UNIVERSITY OF CALIFORNIA SAN DIEGO

Learning Robust Representations in Random Forest and Deep Neural Networks for Semantic Segmentation

A dissertation submitted in partial satisfaction of the requirements for the degree

Doctor of Philosophy

in

Electrical Engineering (Signal and Image Processing)

by

Byeongkeun Kang

Committee in charge:

Professor Truong Q. Nguyen, Chair
Professor Pamela C. Cosman
Professor William S. Hodgkiss
Professor Ravi Ramamoorthi
Professor Mohan M. Trivedi

2018
The dissertation of Byeongkeun Kang is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

__________________________________________

__________________________________________

__________________________________________

__________________________________________

__________________________________________

Chair

University of California San Diego

2018
DEDICATION

To my parents and sister.
EPIGRAPH

Never too late to start something.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature Page</td>
<td>iii</td>
</tr>
<tr>
<td>Dedication</td>
<td>iv</td>
</tr>
<tr>
<td>Epigraph</td>
<td>v</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>vi</td>
</tr>
<tr>
<td>List of Figures</td>
<td>ix</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xii</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>xiii</td>
</tr>
<tr>
<td>Vita</td>
<td>xv</td>
</tr>
<tr>
<td>Abstract of the Dissertation</td>
<td>xvii</td>
</tr>
<tr>
<td><strong>Chapter 1</strong></td>
<td></td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Semantic Segmentation</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Hand Segmentation</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Random Forest</td>
<td>4</td>
</tr>
<tr>
<td>1.4 Deep Neural Network</td>
<td>5</td>
</tr>
<tr>
<td><strong>Chapter 2</strong></td>
<td></td>
</tr>
<tr>
<td>Cascaded Random Forest for Binary Class</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Cascaded Random Forest</td>
<td>7</td>
</tr>
<tr>
<td>2.2.1 Training</td>
<td>8</td>
</tr>
<tr>
<td>2.2.2 Inference</td>
<td>10</td>
</tr>
<tr>
<td>2.3 Modified bilateral filter</td>
<td>11</td>
</tr>
<tr>
<td>2.4 Experiments and Results</td>
<td>12</td>
</tr>
<tr>
<td>2.4.1 Hand-Object Interaction Dataset</td>
<td>12</td>
</tr>
<tr>
<td>2.4.2 Results</td>
<td>14</td>
</tr>
<tr>
<td>2.5 Summary</td>
<td>16</td>
</tr>
<tr>
<td><strong>Chapter 3</strong></td>
<td></td>
</tr>
<tr>
<td>Random Forest with Learned Representations</td>
<td>17</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>17</td>
</tr>
<tr>
<td>3.2 Notation</td>
<td>19</td>
</tr>
<tr>
<td>3.3 Unconstrained Feature</td>
<td>20</td>
</tr>
<tr>
<td>3.4 Random Forest with Learned Features</td>
<td>23</td>
</tr>
<tr>
<td>3.4.1 Training</td>
<td>23</td>
</tr>
<tr>
<td>3.4.2 Inference</td>
<td>29</td>
</tr>
</tbody>
</table>
3.5 Experiments and Results .............................................. 31
  3.5.1 Hand Segmentation .............................................. 32
  3.5.2 Semantic Segmentation ......................................... 34
3.6 Summary ............................................................... 40

Chapter 4 Depth Adaptive Deep Neural Network ......................... 42
  4.1 Introduction .......................................................... 42
  4.2 Related Works ........................................................ 45
  4.3 Notation ............................................................... 48
  4.4 Depth-adaptive Multiscale Convolution Layer ..................... 50
    4.4.1 Adaptive Perception Neuron ................................ 51
    4.4.2 In-layer Multiscale Neuron ................................... 51
    4.4.3 Depth Difference .............................................. 53
  4.5 Architecture ........................................................ 54
  4.6 Back Propagation .................................................... 55
  4.7 Proof of Depth-Invariance ........................................ 57
  4.8 Experiments and Results .......................................... 60
    4.8.1 Semantic Segmentation ....................................... 60
    4.8.2 Hand Segmentation ............................................ 64
  4.9 Summary ............................................................... 67

Chapter 5 Applications of Segmentation ................................. 69
  5.1 Sign Language Fingerspelling Recognition ........................ 69
    5.1.1 Introduction ................................................... 69
    5.1.2 Dataset ........................................................ 72
    5.1.3 Hand Segmentation ............................................ 73
    5.1.4 Classification .................................................. 74
    5.1.5 Experiments and Results .................................... 76
  5.2 Hand Articulations Tracking ....................................... 79
    5.2.1 Introduction ................................................... 79
    5.2.2 Hand Segmentation ............................................ 81
    5.2.3 Hierarchical Random Sampling ................................ 82
    5.2.4 Adaptive Hand Model ........................................ 83
    5.2.5 Optimization .................................................. 85
    5.2.6 Initialization at Each Frame ................................ 87
    5.2.7 Experiments and Results .................................... 90
  5.3 Summary ............................................................... 93

Chapter 6 Data Augmentation ............................................ 94
  6.1 Introduction ........................................................ 94
  6.2 Related Works ...................................................... 97
    6.2.1 Generative Adversarial Networks ............................ 97
    6.2.2 Image Compression: JPEG .................................... 98
| Figure 1.1: | Illustration of semantic segmentation. | 2 |
| Figure 1.2: | Illustration of random forest. | 4 |
| Figure 2.1: | Illustration of random forest and notations. | 8 |
| Figure 2.2: | Detection of hands using the random forest in the first stage. | 9 |
| Figure 2.3: | Scores depending on the decision boundary using the validation dataset. (a) Score of the random forest in the first stage. (b) Score of the two-stage random forest with the modified bilateral filtering. | 11 |
| Figure 2.4: | The HOI dataset. (a) A collected depth map. (b) The depth map transformed to closer distance. (c) The depth map transformed to further distance. | 13 |
| Figure 2.5: | Analysis of the collected dataset and the augmented dataset. (a) The standard deviation of the depth of hands in mm. (b) The distribution of the distance from the depth sensor to hands. | 13 |
| Figure 2.6: | Qualitative comparison of the results. | 15 |
| Figure 2.7: | Analysis of accuracy and efficiency. | 16 |
| Figure 3.1: | Illustration of the random forest with learned representations for semantic segmentation. | 20 |
| Figure 3.2: | Examples of the proposed feature. | 21 |
| Figure 3.3: | Comparison of the proposed representation and the feature of a convolutional layer in convolutional neural networks. The proposed feature is not constrained by a specific shape. | 22 |
| Figure 3.4: | An example of boosting and bootstrap aggregating. | 26 |
| Figure 3.5: | Scores depending on decision boundary. The scores are computed on the validation set of hand segmentation data using the proposed method (30 trees) without the bilateral filtering. | 31 |
| Figure 3.6: | The qualitative comparison of the result for the HOI dataset. | 34 |
| Figure 3.7: | Analysis in accuracy and efficiency. | 35 |
| Figure 3.8: | The qualitative comparison of the result for the Cityscapes dataset. | 39 |
| Figure 4.1: | Illustration of the captured images and the proposed neural networks. The size of a captured object on the image plane varies with the distance from the object to the camera. | 44 |
| Figure 4.2: | Visual comparison of the receptive fields for typical neural networks and the proposed neural network. | 45 |
| Figure 4.3: | An example of applying the different sizes (sparsities) of the receptive field at each spatial location. | 46 |
| Figure 4.4: | Illustration of (a) typical convolution layer and (b) depth-adaptive multiscale convolution layer. | 46 |
| Figure 4.5: | Notations and the adaptive perception neuron. This neuron adjusts the size of the receptive field based on the depth information at each spatial location. | 49 |
Figure 4.6: The in-layer multiscale neuron. This neuron is able to learn features at different scales in a layer. ................................................................. 53
Figure 4.7: Example of the proposed DaM convolution layers at different distances to demonstrate depth-invariance. ......................................................... 57
Figure 4.8: The qualitative comparison of the result for the NYUDv2 dataset. ........ 64
Figure 4.9: The comparison of convergence curves between the Frontend module and the network with the proposed DaM convolution for the input of RGB. .... 67
Figure 4.10: The qualitative comparison of the result for the HOI dataset. ............ 68
Figure 5.1: ASL fingerspelling alphabets and numbers. .................................. 71
Figure 5.2: Captured image before pre-processing. ........................................ 73
Figure 5.3: Examples of pre-processed dataset from A to Z and from 1 to 9. In real dataset, background is set to zero. ..................................................... 74
Figure 5.4: The collected data for the same meaning. It shows the importance of consideration of different signers and viewpoint variation. ....................... 75
Figure 5.5: Block diagram of the proposed method. ........................................ 81
Figure 5.6: Hierarchical random sampling of pixels from each depth map. ........ 83
Figure 5.7: Hand model. (a) Skeletal structure of hand model with 26 DOFs. Each DOF represents a parameter in hand pose parameter vector. (b) An example of hand model with 48 spheres. Two spheres are hidden at connection between palm and thumb. Hand size parameters are visualized. ................. 84
Figure 5.8: Finger detection. (a) Distance transform. (b) Fingertips (red) and palm center (blue). (c) Computed finger orientation using PCA. ......................... 88
Figure 5.9: Finger classification. (a) Palm orientation. (b) Joint junctions (white: detected, colored: predicted using hand model and palm orientation). (c) Classification result. ......................................................... 89
Figure 5.10: Comparison of adaptive hand model and fixed scale hand model for three subjects ((a) subject 1, (b) subject 2, (c) subject 3). ................................. 92
Figure 6.1: The proposed framework to generate compressed images. The framework consists of a generator, a discriminator, and a transformer between the generator and the discriminator. The visualized generator output is an intermediate example. .............................................................. 96
Figure 6.2: Illustration of the proposed architecture for JPEG compressed image generation. (a) Generator. (b) Discriminator. (c) Residual block. ....................... 101
Figure 6.3: Visual comparison of (a) convolution layer with filter-size of $3 \times 3$, (b) fully connected layer, and (c) proposed locally connected layer with block-size of $4 \times 4$. ................................................................. 103
Figure 6.4: Visual results of generating compressed images using the CIFAR-10 dataset. On the left side, we denote the quality factor for quantization and chroma subsampling ratios. (a) Real data. (b) TTUR (RGB). (c) TTUR (encoded). (d) FC generator (encoded). (e) Proposed method. ............................ 111
Figure 6.5: Visual results of generating compressed images using the LSUN dataset. On the left side, we denote the quality factor for quantization and chroma subsampling ratios.
LIST OF TABLES

Table 2.1: Quantitative comparison. The two boundaries for the proposed method are for each stage. ................................................................. 15
Table 3.1: The quantitative results of the HOI dataset. ................................. 33
Table 3.2: The quantitative results of the Cityscapes dataset. .......................... 37
Table 3.3: Analysis of the proposed method using the validation set of the Cityscapes dataset. ............................................................... 38
Table 3.4: Analysis on learning features using particle swarm optimization. ........ 40
Table 3.5: Analysis on applying neural networks to input images with different resolution. ......................................................... 40
Table 3.6: Analysis on hyperparameters in particle swarm optimization. ............ 41
Table 4.1: The quantitative results of the NYUDv2 dataset. ........................... 63
Table 4.2: Ablation study of selecting multiscale paramter for Frontend module. .... 65
Table 4.3: The effects of applying the DaM convolution to Frontend module for the input of RGB. ................................................................. 65
Table 4.4: The analysis of multi/random scale evaluation using DeepLab-based networks for the input of RGB. ........................................... 66
Table 4.5: The effects of increasing receptive fields in DeepLab. ....................... 66
Table 4.6: The quantitative results of the HOI dataset. ................................. 67
Table 5.1: Quantitative comparison of gesture and ASL fingerspelling recognition systems in accuracy. ....................................................... 77
Table 5.2: Quantitative comparison for each fingerspelling class. ...................... 78
Table 5.3: Processing time of each component. ............................................. 90
Table 5.4: Average error for varying standard deviations of hand size parameter randomness. ........................................... 91
Table 5.5: Correct classification rate (in %) of finger classification. .................... 91
Table 5.6: Performance comparison of random sampling and the proposed hierarchical random sampling. ..................................................... 93
Table 6.1: Quantitative comparison using FIDs for the CIFAR-10 dataset. .......... 109
Table 6.2: Quantitative comparison using FIDs for the LSUN dataset. .............. 110
ACKNOWLEDGEMENTS

I sincerely appreciate my advisor, Professor Truong Nguyen for giving me an opportunity and guiding me throughout my Ph.D. He has always been supportive and accessible, that started even before arriving San Diego. He has always encouraged my research interests, career plans, other experiences, and collaborations. It gave me lots of opportunities to think about and choose what I really want to do. He has also been around me whenever I need his supervision or advice. It certainly helped to resolve many matters smoothly and efficiently.

I am also grateful to my committee members, Professor Pamela Cosman, Professor William Hodgkiss, Professor Ravi Ramamoorthi, and Professor Mohan Trivedi for their helpful suggestions and advice.

It has been delightful working with the members of the Video Processing Lab. Subarna Tripathi has been the closest collaborator during my Ph.D. She has impacted on many projects related to deep neural networks for years. I also thank my other collaborators, Yeejin Lee, Nan Jiang, Chen Du, and Yuanyuan Wu. They are not just collaborators but also intimate friends. I have also enjoyed discussing many diverse things with Shibin Parameswaran. I also thank Zucheul Lee, Kyoungrok Lee, Can Bal, Jason Juang, Bumshik Lee, and Haleh Azartash for their advice about life at the group and at San Diego during my first year. I also thank Lee-Kang Liu, Ji Dai, Yujia Wang, Igor Fedorov, Enming Luo, Junkang Zhang, and Menglin Zeng for making the group more delightful.

I appreciate my parents and sister. My father is the best supporter of what I want to do. My mother is the best advisor to discuss my life. My sister is the best friend who will always be with me. My memory at San Diego has been more pleasant because of my friends.
Parts of this dissertation are based on published or submitted papers.

Chapter 2, in part, is a reprint of materials in the conference paper by “Hand Segmentation for Hand-Object Interaction from Depth map”, B. Kang, K.-H. Tan, N. Jiang, H.-S. Tai, D. Tretter, and T. Q. Nguyen [KTJ+17]. The dissertation author is the primary investigator and author of this paper.

Chapter 3, in part, is a reprint of materials in the submitted journal paper “Random Forest with Learned Representations for Semantic Segmentation”, B. Kang and T. Q. Nguyen [KN18]. The dissertation author is the primary investigator and author of this paper.

Chapter 4, in part, is a reprint of materials in the journal paper “Depth-Adaptive Deep Neural Network for Semantic Segmentation”, B. Kang, Y. Lee, and T. Q. Nguyen [KLN18]. The dissertation author is the primary investigator and author of this paper.

Chapter 5, in part, is a reprint of materials in the conference papers “Real-time Sign Language Fingerspelling Recognition using Convolutional Neural Networks from Depth map”, B. Kang, S. Tripathi, and T. Q. Nguyen [KTN15] and “Efficient Hand Articulations Tracking using Adaptive Hand Model and Depth map”, B. Kang, Y. Lee, and T. Q. Nguyen [KLQN15]. The dissertation author is the primary investigator and author of these papers.

Chapter 6, in part, is a reprint of materials in the submitted journal paper “Toward Joint Image Generation and Compression using Generative Adversarial Networks”, B. Kang, S. Tripathi, and T. Q. Nguyen [KTN18]. The dissertation author is the primary investigator and author of this paper.
VITA

2013  B. S. in Electrical and Electronic Engineering, Yonsei University, Seoul, Republic of Korea

2015  M. S. in Electrical Engineering (Signal and Image Processing), University of California San Diego

2018  Ph. D. in Electrical Engineering (Signal and Image Processing), University of California San Diego

PUBLICATIONS


ABSTRACT OF THE DISSERTATION

Learning Robust Representations in Random Forest and Deep Neural Networks for Semantic Segmentation

by

Byeongkeun Kang

Doctor of Philosophy in Electrical Engineering (Signal and Image Processing)

University of California San Diego, 2018

Professor Truong Q. Nguyen, Chair

As semantic segmentation provides the class and the location of objects in a captured scene, it has been one of the core algorithms in many computer vision applications including autonomous driving, robot navigation, surveillance camera system, and human-machine interaction. Most of these applications demand high accuracy, robustness, and efficiency to understand a captured scene accurately in a timely manner in order to avoid accidents, to provide a meaningful warning, and to communicate naturally. We address this needs by using two popular approaches: random forest and deep neural network.

We start by introducing a cascaded random forest for binary class segmentation. The
framework first detects regions of interest and then segments foreground in the regions. Since the detection reduces the regions for the segmentation forest, the cascaded scheme improves efficiency and accuracy. We then explore learning more robust representations in a random forest. Since predetermined constraints in typical feature extractors restrict learning and extracting optimal features, we present a random forest framework that learns the weights, shapes, and sparsities of feature extractors. We propose an unconstrained filter, an iterative optimization algorithm for learning, a processing pipeline for inference. Experimental results demonstrate that the proposed method achieves real-time semantic segmentation using limited computational and memory resources.

Moreover, we present a method to learn/extract depth-adaptive features in a deep neural network. It accomplishes a step toward depth-invariant feature learning and extracting. Since typical neural networks receive inputs from predetermined locations regardless of the distance from the camera, it is challenging to generalize the features of objects at various distances. Hence, we propose the depth-adaptive multiscale convolution layer consisting of the adaptive perception neuron and the in-layer multiscale neuron. The adaptive neuron is to adjust the receptive field at each spatial location using the depth information. The multiscale neuron is to learn features at multiple scales. Experimental results show that the proposed method outperforms the state-of-the-art methods without any additional layers or pre/post-processing.

Lastly, we present applications of segmentation including sign language fingerspelling recognition and hand articulation tracking. We also present a potential data augmentation method using generative adversarial networks.
Chapter 1

Introduction

1.1 Semantic Segmentation

Semantic segmentation is classifying each pixel in an image to a class label in order to provide both the class and the location of things as shown in Figure 1.1. Consequently, it has become vital for many computer vision applications including autonomous driving, robot navigation/task, smart factory/market, and augmented reality. Autonomous vehicles need to recognize potential pathways, pedestrians, other vehicles, and traffic signs. Robots need to understand obstacles in their pathways and recognize various objects for their tasks. Augmented reality needs to recognize user’s view and visualize proper information on top of it. Despite substantial advancements in computer vision and machine learning, current techniques still fail to deliver sufficient robustness and accuracy for safety-related applications in varying environments and conditions. To improve it, systems with multiple/advanced sensors have been developed such as multi-camera system and light detection and ranging (LiDAR) sensors. One of the principal advantages of these systems is the ability to acquire distance information. However, research on utilizing the distance information has been limited. Hence, we introduce systems that learn/extract more optimal feature extractors for all the obtained information in order to improve the robustness,
Figure 1.1: Illustration of semantic segmentation. Given an input image on the left side, algorithms aim to predict a class label for each pixel as shown in the right figure.

accuracy, and efficiency of semantic segmentation.

To achieve accurate and efficient pixel-wise classification, many researches have been conducted [SJC08, SFC+11, SGF+13, LSD15, SLD17, YK16, ZJRP+15, CPK+15, CPK+18, PCKC16]. Shotton et al. presented a random forest-based method and applied it for semantic segmentation and body pose estimation in [SJC08, SFC+11, SGF+13]. This work has been broadly employed in many related applications [RYMZ13, SKR+15, TSLP14] and in Microsoft Kinect [Zha12]. To improve accuracy in semantic segmentation, convolutional neural network-based methods have been proposed in [LSD15, YK16, ZJRP+15, CPK+15, CPK+18, PCKC16]. Part of the reason that deep learning-based methods outperform other methods is its ability to learn fine features along with the hierarchical structure and non-linear activations. The methods using random forest and using deep learning outperform most of the other methods in the task of semantic segmentation in the aspect of efficiency and accuracy. Between two approaches, random forest-based methods have an advantage in computational complexity and memory usage while deep learning-based methods achieve higher accuracy. Therefore, we present random forest-based methods and convolutional neural network-based methods.
1.2 Hand Segmentation

Along with typical semantic segmentation tasks for road scenes and indoor scenes, we also explore hand segmentation for hand-object interaction. Recently, with the expansion of virtual reality (VR), augmented reality (AR), robotics, and intelligent vehicles, the development of new interaction technologies has become unavoidable since these applications require more natural interaction methods rather than input devices. For these applications, many researches have been conducted such as gesture recognition and hand pose estimation. However, most technologies focus on understanding interactions which do not involve touching or handling any real-world objects although understanding interactions with objects is important in many applications. We believe that this is because hand segmentation is much more difficult in hand-object interaction. Thus, we explore frameworks of hand segmentation for hand-object interaction.

Hand segmentation has been studied for many applications such as hand pose estimation [TSLP14, SKR⁺15, QSW⁺14, RKK10, RKEK13, WMZ⁺13], hand tracking [OKA11b, AL04, KLQN15], and gesture/sign/grasp recognition [CKS15, KTN15]. In color image-based methods, skin color-based method has been popular [JR02, KHSB12, LK13, CKS15, PBC05, KMB07]. For hand-object interaction, Oikonomidis et al. and Romero et al. segmented hands by thresholding skin color in the hue-saturation-value (HSV) space [OKA11b, RKK10, RKEK13, AL04]. Wang et al. processed hand segmentation using a learned probabilistic model where the model is constructed from the color histogram of the first frame [WMZ⁺13]. Tzionas et al. applied skin color-based segmentation using the Gaussian mixture model [TG15]. However, skin color-based segmentation has limitations in interacting with objects in skin color, segmenting from other body parts, skin pigment difference, and light condition variations. An alternative method is wearing a specific color glove [WP09].

For depth map-based methods, popular methods are using a wrist band [KTN15, KLQN15, QSW⁺14] or using random decision forest (RDF) [TSLP14, SFC⁺11, SKR⁺15]. Although the
method using a black wristband is uncomplicated and effective, it is inconvenient. Moreover, the method cannot segment hands from objects during hand-object interaction since it processes segmentation by finding connected components. Tompson et al. [TSLP14] and Sharp et al. [SKR+15] proposed the RDF-based methods based on [SFC+11]. We also present depth map-based methods for hand segmentation to apply for hand-object interaction.

1.3 Random Forest

Random forest is an ensemble learning method and consists of a set of decision trees [Ho95, Bre01, CS13] as shown in Figure 1.2. It is robust to noisy and variant data because of the combination of multiple trees with varying features and splitting criteria. Also, it is computationally less complex than typical neural networks. Part of the reason is that an input data is processed only log-scale portion of each tree based on the conditions in the ancestral nodes (see blue dotted lines in Figure 1.2).

Shotton et al. presented semantic texton forest for image categorization and semantic segmentation [SJC08]. The forest uses splitting functions of the value of a single pixel, the sum, the difference, and the absolute difference of a pair of pixels. Hence, the method does

---

**Figure 1.2:** Illustration of random forest. Red, black, and green circles represent root nodes, split nodes, and leaf nodes, respectively. At leaf nodes, the forest estimates and uses the conditional probability of being each class given the specific leaf node.
not require expensive computations of local descriptors (e.g. histogram of oriented gradients (HOG) \([\text{FR94}]\), scale-invariant feature transform (SIFT) \([\text{Low99}]\)) or filter-bank responses. It is extremely fast comparing to k-means clustering or nearest-neighbor assignment of feature descriptor. Schroff \textit{et al.} investigated using not only local features but also global and context-rich features in the random forest for semantic segmentation \([\text{SCZ08}]\). They showed that combining multiple features leads to the improvements in accuracy. They further demonstrated that relaxing constraints on features leads to higher classification accuracy. Shotton \textit{et al.} extended the random forest \([\text{SJC08, LF06}]\) to classify each pixel to body parts in real-time from a single depth map \([\text{SFC}^{+11}, \text{SGF}^{+13}]\). They employed depth difference feature of a pair of pixels with depth invariance property by compensating offsets using depth information. Tompson \textit{et al.} applied the random forest to perform per-pixel classification for hand segmentation \([\text{TSLP14}]\). Sharp \textit{et al.} employed the random forest-based method and its memory-efficient method \([\text{SSK}^{+13}]\) to predict initial hand pose in hand tracking \([\text{SKR}^{+15}]\).

The earlier works used hand-crafted features such as local descriptors (e.g. HOG, SIFT) or filter-bank responses \([\text{YGYL11}]\). Then, relatively recent works used pixel value difference as a feature that learns offsets while using fixed weights (+1 and -1) \([\text{LF06, SFC}^{+11}, \text{SGF}^{+13}, \text{TSLP14}]\). In Chapter 2, we present the cascaded random forest method consisting of detection and segmentation for hand segmentation. The cascaded method uses pixel value difference as a feature, following previous works. In Chapter 3, we present the random forest framework that employs unconstrained features and learns optimal features by using an optimization algorithm.

### 1.4 Deep Neural Network

Deep convolutional neural network-based methods have achieved the state-of-the-art accuracy in per-pixel classification in the recent years. Long \textit{et al.} proposed fully convolutional neural networks (FCN) for semantic segmentation by converting fully connected layers
to convolutional layers in the neural networks for image classification [LSD15, SLD17]. Consequently, it takes an input of arbitrary size and produces the output of the corresponding size with pixel-wise prediction. Additional efforts have been made to improve the performance in [ZJRP15, CPK15, CPK18, YK16]. Zheng et al. proposed the convolutional neural networks that combine the strength of convolutional neural networks and conditional random field (CRF)-based probabilistic graphical modeling. They formulated CRF as recurrent neural networks and attached the recurrent neural networks following FCN [ZJRP15]. Chen et al. improved semantic segmentation using convolution with upsampled filters, atrous spatial pyramid pooling, and fully connected CRF [CPK15, CPK18]. Yu et al. proposed an additional context module to aggregate multiscale information without reducing spatial resolution [YK16].

Several other methods designed neural networks considering computational complexity [BKC17, PCKC16]. Badrinarayanan et al. proposed an efficient convolutional neural network in terms of memory and computational time during inference [BKC17]. The network consists of an encoder network and a decoder network. The encoder network records pooling indices in the max-pooling step and the corresponding decoder network uses the recorded indices to perform nonlinear upsampling. Chaurasia et al. presented an efficient neural network to achieve real-time semantic segmentation [PCKC16]. They analyzed the network design in many aspects including early downsampling, nonlinear operations, and factoring filtering.

While neural network-based methods achieve state-of-the-arts accuracy by learning optimal weights and biases in multiple layers with nonlinear activation functions and by extracting meaningful information from an input data, they demand high computational and memory resources.

In Chapter 4, we present the depth-adaptive deep neural network that compensates depth variation to achieve depth-invariance property for semantic segmentation.
Chapter 2

Cascaded Random Forest for Binary Class

2.1 Introduction

Random forest consists of a collection of decision trees as shown in Figure 2.1. Each decision tree is composed of a root node, splitting nodes, and leaf nodes. Given an input data at the root node, it is classified to child nodes based on the split function at each splitting node until it reaches a leaf node. In this chapter, the input data is the location of each pixel on a depth map. The split function uses the feature of the depth difference between two relative points on the depth map in \([SFC^{+11}]\). At a leaf node, a conditional probability distribution is learned in a training stage, and the learned probability is used in a testing stage. For more details about random forest, we refer the readers to [CS13, Bre01, Ho95].

2.2 Cascaded Random Forest

We propose a two-stage random forest for binary class segmentation to achieve high accuracy and efficiency. Among many binary class segmentation problems, we select to approach hand segmentation for hand-object interaction considering high demands in many applications.
We also choose to use only a depth map to avoid the limitations of skin color-based methods.

In the two-stage random forest, the first random forest detects hands by processing the random forest on an entire depth map. Then, the second random forest segments hands in pixel-level by applying the random forest in the detected region. This cascaded architecture is designed for the second random forest to focus on the segmentation of hands from objects and close body parts such as an arm.

2.2.1 Training

Given a training dataset $\mathcal{D}$, the algorithm randomly selects a set $\mathcal{D}_i$ of depth maps $D$ and then randomly samples a set of data points $x$ in the region of interest (ROI) on the selected depth maps $D$. The ROI is the entire region of the depth maps in the first stage. It is the detected regions using the first random forest in the second stage (see Figure 2.2). The sampled set of data points $x$ and the corresponding depth maps $D$ are inputs to the training of a decision tree.

Using the inputs $(x, D)$, the algorithm learns a split function at each splitting node and a conditional probability distribution at each leaf node. First, learning the split function includes learning a feature $f(\cdot)$ and a criteria $\theta$. We use the feature $f(\cdot)$ of the depth difference between
two relative points \( \{x + u/D_x, x + v/D_x\} \) in [SFC+11] as follows:

\[
f(x, D, u, v) = D_{x+u/D_x} - D_{x+v/D_x}
\]

(2.1)

where \( D_x \) denotes the depth at a pixel \( x \) on a depth map \( D \); \( u \in \mathbb{R}^2 \) and \( v \in \mathbb{R}^2 \) represent offset vectors for each relative point. Then, the criteria \( \theta \) decides to split the data \( x \) to the left child or the right child.

\[
f(x, D, u, v) \leq \theta.
\]

(2.2)

Thus, the algorithm learns two offset vectors \((u, v)\) and a criteria \( \theta \) at each splitting node.

Since the goal is separating the data points \( x \) of different classes to different child nodes, the objective function is designed to evaluate the separation using the learned offset vectors and criteria as follows:

\[
L(x, D, u, v, \theta) = -\sum_{c \in \{l, r\}} \sum_{h \in \{0, 1\}} \frac{|x_c|}{|x|} p(h|c) \log p(h|c)
\]

(2.3)

where \( c \) and \( h \) are indices for child nodes \( \{l, r\} \) and for classes, respectively; \(|x_c|\) denotes the number of data points in the \( c \) child node; \( p(h|c) \) is the estimated probability of being the class \( h \) at the child node \( c \).
To learn offsets and a criteria, the algorithm randomly generates possible candidates and selects the candidate with a minimum loss $L(\cdot)$ as follows:

$$(u, v, \theta) = \arg\min_{(u, v, \theta)} L(x, D, u, v, \theta).$$

(2.4)

Learning a split function at each splitting node is repeated until the node satisfies the condition for a leaf node. The condition is based on (1) the maximum depth of the tree, (2) the probability distribution $p(h|c)$, and (3) the amount of training data $|x|$ at the node. Specifically, it avoids too many splitting nodes by limiting the maximum depth of the tree and by terminating if the child node has a high probability for a class or if the amount of remaining training data is too small.

At each leaf node, the algorithm stores the conditional probability $p(h|l)$ (probability of being each class $h$ given reaching the node $l$) for the prediction in a testing stage.

### 2.2.2 Inference

Using the learned random forest, the algorithm predicts the probability of being a class for a new data $x$. The new data is classified to child nodes using the learned split function at each splitting node until it reaches a leaf node. At the leaf node $l$, the learned conditional probability $p_T(h|l)$ is loaded. These steps are repeated for entire trees $T$ in the forest $\mathcal{T}$. Then, the probabilities are averaged to predict the probability $p(h|x)$ of being a class $h$ for the new data $x$.

$$p(h|x) = \frac{1}{|\mathcal{T}|} \sum_{T \in \mathcal{T}} p_T(h|l)$$

(2.5)

where $|\mathcal{T}|$ is the number of trees in the learned forest $\mathcal{T}$.

In the first stage, the first random forest is applied on an entire depth map to compute a probability map. Then, the probability map is used to detect hands as shown in Figure 2.2. In the second stage, the second random forest processes the data points in the detected regions to
predict the probability of being each class. The proposed two-stage random forest improves both accuracy and efficiency by focusing on each task in each stage.

Decision boundaries are exhaustively searched with the step size of 0.01 using the predicted probability maps of the validation dataset as shown in Figure 2.3. Although the most typical boundary is 0.5 for a probability map, we found that it is not the best parameter. The selected boundaries are shown in Table 2.1.

### 2.3 Modified bilateral filter

Before classifying a data \( x \) to a class \( h \), modified bilateral filter is applied to the predicted probability \( p(h|x) \) to make the probability more robust. Since the probability \( p(h|x) \) is predicted for each pixel independently, the probability is stabilized by averaging the probabilities of the data points in close distance and similar intensity on the depth map.

Unlike typical bilateral filter whose weights are based on the input image (in this case, the probability map) [TM98], the weights in the modified bilateral filter are based on a separate
The filtering is defined as follows:

\[ \tilde{p}(h|x) = \frac{1}{w} \sum_{x_i \in \Omega} g_r(|D_{x_i} - D_x|) g_s(||x_i - x||) p(h|x_i) \]  \hspace{1cm} (2.6)

where \( \Omega \) is the set of pixels within the filter’s radius and the pre-defined depth difference; \( w \) is the normalization term, \( w = \sum_{x_i \in \Omega} g_r(|D_{x_i} - D_x|) g_s(||x_i - x||) \); \( g_r(\cdot) \) and \( g_s(\cdot) \) are the Gaussian functions for the depth difference and for the spatial distance from the data point \( x \), respectively.

\[ g_r(r) = \exp\left(-\frac{r^2}{2\sigma_r^2}\right), \quad g_s(s) = \exp\left(-\frac{s^2}{2\sigma_s^2}\right). \]  \hspace{1cm} (2.7)

The parameters in the filter were selected based on the experiments using validation dataset. The selected parameters are as follows: the maximum depth difference to be considered is 400mm. Both standard deviations (\( \sigma_r \) and \( \sigma_s \)) are 100.

\section*{2.4 Experiments and Results}

\subsection*{2.4.1 Hand-Object Interaction Dataset}

We collected a new dataset\(^1\) using Microsoft Kinect v2 since we were not able to find a publicly available dataset for hand-object interaction with pixel-wise annotation. The collected dataset consists of more than 9,175 pairs of depth maps and color images from 6 people (3 males and 3 females) interacting with 21 different objects. In addition, the dataset includes the cases of one hand and both hands in a scene. Ground truth was labeled by wearing a color glove during data collection and by finding the color of the glove on the color images.

To increase the variation of the dataset further (e.g. the distance from the camera to hands), 18,350 pairs of images were augmented by moving the camera closer/further to/from the scene as

\(^1\)https://github.com/byeongkeun-kang/HOI-dataset
Figure 2.4: The HOI dataset. (a) A collected depth map. (b) The depth map transformed to closer distance. (c) The depth map transformed to further distance.

Figure 2.5: Analysis of the collected dataset and the augmented dataset. (a) The standard deviation of the depth of hands in mm. (b) The distribution of the distance from the depth sensor to hands.

shown in Figure 2.4. In total, the augmented dataset has 27,525 pairs of depth maps and ground truth labels. Indeed, the standard deviation of the augmented data increases to 725 relative to that of the collected dataset is 225, as evidenced in Figure 2.5(a). The distances of the augmented dataset are distributed at more diverse distances as demonstrated in Figure 2.5(b).

Among 27,525 pairs, we used 19,470 pairs for training, 2,706 pairs for validation, and 5,349 pairs for testing.
2.4.2 Results

The proposed method is analyzed by demonstrating the results on the dataset in Section 2.4.1. For the quantitative comparison of accuracy, we measure $F_1$ score, precision, and recall as follows:

$$
\text{precision} = \frac{tp}{tp + fp}, \quad \text{recall} = \frac{tp}{tp + fn}
$$

$$
F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{2.8}
$$

where $tp$, $fp$, and $fn$ represent true positive, false positive, and false negative, respectively. For the comparison of efficiency, we measure the processing time using a machine with Intel i7-4790K CPU and Nvidia GeForce GTX 770.

The proposed method is compared with the random forest-based method in [TSLP14, SFC$^+$11] and the fully convolutional networks (FCN) in [LSD15, SLD17] using only a depth map. The proposed method is not compared with color-based methods since the characteristics of depth sensors and color imaging sensors are quite different. For example, a captured depth map using a depth sensor does not vary depending on light condition. However, a captured color image varies a lot depending on light condition. Thus, choosing the capturing environment affects the comparison of results using depth maps and color images. Hence, we only compare the proposed method with the state-of-the-art methods which can process using only depth maps.

Table 2.1 and Figure 2.6 show quantitative results and visual results. The scores in Table 2.1 are scaled by a factor of 100. The quantitative results show that the proposed method achieves about 25% and 8% relative improvements in $F_1$ score comparing to the random forest-based methods [TSLP14, SFC$^+$11] and its combination with the proposed method in Section 2.2.2, respectively. Comparing to the deep learning-based methods [LSD15, SLD17], the proposed method achieves about 7% lower accuracy, but processes in about 42 times faster processing time. Thus, deep learning-based methods can not be used in real-time applications.
Table 2.1: Quantitative comparison. The two boundaries for the proposed method are for each stage.

<table>
<thead>
<tr>
<th>Method</th>
<th>Boundary</th>
<th>Filter</th>
<th>Precision</th>
<th>Recall</th>
<th>$F_1$ score</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF [TSLP14, SFC$^+$11]</td>
<td>0.50</td>
<td>-</td>
<td>38.1</td>
<td>91.2</td>
<td>53.7</td>
<td>6.7</td>
</tr>
<tr>
<td>RF [TSLP14, SFC$^+$11] +</td>
<td>0.78</td>
<td>-</td>
<td>54.5</td>
<td>72.7</td>
<td>62.3</td>
<td>6.7</td>
</tr>
<tr>
<td>Proposed in Section 2.2.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FCN-32s [LSD15, SLD17]</td>
<td>-</td>
<td>-</td>
<td>70.0</td>
<td>68.6</td>
<td>69.3</td>
<td>376</td>
</tr>
<tr>
<td>FCN-16s [LSD15, SLD17]</td>
<td>-</td>
<td>-</td>
<td>68.0</td>
<td>72.2</td>
<td>70.1</td>
<td>376</td>
</tr>
<tr>
<td>FCN-8s [LSD15, SLD17]</td>
<td>-</td>
<td>-</td>
<td>70.4</td>
<td>74.4</td>
<td>72.3</td>
<td>377</td>
</tr>
<tr>
<td>Proposed method</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.50, 0.50</td>
<td>-</td>
<td>59.2</td>
<td>77.4</td>
<td>67.1</td>
<td>8.9</td>
</tr>
<tr>
<td></td>
<td>0.50, 0.52</td>
<td>-</td>
<td>60.8</td>
<td>75.1</td>
<td>67.2</td>
<td>8.9</td>
</tr>
<tr>
<td></td>
<td>0.50, 0.52</td>
<td>11 × 11</td>
<td>62.9</td>
<td>75.6</td>
<td>68.7</td>
<td>10.7</td>
</tr>
</tbody>
</table>

Figure 2.6: Qualitative comparison of the results. (a) Ground truth label. (b) Result using RF [TSLP14, SFC$^+$11]. (c) Result using RF [TSLP14, SFC$^+$11] with the proposed method in Section 2.2.2. (d) Result using FCN-8s [LSD15, SLD17]. (e) Result using the proposed method. The results and ground truth label are visualized using different color channels for better visualization.
Figure 2.7: Analysis of accuracy and efficiency. 

shows the comparison of methods in accuracy and efficiency. The proposed method achieves high accuracy in short processing time.

2.5 Summary

In this chapter, we present two-stage random forest method for hand segmentation for hand-object interaction using only a depth map. The two stages consist of detecting the region of interest and segmenting hands. The proposed method achieves high accuracy in short processing time comparing to the state-of-the-art methods.

This chapter, in part, is a reprint of materials in the conference paper by “Hand Segmentation for Hand-Object Interaction from Depth map”, B. Kang, K.-H. Tan, N. Jiang, H.-S. Tai, D. Tretter, and T. Q. Nguyen [KTJ+17]. The dissertation author is the primary investigator and author of this paper.
Chapter 3

Random Forest with Learned Representations

3.1 Introduction

Accurate and efficient semantic segmentation is a fundamental task in a variety of computer vision applications including autonomous driving, human-machine interaction, and robot navigation. Reducing computational complexity and memory use is important to minimize response time and power consumption for portable devices such as robots and virtual/augmented devices. Also, it is beneficial for vehicles and robots to navigate in actively changing environments and for human-machine interaction devices to communicate without delay. However, it is challenging to achieve accurate and efficient semantic segmentation because every pixel needs to be classified using limited computational resources.

To achieve accurate and efficient pixel-wise classification, Shotton et al. presented a random forest-based method and applied it for semantic segmentation and body pose estimation in [SJC08, SFC+11, SGF+13]. This work has been broadly employed in many related applications [RYMZ13, SKR+15, TSLP14] and in Microsoft Kinect [Zha12]. To improve accuracy
in semantic segmentation, convolutional neural network-based methods have been proposed in \[\text{LSD15, YK16, ZJRP}^{+15, \text{CPK}^{+15, \text{CPK}^{+18}}\}.\] Part of the reason that deep learning-based methods outperform other methods is its ability to learn fine features along with the hierarchical structure and non-linear activations. The methods using random forest and using deep learning outperform most of the other methods in the task of semantic segmentation in the aspect of efficiency and accuracy. Between two approaches, random forest-based methods have an advantage in computational complexity and memory usage while deep learning-based methods achieve higher accuracy. Therefore, we present the random forest framework that employs unconstrained features, learns optimal features by using an optimization algorithm, and inferences in real-time with limited computational and memory resources.

The proposed method can be applied to any input (e.g. color image, depth map, point cloud) and for any pixel-wise classification task. We validate the proposed method in the task of semantic segmentation and hand segmentation for hand-object interaction. Semantic segmentation is needed in many applications such as autonomous driving and robot navigation \[\text{[COR}^{+16, \text{GLU12, GLSU}13a, \text{BSFC}08, \text{BFC}09}\].\] We apply the proposed method to road scene dataset \[\text{[COR}^{+16}\] and use color images along with depth maps as an input. Hand segmentation is also a fundamental task in human-machine interaction that is demanded in virtual reality (VR), augmented reality (AR), robotics, and user interfaces in an automobile \[\text{[TG15, CKS15, RYMZ13, WP09, WMZ}^{+13, \text{SKR}^{+15, \text{RKEK}13, \text{RKK}10, \text{TSLP}14, \text{QSW}^{+14, \text{OKA}11b, \text{KLQN}15, \text{KTN}15, \text{GLYT}16, \text{LK}13, \text{SCR}16, \text{LYT}14, \text{SRHC}12}\].\] We apply the method to depth map-based hand segmentation dataset for hand-object interaction \[\text{[KLN}18\].\] In experiments, we show that the proposed method can be applied to real-time semantic segmentation task with limited computational and memory resources.

In this chapter, we propose the novel unconstrained filter that is able to learn weights, shapes, and sparsities in Section 3.3. Then, we propose the random forest framework that learns the weights, shapes, and sparsities of the feature and inferences input data using the learned
feature in Section 3.4. The section explains selecting training data using bootstrap aggregation and boosting, learning splitting functions using particle swarm optimization, and inferencing input images. In Section 3.5, we demonstrate the effectiveness of the proposed method on two tasks: road scene semantic segmentation and hand segmentation for hand-object interaction. We use publicly available Cityscapes dataset [COR+16] and HOI dataset [KLN18].

In summary, the contributions of our work are as follows:

• We propose the unconstrained feature that is able to represent any weights, shapes, and sparsities.

• We develop the random forest framework including learning the weights, shapes, and sparsities of the feature by using particle swarm optimization.

• We verify the effectiveness and efficiency of the proposed method on semantic segmentation and hand segmentation tasks.

3.2 Notation

Let \( X \in \mathbb{R}^{p \times q} \) and \( Y \in \mathbb{R}^{p \times q \times n_c} \) be the matrices denoting an input image and an output probability map of a random forest where \( p, q, \) and \( n_c \) represent the height, the width, and the number of classes, respectively. At the location \( x \in \mathbb{R}^2 \) on the image \( X \), the intensity is represented as \( X_x \). Also, let \( \mathcal{T}, T_i, \) and \( n_t \) indicate the random forest, the \( i \)-th decision tree, and the number of trees in the forest.

\[
\mathcal{T} = \{ T_i | i \leq n_t \text{ and } i \in \mathbb{Z}^+ \}.
\] (3.1)
Figure 3.1: Illustration of the random forest with learned representations for semantic segmentation. The top images show input images (a color image and a depth map). The bottom image shows an expected output (class labels for each pixel). The rectangles with grid show examples of learned features with various shapes, sparsities, and weights. Red bounding boxes represent input data points, and blue color squares denote the offset points that are used to compute the features.

3.3 Unconstrained Feature

Learning optimal feature is essential to achieve higher accuracy and to avoid the unnecessary use of computation and memory. Hence, we design an unconstrained filter that is able to learn optimal weights, shapes, and sparsities. In the proposed framework, the unconstrained filter’s weights, shapes, and sparsities are learned at each node to split the data points of different classes to separate child nodes as shown in Figure 3.1. Moreover, the filter extracts depth and shift invariant features by compensating the associated changes as described in Figure 3.2. Hence, the proposed novel feature improves accuracy and reduces computational complexity and memory use.

Given an input data (spatial location) $x$ and the corresponding input image and depth map
Figure 3.2: Examples of the proposed feature. Each figure shows two different features where the same color line denotes the same feature. Each red circle represents a data point \( x \). The length of each line denotes the magnitude of each offset \( u_i/D_x \) considering the depth \( D_x \) at the data point \( x \). The end-point of each line shows each offset point \( x + u_i/D_x \) used to compute the corresponding feature. The feature is computed by Eq. (3.2) where the training algorithm in Section 3.4.1 learns weights (coefficients) \( w \), offsets \( u \), and the number \( n_f \) of data points.

\[
f(x,X,D) = \sum_{i=1}^{n_f} w_i X_{x+u_i/D_x,h}
\]

where \( n_f \) is the number of data points used to compute each feature; \( w \) and \( u \in \mathbb{R}^2 \) are a weight and an offset parameter vector, respectively; \( h \) is the channel index of \( X \).

In details, the offset data point \( x + u/D_x \) (the other end-point from the red circle \( x \) in Figure 3.2) is determined based on \( x \), \( u \), and \( D_x \). The weight \( w \) manages the influence of the information at the offset data point \( X_{x+u/D_x} \). The feature response is computed by the weighted summation of the offset data points. All the parameters including the number \( n_f \) of data points, the weights \( w \), the offsets \( u \), and the channel \( h \) are learned in the training procedure in Section 3.4.1. Each learned feature consists of \( 4 \times n_f + 1 \) parameters where 4 is from a weight, two offset parameters, and a channel index and 1 is from the number of data points. The proposed feature is described in Figures 3.2 and 3.3.

If any offset data point \( x + u/D_x \) is beyond the boundary of the image, the intensity
Figure 3.3: Comparison of the proposed representation and the feature of a convolutional layer in convolutional neural networks. The proposed feature is not constrained by a specific shape. Color squares denote offset points $x + u/D_x$ used to compute the feature, and the red bounding boxes represent the data points $x$.

$X_{x+u/D_x}$ is replaced by a constant (the maximum intensity of the input image $X$ if $X$ is a depth map; 0 otherwise).

$$
X_{x+u/D_x} = \begin{cases} 
X_{x+u/D_x} & \text{if } x + u/D_x \in \mathbb{Z}^+ \text{ and } x + u/D_x \leq (p,q), \\
\max(X) & \text{else if } X \text{ is a depth map}, \\
0 & \text{otherwise}.
\end{cases}
$$

(3.3)

Obviously, when an input image $X$ is a depth map $D$, the feature can be expressed as follows:

$$f(x,X) = \sum_{i=1}^{n_f} w_i X_{x+u_i/x_x}.$$ 

(3.4)

Additionally, if depth information is not available, the requirement of depth data can be
relaxed by sacrificing the property of depth invariance. The relaxed feature is defined as follows:

\[ f(x, X) = \sum_{i=1}^{n_f} w_i X_{x+u_i}. \]  

(3.5)

### 3.4 Random Forest with Learned Features

Random forest consists of a set of decision trees as shown in Figure 3.1. It is robust to noisy and irregular data because of the combination of multiple trees with varying features. It is also computationally less complex than typical neural networks since, in the inference procedure, each input data is processed only small part of the forest based on the conditions in the ancestral nodes (see blue dotted lines in Figure 3.1).

Each decision tree in the random forest is composed of a root node, splitting nodes, and leaf nodes. The input of each tree is the location \( x \in \mathbb{R}^2 \) of an input data and the corresponding input image and depth map \((X, D)\). Given the input \((x, X, D)\) at a root node, the input data is classified to a child node based on the splitting criteria \( f_n(x, X, D) \leq \theta \) where \( f_n(\cdot) \) represents a learned feature at the node \( n \). The classification to a descendant node is terminated when the input data reaches a leaf node. At the leaf node, the conditional probability \( p(c|f_n(x, X, D) \leq \theta) \) for \( \forall n \) until the leaf node) of being each class \( c \) is learned in a training stage and is used in an inference stage. The leaf nodes are also learned in a training procedure. For more details about the random forest, we refer the readers to [CS13, Bre01, Ho95, LF06].

#### 3.4.1 Training

In a training stage, the random forest learns a splitting criteria \( f_n(x, X, D) \leq \theta \) at each splitting node \( n \) and a conditional probability distribution \( p(c|f_n(x, X, D) \leq \theta) \) for \( \forall n \) until the leaf node)
of being each class \( c \) at each leaf node. The splitting criteria at a node \( n \) is denoted as follows:

\[
f_n(x, X, D) = \sum_{i=1}^{n_f} w_i X_{x+u_i/D_x} \leq \theta. \tag{3.6}
\]

In (3.6), one redundant parameter can be eliminated by dividing each side by \( \theta \) while the data split remains equivalent.

\[
\sum_{i=1}^{n_f} \frac{w_i}{\theta} X_{x+u_i/D_x} \leq 1,
\]

\[
\sum_{i=1}^{n_f} w'_i X_{x+u_i/D_x} \leq 1 \text{ where } w'_i = \frac{w_i}{\theta}. \tag{3.7}
\]

Thus, the training algorithm only needs to learn the parameters in the feature function \( f(\cdot) \) while the splitting boundary is always 1. In the rest of this chapter, a weight \( w \) represents \( w' \).

In the following paragraphs, we introduce the strategy of selecting training data using bootstrap aggregating and boosting (see Figure 3.4). We then present the method of learning features using particle swarm optimization. Finally, we describe the condition for determining leaf nodes and the process of learning conditional probability distribution.

**Bootstrap aggregating.**

Bootstrap aggregating (bagging) is to amalgamate multiple classifiers trained using randomly selected training data sets. This method improves robustness and accuracy by integrating multiple classifiers with variance caused by the randomly selected training data. Given the training data sets \( X \) of images, the bagging algorithm selects random sets \( X_i \subset X \) of images for the \( i \)-th tree. Then, the algorithm samples \( n_d \) data points (pixels) for each class from the images \( X \in X_i \). Thus, each tree is trained using \( n_d \times n_c \) data points where \( n_c \) is the total number of classes.

For hand segmentation, we sample the same number of data points for each image and for each class to consider diverse images equally regardless of the size of the hand, etc. For
semantic segmentation, since all objects do not appear on all the images, we only constrain the same number of data points for each class.

**Boosting.**

Boosting is to combine a set of weak classifiers to form a stronger classifier. Especially, adaptive boosting is learning weak classifiers based on the obtained result by applying previously trained weak classifiers [FS99]. We apply adaptive boosting at each set of trees by adjusting the sampling of training data based on the segmentation result at the stage.

As explained in the bagging, a random set $X_i \subset X$ of images is selected for the $i$-th tree $T_i$. Then, $n_d$ data points $x$ are sampled for each class $c$. The proposed boosting algorithm adjusts this sampling of data points $x$ for each set of trees. For the first set of trees, data points $x$ are sampled randomly with uniform distribution for each class. Thus, the probability $p_s(\cdot)$ of being sampled for a data point $x$ of a class $c$ is as follows:

$$p_s(x) = \frac{n_d}{n_d \cdot n_X}$$  \hspace{2cm} (3.8)

where $n_{dc}$ is the number of the data points of a class $c$ in the image $X$ for hand segmentation and in the set $X$ of images for semantic segmentation, respectively. $X$ is the image that $x$ belongs to. $n_X$ is 1 for semantic segmentation and is the number of images in $X$ for hand segmentation.

After training the first set of trees, the sampling probability $p_s(\cdot)$ is updated to train the second set of trees more effectively. It is achieved by adjusting $p_s(\cdot)$ to sample more data points with higher errors. Hence, $p_s(\cdot)$ is increased for the data points with a high error and is decreased for the data with a low error. Given the learned set of trees $T$ until the current iteration, an inference is processed to estimate the probability $p_c(c|x,X,D)$ of being a target class $c$ for the data in the training dataset $X$. Then, $p_s(\cdot)$ is updated as follows:

$$p_s(x) = 1 - p_c(c|x,X,D).$$  \hspace{2cm} (3.9)
Figure 3.4: An example of boosting and bootstrap aggregating. (a) Depth map. (b) Sampling probability $p_s(\cdot)$ in the first iteration of boosting. (c) Sampled data points in the first iteration. (d) Inferred probability using the first set of learned trees. (e) Ground truth label. Gray and black color denote hand and non-hand class, respectively. (f) Sampling probability $p_s(\cdot)$ in the second iteration of boosting. (g) Sampled data points in the second iteration. (h) Inferred probability using the first and the second set of trees. Magenta and yellow color represent non-hand and hand class, respectively. Brighter color in (b) and (f) denotes higher sampling probability $p_s(\cdot)$. Brighter color in (d) and (h) represents the higher probability of being hand class. The number $n_d$ of sampled data points is 250 for each class in this example.

By using the data points sampled based on the updated $p_s(\cdot)$, the next set of trees is trained. $p_s(\cdot)$ is adjusted repeatedly at each iteration until the training terminates.

Figure 3.4 shows a visual example of boosting and bootstrap aggregating. The first row and the second row show the first iteration and the second iteration of boosting, respectively. In the first iteration, $p_s(\cdot)$ in (b) is the same for all data points of the same class. Thus, the sampled data points in (c) are distributed uniformly. In the second iteration, $p_s(\cdot)$ in (f) varies for each data point depending on the inferred probability in (d) that is obtained using the first set of trees. Consequently, the sampled data points in (g) are distributed based on $p_s(\cdot)$ in (f).
Particle swarm optimization.

Particle swarm optimization (PSO) is applied to learn a feature that splits the data points of different classes into separate child nodes at each splitting node. PSO is selected to find a more optimal solution in a high-dimensional parameter space \( \mathbb{R}^{3 \times n_f} \cdot \mathbb{Z}^1 \) if \( X \) is a depth map, and additional \( \mathbb{Z}^{n_f} \) if \( X \) is a color image).

Assuming \( X \) is a depth map, a feature parameter vector \( p \in \mathbb{R}^{3 \times n_f \cdot \mathbb{Z}^1} \) consists of the number \( n_f \in \mathbb{Z}^+ \) of data points and a weight \( w \in \mathbb{R} \) and an offset parameter vector \( u \in \mathbb{R}^2 \) for each data point. Thus, the total number of parameters is \( 3 \times n_f + 1 \). To limit the solution space \( \mathbb{R}^{3 \times n_f \cdot \mathbb{Z}^1} \), the maximum number of data points \( n_f \) is chosen as 9 which is equivalent to the number of data points in a filter (kernel) with the size of \( 3 \times 3 \). It is also the most common size of a kernel in convolutional neural networks.

In the first iteration, the algorithm generates 100 offset candidates \( u \) and 100 weight candidates \( w \) where the candidates are sampled from the uniform distribution. Then, the algorithm tries the combinations of the 9 different numbers \( n_f \) of data points, 100 offset candidates \( u \), and 100 weight candidates \( w \), totaling 90,000 particles (candidates). By applying the particles, the optimization algorithm learns the global best state \( q_g \) that is the best solution among the entire particles. To simplify and expedite training, the number \( n_f \) of data points is decided as the number of data points of the global best state \( q_g \) in the first iteration. Also, among 10,000 particles, 100 particles are chosen by selecting the best weight candidate for each offset candidate. It is feasible since 10,000 particles are generated by the combinations of 100 offset candidates \( u \) and 100 weight candidates \( w \).

From the second iteration, the 100 particles are used to find the optimal solution. Let \( q \in \mathbb{R}^{3 \times n_f} \) represents each particle consisting of weights \( w \) and offsets \( u \). At an iteration \( t \), the particles \( q^t \) are first updated using the particles \( q^{t-1} \), the personal best states \( q^{t-1}_p \), and the global best state \( q^{t-1}_g \) in the previous iteration \( t - 1 \). The personal best state \( q^{t-1}_p \) is the best state of each particle until the current iteration \( t \). The global best state \( q^{t}_g \) is the best state among all the particles.
until the current iteration \( t \). The particles at an iteration \( t \) are updated as follows:

\[
q_{p,i}^{t-1} = \{ q_{\tilde{i},\tilde{r}}^{t} = \arg\min_{i} L(q_{i}^{t}) \}, \\
q_{g}^{t-1} = \{ q_{\tilde{i},\tilde{r}}^{t} = \arg\min_{i} L(q_{i}^{t}) \}, \\
q_{i}^{t} = q_{i}^{t-1} + \alpha_{p}(q_{p,i}^{t-1} - q_{i}^{t-1}) + \alpha_{g}(q_{g}^{t-1} - q_{i}^{t-1})
\]

(3.10)

where \( i \) denotes the index for each particle. \( L(\cdot) \) is the objective function in (3.12). \( \alpha_{p} \) and \( \alpha_{g} \) are the weights towards the personal best state \( q_{p} \) and the global best state \( q_{g} \). Both parameters (\( \alpha_{p} \) and \( \alpha_{g} \)) are randomly generated from the Normal distribution \( \mathcal{N}(\mu, \sigma^{2}) \) as follows:

\[
\tilde{\alpha} \sim \mathcal{N}(1.0, 0.25), \\
\alpha = \max(0, \tilde{\alpha})
\]

(3.11)

where \( \mu \) and \( \sigma \) represent a mean and a standard deviation. The iteration of PSO is terminated when the loss \( L(\cdot) \) in (3.12) is not decreased at each iteration or after maximum 100 iterations.

The objective function \( L(\cdot) \) consists of a term for classification loss and two terms for regularization. The classification loss is to evaluate a feature’s ability of separating the data points of different classes to separate child nodes. The regularization is to prefer smaller weights \( w \) and the smaller number \( n_{f} \) of data points.

\[
L(p) = \underbrace{C(p)}_{\text{classification loss}} + \underbrace{\lambda_{w} \sum_{i=1}^{n_{f}} w_{i}^{2} + \lambda_{n_{f}} n_{f}}_{\text{regularization}} \\
= -\sum_{h} \sum_{c} \frac{n(h)}{\sum_{h} n(h)} p(c|h) \log p(c|h) + \underbrace{\lambda_{w} \sum_{i=1}^{n_{f}} w_{i}^{2} + \lambda_{n_{f}} n_{f}}_{\text{regularization}}
\]

(3.12)

where \( h \) is an index for a child node (e.g. left child, right child); \( c \) is an index for a class; \( n(h) \) denotes the number of data points in a child node \( h \); \( p(c|h) \) is the probability of being the class \( c \).
at the node $h$; $\lambda_w$ and $\lambda_{nf}$ are weights for each regularization term.

Following learning an optimal feature at a splitting node, the data is split into child nodes. If the child node does not meet the condition for becoming a leaf node, learning a feature and splitting to child nodes are repeated. Otherwise, splitting is terminated and a leaf node is formed.

**Leaf node.**

A leaf node is formed based on the following criteria: (1) the maximum depth of a tree, (2) the probability distribution $p(c|h)$, and (3) the number of training data $x$ at the node. In details, a leaf node is generated if (1) the current depth of a tree is deeper than the maximum depth; (2) the probability at the node is considerably confident for a class; (3) the number of remaining training data is too small. When a leaf node is formed, the conditional probability is stored for inference processing. The conditional probability $p(c|f_n(x,X,D) \leq \theta \text{ for } \forall n \text{ until the leaf node})$ is computed using the number of data points for each class at the leaf node $h$.

$$
p(c|f_n(x,X,D) \leq \theta \text{ for } \forall n \text{ until the leaf node}) = p(c|h) = \frac{n(h,c)}{\sum_{c=1}^{N_c} n(h,c)} \tag{3.13}
$$

where $n(h,c)$ represents the number of data points for each class $c$ at the leaf node $h$.

### 3.4.2 Inference

Given the trained random forest $T$, each data point $x$ on an image $X$ is classified to child nodes using each tree until it reaches a leaf node. When it reaches a leaf node using each tree $T_i$, the learned conditional probability distribution $p_{T_i}(c|f_n(x,X,D) \leq \theta \text{ for } \forall n \text{ until the leaf node})$ of being each class $c$ at the leaf node is loaded. Then, the conditional probability distributions from the entire trees $T_i \in T$ are averaged to estimate the inferred probability $p_c(c|x)$ of the data.
point $x$ being a class $c$.

$$p_c(c|x) = \frac{1}{n_t} \sum_{i=1}^{n_t} p_{T_i}(c|f_n(x,X,D) \equiv \theta \text{ for } \forall n \text{ until the node})$$  \hspace{1cm} (3.14)$$

where $n_t$ is the number of trees in the random forest $T$.

**Modified bilateral filter.**

Since the probability $p_c(c|x)$ is predicted for each pixel independently, the probability can be stabilized considering nearby predictions [KK11, KTJ+17]. We apply simple modified bilateral filter [TM98, KTJ+17] that processes weighted averaging of the probabilities of the data points in close distance and similar intensity on the input $X$. The filtering is defined as follows:

$$\tilde{p}_c(c|x) = \frac{1}{N} \sum_{x_i \in \Omega} g_r(|X_{x_i} - X_x|) g_s(\|x_i - x\|) p_c(c|x)$$  \hspace{1cm} (3.15)$$

where $\Omega$ is the set of pixels within the filter’s radius and the input difference; $N$ is the normalization term.

$$N = \sum_{x_i \in \Omega} g_r(|X_{x_i} - X_x|) g_s(\|x_i - x\|).$$  \hspace{1cm} (3.16)$$

$g_r(\cdot)$ and $g_s(\cdot)$ are the Gaussian functions for an input difference $r$ and a distance $s$, respectively.

$$g_r(r) = \exp \left( -\frac{r^2}{2\sigma_r^2} \right), \quad g_s(s) = \exp \left( -\frac{s^2}{2\sigma_s^2} \right).$$  \hspace{1cm} (3.17)$$

By experimenting on the validation dataset, the maximum color and depth difference to consider are 100 and 400mm, respectively. Both the standard deviations ($\sigma_r$ and $\sigma_s$) are 100.
Figure 3.5: Scores depending on decision boundary. The scores are computed on the validation set of hand segmentation data using the proposed method (30 trees) without the bilateral filtering.

**Decision boundary.**

Given the filtered probability $\tilde{p}_c(c|x)$, a decision boundary needs to be determined for classification. Although the most common method is choosing the class $c$ with the highest probability $\tilde{p}_c(c|x)$, it is not guaranteed to be the best solution. Thus, for hand segmentation, the possible boundaries are tested with the step size of 0.01 using the validation dataset. Figure 3.5 shows the $F_1$ score, precision, and recall on the validation dataset depending on the decision boundary. The scores are computed using the proposed method with 30 trees and without the bilateral filtering. The best $F_1$ score is achieved at the decision boundary of 0.59. For semantic segmentation, the class with the highest probability is selected considering the high complexity caused by the high dimensional decision boundary.

### 3.5 Experiments and Results

The proposed method is applied to two applications: road scene semantic segmentation and hand segmentation for hand-object interaction. The experimental results demonstrate that
the proposed method outperforms typical random forest by learning unconstrained features using particle swarm optimization and processes efficiently comparing to neural network-based methods.

For comparison, we report mean intersection over union (IU) per category and per class for road scene semantic segmentation dataset and precision, recall, and $F_1$ score for hand segmentation for hand-object interaction. Let $n_{ij}$ be the number of pixels that belong to the class $i$ and are predicted to the class $j$, and $n_c$ be the total number of classes or categories.

$$IU = \frac{1}{n_c} \sum_i \left( \frac{n_{ii}}{\sum_j (\sum_j n_{ij} + \sum_j n_{ji} - n_{ii})} \right),$$

$$\text{Precision} = \frac{n_{11}}{n_{11} + n_{01}},$$

$$\text{Recall} = \frac{n_{11}}{n_{11} + n_{10}},$$

$$F_1 = \frac{2n_{11}}{2n_{11} + n_{01} + n_{10}},$$

where for hand segmentation, class 1 is hand, and class 0 is others.

For the quantitative comparison of efficiency, we measure processing time using a machine with Intel i7-4790K CPU with 4.00GHz, 16.0GB RAM, and NVIDIA GeForce GTX 770 for hand segmentation and the same machine with NVIDIA Tesla K40c for semantic segmentation.

### 3.5.1 Hand Segmentation

**Experiments**

We experimented using the dataset in Section 2.4.1. We trained 30 trees using the proposed method. Each tree is trained with 2,000 pairs of depth maps and ground truth labels and 500 data points from each image. After training every 10 trees, we applied boosting by computing errors and by sampling based on the errors. The condition for becoming a leaf node was 0.99 for class probability, 0.0001 for remaining data portion, and 25 for the depth of the tree.
Table 3.1: The quantitative results of the HOI dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th># of trees</th>
<th>Decision boundary</th>
<th>Bilateral filter</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 score</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF [SFC+11, TSLP14]</td>
<td>20</td>
<td>0.50</td>
<td>-</td>
<td>37.9</td>
<td>91.9</td>
<td>53.7</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>0.50</td>
<td>-</td>
<td>37.9</td>
<td>92.1</td>
<td>53.7</td>
<td>14</td>
</tr>
<tr>
<td>RF [SFC+11, TSLP14]</td>
<td>20</td>
<td>0.79</td>
<td>-</td>
<td>54.9</td>
<td>72.7</td>
<td>62.5</td>
<td>10</td>
</tr>
<tr>
<td>+ DB adjustment</td>
<td>30</td>
<td>0.79</td>
<td>-</td>
<td>54.8</td>
<td>73.0</td>
<td>62.6</td>
<td>14</td>
</tr>
<tr>
<td>FCN-32s [LSD15]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>70.0</td>
<td>68.6</td>
<td>69.3</td>
<td>376</td>
</tr>
<tr>
<td>FCN-16s [LSD15]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>68.0</td>
<td>72.2</td>
<td>70.1</td>
<td>376</td>
</tr>
<tr>
<td>FCN-8s [LSD15]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>70.4</td>
<td>74.4</td>
<td>72.3</td>
<td>377</td>
</tr>
<tr>
<td>Frontend [YK16]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>72.4</td>
<td>70.2</td>
<td>71.3</td>
<td>718</td>
</tr>
<tr>
<td>Proposed</td>
<td>20</td>
<td>0.50</td>
<td>-</td>
<td>50.7</td>
<td>89.4</td>
<td>64.7</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>0.59</td>
<td>-</td>
<td>62.2</td>
<td>76.3</td>
<td>68.5</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>0.50</td>
<td>-</td>
<td>53.4</td>
<td>88.4</td>
<td>66.5</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>0.59</td>
<td>5 × 5</td>
<td>65.3</td>
<td>76.0</td>
<td>70.3</td>
<td>41</td>
</tr>
</tbody>
</table>

Result

The quantitative results and the qualitative results are shown in Table 3.1 and Figure 3.6, respectively. The proposed method achieves about 31% and 12% relative improvement in $F_1$ score comparing to the typical RF-based method [SFC+11, TSLP14] and its combination with the decision boundary adjustment in Section 3.4.2. Comparing to the deep learning-based methods [LSD15, YK16], it achieves quite competitive results (3% lower than the best method) in $F_1$ score. Also, the processing time of the proposed method is about 9 times shorter than those of the deep learning-based methods [LSD15, YK16]. Figure 3.7 shows the analysis of the methods in accuracy and efficiency.
3.5.2 Semantic Segmentation

Dataset

The Cityscapes dataset contains the images of urban street scenes [COR⁺16]. The dataset consists of 5,000 finely annotated images and 19,998 sparsely annotated images. We train models for the standard 19 classes problem using the standard data separation of 2,975 finely annotated images and 19,998 sparsely annotated images for training, 500 images for validation, and 1,525 images for testing.
Figure 3.7: Analysis in accuracy and efficiency.

Experiments

We trained five trees using the proposed framework. Each tree is trained with [12,974, 12,973, 7,658, 7,656, 7,658] pairs of images and ground truth labels. The first two trees are trained using the entire finely annotated images and the half of sparsely annotated images. The last three trees are trained with 1/3 of entire training data sets. The selection of the number of images is based on both experimental and intuitive choice. After training each tree, we applied boosting by sampling the set of data points for the next tree based on the current predictions. The condition for becoming a leaf node is 0.99 for class probability, [0.000001 or 0.000002] for remaining data portion, and 25 for the depth of the tree.

Although predicting using random forest is computationally and memory efficient process, training large forest with a huge amount of data is computationally expensive. It is especially time-consuming during training procedure since each node needs to be optimized conditioned on ancestor nodes where the number of nodes can be up to 67,108,863 considering the maximum depth of 25. Hence, we altered the training algorithm to reduce learning time.

One of the most time-consuming processes is loading training images repeatedly. We used over 7,656 images to train each tree where the original resolution of each image is $2048 \times 1024$. The required memory to hold 7,656 color images is $7,656 \times 2,048 \times 1,024 \times 3$ bytes (44.9 GB).
which are not accessible in modern single GPU. One option is loading partial sets of images multiple times at each node. However, it is a considerably time-consuming process since loading 1,000 images from a hard disk drive needs 51 seconds and repeating at 1,000 nodes demands 14 hours. Hence, we proposed to decrease the resolution of images to hold the entire set of training data in memory after loading once for a tree. Specifically, we resized images to $512 \times 256$ so that the required memory to hold 12,974 color images is 4.8 GB. Given 11 GB in modern single GPU, the rest of memory is utilized to process learning algorithms.

Using the remaining memory, we were able to load and process 103,792/382,900 samples for 12,974/7,658 color images where each sample consists of frame index, $x$, $D_x$, and label. However, this number of samples is too small since the average number of data points at depth 15 becomes 3.2 ($103,792 / 2^{15}$) and 11.7 ($382,900 / 2^{15}$). Hence, we employ multiple sets of data points (specifically, 64 sets) so that the average number at depth 15 becomes 202.7 and 747.9 data points. We start training with 64 sets at a root node and merge the sets at a deeper node since the number of data points in a set decreases as the data splits to child nodes. While multiple sets are employed to provide enough data, we use a single set (possibly, merged from multiple sets) among them to train a splitting node to reduce computation and data transfer time.

### Results and Analysis

We compare the proposed method to deep learning-based methods [LSD15, SLD17, ZJRP$^+$15, PCKC16, BKC17] in Table 3.2 and Figure 3.8. We also show the results of the proposed method with various conditions on the validation set in Table 3.3. We report mean IU per category and per class, processing time, and memory usage. Since optimizing post-processing is beyond the scope of this work, we report computation and memory use excluding the demand for bilateral filtering. The overall results show that the proposed method processes each image using small computation and memory resources while achieving meaningful precision. It demonstrates that the proposed method can be applied for real-time semantic segmentation and...
Table 3.2: The quantitative results of the Cityscapes dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th># of trees</th>
<th>Bilateral filter</th>
<th>Input size</th>
<th>Accuracy (IU)</th>
<th>Time (ms)</th>
<th>GPU memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCN-8s [LSD15, SLD17]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>85.7</td>
<td>1365</td>
<td>5800</td>
</tr>
<tr>
<td>CRF as RNN [ZJRP+ 15]</td>
<td>-</td>
<td>-</td>
<td>2048</td>
<td>82.7</td>
<td>10704</td>
<td>31836</td>
</tr>
<tr>
<td>ENet [PCKC16]</td>
<td>-</td>
<td>-</td>
<td>×</td>
<td>80.4</td>
<td>2966</td>
<td>284</td>
</tr>
<tr>
<td>SegNet basic [BKC17]</td>
<td>-</td>
<td>-</td>
<td>1024</td>
<td>79.1</td>
<td>1392</td>
<td>4564</td>
</tr>
<tr>
<td>SegNet extend [BKC17]</td>
<td>-</td>
<td>-</td>
<td>×</td>
<td>79.8</td>
<td>1992</td>
<td>9158</td>
</tr>
<tr>
<td>Proposed method</td>
<td>5</td>
<td>23×23</td>
<td>512×256</td>
<td>60.2</td>
<td>30</td>
<td>1271</td>
</tr>
</tbody>
</table>

can be employed in a low-end GPU or embedded system that demand small power and memory consumption.

We show the results of applying the same random forest model to the inputs of varying resolutions using the validation set in Table 3.3. The results demonstrate that the proposed method is robust to scaled input images by simply adjusting the sparsities in the learned features. It is feasible since the sparsity in the proposed feature is defined in floating-point precision. Table 3.5 shows the results of applying the FCN model trained with the input resolution of 2048×1024 to the inputs of different resolutions using the validation set. The accuracy degrades significantly since deep learning-based methods utilize integer-point precision (mostly, 1) for sparsity.

We verify the effectiveness of particle swarm optimization comparing to random search [SFC+11, TSLP14, KTJ+17] in Table 3.4. The analysis shows that the decision tree trained with particle swarm optimization outperforms the other decision tree optimized with random search while complexity (the number of nodes, processing time, and memory use) is similar. It verifies that particle swarm optimization learns more optimal weights, shapes, sparsities for the proposed feature.

In Table 3.6, we show empirical results to decide hyperparameters for particle initialization in particle swarm optimization using the validation set. The hyperparameters denote the range for
Table 3.3: Analysis of the proposed method using the validation set of the Cityscapes dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th># of trees</th>
<th>Bilateral filter resolution</th>
<th>Input resolution</th>
<th>Accuracy (IU)</th>
<th>Time (ms)</th>
<th>GPU memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Category</td>
<td></td>
<td>Class</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proposed</td>
<td>1</td>
<td>-</td>
<td>15×15, 91×91</td>
<td>50.0, 23.0</td>
<td>75</td>
<td>329</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>-</td>
<td>13×13, 91×91</td>
<td>53.3, 25.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>-</td>
<td>1024×512</td>
<td>41.1, 17.4</td>
<td>21</td>
<td>290</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>-</td>
<td>13×13, 43×43</td>
<td>54.0, 26.0</td>
<td>106</td>
<td>1280</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>-</td>
<td>512×256</td>
<td>41.8, 17.8</td>
<td>6</td>
<td>281</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>-</td>
<td>13×13, 23×23</td>
<td>54.9, 26.6</td>
<td>30</td>
<td>1271</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>-</td>
<td>256×128</td>
<td>52.8, 25.2</td>
<td>2</td>
<td>279</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>-</td>
<td>13×13</td>
<td>54.2, 26.3</td>
<td>8</td>
<td>1269</td>
</tr>
</tbody>
</table>
Figure 3.8: The qualitative comparison of the result for the Cityscapes dataset. (a) Input image. (b) Results of the FCN-8s model [LSD15, SLD17]. (c) Results of the proposed method using five trees, bilateral filter of $13 \times 13$, and the input resolution of $256 \times 128$. (d) Results of the proposed method using five trees, bilateral filter of $23 \times 23$, and the input resolution of $512 \times 256$. (e) Ground truth labels.
Table 3.4: Analysis on learning features using particle swarm optimization.

<table>
<thead>
<tr>
<th>Method</th>
<th># of nodes</th>
<th>Bilateral filter</th>
<th>Accuracy (IU)</th>
<th>Time (ms)</th>
<th>GPU memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Category</td>
<td>Class</td>
<td></td>
</tr>
<tr>
<td>Random</td>
<td>855,974</td>
<td>- 11x11 51x51 71x71</td>
<td>31.5</td>
<td>11.3</td>
<td>102</td>
</tr>
<tr>
<td>PSO</td>
<td>767,953</td>
<td>- 11x11 51x51 71x71</td>
<td>37.2</td>
<td>14.9</td>
<td>83</td>
</tr>
</tbody>
</table>

Table 3.5: Analysis on applying neural networks to input images with different resolution.

<table>
<thead>
<tr>
<th>Method</th>
<th>Input resolution</th>
<th>Accuracy (IU)</th>
<th>Time (ms)</th>
<th>GPU memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Category</td>
<td>Class</td>
<td></td>
</tr>
<tr>
<td>FCN-8s</td>
<td>2048x1024</td>
<td>84.0</td>
<td>64.0</td>
<td>1365</td>
</tr>
<tr>
<td>[LSD15, SLD17]</td>
<td>1024x512</td>
<td>77.9</td>
<td>57.4</td>
<td>468</td>
</tr>
<tr>
<td></td>
<td>512x256</td>
<td>66.9</td>
<td>42.3</td>
<td>216</td>
</tr>
<tr>
<td></td>
<td>256x128</td>
<td>50.8</td>
<td>23.0</td>
<td>162</td>
</tr>
</tbody>
</table>

the uniform distribution described in Section 3.4.1. Although evolved particles can move outside of the initial boundary, initializing particles in a desirable range is important. Initial locations are important since the number of particles is limited and computational resources can be preserved.

3.6 Summary

In this chapter, we present the random forest framework that employs the novel unconstrained feature to achieve real-time semantic segmentation. The unconstrained feature is proposed to learn optimal features in the random forest framework to achieve higher accuracy. The random forest framework is selected to achieve high efficiency for real-time processing. The results verify that the proposed method achieves higher accuracy comparing to previous ran-
Table 3.6: Analysis on hyperparameters in particle swarm optimization.

<table>
<thead>
<tr>
<th>Weight</th>
<th>Offset</th>
<th># of nodes</th>
<th>Accuracy (IU)</th>
<th>Category</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.2</td>
<td>1,960</td>
<td>43.0</td>
<td></td>
<td>17.6</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>1,984</td>
<td>43.1</td>
<td></td>
<td>17.6</td>
</tr>
<tr>
<td>0.2</td>
<td>0.1</td>
<td>1,942</td>
<td>42.5</td>
<td></td>
<td>16.8</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>1,954</td>
<td>43.2</td>
<td></td>
<td>17.6</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>1,988</td>
<td><strong>43.3</strong></td>
<td></td>
<td><strong>17.8</strong></td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>1,964</td>
<td>43.1</td>
<td></td>
<td>17.5</td>
</tr>
<tr>
<td>0.3</td>
<td>0.2</td>
<td>1,952</td>
<td>43.2</td>
<td></td>
<td>17.3</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>1,950</td>
<td>43.2</td>
<td></td>
<td>17.7</td>
</tr>
</tbody>
</table>

Random forest frameworks and processes using much smaller computational and memory resources comparing to deep learning-based methods.

This chapter, in part, is a reprint of materials in the submitted journal paper “Random Forest with Learned Representations for Semantic Segmentation”, B. Kang and T. Q. Nguyen [KN18]. The dissertation author is the primary investigator and author of this paper.
Chapter 4

Depth Adaptive Deep Neural Network

4.1 Introduction

Depth perception, which is one of the crucial abilities in the human visual system, allows human to perceive the distance to an object and to understand the world in three dimensions. The human visual system uses the perceived depth information to robustly estimate the size and shape of objects in three dimensions. The three-dimensional information helps to better understand the objects and scenes along with other cues such as color information. Thus, depth information plays a key role in understanding the visual world.

As depth information is crucial for understanding the visual world, many researches have been explored ways to acquire accurate depth information efficiently in both hardware systems and software systems. In hardware-based solutions, advanced depth sensors, such as Microsoft Kinect and light detection and ranging (LiDAR) sensors have been developed to capture better quality depth information with portability and low cost [Zha12, AP96, Sch10]. In software-based solutions, disparity estimation algorithms using single or multiple cameras have been studied to estimate accurate depth cues in shorter processing time [LN15, LN14, RVCK16]. Owing to these successes in both communities, depth information has been widely usable in many
computer vision applications such as human pose estimation [SFC+11, SGF+13], indoor scene understanding [SHKF12], and autonomous driving [GLSU13b, COR+16].

After perceiving depth and/or color information, a machine processes the perceived information to understand the visual world. One of the recent popular frameworks for learning visual information is the deep neural network, which is loosely inspired by the neurons of a human brain. As computing capability of machines has increased drastically, deep neural networks have attained a huge improvement in understanding visual information and shown the state-of-the-art performance in many tasks such as image classification [KSH12a, SZ15, HZRS16], object detection [GDDM14, Gir15, RHGS15, RDGF16, TLBN16], and semantic segmentation [LSD15, SLD17, CPK+15, CPK+18, YK16].

Because of the importance of depth information and the improvements by using deep neural networks, it has been speculated that incorporating depth information with neural networks has the advantage in understanding visual information. In most researches on deep neural networks using depth information, the depth map has been treated as an image equivalent input to the networks [TSLP14, GLYT16, SCR16, WZC+16, KTN15]. In such networks, the neurons share the predetermined receptive fields in a convolutional layer, which hinders the networks from learning common representations of an object. Considering that a pinhole camera captures an object at different distances, the camera captures the same object in different sizes, as demonstrated in Figure 4.1. The illustration implies that a neural network can possibly learn/extract different features for the same object at various distances, yielding the confusions of recognizing objects. Hence, we propose the novel deep neural networks that learn common features of the same object by leveraging depth information (Section 4.5). The proposed neural networks perceive the same region of the object regardless of the distance from the camera to each pixel as described in Figure 4.2. This is achieved by the novel Depth-adaptive Multiscale or DaM convolution layer consisting of the adaptive perception neuron and the in-layer multiscale neuron in Section 4.4. The adaptive perception neuron adjusts the size of the receptive field at each spatial
Figure 4.1: Illustration of the captured images and the proposed neural networks. The size of a captured object on the image plane varies with the distance from the object to the camera.

location corresponding to the distance from the camera. The adjustment requires a coefficient to decide the ideal correlation between the size of the receptive field and the distance. Since the optimal coefficient differs depending on the objects, better performance can be achieved by utilizing multiple coefficients in a layer. This is implemented by the proposed in-layer multiscale neuron. The in-layer multiscale neuron learns/extracts diversely scaled representations in a layer by applying a different size of the receptive field at each feature representation. The adjustment of the receptive field is applied using the sparse convolution (dilated convolution) as demonstrated in Figure 4.3. In Section 4.8, we verify the effectiveness of the proposed method on two tasks: indoor semantic segmentation and hand segmentation for hand-object interaction. We use publicly available NYUDv2 dataset [SHKF12] for indoor semantic segmentation and collect a new challenging dataset including hand-object interaction for hand segmentation.
4.2 Related Works

Most researches of deep neural networks using depth maps treated a raw depth map as an image equivalent. For instance, a raw depth map was given as a direct input to the networks in hand pose estimation [TSLP14, GLYT16, SCR16], human pose estimation [WZC+16], and fingerspelling recognition [KTN15].

Alternatively, Gupta et al. proposed the geocentric embedding for a depth map to learn better representations in convolutional neural networks [GGA+14]. Specifically, the geocentric embedding encodes horizontal disparity, height above ground, and angle with gravity (HHA) for each pixel. The work showed that using the HHA encoded images, convolutional neural networks can learn better features for object detection and segmentation.
Figure 4.3: An example of applying the different sizes (sparsities) of the receptive field at each spatial location. Suppose the indices of the matrix start from the top-left corner with \((1, 1)\), and the kernel size is \(3 \times 3\). The figure shows the cases of \(S_{5,3}^\ell = 1\), \(S_{8,5}^\ell = 2\), and \(S_{16,5}^\ell = 3\).

The networks we introduce are distinct from the works in [TSLP14, GLYT16, SCR16, WZC+16, KTN15, GGA+14]. First, our proposed method utilizes depth information in convolution layers rather than converting a raw depth map into a better representation in a preprocessing stage. In other words, our method does not require any additional preprocessing to manipulate the raw data. Second, our proposed method can take any type of input (e.g. color image, depth map, etc.) to learn feature representations by giving the corresponding depth information as shown in Figure 4.4.

**Convolution layer.** Conventionally, most convolutional neural networks used typical,
dense, and fixed convolution [KSH12a, SZ15]. Recently, dilated (atrous) convolution was applied for semantic segmentation to extract sparse features in higher resolution [CPK+15]. The structure excluded pooling layers (which cause the decrement of spatial resolution) and replaced typical convolutions following the pooling layers by dilated convolutions. The dilated convolution was employed to increase the size of receptive fields and compensate the exclusion of pooling layers [CPK+15, CPK+18, YK16]. The dilated convolution in these methods has different sparsity at each layer depending on the excluded pooling layers while it has the same sparsity for all spatial locations and all feature representations in a layer.

Contemporarily, active convolution and deformable convolution are presented in [JK17, DQX+17]. The goal of both methods is to learn the shape of convolutions using a training dataset. Active convolution defines the learnable position parameters to represent various forms of the receptive fields in the task of image classification [JK17]. The position parameters are shared across all kernels in a layer. Thus, the learned receptive field is the same at all spatial locations and for all feature representations. Deformable convolution uses the offset field similar to the position parameters [DQX+17]. The offset field is computed using the input feature map and has different receptive fields at each spatial location. This deformable convolution was tested on semantic segmentation task and object detection task.

In the proposed networks, we apply dilated (sparse) convolution [SN96] to adjust the size of receptive fields for two purposes. First, we adapt the sparsities in convolutions to learn/extract near depth-invariant representations using distance information. Thus, the sparsity is adjusted at each spatial location depending on the distance at the location. Second, we adapt the sparsity at each feature representation to learn variously scaled representations. That is, the proposed dilated convolution generates different sparsities at each spatial location and at each feature representation.

In summary, the contributions of our work are as follows:

- We develop the depth-adaptive neural networks using the DaM convolution. The DaM
convolution consists of the adaptive perception neurons and the in-layer multiscale neurons.

- We propose the adaptive perception neuron. The neuron learns/extracts depth-adaptive representations.
- We propose the in-layer multiscale neuron. The neuron learns/extracts variously scaled representations in a convolution layer.
- We verify the effectiveness of the proposed networks on the task of semantic segmentation.

The goal of this work is to learn depth-invariant representations in deep neural networks using depth information. To achieve this goal, we propose the novel DaM convolution layer conceiving the adaptive perception neurons and the in-layer multiscale neurons as described in Figure 4.4. The adaptive perception neuron is proposed to adjust the receptive field using the depth information at each spatial location. The in-layer multiscale neuron is designed to learn features in different scales at each feature space (or channel) in a layer.

In Section 4.3, we introduce key notations for networks. We provide the detailed explanation of the DaM convolution consisting of the adaptive perception neuron and the in-layer multiscale neuron in Section 4.4. The overall architecture of the proposed neural network is developed in Section 4.5. In Section 4.6, the details of the training procedure are derived for the proposed networks. Finally, we provide the mathematical proof of the depth invariant property of the proposed networks in Section 4.7.

### 4.3 Notation

Let \( X^\ell \in \mathbb{R}^{c \times h^\ell \times w^\ell} \) and \( X^{\ell+1} \in \mathbb{R}^{c^{\ell+1} \times h^{\ell+1} \times w^{\ell+1}} \) be the matrices representing an input and an output of a certain layer \( \ell \) (either convolution, pooling, softmax, or loss layer), where \( c, h, \) and \( w \) denote the number of feature spaces (channels), height, and width, respectively. Also, let \( D^\ell \in \mathbb{R}^{h^\ell \times w^\ell} \) be the (pooled) depth map in the convolution layer \( \ell \) whose spatial resolution
Figure 4.5: Notations and the adaptive perception neuron. This neuron adjusts the size of the receptive field based on the depth information at each spatial location.

corresponds to the spatial resolution of the input $X^{\ell}$ (see Figures 4.4 and 4.5). The size of $D^{\ell}$ is determined by pooling, convolution, and padding in the previous layers.

The output $X^{\ell+1}$ of the convolution layer $\ell$ is computed by convolving the input $X^{\ell}$ with a shared weight matrix $W^{\ell} \in \mathbb{R}^{c^{\ell+1} \times c^{\ell} \times k_{h}^{\ell} \times k_{w}^{\ell}}$ and by adding a bias vector $b^{\ell} \in \mathbb{R}^{c^{\ell+1}}$, where $k_{h}^{\ell}$ and $k_{w}^{\ell}$ denote the dimensions of kernels along the height and width directions. In a typical convolution layer, the output $X_{t,m,n}^{\ell+1}$ of the $t$-th output feature space at the spatial location $(m,n)$ is computed as

$$X_{t,m,n}^{\ell+1} = f \left( \sum_{r} \sum_{u} \sum_{v} W_{t,r,u,v}^{\ell} X_{r,m+u,n+v}^{\ell} + b_{t}^{\ell} \right),$$

(4.1)

where $r \in [0,c^{\ell}-1]$ and $t \in [0,c^{\ell+1}-1]$ are the indices for the feature spaces of the input and the output, respectively, $u \in [-\lfloor k_{h}/2 \rfloor, -\lfloor k_{h}/2 \rfloor + k_{h} - 1]$ and $v \in [-\lfloor k_{w}/2 \rfloor, -\lfloor k_{w}/2 \rfloor + k_{w} - 1]$ are the indices for the weight matrix $W^{\ell}$ along the height and width direction, and $f(\cdot)$ is a transfer
function (e.g. rectified linear unit (ReLu), etc.).

4.4 Depth-adaptive Multiscale Convolution Layer

As observed in Figure 4.1, an object appears to have different sizes in the image plane depending on its distance from the camera. The generalization performance of the trained networks using these depth-variant features may not be sufficiently good because learning a common representation is challenging from the features. As such, it is necessary to learn depth-invariant features for neural networks in order to achieve better generalization performance. To this end, we propose the DaM convolution layer containing the adaptive perception neurons and the in-layer multiscale neurons. The adaptive perception neuron in Section 4.4.1 adjusts its receptive field to offset the change of the spatial size of objects on captured images. The receptive field adjusted by the adaptive perception neuron is clearly sub-optimal because the ideal correlation between the size and the distance varies over objects (e.g. due to different sizes). Hence, we develop the in-layer multiscale neuron in Section 4.4.2 that effectively controls the size of receptive fields over individual objects. The in-layer multiscale neuron extracts the diversely scaled depth-invariant features by tuning a parameter that determines sparsity at each feature representation.

Given a depth map as an input of the networks, unlike color images, the intensity (value) of an object on the depth map is scaled by the distance from the camera. This implies that the networks may learn intensity-variant features for the same object. To avoid this misleading, we propose to employ depth difference (relative depth) as an input for the feature extraction in Section 4.4.3.
4.4.1 Adaptive Perception Neuron

The proposed adaptive perception neuron determines its size of receptive field based on the depth information at each spatial location while other methods [CPK⁺15, CPK⁺18, YK16] used the predetermined receptive field in a convolution layer. Thus, the proposed networks having such adaptive perception neurons can apply different receptive field at each spatial location. Specifically, we increase the receptive field for objects at a close distance and decrease it for objects at a long distance to compensate for the variation of objects’ size on the captured images.

To determine the receptive field of each neuron, the depth map $D^\ell$ is fed to the adaptive perception neuron. The size of the receptive field $S^\ell_{r,m,n} \in \mathbb{R}^{c^\ell \times h^\ell \times w^\ell}$ at a spatial location $(m,n)$ inversely increases to the depth from the camera $D^\ell_{m,n}$, as follows:

$$S^\ell_{r,m,n} \propto \frac{1}{D^\ell_{m,n}}. \quad (4.2)$$

Applying the $S^\ell$ for the convolution layer $\ell$, the adaptive perception neuron takes different entries of the input $X^\ell$ connected by $S^\ell_{r,m,n}$ as demonstrated in Figures 4.3 and 4.5. Thus, the output in (4.1) is replaced by:

$$X^{\ell+1}_{i,m,n} = f \left( \sum_r \sum_u \sum_v W^{\ell}_{i,r,u,v} X^\ell_{r,m+n, u+n+S^\ell_{r,m,n} v+b^\ell_{r,m,n}} \right). \quad (4.3)$$

4.4.2 In-layer Multiscale Neuron

Conventionally, learning/extracting features in various scales is advantageous in achieving higher segmentation accuracy by learning variant features. To learn features in multiple scales, the neural networks comprised of multiple neural networks were proposed in [TSLP14], known as the multiscale neural networks. In this type of neural networks, each constituting neural network takes an input in different resolution and learns features in various scales. However, these networks are structurally complex and require higher computational complexity. Thus, we propose the
in-layer multiscale neuron that takes only an input and learns features with multiple scales in
a network (see Figure 4.6). The proposed in-layer multiscale neuron learns features at various
scales by having a different parameter for the sparsity at each feature representation (channel).

The in-layer multiscale neuron determines sparsity at each feature space \( r \) using the
multiscale parameter \( p^r_\ell \), whereas the adaptive perception neuron in the previous section spatially
determines the sparsity depending on the depth \( D^\ell_{m,n} \). The parameter \( p^r_\ell \) is determined as follows:

\[
p^r_\ell = \frac{s^r_\ell}{\prod_{\ell' \in \mathcal{L}} z'^\ell} \left[ \frac{1}{|T|h^\ell w^\ell} \sum_{d \in T} \sum_m \sum_n D^d_{m,n} \right] \cdot q^\ell \tag{4.4}
\]

where \( s^r_\ell \) is the scaling factor for each feature space (channel) \( r \), \( z'^\ell \) is the stride of pooling layers
\( \ell' \in \mathcal{L} \) up to the current layer, \( |T| \) represents the number of data in the training dataset \( T \), and \( q^\ell \)
is the dilation parameter from the ancestor architecture.

The \( p^r_\ell \) is interpreted as three factors: one is the scaling factor \( s^r_\ell \) with the mean \( \frac{1}{|T|h^\ell w^\ell} \sum_{d \in T} \sum_m \sum_n D^d_{m,n} \) of the depth maps in the training dataset, another is the factor \( 1/z'^\ell \) regarding
pooling layers, and the other is the dilation parameter \( q^\ell \) from the ancestor architecture. The
scaling factor \( s^r_\ell \) with the mean of the depth determines different sparsities at each feature space
considering the mean of the depth. Precise parameters for \( s^r_\ell \) is explained in Section 4.8. The
term \( 1/z'^\ell \) compensates for the decrement of the spatial resolution of the feature map, caused by
pooling layers. That is, the size of the receptive field is decreased as pooling layer reduces the
spatial resolution. The term \( q^\ell \) is to retain the dilation parameter from the ancestor architecture.

Finally, the size of receptive field is determined by incorporating adaptive perception
neuron and in-layer multiscale neuron. The size \( S^\ell_{r,m,n} \) at a feature space \( r \) and a spatial location
\( (m,n) \) is as follows:

\[
S^\ell_{r,m,n} = \frac{p^r_\ell}{D^\ell_{m,n}}, \tag{4.5}
\]

where denominator is contributed by the adaptive perception neuron, and numerator is from the
in-layer multiscale neuron.
4.4.3 Depth Difference

In practice, values on a depth map vary as the distance from the camera changes. For instance, objects at different distances are represented by different intensity levels. However, the relative distance between these objects is constant regardless of their distance from the camera \([SFC^{+}11, \, SGF^{+}13, \, KTJ^{+}17]\). Consequently, we instead use the relative depth to measure distance-independent depth in the first convolution layer. The relative depth is computed as the difference between the depth at the receptive field and the depth at the center location of the receptive field. Replacing a depth by the relative depth, (4.3) is rewritten as

\[
X_{i,m,n}^2 = f\left(\sum_r \sum_u \sum_v W_{i,r,u,v} X_{r,m+n+u, n+v}^1 - X_{r,m,n}^1 + b_i^1\right). \tag{4.6}
\]

Although the input \(X^1\) to the networks is replaced by the relative depth \(X_{r,m+n+u, n+v}^1 - X_{r,m,n}^1\), the size \(S^1\) of the receptive fields is computed using the raw depth map \(D^1\).
4.5 Architecture

The proposed DaM convolution layer is applied to all convolution layers in two fully convolutional neural networks (Frontend module [YK16] and DeepLab [CPK+18]). All original convolution layers are replaced by the proposed layers to achieve depth-invariance as demonstrated in Section 4.7. Frontend module and DeepLab are selected as our baseline model since they are two of the state-of-the-art methods. For DeepLab, we employed the VGG-16 [SZ15] network-based architecture with large atrous spatial pyramid pooling (ASPP-L) and without conditional random field (CRF) [CPK+18].

We train the proposed neural networks by back-propagating the multinomial logistic loss $e_a$ while penalizing the increment of weights using the $L_2$ regularization (denote as $e_b$) [Bis06]. Thus, the total loss $e$ is the weighted sum of $e_a$ and $e_b$ (i.e. $e = e_a + \lambda e_b$), where $\lambda$ is the decay factor. To compute the multinomial logistic loss $e_a$, we apply the softmax function that transfers the input $X^\ell \in \mathbb{R}^{c_t \times h^\ell \times w^\ell}$ from the last convolution layer to the output $X^{\ell+1} \in \mathbb{R}^{c_t \times h^{\ell+1} \times w^{\ell+1}}$, where $c_t$ denotes the total number of classes. In softmax layer, the spatial resolutions of the input $X^\ell$ and the output $X^{\ell+1}$ are equivalent (i.e. $(h^{\ell+1}, w^{\ell+1}) = (h^\ell, w^\ell)$). The softmax output of the $r$-th feature space at the spatial location $(m,n)$ is defined as

$$X^{\ell+1}_{r,m,n} = \frac{\exp(X^\ell_{r,m,n})}{\sum_r \exp(X^\ell_{r,m,n})}, \quad (4.7)$$

The output $X^{\ell+1}_{r,m,n}$ is equivalent to the predicted probability of being the class $r$ at the spatial location $(m,n)$. Then, the multinomial logistic loss $e_a$ is the weighted sum over the logistic outputs of $X^{\ell+1}$:

$$e_a = -\frac{1}{h^\ell w^\ell} \sum_r \sum_m \sum_n \mathbb{1}(r = L_{m,n}) \log(X^{\ell+1}_{r,m,n}), \quad (4.8)$$

where $\mathbb{1}(\cdot)$ is an indicator function and $L \in \mathbb{R}^{h^{\ell+1} \times w^{\ell+1}}$ is a target class label matrix.
4.6 Back Propagation

To train the proposed networks, the loss $e$ is propagated backward and used to update the weights. The weights are updated by minimizing $e$ using the gradient $\frac{\partial e}{\partial W^\ell}$, where the gradient $\frac{\partial e}{\partial X^\ell}$ is required to back-propagate to the lower layer. Considering the total loss $e$ is the sum of the multinomial logistic loss $e_a$ and the regularization loss $e_b$, the gradient of $e$ with respect to $W^\ell$ is represented as

$$\frac{\partial e}{\partial W^\ell} = \frac{\partial e_a}{\partial W^\ell} + \frac{\partial e_b}{\partial W^\ell}, \quad (4.9)$$

and this is rewritten by the chain rule [GBC16, Bis06, Apo74], as follows:

$$\frac{\partial e}{\partial W^\ell} = \frac{\partial e_a}{\partial X^{\ell+1}} \frac{\partial X^{\ell+1}}{\partial W^\ell} + \frac{\partial e_b}{\partial W^\ell}. \quad (4.10)$$

For the shared weight $W^\ell_{t,r,u,v}$, the gradient of (4.10) is expanded as

$$\frac{\partial e}{\partial W^\ell_{t,r,u,v}} = \frac{\partial e_a}{\partial W^\ell_{t,r,u,v}} + \frac{\partial e_b}{\partial W^\ell_{t,r,u,v}} = \sum_m \sum_n \frac{\partial e_a}{\partial X^{\ell+1}_{t,m,n}} \frac{\partial X^{\ell+1}_{t,m,n}}{\partial W^\ell_{t,r,u,v}} + \lambda W^\ell_{t,r,u,v}. \quad (4.11)$$

Recalling (4.3), since an output node has the input nodes determined by the depth-adaptive receptive field, $S$ is required to decode the connections from input nodes to output nodes (see Figure 4.3). Considering this variation of receptive field, the second factor of the multinomial logistic loss $e_a$ is evaluated as

$$\frac{\partial X^{\ell+1}_{t,m,n}}{\partial W^\ell_{t,r,u,v}} = X^\ell_{r,m+S_{r,m,n}u,n+S_{r,m,n}v}. \quad (4.12)$$

To compute the first factor of $e_a$, let’s first consider a specific connection between the input node $(r, m + S^\ell_{r,m,n}u, n + S^\ell_{r,m,n}v)$ and the output node $(t, m, n)$. The gradient of this specific connection
is back-propagated as follows:

\[
\frac{\partial e_a}{\partial X^\ell_{r,m+n+S_{r,m,n}u,n+S_{r,m,n}v}} = \frac{\partial e_a}{\partial X_{t,m,n}^\ell} \frac{\partial X_{t,m,n}^{\ell+1}}{\partial X_{t,m,n}^\ell} = \frac{\partial e_a}{\partial X_{t,m,n}^\ell} W_{t,r,u,v}^\ell, \tag{4.13}
\]

In (4.13), the output node \(X_{t,m,n}^{\ell+1}\) is influenced by the multiple input nodes, then the gradient \(\partial e_a/\partial X^\ell\) is computed by the iterative accumulations over the feature spaces and the spatial locations, as summarized in Algorithm 1.

Algorithm 1 Gradient of loss with respect to input

\begin{itemize}
  \item \textbf{Input:} \(\partial e_a/\partial X^{\ell+1}, W^\ell, S^\ell\)
  \item \textbf{Output:} \(\partial e_a/\partial X^\ell\)
\end{itemize}

initialize \(\partial e_a/\partial X^\ell = 0\)

\begin{algorithmic}
  \For {all \(t, r, m, n, u, v\)}
  \State \(\frac{\partial e_a}{\partial X^\ell_{r,m+n+S_{r,m,n}u,n+S_{r,m,n}v}} = \frac{\partial e_a}{\partial X_{t,m,n}^\ell} W_{t,r,u,v}^\ell\)
  \EndFor
\end{algorithmic}

Finally, the weight matrix \(W^\ell\) is updated using the stochastic gradient descent algorithm with momentum [GBC16] because we use small batch of training data to compute the gradients. At an iteration \(i\), suppose the current weight matrix is denoted as \(W^{\ell,i}\), then, the weight matrix \(W^{\ell,i+1}\) at the iteration \(i+1\) is updated considering the previous update and the computed gradient as follows:

\[
W^{\ell,i+1} = W^{\ell,i} + \mu(W^{\ell,i} - W^{\ell,i-1}) - \gamma(\partial e/\partial W^{\ell,i}) \tag{4.14}
\]

where \(\mu\) and \(\gamma\) denote the momentum and the learning rate, respectively. The momentum \(\mu\) was chosen as 0.99 for Frontend module and 0.9 for DeepLab, and the learning rate is explained in Section 4.8.
4.7 Proof of Depth-Invariance

In this section, we present the mathematical proof of the depth-invariance property of the proposed networks. We first simplify the convolution in (4.3) by considering a single channel one-dimensional input and output. We, then, apply the proposed convolution to an input at different distances from the camera. By demonstrating that the outputs are equivalent regardless of the distances, we prove that the proposed DaM convolution is depth-invariant.

Considering a neural network having a single channel (feature space), (4.3) is substituted as follows:

\[ X_{m,n}^{\ell+1} = f\left( \sum_u \sum_v W_{u,v} X_{m,S_{u,n},n+1}^{\ell+1} + b^{\ell} \right). \]  

(4.15)

For the one-dimensional input, (4.15) is further simplified as

\[ x_{m}^{\ell+1} = f\left( \sum_u W_{u,m,Sm} x_{m+S_{m,u}}^{\ell} + b^{\ell} \right) \]  

(4.16)
Let’s first consider the example in Figure 4.7, showing the proposed convolution layers for the input at distance \(d\) in Figure 4.7(a) and at distance \(d/2\) in Figure 4.7(b). In the example, the size of kernel is set to 3, and the size \(s\) of receptive field is 1 at distance \(d\). Then, the output \(x^2_5\) of the first convolution layer in Figure 4.7(a) is

\[
x^2_5 = f \left( \sum_{u^1 = -1}^{1} w^1_{u^1} \hat{x}^1_{5+u^1} + b^1 \right) = f \left( w^1_{-1} x^1_4 + w^1_0 x^1_5 + w^1_1 x^1_6 + b^1 \right),
\]

(4.17)

and the output \(x^3_5\) of the second convolution layer is

\[
x^3_5 = f \left( w^2_{-1} x^2_4 + w^2_0 x^2_5 + w^2_1 x^2_6 + b^2 \right)
\]

\[
= f \left( w^2_{-1} \cdot f (w^1_{-1} x^1_3 + w^1_0 x^1_4 + w^1_1 x^1_5 + b^1) + w^2_0 \cdot f (w^1_{-1} x^1_4 + w^1_0 x^1_5 + w^1_1 x^1_6 + b^1) \right. \\
\left. + w^2_1 \cdot f (w^1_{-1} x^1_5 + w^1_0 x^6 + w^1_1 x^1_7 + b^1) + b^2 \right).
\]

(4.18)

In Figure 4.7(b), the distance from the camera decreases to \(d/2\), thus the size of the object on an image plane is doubled comparing to the size at \(d\) (see Figure 4.1). Let \(\hat{x}\) denote the input at distance \(d/2\) and suppose \(\hat{x}_5\) corresponds to \(x_5\). Then, \(\hat{x}_{5+2v}\) is equivalent to \(x_{5+v}\) for \(\forall v \in \mathbb{Z}\) (e.g. \(x_6 = \hat{x}_7\) for \(v = 1\)). Since its receptive field increases by a factor of 2 by the relation of (4.5), the output \(\hat{x}^2_5\) of the first convolution layer is consequently equivalent to \(x^2_5\):

\[
\hat{x}^2_5 = f \left( \sum_{u^1 = -1}^{1} w^1_{u^1} \hat{x}^1_{5+2u^1} + b^1 \right) = f \left( w^1_{-1} \hat{x}^1_3 + w^1_0 \hat{x}^1_5 + w^1_1 \hat{x}^1_7 + b^1 \right)
\]

\[
= f \left( w^1_{-1} x^1_4 + w^1_0 x^1_5 + w^1_1 x^1_6 + b^1 \right)
\]

(4.19)

= \(x^2_5\).
and the output \( \hat{x}_3^3 \) of the second convolution layer is

\[
\hat{x}_3^3 = f\left( w_{-1}^2 \hat{x}_3^2 + w_0^2 \hat{x}_5^2 + w_1^2 \hat{x}_7^2 + b^2 \right)
\]

\[
= f\left( w_{-1}^2 \cdot f( w_{-1}^1 \hat{x}_1^1 + w_0^1 \hat{x}_3^1 + w_1^1 \hat{x}_5^1 + b^1 ) + w_0^2 \cdot f( w_{-1}^1 \hat{x}_3^1 + w_0^1 \hat{x}_5^1 + w_1^1 \hat{x}_7^1 + b^1 ) + w_1^2 \cdot f( w_{-1}^1 \hat{x}_5^1 + w_0^1 \hat{x}_7^1 + w_1^1 \hat{x}_9^1 + b^1 ) + b^2 \right)
\]

\[
= f\left( w_{-1}^1 \cdot f( w_{-1}^1 \hat{x}_3^1 + w_0^1 \hat{x}_4^1 + w_1^1 \hat{x}_5^1 + b^1 ) + w_0^2 \cdot f( w_{-1}^1 \hat{x}_4^1 + w_0^1 \hat{x}_5^1 + w_1^1 \hat{x}_6^1 + b^1 ) + w_1^2 \cdot f( w_{-1}^1 \hat{x}_5^1 + w_0^1 \hat{x}_6^1 + w_1^1 \hat{x}_7^1 + b^1 ) + b^2 \right)
\]

\[
= x_3^3.
\]

(4.20)

We conclude from this simple example that the proposed convolution extracts depth-invariant activations.

From the fact that \( \hat{x}_{m+n}^\ell \cdot \hat{x}_{m+n}^\ell \) is equivalent to \( x_{m+n}^\ell \), the demonstration of depth-invariant activations is generalized as

\[
\tilde{X}_{r',m,n}^{\ell+1} = f\left( \sum_{r'} \sum_{u'} \sum_{v'} W_{r',r',u',v'}^\ell \tilde{X}_{r',m+n,u',v'}^{\ell} + b^\ell \right)
\]

(4.21)

where \( g \) is the ratio of distances between \( \hat{x} \) and \( x \). From the example of (4.19), (4.20) and the generalization of (4.21), we conclude that the proposed convolution extracts depth-invariant activations by adjusting the size of receptive field.
4.8 Experiments and Results

The proposed neural networks were tested on two applications: indoor semantic segmentation and hand segmentation for hand-object interaction. The experimental results verify that the proposed neural networks outperform original Frontend module [YK16] and DeepLab [CPK18] without any additional layer or pre/post-processing.

For comparison, we report pixel-wise accuracy, mean accuracy, mean intersection over union (IoU), and frequency weighted (FW) IoU for both experiments. Additionally, for hand segmentation, we report precision, recall, and $F_1$ score where they are explained in Section 3.5. Let $n_{ij}$ be the number of pixels which belong to the class $i$ and are predicted to the class $j$, and $c_t$ be the total number of classes.

\[
\text{Pixel accuracy} = \frac{\sum_i n_{ii}}{\sum_i \sum_j n_{ij}},
\]

\[
\text{Mean accuracy} = \frac{1}{c_t} \sum_i \left( \frac{n_{ii}}{\sum_j n_{ij}} \right),
\]

\[
\text{Mean IoU} = \frac{1}{c_t} \sum_i \left( \frac{n_{ii}}{\sum_j n_{ij} + \sum_j n_{ji} - n_{ii}} \right),
\]

\[
\text{FW IoU} = \frac{1}{\sum_i \sum_j n_{ij}} \sum_i \left( \frac{\sum_j n_{ij} n_{ii}}{\sum_j n_{ij} + \sum_j n_{ji} - n_{ii}} \right) .
\]  

(4.22)

4.8.1 Semantic Segmentation

Dataset

The NYUDv2 dataset consists of 1,449 pairs of RGB-D images including various indoor scenes with pixel-wise annotations [SHKF12]. The pixel-wise annotations were coalesced into 40 dominant object categories by Gupta et al. [GAM13]. We experimented with this 40 classes problem with the standard separation [SHKF12, GAM13] of 795 training images and 654 testing
images.

Experiments

All the models were initialized using the VGG-16 model [SZ15] trained using the ImageNet ILSVRC-2014 dataset [RDS+15] except for the input of RGB-HHA. Then, the models were fine-tuned using the NYUDv2 training dataset [SHKF12]. For the input of RGB-HHA, we initialized the model using the two fine-tuned models using NYUDv2 dataset (one model using RGB images and the other model using HHA images). Then, we fine-tuned the model using the pair of RGB images and HHA images similar to [LSD15, SLD17]. The initial base learning rate was selected by trying several learning rates ($\gamma$) with a factor of 10 such as $[10^{-9}, 10^{-10}, 10^{-11},...]$. The decay factor ($\lambda$) of the weight matrix is chosen as 0.0005. The models used in the experiments were selected based on the mean IoU score. During training, we computed the mean IoU score at every 1,000 iterations for the input of RGB-HHA and at every 2,000 iterations for the other inputs.

For Frontend module, we used the multinomial logistic loss without normalization during training. So, the normalization term $1/h^\ell w^\ell$ was removed from (4.8). The initial base learning rate was selected as $10^{-12}$ for the input of RGB-HHA and $10^{-10}$ for the other inputs. The scaling parameter $s^1_r$ for the first layer was set to $\{1, 1.5, 2\}$ and $s_r$ for other layers was set to $\{0.5, 0.75, 1.0, 1.25\}$ for color images and depth maps and $\{0.75, 1.0, 1.25, 1.5\}$ for HHA images. If the mean IoU score stops improving, the base learning rate was decreased by a factor of 10. The training was terminated if the improvement of the score is negligible ($< 0.001$) or the score is not improved.

For DeepLab, the initial base learning rate was selected as $10^{-4}$ for HHA images and $10^{-3}$ for other inputs. The learning rate was decreased using polynomial decay with the power of 0.9 and the maximum iteration of 40,000. The scaling parameters $s_r$ for all layers and for all inputs were set to be linearly distributed in $[0, 1.5]$.
Results

We adopted the experimental settings in [LSD15, SLD17]. We considered the inputs of an RGB image, the concatenated image of an RGB image and a depth map (early fusion), and an HHA encoded image [GGA+14]. We also experimented with combining the scores from an RGB image and from an HHA encoded image [GGA+14] at the last layer (late fusion). Table 4.1 and Fig. 4.8 show the quantitative results and the qualitative results. The proposed method achieves the improvements without any additional layers or pre/post-processing.

Analysis

We experimentally analyze the effects of multiscale parameters $s^l_r$ in Table 4.2. The analysis shows that the proposed method outperforms other methods using the parameters in the reasonable ranges. We also analyze the effects of applying the different number of the DaM convolution in Table 4.3. The experiments demonstrate that replacing all convolution layers outperforms other settings. The processing time is measured using a machine with Intel i7-4790K CPU and Nvidia Tesla K40c. Table 4.4 shows that multi/random scale evaluation has the chance of further improving the segmentation performance. In the multiscale evaluation, the final results are combined with the results of original, twice enlarged, and half-scaled inputs. In the random scale evaluation, the final results are fused from the results of original and two randomly scaled inputs. The results using both scaling are combined with the results of the previously mentioned five inputs. Table 4.5 demonstrates that simply increasing receptive fields in DeepLab does not improve the accuracy. Lastly, we show the convergence curve for Frontend module [YK16] and the network with the proposed DaM convolution in Figure 4.9. The average loss is computed using the losses from 100 iterations. The graph shows that the proposed method converges slightly faster than Frontend module.
Table 4.1: The quantitative results of the NYUDv2 dataset. The scores are scaled by a factor of 100. Bold face and blue color emphasize the best performance.

<table>
<thead>
<tr>
<th>Input</th>
<th>Architecture</th>
<th>Method</th>
<th>Pixel accu.</th>
<th>Mean accu.</th>
<th>FW IoU</th>
<th>Mean IoU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DaM conv.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gupta et al.</td>
<td>[GGA+ 14]</td>
<td></td>
<td>60.3</td>
<td>-</td>
<td>47.0</td>
<td>28.6</td>
</tr>
<tr>
<td>RGB</td>
<td></td>
<td>FCN-32s [LSD15]</td>
<td>-</td>
<td>60.0</td>
<td>42.2</td>
<td>43.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FCN-32s [SLD17]</td>
<td>-</td>
<td>61.8</td>
<td>44.7</td>
<td>46.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FCN-16s [SLD17]</td>
<td>-</td>
<td>62.3</td>
<td>45.1</td>
<td>46.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FCN-8s [SLD17]</td>
<td>-</td>
<td>62.1</td>
<td>46.1</td>
<td>47.2</td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td>Frontend [YK16]</td>
<td>62.1</td>
<td>45.8</td>
<td>46.6</td>
<td>32.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DeepLab [CPK+18]</td>
<td>63.8</td>
<td>46.2</td>
<td>48.3</td>
<td>33.3</td>
</tr>
<tr>
<td>RGB-D</td>
<td></td>
<td>FCN-32s [LSD15]</td>
<td>-</td>
<td>61.5</td>
<td>42.4</td>
<td>45.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FCN-32s [SLD17]</td>
<td>-</td>
<td>62.1</td>
<td>44.8</td>
<td>46.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FCN-16s [SLD17]</td>
<td>-</td>
<td>62.3</td>
<td>45.4</td>
<td>46.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FCN-8s [SLD17]</td>
<td>-</td>
<td>62.7</td>
<td>46.0</td>
<td>47.4</td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td>Frontend [YK16]</td>
<td>62.1</td>
<td>46.2</td>
<td>46.8</td>
<td>32.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DeepLab [CPK+18]</td>
<td>63.7</td>
<td>47.1</td>
<td>48.3</td>
<td>33.2</td>
</tr>
<tr>
<td>HHA [GGA+ 14]</td>
<td></td>
<td>FCN-32s [LSD15]</td>
<td>-</td>
<td>57.1</td>
<td>35.2</td>
<td>40.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FCN-32s [SLD17]</td>
<td>-</td>
<td>58.3</td>
<td>35.7</td>
<td>41.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FCN-16s [SLD17]</td>
<td>-</td>
<td>57.5</td>
<td>36.0</td>
<td>41.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FCN-8s [SLD17]</td>
<td>-</td>
<td>56.8</td>
<td>36.7</td>
<td>41.9</td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td>Frontend [YK16]</td>
<td>56.7</td>
<td>38.5</td>
<td>41.8</td>
<td>25.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DeepLab [CPK+18]</td>
<td>57.9</td>
<td>40.0</td>
<td>42.6</td>
<td>26.9</td>
</tr>
<tr>
<td>RGB-HHA</td>
<td></td>
<td>FCN-32s [LSD15]</td>
<td>-</td>
<td>64.3</td>
<td>44.9</td>
<td>48.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FCN-32s [SLD17]</td>
<td>-</td>
<td>65.3</td>
<td>44.0</td>
<td>48.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FCN-16s [LSD15]</td>
<td>-</td>
<td>65.4</td>
<td>46.1</td>
<td>49.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FCN-16s [SLD17]</td>
<td>-</td>
<td>67.0</td>
<td>47.2</td>
<td>51.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FCN-8s [SLD17]</td>
<td>-</td>
<td>66.8</td>
<td>47.8</td>
<td>51.4</td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td>Frontend [YK16]</td>
<td>66.6</td>
<td>48.1</td>
<td>51.0</td>
<td>36.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DeepLab [CPK+18]</td>
<td>66.9</td>
<td>49.6</td>
<td>51.5</td>
<td>37.0</td>
</tr>
</tbody>
</table>
Figure 4.8: The qualitative comparison of the result for the NYUDv2 dataset. The odd rows show the results of Frontend module [YK16], and the even rows show the results of the network with the proposed DaM convolution. The results and the ground truth labels are visualized on the color images for better visualization.

4.8.2 Hand Segmentation

Experiments

We experimented using the dataset in Section 2.4.1. All the models were initialized using the VGG-16 model [SZ15] that were trained using the ImageNet ILSVRC-2014 dataset [RDS+15].
Then, the initial models were fine-tuned using the HOI training dataset. The initial base learning rate $\gamma$ was selected by trying several learning rates with the factor of 10 such as $[10^{-3}, 10^{-4}, 10^{-5}, ...]$. In most cases, the initial learning rate was selected as $10^{-4}$. The decay factor $\lambda$ of the weight matrix is chosen as 0.0005.

The models used in the experiments were selected based on the $F_1$ score on the validation dataset. During training, we computed the $F_1$ score on the validation dataset at every 4,000 iterations. If the $F_1$ score stops improving, the base learning rate was decreased by a factor of 10. The training was terminated if the improvement of the score is negligible ($< 0.001$) or the score is not improved. The multiscale parameter $s_r$ was set to $[1, 1.5, 2]$ and $s_r$ for other layers was set to $[0.75, 1.0, 1.25, 1.5]$ for each quarter of the feature spaces in each convolution layer.

### Table 4.2: Ablation study of selecting multiscale parameter for Frontend module.

<table>
<thead>
<tr>
<th>Multiscale parameter $s_r$</th>
<th>Pixel accu.</th>
<th>Mean accu.</th>
<th>FW IoU</th>
<th>Mean IoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>First conv.</td>
<td>Other conv.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$[1, 1.25, 1.5]$</td>
<td>$[0.5, 0.75, 1, 1.25]$</td>
<td>63.6</td>
<td>46.9</td>
<td>48.1</td>
</tr>
<tr>
<td>$[1, 1.5, 2]$</td>
<td>$[0.25, 0.5, 0.75, 1]$</td>
<td><strong>63.7</strong></td>
<td>46.2</td>
<td>48.2</td>
</tr>
<tr>
<td>$[1, 1.5, 2]$</td>
<td>$[0.5, 0.75, 1, 1.25]$</td>
<td><strong>63.7</strong></td>
<td><strong>47.2</strong></td>
<td><strong>48.3</strong></td>
</tr>
<tr>
<td>$[1, 1.5, 2]$</td>
<td>$[0.75, 1, 1.25, 1.5]$</td>
<td>63.6</td>
<td>46.6</td>
<td>48.3</td>
</tr>
<tr>
<td>$[1, 1.5, 2]$</td>
<td>$[1, 1.25, 1.5, 1.75]$</td>
<td>63.5</td>
<td>46.4</td>
<td>48.0</td>
</tr>
<tr>
<td>$[1, 1.75, 2.5]$</td>
<td>$[0.5, 0.75, 1, 1.25]$</td>
<td>63.4</td>
<td>46.4</td>
<td>48.0</td>
</tr>
</tbody>
</table>

### Table 4.3: The effects of applying the DaM convolution to Frontend module for the input of RGB.

<table>
<thead>
<tr>
<th>DaM conv.</th>
<th>Pixel accu.</th>
<th>Mean accu.</th>
<th>FW IoU</th>
<th>Mean IoU</th>
<th>Processing time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>62.1</td>
<td>45.8</td>
<td>46.6</td>
<td>32.3</td>
<td>417</td>
</tr>
<tr>
<td>1st</td>
<td>63.4</td>
<td>46.7</td>
<td>48.0</td>
<td>32.9</td>
<td>467</td>
</tr>
<tr>
<td>1st, 3rd, 5th</td>
<td>63.5</td>
<td>47.0</td>
<td>48.2</td>
<td>32.9</td>
<td>470</td>
</tr>
<tr>
<td>All</td>
<td><strong>63.7</strong></td>
<td><strong>47.2</strong></td>
<td><strong>48.3</strong></td>
<td><strong>33.3</strong></td>
<td><strong>481</strong></td>
</tr>
</tbody>
</table>
Table 4.4: The analysis of multi/random scale evaluation using DeepLab-based networks for the input of RGB.

<table>
<thead>
<tr>
<th>Method</th>
<th>Scaling</th>
<th>Pixel accu.</th>
<th>Mean accu.</th>
<th>FW IoU</th>
<th>Mean IoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>DaM conv.</td>
<td>Multi</td>
<td>Random</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>63.8</td>
<td>46.2</td>
<td>48.3</td>
<td>33.7</td>
</tr>
<tr>
<td>✓</td>
<td>-</td>
<td>64.7</td>
<td>44.9</td>
<td>48.4</td>
<td>34.1</td>
</tr>
<tr>
<td>-</td>
<td>✓</td>
<td>64.1</td>
<td>45.4</td>
<td>48.2</td>
<td>33.6</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>64.6</td>
<td>45.0</td>
<td>48.4</td>
<td>33.9</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>65.1</td>
<td>47.3</td>
<td>49.0</td>
<td>34.3</td>
</tr>
</tbody>
</table>

Table 4.5: The effects of increasing receptive fields in DeepLab.

<table>
<thead>
<tr>
<th>Conv.</th>
<th>Pixel accu.</th>
<th>Mean accu.</th>
<th>FW IoU</th>
<th>Mean IoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>63.8</td>
<td>46.2</td>
<td>48.3</td>
<td>33.7</td>
</tr>
<tr>
<td>×2</td>
<td>62.0</td>
<td>42.8</td>
<td>46.4</td>
<td>31.1</td>
</tr>
<tr>
<td>×4</td>
<td>57.3</td>
<td>36.4</td>
<td>41.2</td>
<td>25.8</td>
</tr>
<tr>
<td>DaM conv</td>
<td>64.3</td>
<td>47.3</td>
<td>49.0</td>
<td>34.3</td>
</tr>
</tbody>
</table>

Results

The performance of the proposed methods and the comparing methods is tabulated in Table 4.6 for the inputs of the depth maps and the HHA encoded images [GGA+14]. The visual segmentation results are displayed in Figure 4.10. The proposed neural network improves about 14% (depth maps) and 3% (HHA) in $F_1$ score relative to the baseline Frontend model [YK16]. Moreover, the proposed network with the input of depth map achieves higher $F_1$ score and mean IoU than Frontend module with the input of the HHA encoded image. These results verify that the proposed networks improve segmentation performance without any additional layer or pre/post-processing.
Table 4.6: The quantitative results of the HOI dataset. The scores are scaled by a factor of 100. Bold face and blue color emphasize the best performance for each input and for entire cases, respectively.

<table>
<thead>
<tr>
<th>Input</th>
<th>Method</th>
<th>Precision</th>
<th>Recall</th>
<th>F&lt;sub&gt;1&lt;/sub&gt; score</th>
<th>Pixel accu.</th>
<th>Mean accu.</th>
<th>FW IoU</th>
<th>Mean IoU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth map</td>
<td>Frontend [YK16]</td>
<td>72.4</td>
<td>70.2</td>
<td>71.3</td>
<td>99.0</td>
<td>84.9</td>
<td>98.2</td>
<td>77.2</td>
</tr>
<tr>
<td></td>
<td>Frontend + DaM conv.</td>
<td><strong>79.7</strong></td>
<td><strong>82.5</strong></td>
<td><strong>81.1</strong></td>
<td><strong>99.3</strong></td>
<td><strong>91.1</strong></td>
<td><strong>98.7</strong></td>
<td><strong>83.8</strong></td>
</tr>
<tr>
<td>HHA</td>
<td>Frontend [YK16]</td>
<td>76.3</td>
<td><strong>85.8</strong></td>
<td>80.8</td>
<td>99.3</td>
<td><strong>92.7</strong></td>
<td>98.7</td>
<td>83.5</td>
</tr>
<tr>
<td>[GGA+14]</td>
<td>Frontend + DaM conv.</td>
<td><strong>83.6</strong></td>
<td>84.1</td>
<td><strong>83.9</strong></td>
<td><strong>99.4</strong></td>
<td>91.9</td>
<td><strong>98.9</strong></td>
<td><strong>85.8</strong></td>
</tr>
</tbody>
</table>

4.9 Summary

In this chapter, we presented the novel fully convolutional neural networks that adjust the receptive field using depth information to learn/extract depth-invariant feature representations. In the proposed neural networks, we introduced the DaM convolution layer consisting of the adaptive perception neuron and the in-layer multiscale neuron. The proposed neural networks were applied to indoor semantic segmentation and hand segmentation for hand-object interaction. The experimental results demonstrate that the proposed neural networks improve the accuracy of
Figure 4.10: The qualitative comparison of the result for the HOI dataset. (a) Ground truth labels. (b) Results of Frontend module [YK16] for the input of a depth map. (c) Results of the network with the DaM convolution for the input of a depth map. (d) Results of Frontend module [YK16] for the input of an HHA encoded image [GGA+14]. (e) Results of the network with the DaM convolution for the input of an HHA encoded image [GGA+14]. The results and the ground truth labels are visualized on the depth maps with different color channels for better visualization.

segmentation without any additional layers or pre/post-processing.

This chapter, in part, is a reprint of materials in the journal paper “Depth-Adaptive Deep Neural Network for Semantic Segmentation”, B. Kang, Y. Lee, and T. Q. Nguyen [KLN18]. The dissertation author is the primary investigator and author of this paper.
Chapter 5

Applications of Segmentation

5.1 Sign Language Fingerspelling Recognition

5.1.1 Introduction

Sign language recognition is important for natural and convenient communication between deaf community and hearing majority. Currently, most communications between two communities highly rely on human-based translation services. However, this is inconvenient and expensive as human expertise is involved. Therefore, automatic sign language recognition aims to understand the meaning of signs without the assistance from experts. Then it can be translated to sound or text based on end users’ needs. We believe that sign language recognition is important for providing equal opportunity to every person and improving public welfare.

Sign language recognition is still a challenging problem despite of many research efforts during the last few decades [SWP98, CHB11]. It requires the understanding of combination of multi-modal information such as hand pose and movement, facial expression, and human body posture. Also, sign language has at least thousands of words including very similar hand poses while gesture recognition generally includes a small set of well specified gestures. Moreover, even same signs have significantly different appearances for different signers and different viewpoints.
In this work, we focus on static fingerspelling in American Sign Language (ASL) which is a small, but important part of sign language recognition. This is a small set of sign languages as shown in Figure 5.1, but is used in many situations in conveying names, addresses, brands, and so on. Static fingerspelling is still challenging because of visually similar yet different signs. For example, some of the signs are only distinguished by the position of thumb. Also, a large variation occurs from different camera viewpoint and different signers.

Depth sensors enable us to capture additional information to improve accuracy and/or processing time. Also, with recent improvement of GPU, CNNs have been employed to many computer vision problems. Therefore, we take advantage of a depth sensor and convolutional neural networks to achieve a real-time and accurate sign language recognition system.

Although gesture recognition only considers well specified hand gestures, some approaches are related to sign language recognition. Nagi et al. proposed a gesture recognition system for human-robot interaction using CNNs [NDC+11]. Van den Bergh et al. proposed a hand gesture recognition system using Haar wavelets and database searching [dBG11]. The system extracts features using Haar wavelets and classifies input image by finding the nearest match in the database. Although both systems show good results, these methods consider only six gesture classes.

Different sign languages are used in different countries or regions. There have been efforts towards sign language recognition systems other than ASL as well. Pigou et al. proposed an Italian sign language recognition system using CNNs [PDKS15]. Although they reported 95.68% accuracy for 20 classes, they mentioned that users in test set can be in training set and/or validation set. Liwicki et al. described a British Sign Language recognition system that understands fingerspelled words from video [LE09]. The system first recognized letters using Histogram of Gradients (HOG) descriptors. Then that recognized words using Hidden Markov Models (HMM). That system is different from recognizing a single fingerspelling. The dataset in use corresponded to a single signer.
Figure 5.1: ASL fingerspelling alphabets and numbers. We follow the real demonstrations of formal signs on [Uni, Wik] to collect our dataset. The demonstration images are also available on our repository.

ASL sentence recognition and verification has also been explored. Zafrulla et al. proposed a system which recognizes a sentence of three to five words [ZBS+11]. The word should be one of 19 signs in their dictionary. They also used Hidden Markov Models on extracted features.

Our work belongs to the category of ASL fingerspelling recognition systems. Isaac et al. proposed an ASL fingerspelling recognition system based on neural networks applied to wavelet features [IF04]. They reported a recognition rate of 99%. However, they did not specify the size of the dataset and the number of different subjects. Later, Pugeault et al. proposed a real-time ASL fingerspelling recognition system using Gabor filters and random forest [PB11]. Their system recognizes 24 different ASL fingerspelling for alphabets. They collected dataset from five subjects and reported a recognition rate of 75% using both color and depth, 73% using only color, and 69% using only depth. Although Pugeault et al. reported that combination of
color and depth improves the recognition rate, we only use depth to achieve better consistency to illumination changes and skin pigment differences and to avoid calibration process for general users [PB11]. Kuznetsova et al. also proposed a real-time ASL fingerspelling recognition system using multi-layered random forest [KLTR13]. They reported and Dong et al. also analyzed that they achieved 87% accuracy for the subjects whom the system has been trained on and 57% accuracy for new subject [DLY15]. Very recently, Dong et al. proposed an ASL alphabet recognition system [DLY15]. They first localized hand joint positions using random forest and hierarchical mode-seeking method. Then the system recognized ASL signs by applying random forest classifier to joint angle vector. They reported 90% accuracy for the subjects whom the system has been trained on and 70% accuracy for new signers.

This work differs from previous works in several ways. First, to the best of our knowledge, ours is the first fingerspelling recognition system to classify total 31 alphabets and numbers compared with the state of the art approach to classify only 24 classes reported in the literature. Second, we extract features by fine-tuning convolutional neural network parameters which are pre-trained for image classification task using 1.28 millions of color images [RDS+15]. Moreover, it achieves both real-time and the state of the art accuracy across different users. Our contribution also includes providing publicly available dataset which is currently limited both in quantity and quality.

5.1.2 Dataset

We have collected 31,000 depth maps using a depth sensor, Creative Senz3D camera of the resolution of 320×240. The dataset consists of 1,000 images for each of the 31 different hand signs from five subjects. 31 hand signs include all the fingerspellings of both alphabets and numbers except J and Z which require temporal information for classification. Since (2/V) and (6/W) are differentiated based on context, we have only one class to represent both one alphabet and one number. Although some informal signs are clearer and easier to recognize, we follow
formal signs to avoid ambiguity between signers \[ \text{Wik} \]. To collect dataset from various viewpoint, the dataset is collected while subjects are moving their hand around both on image plane and along z-axis.

5.1.3 Hand Segmentation

We assume that the closest object from camera is the user’s hand. This assumption is valid in fingerspelling and most of gesture recognition tasks. In addition, we use a black wrist band to get depth voids around wrist, since depth sensor cannot capture depth from black objects well. Figure 5.2 shows one example of the captured depth image, where the hand is convincingly the closest object according to the captured depth map and there is a depth void around the hand. Hand segmentation thus ends up in finding the connected components from this closest region of the depth image. This strategy provides a very simple and effective real-time hand segmentation. Figure 5.3 shows segmented hand depth image samples for the 31 signs including alphabets and digits generated using this method. We find a bounding box of hand region and scale it to $227 \times 227$. Then we include redundant 14 or 15 pixels at each edge of the bounding box to make it
the same input size of $256 \times 256$ described by Krizhevsky et al. [KSH12b] and thus take the differences of different hand segmentation results into account.

5.1.4 Classification

Architecture

We use Caffe [JSD+14] implementation (CaffeNet) of the CNNs which is almost equivalent to AlexNet [KSH12b]. The architecture consists of five convolution layers, five max-pooling layers, and three fully connected layers. After each convolution layer or fully connected layer except the last one, rectified linear unit layer is followed. For details, we will upload the architecture and also readers can refer to CaffeNet/Caffe [JSD+14] and AlexNet [KSH12b].
Figure 5.4: The collected data for the same meaning. It shows the importance of consideration of different signers and viewpoint variation.

Feature extraction

We extract a 4096-dimensional feature (final fully connected layer feature) vector from each pre-processed depth image using the aforementioned architecture. First, we subtract the mean image from each of the sample training/validation/test image. Then the mean-subtracted image is forward-propagated to extract features.

Training

We train and test neural networks in five different operating modes. These five cases can be looked upon from different perspectives. One way to look at it is from the pre-training perspective and the second way is how we deal with the training/testing data separation for different subjects. In the former case, we categorize the operating modes into two categories, namely re-training and fine-tuning. For re-training, the model is re-trained from randomly generated weights using the collected fingerspelling data. In fine-tuning, we pre-train the CNNs using a large ILSVRC2012 classification dataset [RDS¹⁵]; then we fine-tune the network weights for fingerspelling classification with the same architecture except the last layer which is replaced by 31 output classes. From the subjects’ data separation perspective, in one case, we do not separate the subjects in training, validation, and testing and in the second scenario, we use data from different subjects for training, validation, and testing.
5.1.5 Experiments and Results

As mentioned in Section 5.1.4, we train and test for five experimental settings. The results are compared with other systems in Table ?? . Our system achieves 99.99% accuracy when training and validation data have samples corresponding to the test subject. In this experiment, 50%, 25%, and 25% of dataset is used for training, validation, and testing. We achieve 75.18% and 78.39% for regular training (re-training) and 83.58% and 85.49% for fine-tuning. This shows that fine-tuning outperforms re-training about 7~8% in this depth image dataset even though the nature of the pre-training dataset (ILSVRC2012) is different. For each case, the former represents the average result of training with three subjects, validation with one subject, and test with one subject. The latter represents the average result of training with four subjects and test with one subject. We considered all possible combinations of subjects for training, validation, and test and the final reported accuracy is the average of all. For the latter case, although we use the same training parameters (e.g. the number of iterations) for all combinations, the performance is improved about 2~3%. It shows that by increasing the number of subjects in training, the system’s performance for new subject has high possibility of improvement. The number of iterations in the training phase is fixed to 8000 and 4000 for re-training and fine-tuning respectively. Overall, our system achieves about 10~15% improvement compare to previous state of the art result even with more number of classes and subjects. Moreover, the processing time is about 3 ms for the prediction of a single image using Nvidia GeForce GTX Titan. Table 5.2 shows accuracy for each alphabet or number. It shows that even with more number of classes, the accuracy of each class is higher than or similar to previous state of the art result. Therefore, it is obvious we will achieve better result with the same number of classes of comparing with the other works. On the other hand, the table shows that (E,M,N,T) has low accuracy for both our method and others since the letters are only differentiated by thumb position.

Previous gesture or ASL fingerspelling recognition systems considered only six gestures or 24 signs respectively. We increase the number of signs to 31 to accommodate most of the
Table 5.1: Quantitative comparison of gesture and ASL fingerspelling recognition systems in accuracy. Test with different subject means that the signer in test set is excluded from training/validation set. The corresponding results are highlighted in light gray. (a/b/c)% represents the portion of dataset for (training/validation/test). (a/b/c) and (a/b) represent the number of subjects in (training/validation/test) and (training/test).

<table>
<thead>
<tr>
<th>Method</th>
<th>Class type</th>
<th># of class</th>
<th># of subj.</th>
<th>Test w/ diff.</th>
<th>Input</th>
<th>Accur.(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nagi et al. [NDC+11]</td>
<td>Gesture</td>
<td>6</td>
<td>-</td>
<td>No</td>
<td>Color</td>
<td>96</td>
</tr>
<tr>
<td>Van den Bergh et al. [DBG11]</td>
<td>Gesture</td>
<td>6</td>
<td>-</td>
<td>No</td>
<td>Color &amp; Depth</td>
<td>99.54</td>
</tr>
<tr>
<td>Isaacs et al. [IF04]</td>
<td>Alphabets</td>
<td>24</td>
<td>-</td>
<td>-</td>
<td>Color</td>
<td>99.9</td>
</tr>
<tr>
<td>Pugeault et al. [PB11]</td>
<td>Alphabets</td>
<td>24</td>
<td>5</td>
<td>-</td>
<td>Color</td>
<td>73</td>
</tr>
<tr>
<td>Pugeault et al. [PB11]</td>
<td>Alphabets</td>
<td>24</td>
<td>5</td>
<td>-</td>
<td>Depth</td>
<td>69</td>
</tr>
<tr>
<td>Pugeault et al. [PB11]</td>
<td>Alphabets</td>
<td>24</td>
<td>5</td>
<td>-</td>
<td>Color &amp; Depth</td>
<td>75</td>
</tr>
<tr>
<td>Kuznetsova et al. [KLTR13] (50/50)%</td>
<td>Alphabets</td>
<td>24</td>
<td>5</td>
<td>Yes</td>
<td>Depth</td>
<td>87</td>
</tr>
<tr>
<td>Kuznetsova et al. [KLTR13] (4/1)</td>
<td>Alphabets</td>
<td>24</td>
<td>5</td>
<td>Yes</td>
<td>Depth</td>
<td>57</td>
</tr>
<tr>
<td>Dong et al. [DLY15] (50/50)%</td>
<td>Alphabets</td>
<td>24</td>
<td>5</td>
<td>No</td>
<td>Depth</td>
<td>90</td>
</tr>
<tr>
<td>Dong et al. [DLY15] (4/1)</td>
<td>Alphabets</td>
<td>24</td>
<td>5</td>
<td>Yes</td>
<td>Depth</td>
<td>70</td>
</tr>
<tr>
<td>Ours (re-training) (50/25/25)%</td>
<td>Alph. &amp; Digit</td>
<td>31</td>
<td>5</td>
<td>No</td>
<td>Depth</td>
<td>99.99</td>
</tr>
<tr>
<td>Ours (re-training) (3/1/1)</td>
<td>Alph. &amp; Digit</td>
<td>31</td>
<td>5</td>
<td>Yes</td>
<td>Depth</td>
<td>75.18</td>
</tr>
<tr>
<td>Ours (re-training) (4/1)</td>
<td>Alph. &amp; Digit</td>
<td>31</td>
<td>5</td>
<td>Yes</td>
<td>Depth</td>
<td>78.39</td>
</tr>
<tr>
<td>Ours (fine-tuning) (3/1/1)</td>
<td>Alph. &amp; Digit</td>
<td>31</td>
<td>5</td>
<td>Yes</td>
<td>Depth</td>
<td>83.58</td>
</tr>
<tr>
<td>Ours (fine-tuning) (4/1)</td>
<td>Alph. &amp; Digit</td>
<td>31</td>
<td>5</td>
<td>Yes</td>
<td>Depth</td>
<td>85.49</td>
</tr>
</tbody>
</table>
Table 5.2: Quantitative comparison for each fingerspelling class. Accuracy less than 50% is highlighted in light gray. Bold entries correspond to accuracy higher than 90%.

<table>
<thead>
<tr>
<th>Method</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>K</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M</td>
<td>N</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
</tr>
<tr>
<td>Pugeault et al. [PB11]</td>
<td>75</td>
<td>83</td>
<td>57</td>
<td>37</td>
<td>63</td>
<td>35</td>
<td>60</td>
<td>80</td>
<td>73</td>
<td>43</td>
<td>87</td>
</tr>
<tr>
<td>Color &amp; Depth</td>
<td>17</td>
<td>23</td>
<td>13</td>
<td>57</td>
<td>77</td>
<td>63</td>
<td>17</td>
<td>7</td>
<td>67</td>
<td>87</td>
<td>53</td>
</tr>
<tr>
<td>Ours (re-training)</td>
<td>82.7</td>
<td>94.9</td>
<td>83.3</td>
<td>85.9</td>
<td>45.7</td>
<td>86.6</td>
<td>86.1</td>
<td>81.8</td>
<td>72.5</td>
<td>86.7</td>
<td>93.4</td>
</tr>
<tr>
<td>(3/1/1)</td>
<td>42.1</td>
<td>32.6</td>
<td>73.1</td>
<td>85.4</td>
<td>70.8</td>
<td>58.9</td>
<td>73.6</td>
<td>10.1</td>
<td>61.7</td>
<td>93.0</td>
<td>80.2</td>
</tr>
<tr>
<td>Ours (re-training)</td>
<td>85.8</td>
<td>95.4</td>
<td>91.7</td>
<td>91.4</td>
<td>43.0</td>
<td>83.6</td>
<td>79.9</td>
<td>81.5</td>
<td>78.7</td>
<td>94.4</td>
<td>98.4</td>
</tr>
<tr>
<td>(4/1)</td>
<td>46.8</td>
<td>28.4</td>
<td>74.9</td>
<td>79.0</td>
<td>70.2</td>
<td>69.4</td>
<td>73.2</td>
<td>4.8</td>
<td>70.5</td>
<td>98.1</td>
<td>87.6</td>
</tr>
<tr>
<td>Ours (fine-tuning)</td>
<td>84.2</td>
<td>94.3</td>
<td>86.9</td>
<td>89.7</td>
<td>87.1</td>
<td>92.3</td>
<td>88.0</td>
<td>85.2</td>
<td>90.5</td>
<td>83.0</td>
<td>99.7</td>
</tr>
<tr>
<td>(3/1/1)</td>
<td>62.2</td>
<td>59.0</td>
<td>69.4</td>
<td>82.9</td>
<td>82.6</td>
<td>80.9</td>
<td>57.6</td>
<td>55.3</td>
<td>92.2</td>
<td>94.5</td>
<td>83.7</td>
</tr>
<tr>
<td>Ours (fine-tuning)</td>
<td>89.5</td>
<td>97.1</td>
<td>93.2</td>
<td>88.4</td>
<td>85.7</td>
<td>93.8</td>
<td>84.9</td>
<td>86.1</td>
<td>95.6</td>
<td>90.0</td>
<td>99.4</td>
</tr>
<tr>
<td>(4/1)</td>
<td>59.9</td>
<td>65.1</td>
<td>69.2</td>
<td>80.2</td>
<td>85.7</td>
<td>85.2</td>
<td>53.7</td>
<td>60.6</td>
<td>96.1</td>
<td>98.2</td>
<td>87.0</td>
</tr>
<tr>
<td></td>
<td>75.3</td>
<td>81.9</td>
<td>99.0</td>
<td>82.1</td>
<td>92.1</td>
<td>97.5</td>
<td>95.9</td>
<td>82.5</td>
<td>99.3</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

signs. Also, some previous methods did not separate subjects in training, validation, and testing. Experimenting with training and testing dataset from different subjects is important because then only the system’s performance for a new subject can be measured. Therefore, we include experiments and report results where training and testing subjects are separated to demonstrate how the trained model can be used for anonymous subject. Lastly, although Pugeault et al. reported that combination of color and depth improves the recognition rate [PB11], we only use depth to achieve better consistency to illumination changes and skin pigment differences and to avoid calibration process for general users.
5.2 Hand Articulations Tracking

5.2.1 Introduction

Hands are used in daily lives to handle objects and to better communicate with others. Especially, hands are almost the only way to control with electronic devices except limited usage of speech. It is limited since speech is hard to protect privacy, and understanding of speech is difficult in noisy environment. Recent advancements in mobile devices and wearable devices demand better communication methods rather than touch screens which limit physical space. Due to the demand of more natural and convenient interacting methods, interaction using hand gestures has received lots of attention for human-computer interactions, virtual / augmented reality, and robot controls.

Previously, hand pose estimation methods are classified into single frame-based methods and model-based tracking methods \cite{EBN+07}. Single frame-based methods estimate hand pose by searching huge databases or by recovering hand pose from hand joint classification. Athitsos et al. and Wang et al. used a color image to retrieve hand pose from large databases \cite{AS03, WP09}. The method in \cite{WP09} used a color glove for better searching from a database. However, since the database has limited number of hand pose images, it can estimate only the poses in the database. Recently, Tang et al. and Tompson et al. estimated hand pose by applying hand joint classification using random forest and convolutional neural networks (CNNs) respectively \cite{TYK13, TSLP14}. Tang et al. proposed the semi-supervised transductive regression (STR) forest method with joint refinement procedure \cite{TYK13}. Tompson et al. employed CNNs and pose recovery to achieve continuous pose estimation \cite{TSLP14}. These methods require high performance GPUs to achieve real-time processing and also huge real and synthetic data for training.

Model-based tracking methods estimate hand pose by finding optimal parameters of computer-generated hand model using both current input image and previous results \cite{STTC06}. Rehg et al. and Oikonomidis et al. used multiple RGB cameras to reduce occlusions and to
increase visual features [RK94, OKA11c]. The method in [OKA11a] tracks a full DOF hand motion by minimizing discrepancy between input RGB-D image and computer-generated model using particle swarm optimization (PSO). Generating many possible hand pose images for each frame using computer graphics is computationally expensive and requires high performance GPUs to achieve about 15∼20 frames per second (FPS). Also, this method requires a user to place a hand on pre-determined position and pose to initialize tracking.

Recently, Sridhar et al. proposed the combined method of single frame-based method and model-based tracking method to blend advantages of each method [SOT13]. They used multiple color cameras for model-based tracking and a depth sensor to search their database, then a voting algorithm is applied to combine the results. Although multiple camera system helps to achieve better accuracy, it requires setup and calibration processes. Moreover, it requires GPUs to process multiple inputs at each frame. Qian et al. proposed another combined method using a depth sensor [QSW+14]. They combined an efficient initialization method and a tracking method using PSO and iterative closest point (ICP). Their hand model is designed using only spheres to simplify objective function. Sharp et al. also proposed the combined method of two-layer re-initialization using random forest and model fitting using PSO and genetic algorithm [SKR+15].

Most of existing hand tracking systems are rely on expensive, high power-consuming, and high performance GPUs since hand tracking is challenging because of complex articulations, self-occlusions, deformation, and rapid motions. Consequently, these systems are inappropriate for portable and wearable devices. On the other hand, hand tracking systems for those devices do not need to consider huge viewpoint changes since in general, user’s hand is relatively close to camera. In this work, we focus on efficient hand articulations tracking system that does not require GPUs and considers mainly when user’s hand is close to camera. Although we mainly consider small viewpoint changes, it is very challenging because of very limited computational power. Although we design and test our system without using GPUs, it can be implemented with inexpensive and low power-consuming GPUs for better accuracy while still being able to be
used for mobile and wearable devices. To the best of our knowledge, ours is the first system that can automatically adjust hand shapes which we call adaptive hand model while other methods manually and experimentally decided hand model size for each user. Moreover, unlike most of previous works, we focus on real-time system without using GPUs for mobile and wearable devices. We also propose hierarchical random sampling of pixels on each depth map to achieve better performance with limited computations. Lastly, we improve an efficient initialization method at each frame using finger detection and classification. Figure 5.5 shows the entire process of proposed method. We first process hand segmentation from depth map and choose partial pixels from hand region using hierarchical random sampling. On the other hand, we extract fingertip positions and finger joint rotations using an efficient finger detection and classification algorithm for initialization at each frame. Then hand pose and shape are estimated by minimizing discrepancy between the selected partial pixels from depth map and computer-generated hand model which is initialized by previous frame’s result and finger detection / classification result.

5.2.2 Hand Segmentation

We process a very simple and effective real-time hand segmentation by using a black wrist band and by assuming that a user’s hand is the closest object from camera. This assumption is valid in general environments while interacting with computers and wearable devices. The black wrist band is to get depth voids around wrist since depth sensor cannot capture depth from black object well. The segmentation is processed by finding the connected components from the
closest region of depth map.

5.2.3 Hierarchical Random Sampling

We propose hierarchical random sampling of pixels on each depth map for efficient comparison between depth map from sensor and computer-generated hand model. It is computationally expensive drawing computer-generated hand model on image plane and comparing entire image to input depth map. To reduce required computations, this process is replaced by comparing subset of pixels on input image to computer-generated hand model with only spheres. Thus we do not need to draw computer-generated hand model on image plane since we can compute differences without drawing the model. Although tracking accuracy is improved with more subset points, required computation for comparison is also increased. Therefore, we focus on improving the selection of pixels to process from each depth map by applying hierarchical random sampling which aims to include more pixels on the region which has large depth variations which can be interpreted as more informative on depth map.

First, initial samples $S_1$ are randomly sampled. Then, hierarchical sampling $S_2$ is employed to include more points on large depth variation regions. To find large depth variation regions, the gradient matrix $G$ is computed as the sum of absolute x- and y- directional gradient of depth map $D$. The gradients of x- and y- direction are computed by $3 \times 3$ Sobel operators $(O_x$ and $O_y$) in hand segment region.

$$G = |O_x \ast D| + |O_y \ast D|,$$  \hspace{1cm} (5.1)

where $| \cdot |$ indicates componentwise absolute value and $\ast$ represents convolution.

For initial samples with large gradient, random samples $S_R$ are selected around initial samples by adding random values $u_1, u_2$ from discrete uniform distribution to x- and y- coordinates respectively.

$$S_R = \{ S_1 + (u_1, u_2) \mid G(S_1) > t_1 \}.$$  \hspace{1cm} (5.2)
Figure 5.6: Hierarchical random sampling of pixels from each depth map. This sampling aims to include more pixels on the region which has large depth variations. (a) gradient matrix $G$ of depth map; (b) random sampling $S_1$; (c) hierarchical sampling $S_2$; (d) hierarchical random sampling $S$.

The random samples in $S_R$ are included in hierarchical sample set $S_2$ if the depth difference between initial sample and random sample is greater than a threshold $t_2$.

$$S_2 = \{ S_R \mid |D(S_R) - D(S_1)| > t_2 \}. \quad (5.3)$$

The final sample set $S$ includes both initial samples $S_1$ and hierarchical samples $S_2$. Sampled points $S \in \mathbb{Z}^{2 \times N_s}$ are converted to $X \in \mathbb{R}^{3 \times N_s}$ with $(x,y,z)$ in millimeter, where $N_s$ is the total number of samples.

5.2.4 Adaptive Hand Model

Tracking accuracy increases as the hand model becomes more similar to each user’s hand. Although a personalized hand model can be generated by scanning each user’s hand, it requires a pre-processing for each user. Therefore, we propose adaptive hand model to consider different hand size and shape while avoiding to scan each user’s hand.

Our adaptive hand model consists of a hand size parameter vector $l \in \mathbb{R}^6$ for palm size and finger lengths, and a hand pose parameter vector $p \in \mathbb{R}^{26}$ for hand position and joint rotations. For hand size parameters, one parameter $l_0$ is for palm size and five parameters $\{l_1, \ldots, l_5\}$ are for finger lengths from thumb finger to pinky finger. Finger width parameter is not considered.
since finger width is relatively less important and also computation power is limited. On the other hand, hand pose parameter vector is defined to estimate translations and rotations of hand joints as in Figure 5.7(a).

Since hand shape at each frame is dependent on both size parameters and pose parameters, two parameter vectors should be optimized simultaneously. However, the combination of two parameter vectors is 32 parameters, which is really complex to optimize even without considering correlation between parameters. Therefore, size parameters are considered only when five fingers are detected and classified by the initialization method in Section 5.2.6 since in that case, the optimization of pose parameters is relatively accurate and robust.

Hand model is designed using only spheres to reduce the computational complexity of objective function \([QSW^{+}14]\). The main reason is that in comparison between input depth map and the hand model, the length from a point to the surface of a sphere is simply the length from the point to the center of sphere subtracted by the radius.
5.2.5 Optimization

Objective Function

Our objective function is designed mainly to minimize Euclidean distance between sampled pixels from input depth map and computer-generated hand model. The function consists of two discrepancy terms between hand model and depth map and one validity term. Two discrepancy terms are the Euclidean distance from depth map to hand model and from the model to depth map. The validity term is to check invalid overlapping between parts of hand model. Consequently, total cost is the weighted summation of these three terms. For details of general objective function, we refer authors to the paper [QSW+14]. For our system, since hand shape at each frame is determined by both hand pose parameters and hand size parameters, cost is also determined by both parameters. We conducted experiment with another objective function which incorporates additional cost term to minimize temporal hand size parameter changes. However, tracking accuracy is not improved since the variation of hand size parameters are already regularized by normal distribution.

Particle Swarm Optimization

Modified Particle Swarm Optimization (PSO) is used to find the best hand pose parameters and hand size parameters by minimizing objective function. Each particle represents one state of hand pose parameters and hand size parameters in our algorithm. The optimization method first initializes particles over possible solution range, and then finds the best solution among particles using objective function. Then particles are moved from current state to the direction of the best solution. The algorithm iterates finding the best solution and moving particles to the direction of best solution until it reaches maximum generation or termination condition. Entire particles are initialized as the sum of optimized parameters \((p_o^{(t-1)}, l_o^{(t-1)})\) at previous frame and random values \((r_p, r_l)\) from normal distribution if corresponding finger is
not detected and classified by the method in Section 5.2.6. Otherwise, 75% of corresponding parameters are initialized with the same method, and 25% of them are initialized with the sum of measured parameters from Section 5.2.6 and random values. The distribution of random values are $r_p \sim \mathcal{N}(0, \sigma_1^2)$ and $r_l \sim \mathcal{N}(0, \sigma_2^2)$. However, at the first generation, hand size parameters are not considered to focus on the optimization of pose parameters since inaccurate pose parameters lead to wrong size parameters and the size parameters from previous frame are relatively reliable.

At each generation, for pose parameters, particles are updated using global best particle $g$ and personal best particle $b$. Personal best particle is the state when the particle has the lowest cost until current generation. Global best particle is the lowest cost state among all particles and all generations until current generation. Particles are updated to the direction of personal best particle and global best particle as the following rules:

$$
\begin{align*}
    b_{i,j-1} &= \{ p_{i,k} | \tilde{k} = \arg\min_k C(X^{(t)}, l_{i,1}^{(t)}, p_{i,k}^{(t)}) \}, \\
    g_{j-1} &= \{ p_{i,k} | (\tilde{i}, \tilde{k}) = \arg\min_{i,k} C(X^{(t)}, l_{i,1}^{(t)}, p_{i,k}^{(t)}) \}, \\
    p_{i,j} &= p_{i,j-1} + \alpha_1 (b_{i,j-1} - p_{i,j-1}) + \alpha_2 (g_{j-1} - p_{i,j-1}),
\end{align*}
$$

(5.4)

where $C(\cdot)$ is the objective function, weight $\alpha_1 \sim U[0.5, 1.5]$, weight $\alpha_2 = 2 - \alpha_1$, $i$ is the index of each particle, $j$ and $k$ denote the generation index, and $t$ is current frame index.

For size parameters, particles are not updated to avoid misleading caused by the dependency between size parameters and pose parameters. Although size parameters are not updated at each generation, the particles with better size parameters are likely to have lower cost after many generations since pose parameters will become similar.

After reaching maximum generation, both size parameters and pose parameters are updated with the global best particle.

$$
(p_{o}^{(t)}, k_{o}^{(t)}) = \{ p_{i,k}^{(t)} | (i,k) = \arg\min_{i,k} C(X^{(t)}, l_{i,1}^{(t)}, p_{i,k}^{(t)}) \}.
$$

(5.5)
5.2.6 Initialization at Each Frame

Initialization at each frame is important to avoid manual initialization and error accumulation. However, general initialization methods using random forest or CNNs require too many computations which need high performance GPUs to achieve real-time. Therefore, we improved an efficient finger detection and classification method proposed by [QSW+14]. Although this initialization method works in limited cases, it can automatically initialize hand pose without manual initialization, improves tracking accuracy, and is incorporated in real-time tracking without GPUs. We improve the initialization by estimating palm orientation using both current measurement and prior knowledge from previous frames, which is inspired by Kalman filtering [Kal60].

Finger Detection

A simple finger detection algorithm is employed to detect planar fingers and orthogonal fingers. We define planar fingers as the fingers which are parallel to image plane and orthogonal fingers which are orthogonal to image plane. First, palm center is measured as the maximum of distance transform of hand segment. Then a planar finger candidate is detected by finding connected component from extreme distance point from palm center until the component reaches finger length. The detected finger candidate is classified into either a finger or a non-finger based on the component size. This process is iterated until it detects five fingers or the segment does not have any extreme distance point. After detecting planar fingers, an orthogonal finger candidate is detected by finding connected component from the closest point from a camera on both depth map and hand segment within a small window. It is also classified to either a finger or a non-finger based on the size of region. This process is also iterated until it reaches same condition as planar finger case. For planar fingers, principal component analysis (PCA) is applied to each detected region in order to calculate the orientation of each detected finger. For orthogonal fingers, the orientation is assumed that it is orthogonal to image plane.
Finger Classification

Finger classification is to use the result from finger detection for particle initialization process in Section 5.2.5. The algorithm classifies each detected finger to one of five finger classes. First, palm orientation is measured by applying PCA to palm segment. However, the measured orientation $\theta_m$ is not robust and accurate enough to use directly for classification. Therefore, palm orientation is predicted by the weighted summation of previously estimated palm orientation $\theta_{o,t-1}$ from PSO and currently measured palm orientation $\theta_{m,t}$. The weights in summation are decided by the measurement error $e_m$ and priori estimation error $e_o$. This is inspired by Kalman filtering [Kal60].

\[
\begin{align*}
\theta_{p,t} &= \frac{e_{m,t}}{e_{m,t} + e_{o,t}} (\theta_{o,t-1} + c_{t-1}) + \frac{e_{o,t}}{e_{m,t} + e_{o,t}} \theta_{m,t}, \\
e_{m,t} &= |\theta_{o,t-1} - \theta_{m,t-1}| + a_1 e_{m,t-1}, \\
e_{o,t} &= |\theta_{o,t-1} - (\theta_{o,t-2} + c_{t-2})| + a_2 e_{o,t-1},
\end{align*}
\]

where $t$ is current frame index and $a_i$ is chosen as a constant for simplicity. In our experiments, both $a_1$ and $a_2$ are set to 0.5. After each frame, priori estimation constant $c$ is updated as $c_t = \theta_{o,t} - \theta_{o,t-1}$.

A set $F$ of junction points of fingers and palm is computed using detected fingertips, finger orientations, and finger length. Another set $Q$ of junction points is calculated using hand model
and predicted palm orientation. Each junction point $f_i \in F$ from an input image is matched to the closest junction point $q_j \in Q$ from hand model.

$$J(f_i) = \arg\min_j ||f_i - q_j||_2.$$  

(5.7)

If more than two of detected junctions are classified into the same class, optimization is employed to find the result with minimum cost. The algorithm first finds possible combinations $B$ that have minimum changes in initial classification.

$$B_k = \{(J(f_1) + u_1, \ldots, J(f_n) + u_n) | \min \sum_{i=1}^n |u_i|^2, \forall i \neq j, J(f_i) + u_i \neq J(f_j) + u_j\}.$$  

(5.8)

The final class $L$ is classified as follows:

$$L = \{B_k | \bar{k} = \arg\min_k \sum_{i=1}^n (||f_i - q_{B_k}(i)||_2)\},$$  

(5.9)

where $i$ is the index of detected finger and $n$ is the number of detected fingers. Figure 5.9 illustrates the procedure of finger classification. In Figure 5.9 (b), white circles represent detected finger junctions and colored circles from red to purple indicate hand model junctions from thumb finger to pinky finger. A yellow circle which corresponds to predicted joint junction of index finger is overlapped with a white circle. Although this initialization method cannot detect all the fingers at
Table 5.3: Processing time of each component.

<table>
<thead>
<tr>
<th>Process</th>
<th>Time (in ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finger detection/classification</td>
<td>2.5</td>
</tr>
<tr>
<td>Optimization</td>
<td>53.5</td>
</tr>
<tr>
<td>Others</td>
<td>6.5</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>62.5</strong></td>
</tr>
</tbody>
</table>

every frame, it improves tracking accuracy, can initialize at each frame including the very first frame, and takes only a few milliseconds using only CPU.

5.2.7 Experiments and Results

The algorithm is tested using a Creative Senz3D camera and a computer with Intel Core i7-3770 3.40 GHz CPU, 16 GB RAM, and without GPUs. Although the machine has 16GB RAM, this algorithm only uses about 60MB memory. Also, although we use 3.40 GHz CPU, we believe similar computation power can be obtained using the combination of mobile CPU and mobile GPUs.

We captured 500 frames for each subject from three subjects and labeled wrist and five fingertips of last 400 frames for each sequence. The dataset is available on our repository\(^1\). Initial 100 frames contain open-hand pose that camera can capture at least some fingers using the algorithm in Section 5.2.6 for automatic initialization. The initial frames are not used to compute accuracy. Error in accuracy is computed using 3D Euclidean distance in millimeter.

Unless specifically mentioned, we sample 256 points from each depth map and optimize with 256 particles and 6 generations. Table 5.3 shows the processing time at this setting. Even though the algorithm is not fully optimized, it achieves about 16 FPS using eight threads on CPU.

\(^1\)[https://github.com/byeongkeun-kang/HandTracking](https://github.com/byeongkeun-kang/HandTracking)

90
Table 5.4: Average error for varying standard deviations of hand size parameter randomness. To compute average error, we consider three subjects and seven initial scaling factors chosen from 0.6 to 1.2. The detailed explanation of standard deviation is on Figure 5.10.

<table>
<thead>
<tr>
<th>( \sigma^2 ) in Sec. 2.5</th>
<th>0</th>
<th>0.005</th>
<th>0.01</th>
<th>0.015</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error (in mm)</td>
<td>28.00</td>
<td>25.31</td>
<td>22.90</td>
<td>22.38</td>
</tr>
</tbody>
</table>

Table 5.5: Correct classification rate (in %) of finger classification.

<table>
<thead>
<tr>
<th>CCR</th>
<th>Thumb</th>
<th>Index</th>
<th>Middle</th>
<th>Ring</th>
<th>Pinky</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>87.1</td>
<td>65.5</td>
<td>69.4</td>
<td>59.0</td>
<td>71.8</td>
<td>70.6</td>
</tr>
<tr>
<td>Proposed</td>
<td>97.2</td>
<td>86.3</td>
<td>78.1</td>
<td>70.0</td>
<td>92.1</td>
<td>84.8</td>
</tr>
</tbody>
</table>

Adaptive Hand Model

The performance comparison of hand model with adaptive scaling and fixed scaling is demonstrated in Figure 5.10 for different subjects, initial scaling factors, and hand size parameter randomness. Scaling factors are chosen from 0.6 to 1.2 with 0.1 step size and standard deviation of randomness \( \sigma^2 \) is chosen from 0 to 0.015 with 0.005 step size. Also, Table 5.4 clearly shows that adaptive hand model reduces error about 5 mm. The overall results show that adaptive hand model adjusts hand scaling factor automatically to minimize discrepancy between pre-defined hand model and user’s hand, and users do not need to manually and experimentally select scaling factor of hand model. Also, to consider the case that the user is changed after starting tracking, we keep update hand scaling factor.

Finger Classification

Correct classification rate (CCR) is calculated for each finger with and without the proposed palm orientation prediction in Table 5.5. The result shows that by using the proposed prediction, the average CCR is improved from 70.6% to 84.8%.
Figure 5.10: Comparison of adaptive hand model and fixed scale hand model for three subjects (a) subject 1, (b) subject 2, (c) subject 3. The legend represents standard deviations $\sigma_2$ of hand size parameter randomness. Standard deviation of 0 means that the scaling factor of hand model is always the same with initial scaling factor. Larger standard deviation means more possibility of large update of scaling factor at each frame. The result shows that the performance of adaptive hand model is better than the performance of fixed scaled hand model in general.

Hierarchical Random Sampling

The performance of random sampling and hierarchical random sampling is compared in Table 5.6. The average error is computed using the hand model scaled by the best performance fixed scaling factor, 0.7, 0.9, and 0.7 for subject 1, 2, and 3 respectively. The average accuracy is improved from 17.01\text{mm} to 16.11\text{mm}. The computational cost of calculating gradient is much smaller than increasing the number of generations, particles, or samples to achieve the same amount of improvement. However, if the number of sampling pixels is too small, random sampling might be better since hierarchical random sampling prevents that sampled pixels are more globally distributed.
Table 5.6: Performance comparison of random sampling and the proposed hierarchical random sampling.

<table>
<thead>
<tr>
<th>Sampling</th>
<th>Random [QSW$^+14$]</th>
<th>Hierarchical Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error (mm)</td>
<td>17.01</td>
<td>16.11</td>
</tr>
</tbody>
</table>

5.3 **Summary**

In Section 5.1, we show the efficacy of using convolutional neural networks and a depth sensor for ASL fingerspelling recognition system. We collect and are in a process to share the dataset of depth images for ASL fingerspelling system. Our approach of classifying 31 signs of alphabets and numbers using depth image and CNNs achieves real-time performance and state-of-the-art accuracy even for different signers. We conclude that pre-training from auxiliary task of image classification from color images is helpful for apparently different type of input data such as depth image. The trained model and dataset is available on our repository$^2$.

In Section 5.2, we present an efficient hand articulations tracking system for mobile and wearable devices which do not have high performance GPUs. We show that the proposed system achieves both automatic hand model adjustment using adaptive hand model and real-time tracking without using GPUs. We also achieve improved accuracy using hierarchical random sampling and improved efficient initialization at each frame while limiting required computations.

This chapter, in part, is a reprint of materials in the conference papers “Real-time Sign Language Fingerspelling Recognition using Convolutional Neural Networks from Depth map”, B. Kang, S. Tripathi, and T. Q. Nguyen [KTN15] and “Efficient Hand Articulations Tracking using Adaptive Hand Model and Depth map”, B. Kang, Y. Lee, and T. Q. Nguyen [KLQN15]. The dissertation author is the primary investigator and author of these papers.

$^2$https://github.com/byeongkeun-kang/FingerspellingRecognition
Chapter 6

Data Augmentation

6.1 Introduction

Most images and videos exist in a compressed form since data compression saves lots of data storage and network bandwidth and further enables many applications such as real-time video streaming in cell phones. Compression is indeed crucial, considering compressed image and video can be about 10 times and 50 times smaller than raw data, respectively. Nevertheless, typical generative adversarial networks (GAN) focus on generating raw RGB images or videos [GPAM+14, RMC15, ACB17, GAA+17, VPT16, XLM+18]. Considering one of the most common usages of GANs is generating large-scale synthetic images/videos for data augmentation, the created images/videos often require compression in a post-processing stage to store the large dataset in a hardware [SWL18, WLZ+18b, WLZ+18a, FAKA+18]. Besides, typical GANs are evaluated using the generated raw RGB data although compression is processed to the raw data prior to final applications. Hence, we investigate GAN frameworks that aim to generate compressed data and to evaluate the networks using the generated encoded images.

We focus on the GAN frameworks for compressed image generation since image generation networks [GPAM+14, RMC15, ACB17, GAA+17] have been far more studied comparing
to video generation networks [VPT16, XLM+18]. In image compression, JPEG [ITU82, ITU92] has been one of the most commonly used lossy compression methods [Wal92, PM92, HLNS17]. It has been used in digital cameras and utilized for storing and transmitting images. While JPEG has many variants, typical JPEG consists of color space transformation, chroma subsampling, block-based discrete cosine transform (DCT) [ANR74], quantization, and entropy coding [Huf52]. In more details, the compression method converts an image in the RGB domain to the image in another color space (YCbCr) that separates the luminance component and the chrominance components. The chrominance components are then downsampled. A 8×8 DCT is applied to both the luminance component and the subsampled chrominance components to represent them in the frequency domain. It discards details of high-frequency information by applying quantization. Lastly, the processed data is stored using entropy coding.

We argue that investigating the frameworks of generating compressed images is important to accomplish the creation of more visually plausible large-scale images that require storing in a compressed domain. Typical GAN frameworks optimize and select the networks’ architectures and parameters (weights) based on generated raw RGB images. Accordingly, if we take into account the compression process in the post-processing stage, the choice might not be the optimal decision. Hence, we propose to optimize/determine architectures and parameters by evaluating them using generated images in a compression domain.

We propose a novel framework that generates compressed images using generative adversarial networks as shown in Figure 6.1. The framework consists of a generator, a discriminator, and a transformer between the generator and the discriminator. The proposed generator produces compressed data given a randomly selected noise in a latent space. The transformer is applied to make the data from the generator and the training data to be in the same domain. Since the generator outputs compressed data and the training data is raw RGB images, the transformer should convert encoded data to a raw image, so is a decoder. The discriminator takes synthesized images and real images and aims to differentiate them.
Figure 6.1: The proposed framework to generate compressed images. The framework consists of a generator, a discriminator, and a transformer between the generator and the discriminator. The visualized generator output is an intermediate example.

The proposed generator has three paths, one path for the luminance component and the other two paths for the chrominance components. The separate paths are proposed to process any requested chroma subsampling. We also propose the locally connected layer that takes an input of a subregion and outputs for the corresponding subregion. The proposed locally connected layer is able to handle block-based processing in JPEG compression. In summary, the contributions of our work are as follows:

- We propose the framework that generates JPEG compressed images using generative adversarial networks.

- We propose the locally connected layer to enable block-based operations in JPEG compression.

- We analyze the effects of compression for the proposed method and other methods.
6.2 Related Works

6.2.1 Generative Adversarial Networks

Generative adversarial networks were introduced in [GPAM+14] where the framework is to estimate generative models by learning two competing networks (generator and discriminator). The generator aims to capture the data distribution by learning the mapping from a known noise space to the data space. The discriminator differentiates between the samples from the training data and those from the generator. These two networks compete since the goal of the generator is making the distribution of generated samples equivalent to that of the training data while the discriminator’s objective is discovering the discrepancy between the two distributions.

While the work in [GPAM+14] employed multilayer perceptrons for both generator and discriminator, deep convolutional GANs (DCGANs) replaced multilayer perceptrons by convolutional neural networks (CNNs) to take the advantage of shared weights, especially for image-related tasks [RMC15]. To utilize CNNs in the GAN framework, extensive architectures with the relatively stable property during training, are explored. They examined stability even for models with deeper layers and for networks that generate high-resolution outputs. The analysis includes fractional-stride convolutions, batch normalization, and activation functions. Salimans et al. presented the methods that improve the training of GANs [SGZ+16]. The techniques include matching expected activations of training data and those of generated samples, penalizing similar samples in a mini-batch, punishing sudden changes of weights, one-sided label smoothing, and virtual batch normalization.

Arjovsky et al. presented the advantage of the Earth-Mover (EM) distance (Wasserstein-1) comparing to other popular probability distances and divergences such as the Jensen-Shannon divergence, the Kullback-Leibler divergence, and the Total Variation distance [ACB17]. The advantage of the EM distance is that it is continuous everywhere and differentiable almost everywhere when it is applied for a neural network-based generator with a constrained input.
noise variable. They also showed that the EM distance is a more sensible cost function. Based on these, they proposed Wasserstein GAN that uses a reasonable and efficient approximation of the EM distance. They then showed that the proposed GAN achieves improved stability in training. However, clipping weights for Lipschitz constraint in [ACB17] might cause optimization difficulties [GAA+17]. Hence, Gulrajani et al. proposed penalizing the gradient norm to enforce Lipschitz constraint instead of clipping [GAA+17].

Wasserstein GAN trains its discriminator multiple times at each training of its generator so that the framework can train the generator using the more converged discriminator [ACB17]. To avoid the expensive multiple updates of the discriminator, Heusel et al. proposed to use the two time-scale update rule (TTUR) [PLB15] in a Wasserstein GAN framework [HRU+17]. Since TTUR enables separate learning rates for the discriminator and the generator, they can train the discriminator faster than the generator by selecting a higher learning rate for the discriminator comparing to that of the generator. It is also proved that TTUR converges to a stationary local Nash equilibrium under mild assumptions. They further experimentally showed that their method outperforms most other state-of-the-art methods. Hence, the proposed framework of this work is based on [HRU+17].

6.2.2 Image Compression: JPEG

JPEG [ITU92] has been one of the most commonly used lossy compression methods for still images with continuous tones [Wal92, PM92, HLNS17]. It has been used for digital cameras, photographic images on the World Wide Web, medical images, and many other applications.

In JPEG, discrete cosine transform (DCT) is utilized since it achieves high energy compaction while having low computational complexity. Considering an image contains uncorrelated (various) information, block-based DCT is used so that each block contains correlated data. Using too small block prevents from compressing correlated information. Too large block with uncorrelated pixels increases computational complexity without compression gain. 8×8 block
size is selected based on a psychovisual evaluation.

DCT transforms an $8 \times 8$ block of an image to 64 amplitudes of 2D cosine functions with various frequencies. Since the sensitivity of a human eye is different for each frequency, quantization is applied differently for each amplitude. The amplitudes for low-frequencies are maintained with high accuracy and those of high-frequencies are quantized using larger quantization value. Quantization is responsible for most of the information loss in JPEG.

After quantization, since most of the non-zero components are for low-frequencies, the amplitudes are encoded in zig-zag order using a value pair. The information is then encoded using Huffman coding considering the statistical distribution of the information [Huf52].

While these are the baseline of JPEG, other additional methods and components were also suggested for particular purposes. Also, typical JPEG uses the YCbCr color space and chroma subsampling [ITU82].

6.2.3 Other Related Works

Our work differs from learning neural networks for image compression such as autoencoders [TSCH17, TOH+16, TVJ+17, SBS18] that aims to learn image encoders to compress real images. The proposed method and learning encoders differ in two aspects. First, the goal of the proposed method is generating synthetic JPEG compressed data while that of the latter is compressing real images. Second, the proposed method intends to utilize existing standard decoder in general electronic devices while the latter requires a particular decoder to decode the compressed data.

This work is also distinct from [KTKY17, MGC+18] that are about utilizing GANs for postprocessing real data in a frequency domain. The proposed work generates compressed images from random noises in the latent space.
6.3 Joint Image Generation and Compression

We propose a framework that generates JPEG compressed images using generative adversarial networks. We use the architectures analyzed in the TTUR method [HRU+17] as our baseline networks since the method achieves one of the state-of-the-art results. The generator in the baseline architecture consists of one fully connected layer, four residual blocks, and one convolution layer. The discriminator in the architecture consists of one convolution layer, four residual blocks, and one fully connected layer as shown in Figure 6.2(b). The residual block consists of two paths (see Figure 6.2(c)). One path has two convolution layers with filter-size of $3 \times 3$, and the other has only one convolution layer with filter-size of $1 \times 1$. All the convolution layers outside of residual blocks have filter-size of $3 \times 3$.

Given the baseline architecture, we propose an architecture and training strategy to generate JPEG compressed images in the framework of generative adversarial networks. We first propose a novel generator in Section 6.3.1. The proposed generator has three paths, one for each luminance or chrominance component. The proposed generator also has additional layers including the proposed locally connected layers, chroma subsampling layer, and quantization layer. An entropy encoding layer is omitted since the encoding is lossless and located at the last layer. Hence, this exclusion does not affect the results.

We then present the processing between the generator and the discriminator in Section 6.3.2. Since typical generators generate RGB images which are in the same domain with the training images, any additional processing is not required to use the output of the generator for the input to the discriminator. However, since the proposed generator produces JPEG compressed data, the outputs of the generator and the training images are in different domains and cannot be used together for the discriminator. Consequently, we need to either compress the training images or decode the generated JPEG compressed data so that they are in the same domain.

In Section 6.3.3, we discuss training strategies for the proposed architecture. Although
many studies have been conducted to improve training stability, training GANs for non-typical images is still quite challenging.

6.3.1 Generator

We propose a novel generator that generates JPEG compressed images (see Figure 6.2). The generator consists of six locally connected layers, two chroma subsampling layers, and a quantization layer in addition to the layers in the baseline generator. The generator has three paths where each path generates one of luminance or chrominance components in the YCbCr...
representation. The separated paths are required to handle any required chroma subsampling since the resolution of a luminance component and chrominance components are different if chroma subsampling is applied. The locally connected layer is proposed to operate block-based processing. An entropy encoding layer is not applied since the encoding is lossless and at the last layer, which does not impact the results.

The proposed locally connected layer takes an input of a subregion and produces an output for the corresponding subregion (see Figure 6.3). The layer is proposed to perform operations comparable to block-based processing. Comparing to a convolution layer, the proposed layer is different since a convolution layer takes an input from a region and outputs to only a single location. The nearby outputs from a convolution layer are produced by using different regions of inputs. In other words, to generate $8 \times 8$ outputs using a convolution layer, the layer actually takes inputs from 64 different regions by shifting a filter (weights). The proposed layer is also dissimilar from a fully connected layer since a typical fully connected layer does not share weights while the proposed locally connected layer shares weights between blocks. For all paths in the generator, the first locally connected layer (Loc1) and the second locally connected layer (Loc2) employ the block-size of $1 \times 1$ and $8 \times 8$, respectively. The block-size of $8 \times 8$ is selected considering $8 \times 8$ block-based inverse DCT in a JPEG decoder. The block-size of $1 \times 1$ is a special case that can be reproduced by a convolution layer.

Chroma subsampling is processed by averaging the amplitudes of the chrominance component of each block and by subsampling from the block to a scalar. In this work, we investigate the proposed architecture using the popular subsampling ratios, 4:4:4, 4:2:2, and 4:2:0. The 4:4:4 ratio means no subsampling and preserves all the chrominance information. The 4:2:2 mode averages and subsamples with 2:1 ratio for only the horizontal axis. Consequently, the horizontal resolution of the output is half of the input. For the 4:2:0 subsampling, both horizontal and vertical axes are averaged and subsampled with 2:1 ratio. Consequently, each block of $2 \times 2$ pixels is turned to a scalar (see Figure 6.2(a)).
Figure 6.3: Visual comparison of (a) convolution layer with filter-size of $3 \times 3$, (b) fully connected layer, and (c) proposed locally connected layer with block-size of $4 \times 4$. The locally connected layer operates comparable to block-based processing. Each region of output is produced by the summation of the multiplication of the corresponding region of input and shared weights. The weights are shared between blocks, but not between outputs in a block.

Forward processing all the layers in the proposed generator before quantization generates amplitudes of 2D cosine functions for luminance and chrominance components. Quantization is then performed using a conventional quantization method in JPEG compression. We employ the conventional method so that the final output is able to be de-quantized by using a typical JPEG decoder. Quantization is performed by dividing amplitudes by quantization matrices and by rounding the quantized amplitudes to an integer [CP84, Loh84]. The quantization matrices are determined based on a user-selected quality factor and can also be selected in an encoding process. We employ popular quantization matrices that are shown at (6.1) and (6.2) [PPMP91]. The given quantization matrices $(Q_{l,50}, Q_{c,50})$ are for the quality factor of 50. The former and the
latter matrices are for luminance component ($Q_l$) and chrominance part ($Q_c$).

\[
Q_{l,50} = \begin{bmatrix}
16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\
12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\
14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\
14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\
18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\
24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\
49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\
72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \\
\end{bmatrix}
\]  \hspace{1cm} (6.1)

\[
Q_{c,50} = \begin{bmatrix}
17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\
18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\
24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\
\end{bmatrix}
\]  \hspace{1cm} (6.2)

Quantization matrix for another quality factor for luminance component is computed as follows:

\[
Q_{l,n} = \begin{cases} 
\max(1, \lfloor \frac{100-n}{50} Q_{l,50} + 0.5 \rfloor), & \text{if } n \geq 50, \\
\lfloor \frac{50}{n} Q_{l,50} + 0.5 \rfloor, & \text{otherwise}.
\end{cases}
\]  \hspace{1cm} (6.3)

where $n \in (0, 100]$ denotes a quality factor. The conversion of the quantization matrix for chrominance is equivalent.

The architecture in Figure 6.2 is used for the LSUN bedroom dataset [YZS+15] which
aims to generate images with the resolution of $64 \times 64$. For the CIFAR-10 dataset [Kri09] whose objective resolution is $32 \times 32$, the output dimensions of all the layers in both the generator and the discriminator are reduced by half for both x- and y-axes. Also, the number of activations (feature maps) is reduced by half up to the last residual block in both the generator and the discriminator.

### 6.3.2 Transform between Generator and Discriminator

Since the discriminator takes half of the inputs from the generator and the other half from the training dataset, the two data should be in the same domain (representation). However, the training data set contains real images in the RGB domain and outputs of the proposed generator are JPEG compressed data. Hence, we have to either compress the training data or decode the outputs of the generator so that the two images are in the same domain.

We examined both alternatives. It turns out that it is better to decode outputs of the generator before providing them for inputs to the discriminator. Our opinion is that convolution layers, which are major elements of the discriminator, are invented for real images which usually have a continuous tone. However, the compressed data contains amplitudes of block-based DCT which vary largely at the boundary of blocks and also in the blocks. Consequently, the discriminator provides inferior-quality gradients to the generator and hinders training a good generator. Hence, the proposed framework has a decoder that takes outputs of the generator and renders inputs to the discriminator during training. We do not need this conversion (decoder) after training since we only utilize the discriminator for training and our goal is generating compressed data.

Given an output of the generator, we first de-quantize the amplitudes by multiplying them by the corresponding quantization matrices used in the generator. We then apply inverse DCT to transform the amplitudes in the frequency domain to the contents in the 2D color domain. We upsample chrominance components to the same resolution of the luminance component if chroma subsampling is applied in the generator. We then convert the amplitudes in the YCbCr space to
the RGB space. Lastly, we clip the amplitudes so that after compensating shifting and scaling, the amplitudes are in the range of [0, 255].

6.3.3 Training

As GANs are difficult to train [GPAM+14], many studies have been conducted to improve the stability of training [SGZ+16, ACB17, GAA+17]. Still, by employing current state-of-the-art training algorithms to our problem, we had difficulty in training the proposed networks. Hence, we propose a novel loss function to train the proposed framework.

Given a loss function, $L$, the generator $G$ and the discriminator $D$ are trained by playing a minimax game as follows:

$$\min_G \max_D L(G, D)$$  \hspace{1cm} (6.4)

Considering the objective function in the Wasserstein GAN with gradient penalty [GAA+17], we propose a loss function $L$ by adding an additional loss term for the generator.

$$L(G, D, P) = \mathbb{E}_{\tilde{x} \sim P_{\tilde{x}}} |D(\tilde{x}) + \gamma |P(G(\tilde{x})) - \hat{G}(\tilde{x})|| - \mathbb{E}_{x \sim P_r} [D(x)] + \lambda \mathbb{E}_{\hat{x} \sim P_{\hat{x}}}[\|\nabla_{\hat{x}}D(\hat{x})\|_2^2 - 1]^2$$  \hspace{1cm} (6.5)

where $P_{\tilde{x}}$ and $P_r$ denote the generator distribution and the training data distribution. The authors in [GAA+17] implicitly defined $P_{\tilde{x}}$ as sampling uniformly along straight lines between pairs of points sampled from $P_r$ and $P_{\tilde{x}}$. $\hat{G}$ is the layers in the generator $G$ before any locally connected layer. $\hat{G}$ is initialized by the parameters that are trained using [GAA+17]. $P$ is the transformation between the generator and the discriminator. $\gamma$ is the hyperparameter to weight between typical generator loss and the proposed additional generator loss. Gradient penalty coefficient $\lambda$ is 10 and $\gamma$ is 100 in all experiments. The learning rates for the discriminator and the generator are 0.0003 and 0.0001, respectively.

We believe that further studying on an optimization algorithm for non-typical images
can improve the quality of generated results further. However, developing a novel optimization algorithm is beyond the scope of this work.

6.4 Experiments and Results

6.4.1 Dataset

We experiment using the CIFAR-10 training dataset [Kri09] and the LSUN bedroom training dataset [YZS+15]. The CIFAR-10 dataset consists of 50,000 images with the resolution of 32×32. The dataset includes images from 10 categories (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck). The LSUN bedroom dataset consists of 3,033,042 bedroom images. The images are scaled to 64×64 following the previous works [GAA+17].

6.4.2 Metric

We use the Fréchet Inception Distance (FID) [HRU+17] which was improved from the Inception score [SGZ+16] by considering the statistics of real data. The FID computes the Fréchet distance (also known as Wasserstein-2 distance) [Fre57, DL82] between the statistics of real data and that of generated samples. The distance is computed using the first two moments (mean and covariance) of activations from the last pooling layer in the Inception v3 model [SVI+16]. The FID \(d\) is computed as follows:

\[
d^2((m, C), (m_w, C_w)) = \|m - m_w\|^2_2 + \text{Tr}(C + C_w - 2(CC_w)^{1/2})
\]

where \((m, C)\) and \((m_w, C_w)\) represent the mean and covariance of generated samples and of the real dataset. \((m_w, C_w)\) are measured using entire images in the training dataset. \((m, C)\) are computed using 50,000 generated images.
6.4.3 Results

We analyze the proposed method and the architectures in the TTUR method \([\text{HRU}^+17]\) by training them using the datasets in Section 6.4.1 and by evaluating them quantitatively and qualitatively. For quantitative comparison, we measure the Fréchet Inception Distance (FID) in Section 6.4.2. Table 6.1 and Table 6.2 show the quantitative results for the CIFAR-10 dataset \([\text{Kri09}]\) and the LSUN bedroom dataset \([\text{YZS}^+15]\), respectively. In both tables, we show the FIDs for three chroma subsampling ratios \((4:4:4, 4:2:2, 4:2:0)\) and for four quality factors of quantization \((100, 75, 50, 25)\). The first row shows the FID variations of real images by applying chroma subsampling and quantization. The second row presents the FID of the original TTUR method generating RGB images \([\text{HRU}^+17]\). For this analysis, JPEG compression is processed as a post-processing step that follows the neural networks. The third row shows the result of the TTUR method for generating JPEG compressed images directly. In Table 6.1, the fourth row presents the result of the fully connected (FC) generator in the TTUR method for generating JPEG compressed images directly. In the last row, we show the result of the proposed method.

We show visual results of the CIFAR-10 and the LSUN bedroom datasets in Figures 6.4 and 6.5, respectively. On the left side, we denote the quality factor for quantization and chroma subsampling ratios. We show the results of \((100, 4:4:4), (100, 4:2:2), (100, 4:2:0), (75, 4:4:4), (50, 4:4:4), \) and \((25, 4:4:4)\) from the first row to the last row. On the first column, we show the results of real images that are processed by the corresponding compression. The second column shows the results of the generated images using the original TTUR method \([\text{HRU}^+17]\). The result images are first generated as RGB images and are then coded and decoded in a post-processing stage. The third column shows the result of the TTUR method \([\text{HRU}^+17]\). In Figure 6.4, the fourth column presents the result of the FC generator in the TTUR method \([\text{HRU}^+17]\). The last column shows the results of the proposed method. The last three columns in Figure 6.4 and two columns in Figure 6.5 are the results of generating JPEG compressed images in the networks.

To compute the FID in the first row in both tables, we first encode and decode 50,000
Table 6.1: Quantitative comparison using FIDs for the CIFAR-10 dataset. The lower FID means the better result. The first and second rows show FIDs of training images and of generated RGB images using the TTUR method by processing compression in a post-processing stage. The last three rows present generating compressed data directly using the TTUR method, the FC Generator in the TTUR, and the proposed method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Generator output</th>
<th>Chroma subsampling</th>
<th>Quality factor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>4:4:4</td>
<td>100</td>
</tr>
<tr>
<td>Real data</td>
<td>-</td>
<td></td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4:2:2</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4:2:0</td>
<td>1.86</td>
</tr>
<tr>
<td>TTUR</td>
<td>RGB image</td>
<td>4:4:4</td>
<td>26.10</td>
</tr>
<tr>
<td>[HRU+17]</td>
<td></td>
<td>4:2:2</td>
<td>25.79</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4:2:0</td>
<td>26.90</td>
</tr>
<tr>
<td>TTUR</td>
<td>JPEG compressed</td>
<td>4:4:4</td>
<td>68.80</td>
</tr>
<tr>
<td>[HRU+17]</td>
<td>image</td>
<td>4:2:2</td>
<td>62.23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4:2:0</td>
<td>75.99</td>
</tr>
<tr>
<td>FC generator</td>
<td>JPEG compressed</td>
<td>4:4:4</td>
<td>83.31</td>
</tr>
<tr>
<td>[HRU+17]</td>
<td>image</td>
<td>4:2:2</td>
<td>91.32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4:2:0</td>
<td>71.37</td>
</tr>
<tr>
<td>Proposed method</td>
<td></td>
<td>4:4:4</td>
<td>25.57</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4:2:2</td>
<td>25.80</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4:2:0</td>
<td>26.78</td>
</tr>
</tbody>
</table>

training images and then compute the statistics of the processed images. We then estimate the FID between the statistics of the original training data and those of the processed images. For the CIFAR-10 dataset in Table 6.1, the distance is close to 0 using the chroma subsampling ratio of 4:4:4 and the quality factor of 100. It’s quite small since the encoding/decoding does not impact the images much (only small rounding errors, etc). By decreasing quality factor and by subsampling from a larger region, the encoding/decoding distorts images more and hence, FID increases. Since decreasing quality factor by 25 affects images much more than adjusting chroma subsampling from 4:4:4 to 4:2:0, FID is also increased by a larger amount.

For the LSUN bedroom dataset in Table 6.2, the distance of real data using the chroma
Table 6.2: Quantitative comparison using FIDs for the LSUN dataset. The lower FID means the better result. The first and second rows show FIDs of training images and of generated RGB images using the TTUR method by processing compression in a post-processing stage. The last two rows present generating compressed data directly using the TTUR and the proposed method.

<table>
<thead>
<tr>
<th>Method</th>
<th>Generator output</th>
<th>Chroma subsampling</th>
<th>Quality factor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>75</td>
</tr>
<tr>
<td>Real data</td>
<td>-</td>
<td>9.96</td>
<td>6.82</td>
</tr>
<tr>
<td>TTUR [HRU+17]</td>
<td>RGB image</td>
<td>12.44</td>
<td>10.24</td>
</tr>
<tr>
<td>TTUR [HRU+17]</td>
<td>JPEG compressed image</td>
<td>35.70</td>
<td>27.78</td>
</tr>
<tr>
<td>Proposed method</td>
<td></td>
<td>12.13</td>
<td>9.99</td>
</tr>
</tbody>
</table>

The subsampling ratio of 4:4:4 and the quality factor of 100 is much greater than that in the CIFAR-10 dataset. The FID is defined by the distance between the statistics of the entire training data and those of 50,000 processed or generated images. Since the number of images in the CIFAR-10 dataset is 50,000, the distance is quite small considering the encoding/decoding does not distort much. However, since the LSUN bedroom dataset contains 3,033,042 images, 50,000 images should be sampled to compute the statistics of the processed images. It causes a relatively larger FID for the LSUN bedroom dataset. It is also interesting to note that for the LSUN dataset, the quality factor of 75 is better than that of 100 in most experiments. We believe since the bedroom images often have continuous tone because of its contents or pre-processing, discarding high-frequency components decreases FID.

FIDs in the second row is computed by first generating RGB images using the TTUR
Figure 6.4: Visual results of generating compressed images using the CIFAR-10 dataset. On the left side, we denote the quality factor for quantization and chroma subsampling ratios. (a) Real data. (b) TTUR (RGB). (c) TTUR (encoded). (d) FC generator (encoded). (e) Proposed method.
Figure 6.5: Visual results of generating compressed images using the LSUN dataset. On the left side, we denote the quality factor for quantization and chroma subsampling ratios.
method [HRU⁺17] and by encoding/decoding the generated RGB images. As generating RGB images have been studied a lot in recent years and the TTUR method is one of the state-of-the-art methods, generated RGB images are quite visually plausible. FID is increased by encoding/decoding the images using a lower quality factor and subsampling from a larger region. Some of the distortions can be visually observed in Figures 6.4 and 6.5.

The third row presents applying the same method to generate JPEG encoded images. The results demonstrate that directly applying the method does not produce competitive results. The fourth row in Table 6.1 shows the results of applying the FC generator in the TTUR method [HRU⁺17] for generating encoded images. While FC generator often performs poorer than the selected TTUR method for generating typical images, we tried the FC generator to avoid extensively applied convolution layers in the TTUR method. However, the FC generator does not perform well even for generating encoded images.

The last row in both tables shows the results of the proposed method. The proposed method achieves promising results for generating JPEG encoded images directly. The proposed method outperforms applying the TTUR method for generating JPEG encoded image directly. Moreover, the proposed method is competitive to the method that generates RGB images using the TTUR method and compresses them by post-processing.

### 6.5 Summary

We present a generative adversarial network framework that combines image generation and compression by generating compressed images directly. We propose a novel generator consisting of the proposed locally connected layers, chroma subsampling layers, quantization layers, residual blocks, and convolution layers. We also present training strategies for the proposed framework including the loss function and the transformation between the generator and the discriminator. We demonstrate that the proposed framework outperforms applying the state-of-
the-art GANs for generating compressed data directly. Moreover, we show that the proposed method achieves competitive results comparing to generating raw RGB images using one of the state-of-the-art methods and compressing the images by post-processing. We believe that the proposed method can be further improved by investigating optimization algorithms for learning to generate compressed data. We also consider the scenario where the proposed method can serve as a baseline method for further studies.

This chapter, in part, is a reprint of materials in the submitted journal paper “Toward Joint Image Generation and Compression using Generative Adversarial Networks”, B. Kang, S. Tripathi, and T. Q. Nguyen [KTN18]. The dissertation author is the primary investigator and author of this paper.
Chapter 7

Conclusion

7.1 Summary of Contributions

We investigated learning robust representations in a random forest and in deep neural networks. For a random forest framework, we designed an unconstrained representation that can learn weights, shapes, and sparsities in order to utilize more optimal representations. We then proposed a framework to learn/apply the representation including particle swarm optimization. The experimental analysis demonstrates that the proposed method achieves competitive accuracy in real-time. Regarding deep neural networks, we proposed the depth-adaptive deep neural network that achieves a step toward learning/utilizing depth-invariant representations. To accomplish it, we proposed the adaptive perception neuron and the in-layer multiscale neuron. The adaptive perception neuron is to adjust a receptive field based on the depth at each location. The in-layer multiscale neuron is to learn/extract multiscale representations at each layer. The multiscale neuron enables the networks to efficiently learn representations for diversely sized objects. Experimental results demonstrate that the proposed method improves accuracy without any additional layers or pre/post-processing.

Moreover, we worked on the cascaded random forest for hand segmentation using only
a depth map. The cascaded random forest firstly detects hand region with bounding boxes and segments hands in pixel-level. The method processes very efficiently by reducing region-of-interest by using a small random forest. Furthermore, we studied fingerspelling recognition and hand articulations tracking that often require hand segmentation in preprocessing. Fingerspelling recognition is approached by using convolutional neural networks. Hand articulation is tracked by generating a virtual hand model and comparing it to an input depth map.

Lastly, we investigated generative adversarial networks that can be utilized for data augmentation. The introduced networks combine image generation and compression to train the networks with the consideration of compression.

7.2 Future Work

While the proposed random forest has advantages in computational complexity, memory demand, and ability to control complexity by adjusting the number of trees and the input resolution, the accuracy of the method is limited comparing to deep learning-based methods. We discuss three potential future works that can improve accuracy. First of all, while deep neural networks can describe nonlinear representations by using a hierarchical structure with nonlinear activation functions, the representation of the method is a linear representation of the original input image at the root node. Hence, the accuracy using the random forest can be improved by learning nonlinear representations. Second, while deep neural networks are usually trained end-to-end by using back-propagation, the proposed method is trained in-order from a parent node to child nodes. As all the nodes are trained only once in-order, overall optimization might be relatively distant from global optima. Hence, iteratively updating the learned parameters might improve accuracy. Lastly, increasing the size of a tree can improve the accuracy at the cost of processing time and memory usage. However, training a deeper tree is quite computationally expensive since the number of nodes increases exponentially as the depth increases. Hence, it can be valuable to study training
deeper trees efficiently.

The depth-adaptive deep neural network can be improved by utilizing interpolated filters rather than sparse filters. As digital images are discrete in a spatial domain, the receptive fields of depth-adaptive representations need adjustments to discrete locations. While the proposed method selects a location for each offset point in a receptive field, the method can be improved by considering multiple nearby points by using a decent interpolation method.
Bibliography


