworth bandpass filter (20) to filter those signals (205) since normal heart-beat period is definitely restrained to a certain range. Salient R-peaks can be detected using proposed LGFPD, the intervals are then calculated as the differential of the R-peak coordinates. With the intervals, semantic levels of amplitude can be manually defined using categories range from 0 to $N_l + 2$

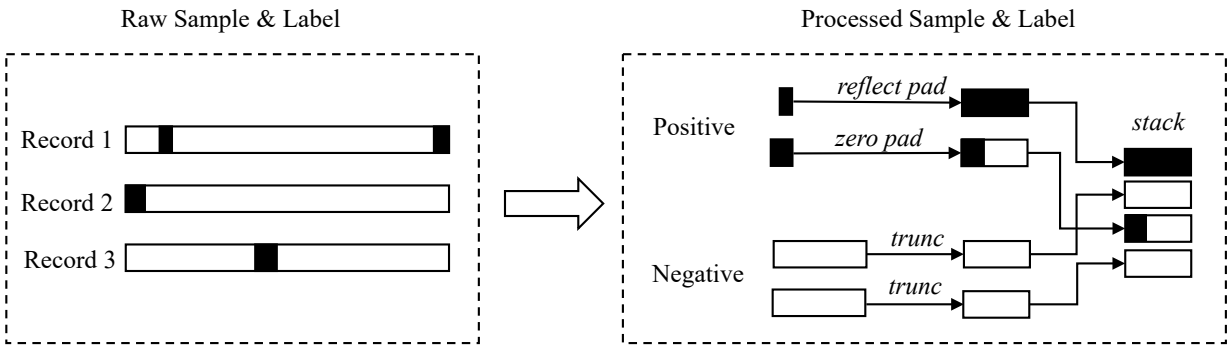


Figure 2.9: Sample re-balance procedure before feeding to BRTransformer.

since the last two category corresponds to the non-PAF event ($N_l + 1$) and PAF ($N_l + 2$) event. Fuzzification is then performed via max-min normalizing and projecting the intervals to the fuzzy categories of levels (0 to N_l). For those extremely large abnormal intervals, we bound the their categories to N_l. In this way, the fuzzification has set the raw training data for a transformer (196).

At this point, the pre-processing stage has not been finished yet. We may observe only a very small proportion of PAF events among the ECG signals collected for hours in training set (205). To balance this improper potion of samples, we construct a mini-batch of training samples as depicted in Fig. 2.9 to collect all PAF and non-PAF episodes and random sample four episodes. We define the PAF episode as the positive sample, and the non-PAF as the negative sample. Moreover, if positive episodes are shorter than the desired length, one shorter episode is padded by reflection, while the other is padded with zeros. If the other way around, we directly sample two samples with the desired length. Also, two negative episodes are resized to the fixed length. Finally, the four samples are stacked alternately before feeding to the BRTransformer. It should be noted that this is not the end of pre-processing stage since the transformer only accepts embedded vectors. In the following part, the rest of pre-processing stage will be presented along with the transformer.

R-peak detection using Local-Global Filtering Peak Detector (LGFPD) We proposed the LGFPD R-peak detector in the pre-processing stage. LGFPD algorithm is shown as LGFPD-B (Alg. 5), LGFPD-C (Alg. 6), and LGFPD-F (Alg. 7) without considering the boundary conditions between global windows. Expending the furthest range of variable j to $\min(iW_g, N_s)$ in Alg. 5 will

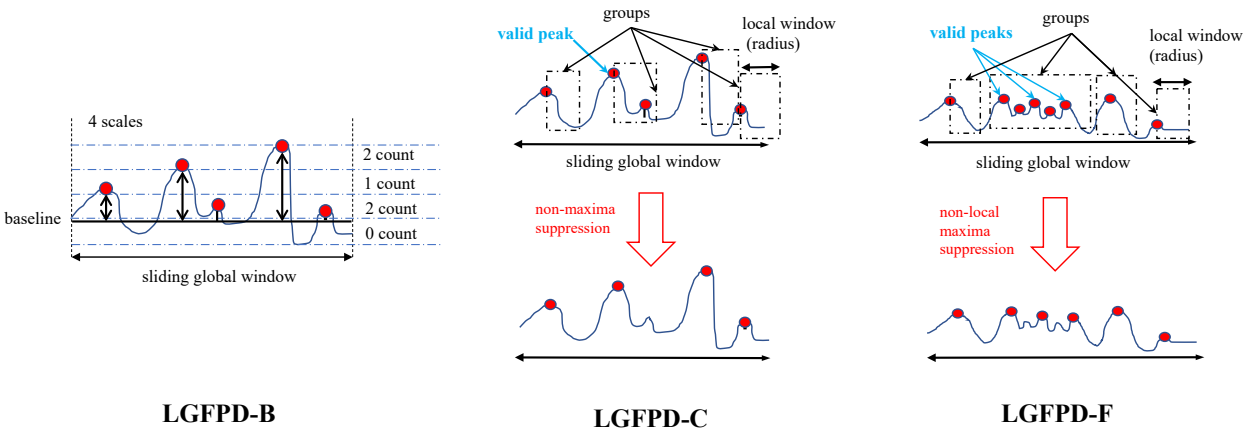


Figure 2.10: Local-Global Filtering Peak Detectors.

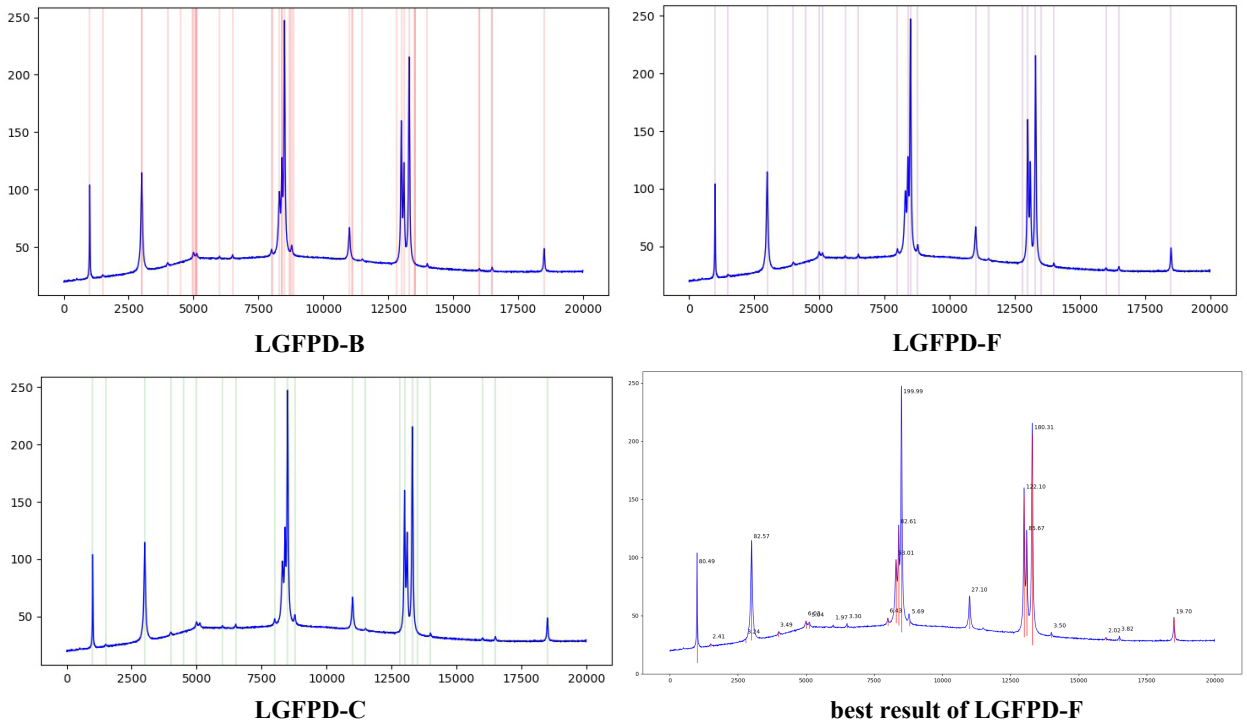


Figure 2.11: LGFPD results of same local window size, and its best result on test signals.

solve the boundary conditions.

Algorithm 5: Local-Global Filtering Peak Detector - Base (LGFPD-B).

Read the discrete input signal $s[t]$, $t = 1, \cdots, N_s$;
Initialize global window size W_g, bin number N_b, height threshold T_h, temporary result $R_t = \{\}$ which is an empty set and the final result $R = \{\}$, $found\_mark = False$;
Partition $s[t]$ to $\lfloor \frac{N_s}{W_g} \rfloor$+1 global windows, let $i = 0$;
for $i := 1$ *to* N_s **do**
    Get values in the $i-th$ global window: $V = \{s[k] | (i-1)W_g \leq k \leq \min(iW_g, N_s) - 1\}$;
    Find maximum and minimum values: $v_{max} := \max(V)$, $v_{min} := \min(V)$;
    for $(i-1)W_g \leq j < min(iW_g, N_s) - 1$ **do**
        if $s[j] \geq s[j+1]$ **then**
            if $found\_mark$ *is False* **then**
                $s[j] \rightarrow R_t$, set $found\_mark$ to $True$
        else
            if $found\_mark$ *is True* **then**
                set $found\_mark$ to $False$
    Calculate resolution $\frac{v_{max} - v_{min}}{N_b}$, project all peaks in R to the bin according to resolution;
    Count the number of peaks falling in each bin as $bin\_counts = \{c_1, \cdots, c_{N_b}\}$;
    From the lowest bin ($1-st$ bin) to the highest ($N_b - th$ bin), find the bin with maximum count: $c^* = \arg\max(bin\_counts)$;
    Calculate the baseline as the average over all peaks in $c^* - th$ bin;
    Calculate amplitiute for all peaks in other bins above (not including) $c^* - th$ bin using the baseline as the common vally of waves;
    Filter the calculated amplitiutes by thresholding with T_h;
    Save the filtered peaks (locations and amplititudes) to R;
return R

LGFPD-F and LGFPD-C are based on the results of LGFPD-B. LGFPD-C remove all non-maxima peaks in the local window, while LGFPD-F only removes all non-local maxima peaks in the local window, which follows the same procedures as the raw-peak detection in Alg. 5. We tested three algorithms on a test signal with a length of 20,000. The results are shown in Fig. 2.11. LGFPD-C refines 41 peaks that are detected by LGFPD-B to 21 peaks, while LGFPD-F locates 23 peaks.

BRTransformer In this example, a transformer is implemented for the classification stage. The input samples are represented by intervals with fuzzification, but still they need to be embedded

Algorithm 6: Local-Global Filtering Peak Detector - Combined (LGFPD-C).

Execute Alg. 5 first to yield the raw result R;
Initialize local window radius r_c
for *the first ungrouped peak to the last ungrouped peak in* R **do**
    collect all peaks fallen within radius r_c, mark all those peaks as grouped peaks;
    Find the highest peak in that group, removes all other peaks in that group;
return R

Algorithm 7: Local-Global Filtering Peak Detector - Filtered (LGFPD-F).

Execute Alg. 5 first to yield the raw result R;
Initialize local window radius r_f
for *the first ungrouped peak to the last ungrouped peak in* R **do**
    collect all peaks fallen within radius r_c, mark all those peaks as grouped peaks;
    Detect local maxima from the group as the inner for-loop in Alg. 5;
    Depricate all other peaks in that group;
return R

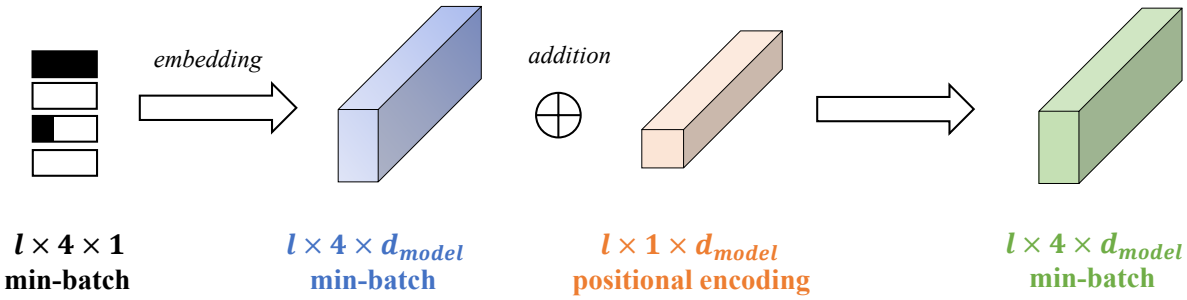
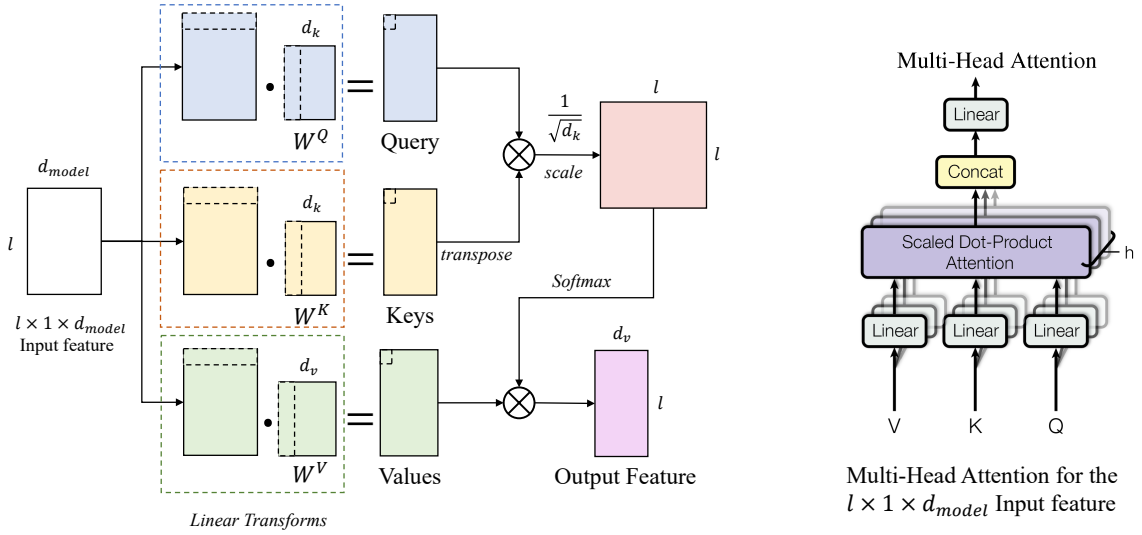


Figure 2.12: Positional embedding of conceptualized and fuzzified batch of samples.



Scaled Dot-Product Attention **Multi-Head Self-Attention Module**

Figure 2.13: Scaled dot-product attention and multi-head attention (196).

into higher-dimensional space. Transformer (196) firstly encodes the position using cosine and sine function as

$$PE(pos,2i) = \sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}}), \qquad PE(pos,2i+1) = \cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}}), \tag{2.34}$$

where $0 \leq i \leq d_{model}$ represents one dimension of embedding feature space, variable pos signifies one position in the fixed length of input batch, which, as depicted in Fig. 2.12, we use the variable 'l' to represent this fixed length adjusted during conceptualization and fuzzification. Positional encoding has the dimension of $l \times 1 \times d_{model}$, which is previously propagated to the size $l \times 4 \times d_{model}$ by duplication. The addition is performed elements wise, which introduces positional information to the embedded batch to boost the performance of transformer.

Transformer leverages on the self-attention mechanism. This type of attention is achieved via the multi-head self-attention module (196), which is the most crucial design in transformer. Multi-head self-attention module is illustrated in Fig. 2.13. It is composed of multiple scaled dot-product attention modules that are based on self-attention mechanism.

As depicted in Fig. 2.13, for each $l \times 1 \times d_{model}$ sample X_i, $1 \leq i \leq 4$, in a $l \times 4 \times d_{model}$

mini-batch after positional encoding (Fig. 2.12), the sample is fed to three branches to generate query (Q), keys (K) and values (V) matrices by linearly projecting the sample to d_k-dimensional space (Q and K), and d_v-dimensional space (V) as the dot-product of matrices $X_i \cdot W^Q$, $X_i \cdot W^K$, $X_i \cdot W^V$, where W^K, W^Q, W^V are $d_{model} \times d_k$, $d_{model} \times d_k, d_{model} \times d_v$ translatable weights matrices, respectively. The next step is to correlate each query with all keys by dot-product as $Q \cdot K^T$. Note that the product may be large such that there could be gradient exploding issues. $Q \cdot K^T$ is scaled by their dimension $\frac{1}{\sqrt{d_k}}$. The last step is to output the feature as $softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$. The $softmax(\cdot)$ operation is necessary since the $l \times l$ correlation matrix of Q and K depicts the feature correlations in l positions. Softmax operation is enforced to each row in the correlation matrix. Not only this softmax operation normalizes the row-wise vectors corresponding to each query, but also assigns the positions w/ the largest responses with higher values. Both functions make the correlation matrix a non-negative weights matrix for V. As a result, the output feature is a weighted-sum of l values along d_v dimensions. The self-attention is achieved via matrix multiplications, which can be regarded as measuring the cosine similarities between feature vectors.

Multi-head self-attention is also shown in Fig. 2.13. The figure presents a multi-head attention module with h scaled dot-product attention modules. The output feature of all scaled dot-product attention modules are concatenated as a $l \times d_v \times h$ feature volume and linearly projected (weighted-sum) to the same dimension $l \times d_v$ as the original output feature. Multi-head attention alleviates the problem of local maximum, that one head tends to predict the strongest correlations and fails to model other weaker but crucial correlations. Multiple heads provide more correlations, and the following weighted-sum operation may rectify those overwhelming correlations to enhance the capability of transformer. Another thing should be noted is the sample X_i from the batch as we discussed above. In fact, all sampling share the same weights matrix W^Q, W^K, W^V, while the multi-head attention does share weights between heads as W_j^Q, W_j^K, W_j^V, $1 \leq j \leq h$.

Intervals Detector	LGFPD		Groundtruth	
Classifiers	baseline	BRTransformer	baseline	BRTransformer
Score	-0.0878	0.1589	0.0000	0.4375
Heads	-	32	-	32
encoders & decoders	-	10	-	10
d_{model}	-	1024	-	1024
d_v, d_k	-	512	-	512

Table 2.1: BRTransformer performance on validation set.

2.7.0.4 Experiments

We divide the entire training set to validation-set ad new training-set to verify the performance of BRTransformer. To compare the performance with the baseline, we further test the validation-set with the baseline approach implemented in the evaluation codes (205). The results are shown in Tab. 2.1. The evaluation metrics are provided by Wang & cpsc2021 and larger value indicates better results. After training for 20 epochs, each includes 200 iterations with a exponentially-decayed learning rate 1.0, though LGFPD may not detect the same peaks as the ground-truth annotations, BRTransformer still outperforms the baseline. When tested on groundtruth, baseline fails to show valid results, while BRTransformer outperform base by a large margin with a faster inference rate at 5.34 $records/s$ on a NVIDIA GTX3090 GPU.

2.7.0.5 Conclusion

Processing dynamic ECG signals is one of the most commonly implemented non-invasive technique in monitoring as well as in clinical diagnosis for cardiovascular disease. Early, fast screening, and detection of PAF events are particularly important for practical diagnosis. In this section, we proposed BRTransformer, a sequential-to-sequential translator for the classification task of PAF events from ECG signals. To detect the R-peaks of ECG, we also designed an efficient local-global filtered peak detector (LGFPD). BRTransformer takes sequences of detected R-peak intervals as

the input, then translates the intervals to PAF events and non-PAF events without rings and bells. Proposed approach is tested on the public-available dataset provided by 4th China Physiological Signal Challenge 2021, and it outperforms the baseline approach by a large margin with fast inferential performance.

Chapter 3

Pre-Processing: Stereo Frustums for 3D Object Detection

Abstract

We proposed a light-weighted stereo frustums matching module for 3D objection detection. The proposed framework takes advantage of a high-performance 2D detector and a point cloud segmentation network to regress 3D bounding boxes for autonomous driving vehicles. Instead of performing traditional stereo matching to compute disparities, the module directly takes the 2D proposals from both the left and the right views as input. Based on the epipolar constraints recovered from the well-calibrated stereo cameras, we propose four matching algorithms to search for the best match for each proposal between the stereo image pairs. Each matching pair proposes a segmentation of the scene which is then fed into a 3D bounding box regression network. Results of extensive experiments on KITTI dataset demonstrate that the proposed Siamese pipeline[1] outperforms the state-of-the-art stereo-based 3D bounding box regression methods.

3.1 Introduction

How to regress accurate 3D bounding boxes (bbox) for autonomous driving vehicles has become a pivotal topic recently. This technique can also benefit mobile robots and unmanned aerial vehicles with regard to scene understanding and reasoning. In this chapter, we propose a Siamese pipeline method for 3D object detection.

[1] Relevant works are published as (124)

Given a pair of stereo images and the point cloud data collected by velodyne (Geiger et al.), many approaches on a basis of deep-learning theories have been proposed to generate 3D bbox artifacts which can also be projected to a bird's-eye view (BEV) of LiDAR data for localization evaluation. According to the number of image views these approaches utilized, they can be divided into three categories: monocular view (208; 143; 36; 175; 212; 99; 86; 171), binocular views (95; 84; 207; 25; 146), and non-view approaches (237; 219; 200; 111; 215; 220; 172) that only processes point cloud. Mono-view based approaches focus on sensor-fusion of the camera and LiDAR sensors in either a global or a local manner, while non-view approaches extract point cloud features from hand-crafted voxels or raw coordinates. Compared to the extensive development in both categories mentioned above, there are fewer stereo-based and stereopsis-LiDAR-fusion works for 3D object detection.

Considering the runtime of stereo matching, coarse disparity map generated by fast stereo matching and GPU acceleration achieves real-time frame-rate, yet less accurate 3D detection results (84) compared with that of coarse-to-fine disparity map (207). However, it usually takes a few minutes to generate one panorama of coarse-to-fine disparity map before performing object detection tasks. Moreover, pixel-level stereo matching is sensitive to the error in the epipolar line calculated from camera calibration as stereo matching assumes all epipolar lines to be horizontal. We propose to reduce runtime by performing RoIs-level stereo matching instead of matching all pixels, and by a fast epipolar line searching strategy which calculates epipolar line from calibration data. We show in Section 3.4.4 that most of the runtime goes to point cloud processing.

Most stereo-based methods rely on stereo matching of stereopsis to generate depth maps for 3D object detection (84; 207; 25), and one Stereo R-CNN based method (95) directly regresses keypoints of 3D bbox from the left-right correspondence of regional proposals (RoIs). In stereo matching, very close objects are usually located on the border area in both views with very large disparities. Therefore, part of the same object can be missing in either view. In this case, stereo-based methods may be unable to locate matched keypoints by stereo matching. Furthermore, considering the perspective changes, the same object may appear distinctively in both views due

to occlusions. These situations may cause intrinsic ambiguities in stereo matching. As shown in Fig. 3.1, the proposed method, by taking advantage of LiDAR-based 3D object detection methods, neither relies on stereo matching at pixel level nor predicts key points from corresponding RoIs.

To circumvent the matching ambiguities, several approaches introduce spatial constraints as 3D anchors (146) and regress point cloud proposals transformed from dense disparity maps (84; 207). However, the design of a grid of anchors which has a proper density as well as high average precision (AP) for continuous space needs to be hand-crafted. Inspired by a novel single-frustum based method (208), we propose to solve the ambiguities by making full use of the spatial information for multi-modal regression. A novel module is proposed to directly map the point cloud onto the RoIs of the stereo image pairs and perform matching, by means of the normalized cross-correlation or the proposed 3D Intersection of Union (IoU) matching cost, and fast epipolar line search. This module correlates the 2D detection and synchronized 3D point cloud, and a novel network pipeline is proposed to accommodate this module.

The sparse nature of LiDAR points substantially reduces the number of points to be processed compared to the point cloud generated by the dense disparity map. Therefore, using LiDAR data can alleviate the computational burden of high-density matching at the pixel level. As to the perspective change, according to the setup of datum collecting vehicle, the baseline of stereo cameras is $0.54m$ such that both views trap the same set of 3D keypoints from an object even though regions of the object appear distinctively. In addition, our method is robust to a slight disturbance on 2D bboxes (see Section 3.4.2).

The main contributions of this chapter include:

- We propose an embedded light-weight matching module to generate 3D segmentation proposals by the RoIs from stereopsises.
- The proposed 3D IoU cost and epipolar line search algorithm are efficient in finding matches without Cython or GPU acceleration.
- The proposed Siamese architecture bridges the gap between stereopsis and real LiDAR points by integrating the point cloud segmentation network with 2D RoIs.

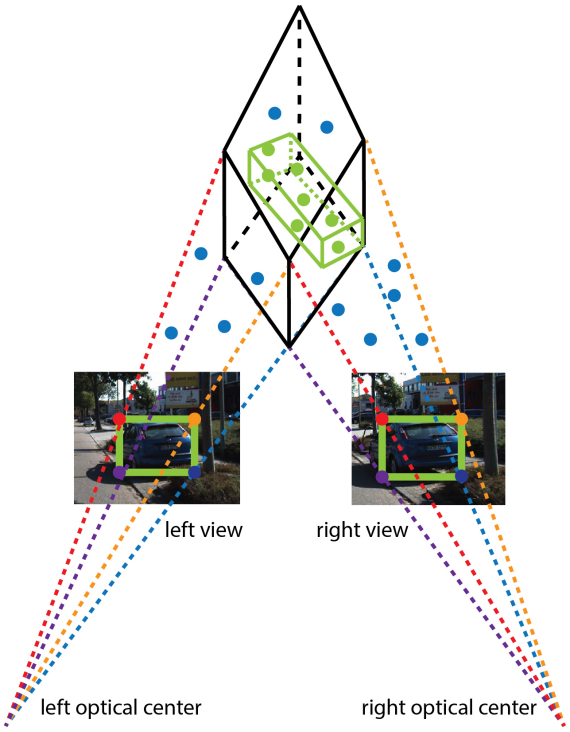


Figure 3.1: Illustration of stereo frustums.
Valid inliers (green points) are segmented from inliers for predicting 3D bounding box (green). Given accurate 2D bboxes, the intersection of two frustums encloses the point cloud of the interested object with less ambiguity than single-frustum based and stereo-only methods, optimizing search space and being robust to perspective change.

The proposed framework has been evaluated extensively on the KITTI dataset (46). The experimental results outperform previous stereo-based 3D bbox regression approaches. When testing on KITTI validation set, our method maintains as high AP on car detection as F-PointNets (FPN) (143), and outperforms FPN on pedestrian detection. The proposed approach runs on average at 2-3 frames per second.

3.2 Related Works

3.2.1 Monocular Pipeline for 3D Detection

Du *et al.* (36) propose a general pipeline to detect cars from point cloud subsets constrained by monocular 2D detections. Three categories of normalized templates generalized from CAD models are fitted to 3D proposals in each subset. Each proposal is generated by RANSAC algorithm. The

voxelized 3D proposals with the highest matching scores are fed to a two-stage refinement network.

Another improvement of the monocular pipeline is FPN for the purpose of regressing amodal 3D bbox including bbox sizes, orientation, and 3D bbox center. To our best knowledge, this work firstly proposes the concept of frustum which assigns bins and scores to point cloud subsets constrained by 2D detections. Instead of voxelizing the point cloud (237) before feeding it into a segmentation network followed by a T-Net to regress offset of 3D bbox center, the authors of (143) design and apply the initial feature extraction networks PointNet(v1) (144) and PointNet++(v2) (145) that learn point coordinates directly. In following sections, we denote FPN with PointNet++ backbone as FPNv2.

RoarNet (175) points out that the performance of FPN degrades if the camera sensors and velodyne sensor are not synchronized. This work proposes a geometric agreement search by selecting the best projection from a 2D detection to its 3D bbox within single frustum with the help of spatial scattering to refine the location of 3D bbox. This improvement alleviates but can not solve the ambiguity of monocular back-projection (see Fig. 3.1). Recently, F-ConvNet (208) ranks leading position on KITTI benchmark. It proposes a sliding-window fashion along frustum to solve the localization ambiguity. As one of the state-of-the-art monocular detectors, F-ConvNet remedies improper hand-crafted divisions by concatenating point features from all windows, and learns valid objectness by a fully-convolutional network.

3.2.2 Stereo-Based Methods

Several works have discussed the possibilities of 3D bbox regression by 2D detection w/o auxiliary depth information. Li *et al.* (95) propose a new end-to-end approach based on stereo R-CNN to perform regional detection, and it is also integrated to a 2D-keypoint prediction network designed for vertex estimation of 3D bbox. Nevertheless, the predicted 2D-keypoints are lack of accuracy considering perspective changes. In addition, the orientation of the object is not included in the regression artifacts. 3DOP (25) trains a structured SVM to generate 3D bbox proposals, which learns weights for an energy function that incorporates the point cloud density, free space, height prior,

and height contrast information. However, without taking advantage of dense epipolar constraints in raw stereopsis, 3DOP predicts less accurate 3D bboxes than our approach.

Recently, a triangulation learning network (146) aims at learning epipolar constraints. The network requires anchors grid and 3D bbox ground truth to train. Left and right RoIs are selected by frustum-like forward-projection of 3D bboxes before cosine similarity is imposed on their feature maps. By cosine coherence scores computed from the left-right RoI pairs, the reweighting process weakens the signals from noisy channels. This method does not utilize dense raw epipolar constraints, and heavily relies on sparse anchor grids for localization. Also, this method brings in ambiguity since it searches among all potential anchors captured by a single frustum.

RT3D (84) and Pseudo-LiDAR (207) estimate RGB-D image from stereopsis, then transform it into the point cloud, and regress 3D bbox with off-the-shelf clustering or LiDAR-based methods. RT3D presents a realtime detection scheme but comes with lower AP, while Pseudo-LiDAR predicts a less accurate depth map than real LiDAR data, which we will show in our experiments that, by the same FPN detector, our method achieves higher AP.

Another trend for object detection is through sensor fusion, i.e., to synchronize signals from multiple sensors. You *et al.* (221) proposed a scheme, namely PL++, to fuse sparse LiDAR points and the corresponding point cloud generated from the RGB-D image. As one of the top performers in 3D object detection, PL++ takes advantage of highly precise LiDAR points in localization. Zhang *et al.* (231) designed a deep network that fuses accurate depth maps, color images, and optical flow data. The model outperforms state-of-the-art works by a large margin. In order to further explore accurate RGB-D image in object detection, Tian *et al.* (194) proposed a novel representation for a 2D convolutional network that encodes depth map, multi-order depth template, and height difference map. This approach achieves real-time performance, as well as being robust to insufficient illumination and partial occlusion.

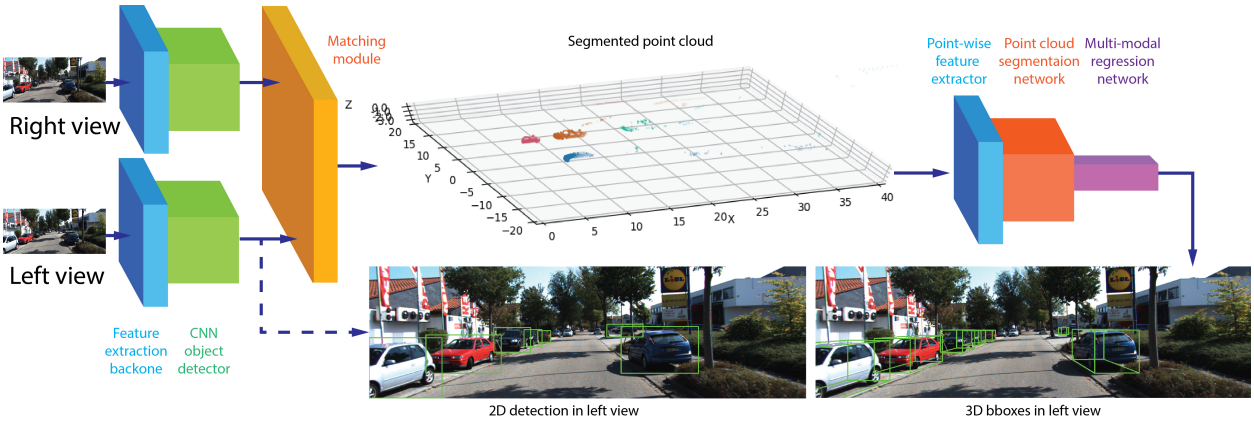


Figure 3.2: Siamese pipeline for 3D object detection.
Arrows signify flow directions, the dash line signifies an inspection of flow node. Feature extraction backbone and CNN object detector of sibling branches share the same weights and parameters.

3.3 Stereo Frustums Pipeline

In this section, we propose a Siamese pipeline (SFPN) that takes advantage of over-constrained epipolar geometry. Section 3.3.1 describes the dense epipolar constraints that SFPN bases on. Section 3.3.2 introduces an overview of stereo frustum pipeline. Section 3.3.3 introduces four RoIs matching methods that our module has implemented. In this section, we assume the 2D object detections on both views are consistent with their ground truths.

3.3.1 Dense Epipolar Constraints

Notations. Let $F \in \mathbb{R}^{3\times3}$ be the fundamental matrix defined by the left-right camera coordinate systems (see Fig. 3.3) whose optical centers are O_2 and O_3 respectively. Let $\mathscr{S}_2 = \{l_i | i = 1,2,...,m\}$ be the set of bboxes centered at l_i in the left view, $\mathscr{S}_3 = \{r_i | i = 1,2,...,n\}$ be the set of bboxes centered at r_i in the right view, P_c denotes the point cloud set of scene, with $P_c(l_i)$ a subset of P_c by forward projecting P_c to the bbox region centered at l_i, N_{thres} denotes the minimum number of spacial points required for the multi-modal regression network, and $N(P_c(l_i))$ denotes the number of inliers of $P_c(l_i)$. SPFN is valid if the following two constraints are satisfied:

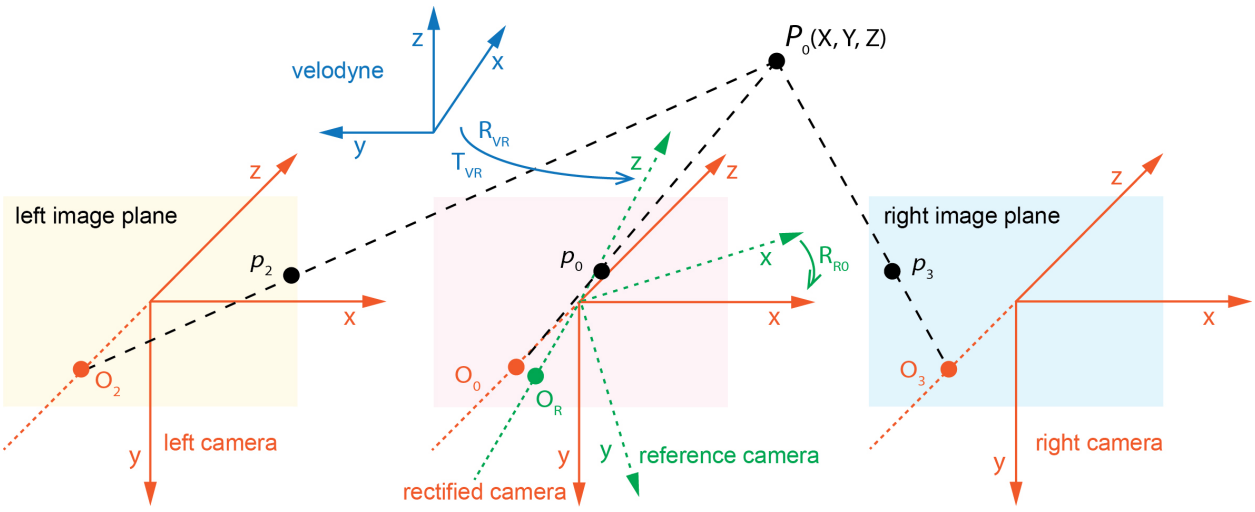


Figure 3.3: Coordinate systems of the KITTI data-collection vehicle (46) . P_0 is a LiDAR point in velodyne coordinate system (blue), its forward-projections in left view and right view are denoted as p_2 and p_3 respectively. O_2 and O_3 are the optical centers (orange) of left color camera and right color camera, O_0 (orange) is the optical center of rectified camera coordinate system (gray-scale camera 0), and O_R (green) is the optical center of reference camera coordinate system. Rotation from reference camera system (green) to rectified camera system (orange) is denoted as R_{R0}.

(1) One-to-one onto mapping. $\exists \mathscr{S}_2^{'} \subseteq \mathscr{S}_2$ and $\exists \mathscr{S}_3^{'} \subseteq \mathscr{S}_3$ subject to, $\forall l_i \in \mathscr{S}_2^{'}$, $\exists r_j \in \mathscr{S}_3^{'}$ such that $r_j^T F l_i = 0$, and $\forall r_j \in \mathscr{S}_3^{'}$, $\exists l_i \in \mathscr{S}_2^{'}$ such that $r_j^T F l_i = 0$.

(2) Minimum intersection. Given conditions (1), suppose N_{sec} is number of inliers in the stereo-frustums intersection as shown in Fig. 3.1, $\forall l_i \in \mathscr{S}_2^{'}$, $\exists r_j \in \mathscr{S}_3^{'}$ subject to $r_j^T F l_i = 0$, then $N_{sec} = N(P_c(l_i) \cap P_c(r_j)) \geq N_{thres}$.

The one-to-one onto mapping constraint finds all spacial points that are forward-projected to matched the left and right RoIs, and the minimum intersection constraint is designed for the multi-modal regression network, ensuring that the regression will not fail due to too few spacial points as the input. It is widely known that the initial step after installing mounted devices is to get calibration information for all sensors including camera sensors, velodyne sensors, and IMU sensors. Then, fine-tune all sensors according to the calibration data until an acceptable error rate is observed. For the KITTI dataset, we assume the data collection system is well-calibrated, thus calibration files are reliable enough.

A necessary constraint for stereo matching based approaches is that, the stereo search can be performed horizontally. To relax this constraint, it is important to estimate the epipolar constraints from calibration files. In Section 3.3.3, we employ a searching algorithm based on the estimated epipolar constraints other than the horizontal search strategy.

Fundamental matrix. In KITTI dataset, the fundamental matrix between the left and the right image planes I_2, I_3 can be calculated as:

$$F = K_{c3}^{-T} [\mathbf{t}_{c2}^{c3}]_{\times} K_{c2}^{-1}, \tag{3.1}$$

where K_{c2}, K_{c3} are non-singular 3×3 intrinsic matrices of the left and right cameras respectively, $\mathbf{t}_{c3}^{c2}$ is a 3D translation vector[2] from the left to the right camera coordinate system (see Fig. 3.3), $[\cdot]_{\times}$ indicates the antisymmetric matrix of a vector. Using homogeneous coordinates of image points, $\forall p_3 \in I_3$ and its correspondent image point $p_2 \in I_2$ subject to $p_3^T F p_2 = 0$, by which the epipolar line through p_3 or p_2 can be deduced directly. To compute $\mathbf{t}_{c2}^{c3}$ of Eq. (6.1), according to the KITTI calibration methodology (46), we know the projection from a 3D point $\tilde{P}_0$ to its image $\tilde{p}_0$ in each image plane can be denoted as $\tilde{p}_i = [K_{ci}|\mathbf{C}_i]\tilde{P}_0 = P_{rect}^{(i)}\tilde{P}_0, i \in \{0,1,2,3\}$, where $\mathbf{C}_i$ is the last column of projection matrix $P_{rect}^{(i)}$, and i denotes the i-th camera: value '2' stands for the left camera, '3' for the right camera, and '0' for the gray-scale camera selected as the rectified camera. Then, we have the result

$$\mathbf{t}_{c2}^{c3} = K_{c2}^{-1}\mathbf{C}_2 - K_{c3}^{-1}\mathbf{C}_3. \tag{3.2}$$

Proof The coordinate systems of KITTI data-collection vehicle are depicted in Fig. 3.3. Assume P_0 is a inhomogeneous-coordinate point in rectified camera coordinate system, its images in left and right image planes are p_2 and p_3 (homogeneous-coordinates), respectively. According to the

[2]Each pair of subscript and superscript indicates the from-to relation, the same notation applies to other translation vectors and rotation matrices in this chapter.

pinhole model, we have:

$$p_3 = K_{c3}(R_{c0}^{c3}P_0 + \mathbf{t}_{c0}^{c3}) \Rightarrow p_3 = [K_{c3}R_{c0}^{c3}|K_{c3}\mathbf{t}_{c0}^{c3}]P_0, \tag{3.3}$$

where R_{c0}^{c3} and $\mathbf{t}_{c0}^{c3}$ are 3×3 rotation matrix and 3×1 translation vector from the rectified camera coordinate system to right camera coordinate system. Likewise,

$$p_2 = K_{c2}(R_{c0}^{c2}P_0 + \mathbf{t}_{c0}^{c2}) \Rightarrow p_2 = [K_{c2}R_{c0}^{c2}|K_{c2}\mathbf{t}_{c0}^{c2}]P_0. \tag{3.4}$$

According to the projection matrix $P_{rect}^{(i)}$ mentioned above, we know $p_i = [K_{ci}|\mathbf{C}_i]P_0$. Comparing the projection matrices $[K_{c2}|\mathbf{C}_i]$ and $[K_{c3}|\mathbf{C}_2]$ with the projection matrices in Eq. (6.3) and Eq. (6.4), respectively, we have the following observations:

$$R_{c0}^{c3} = R_{c0}^{c2} = I \;\Rightarrow\; R_{c2}^{c3} = I,\; \mathbf{t}_{c0}^{ci} = K_{ci}^{-1}\mathbf{C}_i, i = 2,3, \tag{3.5}$$

which indicates the rotation between the left camera and right camera can be ignored. It can be derived from Eq. (6.5) that

$$\mathbf{t}_{c2}^{c3} = \mathbf{t}_{c0}^{c2} - \mathbf{t}_{c0}^{c3} = K_{c2}^{-1}\mathbf{C}_2 - K_{c3}^{-1}\mathbf{C}_3, \text{ QED.} \tag{3.6}$$

It is necessary to prove Equ. (6.1) as well. Consider Eq. (6.3), Eq. (6.4) again, we have

$$K_{c3}^{-1}p_3 = R_{c0}^{c3}P_0 + \mathbf{t}_{c0}^{c3},\; K_{c2}^{-1}p_2 = R_{c0}^{c2}P_0 + \mathbf{t}_{c0}^{c2}, \tag{3.7}$$

and by using the observation $R_{c0}^{c3} = R_{c0}^{c2} = I$ from Eq. (6.5) and eliminating P_0, we have

$$K_{c3}^{-1}p_3 - \mathbf{t}_{c0}^{c3} = K_{c2}^{-1}p_2 - \mathbf{t}_{c0}^{c2} \;\Rightarrow\; K_{c2}^{-1}p_2 - K_{c3}^{-1}p_3 = \mathbf{t}_{c2}^{c3}. \tag{3.8}$$

Premultiply $[\mathbf{t}_{c2}^{c3}]_\times$ to both sides of Eq. (6.8):

$$[\mathbf{t}_{c2}^{c3}]_\times K_{c2}^{-1} p_2 = [\mathbf{t}_{c2}^{c3}]_\times K_{c3}^{-1} p_3, \tag{3.9}$$

by premultiplying $(K_{c3}^{-1} p_3)^T$ to both sides of Eq. (6.9), we have

$$(K_{c3}^{-1} p_3)^T [\mathbf{t}_{c2}^{c3}]_\times K_{c2}^{-1} p_2 = 0. \tag{3.10}$$

which can be rearranged as

$$p_3^T (K_{c3}^{-T} [\mathbf{t}_{c2}^{c3}]_\times K_{c2}^{-1}) p_2 = 0. \tag{3.11}$$

It is obvious that $F = K_{c3}^{-T} [\mathbf{t}_{c2}^{c3}]_\times K_{c2}^{-1}$, and $\mathbf{t}_{c2}^{c3}$ can be calculated by Eq. (6.6), QED.

3.3.2 Pipeline For Stereo Frustums

Each matched RoIs pair from left-right views encodes dense epipolar constraints between pixels and their corresponding 3D spacial points. Though fewer LiDAR points are forward-projected to front-view, we observe that ∼10,000 points are captured by most bboxes. Among these points, there are many outliers that do not belong to the interested objects. As pointed out by Zhou *et al.* (237) that it enforces high computational cost to FPN, our matching module effectively filters out 15% to 49% of points by dense epipolar constraints and thus can greatly speed up the networks that process point cloud.

The proposed Siamese pipeline is shown in Fig. 3.2. The stereopsis is fed to a 2D object detector. The detector can be any of traditional ones, such as the CNN-based detectors as Faster R-CNN (153), RetinaNet (103), the leading PC-CNN-V2 (36) on KITTI object detection benchmark, and others (114; 94). An accurate detector is crucial before populating the RoIs pairs into the matching module. Theoretically, each RoI in the left view should have its counterpart in the right view since one-to-one onto mapping constraint will be inconsistent otherwise. Also, If one salient object fails to be detected simultaneously in both views, it would lower both detection precision

and recall. Note that only the objects in the left view have their ground truths, thus we focus on matching RoIs of the right view to the left RoIs. The matching module follows the one-to-one mapping constraint or its relaxed form and takes all right RoIs as candidates. The goal is to find all matches to objects in the left view by the matching costs, which is illustrated in Section 3.3.3.

In the light of valid pairs proposed by the matching module and forward-projection matrices, the scene is segmented into 3D RoIs subject to minimum intersection constraint, whose semantic information and objectness scores are inherited from the 2D detections. A segmentation network refines the 3D RoIs with the extracted point-wise features. Furthermore, multi-modal regressor estimates the orientation, sizes of 3D bbox, and localization. To verify the validity of SFPN, we employ Faster R-CNN and Mask R-CNN (57) as the 2D detectors, FPNv2 as the point cloud segmentation and multi-modal regression networks.

3.3.3 Matching Algorithms

Notations. Consider the centers of bboxes as potential matches, let D_{thres} be the threshold of distance to the epipolar line, P_{thres} the threshold of regional similarity, P_{l_i} is the RoI image patch centered at l_i, $Prob_{ij} = NCC(P_{l_i}, P_{r_j})$ denotes the normalized cross-correlation operator, $dist(r_j, e_i)$ is the operator to compute the distance from r_j to the epipolar line e_i of l_i, $IoU_{ij} = N(P_c(l_i) \cap P_c(r_j))/N(P_c(l_i) \cup P_c(r_j))$ as 3D IoU matching cost, P_{3d_thres} the probability threshold of the 3D IoU cost. We then formulate four matching algorithms as:

(1) RoIs matching by 3D IoU cost and epipolar line search (3DCES, Alg. 8), which relaxes the one-to-one onto mapping constraint to left view only for alleviating computational burden.
(2) RoIs matching by 3D IoU cost and method of exhaustion (3DCME, Alg. 9), which relaxes the one-to-one onto mapping constraint.
(3) RoIs matching by regional similarity cost (RSC, Alg. 10), which relaxes the one-to-one onto mapping constraint.
(4) RoIs matching by regional similarity cost and left-right consistency check (RSCCC, Alg. 11), which strictly follows the one-to-one onto mapping constraint.

The epipolar line constraint facilitates searching for a fast pixel-level matching. As for RoI-level matching, we adapt this approach for faster search to RSC, RSCCC, and 3DCES. Experiments on both regional similarity and 3D IoU costs show the effectiveness of epipolar line search. The matching module is designed using Python without any Cython and GPU acceleration, or paralleled programming. On account of the compatibility purpose, 3DCME and 3DCES merely return matches as RSC and RSCCC do. We will show in the experiments that SFPN directly processes raw point cloud with high efficiency, and RSC achieves the shortest runtime among the four modules. As previous works done by Li *et al.* (95) and Qin *et al.* (146) have designed learning-based methods to find RoIs matches as RSC and RSCCC, we are seeking the possibilities to design learning-based 3DCME and 3DCES.

In order to deploy the most appropriate module for different tasks, it is recommended to inspect the groud-truth of the labeled dataset. In terms of a stereo camera configuration, if the groud-truths of both views are provided, then RSCCC may be implemented for the reason that 2D detection in both views can be described as 'reliable', which reduces unreliable matches by left-right consistency check while maintains high efficiency in RoIs matching. If the groud-truth of one view is unavailable, the view without ground-truth is thereby unreliable. In this case, although implementation of RSC is the fastest in runtime, 3DCES may be the best choice as a trade-off of runtime and detection performance. Moreover, 3DCME is better implemented in case of a very sparse LiDAR scene for its ability to find accurate matches of frustums. Reader may refer to Section 3.4.2 for comparisons on the performance of the proposed matching modules and Section 3.4.4 for details on the runtime analysis.

3.4 Experiments

In this section, we present both validation and testing results on publicly available KITTI dataset. The dataset consists of 7,481 annotated (2D/3D bbox labeled in left view only) image pairs with corresponding point clouds and calibrations, 7518 unlabeled testing image pairs with corresponding point clouds and calibrations. In our experiments, 20% of 7,481 randomly shuffled training

Algorithm 8: 3DCES

Require:
$\mathscr{S}_2, \mathscr{S}_3, P_{thres}, D_{thres}, P_c$;
Ensure:
1: Compute set of epipolar lines $\mathscr{E} = \{Fl_i | \forall l_i \in \mathscr{S}_2\}$;
2: $\forall l_i \in \mathscr{S}_2$, search leftwards in right view along $e_i \in \mathscr{E}$, calculate $\mathscr{R}_i = \{r_j | \forall r_j \in \mathscr{S}_3,\ dist(r_j, e_i) < D_{thres}\}$;
3: Compute costs $\mathscr{C}_i = \{IoU_{ij} | \forall r_j \in \mathscr{R}_i, \exists IoU_{ij} \neq 0\}$;
4: Find match. If $\mathscr{C}_i \neq \varnothing$ and $\exists r_j \in \mathscr{R}_i$ s.t. $IoU_{ij} \geq max(P_{3d\_thres}, max(\mathscr{C}_i))$, let $m_i = (l_i, r_i)$;
5: **return** $\mathscr{M}$;

Algorithm 9: 3DCME

Require:
$\mathscr{S}_2, \mathscr{S}_3, P_{3d\_thres}, P_c$;
Ensure:
1: $\forall l_i \in \mathscr{S}_2$, let $\mathscr{C}_i = \{IoU_{ij} | \forall r_j \in \mathscr{S}_3, \exists IoU_{ij} \neq 0\}$;
2: Find match. If $\mathscr{C}_i \neq \varnothing$ and $\exists r_j \in \mathscr{S}_3$ s.t. $IoU_{ij} \geq max(P_{3d\_thres}, max(\mathscr{C}_i))$, let $m_i = (l_i, r_i)$;
3: **return** $\mathscr{M}$;

samples are divided as the validation set, 2 categories of objects - 'Car' and 'Pedestrian' are examined for 3D/BEV detections. Also, we compare proposed SPFN with the state-of-the-arts including Pseudo-LiDAR (207), Stereo R-CNN (95), TLNet (146), et. al.

3.4.1 Experiments Setup

The 2D detector Faster R-CNN (153) (VGG-16 backbone) is trained on KITTI training set to provide shared weights and parameters for the sibling branches as shown in Fig. 3.2, and Mask

Algorithm 10: RSC

Require:
$\mathscr{S}_2, \mathscr{S}_3, P_{thres}, D_{thres}$;
Ensure:
1: Compute set of epipolar lines $\mathscr{E} = \{Fl_i | \forall l_i \in \mathscr{S}_2\}$;
2: $\forall l_i \in \mathscr{S}_2$, search leftwards in right view along $e_i \in \mathscr{E}$, calculate $\mathscr{R}_i = \{r_j | \forall r_j \in \mathscr{S}_3,\ dist(r_j, e_i) < D_{thres}\}$;
3: Perform RoI alignment to P_{l_i} and P_{r_j}, compute set of costs $\mathscr{C}_i = \{Prob_{ij} | \forall r_j \in \mathscr{R}_i, \mathscr{R}_i \neq \varnothing\}$;
4: Find match. If $\mathscr{C}_i \neq \varnothing$ and $\exists r_j \in \mathscr{R}_i$ s.t. $Prob_{ij} \geq max(P_{thres}, max(\mathscr{C}_i))$, let $m_i = (l_i, r_i)$;
5: **return** $\mathscr{M}$;

Algorithm 11: RSCCC

Require:

$\mathscr{S}_2, \mathscr{S}_3, P_{thres}, D_{thres}$;

Matches of left-to-right RoIs pairs, $\mathscr{M}_l = \varnothing$;

Matches of right-to-left RoIs pairs, $\mathscr{M}_r = \varnothing$;

Ensure:

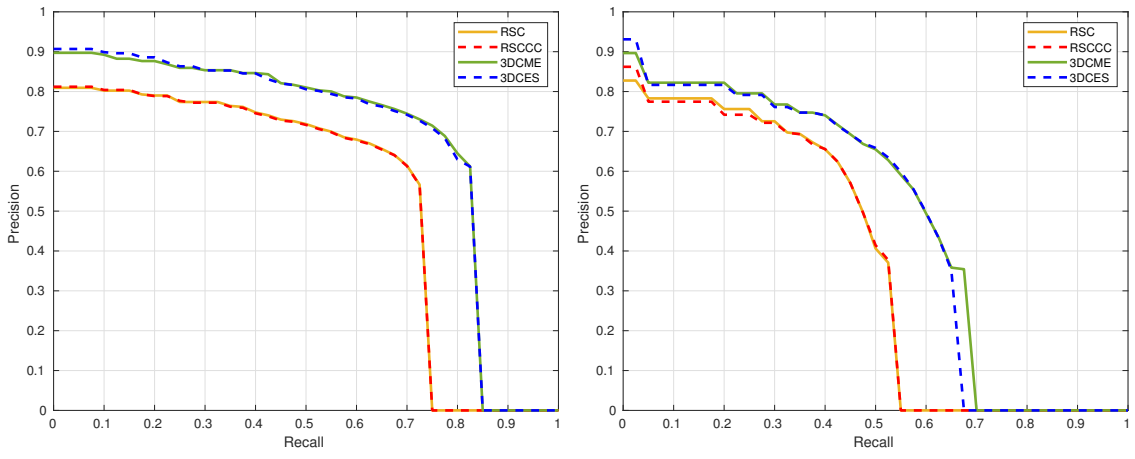
1: Compute sets of epipolar lines $\mathscr{E}_r = \{Fl_i | \forall l_i \in \mathscr{S}_2\}$ and $\mathscr{E}_l = \{r_i^T F | \forall r_i \in \mathscr{S}_3\}$;

2: $\forall l_i \in \mathscr{S}_2$, search leftwards in right view along $e_i \in \mathscr{E}_r$, calculate $\mathscr{R}_i = \{r_j | \forall r_j \in \mathscr{S}_3,\ dist(r_j, e_i) < D_{thres}\}$, $\forall r_i \in \mathscr{S}_3$, search rightwards in left view along $e_i \in \mathscr{E}_l$, calculate $\mathscr{L}_i = \{l_j | \forall l_j \in \mathscr{S}_2,\ dist(l_j, e_i) < D_{thres}\}$;

3: Perform RoI alignment to P_{l_i} and P_{r_j}, compute two sets of costs $\mathscr{C}_{li} = \{Prob_{ij} | \forall r_j \in \mathscr{R}_i, \mathscr{R}_i \neq \varnothing\}$ and $\mathscr{C}_{ri} = \{Prob_{ji} | \forall l_j \in \mathscr{L}_i, \mathscr{L}_i \neq \varnothing\}$;

4: Find matches from either view. If $\mathscr{C}_{li} \neq \varnothing$ and $\exists r_j \in \mathscr{R}_i$ s.t. $Prob_{ij} \geq max(P_{thres}, max(\mathscr{C}_{li}))$, let $\mathscr{M}_l = \mathscr{M}_l \cup \{(l_i, r_i)\}$. If $\mathscr{C}_{ri} \neq \varnothing$ and $\exists l_j \in \mathscr{L}_i$ s.t. $Prob_{ji} \geq max(P_{thres}, max(\mathscr{C}_{ri}))$, let $\mathscr{M}_r = \mathscr{M}_r \cup \{(l_j, r_i)\}$;

5: **return** $\mathscr{M} = \mathscr{M}_l \cap \mathscr{M}_r$;



(a) (b)

Figure 3.4: P-R curves of 3D object detection results. **Moderate** level is presented. (a) Car category at IoU threshold 0.7. (b) Pedestrian category at IoU threshold 0.5.

R-CNN (57) (ResNet backbone + Feature Pyramid Network) is pre-trained on COCO dataset. We train FPNv2 with raw LiDAR data other than segmented point cloud from the matching module. In this way, each module pairs with FPNv2 as 3DCES-FPNv2, 3DCME-FPNv2, RSC-FPNv2, RSCCC-FPNv2. Before testing, several parameters need to be manually set: the percentage of bbox size $S_{enlarge}$ to be enlarged, distance threshold D_{thres} from the bbox center to the epipolar line, similarity threshold P_{thres} for the template matching, the minimum number of inliers of stereo-frustums intersection N_{thres} and P_{3d_thres} the 3D IoU threshold. We set $S_{enlarge}$ to 8%, N_{thres} to 5, D_{thres}=30px, P_{thres}=0.4, and P_{3d_thres}=0.5. In this section, we enforce enlargement to all bboxes unless specified by 'NE' (w/o enlargement).

2D Detector	AP_{2D}	AP_{BEV}	AP_{3D}
Faster R-CNN(NE)	65.9	56.6	55.5
Mask R-CNN(NE)	85.8	74.2	73.5
Faster R-CNN	65.9	57.7	56.7
Mask R-CNN	85.8	83.0	82.2

Table 3.1: 3D object detection on car category using two 2D detectors.
APs(%) at **Moderate** level with IoU=0.5. Enlarged bboxes of both views by 8% on height and width. Only 3DCES-FPNv2 is tested.

It should be noted that the reported 2D detection results come from 2D detection stage other than the projection of the predicted 3D bounding box to the front view. Tab. 5.1 shows that when the 2D detection AP is low, enlarging bboxes slightly increases AP_{BEV} and AP_{3D} since the original bboxes are not precise enough to capture all keypoints. On the other hand, in terms of SFPN's serial pipeline structure, higher 2D detection AP may result in better 3D detection. We also observe that, if setting P_{3d_thres} to a larger threshold, the number of total matches drops greatly and vice versa, which behaves similar to 2D IoU during the detection phase.

3.4.2 Quantitative Evaluation on Validation Set

The results of BEV and 3D object detection are presented in Tab. 3.2, Tab. 3.3, Tab. 3.4, and Tab. 3.5. Both RSC and RSCCC require RoI alignment during matching. RSC is designed to relax the one-to-one onto mapping constraint by merely finding matches to RoIs in the left view while

RSCCC strictly follows this constraint. As shown in the detection results, RSCCC-FPNv2 achieves better performance than RSC-FPNv2 with IoU=0.7, which provides evidence for the effectiveness of this constraint. 3DCME depends on the proposed 3D IoU without RoI alignment or epipolar line search, which achieves better AP than 3DCES with epipolar line search. We will show in Section 3.4.4 that RSC runs faster than the other three matching methods, and 3DCME is the slowest but with the highest AP.

3.4.2.1 Overview on Comparisions

3DCES-FPNv2 and 3DCME-FPNv2 achieve competitive performance among all state-of-the-art stereo-based methods because of highly precise LiDAR data. Although almost all listed stereo-based methods yield higher 2D detection AP (IoU=0.7) on either the validation set or the test set, 3DCES-FPNv2 and 3DCME-FPNv2 regress disproportional accurate 3D bboxes when projected to BEV. It is interesting that RSCCC-FPNv2 and RSC-FPNv2 outperform their learning-based versions (without LiDAR data) Stereo-RCNN ($\sim$10%) and TLNet($\sim$35%) by a large margin.

Method	Type	IoU=0.5			
		AP_{2D}	Easy	Mode	Hard
RSC-FPNv2(**ours**)	Stereo+LiDAR	85.8	71.9	68.4	61.5
RSCCC-FPNv2(**ours**)	Stereo+LiDAR	85.8	71.9	68.2	61.4
3DCME-FPNv2(**ours**)	Stereo+LiDAR	85.8	82.9	82.9	75.7
3DCES-FPNv2(**ours**)	Stereo+LiDAR	85.8	**83.1**	**83.0**	**75.7**
PSMNET-AVOD (207)	Stereo	-	89.0	77.5	68.7
Stereo R-CNN (95)	Stereo	-	87.1	74.1	58.9
3DOP (25)	Stereo	-	55.0	41.3	34.6
TLNet (146)	Stereo	-	62.5	46.0	41.9

Table 3.2: BEV results on validation set.
Car category is evaluated with IoU=0.5 and all APs in '%'. **Moderate** AP_{2D} on validation set is reported.

3.4.2.2 Comparision with Pseudo-LiDAR

Tab. 3.6 shows the comparison with a recently published Pseudo-LiDAR (207) method. Compared with FPNv2 on validation set and the car category, 3DCES-FPNv2 and 3DCME-FPNv2 achieve

Method	Type	IoU=0.7			
		AP_{2D}	Easy	Mode	Hard
RSC-FPNv2(**ours**)	Stereo+LiDAR	54.2	67.1	58.0	51.3
RSCCC-FPNv2(**ours**)	Stereo+LiDAR	54.2	68.4	58.6	51.8
3DCME-FPNv2(**ours**)	Stereo+LiDAR	54.2	**79.8**	**72.9**	**65.7**
3DCES-FPNv2(**ours**)	Stereo+LiDAR	54.2	78.4	72.2	65.1
PSMNET-AVOD (207)	Stereo	-	74.9	56.8	49.0
Stereo R-CNN (95)	Stereo	**88.3**	68.5	48.3	41.5
3DOP (25)	Stereo	-	55.0	9.5	7.6
TLNet (146)	Stereo	-	29.22	21.9	18.8

Table 3.3: BEV results on validation set.
Car category is evaluated with IoU=0.7 and all APs in '%'. **Moderate** AP_{2D} on validation set is reported.

Method	Type	IoU=0.5			
		AP_{2D}	Easy	Mode	Hard
RSC-FPNv2(**ours**)	Stereo+LiDAR	85.8	70.7	67.1	54.0
RSCCC-FPNv2(**ours**)	Stereo+LiDAR	85.8	70.4	66.9	53.9
3DCME-FPNv2(**ours**)	Stereo+LiDAR	85.8	82.5	**82.3**	**75.0**
3DCES-FPNv2(**ours**)	Stereo+LiDAR	85.8	82.4	82.2	74.9
PSMNET-AVOD (207)	Stereo	-	**88.5**	76.4	61.2
Stereo R-CNN (95)	Stereo	-	85.8	66.3	57.2
3DOP (25)	Stereo	-	46.0	34.6	30.1
TLNet (146)	Stereo	-	59.5	43.7	38.0

Table 3.4: 3D detection results on validation set.
Car category is evaluated with IoU=0.5. 3DCME-FPNv2 achieves the best performance on validation set out of stereo-based methods.

Method	Type	IoU=0.7			
		AP_{2D}	Easy	Mode	Hard
RSC-FPNv2(**ours**)	Stereo+LiDAR	54.2	58.5	53.7	47.4
RSCCC-FPNv2(**ours**)	Stereo+LiDAR	54.2	58.7	53.9	47.5
3DCME-FPNv2(**ours**)	Stereo+LiDAR	54.2	**73.0**	**67.3**	**61.2**
3DCES-FPNv2(**ours**)	Stereo+LiDAR	54.2	72.5	66.7	60.9
PSMNET-AVOD (207)	Stereo	-	61.9	45.3	39.0
Stereo R-CNN (95)	Stereo	**88.3**	54.1	36.7	31.1
3DOP (25)	Stereo	-	6.6	5.1	4.1
TLNet (146)	Stereo	-	18.2	14.3	13.7

Table 3.5: 3D detection results on validation set.
Car category is evaluated with IoU=0.7. 3DCME-FPNv2 achieves the best performance on validation set out of stereo-based methods.

competitive results although they suffer from lower recall (see Fig. 3.4a) considering mismatches in both views. In order to explore potentials of proposed method, we specifically select PSMNET-AVOD that achieves the best performance among the methods proposed in (207). As mentioned in Section 3.1, dense disparity map generates a coarse estimation of the scene which is sensitive to the focal length and baseline of the stereo cameras. According to Pseudo-LiDAR (207), its point cloud detector AVOD and FPN are trained by fine-grained, sparse LiDAR points, therefore its performance may not compete with the state-of-the-art methods based on real-LiDAR data.

Method	AP_{2D}	Easy	Mode	Hard
3DCME-FPNv2(**ours**)	57.4	57.6	47.1	40.7
3DCES-FPNv2(**ours**)	57.4	**57.8**	47.3	40.8
FPNv2(**our results**)	57.4	55.6	**49.1**	**42.6**
PSMNET-FPN (207)	-	33.8	27.4	24.0
FPN (207)	-	64.7	56.5	49.9

Table 3.6: 3D detection results on validation set.
Pedestrian category is evaluated.

Method	AP_{2D}	Easy	Mode	Hard
3DCME-FPNv2(**ours**)	57.4	**62.6**	**53.7**	**46.6**
3DCES-FPNv2(**ours**)	57.4	60.8	52.8	45.9
FPNv2(**our results**)	57.4	58.5	51.5	44.8
PSMNET-FPN (207)	-	41.3	34.9	30.1
FPN (207)	-	69.7	60.6	53.4

Table 3.7: BEV detection results on validation set.
Pedestrian category is evaluated. Results of FPN in the table are presented in (208).

Tab. 4.7 shows the BEV results compared to FPNv2 on validation set, along with another similar comparison with (207). 3DCME-FPNv2 achieves better performance than FPNv2 by 4.1% at Easy level, 2.2% at Moderate level, and 1.8% at Hard level. The improvements are mainly derived from enlarged bboxes since they enrich sparse segmentation captured by stereo frustums. These enlarged bboxes are especially efficient for pedestrians whose poses vary. For 3D object detection, 3DCES-FPNv2 achieves higher AP at Easy level, but is less precise than FPNv2 at Moderate and Hard levels. To conclude, when comparing margins to corresponding FPN detection

results in Tab. 4.7 and Tab. 3.6, our methods outperform pseudo-LiDAR based PSMNET-FPN by significant margins.

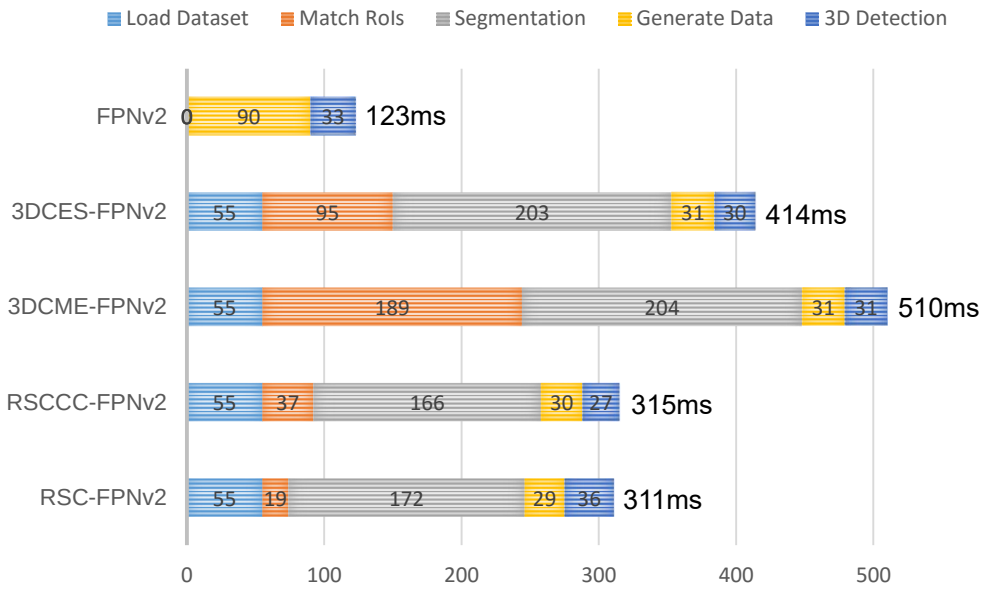
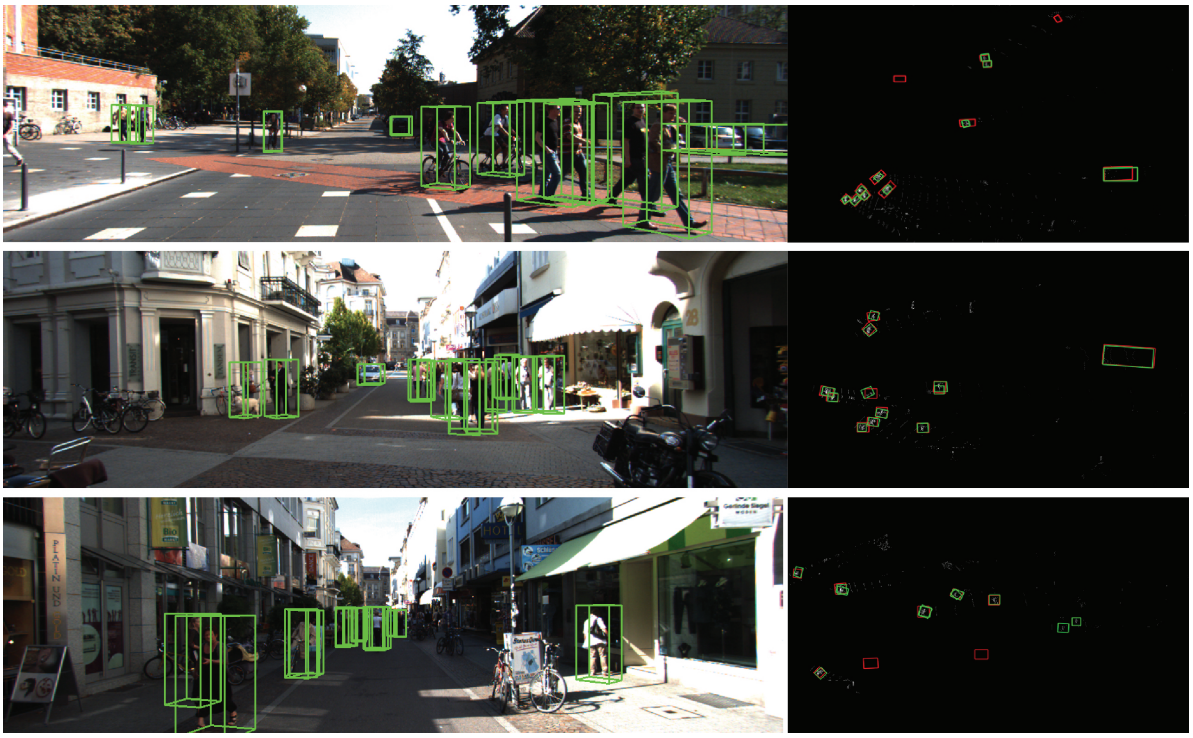


Figure 3.5: Runtime comparison between matching modules and original FPNv2. The figure is best viewed in colour mode. Processing time (millionsecond) is calculated by averaging runtime of validating 1,496 samples.

3.4.3 Quantitative Evaluation on Test Set

In this subsection, we present the performance of 3DCES-FPNv2 on KITTI leaderboard (Geiger et al.) among all stereo-based submissions. We take Mask R-CNN (57) as the 2D detector as it has a higher detection AP over Faster R-CNN (see Section 3.4.1). FPNv2 is trained on our training set other than all 7,481 samples. Tab. 3.8 and Tab. 3.9 present 3D object detection results, Tab. 3.10 and Tab. 3.11 present BEV detection results.

It can be observed that from Tab. 3.9 and Tab. 3.11, proposed method outperforms stereo-based method in detecting pedestrians. However, as there are more samples in the cyclist category than that of the validation set, while Mask R-CNN mistakes the cyclist category as the pedestrian category, we believe this is one of the main factors that lower the detection APs. In terms of the performance in 3D object detection, 3DCES-FPNv2 achieves competitive performance among



(a) 3D(left) and BEV(right) detections. More persons than cars.

(b) 3D(left) and BEV(right) detections. **Car** category only.

Figure 3.6: Qualitative evaluation on validation set using Mask R-CNN detector. Red bboxes in BEV indicate ground truths, and green bboxes indicate detections. Cyclist category is not evaluated in our experiments. However, Mask R-CNN (57) misclassifies cyclists as pedestrians, therefore significantly increases false positive rate in detecting pedestrians.

stereo-based methods. It should be noted that untrained Mask R-CNN provides much lower 2D detection AP at 46.68% which is 38.47% less than 85.15% of PL++(SDN+GDC) (221). Nonetheless, the proposed method maintains disproportionate performance in 3D object detection, which proves the effectiveness of the proposed pipeline.

Method	Type	AP_{2D}	Easy	Mode	Hard
3DCES-FPNv2(**ours**)	Stereo+LiDAR	**46.68**	**58.88**	**51.92**	**44.59**
Pseudo-LiDAR (207)	Stereo	67.79	54.53	34.05	28.25
Pseudo-LiDAR++ (221)	Stereo	82.90	61.11	42.43	36.99
PL++(SDN+GDC) (221)	Stereo+LiDAR	**85.15**	**68.38**	**54.88**	**49.16**
ZoomNet (213)	Stereo	83.92	55.98	38.64	30.97
Stereo R-CNN (95)	Stereo	85.98	47.58	30.23	23.72
StereoFENet (9)	Stereo	85.70	29.14	18.41	14.20
OC Stereo (141)	Stereo	74.60	55.15	37.60	30.25
RT3DStereo (84)	Stereo	45.81	29.90	23.28	18.96
TLNet (146)	Stereo	63.53	7.64	4.37	3.74

Table 3.8: 3D detection APs(%) on test set(Geiger et al.).
Car category is evaluated. **Moderate** 2D detection APs are reported. With much lower 2D detection AP, proposed method has the potential to outperform PL++(SDN+GDC).

Method	Type	AP_{2D}	Easy	Mode	Hard
3DCES-FPNv2(**ours**)	Stereo+LiDAR	**51.83**	**37.16**	**29.77**	**26.61**
OC Stereo (141)	Stereo	30.79	29.79	20.80	18.62
RT3Dstereo (84)	Stereo	29.30	4.72	3.65	3.00

Table 3.9: 3D detection APs(%) on test set (Geiger et al.).
Pedestrian category is evaluated.

3.4.4 Runtime

With a fixed number of LiDAR points (e.g., 1024) fed into FPNv2, 3D detection phases have almost the same computation costs (see Fig. 3.5). Therefore, it is necessary to show runtime efficiency of the stereo frustums pipeline. As depicted in Fig. 3.5, all shown phases are tested with a 2.5Ghz CPU except for the 3D detection phase which is tested with a P100 GPU. RSC-FPNv2 is the fastest among the four methods, RSCCC-FPNv2 doubles the matching time of RSC-FPNv2 but only slightly increases the overall preparation time, 3DCME significantly increases RoIs matching

Method	Type	AP_{2D}	Easy	Mode	Hard
3DCES-FPNv2(**ours**)	Stereo+LiDAR	**46.68**	**74.20**	**65.74**	**58.35**
Pseudo-LiDAR (207)	Stereo	67.79	67.30	45.00	38.40
Pseudo-LiDAR++ (221)	Stereo	82.90	78.31	58.01	51.25
PL++(SDN+GDC) (221)	Stereo+LiDAR	**85.15**	**84.61**	**73.80**	**65.59**
ZoomNet (213)	Stereo	83.92	72.94	54.91	44.14
Stereo R-CNN (95)	Stereo	85.98	61.92	41.31	33.42
StereoFENet (9)	Stereo	85.70	49.29	32.96	25.90
OC Stereo (141)	Stereo	74.60	68.89	51.47	42.97
RT3Dstereo (84)	Stereo	45.81	58.81	46.82	38.38
TLNet (146)	Stereo	63.53	13.71	7.69	6.73

Table 3.10: BEV detection APs(%) on test set (Geiger et al.).
Car category is evaluated.

Method	Type	AP_{2D}	Easy	Mode	Hard
3DCES-FPNv2(**ours**)	Stereo+LiDAR	**51.83**	**31.61**	**24.84**	**21.96**
OC Stereo (141)	Stereo	30.79	24.48	17.58	15.60
RT3Dstereo (84)	Stereo	29.30	3.28	2.45	2.35

Table 3.11: BEV detection APs(%) on test set (Geiger et al.).
Pedestrian category is evaluated.

time while 3DCES reduces the time almost by half. Thus, 3DCES is efficient in matching as well as maintaining detection AP as 3DCME does. Most runtime of data-preparation phase goes to point cloud processing, which remains to be optimized by high-performance computing techniques.

3.4.5 Qualitative Results

The results of 3D detection and BEV detection using SPFN on the validation set are visualized in Fig. 3.6. The BEV detections are depicted on a sparse point cloud generated by the 3DCES matching module. Fig. 3.6a shows that two pedestrians are missed due to unsatisfying illumination condition that invalids the 2D detector, another two pedestrians are missed due to large occlusion and too small in dimensions. Despite the missed detections, SFPN is capable of regressing highly precise bboxes that enclose irregular objects as pedestrians. There is a very close object in Fig. 3.6b which is not detected due to too few clues of objectness in that region. Also, largely occluded objects in Fig. 3.6b can hardly be detected. Nevertheless, RoIs locate both very near and near

(around 40m, which almost reaches the valid-range limit of LiDAR sensor) objects precisely. Some faraway objects located at 50-70m can be detected by SFPN but with less precision. To further study segmented point cloud of the faraway objects, the number of points in each segmented point cloud is usually less than 100, we believe this number is around the borderline threshold N_{thres}.

3.5 Conclusion

In this chapter, we have proposed four matching modules to bridge the gap between 2D object detection on stereopsis and real LiDAR data via dense epipolar geometry constraints: the one-to-one onto mapping and minimum intersection. To accommodate the proposed matching modules, we have proposed a stereo frustum pipeline for 3D object detection where the 2D detection results are fed to the matching module to generate matches to segment the point cloud of the scene, and the 3D segmentation proposals are then fed to a refinement network for more precise objectness segmentation, followed by a multi-modal regression network.

By integrating with the F-PointNets, our stereo frustum pipeline can achieve 2-3 frames per second without coding optimization. Although this frame rate is not yet applicable for real-time applications, its speed can be further increased if we adopt techniques like GPU acceleration. More efficient matching algorithms and 2D detection models are also expected. The proposed pipeline outperforms the state-of-the-art stereo-based approaches with a lower 2D detection average precision, it has the potential to outperform the state-of-the-art LiDAR and stereo fusion approaches if better 2D detection models are adopted.

We are currently working to design end-to-end matching modules for 3DCES and 3DCME to achieve more effective representation to encode the sparse point cloud. Some future work includes leveraging the reliability of both views for better performance in the detection accuracy and recall, optimizing runtime not only at the coding level, but also seeking into the possibilities of implementing distributed parallel computing techniques.

Chapter 4

Inference: 2D Object Detection Using Faster R-CNN

Abstract

Polyp has long been considered as one of the major etiologies to colorectal cancer which is a fatal disease around the world, thus early detection and recognition of polyps plays an crucial role in clinical routines. Accurate diagnoses of polyps through endoscopes operated by physicians becomes a chanllenging task not only due to the varying expertise of physicians, but also the inherent nature of endoscopic inspections. To facilitate this process, computer-aid techniques that emphasize on fully-conventional image processing and novel machine learning enhanced approaches have been dedicatedly designed for polyp detection in endoscopic videos or images. Among all proposed algorithms, deep learning based methods take the lead in terms of multiple metrics in evolutions for algorithmic performance. In this work, a highly effective model, namely the faster region-based convolutional neural network (Faster R-CNN) is implemented for polyp detection. In comparison with the reported results of the state-of-the-art approaches on polyps detection, extensive experiments demonstrate that the Faster R-CNN achieves very competing results, and it is an efficient approach for clinical practice[1].

4.1 Introduction

It is well known that the predecessor of colorectal cancer (CRC), also termed as colon cancer, is most likely to be a polyp. According to the statistics of American Cancer Society, colorectal

[1] Relevant works are published as (125).

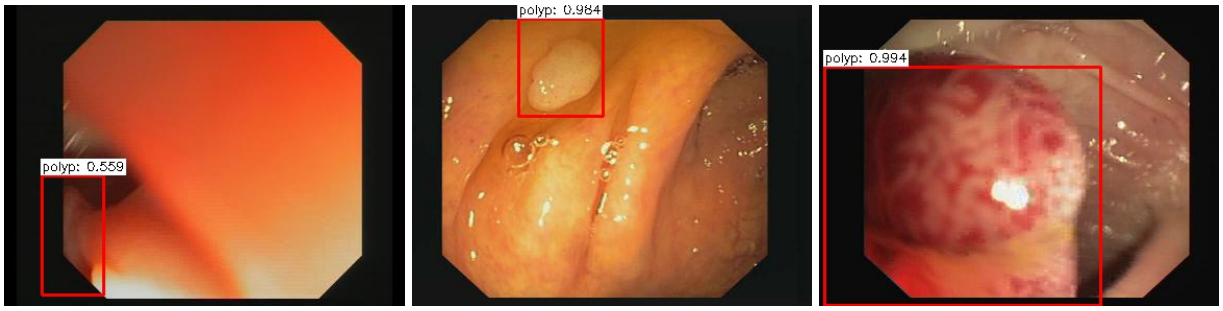
carcinoma is the third most commonly diagnosed cancer and the second leading cause of death from cancers in the United States (203). CRC is the fourth cause of cancer death worldwide with around 750,000 new cases diagnosed in 2012 alone (49).

Pathologically, neoplastic polyps may chronically turn into cancer, located hiddenly on colorectal wall unless filmed during colonoscopy, which is the main diagnostic procedure of doctors. Though this process may be intuitively achieved successfully, approximately 25% of polyps are missed (189), which brings about potential risks to patients' lives. For early diagnoses and prevention of colon cancer, an urgent task for physicians and computer vision researchers is to find more reliable, accurate and even faster approaches for polyp detection. In response to the demands, well-designed grand challenges organized by Medical Image Computing and Computer Assisted Intervention (MICCAI) and International Symposium on Biomedical Imaging (ISBI), have attracted a lot of attention worldwide.

Specifically, multiple factors affect either the process of manual inspection or computer-aided detection significantly. For the first and foremost, it is common that during the clinical practices, physicians usually operate conventional colonscope for hours to seek, observe, and diagnose polyps, when considering the heavy workload of physicians that leads to both mental and physical fatigue, even an experienced doctor would miss or wrongly diagnose benign polyps. Therefore, automatic computer-aid system is urgently in modern medicine communities (5; 75).

In regard to computer-aid methods, the factors are diverse. There are varieties of noises in the videos which can be classified as the specular highlights caused by illumination along with the non-Lambertian colorectal walls, the curving veins distributed around the polyps, the polyp like bulges on internal wall to lumen, blob-like matters such as bubbles that always being observed, and the insufficient illumination that shield all regions of interest (ROI). These noises may invalidate the state-of-the-art conventional and learning-based approaches (77; 190; 189). Our experiments show that, in some rare cases, Faster R-CNN (154) may mistake some oval specular highlights for polyps.

Another factor is the shape information. Polyps are not always appeared as regular oval lumps,



(a) (b) (c)

Figure 4.1: Features of Faster R-CNN detector.
(a) Successfully detect a largely occluded (approximately 50%) polyp with low intensities. (b) Blob-like objects as bubbles are neglected. (c) Very large polyp detected.

furthermore, they can be various in their sizes from 3*mm* to more than 10*mm* or more variable due to projective transformation and distortions of imaging sensors. Conventional hand-crafted approaches and some fusion approaches often suffer from this factor in that they are initially designed according to the morphological features of polyp (203; 189; 190; 216; 119).

Bernal *et al.* (14) categorize off-the-shelf methods for polyp detection into three classes: hand-crafted, hybrid, and end-to-end learning. Our work emphasizes on the deep learning solution to polyp detection, and provide evaluation of variations in parameters. Our contributions include:

- To our best knowledge, this work provides the first evaluation for polyp detection using Faster-RCNN framework. In addition to reducing the false positive rate during the test phase whose goal is to lower the risks for misdiagnosis when taking the detector for clinical practice, our system provides a good trade-off between efficiency and accuracy.

- We demonstrate a fine-tuned set of parameters for polyps detection in endoscopic videos that outperform many state-of-the-art methods. The testing results set a novel baseline for polyp detection.

- We compare and analyze the experimental results and reveal insights for better solution when deal with small dataset consisted of endoscopic videos. The proposed framework together with the trained parameters are available for the research community on the author's website.

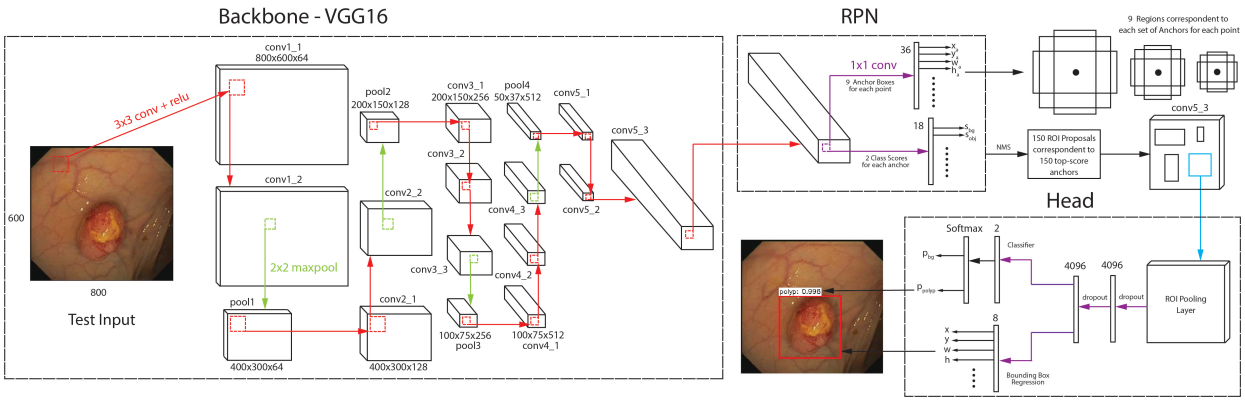


Figure 4.2: Structure for polyp detection.
The image is best viewed in its colored version that the red arrow signifies the convolution-ReLU flow, green arrow the max pooling flow, purple arrow the 1×1-convolution or fully-connected flow, blue arrow the ROI pooling flow, and black arrow represents the normal datum flow. Noted that all operations of the same sized convolutional kernel is only labeled at the first appearance of that flow. The input frame is selected from new CVC-ClinicDB (CVC-ClinicDB2017), resizing to 800×600.

4.2 Related Works

According to MICCAI 2015 challenge evaluations, fully CNN based methods with or without data augmentation outperform fusion methods and hand-crafted when considering the evaluation metrics in most cases: Recall, Precision, F-scores (e.g. CUMED, OUS in all videos and videos with only polyp frames). However, high false positive rate has been observed during the experiments (158) that a novel data augmentation technique - random view aggregation is implemented, while for pursuing the highest F-scores and remedying the deficiencies of 2D-CNNs, online and offline 3D-fully convolutional networks (FCNs) are integrated to acquire the final confidence map (223). For 2D-CNNs, most of related works focus on no more than 5-convolutional layered deep CNNs such as AlexNet (191), but a few (158) have experimented on deeper networks. It remains to be a key topic whether light weighted CNNs can achieve the same capacity as their very deep counterparts. Still, we believe that a trade-off between architectural complexity and runtime would contribute to the ideal design, which is the main reason that we choose VGG16 (177), once achieved 92.7% top-5 test accuracy in ImageNet dataset as the feature extractor.

To the best of our knowledge, Faster-RCNN is the first detector so far that replaces hand-crafted ROI selection step with a network i.e., the regional proposal network (RPN) towards fully end-to-end fashion. Its structure is developed from previous R-CNN (52) and Fast-RCNN (51). Recently, an improvement of Faster R-CNN, i.e., the Mask R-CNN (58) is proposed by extending a novel multi-task branch: mask sub-network for segmentation purpose along with replacement of ROI Pooling layer by ROI Align layer. We apply Faster R-CNN without the sub-network for its redundancy in detecting polyps.

Other novel end-to-end detectors such as You Only Look Once (YOLOv1) (82), YOLOv2 (152), SSD (104), so far, most of them have been implemented and tested on other public or private datasets such as COCO, ImageNet, etc. Although these approaches could fulfill realtime requirements (up to more than 24fps), the ROIs are randomly chosen without an end-to-end fashion, and the mAP is compromised in terms of polyp detection as reported in (140) that examines YOLOv1 on ASU-Mayo Clinic dataset(190).

4.3 Architecture of Faster R-CNN

4.3.1 Backbone Structure

Fig. 4.2 illustrates the complete testing structure of this work. The backbone (58) computes high-level features of entire test frame such that the weights between ROIs are shared, which is different from previous R-CNN and patchwise OUS (14) methods. Faster R-CNN removes all subsequent layers of 512 feature maps *conv5_3* whose shape is 50×37 for each. In reference to VGG16 and ZFnet, it is reported that the latter runs faster up to 17fps, while the former runs at 5fps (154), on a K40 GPU. When comparing mAPs on PascalVOC 2007, ZFnet backbone achieves highest 59.9%, and VGG16 78.8%. VGG16 thus benefits for its deep feature extraction process besides its relatively high speed compared to CUMED (14) that runs at 5fps on a more advanced TitanX GPU for former CVC-ClinicDB (CVC-ClinicDB2015) (12).

4.3.2 RPN and Head Networks

The *conv*5_3 is fed to two sibling branches - RPN and Head (58). After performing 3×3 convolution, RPN constructs 9 anchors at each position on the resulted feature map, the anchors are designed according to 3 scales (small, medium, large) with 3 different ratios of 1:1, 1:2, 2:1. As a result, it outputs maximum 50×37×9×4=66600 positional coordinates of all 16650 potential proposals (for each proposed, the coordinates in the test image are the center (x_a, y_a), width w_a and height h_a of the bounding box), and 50×37×9×2=33300 scores per proposal being the background or polyp. During the training, not all proposals are transformed to training samples, of which a limited number of refined proposals e.g. 2000, are selected by trimming invalided bounding boxes along the borders; proposals with intersection of union (IoU) between 0.3 and 0.7, and in the meantime, keep as many positive samples (IoU>0.7) as possible, and replenish with negative samples (IoU<0.3); and applying non-maximum suppression (NMS) to the scores S_{bg} and S_{obj}, as depicted in Fig. 4.2.

During the testing process, we let RPN generate 150 top proposals further trimmed by NMS of the scores s_{obj} and s_{bg}, afterall, RPN is trained for valid regional proposals better than its counterpart - selective search. Refined candidates are then mapped to anchors on *conv*5_3. The Head network leverages on each anchor to yield the detection outcomes.

As shown in Fig. 4.2, the blue arrow represents the ROI pooling process. All 150 anchors are resized to the same size, which is equivalent to a single-layered SPPnet (59). This procedure is essential as it transforms different scaled feature map into the two following 4096 fix-length fully-connected layers, each is followed by a dropout layer with a probability of 0.5, which makes the softmax classifier applicable. In addition to regress bounding box of predicted ROIs (x, y, w, h), the Head output 2-class probabilities of the correspondent ROIs to be either background P_{bg} or polyp P_{polyp}.

4.3.3 Loss Function

Either RPN or the Head loss functions (154) of Faster R-CNN consists of two parts i.e., the classification loss L_{cls} and bounding box regression loss L_{reg}. Suppose the ground truth of a proposal to be $\{x^*, y^*, w^*, h^*, P_i^*\}$, among which $P_{bg}^* = 1$ and $P_{polyp}^* = 0$ if the proposal is positive, and $P_{bg}^* = 1$ and $P_{polyp}^* = 0$ if negative. To alleviate the influence of scales during training, the coordinates are parameterized as

$$\begin{cases} t_x = (x - x_a)/w_a, & t_y = (y - y_a)/h_a, \\ t_w = log(w/w_a), & t_h = log(h/h_a), \end{cases}$$
$$\begin{cases} t_x^* = (x^* - x_a)/w_a, & t_y^* = (y^* - y_a)/h_a, \\ t_w^* = log(w^*/w_a), & t_h = log(h^*/h_a), \end{cases} \tag{4.1}$$

and the general loss function is denoted as

$$L(\{P_{bg}, P_{polyp}\}, \{t_i\}) = \frac{1}{N_{cls}}[L_{cls}(P_{bg}, P_{bg}^*) + L_{cls}(P_{polyp}, P_{polyp}^*)] + \lambda \frac{1}{N_{reg}} \sum_i P_i^* L_{reg}(t_i, t_i^*), \tag{4.2}$$

where N_{cls} denotes the mini-batch size, N_{reg} the number of all proposals from an image for training. Here the classification loss $L_{cls}(P_i, P_i^*) = -P_i^* log(P_i)$, where $P_{poyp} + P_{bg} = 1$, P_{poyp} and P_{bg} are outputs of softmax classifier, and the bounding box regression loss $L_{reg}(t_i, t_i^*) = R(t_i - t_i^*)$, in which $R(\cdot)$ is smooth L_1 function for Head loss denoted as

$$R(x) = \begin{cases} 0.5x^2 & |x| < 1 \\ |x| - 0.5 & otherwise. \end{cases} \tag{4.3}$$

For joint training, the total loss is the sum of RPN and Head losses. while applying 4-step training, two losses are tuned alternately.

Table 4.1: Validation metrics for polyp detection.

	Polyp Detection
True Positive (TP)	Indicate polyp presence in a frame with polyp
False Positive (FP)	Indicate polyp presence in a frame without polyp
True Negative (TN)	Indicate polyp missing in a frame without polyp
False Negative (FN)	Indicate polyp missing in a frame with polyp
Precision	$100 \times \frac{\text{TP}}{\text{TP+FP}}$
Recall	$100 \times \frac{\text{TP}}{\text{TP+FN}}$
Accuracy	$100 \times \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}}$
F1-score	$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision+Recall}}$
F2-score	$5 \times \frac{\text{Precision} \times \text{Recall}}{4 \times \text{Precision+Recall}}$
Reaction Time (RT)	Delay between the first TP and polyp frame
Mean Distance(MD)	N/A

4.4 Implementation Details

4.4.1 Data Preparation

The framework is tested using the following public datasets tested during our experiments include:

- **CVC-Clinic2015 (CVC15)**. Contains 612 still frames whose ground-truths are labeled by the Computer Vision Center (CVC), Barcelona, Spain are selected from 29 endoscopic videos by courtesy of Hospital Clinic, Barcelona, Spain. This dataset is designed as the training set for MICCAI2015 and ISBI2015 sub-challenges for polyp detection in endoscopic videos.

- **CVC-Clinic2017**. A new database for MICCAI2017 endoscopic sub-challenge, which consists of 18 different sequences, and all of which showing no more than one polyp and have up to 11954 frames. The test set contains 18 different videos, and has up to 18733 frames.

- **CVC-ColonDB** (13). Small public dataset maintained by the CVC group, which contains 300 frames from 15 different videos along with their corresponding ground-truth masks, non-

Table 4.2: Validation metrics for polyp localization.

	Polyp Localization
True Positive (TP)	Correctly predict polyp location within polyp frame
False Positive (FP)	Wrongly predict polyp location within polyp frame
True Negative (TN)	N/A
False Negative (FN)	Indicate polyp missing in a frame with polyp
Precision	$100 \times \frac{\text{TP}}{\text{TP+FP}}$
Recall	$100 \times \frac{\text{TP}}{\text{TP+FN}}$
Accuracy	N/A
F1-score	$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision+Recall}}$
F2-score	$5 \times \frac{\text{Precision} \times \text{Recall}}{4 \times \text{Precision+Recall}}$
Reaction Time (RT)	N/A
Mean Distance(MD)	Mean Euclidean distance between polyp centers

informative region masks, contour of the polyp masks.

- **CVC-EndoSceneStill** (197). The CVC group combines CVC-ColonDB with CVC-ClinicDB2015 into a new dataset with explicit divisions for train, test, and validation respectively, which is composed of 912 frames obtained from 44 video sequences collected from 36 patients.

We randomly select 16 sequences from CVC-ClinicDB2017 training set for training Faster R-CNN. To test the performance of trained model on CVC-ColonDB, CVC-ClinicDB2015 and CVC-EndoSceneStill, only the training sets are chosen.

Only simple transformations are made to the raw images without augmentation. All training frames are resized to 384×288, which is close to original resolutions of samples for not incorporating much distortions, and in the validation set, monochrome tiff images from CVC-Clinic2015 are transformed to chromatic counterparts. In addition, the training samples are flipped horizontally.

Table 4.3: Fine-tuned detection results for 300 proposals on accuracy, precision and recall.

Dataset	TP	FP	TN	FN	Accuracy	Precision	Recall
CVC-Clinic2015-train	607	0	0	5	99.2	100.0	99.2
CVC-ColonDB	292	0	0	8	97.3	100.0	97.3
CVC-EndoSceneStill	181	0	0	2	98.9	100.0	98.9
Average	-	-	-	-	**98.5**	**100.0**	**98.5**

Table 4.4: Fine-tuned detection results for 300 proposals on F1-/F2-scores and RT.

Dataset	TP	FP	TN	FN	F1-score	F2-score	RT (in frame)
CVC-Clinic2015-train	607	0	0	5	99.6	99.3	0
CVC-ColonDB	292	0	0	8	98.6	97.9	0
CVC-EndoSceneStill	181	0	0	2	99.5	99.1	0
Average	-	-	-	-	**97.1**	**99.2**	**0**

4.4.2 Training

Instead of the 4-steps alternately training strategy to optimize RPN and Head losses, we test another approximately joint optimization (AJO) proposed by authors of (154) that takes a mini-batch as input and optimizes both losses at the same time. Nevertheless, there is no differential error increments for stochastic gradient descent (SGD) method at RoI pooling layer, the remedy is to propagate these increments backwards without processing. In contrast the 4-steps training methods, AJO has nearly the same test mAP on PascalVOC 2007 whereas faster during training (save up to 9 hours).

The training datasets contains 11954 images in total. We train Faster R-CNN on a K40c GPU with default parameters except setting mini-batch size to 128, all batches are normalized by subtraction of fix mean values. Training took no more than 4 days for fine-tuned network without observation of overfitting. In addition, VGG16 is initialized by ImageNet weights. And after 70000 iterations, fully-trained network saw the convergence except for class loss, which indicates that the fully trained Faster-RCNN using AJO may fail to detect polyps.

Table 4.5: Fine-tuned localization results for 300 proposals and comparison (part. 1).

Method	Dataset	TP	FP	FN	Precision	Recall
Faster R-CNN	CVC-Clinic2015-train	523	81	8	86.6	98.5
	CVC-ColonDB	262	30	8	89.7	97.0
	CVC-EndoSceneStill	149	32	2	82.3	98.7
Average	-	-	-	-	**86.2**	**98.1**
Darknet-YOLO-EIR (140)	ASU-Mayo Clinic	2245	1005	2068	69.1	52.1

Table 4.6: Fine-tuned localization results for 300 proposals and comparison (part. 2).

Method	Dataset	F1-score	F2-score	MD (in pixels)
Faster R-CNN	CVC-Clinic2015-train	92.2	95.9	27
	CVC-ColonDB	93.2	95.5	21
	CVC-EndoSceneStill	89.8	95.4	25
Average	-	**91.7**	**95.6**	**25**
Darknet-YOLO-EIR (140)	ASU-Mayo Clinic	59.4	62.5	-

4.4.3 Validation

Our polyp detection tasks include predicating whether a frame shows a polyp, and localizing the exact location of a polyp. To track training status, we utilize the rest two sequences of CVC-ClinicDB2017 training set as validation sets for evaluating the performance that contain 1178 frames, 910 of which contain a polyp. All evaluation metrics are consistent with MICCAI2017 sub-challenge except F-scores as shown in Tab. 4.1 and 4.2. Noted that FN, TP are counted once per frame, and FP, FN multiple times per frame.

Training sets of other datasets are considered as validation sets except for CVC-EndoSceneStill where the dataset has its own division up to 183 frames. 1, 25, 50, 100, 200, 300 regional proposals are tested respectively for each dataset.

4.5 Experiments

4.5.1 Detection

Tab. 4.3 and 4.4 show the fined-tuned results of 300 proposals which yields the best performance upon metrics whereas having the longest runtime. Typically, the detector runs at 17fps for 1 proposal which reaches the lower bound of realtime application, and 0.9fps for 300. Parameters are set as follows: Thresholds for RPN NMS, confidence of detection are 0.7 and 0.3, top 1,000,000 proposals before feeding to RPN NMS to ensure 100% detection rate, a higher confidence threshold 0.5 would drop the rate to 97.4%. It can be inferred from the detection results that the detection rate reaches a high level for CVC-Clinic2015, CVC-ColonDB and CVC-EndoSceneStill for the reason that each frame of these datasets contains at least one polyp. During the test of the experiments, due to the lower threshold set for confidence, the higher FN rate is observed during detection. Moreover, it is crucial to make a good trade-off between the performance and speed if an automatic detector is designed for real practice. We found on CVC-ColonDB that number of proposals influenced the detection rate greatly that the accuracy reduced from 97.3% for 300 proposals to 88.3% for 1 proposal. This trend is identical with that of CVC-Clinic2015 and CVC-EndoSceneStill.

Table 4.7: Performance comparison.

Novel detectors (14) implemented on CVC-Clinic2015DB testing set, and Faster R-CNN implemented on training set.

Method	Dataset	Precision	Recall	F1-score	F2-score
ASU	CVC15test	97.2	85.2	90.8	87.4
CUMED	CVC15test	91.7	98.7	95.0	97.2
CVC-Clinic	CVC15test	83.5	83.1	83.3	83.2
OUS	CVC15test	90.4	94.4	92.3	93.6
PLS	CVC15test	28.7	76.1	41.6	57.2
SNU	CVC15test	26.8	26.4	26.6	26.5
Ours	**CVC15train**	**86.2**	**98.1**	**91.7**	**95.6**

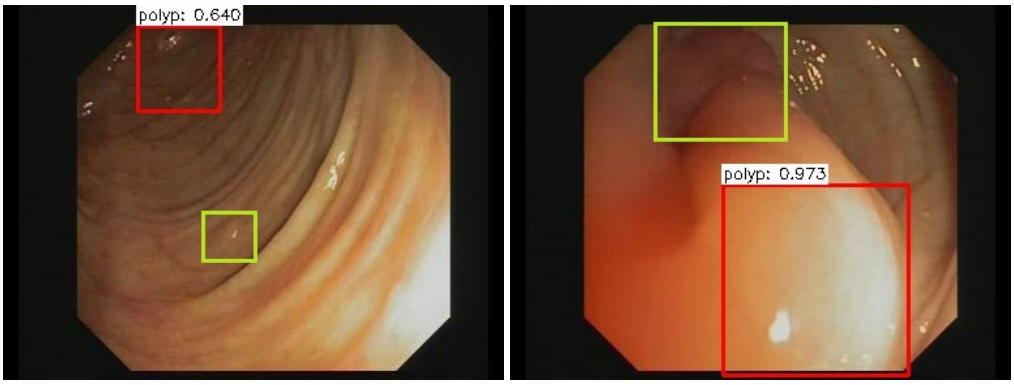
4.5.2 Localization

Corresponding localization results are shown in Tab. 4.5, 4.6 and 4.7. To compute MD, the Euclidean distance between the center of detected bounding box and that of ground truth is considered as the reference to judge if the detected center locates within the ground truth radius. Relatively high FP, FN rates are observed on CVC-Clinic2017 dataset, while for other three datasets, all metric values lay around 80%. These results can be regarded as the baseline for polyp localization in future works. In Tab. 4.5 and 4.6, it is denoted that though Darknet YOLO-EIR achieves realtime performance, the metrics are not sufficient for clinic use yet considering our 1 proposal results on CVC-ColonDB that the precision, recall, F1 and F2 scores, MD are 91.3%, 87.4%, 89.3%, 88.1%, 18 pixels respectively.

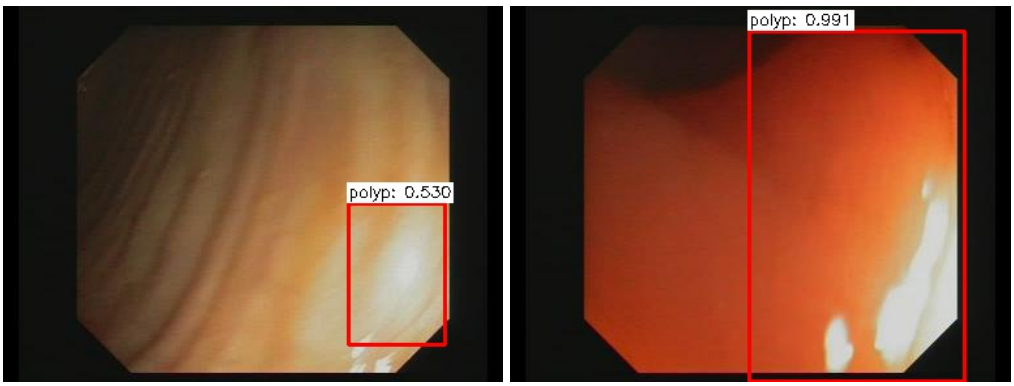
In comparison, as is manifested in Tab. 4.7, the outcomes indicate that Faster R-CNN achieves competitive performance compared to novel learning-based techniques, CUMED, ASU, and OUS (14) on videos with only polyp frames. Noted that these methods take one detection as TP if the detected center falls within the area of ground-truth mask, which is slightly different from MD metric. To be more specific, MD metric implemented here is more strict for it only considers the shortest side of the ground-truth box. During the experiments, we did not validate Faster R-CNN on the private ASU-Mayo Clinic dataset and the MICCAI2015 testing dataset due to their unavailability. However, the design of test set may differ from that of training set, this potential problem is alleviated by the various sets of polyps under different conditions from CVC15 training set and the similar sources of samples.

4.5.3 Fine-Tuning vs from Scratch

On small polyp datasets, we are interested in the resultant performances by training from scratch or fine-tuned. For fully trained Faster R-CNN, all weights are initialized by random sampling from Gaussian distribution with zero mean and a standard deviation of 0.01. Fine tuned network manifests high performance during the test as shown in Tab. 4.3, 4.4, 4.5 and 4.6. The fully-trained network, on the other hand, requires a few more days for training, and it has been observed that



(a) Small polyp. (b) Irrgular shape.



(c) Oval specular highlight. (d) Polyp-like intervention.

Figure 4.3: Failure modes for Faster R-CNN.
The green bounding box signifies groudtruth. (a) Small-sized polyps are missed. (b) Detector fails to locate low-contrast irregular polyp. (c) Area of specular highlight tricks the detector where there is no polyp. (d) In a frame without polyp, some suspicious area may trick both the detector and human eyes.

the lower mAP of fully-trained network might due to the AJO strategy in that the anchors are more sensitive to the initialized weights and RPN fails to provide sufficient positive samples.

The Faster R-CNN detector can detect largely occluded polyp and being robust to illumination changes as is depicted in Fig. 4.1a, also, noises as circular bubbles (Fig. 4.1b) are correctly predicted by the detector, even in the case that there are other tissues except polyp, and the detector correctly localizes the polyp in frames. Another advantage is that very large polyps that may occupy whole receptive field are successfully detected.

On the other aspect, although the detector is more liable to locate large polyps, it misses some very small polyps in the frames, as depicted in Fig. 4.3a, which accounts for the high FP rate in

Tab. 4.5, especially when predicting validation sequence 17 of CVC-Clinic2017. It should note that the detector learns the oval shape of polyp so firmly that it mistakes false areas (Fig. 4.3b-4.3d) as the real polyps, which causes high localization FP rate with respect to all datasets. In our future work, we would focus on solutions to these issues.

4.6 Conclusion

Faster R-CNN has been a fully end-to-end approach for object detection tasks on public datasets of natural scenes. For polyp detection and localization in endoscopic videos, this work first applies Faster R-CNN with VGG16 as the backbone. Through extensive experimental evaluation, the proposed approach exhibits potentials for reaching the best performance on precision, as well as yields competitive results in other metrics. The high detection performance indicates that Faster R-CNN could help lower the risk of missing polyps during colonoscopy examination even if RPN predicts only 1 proposal per test. On the other side, Faster R-CNN shows high false-positive rate in frames with presence of polyp during localization tests, which needs to be further investigated and discussed.

Chapter 5

Inference: Self-Attention Mechanism for Semantic Segmentation

Abstract

In this chapter, we proposed an end-to-end realtime global attention neural network (RGANet) for the challenging task of semantic segmentation. Different from the encoding strategy deployed by self-attention paradigms, the proposed global attention module encodes global attention via depthwise convolution and affine transformations. The integration of these global attention modules into a hierarchical architecture maintains high inferential performance. In addition, an improved evaluation metric, namely MGRID, is proposed to alleviate the negative effect of non-convex, widely scattered ground-truth areas. Results from extensive experiments on state-of-the-art architectures for suction region segmentation manifest the leading performance of proposed approaches[1] for robotic monocular visual perception.

5.1 Introduction

Correct bin picking by suction from cluttered environment is nontrivial for a robotic hand (230), since robotic hands have little prior knowledge of spatial shape by category, texture of material, or normal vectors of surfaces. To assist robotic hand in localizing feasible areas to pick by suction, Zeng et al (230) names the adsorbability of objects in a clustered scene as the affordance map (see Fig. 5.5), indicating the possibilities of objects being picked up.

[1] Relevant works are published as (122)

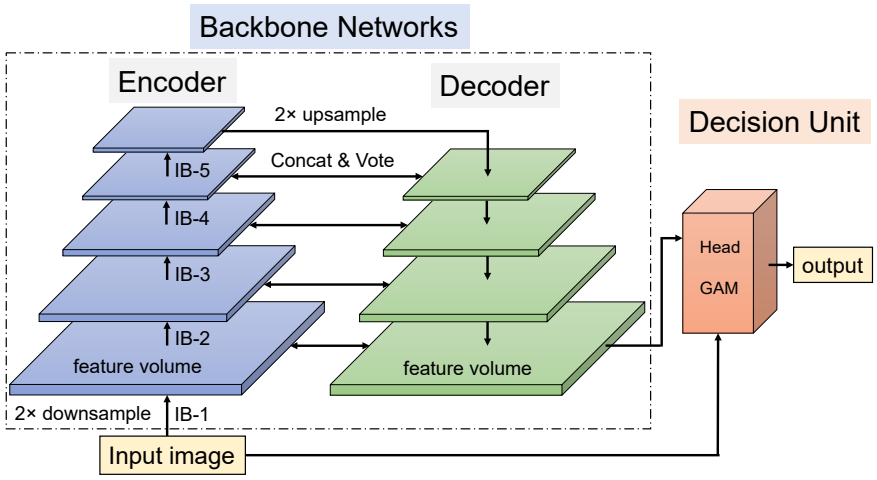


Figure 5.1: Hierarchical inference architecture of 5-scales RGANet5.
Backbone network consists of encoder and decoder (167) networks. One IB processes feature volume at a scale, and the decoder network interacts with encoder at all scales, generating high-order feature volumes. The decision unit has a Head to group both high-order features and raw image channels, and a GAM to acquire the final segmentations (affordance maps).

Predicting affordance maps is a variant of semantic segmentation. In practice, realtime visual perception with light computational liability is preferred, especially for robotic bin picking by suction. It should be noted that, the predicted affordance maps are not yet applicable enough for actual bin picking. Post-Refinement of estimating normal vectors of 3D surfaces, registering affordance maps to multiple coordinates systems, calibrations on robotic hand and cameras, and data stream synchronization etc. are necessary though time-consuming. Therefore, predicting more reliable affordance maps in real time with low-cost inferential processors will greatly benefit the real-world implementations.

We propose to efficiently predict affordance maps by realtime global attention network (RGANet), which can be easily adapted to other semantic segmentation tasks. As depicted in Fig. 5.1, RGANet is a light-weighted hierarchical architecture composed of inference blocks (IBs) that are based on attention mechanism (71; 196). For the purpose of preserving full-size activation of feature maps, pooling and dropout layers are deprecated. To explore the potentials of proposed global attention modules (GAMs), we adopt a GAM-enhanced encoder-decoder (167) backbone network.

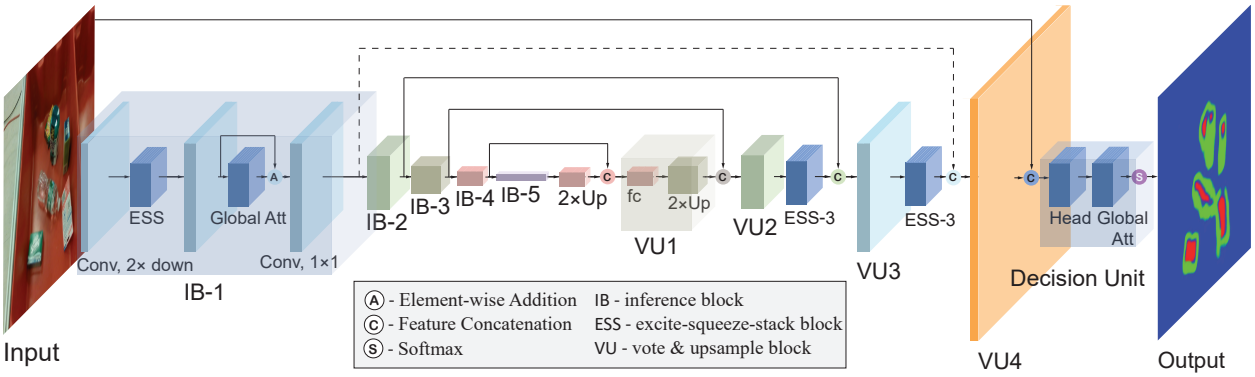


Figure 5.2: Forward pass of RGANet5.
It should be noted that detailed framework of ESS, GAM and decision unit are presented in Section 5.4, thereby corresponding blocks (in deeper blue color) do not represent feature maps. The network takes monocular input image, and outputs pixel-wise, one-hot encoded predictions of background (blue), negative sample (green), suction areas (red) respectively. The dashed line indicates the highway between encoder and decoder can be disconnected.

5.2 Motivation

RGANet for the challenging segmentation task It becomes more intricate given the constraint that, the very limited information offered is the RGB images of cluttered scene, and plausible areas for suction annotated by various human experts. Individuals may make mistakes while annotating images in their unique styles, which diversifies annotations greatly and makes it more difficult for traditional machine vision techniques to forge ahead with.

Realtime inference with global attention Vanilla convolution convolves and aggregates feature maps faster than fully connected layers due to shared weights. Depth-wise convolution (27) further reduces parameters by convolving groups of feature maps separately then concatenates. We propose to reformulate self-attention mechanism by implementing long convolutional kernels (please refer to section 5.4.2) and depth-wise convolutions instead of computing cosine similarities of feature vectors (196) or pure matrix multiplication (71) upon feature volumes.

Our perception is that, single convolutional kernel only needs to encode neighboring features other than entire feature volume, which can be mutually correlated by affine-transforming into global encoding. This is beneficial for reducing the computational complexity of self-attention

mechanism (196), which significantly increases inferential frame-rate to meet the requirement of realtime performance.

Rethink the bottleneck layers for global attention Dense block (70) densely concatenates feature maps derived from a sequential of bottleneck layers, then outputs a squeezed feature volume that may loss crucial contextual information. Therefore, we feed GAM the concatenated features, as a replacement of the squeezed feature volume, to enhance its global encoding capability.

Improved metric to evaluate unbalanced predictions During the evaluation phase, due to those unconnected ground-truth areas that are distributed across the entire image per the trade-off circumstance illustrated in Fig. 5.4b, generally applied segmentation metrics fail to correctly evaluate predictions. Thus, we propose Mean-Grid Fbeta-score (MGRID), a novel metric for segmentation evaluation, to alleviate the flaw via two-stage operations: partition and synthetics.

To summarize our contributions, this paper highlights the following novelties:

i. Propose a one-stage hierarchical inference architecture for semantic segmentation without any auxiliary losses.

ii. Propose the GAM for realtime inference[2] - 54fps on a GTX 1070 laptop, and 134fps with a V100 GPU.

iii. Propose the metric MGRID for evaluating widely-distributed predictions.

5.3 Related Works

5.3.1 Predict Affordance Maps

Zeng et al (230) implemented two FCNs (105) with ResNet-101 (188) backbones to fuse color and depth streams of cluttered scene. This approach yields 83.4% precision at Top-1% percentile affordance proposals. Azpiri et at (6) further refined the Top-1% percentile precision to ∼94% by

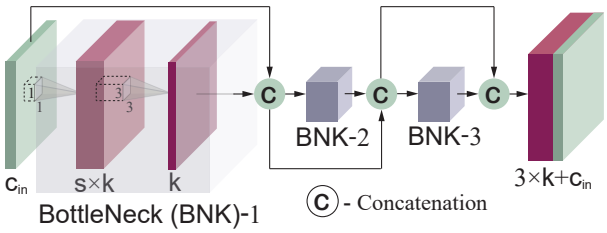
[2]Our Pytorch implementation of RGANet can be found at https://github.com/xiangyu8/RGANet_evaluation

a deep Graph Convolutional Network backbone, which outperforms FCN (ResNet-101 backbone) by $\sim$2%. Neither of these two approaches takes colored images as the only input, thus consuming much inference time in processing depth information. Shao et at (168) proposed to predict affordance map by sharing a ResNet-50 backbone between colored image stream and depth image stream, and a U-Net (155) to fuse features at different scales. Their approach is annotation-free, but requires the robotic hand to attempt to find the most possible points for suction, which is more expensive than deriving direct predictions from monocular images.

5.3.2 Self-Attention Modules in Semantic Segmentation

Self-Attention mechanism was first proposed in the field of nature language processing (196) for temporal domain. Wang et al (206) adapted the idea to domain within which a feature vector is spatially related to all other features in the feature volume. The features that are highly related will generate strong response, which facilitates network to model the non-local relations across the entire volume.

Li et al (93) designed a fully convolutional feature pyramid attention module for replacing the spatial-pyramid-pooling module (233) and a global-attention-upsample module with which high-level features perform global average pooling (236) as the guidance for low-level features. Woo et al (209) believed that simple channels-wise attention and spatial-wise attention sub-modules can boost representation power of CNNs. Recently, OCNet (224) efficiently aligned a global relation module and a local relation module which divided and merged feature volumes in different styles. Bello et al (11) proposed to augment standard convolution by attention mechanism. Cao et at (21) proposed a global-context block that benefited from simplified non-local block (206) and squeeze-excitation module (69). The proposed GAM in this paper is inspired by the criss-cross attention (CCA) mechanism of CCNet (71). CCNet approximates non-local attention by two cascade CCA modules, each of which correlates all feature vectors aligned horizontally and vertically. Though current attention-based approaches achieve state-of-the-art quantitative performance among semantic segmentation approaches, they usually have very high computational complexities



(a) Illustration of ESS-3 framework (3 BottleNecks).

CHapter5/Fig-3b.pdf

(b) The structure of GAM.

Figure 5.3: ESS-3 and realtime GAM.
(a) ESS-n block first expands input channel C_{in} channels to $s \times k$ channels by 1×1 convolution, where scalers s and k control the number of expanded channels; then it squeezes intermediate features to k channels by 3×3 convolution, densely stacking all k-channel features gradually towards final $n \times k + C_{in}$ channels. Specially, we set $C_{in} = k$ to enforce the entire network scalable w.r.t. ratio k. (b) Query and key of GAM are derived from affine transforms, and either query or key is channel-wise ($1 \times c$ and $c \times 1$ depth-wise convolutions) and spatial-wise ($w \times 1$ and $1 \times h$ depth-wise convolutions), globally related by matrix multiplication and aggregation layers.

that reduce the practicability for deployment.

5.4 Methodology

One specific forward pass of the proposed framework is illustrated in Fig. 5.2. RGANet5 has 5 levels of inference blocks (IBs), and each block applies to a $2\times$ down-sampling of its previous feature map to capture features at each scale. The decoder network is composed of vote & upsample (VU) blocks and excite-squeeze-stack (ESS) bottleneck layers for better decoding capabilities. Due to the enhancement of GAM at each IB, we only utilize 3×3 standard convolution w/o pooling layers to downsample features, and 1×1 convolution for reformulating feature depth. We have tested large kernels for down-sampling, but they didn't outperform 3×3 convolutions.

5.4.1 Hierarchical Inference Architecture

RGANet5 is designed as a single 'distillation tower' (see Fig. 5.2) with 5 temperature levels, each level is composed of one IB. The architecture is expandable such that RGANet*n* has n cascade IBs. Each IB halves input feature size, then modulates channels to ratio $k = 15$ by a standard 3×3 convolution. ESS block (see Fig. 5.3a) is regarded as the backbone for IB, and we adopt ESS-3, ESS-6, ESS-12, ESS-24 for IB-1 and IB-2, IB-3, IB-4, IB-5 respectively. Output of GAM (refer to Fig. 5.3b) is directly added to the output feature volume $\mathbf{x}$ of ESS block as the residual $\lambda \cdot \mathbf{x}$, which formulates the incremental up-sampling layers accordingly. The final output takes the form $(1+\lambda)\cdot\mathbf{x}$, where λ is the learnable weights volume that has the same size of $\mathbf{x}$, and operator '$\cdot$' signifies element-wise product. Therefore, non-negative activation of GAM artifact is preferred, and Batch-Norm (BN) layers are necessary to restrict its upper bound.

As the synthesis of 5-scales distillation artifacts, highway connections not only populate previous inferences to up-sampling layers accordingly, they also facilitate gradient back-propagation during training phase, especially for a very deep network. The 'vote & upsample' (VU) block (see Fig. 5.2) weights concatenated input feature maps by 1×1 convolution without bias, and these weighted features are then $2\times$ upsampled via nearest interpolation. If inferring without last two ESS-3 blocks, RGANet will not yield fine-grained predictions. Also, we notice that IB-1 is directly linked to the last UV4 block, which performs poorly without Decision Unit (DU) shown in Fig. 5.2. DU consists of feature modulating head to deduct channels from $k+3$ to the number of classes by 1×1 convolution, and one last GAM that yields predictions.

5.4.2 Realtime Global Attention

Fig. 5.3b illustrates the pipeline of the GAM. With a batch-size of 1, input feature volume $\mathbf{x}$ filtered by previous ESS block has a shape of $h\times w\times c$, which is rotated to $c\times h\times w$ query $Q(c,h,w)$ (upper branch, blue color) and $w\times c\times h$ key $K(w,c,h)$ (lower branch, purple color). Query conducts channel-wise attention via depth-wise sliding w kernels $W(c,1)$ shaped $c\times1$ across $c-h$ view, which results in a $c\times1\times w$ feature volume Q^c. Similarly, h horizontal positions are encoded by

$1 \times h$ depth-wise convolutional kernel $W(1,h)$, mapping into the $c \times 1 \times w$ horizontally positional encoding Q^h. Key is transformed accordingly except the sizes of kernels, and we denote its artifacts as channel-wise encoding K^c and vertically positional encoding K^v. The final output F_{out} is then formulated as:

$$F_{out} = \lambda \cdot \mathbf{x} = f\{\text{rot}[Q^h Q^c] \bigcup \text{rot}[K^v K^c]\} \cdot \mathbf{x}, \tag{5.1}$$

where rot$[\cdot]$ operator signifies rotation; $f\{\cdot\}$ operator performs 1×1 convolution to resize $2c$-channels to c-channels with Swish activation function (149), then maps to [0, 1] weights volume via Sigmoid or Softmax functions; '$lhs \bigcup rhs$' operator concatenates lhs and rhs features. Let '$*$' denote the depth-wise convolution, we know:

$$\begin{aligned} Q^h = Q(c,h,w) * W(1,h), & Q^c = Q(c,h,w) * W(c,1), \\ K^v = Q(w,c,h) * W(w,1), & K^c = Q(w,c,h) * W(1,c). \end{aligned} \tag{5.2}$$

Considering Eqn. (6.1) and Eqn. (6.2), RGA's total number of trainable parameters is $2hw + cw + hc$ (excluding BN layer, bias and 1×1 point-wise convolution), while for vanilla convolution with the same kernel height and width, this number becomes $hw^2 + wh^2 + cw^2 + ch^2$. In terms of global attention, although matrix multiplication operation only correlates features horizontally and vertically aligned for any feature vector, Q^h, Q^c, K^v and K^c themselves are the artifacts of global depth-wise convolutions, resulting in each element in these 4 Queries and Keys bounds other elements with shared weights. We can also treat RGA as the type of 'learnable global attention'. Furthermore, all affine operations of RGA module are differentiable. We didn't observe vanishing, or exploding gradients issues during training.

Different from the criss-cross attention mechanism (71) via two cascade CCA Modules with shared weights, GAM realizes global correlation of feature vectors in an all-in-one fashion without calculating the standard matrices multiplication of Query and Key. Furthermore, GAM does not rely on the dot-product of channel-wise feature vectors, but the depth-wise convolutional kernels to encode the correlations.

5.4.3 Rethink Densely-Stacked Bottlenecks

A Dense block (70) has multiple densely connected Bottlenecks (BNK) modules, each BNK acts in a stack-squeeze manner, and the last one outputs a k-channels feature volume.

The ESS-n block, as illustrated in Fig. 5.3a, outputs a stacked $(n+1)k$-channels feature volume by n densely connected BNKs, each contributes k-channels of features, and we let $C_{in} = k$. Each channel of stacked feature maps shows one degree of filtered raw input features, such that RGA modules are capable of correlating high-order features to low-level features. RGANet5 adopts ESS-3 for IB-1, ESS-3 for IB-2, ESS-6 for IB-3, ESS-12 for IB-4, and ESS-24 for IB-5. One of our future works is to let RGA module ignore noisy low-level features, and to locate more reliable high-order features.

5.4.4 MGRID metric for Evaluation

Existing evaluation metrics poorly treat the special case of a prediction map as illustrated in Fig. 5.4b that, when original predictions cover object-1 (red), but fail to locate object-2 (green) at the left-bottom corner of image, off-the-shelf metrics yield the same results if part of object-1 relocates to the object-2, because this relocation does not affect paranormal statistics of TP, FN, TN and FP. In reality, we want predictions to be able to cover more objects, such that a robotic hand would seek-and-pick all objects even though Precision or Recall is yet not favorable enough (e.g., $\sim$15% Recall on object-2 alone). It would be more reasonable to assign higher score for the prediction map that covers 2 objects.

Partition 2-Stage MGRID metric aims to remedy the issue mentioned above. As shown in the third image of Fig. 5.4b, during partition stage, predication map is manually divided into four cells, and each cell is treated fairly against any other cell. An ideal partition will separate objects by different cells. Next, only calculate Fbeta-scores (or any other existing metrics) for all cells that contain predictions and categorical ground-truth (non-zero TP, FP or FN samples), using the

definition

$$F_\beta = \frac{(1+\beta^2)TP}{\beta^2(TP+FN)+TP+FP+\varepsilon}, \tag{5.3}$$

where $\varepsilon = 1 \times 10^{-31}$, $\beta > 1$ weights Recall more than Precision, and $0 < \beta < 1$ weights the opposite. While instance-based metric is intractable for localizing instances, the partition step efficiently simplifies this process. Furthermore, it is feasible to substitute the Fbeta-score metric by other existing metrics, e.g., IoU, Precision, Accuracy etc.

Synthetics The second step is to synthesize all n Fbeta-scores $\mathscr{F} = \{F_\beta(i)|i = 1,2,...,n\}$ collected from all partitions, and any Fbeta-score $F \in \mathscr{F}$ is curved by a regulator as shown in Fig. 5.4a, which takes the form

$$\Gamma(F) = \begin{cases} S(F-F_m)^3 + C_m, & 0 < F \leq 100\% \\ 0, & F = 0 \end{cases}, \tag{5.4}$$

where coefficients $0 < F_m,\ C_m < 1$ require to be manually set. Fig. 5.4a shows the curve by setting $(F_m, C_m) = (0.5, 0.525)$, then S is calculated as $S = (1-C_m)/(1-F_m)^3$. Let $T = [F_m/(1-F_m)]^3$, then intercept B at $F = 0$ can be denoted as $C_m(1+T) - T$. To make the regulator effective, B should fall within the interval $(0, F_m)$, which leads to valid ranges of F_m and C_m:

$$\frac{T}{1+T} < C_m < \frac{F_m+T}{1+T},\ 0 < F_m < 1. \tag{5.5}$$

The final confidence score is derived by

$$MGRID = \frac{1}{n}\sum_{F\in\mathscr{F}} \Gamma(F). \tag{5.6}$$

If calculating the average metrics over entire image without non-linear curvature function $\Gamma(F)$, according to our tests, the average is much similar to the Fbeta-score without partition, since scores of all cells tend to compensate each other during the phase of addition. Also, the cubic form of

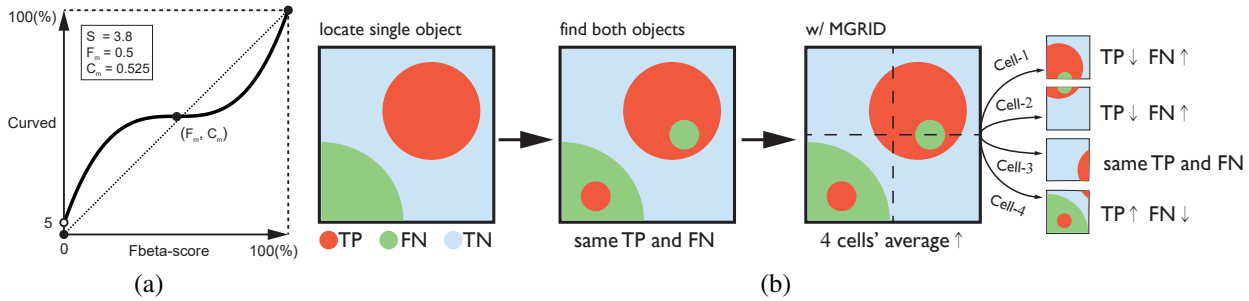


Figure 5.4: MGRID metric.
(a) Synthetic function. (b) A special case for predicting suction areas when the same precision and recall indicate better predictions for real-world implementation, and how MGRID mitigates the circumstance by partition and synthetics. True Negative (TN) and False Positive (FP) samples stay unchanged during the processes.

$\Gamma(F)$ consumes less computational resources than slanted sine and cosine curvature functions. Eventually, non-linear $\Gamma(F)$ strengthens the impact of cells with low Fbeta-scores, and weakens those with high scores.

5.5 Experiments

5.5.1 Configuration

Public-available suction dataset (229) consists of camera intrinsics and pose records, RGB-D images of clutter scenes and their backgrounds, and labels. We only adopted color images and labels w/ a train-split of 1470 images and a test-split of 367 images, each image has a resolution of 480×640. All colored images were normalized to tensors valued between 0 and 1, which were not resized or padded during training and testing.

5.5.2 Train and Test

We implemented AdamW optimizer (106) with AMSGrad (151), compared two weighted loss function during training - focal loss (FLoss) (103) and cross-entropy loss (CELoss). Assume y denotes the prediction, y' the one-hot encoded ground-truth, n the total number of classes, and γ and α are constants, then FLoss and CEloss are denoted as:

$$
\begin{aligned}
FLoss(y,y') &= -\sum_{c=1}^{n} \alpha_c \sum_{c} (1-y)^{\gamma} \log y, \\
CELoss(y,y') &= -\sum_{c=1}^{n} \alpha_c \sum_{c} \log y
\end{aligned}
\tag{5.7}
$$

where $\sum_{c=1}^{n} \alpha_c = 1$, $\alpha_c \geq 0$, $\gamma > 0$, and all $y \in [0,1]$. Training-set is augmented during training by random hue, flip, rotation, blur, shift etc.

Online testing comes simultaneously during training, which shows calculations of IoU, precision, and recall of running batches. The offline testing loads checkpoint, merely evaluates the class corresponds to predicted suction areas.

All experiments were conducted using a GTX1070 laptop, and one Tesla V100 GPU. Network scaler k was set to 15 constantly for a better trade-off between module size and performance. We set constant learning rate to 1.5×10^{-4}, weights-decay rate to 0; $\gamma = 1.3$, $\alpha_1 = 0.25$ and $\alpha_2 = 0.25$ upon background and negative samples, $\alpha_3 = 0.5$ upon suction areas because of unbalanced proportions; default MGRID parameters $\beta = 0.5$, grid intervals $(\delta_H, \delta_W) = (12px, 12px)$, C_m and F_m took the same values as illustrated in Fig. 5.4a. The training of RGANet5 lasted for $\sim$2 days.

Multiple metrics are implemented to evaluate affordance maps. Note that for the suction dataset, users only care about feasible regions to adsorb. Therefore, only the category that represents predicted suction areas is evaluated, and the final scores are the averages by all frames of test-set.

5.5.3 Ablation Study

NVIDIA mixed-precision training technique is adopted by all frameworks in Tab. 5.1 to enlarge batch size. In our experiments, larger batch size enables faster convergence.

Presence of GAM In Tab. 5.1, we present several cases when GAMs in DU or IBs are removed. Blocking highways have unpredictable influence on the performance over test-set, since each branch of highway provides both useful and noisy contextual information. Nevertheless, GAM

Frameworks	Block	DU	GA	AW	MGRID	IoU
RGANet-B4	1,2,3,4				31.34	33.73
RGANet-B4	1,2,3,4	✓	✓		33.46	36.49
RGANet-B3	1,2,3	✓	✓		34.20	37.84
RGANet-B2	1,2	✓	✓		34.58	37.95
RGANet-B1	1	✓	✓		34.96	37.11
RGANet-NB	–				33.66	36.65
RGANet-NB	–	✓			33.83	36.80
RGANet-NB	–		✓		34.63	38.08
RGANet-NB	–	✓	✓		34.80	38.01
RGANet-NB	–	✓	✓	✓	**36.55**	**40.58**

Table 5.1: Ablation study on GAM and CELoss w/ adaptive weights (AW).
All averaged evaluation metrics are presented in '%'. 'Block' indicates the highway to which IB is blocked, 'DU' refers to the existence of GAM in DU, 'GA' indicates all GAMs in IBs, and 'RGANet' denotes the RGANet5 architecture. FLoss performs poorly due to the uncertainty arisen from γ.

shows a favor of features at all scales, especially the high-order IB-4 features.

As illustrated in Tab. 5.1, GAM in DU behaves as the decoder in self-attention mechanism (196), and it should pair with the encoding GAMs in the backbone network. We also noted that the fully convolutional version of RGANet5-NB alone shows a competitive performance, which indicates the effectiveness of densely-stacked BNKs.

Adaptive weights for CELoss Adaptive weights (AW) are computed as the reversed global proportions of all categories in training-set. Compared to the roughly-estimated weights $[\alpha_1, \alpha_2, \alpha_3]$ = [0.25, 0.25, 0.5], normalized AW takes the values [0.063, 0.266, 0.671]. According to Tab. 5.1, AW efficiently boosts the performance of RGANet5-NB by $\sim$2% in the metrics of MGRID and IoU.

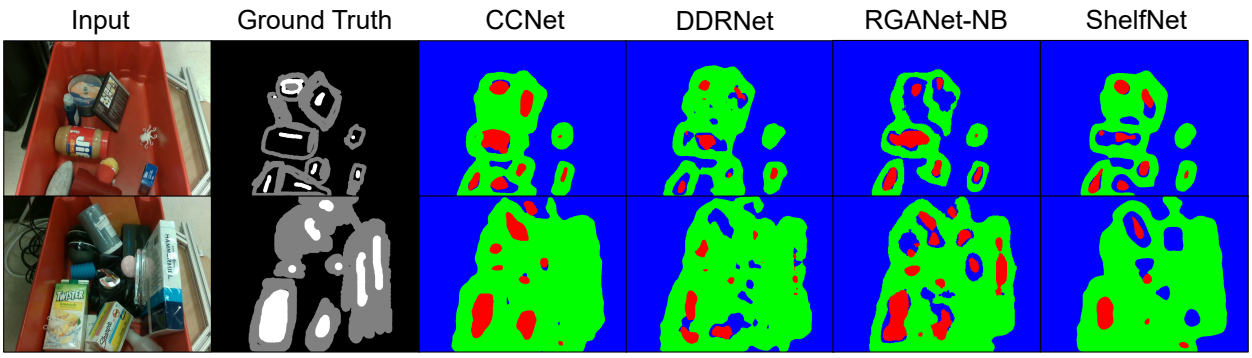


Figure 5.5: Qualitative results on test-set.
Red regions indicate the highest affordance for adsorption, which corresponds to white regions in ground-truth.

Group 1 - Large Models	Backbone	Parameters	Inference Time	FLOPs
RGANet5-NB + AW (**Ours**)	-	**3.41M**	**7.51ms / 133fps**	**1.57B**
CCNet (ResNet101) (71)	ResNet101	71.27M	51.03ms / 19fps	73.03B
FCN (ResNet50) (105)	ResNet50	32.96M	22.85ms / 43fps	32.48B
FCN (ResNet101) (105)	ResNet101	51.95M	39.41ms / 25fps	50.72B
DeepLabv3 (ResNet50) (43)	ResNet50	39.64M	32.71ms / 30fps	38.40B
DeepLabv3 (ResNet101) (43)	ResNet101	58.63M	49.17ms / 20fps	56.63B
BiSeNetv1 (222)	ResNet18	23.08M	9.65ms / 103fp/s	9.53B

Table 5.2: Comparison with large semantic segmentation models in model size.
Using a Tesla V100 GPU, all averaged evaluation metrics are presented in '%'. Proposed approaches achieve competitive performance with the least total parameters, indicating a better trade-off between model size and performance.

Group 1 - Large Models	Backbone	IoU	F1	MGRID
RGANet5-NB + AW (**Ours)**	-	40.58	53.26	36.55
CCNet (ResNet101) (71)	ResNet101	**43.83**	**57.00**	**38.90**
FCN (ResNet50) (105)	ResNet50	42.15	55.32	37.91
FCN (ResNet101) (105)	ResNet101	42.28	55.51	37.69
DeepLabv3 (ResNet50) (43)	ResNet50	43.17	56.32	38.42
DeepLabv3 (ResNet101) (43)	ResNet101	41.98	55.05	37.55
BiSeNetv1 (222)	ResNet18	37.64	50.38	34.18

Table 5.3: Comparison with large semantic segmentation models in metrics. Using a Tesla V100 GPU, all averaged evaluation metrics are presented in '%'. Proposed approaches achieve competitive performance with the least total parameters, indicating a better trade-off between model size and performance.

Group 2 - Light-Weighted Models	Backbone	Parameters	Inference Time	FLOPs
RGANet5-NB (**Ours)**	-	3.41M	7.51ms / 133fps	1.57B
DeepLabv3 (164)	MobileNetv2	4.12M	2.92ms / 342fps	1.16B
DDRNet-23-slim (64)	-	5.69M	**1.23ms / 813fps**	1.07B
HRNet-small-v1 (204)	-	**1.54M**	1.84ms / 543fps	**0.97B**
HarDNet (23)	-	4.12M	1.63ms / 613fps	1.03B
ShelfNet (238)	ResNet18	14.57M	2.06ms / 485fps	2.91B
STDCv1 (41)	-	14.23M	2.45ms / 408fps	5.48B

Table 5.4: Comparison with light-weighted semantic segmentation models in model size. Using a Tesla V100 GPU. All averaged evaluation metrics are presented in '%'.

Group 2 - Light-Weighted Models	Backbone	IoU	F1	MGRID
RGANet5-NB (**Ours**)	-	**38.01**	**50.21**	**34.80**
DeepLabv3 (164)	MobileNetv2	34.61	47.33	31.76
DDRNet-23-slim (64)	-	32.30	43.95	32.30
HRNet-small-v1 (204)	-	34.31	46.28	31.97
HarDNet (23)	-	35.15	47.13	32.61
ShelfNet (238)	ResNet18	36.17	48.61	33.32
STDCv1 (41)	-	36.10	48.34	33.19

Table 5.5: Comparison with light-weighted semantic segmentation models in metrics. Using a Tesla V100 GPU, all averaged evaluation metrics are presented in '%'.

5.5.4 Compare to State-of-the-Arts

We conducted experiments to compare RGANet5 with several novel semantic segmentation approaches. These approaches can be divided into two groups - one group that has much more parameters/FLOPs that substantially can outperform RGANet, and the other one that has comparable model sizes. As shown in Tab. 5.2 and 5.3, Deeplabv3 (43) and FCN (105) do not rely on attention mechanism, while CCNet (71) has merely two cascade CCA modules that bring in tremendous amount of trainable parameters and operations. Light-weighted RGANet with 6 GAMs, on the other hand, achieves competitive performance (2-4% less) against the best performer on the test-set. Also, RGANet5 outperforms BiSeNetv1 (222), another attention-based realtime approach, by $\sim 6\times$ less parameters and FLOPs.

As illustrated in Tab. 5.5, the proposed approach achieves the best IoU, F1 and MGRID score when compared with top-tier realtime approaches selected from Cityscape Leader Board (135). Although behaving better in metrics evaluation, RGANet runs relatively slow due to the fact that PyTorch is well-optimized for convolutional neural networks. It is one of our future works to further optimize RGANet for faster and more accurate inference. Readers may refer to Fig. 5.5 for our qualitative evaluation.

5.6 Conclusion and Future Works

Firstly, we introduced a novel light-weighted, hierarchical inference network embedded with realtime global attention modules. Densely connected excite-squeeze-stack blocks generate feature volume as the input to realtime global modules, and the attention module correlates features via learnable weights and affine transformations. Ablation study, as well as the comparison with the state-of-the-art approaches manifests the competitive performance of the proposed RGANet5. Secondly, we designed the MGRID metric, which effectively leverages on the weights of predictive regions via partition and synthesis stages. Our future works include but not limit to, enhancing the encoding capability of inferential blocks by efficient backbone networks and optimizations.

Chapter 6

Post-Processing: Dilated Continuous Random Field for Semantic Segmentation

Abstract

Mean field approximation methodology has laid the foundation of modern Continuous Random Field (CRF) based solutions for the refinement of semantic segmentation. In this chapter, we propose to relax the hard constraint of mean field approximation - minimizing the energy term of each node from probabilistic graphical model, by a global optimization with the proposed dilated sparse convolution module (DSConv). In addition, adaptive global average-pooling and adaptive global max-pooling are implemented as replacements of fully connected layers. In order to integrate DSConv, we design an end-to-end, time-efficient DilatedCRF pipeline. The unary energy term is derived either from pre-softmax and post-softmax features, or the predicted affordance map using a conventional classifier, making it easier to implement DilatedCRF for varieties of classifiers. We also present superior experimental results of proposed approach[1] on the suction dataset comparing to other CRF-based approaches.

6.1 Introduction

An affordance map indicates valid areas for suction-based bin-picking, ignored background, and negative samples that signify an object w/o valid suction areas. The annotations are labeled by

[1] Relevant works are published as (123)

multiple human experts, which is not precise enough due to several reasons: the same object is annotated in various styles by different experts; annotations are yielded via seemingly paradoxical inferences, e.g., a glass is pickable while a plastic ball is not; some objects, such as a laser pen, or largely occluded objects, are labeled as pickable in some affordance maps, whereas being unpickable in others.

Therefore, the task to predict precise suction area is challenging given such conditions. No matter which approach is implemented to solve this challenging semantic segmentation task, the goal of CRF based post-processing is to refurbish the predictions into more fine-grained artifacts. As to the post-processing itself, real-time performance is preferred for robotic applications.

As one of the most popular works in demonstrating CRF for the field of semantic segmentation, the DenseCRF (85), proposes mean field approximation theory to approximate global energy minimization by optimizing local unary and pairwise potentials that encode relative positional and color information of entire image (see Fig. 6.1), and the message passing from other features to the local feature becomes the computational bottleneck, especially when the image is large in size. DenseCRF proposes to further boost the efficiency by lattice permutation (4), which reduces the computational complexity from $O(N^2)$ to $O(N)$. Still, it remains to be a key topic seeking the perfect trade-off between better global approximation and efficiency.

We investigate the implementation of standard convolution upon CRF, and decreasing the usage of fully-connect layers. During the message-passing stage of CRF shown in Fig. 6.1, it seems natural that multiple linear layers can learn relative relationship between a local feature and all other features by connecting all these features and output pairwise energy terms. This assumption relies on massive multiplications between large volumes of matrices, and may overfit small datasets due to the huge number of trainable parameters. In order to reduce trainable parameters, and design a high-performance CRF for the general task of semantic segmentation, we propose the dilated sparse convolution (DSConv) module[2] that performs fast channel-wise convolution with a static kernel, then extract global energy terms using adaptive global max pooling (AGMP) and

[2]Our Pytorch implementation can be found at https://github.com/dunknowcoding/DilatedCRF

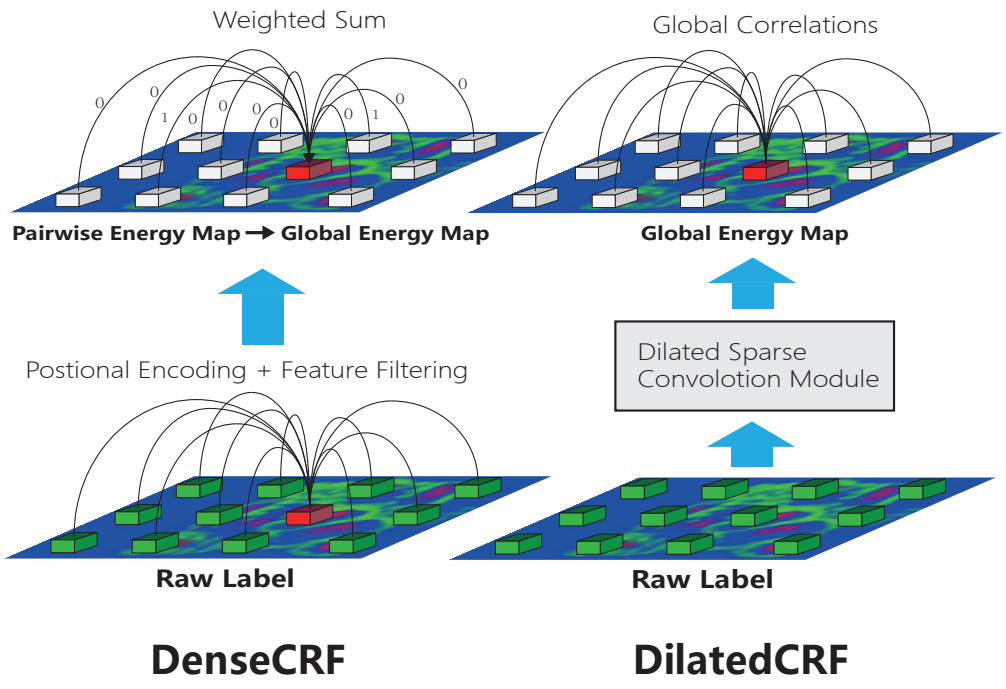


Figure 6.1: Message-passing sketches for DenseCRF and DilatedCRF. Better viewed in colored mode. The green blocks indicate feature vectors indexed by pixel coordinates, one red block represents a focus on the active message, and the white blocks signify the encoded pairwise energy terms indexed by pixel coordinates. Different from DenseCRF that encodes relative positional information and features then aggregates via weighted sum, our proposed DilatedCRF predicts the global energy map end-to-end via proposed DSConv module.

adaptive global average pooling (AGAP) as replacements of linear layers. With DSConv modules, a full-size Dilated CRF

successfully boost the quantitative performance of the state-of-the-art LR-ASPP (67) approach by total 0.36M parameters.

DSConv module correlates features in a global manner other than following the mean field approximation methodology. Since DilatedCRF is end-to-end trained with a GPU, it can learn trainable parameters globally without customized CPU-only operations, e.g., the computation of potentials with L2-norms and convolution optimized by lattice permutation (85; 235), patch-wise multiplication (193), etc. Also, it should be noted that, the massive concatenation of output features, and memory transfer consume the majority of runtime. One of our future works is to accelerate these trivial operations.

6.2 Continuous Random Field (CRF)

6.2.1 The General Form

Suppose a label space $\mathscr{L} = \{l_i\}, i = 1, ..., h \times w$, h and w are the height and width of the ground-truth, random variable $l_i \in \{y_i\}, i = 1, 2, 3$, y_i corresponds to a category of the suction dataset (229), and input feature volume $\mathscr{X} = \{\mathbf{x}_i\}, i = 1, ..., h \times w$, each $\mathbf{x}_i$ is a 3-D feature vector. According to Hammersley-Clifford theorem (Refer to Appendix. B), the CRF defined by $\mathscr{L}$ and $\mathscr{X}$ is then denoted as a Gibbs distribution over the set $\mathscr{C}$ of all maximal cliques:

$$P(\mathscr{L}|\mathscr{X}) = \frac{\prod_{c \in \mathscr{C}} \psi_c(\mathscr{L}|\mathscr{X})}{Z(\mathscr{X})}, \tag{6.1}$$

potential $\psi_c(\mathscr{L}|\mathscr{X})$ signifies a strict positive function defined on maximal clique c of undirect graph G. In the context of semantic segmentation, let V be the set of all spatial locations, E the edges that connect vertices from V, then $G = (V, E)$. $Z(\mathscr{X})$ is the normalization factor that ensures $P(\mathscr{L}|\mathscr{X})$ a valid probabilistic distribution,

$$Z(\mathscr{X}) = \sum_{\mathscr{L}} \prod_{c \in \mathscr{C}} \psi_c(\mathscr{L}|\mathscr{X}). \tag{6.2}$$

CRF refinement is achieved by the maximum a posteriori (MAP) labeling $\mathscr{L}^* = \arg\max_{\mathscr{L}} P(\mathscr{L}|\mathscr{X})$ w.r.t. ground-truth.

Complete graph G is ideal for refinement. However, its defect is also noteworthy - massive computation to exhaustively enumerate all possible combinations of random variable $\mathscr{L}$. To avoid the prod operation, exponential function is applied to potentials such that

$$P(\mathscr{L}|\mathscr{X}) = \frac{\exp(-E(\mathscr{L}|\mathscr{X}))}{Z(\mathscr{X})}, \tag{6.3}$$

where the Gibbs energy $E(\mathscr{L}|\mathscr{X}) = \sum_{c \in \mathscr{C}} \psi_c(\mathscr{L}|\mathscr{X})$. According to (6.3), optimization of a CRF can be denoted as $\mathscr{L}^* = \arg\min_{\mathscr{L}} E(\mathscr{L}|\mathscr{X})$, $Z(\mathscr{X})$ is ignored because it only depends on input

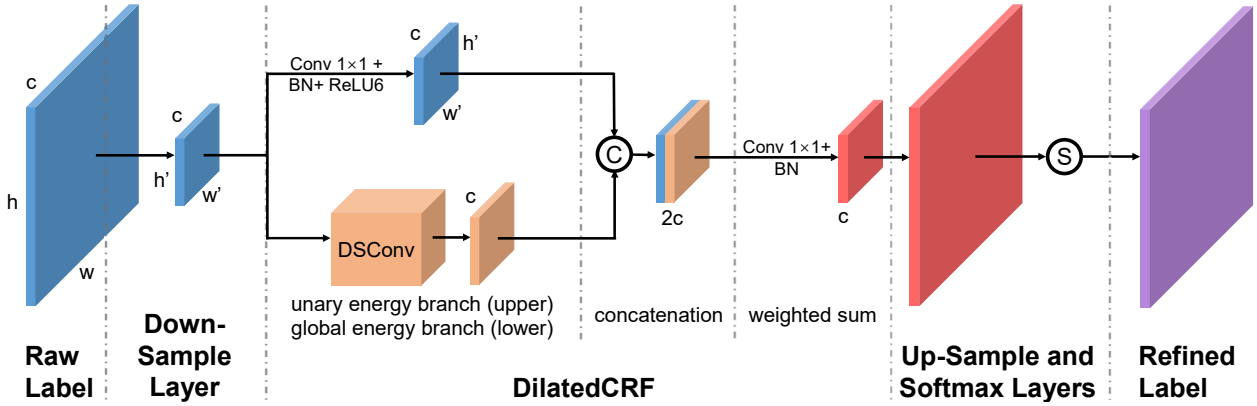


Figure 6.2: DilatedCRF pipeline.
'Raw label' signifies the predicted affordance map or raw feature volume since all of them encode the unary energy term presented in (6.7). The forward passing has three stages - down-sampling by a manually specified scale (multipliers 1/8, 1/4 and 1/2 for the suction dataset (229)), DilatedCRF, and up-sampling to original label sizes, the final softmax operation interprets Gibbs energy (refer to (6.3)) optimization as minimizing the KL-Divergence between refined label and one-hot encoded ground-truth[2].

feature volume $\mathscr{X}$.

6.2.2 DenseCRF for Semantic Segmentation

Dense CRF employs a fully-connected graph G such that there's only one maximal clique. Let $N = h \times w$, the Gibbs energy has the form

$$E(\mathscr{L}|\mathscr{X}) = \sum_{1\leq i\leq N} \psi_u(l_i|\mathscr{X}) + \sum_{1\leq i<j\leq N} \psi_p(l_i,l_j|\mathscr{X}), \tag{6.4}$$

where ψ_u, ψ_p are unary and pairwise potentials respectively. Commonly-implemented DenseCRF (85) defines $\psi_u(\mathscr{L}|\mathscr{X})$ as the output of a classifier given input $\mathscr{X}$, and ψ_p as

$$\psi_p(l_i,l_j|\mathscr{X}) = \mu(l_i,l_j) \sum_{m=1}^{M} w^{(m)} k^{(m)}(\mathbf{x}_i,\mathbf{x}_j), \tag{6.5}$$

where $\mu(l_i,l_j)$ is the compatibility function which is manually assigned by Potts model $\mu(l_i,l_j) = [l_i \neq l_j]$, each $w^{(m)}$ is a learnable weight, and $k^{(m)}$ indicates a Gaussian kernel.

DenseCRF also approximates true distribution $P(\mathscr{L}|\mathscr{X})$ via mean field. It computes a distri-

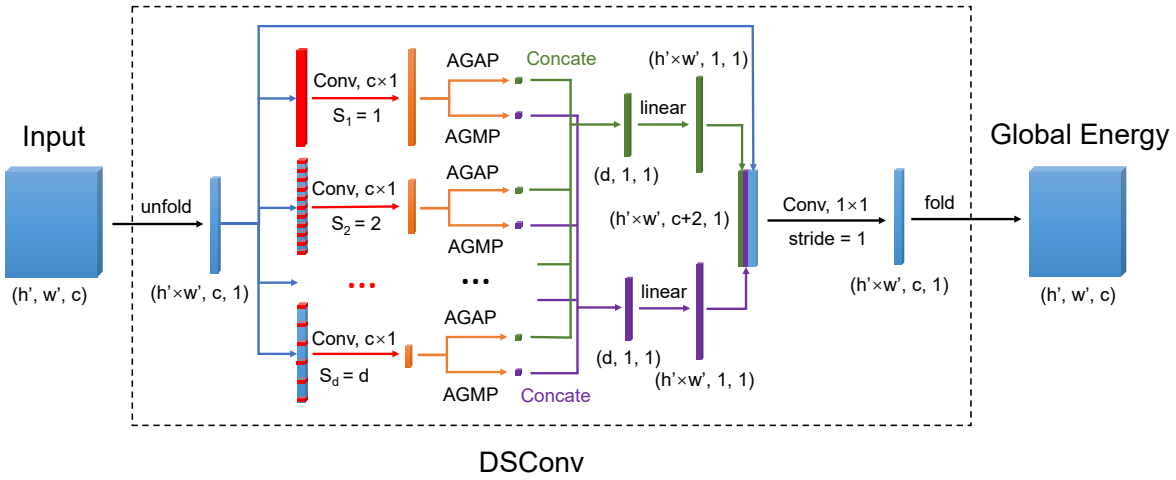


Figure 6.3: DSConv module.
A DSConv module takes down-sampled label volume as the input. Two up-sampled features after linear layers are normalized using Layer Normalization (7), and activated by ReLU6 function. Concatenation of global pooling artifacts (in green or purple) is the computational bottleneck of proposed DilatedCRF. The folding operation reverses the unfolding of the input feature volume.

bution $Q(\mathscr{L}|\mathscr{X})$ that minimize KL-divergence against $P(\mathscr{L}|\mathscr{X})$, which takes the form

$$Q(\mathscr{L}|\mathscr{X}) = \prod_{i=1}^{N} Q_i(l_i|\mathscr{X}). \tag{6.6}$$

The assumption of mean field approximation is dedicate, that vertices of G are conditional independent from each other. Therefore, minimizing the energy of each node Q_i results in an approximation of global minimum.

6.3 Related Works

6.3.1 Conventional CRF

Most of works on CRF are based on mean field approximation theory. For super-pixel image segmentation, Sulimowicz et al (184) formulate super-pixel cue onto pairwise potentials, which outperforms DenseCRF; Ma et al (113) introduce two coefficients to unary and pairwise potentials for better encoding super-pixel energy terms; Li et al (97) define a third potential to model the cost of assigning super-pixel labels; Yang et al (218) define a interlayer high order potential to enhance

super-pixel segmentation. These works improve DenseCRF w/o adapting CRF to neural networks, which are categorized as conventional CRF.

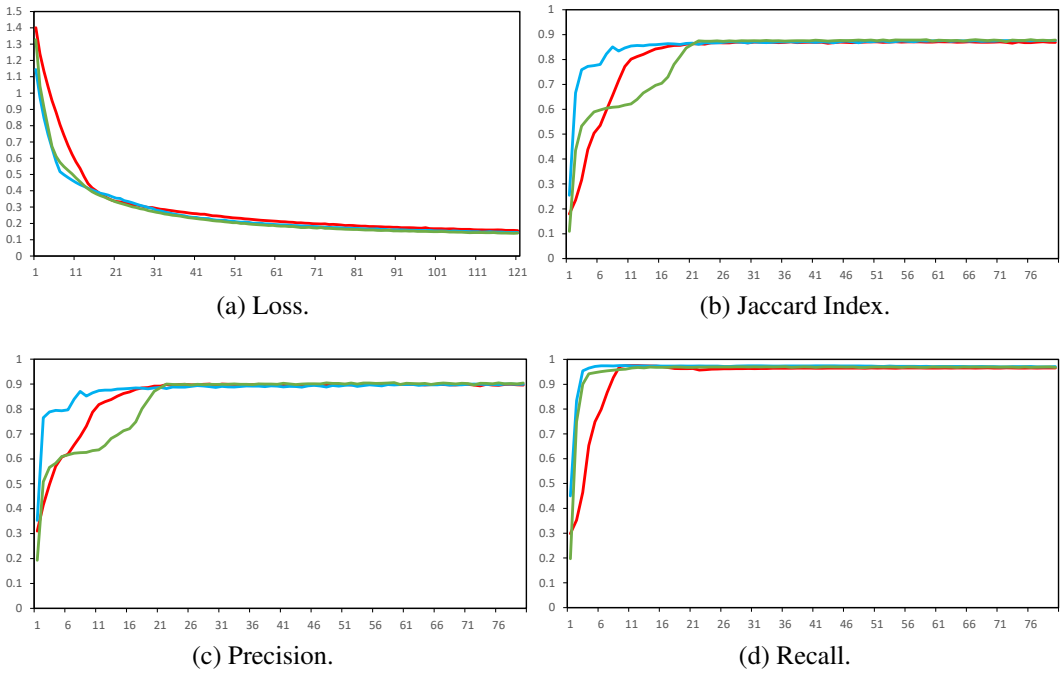
6.3.2 Refinement for Neural Networks

As a widely-applied refinement tool for CNN-based approaches, CRF has been proved to be effective in medical image segmentation, according to the references (232; 112; 33). Lai et al (90) integrate DenseCRF into their DeepSqueezeNet for better performance. Zheng et al (234) present a fusion strategy of DenseCRF and attention module for unsupervised video object segmentation. Shimoda et al (174) compare their proposed approach with the refinement output of CRF, and come to the conclusion that their approach is superior, which indicates the failure of CRF in the processing of certain datasets.

6.3.3 CRF as Neural Networks

Recently, several works emphasize on the possibilities of adapting CRF to neural network. CRF-RNN (235) is the first architecture that trains a DenseCRF end-to-end via Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). This work breaks down DenseCRF to common CNN operations except the lattice permutation during message passing of pairwise energy term, and trains these operations iteratively as a RNN. Vemulapalli et al (198) propose the Gaussian mean field approximation that computes pairwise energy into three steps - a CNN to generate features, a similarity layer that computes similarity scores, and a matrix generation layer that compute Gaussian weights. Although CNN is well-integrated into the GaussianCRF pairwise energy term, the Gaussian filtering process requires non-standard neural network operations.

Another variant is to compute CRF as a CNN. Teichmann and Cipolla (193) design a ConvCRF that constraints the fully-connected computational graph G into locally-connected blocks via Manhattan distance, which greatly boosts the efficiency of message passing process with the cost of losing long-range connections. Nguyen et al (132) take the advantages of Gated Recurrent Unit (26) (GRU), reformulating CRF-RNN as CRF-GRU that avoids the gradient vanishing and



(a) Loss. (b) Jaccard Index.

(c) Precision. (d) Recall.

Figure 6.4: Training Performance w.r.t. epochs.
Red curves: octant-size (60×80), blue curves: quarter-size (120×160), green curves: half-size (240×320). Jaccard Index, precision and recall are calculated by averaging all batches (120 images per batch) of an epoch. Three metrics reach their maximums at epoch 25.

exploding problem of RNN. The gated architecture also enables CRF-GRU to model long-term relationships. Lin et al (101) propose to learn unary and pairwise energies via fully-connected layers from node and edge feature vectors that are yielded from a manually-constructed CRF graph and a convolutional network.

[2]Specifically, the final softmax layer encapsules Gibbs energy term $E(\mathscr{L}|\mathscr{X})$ without the negative sign shown in (6.3), indicating the tendency of neural network to learn the maximal probability distribution $P(\mathscr{L}^*|\mathscr{X})$ via gradients back-propagation, whereas how to reach the optimal $\mathscr{L}^*$ remains to be solved.

[3]Note that $E_g(\mathscr{L}|\mathscr{X})$ is not finalized by an activation function after the weighted sum operation.

6.4 Methodology

6.4.1 Dilated CRF

Proposed DilatedCRF reformulates Gibbs energy term as fully-learnable unary $E_u(\mathbf{x}_i)$ and global potential[3] $E_g(\mathscr{L}|\mathscr{X})$:

$$E(\mathscr{L}|\mathscr{X}) = \underbrace{\sum_{1\leq i\leq N} \psi_u(\mathbf{x}_i)}_{E_u(\mathbf{x}_i)} + \underbrace{\sum_{1\leq i\leq N} \sum_{j=1}^{2} w_j^{(i)} \psi_j^{(i)}(\mathscr{L}|\mathscr{X})}_{E_g(\mathscr{L}|\mathscr{X})}. \tag{6.7}$$

In (6.7), $\psi_u(\mathbf{x}_i) = \mathbf{w}_u^T \mathbf{x}_i$, $\mathbf{w}_u$ is a weights vector with the same length as $\mathbf{x}_i$, which learns the unary term in (6.4). Particularly, $\mathscr{X}$ is the raw input affordance map, which can either be the artifacts after final softmax layer, or the output using conventional classifiers such as random forest (217). Unary potential ψ_u, global potentials ψ_1 and ψ_2 can be optimized iteratively during training. ψ_j is defined as

$$\psi_j(\mathscr{L}|\mathscr{X}) := \mathbf{W}_j \mathbf{f}_j(\mathscr{X}), \tag{6.8}$$

where $\mathbf{W}_i$ is a trainable weights matrix that up-samples global feature $\mathbf{f}_i$ to desired length $h \times w$. Moreover, $\mathbf{W}_i$ learns the mapping from $\mathscr{X}$ to potential $\psi_j(\mathscr{L}|\mathscr{X})$ which is not necessarily a valid probability distribution. Follow the DSConv architecture shown in Fig. 6.3, let d be the total number of convolution kernels, and k-th kernel has a convolutional stride of $s_k, 1 \leq k \leq d$, define $n := [\frac{N-1}{s_k} + 1]$ as the length of output feature, we have

$$\mathbf{f}_1(\mathscr{X}) = \frac{1}{n+1} \bigcup_{k=1}^{d} \sum_{\lambda=0}^{n} \mathbf{w}_{(1,\lambda)}^T \mathbf{x}_{1+\lambda s_k}, \tag{6.9}$$

$$\mathbf{f}_2(\mathscr{X}) = \bigcup_{k=1}^{d} \max_{0\leq\lambda\leq n} \mathbf{w}_{(2,\lambda)}^T \mathbf{x}_{1+\lambda s_k}, \tag{6.10}$$

the operator $\bigcup$ signifies the concatenation operation of elements. DilatedCRF is trained end-to-end to minimize KL-Divergence between $\mathscr{L}$ and the ground-truth. All weights are learnable through

gradients back-propagation since (6.7) - (6.10) are differential.

6.4.2 Architecture

Fig. 4.2 illustrates standard operations to perform DilatedCRF inference. We implement a 3×3 fractional max-pooling (54) layer to down-sample the input label, and nearest interpolation to up-sample refined feature maps. Each channel of the 3-channels unary energy term is computed via 1×1 convolution and batch normalization (BN) layer.

The main contribution of this chapter is the DSConv module that encodes the short-term and long-term relationship between feature vectors. As shown in Fig. 6.3, input feature volume is unfolded into a long vector with the shape of $(h' \times w', c, 1)$, sparse convolution using static kernel filtering has $d := [\sqrt{h' \times w'}/\alpha], \alpha = 2, 10$ strides to perform global correlations, which, in our experiment, we let $\alpha = 2$ for octant-size, quarter-size, half-size, and 10 for full-size DilatedCRFs. The AGAP operation (refer to (6.9)) computes average energy intensity, and AGMP operation (refer to (6.10)) locates the strongest response for feature $\mathbf{x}_i$. Both features effectively encode the contribution of $\mathbf{x}_i$ with less parameters than that of implementing two independent linear layers. Additionally, raw input feature is concatenated with the average global feature and maximal global feature for the purpose of reusing features and alleviating the vanishment of gradients. The linear up-sampling operation is based on (6.8), which is the most critical procedure to learn the relation between $\psi_j^{(i)}(\mathscr{L}|\mathscr{X})$ and $\mathbf{x}_i$. Equation (7) is realized by performing 1×1 convolution upon concatenated features.

6.4.3 Training

Assume the true conditional probability distribution of ground-truth is denoted as $G(\mathscr{L}^*|\mathscr{X})$, KL-Divergence is calculated by

$$\mathrm{KLD}(G||P) = \sum_{\mathscr{L}} G(\mathscr{L}^*|\mathscr{X}) \log \frac{G(\mathscr{L}^*|\mathscr{X})}{P(\mathscr{L}|\mathscr{X})}. \tag{6.11}$$

Items	Full-Size	Half-Size	LR-ASPP (67)
Total parameters (M)	**0.09**	21.66	-
Inference time (*ms*)	13.99	159.43	-
Accuracy (%)	96.56 ± **2.76**	96.22	**96.61** ± 2.80
Jaccard Index (%)	40.75 ± 23.42	40.77 ± 23.20	40.07 ± 23.72
Precision (%)	57.37 ± 28.12	53.09 ± 27.68	**58.16** ± 28.20
Recall (%)	58.47 ± 24.78	**64.42** ± 23.74	56.02 ± 25.76
Dice Coefficient (%)	53.94 ± 24.24	54.03 ± 24.01	53.07 ± 24.86

Table 6.1: DilatedCRF evaluation on test set (part. 1), all metrics are formatted as mean ± std. Our training samples are derived from state-of-the-art LR-ASPP (67), and average inferential runtime over test set is reported. Inference time of LR-ASPP is not presented because we only focus on the CRF-based post-processing.

Items	Quarter-Size	Octant-Size	LR-ASPP (67)
Total parameters (M)	2.77	0.36	-
Inference time (*ms*)	18.81	**8.91**	-
Accuracy (%)	96.35 ± 2.78	96.29 ± 2.71	**96.61** ± 2.80
Jaccard Index (%)	**41.21** ± 23.30	39.84 ± **22.67**	40.07 ± 23.72
Precision (%)	54.30 ± **27.58**	54.67 ± 28.23	**58.16** ± 28.20
Recall (%)	62.84 ± 23.85	61.11 ± **23.17**	56.02 ± 25.76
Dice Coefficient (%)	**54.45** ± 24.08	53.20 ± **23.69**	53.07 ± 24.86

Table 6.2: DilatedCRF evaluation on test set (part. 2), all metrics are formatted as mean ± std. Our training samples are derived from state-of-the-art LR-ASPP (67), and average inferential runtime over test set is reported. Inference time of LR-ASPP is not presented because we only focus on the CRF-based post-processing.

If G is one-hot encoded, KL-Divergence equals the value of cross-entropy loss (CELoss), we thereby utilize a weighted form of CELoss function for training DilatedCRF:

$$\text{CELoss}(l, l^*) = -\sum_{cls}(\beta_{cls}\sum_{cls}\log l), \tag{6.12}$$

where β_{cls} is a manually-set weight for the category cls, and the subscript variable $cls \in \{0,1,2\}$ indicates a category of {'background', 'negative samples', 'valid suckable area'}.

We trained the network by setting β to (0.25, 0.25, 0.5), the weight assigned for category 2 is larger than other weights because the valid suckable regions usually occupy a small portion of the image, while we only need to evaluate this category. The suction dataset (229) has a training split of 1470 images, and a testing split of 367 images, each image has a resolution of 480×640.

6.5 Experiments

6.5.1 Implementation Details

Training and test We follow a four-stages strategy to train and test DilatedCRF w/ a RTX3080 GPU:

- Pre-train classifiers on the training set.
- Generate predicated post-softmax[4] affordance maps for both training and test sets.
- Train DilatedCRF using the predicted affordance maps w/ AdamW optimizer (w/o weight decay) and a learning rate of 1×10^{-3}.
- Test the DilatedCRF using a selected checkpoint, and evaluate results by segmentation metrics.

Typically, it takes around 5 hours to train a half-size DilatedCRF. We quantitatively evaluate the results by Jaccard Index, Precision, Recall, Dice coefficient, and Accuracy, which are defined

[4] Alternatively, pre-softmax affordance maps are also theoretically valid.

Items	ConvCRF	DenseCRF
Inference time (*ms*)	14.10	450.16
Accuracy (%)	96.40 ± **2.84**	**96.74** ± 2.89
Jaccard Index (%)	**34.35** ± **22.95**	33.76 ± 25.63
Precision (%)	54.23 ± **29.17**	**60.49** ± 32.35
Recall (%)	**48.76** ± **28.43**	43.31 ± 30.74
Dice Coefficient (%)	**46.77** ± **25.94**	45.05 ± 28.75

Table 6.3: Performance of ConvCRF and DenseCRF on test set.
The metrics are formatted as mean ± std. We implement the same LR-ASPP unary as illustrated in Tab. 6.1 and 6.2 .

as

$$\textbf{Jaccard} = \frac{\mathrm{TP}}{\mathrm{FP}+\mathrm{FN}+\mathrm{TP}}, \tag{6.13}$$

$$\textbf{Accuracy} = \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{FP}+\mathrm{TP}+\mathrm{FN}+\mathrm{TN}}, \tag{6.14}$$

$$\textbf{Precision} = \frac{\mathrm{TP}}{\mathrm{FP}+\mathrm{TP}}, \ \textbf{Recall} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}, \tag{6.15}$$

$$\textbf{Dice} = \frac{2 \cdot \mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision}+\mathrm{Recall}}. \tag{6.16}$$

Performance in Training In Fig. 6.4, we report more training details for half-size, quarter-size and octant-size DilatedCRFs. Over-fitting is not observed as loss curves (in Fig. 6.4a) converge at around 0.11, and curves of metrics arrive at their maximums before the convergence.

It can also be inferred from Fig. 6.4b-6.4d that quarter-size DilatedCRF manifests the fastest convergence speed during training, and Jaccard Index curves behaves extremely similar to precision curves, which indicates precision is more decisive than recall for the suction dataset. Although differ in sizes, all three models converge to the same numerical level upon the training set.

Test performance As shown in Tab. 6.1 and 6.2, all DilateCRFs outperform the original LR-ASPP (67) approach on Recall and Dice scores, which indicates the potential of DilatedCRF in locating more objects, while maintaining high precision and lower uncertainties manifested by

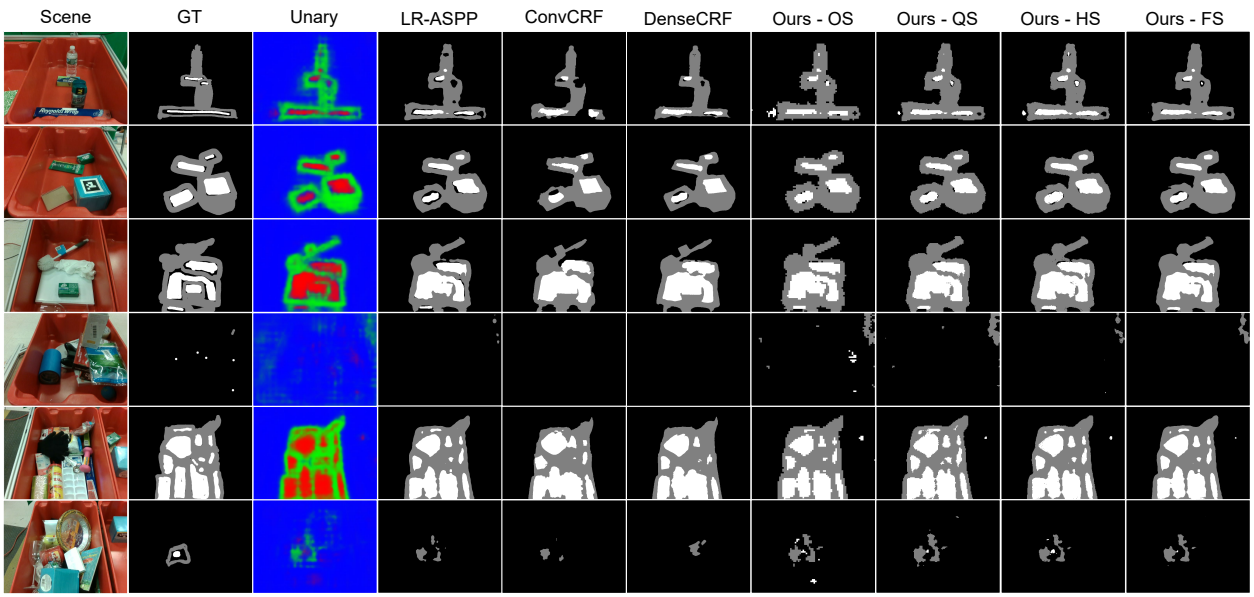


Figure 6.5: Visualization of affordance maps.
Black, grey, white regions correspond to background, negative samples, suction areas respectively. 'OS' - octant-size DilatedCRF, 'QS' - quarter-size DilatedCRF, 'HS' - half-size DilatedCRF, and 'FS' - full-size DilatedCRF.

standard deviation. Octant-size DilatedCRF has the coarsest affordance predictions for the $8\times$ lower down-sampling, however, it shows competitive performance both in Accuracy and Jaccard Index, and only takes additional $9ms$ inference runtime for refinement. The quarter-size DilatedCRF performs the best among all DilatedCRFs considering the trade-off between inference time and all metrics. Furthermore, it achieves the highest Dice score. Half-size DilatedCRFs continue to robustly improve Recall score whereas the twofold enlargements of image height and width greatly slows down inference procedure, bringing in more noisy features. For full-size DilatedCRF w/o the down-sampling and up-sampling layers shown in Fig. 4.2, coefficient α is set to 10, which dramatically boosts the runtime efficiency.

6.5.2 Compare to other CRFs

In this section, we report our quantitative and qualitative results with ConvCRF (193) and DenseCRF (85). ConvCRF is tested w/ a RTX3080 GPU, and DenseCRF is tested w/ a 3.60GHz Intel i9-10850K CPU.

Quantitative Evaluation As shown in Tab. 6.3, DenseCRF achieves the highest average precision 60.49% and accuracy 96.74% among all tested approaches. However, due to the largest variances, DenseCRF does not perform as stably as desired. Also, its lower recall and Jaccard index significantly degrade its performance on test set. ConvCRF, in the other hand, make a relatively better trade-off between recall and precision. It is noteworthy that, although ConvCRF has an average performance, it runs 32$\times$ faster than DenseCRF.

Qualitative Results Fig. 6.5 depicts affordance maps, images of scenes, and post-softmax unary maps. Both ConvCRF and DenseCRF enhance boundaries better than neural network based approaches. Nonetheless, the defects of these two approaches are obvious. First, they tend to filter out small regions, which may lower the success rate of suction-type picking. Second, the relationships between separated regions may not being well-encoded due to the chromatic continuities of raw scenes, which results in largely-connected affordance maps.

In comparison with ConvCRF and DenseCRF, the proposed approaches better recognize the relationships between separated regions, and locate more objects whereas some of them are not from the ground-truth. Although these predictions are categorized as false negatives, as depicted in Fig. 4.3, a robotic hand may successfully pick an object from bins under the guidance of these predictions. Generally, the proposed approaches directly learn vanilla features from unary maps, progressively refine memorized features towards the ground-truth. Furthermore, we test the checkpoint after 200 epoches, and discover a similar performing trend as of DenseCRF, i.e., higher accuracy and precision but lower recall and Jaccard index than LR-ASPP artifacts, which indicates the potential of DSConv module to approximate the global optimum of fully-connect CRF.

6.6 Conclusion and Future Works

We propose DilatedCRF to approximate fully-connected CRF without relying on mean-field approximation. DilatedCRF reformulate unary and pairwise energy terms as learnable unary and global energy terms. The reformulation adapts the end-to-end neural networks to conventional

CRF without CPU-only operations, which is faster and more liable for real-world implementations. To realize this reformulation, we design a dilated sparse convolution module that takes advantages of adaptive global average-pooling and adaptive global max-pooling, as well as the 3-stages DilatedCRF pipeline to accommodate varieties of unary maps. Extensive experimental results show the high efficiency, and competitive performance of the DilateCRF pipeline for realtime robotic suction-type bin-picking.

Our future works include:

- Apply highly-precise backbone networks to better learn the unary terms.
- Use network adaptive searching strategies to find the best strides for DSConv module.
- Integrate high-order contextual information into global energy terms, this may further enhance the regional encoding capabilities of DilatedCRF.
- Speed up the inferential procedures of full-size and half-size models by reducing concatenation operations.

Chapter 7

More Future Works: Knowledge Repository

Abstract

Reinforcement learning is a powerful robotic learning tool by interacting with the environment. Visual clues are crucial for agents to learn about the environment without direct contact. However, visual clues are not perfect, this is the mission of knowledge repository to utilize the clues and refine them. Knowledge repository has two main functions. The first one is to store knowledge of models and inferential results to help evolving visual clues. The second is to update models and make decisions/changes on action graph and multi task is preferred.

7.1 Introduction to Reinforcement Learning

Supervised/semi-supervised/unsupervised machine learning approaches aim at exploring the features encoded by datasets using computational models. Reinforcement learning (RL, 187) is a different category of machine learning methodology that doesn't rely on certain dataset or model, but actively learn states and takes the rewarded actions to better interact with an environment, which has long been implemented by human brain (reinforcement, (35)). RL is realized via greedy strategies to maximize the pay-off and randomized strategies such as the Markov decision process (MDP, 81) that predicts the best action policy for delayed reinforcement and interval-based techniques on second-order information of certainty or variance upon the values of actions. RL can

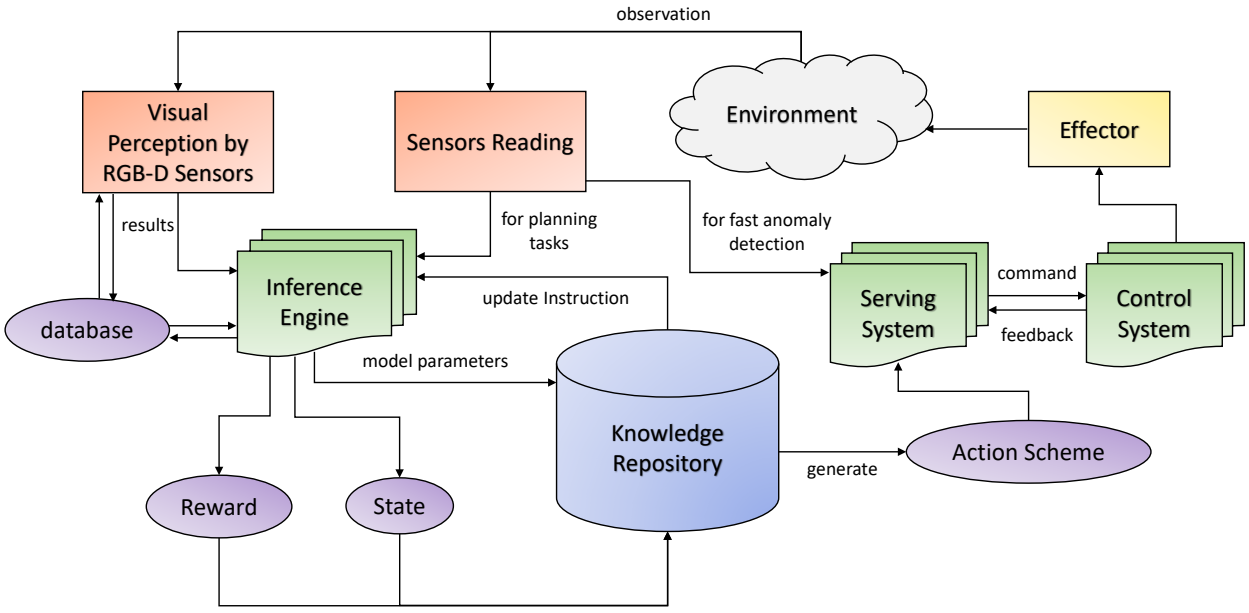


Figure 7.1: Robotic system powered by knowledge repository for reinforcement learning.

also be achieved via leaning a model[1], which indicates the possibilities of combining CNN-based neural networks into the routine of RL.

7.2 Technique Route

As mentioned in previous chapters, CNN can be involved in the inference process as well as the post-processing process. It should be noted that, even the most state-of-the-art approaches are to explore the upper bounds of given compute vision tasks, such that the deduced visual clues could be more reliable for real-world implementations. The visual clues can be the direct semantic results such as the description of an image, affordance maps, bounding boxes, or the features that can be forged into higher-level artifacts. The visual clues can be improved under the framework of RL, as depicted in Fig. 7.1.

[1]Currently there's no absolute conclusion on the superiority of model-based RL approaches over model-free approaches.

7.2.1 Environment

As the source of knowledge, environment is an objective entity composed of knowledge from different levels, such as the rigidity of materials, colors, illumination, temperature, humidity, distance, motion states, and the underlying emotion, psychic states, intents of human beings or animals etc. All these knowledge can be quantified so as to be transformed into data. Visual clues, however, only occupies a small portion of the environment.

7.2.2 Observations

Visual clues are the direct result of observations. The concept observation under the context of robotic implementation is also termed as the reading of sensors. Above all, observations are the knowledge collected from the environment. The quality of sensors defines the quality of observations. In particular, as CNN is directly related to visual clues, the visual perception is regarded as an independent branch as shown in Fig. 7.1.

7.2.3 Inference Engine

The inference engine is the preliminary processor that collects knowledge of observations and outputs low-level visual clues with the assistance of a database. In the meantime, the inference engine also performs necessary computations on the state of environment and estimate short-long term rewards for given tasks based on the visual clues. Though multi-functional the inference engine is, it does provide the optimal control policy not on the action strategy for the agents but more facts on the environment.

7.2.4 Knowledge Repository

The tasks for knowledge repository are to infer optimal strategies for the actions as well as for the policies to update visual clues. It is easier to understand the optimal strategies for actions using RL theories. However, the optimal visual clue is hard to define. From the perspective of tasks, optimal

visual clues are the most helpful evidences for the environment that the agents can perform the simplest actions to achieve the ultimate goal. For instance, a robotic hand can always use the least movements to grasp an object using affordance maps with 100% success rate, then the affordance maps can be defined as the optimal visual clues. As to the visual clue in the context of computer vision, we hope the visual clues are consistent and even more excellent than human's inference. Unfortunately, fewer visual clues have achieves the anticipation. To conclude, the optimal visual clues should meet both criterions.

Knowledge repository refines the visual clues according to the performance of visual clues in practice, as well as improves the quality of visual clues themselves. To achieve this goal, the models stored in the inference engine should be shared with knowledge repository such that knowledge repository can gain full control of the models and their hyper-parameters. The most crucial part is to establish the policy to update the models and fine-tune the parameters based on the rewards and states of the environment.

The other task of knowledge repository is to infer the optimal action scheme for the agents. There are many works (187) in this field. This indicates that knowledge repository can leverage on the benefits of both model-based and model-free approaches.

7.2.5 The Agent

Action scheme directly controls the serving system, which sends quantified controlling signals to the control system and effectors and collect feedback from the control system to achieve the optimal control. The basic actions include translation and rotation, and complex behaviors such as grasping, staring, following, dancing, and pouring water etc. The effector is the last stage of the agent, which directly interacts with the environment. In case of anomaly, such as when an effector grasps an egg and the egg tends to break under improper stress, the sensors immediately send abnormal readings to inference engine, and the inference engine will set the exceptional handling as the first priority, which means it sends direct commands to serving system to stop the anomaly. This is the last piece to fit the whole picture.

Chapter 8

Summary of Contribution and Discussion

8.1 Summary of Contribution

This chapter concludes the contributions of my work at each individual stage. Implementation of CNN is a broad and complicated topic. My study applies CNN approaches in pre-processing, inference, and post-processing phases in computer vision.

Given a fully-convolutional neural network whose final activation layer is directly linked to the loss function layer, strided convolution (Corollary 2.2.0.1) is proposed in Chapter 2.7 to demonstrate the gradients back-propagation from the loss function to the last hidden layer. By solving the dual problem of convolution, inverted strided convolution (Corollary 2.2.0.1) is proposed to express the gradients propagated between two consecutive hidden layers. In Subsection. 2.7, to our best knowledge, it is the first work that proposes to implement a pre-processing transformer (BR-Transformer) for the task of PAF event detection from ECG records. Among the pre-processing steps of BRTransformer, conceptualization and fuzzification are crucial to establish connections between semantic categories and intervals of R-peaks.

In Chapter 3, we proposed a novel stereo-frustums module for classifiers based on stereopair and LiDAR point clouds inputs. Stereo-Frustums module is regarded as the pre-processing technique that matches 2D proposals from left-view and right-view in different styles, as well as reduces the number of LiDAR points fed to its following regression network via segmentation by projection. The proposed module can segment the LiDAR points of scene efficiently without being accelerated by the optimization of compilers and existing toolboxes, which is listed among our future works. Experiments on KITTI benchmark have shown the superior performance of proposed

pre-processing module over stereo-only based approaches in pedestrian and car detection.

Chapter 4 and 5 present my work at the inference phase. In Chapter 4, we proposed to solve the polyp detection problem using Faster R-CNN. As the first work that implemented Faster R-CNN, the popular CNN-based 2D detector in the field of computer vision to the task of polyp detection in the field of medical image processing, we evaluated the performance of polyp detection using convincing metrics and public datasets. Our evaluation is able to identify the advantages and challenges of CNN-based 2D detectors over real-world polyp videos.

In Chapter 5, we proposed an end-to-end realtime global self-attention based neural network (RGANet) for the task of image segmentation. In particular, we emphasize its implementation in the robotic bin-pick task by suction. RGANet outperforms top-tier real-time segmentation approaches. It also achieves competitive performance against larger models. During the evaluation phase, we proposed an evaluation metic, MGRID, to mitigate the negative effect of widely-scattered prediction. Our metric can greatly reduce the necessity of manually annotating every instance in an image.

Implementation of CNN in post-processing phase is presented in Chapter 6. We designed an end-to-end DilatedCRF for the purpose of boosting both the quantitative and qualitative performance of image segmentation by classifiers. The main contribution of DilatedCRF is the approximation of pairwise energy term of a regular CRF using proposed DSConv module that leverages on adaptive global maximum pooling and adaptive global average pooling techniques.

Beyond the implementation of CNN, the future work of knowledge repository is introduced in Chapter 7. Knowledge repository can fuse the CNN-based models that infer visual clues into the architecture of typical reinforcement learning. The advantages of knowledge repository not only lie in its fine-tuning the hyper-parameters of CNN-based models towards high-quality visual clues, but also constructing an action graph that enables an agent's better problem-solving ability in its interaction with the environment.

8.2 Discussion

CNN approaches are powerful and efficient when being applied in the context of computer vision. But they have limitations in general. Corollary 2.2.0.1 and Corollary 2.2.0.1 are designed for the framework of standard fully convolutional neural network with activation functions, they cannot be directly implemented to other types of convolutions, nor to the multi-branch architecture framework. BRTransformer better works with 1-D ECG signal that depicts the very salient R-peaks. The size of on typical checkpoint is $565MB$ such that it may be inappropriate to directly deploy BRTransformer onto a mobile system. Faster R-CNN based 2-D detector for polyp detection only takes no more than one proposal with the highest score into consideration, which indicates that there should be only one polyp at a time. Stereo frustums module may not be able to find correct matches sometimes. In this case, single frustum pre-processing is recommended. Also, Stereo frustums based 3-D object detection heavily relies on the quality of 2-D bounding boxes. Therefore, these 2-D proposals greatly affect the performance of stereo frustums. RGANet is designed for semantic segmentation with few categories. For multi-classification tasks, categories with the highest proportions may dominate such that other categories tend to be merged into those categories. The limitation of DilatedCRF is its weak robustness against noisy unary energy maps since the adaptive global max-pooling operation in DSConv module is liable to memorizing the positions of noise.

Mimicking biological neurons and their operation mechanism in machine learning has always been challenging. Modern CNN approaches have evolved fast in theories and techniques. Where lies the limitations today lies the research opportunities in the future.

References

[1] 2021, I. (2021). Paroxysmal atrial fibrillation events detection from dynamic ecg recordings: The 4th china physiological signal challenge 2021. http://www.icbeb.org/CPSC2021. Accessed 11/7/2021.

[2] Abdi, H. & Williams, L. J. (2010). Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4), 433–459.

[3] Ackley, D. H., Hinton, G. E., & Sejnowski, T. J. (1985). A learning algorithm for boltzmann machines. *Cognitive science*, 9(1), 147–169.

[4] Adams, A., Baek, J., & Davis, M. A. (2010). Fast high-dimensional filtering using the permutohedral lattice. In *Computer Graphics Forum*, volume 29 (pp. 753–762).

[5] Albisser, Z., Riegler, M., Halvorsen, P., Zhou, J., Griwodz, C., Balasingham, I., & Gurrin, C. (2015). Expert Driven Semi-Supervised Elucidation Tool for Medical Endoscopic Videos. In *Proceedings of the 6th ACM Multimedia Systems Conference* (pp. 73–76).

[6] Azpiri, A. I., Ortega, E. L., & Cobo, A. A. (2021). Affordance-based grasping point detection using graph convolutional networks for industrial bin-picking applications. *Sensors*, 21(3), 816.

[7] Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. *stat*, 1050, 21.

[8] Bacciotti, A. & Rosier, L. (2005). *Liapunov functions and stability in control theory*. Springer Science & Business Media.

[9] Bao, W., Xu, B., & Chen, Z. (2019). Monofenet: Monocular 3d object detection with feature enhancement networks. *Transactions on Image Processing*.

[10] Barz, B. & Denzler, J. (2020). Do we train on test data? purging cifar of near-duplicates. *Journal of Imaging*, 6(6), 41.

[11] Bello, I., Zoph, B., Vaswani, A., et al. (2019). Attention augmented convolutional networks. In *Proceedings of the International Conference on Computer Vision (ICCV)* (pp. 3286–3295).

[12] Bernal, J., Sánchez, F. J., Fernández-Esparrach, G., Gil, D., Rodríguez, C., & Vilariño, F. (2015). WM-DOVA Maps for Accurate Polyp Highlighting in Colonoscopy: Validation vs. Saliency Maps from Physicians. *Computerized Medical Imaging and Graphics*, 43, 99–111.

[13] Bernal, J., Sánchez, J., & Vilarino, F. (2012). Towards Automatic Polyp Detection with a Polyp Appearance Model. *Pattern Recognition*, 45(9), 3166–3182.

[14] Bernal, J., Tajkbaksh, N., Sánchez, F. J., Matuszewski, B. J., Chen, H., Yu, L., Angermann, Q., Romain, O., Rustad, B., Balasingham, I., Pogorelov, K., Choi, S., Debard, Q., Maier-Hein, L., Speidel, S., Stoyanov, D., Brandao, P., Córdova, H., Sánchez-Montes, C., Gurudu, S. R., Fernández-Esparrach, G., Dray, X., Liang, J., & Histace, A. (2017). Comparative Validation of Polyp Detection Methods in Video Colonoscopy: Results from the MICCAI 2015 Endoscopic Vision Challenge. *IEEE Transactions on Medical Imaging*, 36(6), 1231–1249.

[15] Besag, J. (1974). Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 36(2), 192–225.

[16] Boyd, S., Boyd, S. P., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

[17] Broomhead, D. S. & Lowe, D. (1988). *Radial basis functions, multi-variable functional interpolation and adaptive networks*. Technical report, Royal Signals and Radar Establishment Malvern (United Kingdom).

[18] Bruck, J. (1990). On the convergence properties of the hopfield model. *Proceedings of the IEEE*, 78(10), 1579–1585.

[19] Burns, E. & Buttner, R. (2021). Atrial fibrillation. https://litfl.com/atrial-fibrillation-ecg-library/. Accessed 11/7/2021.

[20] Butterworth, S. et al. (1930). On the theory of filter amplifiers. *Wireless Engineer*, 7(6), 536–541.

[21] Cao, Y., Xu, J., Lin, S., et al. (2019). Gcnet: Non-local networks meet squeeze-excitation networks and beyond. In *Proceedings of the International Conference on Computer Vision Workshops*.

[22] Castleman, K. R. (1996). *Digital image processing*. Prentice Hall Press.

[23] Chao, P., Kao, C.-Y., Ruan, Y.-S., et al. (2019). Hardnet: A low memory traffic network. In *Proceedings of the International Conference on Computer Vision (ICCV)* (pp. 3552–3561).: IEEE/CVF.

[24] Chen, W., Liu, T.-Y., Lan, Y., Ma, Z.-M., & Li, H. (2009). Ranking measures and loss functions in learning to rank. *Advances in Neural Information Processing Systems*, 22.

[25] Chen, X., Kundu, K., Zhu, Y., Ma, H., Fidler, S., & Urtasun, R. (2017). 3d object proposals using stereo imagery for accurate object class detection. *Transactions on Pattern Analysis and Machine Intelligence*, 40(5), 1259–1272.

[26] Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*.

[27] Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 1251–1258).: IEEE.

[28] Clifford, P. (1990). Markov random fields in statistics. *Disorder in physical systems: A volume in honour of John M. Hammersley*, (pp. 19–32).

[29] Cortes, C. & Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3), 273–297.

[30] Costa, R., Winkert, T., Manhães, A., & Teixeira, J. P. (2021). Qrs peaks, p and t waves identification in ecg. *Procedia Computer Science*, 181, 957–964.

[31] Dahl, G. E., Sainath, T. N., & Hinton, G. E. (2013). Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE international conference on acoustics, speech and signal processing* (pp. 8609–8613).: IEEE.

[32] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6), 141–142.

[33] Deng, W., Shi, Q., Wang, M., Zheng, B., & Ning, N. (2020). Deep learning-based hcnn and crf-rrnn model for brain tumor segmentation. *IEEE Access*, 8, 26665–26675.

[34] Dobruschin, P. (1968). The description of a random field by means of conditional probabilities and conditions of its regularity. *Theory of Probability & Its Applications*, 13(2), 197–224.

[35] Donahoe, J. W., Palmer, D. C., et al. (1993). A selectionist approach to reinforcement. *Journal of the Experimental Analysis of Behavior*, 60, 17–40.

[36] Du, X., Ang, M. H., Karaman, S., & Rus, D. (2018). A general pipeline for 3d detection of vehicles. In *IEEE International Conference on Robotics and Automation* (pp. 3194–3200).

[37] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7).

[38] Dumoulin, V. & Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

[39] Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2), 179–211.

[40] en by Tom Seymour (2021). What's to know about paroxysmal atrial fibrillation? https://www.medicalnewstoday.com/articles/316281. Accessed 11/17/2021.

[41] Fan, M., Lai, S., Huang, J., et al. (2021). Rethinking bisenet for real-time semantic segmentation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 9716–9725).: IEEE/CVF.

[42] Fernandez, J., Harris, M., & Meyer, C. (2005). Combining algorithms in automatic detection of r-peaks in ecg signals. In *18th IEEE Symposium on Computer-Based Medical Systems (CBMS'05)* (pp. 297–302).: IEEE.

[43] Florian, L.-C. C. G. P. & Adam, S. H. (2017). Rethinking atrous convolution for semantic image segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*: IEEE/CVF.

[44] Fukushima, K. & Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets* (pp. 267–285). Springer.

[45] Fushiki, T. (2011). Estimation of prediction error by using k-fold cross-validation. *Statistics and Computing*, 21(2), 137–146.

[46] Geiger, A., Lenz, P., Stiller, C., & Urtasun, R. (2013). Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11), 1231–1237.

[Geiger et al.] Geiger, A., Lenz, P., & Urtasun, R. Official KITTI benchmark. Accessed: 2019-11-19.

[48] Geman, S. & Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6, 721–741.

[49] Gil, D., Sánchez, F. J., Fernández-Esparrach, G., & Bernal, J. (2016). 3D Stable Spatio-Temporal Polyp Localization in Colonoscopy Videos. In *International Workshop on Computer-*

Assisted and Robotic Endoscopy, volume 9515 of *Lecture Notes in Computer Science* (pp. 140–152).

[50] Girosi, F., Jones, M., & Poggio, T. (1995). Regularization theory and neural networks architectures. *Neural computation*, 7(2), 219–269.

[51] Girshick, R. (2015). Fast R-CNN. In *IEEE Conference on Computer Vision and Pattern Recognition* (pp. 1440–1448).

[52] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition* (pp. 580–587).

[53] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, 27.

[54] Graham, B. (2014). Fractional max-pooling. *arXiv:1412.6071*.

[55] Grimmett, G. R. (1973). A theorem about random fields. *Bulletin of the London Mathematical society*, 5(1), 81–84.

[56] Hampshire, J. & Waibel, A. (1989). Connectionist architectures for multi-speaker phoneme recognition. *Advances in neural information processing systems*, 2.

[57] He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017a). Mask r-cnn. In *IEEE International Conference on Computer Vision* (pp. 2961–2969).

[58] He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017b). Mask R-CNN. *IEEE International Conference on Computer Vision*, (pp. 2980–2988).

[59] He, K., Zhang, X., Ren, S., & Sun, J. (2014). Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In *European Conference on Computer Vision* (pp. 346–361).

[60] Hebb, D. O. (2005). *The organization of behavior: A neuropsychological theory*. Psychology Press.

[61] Hinton, G. E. (2009). Deep belief networks. *Scholarpedia*, 4(5), 5947.

[62] Hinton, G. E. & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786), 504–507.

[63] Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735–1780.

[64] Hong, Y., Pan, H., Sun, W., et al. (2021). Deep dual-resolution networks for real-time and accurate semantic segmentation of road scenes. *arXiv preprint arXiv:2101.06085*.

[65] Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8), 2554–2558.

[66] Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the national academy of sciences*, 81(10), 3088–3092.

[67] Howard, A., Sandler, M., Chu, G., et al. (2019). Searching for mobilenetv3. In *Proceedings of the International Conference on Computer Vision (ICCV)* (pp. 1314–1324).: IEEE/CVF.

[68] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

[69] Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 7132–7141).: IEEE/CVF.

[70] Huang, G., Liu, Z., Van Der Maaten, L., et al. (2017). Densely connected convolutional networks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 4700–4708).: IEEE.

[71] Huang, Z., Wang, X., Huang, L., et al. (2019). Ccnet: Criss-cross attention for semantic segmentation. In *Proceedings of the International Conference on Computer Vision (ICCV)* (pp. 603–612).: IEEE/CVF.

[72] Hubel, D. H. & Wiesel, T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3), 574.

[73] Hubel, D. H. & Wiesel, T. N. (1968). Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1), 215–243.

[74] Huber, P. J. (1992). Robust estimation of a location parameter. In *Breakthroughs in statistics* (pp. 492–518). Springer.

[75] Huo, J., Wu, J., Cao, J., & Wang, G. (2018). Supervoxel Based Method for Multi-Atlas Segmentation of Brain MR Images. *NeuroImage*, 175, 201–214.

[76] Ioffe, S. & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)* (pp. 448–456).: PMLR.

[77] Iwahori, Y., Hattori, A., Adachi, Y., Bhuyan, M. K., Woodham, R. J., & Kasugai, K. (2015). Automatic Detection of Polyp Using Hessian Filter and HOG Features. *Procedia Computer Science*, 60, 730–739.

[78] January, C. T., Wann, L. S., Alpert, J. S., Calkins, H., Cigarroa, J. E., Cleveland, J. C., Conti, J. B., Ellinor, P. T., Ezekowitz, M. D., Field, M. E., et al. (2014). 2014 aha/acc/hrs guideline for the management of patients with atrial fibrillation: executive summary: a report of the american

college of cardiology/american heart association task force on practice guidelines and the heart rhythm society. *Journal of the American College of Cardiology*, 64(21), 2246–2280.

[79] January, C. T., Wann, L. S., Calkins, H., Chen, L. Y., Cigarroa, J. E., Cleveland, J. C., Ellinor, P. T., Ezekowitz, M. D., Field, M. E., Furie, K. L., et al. (2019). 2019 aha/acc/hrs focused update of the 2014 aha/acc/hrs guideline for the management of patients with atrial fibrillation: a report of the american college of cardiology/american heart association task force on clinical practice guidelines and the heart rhythm society. *Journal of the American College of Cardiology*, 74(1), 104–132.

[80] Jordan, M. I. (1997). Serial order: A parallel distributed processing approach. In *Advances in Psychology*, volume 121 (pp. 471–495). Elsevier.

[81] Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4, 237–285.

[82] Khatib, A. E., Werghi, N., & Al-Ahmad, H. (2016). Enhancing Automatic Polyp Detection Accuracy Using Fusion Techniques. In *IEEE 59th International Midwest Symposium on Circuits and Systems* (pp. 1–4).

[83] Kingma, D. P. & Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

[84] Königshof, H., Salscheider, N. O., & Stiller, C. (2019). Realtime 3 d object detection for automated driving using stereo vision and semantic information. In *IEEE International Conference on Intelligent Transportation Systems*.

[85] Krähenbühl, P. & Koltun, V. (2011). Efficient inference in fully connected crfs with gaussian edge potentials. *Advances in neural information processing systems*, 24, 109–117.

[86] Ku, J., Mozifian, M., Lee, J., Harakeh, A., & Waslander, S. L. (2018). Joint 3d proposal

generation and object detection from view aggregation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 1–8).

[87] Kulis, B. et al. (2013). Metric learning: A survey. *Foundations and Trends® in Machine Learning*, 5(4), 287–364.

[88] Kullback, S. & Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1), 79–86.

[89] Kwon, O., Jeong, J., Kim, H. B., Kwon, I. H., Park, S. Y., Kim, J. E., & Choi, Y. (2018). Electrocardiogram sampling frequency range acceptable for heart rate variability analysis. *Healthcare informatics research*, 24(3), 198–206.

[90] Lai, D., Deng, Y., & Chen, L. (2019). Deepsqueezenet-crf: A lightweight deep model for semantic image segmentation. In *International Joint Conference on Neural Networks* (pp. 1–8).

[91] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, 521(7553), 436–444.

[92] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.

[93] Li, H., Xiong, P., An, J., & Wang, L. (2018). Pyramid attention network for semantic segmentation. *British Machine Vision Conference (BMVC)*.

[94] Li, K., Ma, W., Sajid, U., Wu, Y., & Wang, G. (2020a). 2 object detection with convolutional neural networks. *Deep Learning in Computer Vision: Principles and Applications*, 30(31), 41.

[95] Li, P., Chen, X., & Shen, S. (2019). Stereo r-cnn based 3d object detection for autonomous driving. In *IEEE Conference on Computer Vision and Pattern Recognition* (pp. 7644–7652).

[96] Li, S. Z. (1994). Markov random field models in computer vision. In *European Conference on Computer Vision (ECCV)* (pp. 361–370).: Springer.

[97] Li, Y., Liu, Y., Liu, G., & Guo, M. (2020b). Weakly supervised semantic segmentation by iterative superpixel-crf refinement with initial clues guiding. *Neurocomputing*, 391, 25–41.

[98] Li, Y. & Yang, T. (2018). Word embedding for understanding natural language: a survey. In *Guide to big data applications* (pp. 83–104). Springer.

[99] Liang, M., Yang, B., Wang, S., & Urtasun, R. (2018). Deep continuous fusion for multi-sensor 3d object detection. In *European Conference on Computer Vision* (pp. 641–656).

[100] Liang, Y., Ryali, C., Hoover, B., Grinberg, L., Navlakha, S., Zaki, M. J., & Krotov, D. (2020). Can a fruit fly learn word embeddings? In *International Conference on Learning Representations*.

[101] Lin, G., Shen, C., Van Den Hengel, A., & Reid, I. (2016). Efficient piecewise training of deep structured models for semantic segmentation. In *Computer Vision and Pattern Recognition* (pp. 3194–3203).

[102] Lin, M., Chen, Q., & Yan, S. (2013). Network in network. *arXiv preprint arXiv:1312.4400*.

[103] Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (pp. 2980–2988).

[104] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). SSD: Single Shot Multibox Detector. In *European Conference on Computer Vision* (pp. 21–37).

[105] Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 3431–3440).: IEEE/CVF.

[106] Loshchilov, I. & Hutter, F. (2018a). Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*.

[107] Loshchilov, I. & Hutter, F. (2018b). Fixing weight decay regularization in adam. In *International Conference on Learning Representations (ICLR)*.

[108] Lowe, D. (2015). Radial basis function networks-revisited. *Mathematics Today*, 51(3), 124–126.

[109] Luo, C., Zhan, J., Xue, X., Wang, L., Ren, R., & Yang, Q. (2018a). Cosine normalization: Using cosine similarity instead of dot product in neural networks. In *International Conference on Artificial Neural Networks* (pp. 382–391).: Springer.

[110] Luo, P., Ren, J., Peng, Z., Zhang, R., & Li, J. (2018b). Differentiable learning-to-normalize via switchable normalization. *arXiv preprint arXiv:1806.10779*.

[111] Luo, W., Yang, B., & Urtasun, R. (2018c). Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net. In *IEEE conference on Computer Vision and Pattern Recognition* (pp. 3569–3577).

[112] Luo, W. & Yang, M. (2018). Fast skin lesion segmentation via fully convolutional network with residual architecture and crf. In *International Conference on Pattern Recognition* (pp. 1438–1443).

[113] Ma, F., Gao, F., Sun, J., Zhou, H., & Hussain, A. (2019). Weakly supervised segmentation of sar imagery using superpixel and hierarchically adversarial crf. *Remote Sensing*, 11(5), 512.

[114] Ma, W., Wu, Y., Cen, F., & Wang, G. (2020). Mdfn: Multi-scale deep feature learning network for object detection. *Pattern Recognition*, 100, 107149.

[115] Maas, A. L., Hannun, A. Y., Ng, A. Y., et al. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of International Conference on Machine Learning (ICML)*, volume 30 (pp.$\tilde{3}$).

[116] Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9), 1659–1671.

[117] Macqueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proc. the Fifth Berkeley Symp. on Math. Statics and Prob., 1967*, volume 1 (pp. 281–297).: Univ. of California Press.

[118] Maiers, J. & Sherif, Y. S. (1985). Applications of fuzzy set theory. *IEEE Transactions on Systems, Man, and Cybernetics*, 1, 175–189.

[119] Mamonov, A. V., Figueiredo, I. N., Figueiredo, P. N., & Tsai, Y. H. R. (2014). Automated Polyp Detection in Colon Capsule Endoscopy. *IEEE Transactions on Medical Imaging*, 33(7), 1488–1502.

[120] Manikandan, M. S. & Soman, K. (2012). A novel method for detecting r-peaks in electrocardiogram (ecg) signal. *Biomedical Signal Processing and Control*, 7(2), 118–128.

[121] Merah, M., Abdelmalik, T., & Larbi, B. (2015). R-peaks detection based on stationary wavelet transform. *Computer methods and programs in biomedicine*, 121(3), 149–160.

[122] Mo, X. & Chen, X. (2022). Realtime global attention network for semantic segmentation. *IEEE Robotics and Automation Letters*.

[123] Mo, X., Chen, X., Zhong, C., Li, R., Li, K., & Sajid, U. (2022). Dilated continuous random field for semantic segmentation. *arXiv preprint arXiv:2202.00162*.

[124] Mo, X., Sajid, U., & Wang, G. (2021). Stereo frustums: a siamese pipeline for 3d object detection. *Journal of Intelligent & Robotic Systems*, 101(1), 1–15.

[125] Mo, X., Tao, K., Wang, Q., & Wang, G. (2018). An efficient approach for polyps detection in endoscopic videos based on faster r-cnn. In *2018 24th international conference on pattern recognition (ICPR)* (pp. 3929–3934).: IEEE.

[126] Moody, G. (2004). Spontaneous termination of atrial fibrillation: a challenge from physionet and computers in cardiology 2004. In *Computers in Cardiology, 2004* (pp. 101–104).: IEEE.

[127] Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

[128] Mycielski, J. (1972). Marvin minsky and seymour papert, perceptrons, an introduction to computational geometry. *Bulletin of the American Mathematical Society*, 78(1), 12–15.

[129] Nair, V. & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning (ICML)*.

[130] Nam, H. & Kim, H.-E. (2018). Batch-instance normalization for adaptively style-invariant neural networks. *Advances in Neural Information Processing Systems*, 31.

[131] Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In *Doklady an ussr*, volume 269 (pp. 543–547).

[132] Nguyen, K., Fookes, C., & Sridharan, S. (2020). Context from within: Hierarchical context modeling for semantic segmentation. *Pattern Recognition*, 105, 107358.

[133] Novikoff, A. B. (1963). *On convergence proofs for perceptrons*. Technical report, STANFORD RESEARCH INST MENLO PARK CA.

[134] Oppenheim, A. V., Willsky, A. S., Nawab, S. H., Hernández, G. M., et al. (1997). *Signals & systems*. Pearson Educación.

[135] paperswithcode.com (2021). real-time semantic segmentation leader board. https://paperswithcode.com/sota/real-time-semantic-segmentation-on-cityscapes. Accessed 9/2/2021.

[136] Patro, S. & Sahu, K. K. (2015). Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*.

[137] Penzel, T., Moody, G. B., Mark, R. G., Goldberger, A. L., & Peter, J. H. (2000). The apnea-ecg database. In *Computers in Cardiology 2000. Vol. 27 (Cat. 00CH37163)* (pp. 255–258).: IEEE.

[138] Pfeiffer, M. & Pfeil, T. (2018). Deep learning with spiking neurons: opportunities and challenges. *Frontiers in neuroscience*, (pp. 774).

[139] Pincus, M. (1970). A monte carlo method for the approximate solution of certain types of constrained optimization problems. *Operations research*, 18(6), 1225–1228.

[140] Pogorelov, K., Riegler, M., Eskeland, S. L., de Lange, T., Johansen, D., Griwodz, C., Schmidt, P. T., & Halvorsen, P. (2017). Efficient Disease Detection in Gastrointestinal Videos – Global Features versus Neural Networks. *Multimedia Tools and Applications*, 76(21), 22493–22525.

[141] Pon, A. D., Ku, J., Li, C., & Waslander, S. L. (2020). Object-centric stereo matching for 3d object detection. In *IEEE International Conference on Robotics and Automation* (pp. 8383–8389).

[142] Preston, C. J. (1973). Generalized gibbs states and markov random fields. *Advances in Applied probability*, 5(2), 242–261.

[143] Qi, C. R., Liu, W., Wu, C., Su, H., & Guibas, L. J. (2018). Frustum pointnets for 3d object detection from rgb-d data. In *IEEE Conference on Computer Vision and Pattern Recognition* (pp. 918–927).

[144] Qi, C. R., Su, H., Mo, K., & Guibas, L. J. (2017a). Pointnet: Deep learning on point sets for 3d classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition* (pp. 652–660).

[145] Qi, C. R., Yi, L., Su, H., & Guibas, L. J. (2017b). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems* (pp. 5099–5108).

[146] Qin, Z., Wang, J., & Lu, Y. (2019). Triangulation learning network: from monocular to stereo 3d object detection. In *IEEE Conference on Computer Vision and Pattern Recognition* (pp. 7607–7615).

[147] Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv e-prints*, (pp. arXiv–1511).

[148] Rajani Kumari, L., Padma Sai, Y., & Balaji, N. (2021). R-peak identification in ecg signals using pattern-adapted wavelet technique. *IETE Journal of Research*, (pp. 1–10).

[149] Ramachandran, P., Zoph, B., & Le, Q. V. (2018). Searching for activation functions. In *International Conference on Learning Representations (ICLR)*.

[150] Rao, K. D. (1997). Dwt based detection of r-peaks and data compression of ecg signals. *IETE Journal of Research*, 43(5), 345–349.

[151] Reddi, S., Kale, S., & Kumar, S. (2018). On the convergence of adam and beyond. In *International Conference on Learning Representations (ICLR)*.

[152] Redmon, J. & Farhadi, A. (2017). YOLO9000: Better, Faster, Stronger. *IEEE Conference on Computer Vision and Pattern Recognition*, (pp. 6517–6525).

[153] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems* (pp. 91–99).

[154] Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Rroposal Networks. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 39 (pp. 1137–1149).

[155] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention* (pp. 234–241).: Springer.

[156] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6), 386.

[157] Rosset, S., Zhu, J., & Hastie, T. (2003). Margin maximizing loss functions. *Advances in neural information processing systems*, 16.

[158] Roth, H. R., Lu, L., Liu, J., Yao, J., Seff, A., Cherry, K., Kim, L., & Summers, R. M. (2017). Efficient False Positive Reduction in Computer-Aided Detection Using Convolutional Neural Networks and Random View Aggregation. In *Deep Learning and Convolutional Neural Networks for Medical Image Computing*, Advances in Computer Vision and Pattern Recognition (pp. 35–48). Springer, Cham.

[159] Rumelhart, D., Hinton, G., & Williams, R. (1988). Learning internal representations by error propagation. In *Neurocomputing: foundations of research* (pp. 673–695). MIT Press.

[160] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088), 533–536.

[161] Sabherwal, P., Agrawal, M., & Singh, L. (2017). Automatic detection of the r peaks in single-lead ecg signal. *Circuits, Systems, and Signal Processing*, 36(11), 4637–4652.

[162] Salakhutdinov, R. & Larochelle, H. (2010). Efficient learning of deep boltzmann machines. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 693–700).: JMLR Workshop and Conference Proceedings.

[163] Salimans, T. & Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29.

[164] Sandler, M., Howard, A., Zhu, M., et al. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 4510–4520).: IEEE/CVF.

[165] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. *IEEE transactions on neural networks*, 20(1), 61–80.

[166] Schultz, M. & Joachims, T. (2003). Learning a distance metric from relative comparisons. *Advances in neural information processing systems*, 16.

[167] Seferbekov, S., Iglovikov, V., Buslaev, A., et al. (2018). Feature pyramid network for multi-class land segmentation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition Workshops* (pp. 272–275).

[168] Shao, Q., Hu, J., Wang, W., et al. (2019). Suction grasp region prediction using self-supervised learning for object picking in dense clutter. In *Proceedings of the International Conference on Mechatronics System and Robots (ICMSR)* (pp. 7–12).: IEEE.

[169] Sherman, S. (1973). Markov random fields and gibbs random fields. *Israel Journal of Mathematics*, 14(1), 92–103.

[170] Sherrington, D. & Kirkpatrick, S. (1975). Solvable model of a spin-glass. *Physical review letters*, 35(26), 1792.

[171] Shi, S., Wang, X., & Li, H. (2019). Pointrcnn: 3d object proposal generation and detection from point cloud. In *IEEE Conference on Computer Vision and Pattern Recognition* (pp. 770–779).

[172] Shi, S., Wang, Z., Shi, J., Wang, X., & Li, H. (2020). From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network. *Transactions on Pattern Analysis and Machine Intelligence*.

[173] Shi, W., Caballero, J., Theis, L., Huszar, F., Aitken, A., Ledig, C., & Wang, Z. (2016). Is the deconvolution layer the same as a convolutional layer? *arXiv e-prints*, (pp. arXiv–1609).

[174] Shimoda, W. & Yanai, K. (2019). Self-supervised difference detection for weakly-supervised semantic segmentation. In *International Conference on Computer Vision* (pp. 5208–5217).

[175] Shin, K., Kwon, Y. P., & Tomizuka, M. (2019). Roarnet: A robust 3d object detection based on region approximation refinement. In *IEEE Intelligent Vehicles Symposium (IV)* (pp. 2510–2515).

[176] Shorten, C. & Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of big data*, 6(1), 1–48.

[177] Simonyan, K. & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*.

[178] Smolensky, P. (1985). Chapter 6: information processing in dynamical systems: foundations of harmony theory. *Parallel distributed processing: explorations in the microstructure of cognition*, 1.

[179] Spitzer, F. (1971). Markov random fields and gibbs ensembles. *The American Mathematical Monthly*, 78(2), 142–154.

[180] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1), 1929–1958.

[181] Stewart, G. W. (1993). On the early history of the singular value decomposition. *SIAM review*, 35(4), 551–566.

[182] Storkey, A. J. & Valabregue, R. (1999). The basins of attraction of a new hopfield learning rule. *Neural Networks*, 12(6), 869–876.

[183] Su, Z., Li, Y., & Chen, K. (2022). A robust r-peaks detection algorithm of ecg signals by using adaptive combined threshold. In *Proceedings of 2021 Chinese Intelligent Systems Conference* (pp. 190–200).: Springer.

[184] Sulimowicz, L., Ahmad, I., & Aved, A. (2018). Superpixel-enhanced pairwise conditional random field for semantic segmentation. In *International Conference on Image Processing* (pp. 271–275).

[185] Sun, Z., Junttila, J., Tulppo, M., Seppänen, T., & Li, X. (2021). Non-contact atrial fibrillation detection from face videos by learning systolic peaks. *arXiv e-prints*, (pp. arXiv–2110).

[186] Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning (ICML)* (pp. 1139–1147).: PMLR.

[187] Sutton, R. S. & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[188] Szegedy, C., Ioffe, S., Vanhoucke, V., et al. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1).

[189] Tajbakhsh, N., Gurudu, S. R., & Liang, J. (2015). A Comprehensive Computer-Aided Polyp Detection System for Colonoscopy Videos. In *International Conference on Information Processing in Medical Imaging*, volume 9123 of *Lecture Notes in Computer Science* (pp. 327–338).: Springer, Cham.

[190] Tajbakhsh, N., Gurudu, S. R., & Liang, J. (2016a). Automated Polyp Detection in Colonoscopy Videos Using Shape and Context Information. *IEEE Transactions on Medical Imaging*, 35(2), 630–644.

[191] Tajbakhsh, N., Shin, J. Y., Gurudu, S. R., Hurst, R. T., Kendall, C. B., Gotway, M. B., & Liang, J. (2016b). Convolutional Neural Networks for Medical Image Analysis: Full Training or Fine Tuning? *IEEE Transactions on Medical Imaging*, 35(5), 1299–1312.

[192] Tan, M. & Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning (ICML)* (pp. 6105–6114).: PMLR.

[193] Teichmann, M. & Cipolla, R. (2019). Convolutional crfs for semantic segmentation. In *British Machine Vision Conference*.

[194] Tian, L., Li, M., Hao, Y., Liu, J., Zhang, G., & Chen, Y. Q. (2018). Robust 3-d human detection in complex environments with a depth camera. *Transactions on Multimedia*, 20(9), 2249–2261.

[195] Ulyanov, D., Vedaldi, A., & Lempitsky, V. (2016). Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*.

[196] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[197] Vázquez, D., Bernal, J., Sánchez, F. J., Fernández-Esparrach, G., López, A. M., Romero, A., Drozdzal, M., & Courville, A. (2017). A Benchmark for Endoluminal Scene Segmentation of Colonoscopy Images. *Journal of Healthcare Engineering*, 2017.

[198] Vemulapalli, R., Tuzel, O., Liu, M.-Y., & Chellapa, R. (2016). Gaussian conditional random field network for semantic segmentation. In *Computer Vision and Pattern Recognition* (pp. 3224–3233).

[199] Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., & Lang, K. J. (1989). Phoneme recognition using time-delay neural networks. *IEEE transactions on acoustics, speech, and signal processing*, 37(3), 328–339.

[200] Wang, B., An, J., & Cao, J. (2020a). Voxel-fpn: multi-scale voxel feature aggregation in 3d object detection from point clouds. *Sensors*, 20(3), 704.

[201] Wang, F. & Liu, H. (2021). Understanding the behaviour of contrastive loss. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 2495–2504).

[202] Wang, G., Ma, S., Wu, Y., Pei, J., Zhao, R., & Shi, L. (2021). End-to-end implementation of various hybrid neural networks on a cross-paradigm neuromorphic chip. *Frontiers in Neuroscience*, 15.

[203] Wang, H., Liang, Z., Li, L. C., Han, H., Song, B., Pickhardt, P. J., Barish, M. A., & Lascarides, C. E. (2015). An Adaptive Paradigm for Computer-Aided Detection of Colonic Polyps. *Physics in Medicine & Biology*, 60(18), 7207–7228.

[204] Wang, J., Sun, K., Cheng, T., et al. (2020b). Deep high-resolution representation learning for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[205] Wang, X. & cpsc2021 (2021). Python example code for the 4th china physiological signal challenge 2021. https://github.com/CPSC-Committee/cpsc2021-python-entry. Accessed 4/24/2022.

[206] Wang, X., Girshick, R., Gupta, A., et al. (2018). Non-local neural networks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 7794–7803).: IEEE/CVF.

[207] Wang, Y., Chao, W.-L., Garg, D., Hariharan, B., Campbell, M., & Weinberger, K. Q. (2019). Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *IEEE Conference on Computer Vision and Pattern Recognition* (pp. 8445–8453).

[208] Wang, Z. & Jia, K. (2019). Frustum convnet: Sliding frustums to aggregate local point-wise features for amodal. In *IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 1742–1749).

[209] Woo, S., Park, J., Lee, J.-Y., & Kweon, I. S. (2018). Cbam: Convolutional block attention module. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 3–19).

[210] Wright, L. G., Onodera, T., Stein, M. M., Wang, T., Schachter, D. T., Hu, Z., & McMahon, P. L. (2022). Deep physical neural networks trained with backpropagation. *Nature*, 601(7894), 549–555.

[211] Wu, Y. & He, K. (2018). Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 3–19).

[212] Xu, D., Anguelov, D., & Jain, A. (2018). Pointfusion: Deep sensor fusion for 3d bounding box estimation. In *IEEE Conference on Computer Vision and Pattern Recognition* (pp. 244–253).

[213] Xu, Z., Zhang, W., Ye, X., Tan, X., Yang, W., Wen, S., Ding, E., Meng, A., & Huang, L. (2020). Zoomnet: Part-aware adaptive zooming neural network for 3d object detection. In *AAAI* (pp. 12557–12564).

[214] Yamaguchi, K., Sakamoto, K., Akabane, T., & Fujimoto, Y. (1990). A neural network for speaker-independent isolated word recognition. In *ICSLP*.

[215] Yang, B., Luo, W., & Urtasun, R. (2018a). Pixor: Real-time 3d object detection from point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition* (pp. 7652–7660).

[216] Yang, J., Wang, Y., Wang, G., & Li, M. (2017). Salient Object Detection Based on Global Multi-Scale Superpixel Contrast. *IET Computer Vision*, 11(8), 710–716.

[217] Yang, M. Y. & Förstner, W. (2011). A hierarchical conditional random field model for labeling and classifying images of man-made scenes. In *International Conference on Computer Vision Workshops* (pp. 196–203).

[218] Yang, Y., Stein, A., Tolpekin, V. A., & Zhang, Y. (2018b). High-resolution remote sensing image classification using associative hierarchical crf considering segmentation quality. *Geoscience and Remote Sensing Letters*, 15(5), 754–758.

[219] Yang, Z., Sun, Y., Liu, S., Shen, X., & Jia, J. (2018c). Ipod: Intensive point-based object detector for point cloud. *arXiv preprint arXiv:1812.05276*.

[220] Yang, Z., Sun, Y., Liu, S., Shen, X., & Jia, J. (2019). Std: Sparse-to-dense 3d object detector for point cloud. In *IEEE International Conference on Computer Vision* (pp. 1951–1960).

[221] You, Y., Wang, Y., Chao, W.-L., Garg, D., Pleiss, G., Hariharan, B., Campbell, M., & Weinberger, K. Q. (2019). Pseudo-lidar++: Accurate depth for 3d object detection in autonomous driving. In *International Conference on Learning Representations*.

[222] Yu, C., Wang, J., Peng, C., et al. (2018). Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 325–341).

[223] Yu, L., Chen, H., Dou, Q., Qin, J., & Heng, P. A. (2017). Integrating Online and Offline Three-Dimensional Deep Learning for Automated Polyp Detection in Colonoscopy Videos. *IEEE Journal of Biomedical and Health Informatics*, 21(1), 65–75.

[224] Yuan, Y., Huang, L., Guo, J., et al. (2021). Ocnet: Object context for semantic segmentation. *International Journal of Computer Vision*, (pp. 1–24).

[225] Zadeh, L. & Desoer, C. (2008). *Linear system theory: the state space approach*. Courier Dover Publications.

[226] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

[227] Zeiler, M. D. & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European Conference on Computer Vision* (pp. 818–833).: Springer.

[228] Zeiler, M. D., Taylor, G. W., & Fergus, R. (2011). Adaptive deconvolutional networks for mid and high level feature learning. In *Proceedings of International Conference on Computer Vision (ICCV)* (pp. 2018–2025).: IEEE.

[229] Zeng, A. (2018). Datasets. https://vision.princeton.edu/projects/2017/arc/. Accessed 5/4/2021.

[230] Zeng, A., Song, S., Yu, K.-T., et al. (2018). Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*: IEEE.

[231] Zhang, G., Liu, J., Li, H., Chen, Y. Q., & Davis, L. S. (2017). Joint human detection and head pose estimation via multistream networks for rgb-d videos. *Signal Processing Letters*, 24(11), 1666–1670.

[232] Zhang, J., Li, C., Kulwa, F., Zhao, X., Sun, C., Li, Z., Jiang, T., Li, H., & Qi, S. (2020). A multiscale cnn-crf framework for environmental microorganism image segmentation. *BioMed Research International*, 2020.

[233] Zhao, H., Shi, J., Qi, X., et al. (2017). Pyramid scene parsing network. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 2881–2890).: IEEE/CVF.

[234] Zhen, M., Li, S., Zhou, L., Shang, J., Feng, H., Fang, T., & Quan, L. (2020). Learning discriminative feature with crf for unsupervised video object segmentation. In *European Conference on Computer Vision* (pp. 445–462).

[235] Zheng, S., Jayasumana, S., Romera-Paredes, B., Vineet, V., Su, Z., Du, D., Huang, C., & Torr, P. H. (2015). Conditional random fields as recurrent neural networks. In *International Conference on Computer Vision* (pp. 1529–1537).

[236] Zhou, B., Khosla, A., Lapedriza, A., et al. (2016). Learning deep features for discriminative localization. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 2921–2929).: IEEE/CVF.

[237] Zhou, Y. & Tuzel, O. (2018). Voxelnet: End-to-end learning for point cloud based 3d object detection. In *IEEE Conference on Computer Vision and Pattern Recognition* (pp. 4490–4499).

[238] Zhuang, J., Yang, J., Gu, L., & Dvornek, N. (2019). Shelfnet for fast semantic segmentation. In *Proceedings of the International Conference on Computer Vision Workshops (ICCVW)* (pp. 0–0).: IEEE/CVF.

Appendix A

Source Codes for BRTransformer

The evaluation and baseline repository is forked from (205). Coding platform: Pytorch 1.8.2, Windows10, Python 3.8.12, C-compiler: MinGW-gcc. Required libs: tqdm, numpy, wfdb 3.4.0, scipy 1.7.1.

The following license must be included for any purpose of implementing source codes.

Listing A.1: **main.c**

```
/*************************************************************************
 * Description:     Local-Global Filtering Peak Detector (LGFPD)
 * Author:          Xi Mo
 * Institute:       University of Kansas
 * License:         MIT License
 * Date:            10/6/2021
```

```
 * HowTo:
 *                  1) Prepare 1D signal dataset as "*.txt"
 *                  2) Set file path and/or parameters and compile, or
 *                     using arguments:
 *                     <path/to/data>:       default - signal.txt;
 *                                           maxlength per line <= 50 chars;
 *                                           ints, floats and seperators '\t'
 *                                           ' ', '\r' and ',' only;
 *                     <GlobalWindowSize>:   default - 400
 *                     <AmplititudeLevel>:   default - 10
 *                     <LocalWindowSize]>:   default - 92
 *                     <HeightThreshold>:    default - 1.9
*******************************************************************************/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <windows.h>
#include "headers/stack.h"
#include "headers/utils.h"


int main(int argc, char** argv) {
    getParameters(argc, argv);
    loadDataset(cfg.dataPath);
    // show_all_pikes(find_all_spikes(dataset));
    // find_combine_spikes(dataset);
    // show_combine_pikes();
    // free(dstack);
    find_filter_spikes(dataset);
    show_filter_pikes();
    free(dstack);
    // system("pause");
    return 0;
}
```

Listing A.2: **headers/stack.c**

```
/*****************************************************************
 * Description:     dynamic stack implementation
 * Modified:        Xi Mo
 * Date:            10/6/2021
 * Reference:
 * https://blog.csdn.net/coding1994/article/details/52745773
*****************************************************************/

#include "stack.h"
#include <stdio.h>
#include <malloc.h>
#include <assert.h>

int top_element_dstack = -1;
int dstack_size;
int top_element_cstack = -1;
int cstack_size;
```

```
18
19
20  /* create double stack */
21  void create_dstack(size_t size) {
22      dstack_size = size;
23      top_element_dstack = -1;
24      dstack = (double**)malloc(sizeof(double*) * dstack_size);
25      for (int i = 0; i < dstack_size; i++) {
26          dstack[i] = (double*)malloc(sizeof(double) * 2);
27      }
28      if(dstack == NULL) {
29          printf("ERROR: stack malloc failed\n");
30          system("pause");
31          exit(3);
32      }
33  }
34
35
36  /* create char stack */
37  void create_cstack(size_t size) {
38      cstack_size = size;
39      top_element_cstack = -1;
40      cstack = (char*)malloc(cstack_size * sizeof(char));
41      if(cstack == NULL) {
42          printf("ERROR: stack malloc failed\n");
43          system("pause");
44          exit(3);
45      }
46  }
47
48
49  /* destroy double stack */
50  void destroy_dstack(void) {
51      assert(dstack_size > 0);
52      for (int i = 0; i < dstack_size; i++) {
53          free(dstack[i]);
54      }
55      free(dstack);
56      dstack_size = 0;
57      dstack = NULL;
58  }
59
60
61  /* destroy char stack */
62  void destroy_cstack(void) {
63      assert(cstack_size > 0);
64      cstack_size = 0;
65      free(cstack);
66      cstack = NULL;
67  }
68
69
70  /* push double */
71  void push_dstack(double lhs, double rhs) {
72      assert(!is_dstack_full());
73      top_element_dstack += 1;
```

```
74      dstack[top_element_dstack][0] = lhs;
75      dstack[top_element_dstack][1] = rhs;
76  }
77
78
79  /* push char */
80  void push_cstack(char value) {
81      assert(!is_cstack_full());
82      top_element_cstack += 1;
83      cstack[top_element_cstack] = value;
84  }
85
86
87  /* pop double */
88  void pop_dstack(void) {
89      assert(!is_dstack_empty());
90      top_element_dstack -= 1;
91  }
92
93
94  /* pop char */
95  void pop_cstack(void) {
96      assert(!is_cstack_empty());
97      top_element_cstack -= 1;
98  }
99
100
101 /* purge double */
102 void purge_dstack() {
103     if (!is_dstack_empty()) {
104         top_element_dstack == -1;
105     }
106 }
107
108
109 /* purge char */
110 void purge_cstack() {
111     if (!is_cstack_empty()) {
112         top_element_cstack = -1;
113     }
114 }
115
116
117 /* top double */
118 void top_dstack(void) {
119     assert(!is_dstack_empty());
120     point.pos = dstack[top_element_dstack][0];
121     point.height = dstack[top_element_dstack][1];
122 }
123
124
125 /* top char */
126 char top_cstack(void) {
127     assert(!is_cstack_empty());
128     return cstack[top_element_cstack];
129 }
```

```
130
131
132 /* is dstack empty */
133 int is_dstack_empty(void) {
134     assert(dstack_size > 0);
135     return top_element_dstack == -1;
136 }
137
138
139 /* is cstack empty */
140 int is_cstack_empty(void) {
141     assert(cstack_size > 0);
142     return top_element_cstack == -1;
143 }
144
145
146 /* is dstack full */
147 int is_dstack_full(void) {
148     assert(dstack_size > 0);
149     return top_element_dstack > dstack_size - 1;
150 }
151
152
153 /* is cstack full */
154 int is_cstack_full(void) {
155     assert(cstack_size > 0);
156     return top_element_cstack > cstack_size - 1;
157 }
```

Listing A.3: **headers/stack.h**

```
1  /*****************************************************************
2   * Description:      dynamic stack implementation
3   * Modified:         Xi Mo
4   * Date:             10/6/2021
5   * Reference:
6   * https://blog.csdn.net/coding1994/article/details/52745773
7  *****************************************************************/
8
9  #ifndef _STACK_H_
10 #define _STACK_H_
11
12 #include<stdlib.h>
13
14 // global variable
15 double**    dstack;
16 char*       cstack;
17 static int  peakNum;
18 double      runtime;
19
20 extern int top_element_dstack;
21 extern int dstack_size;
22 extern int top_element_cstack;
23 extern int cstack_size;
24
```

```
struct POINT {
    double pos;
    double height;
} point;

extern void create_dstack(size_t);
extern void destroy_dstack(void);
extern void push_dstack(double, double);
extern void pop_dstack(void);
extern void top_dstack(void);
extern void purge_dstack(void);
extern int is_dstack_empty(void);
extern int is_dstack_full(void);

extern void create_cstack(size_t);
extern void destroy_cstack(void);
extern void push_cstack(char);
extern void pop_cstack(void);
extern char top_cstack(void);
extern void purge_cstack(void);
extern int is_cstack_empty(void);
extern int is_cstack_full(void);

#endif
```

Listing A.4: **headers/utils.c**

```
/*****************************************************************
 * Description:      Tool functions
 * Author:           Xi Mo
 * Institute:        University of Kansas
 * License:          MIT License
 * Date:             10/6/2021
*****************************************************************/

#include "utils.h"
#include "stack.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <limits.h>
#include <sys/time.h>


/* acquire configuration parameters */
void getParameters(int argc, char** argv) {
    switch (argc) {
        case 2:
            strcpy(cfg.dataPath, argv[1]);
            cfg.winSize = 400;
            cfg.ampLv = 10;
            cfg.combine = 92;
            cfg.thres = 1.9;
            break;
```

```
        case 3:
            strcpy(cfg.dataPath, argv[1]);
            cfg.winSize = atoi(argv[2]);
            cfg.ampLv = 10;
            cfg.combine = 92;
            cfg.thres = 1.9;
            break;
        case 4:
            strcpy(cfg.dataPath, argv[1]);
            cfg.winSize = atoi(argv[2]);
            cfg.ampLv = atoi(argv[3]);
            cfg.combine = 92;
            cfg.thres = 1.9;
            break;
        case 5:
            strcpy(cfg.dataPath, argv[1]);
            cfg.winSize = atoi(argv[2]);
            cfg.ampLv = atoi(argv[3]);
            cfg.combine = atoi(argv[4]);
            cfg.thres = 1.9;
            break;
        case 6:
            strcpy(cfg.dataPath, argv[1]);
            cfg.winSize = atoi(argv[2]);
            cfg.ampLv = atoi(argv[3]);
            cfg.combine = atoi(argv[4]);
            cfg.thres = atof(argv[5]);
            break;
        default:
            strcpy(cfg.dataPath, "signal.txt");
            cfg.winSize = 400;
            cfg.ampLv = 10;
            cfg.combine = 92;
            cfg.thres = 1.9;
    }
}


/* read dimensions from one sample */
int getDimension(char* buffer) {
    int dims = 0, idx = 0;
    bool isNum = false;
    while (buffer[idx] != '\0') { // get data dimension
        if (buffer[idx] == '\t' || buffer[idx] == ' ' ||
            buffer[idx] == ','  || buffer[idx] == '\r' ||
            buffer[idx] == '\n') {
            if (dims || !dims && isNum) { // head
                if (buffer[idx] == '\n' && isNum) { // tali \n
                    dims++;
                }
                else if (buffer[idx] != '\n' && isNum) { // interval
                    isNum = false;
                    dims++;
                }
            }
        }
```

```
85          else {
86              isNum = true;
87              if (buffer[idx+1] == '\0') { // tail EOF
88                  dims++;
89              }
90          }
91          idx++;
92      }
93      return dims;
94  }
95
96
97  /* dataloader */
98  void loadDataset(char* root) {
99      char buffer[51];
100     int rowNum = 0;
101     int preDim = 0, colNum = 0;
102     int idx = 0, dim = 0;
103     bool isFirst = true, isNum = false;
104     FILE* fp = fopen(root, "r");
105
106     if (fp == NULL) {
107         printf("ERROR: incorrect dataset path or file format\n");
108         system("pause");
109         exit(1);
110     }
111     // preload and check dataset
112     printf("Checking dataset ...\n");
113     while ((fgets(buffer, 51, fp)) != NULL) { // get file length
114         if (strlen(buffer) != 1) {
115             rowNum++;
116             colNum = getDimension(buffer);
117             if (isFirst) {
118                 preDim = colNum;
119                 isFirst = false;
120             }
121             if (colNum != preDim) {
122                 printf("ERROR: unmatched dimensions detected,"\
123                        "please check redundant seperators or values\n");
124                 system("pause");
125                 exit(2);
126             }
127         }
128     }
129     // load dataset
130
131     rewind(fp);
132     double **data = (double**)malloc(sizeof(double*) * rowNum);
133     for (int i = 0; i < rowNum; i++) { // create 2D array
134         data[i] = (double*)malloc(sizeof(double) * colNum);
135     }
136     create_cstack(51);
137     while ((fgets(buffer, 51, fp)) != NULL) {
138         if (strlen(buffer) != 1) {
139             while (buffer[idx] != '\0') {
140                 if (buffer[idx] == '\t' || buffer[idx] == ' ' ||
```

```
                    buffer[idx] == '\r' || buffer[idx] == ',' ||
                    buffer[idx] == '\n') {
                    if (dim || !dim && isNum) { //head
                        push_cstack('\0');
                        data[(int)(dim/colNum)][dim%colNum] = atof(cstack);
                        purge_cstack();
                        dim++;
                    }
                    else if (buffer[idx] != '\n' && isNum) {
                        isNum = false;
                        push_cstack('\0');
                        data[(int)(dim/colNum)][dim%colNum] = atof(cstack);
                        purge_cstack();
                        dim++;
                    }
                }
                else {
                    isNum = true;
                    if (buffer[idx+1] == '\0') { // tail EOF
                        push_cstack(buffer[idx]);
                        push_cstack('\0');
                        data[(int)(dim/colNum)][dim%colNum] = atof(cstack);
                        destroy_cstack();
                    }
                    else {
                        push_cstack(buffer[idx]);
                    }
                }
                idx++;
            }
            idx = 0;
        }
    }

    fclose(fp);
    dataset.value = data;
    dataset.row = rowNum;
    dataset.column = colNum;
}


/* LGFPD-B algorithm */
double** find_all_spikes(struct DATASET data) {
    // time stats
    struct timeval startTime, endTime;
    gettimeofday(&startTime, NULL);
    // detection results
    double **spikes = (double**)malloc( // create 2D array
                      sizeof(double*) * (int)(data.row/2));
    for (int i = 0; i < (int)(data.row/2); i++) {
        spikes[i] = (double*)malloc(sizeof(double) * 2);
    }
    // local variables
    int scales = (int)(data.row /cfg.winSize);
    int incr, delta, sIdx, bound, mIdx;
    double min, max, reslution, base, sum, amp;
```

```
    double x0, y0, x1 = 0, y1 = 0;
    double save[2];
    double ***level = (double***)malloc( // create 3D array
                      sizeof(double**) * cfg.ampLv);
    for (int i = 0; i < cfg.ampLv; i++) {
        level[i] = (double**)malloc(sizeof(double*) * cfg.winSize);
        for (int j = 0; j < cfg.winSize; j++) {
            level[i][j] = (double*)malloc(sizeof(double) * 3);
        }
    }
    int* lvIdx = (int*)malloc(sizeof(int) * cfg.ampLv);
    for (int i = 0; i < cfg.ampLv; i++) {
        lvIdx[i] = 0;
    }
    // find peaks
    for (int i = 0; i < scales; i++) {
        incr = i? -1: 0;
        delta = i * cfg.winSize;
        // prepare for amplititude levels
        min = dataset.value[delta][1];
        max = dataset.value[delta][1];
        if (i == scales - 1) { // set bound
            bound = !(data.row % cfg.winSize) ?
                      cfg.winSize : data.row % cfg.winSize;
        }
        else {
            bound = cfg.winSize;
        }
        for (int s = delta + 1; s < delta + bound; s++) {
            if (dataset.value[s][1] < min) {
                min = dataset.value[s][1];
            }
            if (dataset.value[s][1] > max) {
                max = dataset.value[s][1];
            }
        }
        reslution = (max - min) / cfg.ampLv;
        // get amplititude and coordinates
        if (!i) {
            x0 = dataset.value[incr + delta][0];
            y0 = dataset.value[incr + delta][1];
        }
        else {
            x0 = x1; y0 = y1;
        }
        while (incr < bound - 1) {
            incr++;
            x1 = dataset.value[incr + delta][0];
            y1 = dataset.value[incr + delta][1];
            if (y1 < y0) {
                save[0] = x0; save[1] = y0;
                while (incr < bound - 1 && y1 <= y0) {
                    incr++;
                    x0 = x1; y0 = y1;
                    x1 = dataset.value[incr + delta][0];
                    y1 = dataset.value[incr + delta][1];
```

```
253                 }
254                 if (incr < bound) {
255                     sIdx = (int)((save[1] - min) / reslution);
256                     if (sIdx >= cfg.ampLv) {
257                         sIdx = cfg.ampLv - 1;
258                     }
259                     level[sIdx][lvIdx[sIdx]][0] = save[0];
260                     level[sIdx][lvIdx[sIdx]][1] = save[1];
261                     level[sIdx][lvIdx[sIdx]][2] = (save[1] + y0) / 2;
262                     lvIdx[sIdx] += 1;
263                     x0 = x1; y0 = y1;
264                 }
265             }
266             else if (y1 > y0) {
267                 while (incr < bound - 1 && y1 >= y0) {
268                     incr++;
269                     x0 = x1; y0 = y1;
270                     x1 = dataset.value[incr + delta][0];
271                     y1 = dataset.value[incr + delta][1];
272                 }
273             }
274             else {
275                 x0 = x1; y0 = y1;
276             }
277         }
278         // get baseline
279         mIdx = 0;
280         for (int s = 1, tmp = lvIdx[0]; s < cfg.ampLv; s++) { // max index
281             if (lvIdx[s] > tmp) {
282                 tmp = lvIdx[s];
283                 mIdx = s;
284             }
285         }
286         sum = 0.0;
287         for (int s = 0; s < lvIdx[mIdx]; s++) {
288             sum += level[mIdx][s][2];
289         }
290         base = sum / lvIdx[mIdx];
291         // thresholding
292         for (int s = 0; s < cfg.ampLv; s++) {
293             if (s <= mIdx) {
294                 continue;
295             }
296             for (int j = 0; j < lvIdx[s]; j++) {
297                 amp = level[s][j][1] - base;
298                 if (amp >= cfg.thres) {
299                     spikes[peakNum][0] = level[s][j][0];
300                     spikes[peakNum][1] = amp;
301                     peakNum++;
302                 }
303             }
304         }
305         for (int s = 0; s < cfg.ampLv; s++) {
306             lvIdx[s] = 0;
307         }
308     }
```

```
    // sort spikes
    quick_sort(spikes, 0, peakNum - 1);
    // release memory
    for (int i = 0; i < cfg.ampLv; i++) {
        for (int j = 0; j < cfg.winSize; j++) {
            free(level[i][j]);
        }
        free(level[i]);
    }
    free(level);
    free(lvIdx);
    release_dataset_memory();
    gettimeofday(&endTime, NULL);
    runtime = (endTime.tv_sec - startTime.tv_sec) * 1000 +
              (endTime.tv_usec - startTime.tv_usec) * 0.001;
    return spikes;
}


/* LGFPD-C algorithm */
void find_combine_spikes(struct DATASET data) {
    // time stats
    struct timeval startTime, endTime;
    gettimeofday(&startTime, NULL);
    // detection results
    double** spikes = find_all_spikes(data);
    create_dstack(peakNum);
    // local variables
    double idx;
    if (!peakNum) {
        printf("ERROR: cannot find any peaks\n");
        system("pause");
        exit(4);
    }
    for (int i = 0; i < peakNum; i++) {
        if (i == 0) {
            push_dstack(spikes[0][0], spikes[0][1]);
            idx = spikes[0][0];
        }
        else {
            if (spikes[i][0] - idx <= cfg.combine &&
                spikes[i][1] > dstack[top_element_dstack][1]) {
                    pop_dstack();
                    push_dstack(spikes[i][0], spikes[i][1]);
                    idx = spikes[i][0];
            }
            else if (spikes[i][0] - idx > cfg.combine) {
                push_dstack(spikes[i][0], spikes[i][1]);
                idx = spikes[i][0];
            }
        }
    }
    // free space
    for (int i = 0; i < (int)(data.row/2); i++) {
        free(spikes[i]);
    }
```

```
    free(spikes);
    gettimeofday(&endTime, NULL);
    runtime = (endTime.tv_sec - startTime.tv_sec) * 1000 +
              (endTime.tv_usec - startTime.tv_usec) * 0.001;
}


/* LGFPD-F algorithm */
void find_filter_spikes(struct DATASET data) {
    // time stats
    struct timeval startTime, endTime;
    gettimeofday(&startTime, NULL);
    // detection results
    double sentinal = (double)INT_MIN;
    double** temp = find_all_spikes(data);
    if (!peakNum) {
        printf("ERROR: cannot find any peaks\n");
        system("pause");
        exit(4);
    }
    double** spikes = (double**)malloc(sizeof(double*) * (peakNum + 1));
    for (int i = 0; i < peakNum + 1; i++) {
        spikes[i] = (double*)malloc(sizeof(double) * 2);
         if (i != peakNum) {
            for (int j = 0; j < 2; j++) {
                spikes[i][j] = temp[i][j];
            }
        }
        else {
            spikes[i][0] = (double)INT_MAX;
            spikes[i][1] = sentinal;
        }
    }
    for (int i = 0; i < (int)(data.row/2); i++) {
        free(temp[i]);
    }
    free(temp);
    create_dstack(peakNum + 1);
    // local variables
    int flag, cLen = 0;
    double idx, mElem, lhs, rhs;
    double** collect = (double**)malloc(sizeof(double*) * (peakNum + 1));
    for (int i = 0; i < peakNum + 1; i++) {
        collect[i] = (double*)malloc(sizeof(double) * 2);
    }
    for (int i = 0; i < peakNum + 1; i++) {
        if (cLen == 0) {
            collect[cLen][0] = spikes[i][0];
            collect[cLen][1] = spikes[i][1];
            idx = spikes[i][0];
            cLen++;
        }
        else {
            if (spikes[i][0] - idx <= cfg.combine) {
                collect[cLen][0] = spikes[i][0];
                collect[cLen][1] = spikes[i][1];
```

```
                idx = spikes[i][0];
                cLen++;
            }
            else { // empty collection
                mElem = collect[0][1];
                for (int s = 1; s < cLen; s++) {
                    if (collect[s][1] > mElem) {
                        mElem = collect[s][1];
                    }
                }
                for (int s = 0; s < cLen; s++) {
                    if (cLen == 1) {
                        push_dstack(collect[s][0], collect[s][1]);
                        idx = collect[s][0];
                        break;
                    }
                    else if (!s && s < cLen - 1) { // border conditions
                        lhs = sentinal;
                        rhs = collect[s + 1][1];
                        flag = 1;
                    }
                    else if (s == cLen - 1 && cLen > 1) {
                        lhs = collect[s - 1][1];
                        rhs = sentinal;
                        flag = 2;
                    }
                    else {
                        lhs = collect[s - 1][1];
                        rhs = collect[s + 1][1];
                        flag = 3;
                    }
                    if (collect[s][1] == mElem &&
                       (flag == 1 || flag == 2)) {
                           push_dstack(collect[s][0], collect[s][1]);
                    }
                    else {
                        if (lhs < collect[s][1] && rhs < collect[s][1]) {
                            push_dstack(collect[s][0], collect[s][1]);
                        }
                    }
                }
                cLen = 1;
                collect[0][0] = spikes[i][0];
                collect[0][1] = spikes[i][1];
                idx = spikes[i][0];
            }
        }
    }
    // free space
    for (int i = 0; i < peakNum + 1; i++) {
        free(spikes[i]);
        free(collect[i]);
    }
    free(spikes);
    free(collect);
    gettimeofday(&endTime, NULL);
```

```
    runtime = (endTime.tv_sec - startTime.tv_sec) * 1000 +
              (endTime.tv_usec - startTime.tv_usec) * 0.001;
}


/* swap for quick sort */
void swap(double* lhs, double* rhs) {
    double tmp;
    tmp = *lhs;
    *lhs = *rhs;
    *rhs = tmp;
}


/* quick sort */
void quick_sort(double** peaks, int low, int high) {
    int left = low, right = high;
    double pos = peaks[left][0], height = peaks[left][1];

    if (low >= high) {
        return;
    }
    while (left < right) {
        // search from right to left
        while (left < right && peaks[right][0] >= pos) {
            right--;
        }
        swap(&peaks[left][0], &peaks[right][0]);
        swap(&peaks[left][1], &peaks[right][1]);
        // search from left to right
        while (left < right && peaks[left][0] <= pos) {
            left++;
        }
        swap(&peaks[right][0], &peaks[left][0]);
        swap(&peaks[right][1], &peaks[left][1]);
    }
    // pivot location when equals
    peaks[left][0] = pos;
    peaks[left][1] = height;
    // recursion
    quick_sort(peaks, low, left - 1);
    quick_sort(peaks, left + 1, high);
}


/* print LGFPD-B results */
void show_all_pikes(double** peaks) {
    FILE *fp = fopen("LGFPD-B.txt", "w+");
    fprintf(fp, "\n=======================================\n"\
                "     \tParameters:"\
                "\n=======================================\n\n");
    fprintf(fp, "dataset:\t\t%s\n", cfg.dataPath);
    fprintf(fp, "valid data length:\t%d\n", dataset.row);
    fprintf(fp, "global windowSize:\t%d\n", cfg.winSize);
    fprintf(fp, "bin number:\t\t%d\n", cfg.ampLv);
    fprintf(fp, "local windowSize:\t%d\n", cfg.combine);
```

```
    fprintf(fp, "height threshold:\t%.3f\n", cfg.thres);
    fprintf(fp, "runtime:\t\t%.3fms\n", runtime);
    fprintf(fp, "\n=======================================\n"\
                "    %d Peaks detected by LGFPD-B:"\
                "\n=======================================\n\n",
                peakNum);
    printf("\nPeaks detected by LGFPD-B:\n\n");
    for (int i = 0; i < peakNum; i++) {
        printf("[ %-6d %6.2f ]\n", (int)peaks[i][0], peaks[i][1]);
        fprintf(fp, "[ %-6d %6.2f ]\n", (int)peaks[i][0], peaks[i][1]);
    }
    fprintf(fp, "\n---------------------------------------\n");
    printf("\nTotal peaks detected: %d\n", peakNum);
    printf("Time elapse: %.3fms\n", runtime);
    printf("Full report has been written to: LGFPD-B.txt\n\n");
}


/* print LGFPD-C results */
void show_combine_pikes(void) {
    FILE *fp = fopen("LGFPD-C.txt", "w+");
    fprintf(fp, "\n=======================================\n"\
                "     \tParameters:"\
                "\n=======================================\n\n");
    fprintf(fp, "dataset:\t\t%s\n", cfg.dataPath);
    fprintf(fp, "valid data length:\t%d\n", dataset.row);
    fprintf(fp, "global windowSize:\t%d\n", cfg.winSize);
    fprintf(fp, "bin number:\t\t%d\n", cfg.ampLv);
    fprintf(fp, "local windowSize:\t%d\n", cfg.combine);
    fprintf(fp, "height threshold:\t%.3f\n", cfg.thres);
    fprintf(fp, "runtime:\t\t%.3fms\n", runtime);
    fprintf(fp, "\n=======================================\n"\
                "    %d Peaks detected by LGFPD-C:"\
                "\n=======================================\n\n",
                top_element_dstack);
    printf("\nPeaks detected by LGFPD-C:\n\n");
    for (int i = 0; i < top_element_dstack; i++) {
        printf("[ %-6d %6.2f ]\n", (int)dstack[i][0], dstack[i][1]);
        fprintf(fp, "[ %-6d %6.2f ]\n", (int)dstack[i][0], dstack[i][1]);
    }
    fprintf(fp, "\n---------------------------------------\n");
    printf("\nTotal peaks detected: %d\n", top_element_dstack);
    printf("Time elapse: %.3fms\n", runtime);
    printf("Full report has been written to: LGFPD-C.txt\n\n");
}


/* print LGFPD-F results */
void show_filter_pikes(void) {
    FILE *fp = fopen("LGFPD-F.txt", "w+");
    fprintf(fp, "\n=======================================\n"\
                "     \tParameters:"\
                "\n=======================================\n\n");
    fprintf(fp, "dataset:\t\t%s\n", cfg.dataPath);
    fprintf(fp, "valid data length:\t%d\n", dataset.row);
    fprintf(fp, "global windowSize:\t%d\n", cfg.winSize);
```

```
    fprintf(fp, "bin number:\t\t%d\n", cfg.ampLv);
    fprintf(fp, "local windowSize:\t%d\n", cfg.combine);
    fprintf(fp, "height threshold:\t%.3f\n", cfg.thres);
    fprintf(fp, "runtime:\t\t%.3fms\n", runtime);
    fprintf(fp, "\n=======================================\n"\
                "    %d Peaks detected by LGFPD-F:"\
                "\n=======================================\n\n",
                top_element_dstack);
    printf("\nPeaks detected by LGFPD-F:\n\n");
    for (int i = 0; i < top_element_dstack; i++) {
        printf("[ %-6d %6.2f ]\n", (int)dstack[i][0], dstack[i][1]);
        fprintf(fp, "[ %-6d %6.2f ]\n", (int)dstack[i][0], dstack[i][1]);
    }
    fprintf(fp, "\n---------------------------------------\n");
    printf("\nTotal peaks detected: %d\n", top_element_dstack);
    printf("Time elapse: %.3fms\n", runtime);
    printf("Full report has been written to: LGFPD-F.txt\n\n");
}


/* release dataset mem */
void release_dataset_memory() {
    for (int i = 0; i < dataset.row; i++) {
        free(dataset.value[i]);
    }
    free(dataset.value);
}
```

Listing A.5: **headers/utils.h**

```
/*****************************************************************
 * Description:     header for Tool functions
 * Author:          Xi Mo
 * Institute:       University of Kansas
 * License:         MIT License
 * Date:            10/6/2021
*****************************************************************/

#ifndef _UTILS_H_
#define _UTILS_H_

#include <stdlib.h>

// configurations
struct CONFIG {
    char dataPath[200];
    size_t winSize;
    size_t ampLv;
    size_t combine;
    double thres;
} cfg;

// dataset
struct DATASET {
    double** value;
```

```
    int row;
    int column;
} dataset;

extern void getParameters(int, char**);
extern void loadDataset(char*);
extern double** find_all_spikes(struct DATASET);
extern void find_combine_spikes(struct DATASET);
extern void find_filter_spikes(struct DATASET);
extern void show_all_pikes(double**);
extern void quick_sort(double**, int, int);
extern void show_combine_pikes(void);
extern void show_filter_pikes(void);
extern void release_dataset_memory(void);

#endif
```

Listing A.6: **main.py**

```
import copy
import time
import torch.nn as nn
import torch
import math
import os
from tqdm import tqdm
from utils.arch import Transformer
from utils.dataloader import ECG, analyse_validation_results
from utils.configuration import cfg


def train(model, device):
    model.train()  # turn on train mode
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(model.parameters(), lr=cfg['lr'])
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, cfg['lr'], gamma=0.95)
    fileList =  sorted(cfg['ckptPath'].glob("*.pt"), reverse=True, key=lambda item: item.stat().st_ctime)
    if  len(fileList) and not cfg['scratch']:
        ckptPath = fileList[0]
        checkpoint = torch.load(ckptPath, map_location=device)
        model.load_state_dict(checkpoint['model_state_dict'])
    else:
        trainData = ECG([cfg['trainset1']], "train", cfg['limit'], cfg['categories'], cfg['lb'], cfg['ub'], cfg['seqLen'], cfg['iters'])
        trainSet = torch.utils.data.DataLoader(trainData, collate_fn=trainData.collate_fn_train)
        print(f"\nTrain {cfg['epoch']} epoches:")
        for i in  range(cfg['epoch']):
            _iter_, tLoss = 0, 0.
            with tqdm(total = cfg['iters'], desc=f"Epoch {i+1: 3d}", unit="iters") as pbar:
                for batch in trainSet:
                    _iter_ += 1
                    data, label = batch
                    # label = torch.ones(label.shape)
                    output = model(data, label, device)
                    loss = criterion(output.permute(0, 2, 1), label.to(device))
                    optimizer.zero_grad()
```

```
                    loss.backward()
                    tLoss += loss.item()
                    aLoss = tLoss / _iter_
                    # torch.nn.utils.clip_grad_norm_(model.parameters(), 0.5)
                    optimizer.step()
                    lr = scheduler.get_last_lr()[0]
                    ppl = math.exp(aLoss)
                    pbar.set_postfix({'loss_mean': aLoss, 'ppl': ppl, 'lr': lr, 'device': device}, refresh=True)
                    pbar.update(1)
                scheduler.step()
            torch.save({'model_state_dict': model.state_dict()}, cfg['ckptPath'].joinpath(f"{str(i+1)}.pt"))


def test(model, device):
    model. eval()  # turn on evaluation mode
    testData = ECG([cfg['testset']], "test", cfg['limit'], cfg['categories'], cfg['lb'], cfg['ub'], cfg['seqLen'])
    testSet = torch.utils.data.DataLoader(testData, collate_fn=testData.collate_fn_test)
    print(f"\nTest {len(testData)} samples:")
    with tqdm(total= len(testData), desc=None, unit="files") as pbar:
        for batch in testSet:
            pred = torch.zeros(cfg["seqLen"], 1, cfg['d_model'])
            name, data, length = batch
            for bIdx in  range(0,  data.size(1)):
                output = model(data[:, bIdx:bIdx+1], data[:, bIdx:bIdx+1], device)
                pred = torch.cat((pred, output.detach().cpu()), dim=1)

            pred = pred[:, 1:, :]
            pbar.set_postfix({'filename': name, 'length': length, 'device': device}, refresh=True)
            pbar.update(1)
            pred = torch.argmax(torch.softmax(pred, 2), 2)
            analyse_validation_results([name, testData.data[name], pred])


def main():
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    nToken = cfg['limit'] + cfg['categories'] + 2
    model = Transformer(cfg['maxLen'], nToken, cfg['d_model'], cfg['n_head'], cfg['n_encoder'],
                        cfg['n_decoder'], cfg['n_ffwrd'], cfg['dropout'], cfg['activation']).to(device)
    if cfg['train']:
        train(model, device)
    else:
        test(model, device)
    if cfg['evaluation']:
        command = "python " + r"scripts/official/score_2021.py "
        os.system(command +  str(cfg['testset']) + " " + cfg['predictValPath'])


if __name__ == "__main__":
    main()
```

Listing A.7: **uils/arch.py**

```
import torch
import torch.nn as nn
import numpy as np
```

```
import math


class Transformer(nn.Module):
        def __init__(self, maxLen = 5000, token=64, d_model = 256, head=8, encoder=6,
                                    decoder=6, ffwrd=512, dropout=0.1, activation='relu'):
                 super().__init__()
                self.dropout = nn.Dropout( p=dropout)

                self.transformer = nn.Transformer(d_model, head, encoder, decoder, ffwrd, dropout, activation)
                self.seq2vec = nn.Embedding(token, d_model)
                self.d_model = d_model
                position = torch.arange(maxLen).unsqueeze(1)
                div_term = torch.exp(torch.arange(0, d_model, 2) * (-math.log(10000.0) / d_model))
                pe = torch.zeros(maxLen, 1, d_model)
                pe[:, 0, 0::2] = torch.sin(position * div_term)
                pe[:, 0, 1::2] = torch.cos(position * div_term)
                self.register_buffer('pe', pe)

        def forward(self, _input_, _target_, device):
                _input_, _target_ = _input_.to(device), _target_.to(device)
                _input_ = self.seq2vec(_input_) * math.sqrt(self.d_model)
                _target_ = self.seq2vec(_target_)
                in_mask = self.getMask(_input_.shape[0]).to(device)
                tar_mask = self.getMask(_target_.shape[0]).to(device)
                return self.transformer(self.embedding(_input_), _target_, in_mask, tar_mask)

        def embedding(self, _input_):
                return self.dropout(_input_ + self.pe[:_input_.size(0)])

        @classmethod
        def getMask(cls, size):
                return torch.triu(torch.ones(size, size) *  float('-inf'), diagonal=1)
```

Listing A.8: **uils/configuration.py**

```
from pathlib import Path


cfg = {
        # mode for BRTransformer
        'train': False,
        'evaluation': True,
        # Paths for loading/saving data
        'trainset1': Path(r"datasets/training_I").resolve(),  # can list more sets
        'trainset2': Path(r"datasets/training_II_train").resolve(),
        'testset': Path(r"datasets/validation_II").resolve(),
        'ckptPath': Path(r"checkpoint").resolve(),
        'predictValPath': r"predictions_validation",
        # for dataset
        'limit': 100,  # base num of semantic sets
        'categories': 2,  # number of categories, fixed number
        'seqLen': 200,  # trimmed sequence length
        'lb': 20,  # lower bound for determining lowest semantic set
        'ub' : 620,  # upper bound for determing highest semantic set
```

```
        # for transformer
        'd_model': 1024,  # dimension of word vector
        'maxLen': 1001,  # maximum length for positional encoding
        'n_head': 32,  # number of attention heads
        'n_encoder': 10, # number of encoders
        'n_decoder': 10, # number of decoders
        'n_ffwrd': 512,  # dimension of hiden feed-forward layer
        'dropout': 0.0,
        'activation': 'relu',
        # for training
        'scratch': True,  # Turn on to enforce training from scratch
        'iters':  int(200),  # iterations
        'epoch': 50,  # epoches
        'lr': 1,  # learning rate
        # for testing
        'testBatch': 10  # Number of files to be tested as a batch
}
```

Listing A.9: **uils/dataloader.py**

```
import numpy as np
from tqdm import tqdm
from pathlib import Path
import os
from scripts.official.entry_2021 import load_data
from scripts.official.score_2021 import RefInfo
from scripts.official.utils import save_dict
from torch.utils.data import IterableDataset
from utils.configuration import cfg
import torch


class ECG(IterableDataset):
        def __init__(self, pathList, mode = "train", lim = 60, cat = 2,
                                        lb = 20, ub = 620, seqLen = 1000, iters = 1e5):
                 super().__init__()
                self.pathList = pathList
                self.lim = lim
                self.cat = cat
                self.nToken = lim + cat + 1
                self.seqLen = seqLen
                self.iters =  int(iters)
                self.ub = ub
                self.lb = lb
                self.mode = mode
                if mode == "train":
                        self.data, self.posIdx, self.negIdx = self.read_train_set()
                else:
                        self.data = self.read_test_set()

        def __len__(self):
                return  len(self.data)

        def __getitem__(self, item):
                pass
```

```
36
37          def __iter__(self):
38                  if self.mode == "train":
39                          for _ in  range(self.iters):
40                                  Index = np.random.choice(self.posIdx, 2, True).tolist()
41                                  Index.extend(np.random.choice(self.negIdx, 2, False).tolist())
42                                  yield Index
43                  else:
44                          for k, v in self.data.items():
45                                  yield k, v
46
47          @classmethod
48          def batchify(cls, data):
49                  rawLen = data.shape[0]
50                  seqLen =  int(cfg['seqLen'])
51                  rmd = rawLen % seqLen
52                  if rmd:
53                          padLen = rawLen + seqLen - rmd
54                          data = np.pad(data, (0, seqLen - rmd), 'constant')
55                  else:
56                          padLen = rawLen
57                  T = torch.tensor(data, dtype=torch. long)
58                  T = T.view(seqLen, padLen // seqLen).contiguous()
59                  return T, rawLen
60
61          def collate_fn_train(self, idx):
62                  posIdx1, posIdx2, negIdx1, negIdx2 = idx[0]
63                  pos1, pos2 = self.pad(posIdx1), self.truncate(posIdx2)
64                  neg1, neg2 = self.truncate(negIdx1), self.truncate(negIdx2)
65                  sample = np.vstack((pos1[0], neg1[0], pos2[0], neg2[0]))
66                  label = np.vstack((pos1[1], neg1[1], pos2[1], neg2[1]))
67                  sample = torch.from_numpy(sample). long().t().contiguous()
68                  label = torch.from_numpy(label). long().t().contiguous()
69                  return sample, label
70
71          def collate_fn_test(self, idx):
72                  name, sample = idx[0]
73                  sample, length = self.batchify(sample)
74                  return name, sample, length
75
76          def truncate(self, idx, positive = False):
77                  data, label = self.data[idx]
78                  if not positive:
79                          start = np.random.randint(0,  max(label.size - self.seqLen, 1), 1)[0]
80                          delta = data.size - start
81                          padLen = self.seqLen - delta if delta < self.seqLen else 0
82                          if not padLen:
83                                  data = data[start: start + self.seqLen]
84                                  label = label[start: start + self.seqLen]
85                          else:
86                                  data = np.pad(data, (0, padLen), constant_values=0)
87                                  label = np.pad(label, (0, padLen), constant_values=cfg['limit'] + cfg['categories'])
88                  else:
89                          padLen = self.seqLen - data.size if data.size < self.seqLen else 0
90                          if not padLen:
91                                  data = data[: self.seqLen]
```

```
                        label = label[: self.seqLen]
                else:
                        data = np.pad(data, (0, padLen), constant_values = 0)
                        label = np.pad(label, (0, padLen), constant_values = cfg['limit'] + cfg['categories'])
        return [data, label]

    # for cat = 2 only
    def pad(self, idx):
        data, label = self.data[idx]
        tail = label.size - 1
        while label[tail] != cfg['limit'] + cfg['categories'] + 1 and tail > 0: tail -= 1
        if not tail: tail = 1
        tail += 1
        if tail < self.seqLen:
                padLen = self.seqLen - tail
                data = np.pad(data[: tail], (0, padLen), "reflect")
                label = np.pad(label[: tail], (0, padLen), "reflect")
        else:
                data = data[: self.seqLen]
                label = label[: self.seqLen]
        return [data, label]

    @classmethod
    def raw2fuzzy(cls, raw, lb = 20, ub= 620, lim = 60):
        raw[raw > ub] = ub
        raw[raw < lb] = lb
        slope =  float(lim / (ub - lb))
        fuzzySet = slope * (raw - lb)
        return fuzzySet.astype(np.int16)

    @classmethod
    def get_label_from_array(cls, posIntv, label):
        for item in posIntv:  # for self.cat = 2 only
                for cIdx in  range(item[0], item[1]):
                        label[cIdx] += 1
        return label

    def read_train_set(self):
        tranSet, posIdx, negIdx = [], [], []
        cumSum = 0
        for path in self.pathList:
                fileList =  open(os.path.join(path, "RECORDS"), 'r').read().splitlines()
                totalFile =  len(fileList)
                with tqdm(total = totalFile, unit = "file", desc=f"Collecting training set from {str(path)}") as pbar:
                        for idx,  file in  enumerate(fileList):
                                filePath = os.path.join(path,  file)
                                TrueRef = RefInfo(filePath)
                                beatLoc = TrueRef.beat_loc
                                data = self.raw2fuzzy(np.diff(beatLoc), self.lb, self.ub, lim = self.lim)
                                label = np.ones(data.size) * (cfg['limit'] + cfg['categories'])
                                if not TrueRef.endpoints_true.size:
                                        negIdx.append(cumSum + idx)
                                else:
                                        posIdx.append(cumSum + idx)
                                        label = self.get_label_from_array(TrueRef.endpoints_true, label)
                                tranSet.append([data, label])
```

```
                                        pbar.update(1)
                                cumSum += totalFile
                return tranSet, posIdx, negIdx

        def read_test_set(self):
                testSet = {}
                for path in self.pathList:
                        fileList =  open(os.path.join(path, "RECORDS"), 'r').read().splitlines()
                        totalFile =  len(fileList)
                        with tqdm(total = totalFile, unit = "file", desc=f"Collecting test set from {str(path)}") as pbar:
                                for  file in fileList:
                                        TrueRef = RefInfo(os.path.join(path,  file))
                                        beatLoc = TrueRef.beat_loc
                                        testSet[ file] = self.raw2fuzzy(np.diff(beatLoc), self.lb, self.ub, lim = self.lim)
                                        pbar.update(1)
                return testSet


def analyse_validation_results(patch):
        if not Path(cfg['predictValPath']).is_dir(): Path(cfg['predictValPath']).mkdir()
        name, raw, pred = patch
        pred = pred.view(-1).contiguous()
        idx, start, end, results = 0, None, None, []
        while idx <  len(raw):
                if idx ==  len(raw)-1 and start is not None:
                        results.extend([[start, idx]])
                if pred[idx] >= cfg['limit'] + cfg['categories'] + 1 and end is None:
                        start, end = idx, False
                elif pred[idx] < cfg['limit'] + cfg['categories'] + 1 and start is not None:
                        results.extend([[start, idx]])
                        start, end = None, None
                idx += 1
        results = {'predict_endpoints': results}
        save_dict(os.path.join(cfg['predictValPath'], name + '.json'), results)
        # results = {'predict_endpoints': results}
        # save_dict(os.path.join(cfg['predictValPath'], name + '.json'), results)


def evaluate_results():
        command = "python " + r"scripts/official/score_2021.py "
        os.system(command +  str(cfg['valset']) + " " +  str(cfg['predictValPath']))


if __name__ == "__main__":
        trainPath = Path(r"../datasets/training_II_train").resolve()
        dataSet = ECG([trainPath])
        dataset = torch.utils.data.DataLoader(dataSet, collate_fn=dataSet.collate_fn)
        for item in dataset:
                print(item)
        print(dataset)
```

Appendix B

Hammersley-Clifford Theorem

B.1 Markov Random Field (MRF)

MRF (170; 96) is the probabilistic undirected graph (PUG) model that represents a joint probability distribution with Markov properties. The undirected graph is denoted as $G = \langle \mathscr{V}, \mathscr{E} \rangle$, which denotes a probabilistic distribution $P(Y),\ Y \in \mathscr{Y}$, where $\mathscr{Y} \in \mathbb{R}^n$, Y is a set of random variables. Each vertice of G represents a random variable $Y_v \in Y,\ v \in \mathscr{V}$ defined on v. Probabilistic dependency of random variables is depicted by the edge set $\mathscr{E}$. MRF has three distinguished Markov properties:

- ***Pairwise Markov property.*** **Let $u, v \in \mathscr{V}$ denote two disconnected vertices, the rest of vertices $R = \{\forall r | r \neq u,\ r \neq v,\ r \in \mathscr{V}\}$ and $Y_R = \{\forall Y_r | r \in R\}$, then the conditional probability $P(Y_u, Y_v | Y_R) = P(Y_u | Y_R) \cdot P(Y_v | Y_R)$, where $P(Y_u | Y_R)$ and $P(Y_v | Y_R)$ are marginal probabilistic distributions of $P(Y_u, Y_v | Y_R)$.**

- ***Local Markov property.*** **Let $v \in \mathscr{V}$, $C_v = \{\forall u | u \perp v,\ u \neq v,\ u \in \mathscr{V}\}$, the operator $\perp$ signifies the existence of direct edge connections, let $R_v = \{\forall u | u \neq v,\ u \notin C_v,\ u \in \mathscr{V}\}$ denotes the rest of vertices, then $P(Y_v, Y_{R_v} | Y_{C_v}) = P(Y_v | Y_{C_v}) \cdot P(Y_{R_v} | Y_{C_v})$.**

- ***Global Markov property.*** **Let $V \subset \mathscr{V}$, $C_V = \{\forall u | u \perp V,\ u \in \mathscr{V}\}$, $R = \{\forall u | u \notin V,\ u \notin C_V,\ u \in \mathscr{V}\}$ denotes the rest of vertices, then $P(Y_V, Y_R | Y_{C_V}) = P(Y_V | Y_{C_V}) \cdot P(Y_R | Y_{C_V})$. This property is also called 'the Markov blanket' that separates vertices and ensures the conditional independence of partitioned random variables.**

The sufficient condition of a joint probability distribution $P(Y)$ defined on an undirected graph G being a MRF is one of the three Markov properties listed above. Another equivalent form for local Markov property can be described as corollary. B.1.0.1:

Corollary B.1.0.1. *Local Markov property* $\Leftrightarrow P(Y_v | Y_{C_v}) = P(Y_v | Y_{\mathscr{V} \setminus v})$.

Proof. **Sufficiency.** If $P(Y_v, Y_{R_v} | Y_{C_v}) = P(Y_v | Y_{C_v}) \cdot P(Y_{R_v} | Y_{C_v})$, then $P(Y_v, Y_{R_v} | Y_{C_v}) = P(Y_v | Y_{C_v}, Y_{R_v}) \cdot P(Y_{R_v} | Y_{C_v}) = P(Y_v | Y_{C_v}) \cdot P(Y_{R_v} | Y_{C_v}) \Rightarrow P(Y_v | Y_{C_v}, Y_{R_v}) = P(Y_v | Y_{\mathscr{V} \setminus v}) = P(Y_v | Y_{C_v})$. Next, prove the necessity. □

Proof. **Necessity.** If $P(Y_v|Y_{\mathscr{V}\setminus v}) = P(Y_v|Y_{C_v})$, then $P(Y_v, Y_{R_v}|Y_{C_v}) = P(Y_v|Y_{C_v}, Y_{R_v}) \cdot P(Y_{R_v}|Y_{C_v}) = P(Y_v|Y_{\mathscr{V}\setminus v}) \cdot P(Y_{R_v}|Y_{C_v}) = P(Y_v|Y_{C_v}) \cdot P(Y_{R_v}|Y_{C_v})$. Therefore, corollary B.1.0.1 is proved. □

B.2 Hammersley-Clifford Theorem

A clique is defined as one compete subgraph of G that each pair of vertices in that subgraph are connected by an edge. Maximal clique is the clique that doesn't serve as the proper subset of any other clique. Note that maximal clique is different from the maximum clique, which has the largest number of vertices than any other clique. Factorization of G is defined as the product of functions defined on the set of all maximal cliques $\mathscr{C}$ of G, and this product equals to $P(Y)$:

$$P(Y) = \frac{1}{Z} \prod_{C \in \mathscr{C}} \Psi_C(Y_C), \qquad Z = \sum_{Y} \prod_{c \in \mathscr{C}} \Psi_C(Y_C), \tag{B.1}$$

where Ψ_C is a positive potential function defined on the random variables Y_C of the maximal clique C, Z is a normalizer that sums up the prods of potentials with all possible values of the random variable set Y such that $P(Y) \leq 1$, and becomes a valid probability distribution by ensuring $\sum_Y P(Y) = 1$. $P(Y)$ that can be factorized as Eqn. B.1 is named after a Gibbs distribution. Hammersley-Clifford theorem (28; 34; 179; 55; 142; 169; 15) describes the relationship between a Gibbs distribution and MRF that, Gibbs distribution is equivalent to MRF.

Proof. **Sufficiency of Hammersley-Clifford theorem.** For any vertice $v \in \mathscr{V}$, if $P(Y_v|Y_{C_v}) = P(Y_v|Y_{\mathscr{V}\setminus v})$, where $C_v = \{\forall u | u \perp v,\ u \in \mathscr{V}\}$, which is equivalent to the local Markov property according to corollary. B.1.0.1, then the proposition - 'a Gibbs distribution is a MRF' will be proved. Let set $D_v = v \cup C_v$, we have

$$P(Y_v|Y_{C_v}) = \frac{P(Y_v, Y_{C_v})}{P(Y_{C_v})} = \frac{\sum_{\mathscr{V}\setminus D_v} P(Y)}{\sum_v \sum_{\mathscr{V}\setminus D_v} P(Y)} = \frac{\sum_{\mathscr{V}\setminus D_v} \prod_{C \in \mathscr{C}} \Psi_C(Y_C)}{\sum_v \sum_{\mathscr{V}\setminus D_v} \prod_{C \in \mathscr{C}} \Psi_C(Y_C)}, \tag{B.2}$$

where $P(Y_v, Y_{C_v})$, $P(Y_{C_v})$ are signified by their marginal probabilities, and the normalizer Z is reduced by substituting $P(Y)$ with Eqn. B.1. In addition, let $R_v = \{\forall C | v \in C,\ C \in \mathscr{C}\}$ be the set of cliques that contains vertice v (different from the set R_v defined for local Markov property), $R_v^c = \mathscr{C} - R_v$ be the complement set. Eqn. B.2 is then reformatted as

$$\frac{\sum_{\mathscr{V}\setminus D_v} \prod_{C \in \mathscr{C}} \Psi_C(Y_C)}{\sum_v \sum_{\mathscr{V}\setminus D_v} \prod_{C \in \mathscr{C}} \Psi_C(Y_C)} = \frac{\sum_{\mathscr{V}\setminus D_v} \left(\prod_{C \in R_v} \Psi_C(Y_C) \cdot \prod_{C \in R_v^c} \Psi_C(Y_C)\right)}{\sum_v \sum_{\mathscr{V}\setminus D_v} \left(\prod_{C \in R_v} \Psi_C(Y_C) \cdot \prod_{C \in R_v^c} \Psi_C(Y_C)\right)}. \tag{B.3}$$

The maximal cliques that contains v must come from D_v, since if $v \notin D_v,\ v \in C'$, while maximal clique C' contains v, which, according to the definition of maximal clique, C' must be connected to v such that $C' \subseteq D_v$ thereby $v \in D_v$,

which contradicts with $v \notin D_v$. Thus, $R_v \subseteq D_v$,we can rewrite Eqn. B.3 as

$$\begin{aligned}\frac{\sum_{\mathscr{V}\setminus D_v}\left(\prod_{C\in R_v}\Psi_C(Y_C)\cdot\prod_{C\in R_v^c}\Psi_C(Y_C)\right)}{\sum_v\sum_{\mathscr{V}\setminus D_v}\left(\prod_{C\in R_v}\Psi_C(Y_C)\cdot\prod_{C\in R_v^c}\Psi_C(Y_C)\right)} &= \frac{\prod_{C\in R_v}\Psi_C(Y_C)\cdot\sum_{\mathscr{V}\setminus D_v}\prod_{C\in R_v^c}\Psi_C(Y_C)}{\sum_v\prod_{C\in R_v}\Psi_C(Y_C)\cdot\sum_{\mathscr{V}\setminus D_v}\prod_{C\in R_v^c}\Psi_C(Y_C)} = \frac{\prod_{C\in R_v}\Psi_C(Y_C)}{\sum_v\prod_{C\in R_v}\Psi_C(Y_C)} \\ &= \frac{\prod_{C\in R_v}\Psi_C(Y_C)\cdot\prod_{C\in R_v^c}\Psi_C(Y_C)}{\sum_v\prod_{C\in R_v}\Psi_C(Y_C)\cdot\prod_{C\in R_v^c}\Psi_C(Y_C)} = \frac{\prod_{C\in\mathscr{C}}\Psi_C(Y_C)}{\sum_v\prod_{C\in\mathscr{C}}\Psi_C(Y_C)}.\end{aligned} \tag{B.4}$$

The nominator and denominator of last equity of Eqn. B.4 can both divide the normalizer Z at same time, which formulates the form as shown in Eqn. B.1, and draws to the conclusion that

$$\frac{\prod_{C\in\mathscr{C}}\Psi_C(Y_C)/Z}{\sum_v\prod_{C\in\mathscr{C}}\Psi_C(Y_C)/Z} = \frac{P(Y)}{P(Y_{\mathscr{V}\setminus v})} = \frac{P(Y_v,Y_{\mathscr{V}\setminus v})}{P(Y_{\mathscr{V}\setminus v})} = P(Y_v|Y_{\mathscr{V}\setminus v}). \tag{B.5}$$

□

Proof. **Necessity of Hammersley-Clifford theorem.** For any subset $S \subseteq \mathscr{V}$, construct a potential function $f_S(Y_S)$ on the subset S with the form

$$f_S(Y_S = y_S) = \prod_{V\subseteq S} P(Y_V = y_V, Y_{\mathscr{V}\setminus V} = 0)^{(-1)^{|S|-|V|}}, \tag{B.6}$$

in which keep the random variables in subset V active, while setting probabilities of the rest in subset S to zero, and the $|\cdot|$ operator denotes the number of vertices in the subset. If $|S|-|V|$ is even, then $(-1)^{|S|-|V|}$ equals to 1, otherwise -1. Eqn. B.6 has two properties that are crucial to prove the necessity of Hammersley-Clifford theorem.

- The 1st property - $\prod_{S\subseteq\mathscr{V}} f_S(Y_S) = P(Y)$.
- The 2nd property - If S is not a clique, then $f_S(Y_S) = 1$.

Prove the 1st property first. It can be deducted that

$$\prod_{S\subseteq\mathscr{V}} f_S(Y_S) \Leftrightarrow \prod_{V\subseteq\mathscr{V}} f_V(Y_V). \tag{B.7}$$

$\prod_{S\subseteq\mathscr{V}} f_S(Y_S) = \prod_{S\subseteq\mathscr{V}}\prod_{V\subseteq S} P(Y_V = y_V, Y_{\mathscr{V}\setminus V} = 0)^{(-1)^{|S|-|V|}}$ according to Eqn. B.6, in which S can be viewed as the independent variable with elements of V the constants given S. On the other hand, it can also be viewed as the function depends on V, i.e., $\prod_{V\subseteq\mathscr{V}} f_V(Y_V) = \prod_{S\subseteq\mathscr{V}}\prod_{V\subseteq S} P(Y_V = y_V, Y_{\mathscr{V}\setminus V} = 0)^{(-1)^{|S|-|V|}}$ once $\mathscr{V}$ is given such that elements of S are constants. Both views are concluded as the Eqn. B.7. The largest benefit of changing the viewpoint is that the original problem related to $f_S(Y_S)$ can be interpreted as solving the duality $f_V(Y_V)$. Let's focus on this dual problem of calculating $\prod_{S\subseteq\mathscr{V}} f_S(Y_S)$.

Given some certain $V \subseteq \mathscr{V}$, we can use a strategy to search for all prodding terms related to S. Firstly, let $S := V$, then we find the only prodding term $\Delta_V^{(-1)^0} = P(Y_V = y_V, Y_{\mathscr{V}\setminus V} = 0)^{(-1)^0}$; Next, let $S := V \cup v_1,\ v_1 \in \mathscr{V}$ and

$v_1 \notin V$, as there are $C^1_{|\mathscr{V}|-|V|}$ ways to choose v_1, and in each way there exists only one valid (others are all ones since $|S := V| - |V| = 0$) prodding item $\Delta_V^{(-1)^1}$ according to Eqn. B.6. Continue searching, let $S := V \cup \{v_1, v_2\},\ v_1, v_2 \in \mathscr{V}$ and $v_1, v_2 \notin V$, note that we've covered the case for V w/ additional one vertice, thus there exists only one valid prodding item $\Delta_V^{(-1)^2}$ for each choice according to Eqn. B.6, and there are $C^2_{|\mathscr{V}|-|V|}$ ways to choose. Similarly, choose S until $S := \mathscr{V}$ when at most $|\mathscr{V}| - |V|$ additional elements can be chosen, and the only valid prodding term for choosing additional $|\mathscr{V}| - |V|$ elements is $\Delta_V^{(-1)^{|\mathscr{V}|-|V|}}$, which has $C^{|\mathscr{V}|-|V|}_{|\mathscr{V}|-|V|}$ ways to choose. Collect all valid prodding terms using this searching strategy, we have:

$$\prod_{V \subseteq \mathscr{V}} f_V(Y_V) = \prod_{V \subseteq \mathscr{V}} \Delta_V^{C^0_{|\mathscr{V}|-|V|}(-1)^0 + C^1_{|\mathscr{V}|-|V|}(-1)^1 + C^2_{|\mathscr{V}|-|V|}(-1)^2 + \cdots + C^{\mathscr{V}|-|V|}_{|\mathscr{V}|-|V|}(-1)^{\mathscr{V}|-|V|}}. \tag{B.8}$$

According to Binomial theorem that $(a+b)^n = C_n^0 a^n b^0 + C_n^1 a^{n-1} b^1 + \cdots + C_n^n a^0 b^n$, let $a = 1,\ b = -1$ such that $(1-1)^n = C_n^0 - C_n^1 + \cdots + C_n^r(-1)^r + \cdots + C_n^n(-1)^n$, Eqn. B.8 is then reformulated as

$$\prod_{V \subseteq \mathscr{V}} f_V(Y_V) = \prod_{V \subset \mathscr{V}} f_V(Y_V) \cdot f_{\mathscr{V}}(Y_{\mathscr{V}}) = f_{\mathscr{V}}(Y_{\mathscr{V}}) = P(Y), \tag{B.9}$$

which proves the first property according to Eqn. B.7.

The second step is to prove the second property. Since S is not a clique, we can always find two vertices a and b such that there's no connection (both direct or indirect) between a and b. $f_S(Y_S)$ can be denoted by vertices a, b that,

$$f_S(Y_S) = \prod_{V \subseteq S} P(Y_V = y_V, Y_{\mathscr{V} \setminus V} = 0)^{(-1)^{|S|-|V|}} = \prod_{V \subseteq S \setminus \{a,b\}} \left[\frac{P(Y_V, Y_{\mathscr{V} \setminus V} = 0) \cdot P(Y_{V \cup \{a,b\}}, Y_{\mathscr{V} \setminus (V \cup \{a,b\})} = 0)}{P(Y_{V \cup \{a\}}, Y_{\mathscr{V} \setminus (V \cup \{a\})} = 0) \cdot P(Y_{V \cup \{b\}}, Y_{\mathscr{V} \setminus (V \cup \{b\})} = 0)} \right]^{(-1)^{|S|-|V|}}, \tag{B.10}$$

which remains to be proved. Eqn. B.10 can be more explicitly denoted by Δ_V as

$$f_S(Y_S) = \prod_{V \subseteq S \setminus \{a,b\}} \left[\frac{\Delta_V \cdot \Delta_{V \cup \{a,b\}}}{\Delta_{V \cup \{a\}} \cdot \Delta_{V \cup \{b\}}} \right]^{(-1)^{|S|-|V|}}. \tag{B.11}$$

To prove Eqn. B.11, for each $V \subseteq S \setminus \{a,b\}$, construct three sets $V \cup \{a\},\ V \cup \{b\},\ V \cup \{a,b\}$. Next, for all $V \subseteq S \setminus \{a,b\}$, construct all their three-types sets, as a result, the union of all these sets along with all Vs is equivalent to $\{\forall V | V \subseteq S\}$. Under this circumstance, $f_S(Y_S)$ can be reformulated as

$$\begin{aligned} f_S(Y_S) &= \prod_{V \subseteq S} \Delta_V^{(-1)^{|S|-|V|}} = \prod_{V \subseteq S \setminus \{a,b\}} \left[\Delta_V^{(-1)^{|S|-|V|}} \cdot \Delta_{V \cup \{a\}}^{(-1)^{|S|-|V|-1}} \cdot \Delta_{V \cup \{b\}}^{(-1)^{|S|-|V|-1}} \cdot \Delta_{V \cup \{a,b\}}^{(-1)^{|S|-|V|-2}} \right] \\ &= \prod_{V \subseteq S \setminus \{a,b\}} \left[\frac{\Delta_V^{(-1)^{|S|-|V|}} \cdot \Delta_{V \cup \{a,b\}}^{(-1)^{|S|-|V|}}}{\Delta_{V \cup \{a\}}^{(-1)^{|S|-|V|}} \cdot \Delta_{V \cup \{b\}}^{(-1)^{|S|-|V|}}} \right] = \prod_{V \subseteq S \setminus \{a,b\}} \left[\frac{\Delta_V \cdot \Delta_{V \cup \{a,b\}}}{\Delta_{V \cup \{a\}} \cdot \Delta_{V \cup \{b\}}} \right]^{(-1)^{|S|-|V|}}, \end{aligned} \tag{B.12}$$

which proves proposition Eqn. B.11. Next, rewrite Δ_V and $\Delta_{V\cup\{a\}}$ as

$$\begin{aligned}\Delta_V &= P(Y_V, Y_{\mathscr{V}\setminus V}=0) = P(Y_a=0|Y_b=0, Y_{\mathscr{V}\setminus(V\cup\{a,b\})}=0, Y_V)\cdot P(Y_b=0, Y_{\mathscr{V}\setminus V\cup\{a,b\})}=0, Y_V),\\ \Delta_{V\cup\{a\}} &= P(Y_{V\cup\{a\}}, Y_{\mathscr{V}\setminus(V\cup\{a\})}=0) = P(Y_a|Y_b=0, Y_{\mathscr{V}\setminus(V\cup\{a,b\})}=0, Y_V)\cdot P(Y_b=0, Y_{\mathscr{V}\setminus(V\cup\{a,b\})}=0, Y_V),\end{aligned} \tag{B.13}$$

where $Y_{\mathscr{V}\setminus V}=0$ indicates $Y_a=0$. Similarly, rewrite $\Delta_{V\cup\{a,b\}}$ and $\Delta_{V\cup\{b\}}$ as

$$\begin{aligned}\Delta_{V\cup\{a,b\}} &= P(Y_a|Y_b, Y_{\mathscr{V}\setminus(V\cup\{a,b\})}=0, Y_V)\cdot P(Y_b, Y_{\mathscr{V}\setminus(V\cup\{a,b\})}=0, Y_V),\\ \Delta_{V\cup\{b\}} &= P(Y_a=0|Y_b, Y_{\mathscr{V}\setminus(V\cup\{a,b\})}=0, Y_V)\cdot P(Y_b, Y_{\mathscr{V}\setminus(V\cup\{a,b\})}=0, Y_V).\end{aligned} \tag{B.14}$$

It can be derived from Eqn. B.13 and B.14 that,

$$\frac{\Delta_V}{\Delta_{V\cup\{a\}}} = \frac{P(Y_a=0|Y_b=0, Y_{\mathscr{V}\setminus(V\cup\{a,b\})=0}, Y_V)}{P(Y_a|Y_b=0, Y_{\mathscr{V}\setminus(V\cup\{a,b\})}=0, Y_V)}, \quad \frac{\Delta_{V\cup\{a,b\}}}{\Delta_{V\cup\{b\}}} = \frac{P(Y_a|Y_b, Y_{\mathscr{V}\setminus(V\cup\{a,b\})=0}, Y_V)}{P(Y_a=0|Y_b, Y_{\mathscr{V}\setminus(V\cup\{a,b\})}=0, Y_V)}. \tag{B.15}$$

According to the assumption of MRF, as well as the equivalent fact of pairwise Markov property (proof is similar to corollary. B.1.0.1), as vertice a and b are not connected, they are conditional independent such that

$$\begin{aligned}P(Y_a=0|Y_b=0, Y_{\mathscr{V}\setminus(V\cup\{a,b\})=0}, Y_V) &= P(Y_a=0|Y_b, Y_{\mathscr{V}\setminus(V\cup\{a,b\})=0}, Y_V),\\ P(Y_a|Y_b=0, Y_{\mathscr{V}\setminus(V\cup\{a,b\})=0}, Y_V) &= P(Y_a|Y_b, Y_{\mathscr{V}\setminus(V\cup\{a,b\})=0}, Y_V).\end{aligned} \tag{B.16}$$

Take Eqn. B.15 and B.16 into Eqn. B.12, then the second property is proven. Note that under the assumption of MRF properties, the probabilities in Eqn. B.13 and B.14 are positive to ensure that the potential function $f_S(Y_S)$ is positive. The final step is to deduce the necessity of Hammersley-Clifford theorem by two properties, and it is obvious that $\prod_{S\subseteq\mathscr{V}} f_S(Y_S) = P(Y)$ for all cliques S of $\mathscr{V}$, which also satisfies the condition of all maximal cliques S of $\mathscr{V}$ since a maximal clique is not a proper subset of any other cliques. This indicates that a MRF can be represented by a Gibbs distribution, therefore the necessary condition of Hammersley-Clifford theorem is proven.

□

Appendix C

Copyright Statements

IEEE Copyright Confidential

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of the University of Kansas's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html **to learn how to obtain a License from RightsLink. If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.**

Accepted Papers During the Academic Years

©IEEE. Reprinted, with permission, from

Mo, Xi and Tao, Ke and Wang, Quan and Wang, Guanghui. An Efficient Approach for Polyps Detection in Endoscopic Videos Based on Faster R-CNN, *24th International Conference on Pattern Recognition (ICPR)*, pp.3929–3934, 2018, doi: 10.1109/ICPR.2018.8545174, IEEE.

Mo, Xi and Chen, Xiangyu. Realtime Global Attention Network for Semantic Segmentation, *IEEE Robotics and Automation Letters with ICRA presentation*, 7(2): pp.1574–1580, 2022, doi: 10.1109/LRA.2022.3140443, IEEE.

Mo, Xi and Chen, Xiangyu and Zhong, Cuncong and Li, Rui and Li, Kaidong and Sajid, Usman. Dilated Continuous Random Field for Semantic Segmentation, *IEEE International Conference on Robotics and Automation (ICRA)*, 2022, IEEE.

©Springer. Reprinted, with permission, from

Mo, Xi and Sajid, Usman and Wang, Guanghui. Stereo frustums: A Siamese Pipeline for 3D Object Detection, *Journal of Intelligent & Robotic Systems*, 101(1): pp.1–15, 2021, Springer.