TOWARDS REAL-TIME OBJECT DETECTION
ON EDGE WITH DEEP NEURAL NETWORKS

A Dissertation presented to
the Faculty of the Graduate School
at the University of Missouri

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

by

ZHI ZHANG
Dr. Zhihai He, Dissertation Supervisor
December 2018
The undersigned, appointed by the Dean of the Graduate School, have examined the
dissertation entitled:

TOWARDS REAL-TIME OBJECT DETECTION
ON EDGE WITH DEEP NEURAL NETWORKS

presented by Zhi Zhang,
a candidate for the degree of Doctor of Philosophy and hereby certify that, in their opinion,
it is worthy of acceptance.

__________________________________________
Dr. Zhihai He

__________________________________________
Dr. Guilherme DeSouza

__________________________________________
Dr. Dominic Ho

__________________________________________
Dr. Jianlin Cheng
ACKNOWLEDGMENTS

This is the perfect time to replay my memories during my doctorate life, and I suddenly recall so many people to thank. First and foremost, I would like to sincerely thank my advisor Dr. Zhihai He, who guided me into the research community. I am fortunate enough to have Dr. He’s professionalism backing my research and study. Dr. He is an awesome friend and tutor as well, which is warm and nice considering I am studying ten thousand miles away from home. I would also like to express my deep gratitude to Dr Guilherme DeSouza, Dr. Dominic Ho, Dr Jianlin Chen, for being so supportive committee members. Besides, it is a good time to sincerely thank professors and department faculties Dr. Tony Han, Dr. Michela Becchi, Dr. James Keller and so many more, who teach me knowledge and help me walk through the doctoral degree.

And of course, how can I forget my dear colleagues and friends in Mizzou and Columbia: Xiaobo Ren, Yifeng Zeng, Chen Huang, Guanghan Ning, Zhiqun Zhao, Yang Li, Hao Sun and Hayder Yousif. I would like to emphasis my appreciation working and playing with you guys, thank you!

Also I would like to thank all members in Amazon AI team I worked with during my summer internship. Special thanks should be given to my supervisor Mu Li and mentor Eric Xie, for being so nice and supportive during my first attempts to get into the industry.

Last but not least, I want to dedicate my dissertation to my dear parents for their love. Without their spiritual support I cannot make it so far.
# TABLE OF CONTENTS

ACKNOWLEDGMENTS .......................................................... ii

LIST OF TABLES ........................................................... vi

LIST OF FIGURES ........................................................... viii

ABSTRACT ................................................................. xi

CHAPTER .................................................................

1 Introduction ............................................................... 1

1.1 Object Detection ....................................................... 1

1.2 Object Detection with Deep Neural Network ..................... 3

1.3 Deep Models on Edge Devices ..................................... 5

1.4 Summary ........................................................... 6

2 Wild Animal Detection in Video Sequences ....................... 7

2.1 Introduction ........................................................... 7

2.2 Related Work ........................................................ 10

2.2.1 Foreground-Background Segmentation ....................... 11

2.2.2 Region Proposals and Object Detection Using DCNN Methods ... 12

2.2.3 Image Verification ............................................. 12

2.3 Methods ............................................................. 13

2.3.1 Video Object Graph Cut ..................................... 14

2.3.2 Iterative Embedded Graph Cut for Animal Object Proposals .... 14

2.3.3 Cross-Frame Animal Object Verification ..................... 17

2.3.4 Learning the Animal-Background Verification Model ........... 20

2.3.5 Animal-Background Verification Model Training ............ 23
4.4 Confidence Analysis and Decision Policy Learning .......................... 73
4.5 Experimental Results ............................................................. 76
  4.5.1 Experimental Results on the CIFAR-10 Dataset ...................... 79
  4.5.2 Experimental Results on the ImageNet ................................. 80
  4.5.3 Experimental Results on the Sports-1M Dataset .................... 82
4.6 Discussion .................................................................................. 83
4.7 Conclusion ................................................................................. 86

5 Real-time Deep Object Detection Models on Edge Devices ............... 88
  5.1 Introduction ............................................................................... 88
  5.2 Challenges ............................................................................... 90
    5.2.1 Hardware limitations .......................................................... 90
    5.2.2 Missing Operator Support .................................................. 91
    5.2.3 Post-processing ................................................................. 92
  5.3 Methods ................................................................................... 93
    5.3.1 Truly End-to-end Neural Network for Object Detection ......... 94
    5.3.2 Operator Decomposition ..................................................... 97
    5.3.3 Graph-level Optimization ................................................... 97
    5.3.4 On-device Optimization ....................................................... 99
  5.4 Experimental Results ............................................................. 99
  5.5 Conclusion ................................................................................. 102

6 Summary and concluding remarks ..................................................... 103

BIBLIOGRAPHY ............................................................................... 105

VITA .................................................................................................. 127
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Performance comparison on Object Proposals in the Camera Trap dataset with other methods</td>
<td>27</td>
</tr>
<tr>
<td>2.2 Recall comparison on Camera Trap dataset</td>
<td>29</td>
</tr>
<tr>
<td>2.3 Precisions comparison on Camera Trap dataset</td>
<td>30</td>
</tr>
<tr>
<td>2.4 F-scores comparison on Camera Trap dataset</td>
<td>31</td>
</tr>
<tr>
<td>2.5 Performance comparison on Change-Detection 2014 dataset using three categories</td>
<td>33</td>
</tr>
<tr>
<td>2.6 Average processing time per image in second with various experimental choices</td>
<td>34</td>
</tr>
<tr>
<td>3.1 CIFAR-10 accuracy and network capacity comparisons with state-of-the-art methods</td>
<td>53</td>
</tr>
<tr>
<td>3.2 PASCAL VOC 2007 test object classification performances comparison</td>
<td>54</td>
</tr>
<tr>
<td>3.3 Test error rate comparisons between experimental settings and baseline methods</td>
<td>56</td>
</tr>
<tr>
<td>3.4 Network configurations for extensive benchmarks on Omniglot dataset</td>
<td>57</td>
</tr>
<tr>
<td>4.1 Network Configurations for CIFAR-10 and ImageNet Datasets</td>
<td>77</td>
</tr>
</tbody>
</table>
4.2 Fully inferred ProgNet performance and other state-of-the-art models on CIFAR-10 classification. MACC stands for the number of composite Multiply-Accumulation operations. Error rates reported in parentheses of ProgNets are post-controller results.  

4.3 Complexity-accuracy comparison between ProgNet and existing networks on the ImageNet dataset.  

4.4 Video Frame Classification Results on Sports-1M Dataset  

5.1 Theoretical Billion Ops Estimations.  

5.2 Operators used by Object Detection Networks.  

5.3 Comparison between E2E net and Tnet with Faster-RCNN.  

5.4 Decomposed Operators used by Object Detection Networks.  

5.5 Object detection model real on-device inference speed.
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Samples of camera-trap images.</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>Samples of camera-trap images.</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>Examples of bounding box proposals from spatial only and spatial-temporal</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>methods.</td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>Initial foreground segmentation with different energy level parameters.</td>
<td>17</td>
</tr>
<tr>
<td>2.4</td>
<td>Pairs of image patches for verification.</td>
<td>18</td>
</tr>
<tr>
<td>2.5</td>
<td>Work-flow of animal verification</td>
<td>19</td>
</tr>
<tr>
<td>2.6</td>
<td>Background modulation using MSDE.</td>
<td>21</td>
</tr>
<tr>
<td>2.7</td>
<td>Proposed feature extraction module</td>
<td>22</td>
</tr>
<tr>
<td>2.8</td>
<td>Example ground-truth images from CameraTrap dataset</td>
<td>25</td>
</tr>
<tr>
<td>2.9</td>
<td>Examples of segmentation and verification results on the CameraTrap dataset</td>
<td>27</td>
</tr>
<tr>
<td>2.10</td>
<td>Examples of final animal object detection and segmentation results.</td>
<td>28</td>
</tr>
<tr>
<td>2.11</td>
<td>Effect of number of frames used in validation on the overall F-score</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>performance.</td>
<td></td>
</tr>
<tr>
<td>2.12</td>
<td>Segmentation results comparison with state-of-art methods on Change</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>Detection dataset 2014</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Knowledge Projection system overview.</td>
<td>38</td>
</tr>
<tr>
<td>3.2</td>
<td>KPN architecture.</td>
<td>42</td>
</tr>
</tbody>
</table>
3.3 Convolution bottleneck structures .................................................. 43
3.4 Candidate Routes of Knowledge Projection. ................................. 48
3.5 Example images of different datasets ................................. 55
3.6 Network capacity and performance analysis. ................................. 58
3.7 Iterative pruning analysis. ................................................................. 59

4.1 **Left:** Validation results of different models, images are sorted by average top-1 accuracies across models. **Right:** Individual image difficulties across different models. ................................................................. 63

4.2 Each prediction result is produced by pre-trained model of different size. Top row: hard example which requires full network inference. Bottom row: easy example that is predicted confidently by even tiny models. **Best viewed in color.** ................................. 64

4.3 The proposed framework for progressive deep neural networks. .......... 69

4.4 ProgNet prototype structures of parallel(left) and serial(right) for image classification. Spatial reduction cells are omitted in figures for simplicity. ................................................................. 72

4.5 Early termination control during inference by RNN controller. .......... 74

4.6 Error rate versus actual average inference speed on CIFAR-10 validation set. Black squares indicate results from previously published networks running on our machine. 78


4.8 Complexity-accuracy performance comparison between our CADP controller and random termination. ................................................................. 82

4.9 Termination performance of ProgNet-s6-dense on ILSVRC12 validation set. For reference, prob is using raw softmax probability for decision by comparing maximum class probability with threshold $T$, and random uses a Bernoulli distribution with 0.5 probability as decision rule. ......................... 84

4.10 Distribution of terminated extremely easy and hard images in each stage of ProgNet-s6-dense model on ILSVRC12 validation set. ......................... 85
4.11 Easy(left) vs. Hard(right) examples in ImageNet. Visually there is no significant difference between easy and hard examples. Best viewed in color. 85

4.12 Most activated CIFAR-10 validation images from different stages in ProgNet-p6-residual. Left: most activated images of stage 0(first) 17th layer filters. Left: most activated images of stage 5(last) 17th layer filters. Best viewed in color. 87

5.1 COCO object bounding box detection mean average precision leader in recent years, comparing with baseline. 89

5.2 CPU, GPU and TPU-like accelerators require different on-chip memory architectures and compute primitives [1]. 90

5.3 Non-maximum Suppression is a commonly used post-processing step to remove duplicate detection. 93

5.4 Operator decomposition example: sigmoid. 97

5.5 Graph-level operator fusing strategy. 98

5.6 On-device optimization with TVM. [1] 99

5.7 Inference time comparison. 100

5.8 Inference FPS comparison. 101
ABSTRACT

Despite being a core topic for more than several decades, object detection is still receiving increasing attentions due to its irreplaceable importance in a wide variety of applications. Abundant object detectors based on deep neural networks have shown significantly revamped accuracies in recent years. However, it’s still the day one for these models to be effectively deployed to real world.

In this dissertation, we focus on object detection models which tackle real world problems that are unavailable few years ago. We also aim at making object detectors on the go, which means detectors are not longer required to be run on workstations and cloud services which is latency unfriendly. To achieve these goals, we addressed the problem in two phases: application and deployment. We have done thoughtful research on both areas. Our contribution involves inter-frame information fusing, model knowledge distillation, advanced model flow control for progressive inference, and hardware oriented model design and optimization. More specifically, we proposed a novel cross-frame verification scheme for spatial temporal fused object detection model for sequential images and videos in a proposal and reject favor. To compress model from a learning basis and resolve domain specific training data shortage, we improved the learning algorithm to handle insufficient labeled data by searching for optimal guidance paths from pre-trained models. To further reduce model inference cost, we designed a progressive neural network which run in flexible cost enabled by RNN style decision controller during runtime. We recognize the awkward model deployment problem, especially for object detection models that require excessive customized layers. In response, we propose to use end-to-end neural network which use pure neural network components to substitute traditional post-processing operations. We also applied operator decomposition and graph level and on-device optimization towards real-time object detection on low power edge devices. All these works have achieved state-of-the-art performances and converted to successful applications.
Chapter 1

Introduction

1.1 Object Detection

For animals to survive, the ability to detect environmental surroundings and the capability to spot both predator and prey are crucial and inevitable. Therefore, detecting and recognizing objects is one of the most important use-case of natural visual perception, and as a result, evolved as highly sophisticated systems. Indeed, human can distinguish more than 30000 visual categories, and can detect objects in a few hundred millisecond even in complex scenes [2].

Although object detection in computer vision domain is nowhere close to human performance in many aspects, it is still a long-lasting hot topic due to its inherent significance, needless to mention it is a fundamental component and enabling step in a broad range of tasks (Visual Tracking, Pose Estimation, Event Detection) and applications (Self-checkout grocery, Self-driving car, Smart-surveillance).

Object detection is a composite task, its fundamental sub-tasks are classification and localization. Image classification is probably the most well-known problem in computer vision. It consists of classifying an image into one class out of many different categories.
Localization task finds the location of a single object inside an image. If not specified, object detection usually means multiple object detection, i.e., it is required to find all objects in a single image, any missing object or inaccurate localization will be regarded as a failure. Iterating over the problem of localization plus classification we end up with the need for detecting and classifying multiple objects at the same time. Object detection is the problem of finding and classifying a variable number of objects on an image. The important difference is the “variable” part. In contrast with problems like classification, the output of object detection is variable in length, since the number of objects detected may change from image to image. Therefore, object detection is a challenging task.

Historically, the variable number of detection results has been tackled using a sliding window based approach, generating the fixed-sized features of that window for all the different positions of it. After getting all predictions, some are discarded and some are merged to get the final result by mechanisms like Non-Maximum-Suppression. In this step, another big challenge is the different conceivable sizes of objects that require special handling of feature extractions. A spatial pyramid is a traditional way to handle the sizing issue. A third challenge is solving classification and localization problems at the same time. With the tangling of these challenges, research society was moving slowly a few years ago. The most worth-noting algorithm is Viola-Jones object detection framework [3]. The approach adopt haar-like features and multi-scale sliding window which is fast and relatively simple, allowing real-time detection tasks even on lower end devices.

The evolution of object detection algorithms comes along with the emergence of better feature descriptors and more efficient machine learning algorithms. Among them, deep convolutional neural networks are the game changer.
SIFT [4] and HOG [5] and similar handcrafted features once were considerably the major building blocks of visual recognition tasks. SIFT and HOG are lower level block-wise orientation histograms, a representation we could associate with the biological structured visual pathway. However, we would expect a hierarchical, multi-stage processes for computing more informative features as hinted by animals. Convolutional Neural Networks (CNNs) saw heavy use in the 1990s [6], but then shaded under the rise of support vector machines. Starting from 2012 [7], CNNs triggered interests of researchers again by showing competitive image classification accuracy on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [8]. Together with the rekindled interest in CNNs is the debates of the generalization of features generated by CNNs. Ross et al. [9] first answered this question by bridging the gap between image classification and object detection. The RCNN method took external ROI proposals and run classifiers on cropped image by neural networks. This approach is expensive because thousands of highly overlapped proposals need to be cropped and classified, introducing significant amount of duplicate computa-
tions. However, the dramatically improved performance on PASCAL VOC [10] proved the effectiveness of CNNs and thus enlightened a series of consecutive work. Fast-RCNN [11] alleviated the duplicate computation by pushing the entire image once through a pre-trained neural network feature extractor and perform cropping on the cached feature map multiple times. The duplicate part now reside on fully-connected classifier for classification and bounding-box refinement, which is no longer the bottleneck compared to base convolutional feature extractor. While Fast RCNN have one more step towards efficient object detection, recent work Faster-RCNN [12] have shown that it is possible to generate proposals directly using a neural network, make it even faster and end-to-end capable detection framework. Faster-RCNN introduces a component called "anchors" (or "priors", "default boxes" in similar work). Anchors are bounding boxes centered at each pixel on feature map, with preset sizes. A network is trained to predict two predictions for each anchor: 1) a discrete class prediction, 2) a continuous prediction (usually 4 dimension, e.g. x-y-w-h) of an offset by which the corresponding anchor needs to be shifted to the ground-truth object bounding box. With preset anchors, convolutional predictor can generate proposals that populate across the entire image, the generated boxes substitute the traditional proposal ROIs, to be evaluated during the second step in RCNN approaches. Unlike Faster-RCNN which uses convolutional predictors as proposal generator, SSD [13] and YOLOv2 [14] skipped time-consuming cropping and classification branches in RCNN series, and sorely use convolutional predictors to generate final bounding box predictions.

The performance of CNNs has been rapidly improving. The original CNNs used in 2012 [7] was merely a stack of several convolutional layers, in comparison, the modern CNNs can reach more than 1000 layers on top of the latest building blocks of ResNet [15], DenseNet [16]. Deep CNNs now can beat human performance in image classification tasks by providing sufficient training data. Object detection systems such as RFCN [17] and RetinaNet [18] built on these extremely deep CNNs illustrate extraordinary performances that almost doubles (from 40+ to 80) the mean average precision achieved by ensembles of
multiple handcrafted systems on PASCAL VOC object detection tasks. The current state-of-the-art models are capable of providing accurate and meaningful detection results on wild scenes as depicted in Figure 1.1 which is a clip from an street surveillance video.

1.3 Deep Models on Edge Devices

The previously mentioned object detection models in recent years are good enough to be deployed in consumer products. However, as a result of significantly increased computational cost in inferencing deep CNNs, deploying to lower-end devices can be troublesome. As a reference, a popular Faster-RCNN model built on VGG-16 [19] can roughly achieve 12 images per second on a powerful Nvidia Titan X Pascal GPU whose power consumption is 250 watt per hour. A more efficient YOLO model can achieve 60 FPS on the same GPU. If we consider the whopping 11 Teraflops of single precision and do a simple math, it is easy to understand that mobile devices will require seconds to minutes to detect a single image.

    Edge computing is a method of optimizing cloud computing systems by performing data processing at the edge of the network, near the source of the data. Considering the explosive amount of IoT sensors, surveillance cameras and mobile phones, object detection on edge devices is a overwhelming trend because it is secure, latency free and bandwidth unrelated.

    A key research topic is to reduce the inference time of deep CNNs. Depending on network structure, convolutional layers consist of the majority of computational cost ranging from 80 % to 95 %. Therefore, speeding up convolution is the optimization target. One straight-forward way to accelerate without modifying network structure is to compress or optimize computation. Available choices include Fast Fourier Transform (FFT), separable convolution kernels, quantization, tensor decomposition, weight pruning and proposing new architectures with less computations. These methods can work jointly and thus pro-
duce promising results and don’t require training new models. Compressing an existing network by replacing it with a small network to fulfill the same functionality is an important and interesting orientation, which will be discussed in Chapter 3 in more details. Another approach is to explore the computational limitation and develop a faster network that matches the resource restrictions (Flops, latency, model size) for specific tasks. For example, Mobilenets [20] and ShuffleNets [21] successfully reduced computation cost by roughly 9-10x with merely 1% of accuracy drop in ImageNet challenge compared to state-of-the-art CNNs by exploring grouped computation, which is preferable for inferencing on mobile devices.

1.4 Summary

We recognize the emerging demand of high performance and high throughput object detection systems, and introduce a series of work to tackle this problem. We first introduce object detection for animal detection in wild, showing significantly improved performance using deep CNNs. Second, we present a transfer knowledge projection network to train better and faster CNNs on datasets with limited training data. Finally, we propose a progressive convolutional neural network that is designed to handle speed-accuracy trade-off of image classification, object detection and anything relying on deep CNNs.
Chapter 2

Wild Animal Detection in Video Sequences

2.1 Introduction

Wildlife monitoring with camera-trap networks, especially with the collaborative efforts of citizen scientists, enable us to collect wildlife activity data at large space and time scales and to study the impact of climate change, habitat modification and human disturbance on species richness and biodiversity along the dimensions of scale, region, season, and species [22]. Camera-traps are stationary camera-sensor systems attached to trees in the field. Triggered by animal motion, they record short image sequences of the animal appearance and activities associated with other sensor data, such as light level, moisture, temperature, and GPS sensor data. They are an important visual sensor for wildlife that can record animal appearance without disturbance [23]. Due to their relatively low cost, rapid deployment, and easy maintenance, camera traps are now being extensively used in wildlife monitoring, with the potential to be deployed at large scales in space and time. From camera-trap images, we can extract a rich set of information about animal appearance, biometric features, species, behaviors, their resource selection, as well as important environmental
features about the surrounding habitats \[24\]. During the past several years, a vast amount of camera-trap data has been collected, far exceeding the capability of manual image processing and annotation by human. There is an urgent need to develop animal detection, segmentation, tracking, and biometric feature extraction tools for automated processing of these massive camera-trap datasets. In this work, we focus on accurate and reliable animal object detection and segmentation from camera-trap images.

Detecting and segmenting moving objects from the background is an important and enabling step in intelligent video analysis \[25, 26\]. There is a significant body of research conducted during the past two decades on background modeling and foreground object detection \[27, 28, 29\]. However, the availability of methods that are robust and generic enough to handle the complexities of natural dynamic scenes is still very limited \[30\]. Videos captured in natural environments represent a large class of challenging scenes that have not been sufficiently addressed in the literature \[26\]. These types of scenes are often highly cluttered and dynamic with swaying trees, rippling water, moving shadows, sun spots, rain, etc. It is getting more complicated when natural animal camouflage added extra complexity to the analysis of these scenes. Fig. 2.1 shows some examples of image sequences captured by camera-traps at days (with color images) and nights (with infrared images). Here, each column represents a camera-trap image sequence triggered by animal motion. The key challenge here is how to establish effective models to capture the complex background motion and texture dynamics while maintaining sufficient discriminative power to detect and segment the foreground animals. Traditional motion-based techniques are not suitable here since the background is highly dynamic.

Recently, approaches based on deep neural networks, such as RCNN \[31\] and its variations Fast-RCNN \[32\] and Faster-RCNN \[12\], are achieving the state-of-the-art performance in object detection. Typically, these methods have two major components: (1) object region proposal which scans the whole image to generate a set of candidate image regions (or bounding boxes) at different locations and scales that could possibly contain the target
objects, and (2) image classification which determines if these proposed regions are truly the objects or not. We observe that, within the context of animal detection from camera-trap images, these methods suffer from two major issues: speed and accuracy. First, the natural scenes in camera-trap images are highly cluttered. Existing object region proposal methods [33, 34] often generate a large number (thousands) of candidate object regions. We know that the deep convolutional neural network (DCNN) for region classification is computationally intensive and, more importantly, it needs to be performed thousands of times for each of these proposed object regions. Therefore, it is critical to consider the unique characteristics of camera-trap images in the spatiotemporal domain and design a new and
efficient object region proposal method which can generate a small number of animal object proposals. To this end, we develop an Iterative Embedded graph Cut (IEC) method with different foreground/background cut-off energy levels to create an embedded group of objects regions for the camera-trap image sequences. The examples in Fig. 2.2 show that IEC could significantly reduce the number of proposals while maintaining a sufficiently high object coverage rate. Second, we find that the direct application of DCNN to object regions in a single image is not efficient for animal-background classification. The performance can be significantly improved by extending the classification into the temporal domain using our proposed cross-frame patch verification method. Furthermore, for efficient animal object region classification, we find that a combination of DCNN and hand-crafted features achieves better classification performance. Our extensive experimental results demonstrate that the proposed method significantly improves the performance while maintaining low computational complexity.

Figure 2.2: Examples of bounding box proposals from spatial only and spatial-temporal methods. Each bounding box denotes a candidate object region. Top: selective search algorithm. Bottom: proposed iterative embedded graph cut. Columns represent different scenes.

2.2 Related Work

This work is closely related to foreground-background segmentation, image verification, object region proposal, object detection and image classification. In the following, we
provide a review of related work on these topics.

### 2.2.1 Foreground-Background Segmentation

Early work on background subtraction often operated on the assumption of stationary background. Several methods model the background explicitly, assuming a bootstrapping phase where the algorithm is presented with frames containing only the background [35, 36]. The use of multiple hypotheses to describe the behavior of an evolving scene at the pixel level significantly improves the performance of background modeling and subtraction [36]. Elgammal *et al.* [37] used a non-parametric background model to achieve better accuracy under the same constraints as the mixture of Gaussians. Sheikh and Shah incorporate the temporal and spatial consistencies into a single model [25]. Oliver *et al.* [38] focused on global statistics rather than local constraints to create a small number of eigen-backgrounds to capture the dominant variability of background. Considering spatial context and neighborhood constraints, graph cut optimization has achieved fairly good performance in image segmentation [29]. Iterated graph cut is used in [28] to search over a nonlocal parameter space. Background cut is proposed in [39] which combines background subtraction and color or contrast-based models.

To handle background motion, various dynamic background texture models have been developed [40, 36]. Principal component analysis and autoregressive models are used in [38]. Wiener filters are used to predict the expected pixel value based on the past $K$ samples. To reduce the computational complexity, Kahl *et al.* [41] demonstrated that using eigen-background on patches in an image is sufficient to capture the variance in dynamic scenes. In [42], for each pixel, it builds a codebook. Samples at each pixel are clustered into the set of codewords based on a color distortion metric. Gregorio *et al.* [43] use a weightless neural network to model the change in background. St-Charles and Bilodeau *et al.* [44] introduce a new strategy to tackle the problem of non-stationary background with pixel-level feedback loops to balance the local segmentation sensitivity automatically.
We recognize that, for accurate and robust video object detection and segmentation in dynamic scenes, background modeling of the dynamic pixel process at the image patch level, spatial context analysis and graph cut optimization at the region-level, and embedded foreground-background classification at the sequence level should be jointly considered. In this work, we propose to establish a new framework which tightly integrates these three important components for accurate and robust video object cut in highly dynamic scenes.

### 2.2.2 Region Proposals and Object Detection Using DCNN Methods

Recent studies [45, 46] have shown the extraordinary performance of Deep Convolutional Neural Networks (DCNNs) on image classification, object detection and recognition. To speed up the DCNN-based object detection process and avoid scanning of the whole image, object proposal methods have been recently developed for predicting object bounding boxes [12, 47, 48, 49, 50]. Szegedy et al. [47] used a deep neural network as a regression model to predict the object bounding box. Sermanet et al. [49] developed a fully connected layer that is trained to predict the box coordinates for the localization task that assumes a single object. The fully connected layer is then turned into a convolutional layer for detecting multiple class-specific objects, which won the ILSVRC2013 localization competition. The original work on MultiBox [48] also used deep neural networks. Instead of producing bounding boxes, the MultiBox approach generates region proposals from a network whose last layer simultaneously predicts multiple class-agnostic boxes.

### 2.2.3 Image Verification

This work is also related to image verification. Image verification, in our particular problem, is regarded as a two-class classification problem: to verify if a proposed object image patch is an animal or belongs to the background scene. Classic learning-based image verification often involves two major steps: feature representation and distance or met-
Learning. Features used for image verification include colors, Histogram of Oriented Gradients (HOG), Haar-like descriptors, SIFT or SURF key point descriptors, Maximally Stable Color Regions (MSCR), texture filters, differential local information, co-occurrence matrices, etc [51]. Statistics of low-level features, such as Bag of Words (BoW) descriptors, are also used for image verification to handle spatial variations. Recently, Fisher vectors [52] are developed which provides a better model to encode the local features. A number of methods built upon this Fisher vector approach [53, 54] have shown outstanding performance in image representation.

In this work, we propose to develop an effective cross-frame image verification method to determine if an image patch belongs to the background or not. This problem becomes very challenging since the background is highly dynamic and cluttered. In this work, we will demonstrate that a combination of DCNN features and hand-crafted image features specifically designed for camera-trap data is able to achieve significantly improved performance in animal image patch verification.

2.3 Methods

In this work, we propose to develop a new and efficient animal object proposal method using iterative embedded graph cut (IEC) which exploit animal motion and temporal correlation in the camera-trap image sequence. Using a sequence of energy levels, the proposed IEC method is also able to address the over-proposal issue in existing methods [33]: generating too many object proposals in highly cluttered scenes. In our previous work [55], we have developed video object graph cut method, which achieved the state-of-the-art performance on object segmentation from videos with dynamic scenes and outperformed existing methods by up to 12% in segmentation accuracy on the 2012 Change Detection Challenge dataset [55]. In this work, built upon this method, we propose to develop an embedded graph cut method for animal object proposal.
2.3.1 Video Object Graph Cut

The video object graph cut algorithm developed in our previous work [55] constructs background models using HOG (histogram of oriented gradients) and BoW (bag of words) features. It constructs a foreground salience graph (FSG) to characterize the salience of an image patch in the spatio-temporal domain. It then formulates the object segmentation as an energy minimization problem which can be solved using the graph cut method. The FSG consists of two components: temporal salience and spatial salience. The temporal salience measures the dis-similarity $d(P_{(x,y)}, P_{(x,y)k})$ between the current image patch $P_{(x,y)}$ at location $(x, y)$ and the background model. The spatial salience measure between the dis-similarity between the current patch and its neighboring patches. Based on the temporal and spatial salience measures, we construct the foreground salience graph. We represent the image by an 8-connectivity undirected graph. A graph cut minimization procedure is used to find the segmented foreground object.

2.3.2 Iterative Embedded Graph Cut for Animal Object Proposals

We recognize that, in cluttered scenes of camera-trap images, the initial segmentation often yields incorrect segmentation results and object contours. For example, in the Camera-trap dataset, we find that some parts of the animal body are well segmented in some video frames but poorly segmented in other frames since the foreground object has moved to different background regions. Motivated by this observation, we propose to propagate the foreground-background segmentation information across frames, helping each other to refine the segmentation in an iterative manner. More specifically, from the existing foreground-background segmentation results of all frames (typically about 10 frames in each camera-trap sequence), we estimate the foreground-background probability map for
each frame. Specifically, for each pixel \( x \), its foreground probability is defined as

\[
p(x) = 0.5 + \frac{1}{\pi} \tan^{-1}(\beta \cdot s \cdot d[x]),
\]

where \( d[x] \) is the minimum distance from the pixel \( x \) to the boundary of the foreground regions, \( s \) is 1 if \( x \) is a foreground pixel, otherwise \( s \) is -1. \( \beta \) is a constant controlling the transition range. In this work, we set \( \beta \) to be 0.5. For each image patch \( P \) at location \( x_P \), we use this probability map to modulate the foreground salience graph for graph cut of the new iteration.

It should be noted that, when performing the video object graph cut, there is an important parameter \( \alpha \) which controls the level of penalty for foreground-background classification errors.

\[
E_p(x_p) = \begin{cases} 
\frac{D^t(p)}{\sigma}, & x_p = 0; \\
\alpha(1 - \frac{D^t(p)}{20\sigma}), & x_p = 1; 
\end{cases}
\]

Here \( E_p \) is the T-link graph-cut energy term, \( D^t(p) \) is the temporal saliency at position \( x_p \), as described in [55]. Intuitively, \( \alpha \) serves as a foreground energy level parameter, it is simply the weight controlling of how many positive cut error are counted towards the objective function, for example, if \( \alpha = 0.5 \), then only half of graph-cut energy is generated. From the animal segmentation example in Fig. 2.3, we can see that, as \( \alpha \) decreases, the graph-cut produces more and larger patches, classifying more pixels into the foreground and thus increasing the probability for the animals to be covered by these foreground regions. From our experiments, we recognize that it is difficult to determine a fixed optimal setting for the value of \( \alpha \) since different camera-trap image sequences will require different \( \alpha \) to achieve the best segmentation results. More importantly, for the same \( \alpha \), the segmentation performance is also region-dependent: some foreground regions are over segmented while others are under segmented.

In this work, we propose to use \( \alpha \) as the control parameter to generate an embedded
set of region proposals for animal detection. More specifically, in our proposed iterative embedded graph cut scheme, we perform the video object graph cut with multiple values of the control parameter \( \alpha \), denoted by \( \{ \alpha^n | 1 \leq n \leq N \} \). \( N \) is typically 3 to 10 controlling the number of different \( \alpha \). We choose \( N = 5 \) in the experiment. Let \( M^n \) be the corresponding foreground regions. We have

\[
M^1 \subseteq M^2 \cdots \subseteq M^N.
\] (2.3)

We define patch \( P^1 = M^1 \), and \( P^n = M^n - M^{n-1} \) for \( n = 2, \cdots, N \). To increase the granularity of the proposed regions, we select those large patches from \( P = \{ P^n | 1 \leq n \leq N \} \),

\[
P_L = \{ P^k | S(P^k) > \Delta, P^k \in P \},
\] (2.4)

where \( S(P^k) \) represents the size or number of pixels in the patch, and \( \Delta \) is a threshold parameter controlling the granularity levels of the region proposals. We further segment each patch \( P^k \) in \( P_L \) into smaller ones \( \{ P^k_j \} \) of size less than \( \Delta \) using a k-mean clustering algorithm on its pixels inside with location and color features. We always use 2 clusters in an iterative fashion, which is enough for finer segmentation. We denote all small patches \( \{ P^k_j | P^k \in P \} \), plus those small patches in the original set \( P_L \), by \( p = \{ p_l | 1 \leq l \leq L \} \).

Similar to other region proposal approaches [33], we will use these small patches to generate region proposals using a neighborhood grouping procedure. Due to the bounding box constraints, we use rectangle template matching for region proposals. We denote the set of region proposals, or candidate object bounding boxes, by \( B = \{ B_i | 1 \leq i \leq I \} \).

In the previous section, using iterative embedded graph cut, we have generated a set of animal object proposals \( B \). In this section, we will develop a cross-frame animal-background verification method to score each bounding box or image patch in \( B \), determining if it is an animal or not. We observe that false positive foreground patches generated by the animal object proposal often have significantly different characteristics from true animal image patches. Fig. 2.4 (a)-(d) illustrate four examples of false positives: (a)
displacement of leaves and branches caused by wind, (b) dramatic change of sun lighting, (c) moving shadow, and (d) sensor noises in low light conditions. Fig. 2.4(e) shows one example of true animal. Here, the top and bottom rows are from two different frames with interval of 3 seconds. The task of image verification here is to determine if the image patch in the top row is the same as the one in the bottom row.

2.3.3 Cross-Frame Animal Object Verification

In Fig. 2.5, we use one example to show the work flow of our cross-frame animal-background verification. The first row (a) shows the original camera-trap images where a deer passes by. Row (b) shows the foreground-background segmentation results obtained by the ensemble graph cut method explained in Section 2.3. We can see that image patches 1 and 2 are an-
Figure 2.4: Pairs of image patches for verification. (a)-(d) are examples of false positive: (a) displacement of leaves, (b) illumination change, (c) shadow, and (d) sensor noises in low light conditions. (e) is an example of one true animal object.

imal while patch 3 is wrongly detected due to a moving shadow. To perform cross-frame animal-background verification, for each foreground patch $B_i \in B$, we extract co-located image patches across the entire sequence at the same location of the same size. We exclude those patches which have significant overlap with foreground regions in the same frame. Here, we set the max overlap ratio to be 0.25, which is the ratio between the intersection part and the size of $B_i$. We denote the remaining patches by $O_{i,m}$, $1 \leq m \leq M_i$. We will learn a pairwise verification model $V$ to determine if a pair of image patches $B_i$ and $O_{i,m}$ are similar to each other up to a certain amount of background changes, including texture dynamics, shadows, noise, lighting changes, etc. Specifically, $V(B_i, O_{i,m}) = 1$ implies that $B_i$ and $O_{i,m}$ are similar to each other. Otherwise, $V(B_i, O_{i,m}) = -1$. To form a joint decision for $B_i$, we use the following weighted summation

$$S = \sum_{m=1}^{M} w_m \times V(B_i, O_{i,m}),$$

(2.5)

where $w_m$ is the normalized weight of the $m$-th background patch. We introduce $w_m$ to accommodate the background transition, especially in a long sequence. In this work, we
choose $w_m$ as the inverse of the frame distance between $B_i$ and $O_{i,m}$,

$$w_m = \frac{1}{|N[B_i] - N[O_{i,m}]| \epsilon},$$

(2.6)

Here, $N[\cdot]$ represents the frame index of the image patch. $\epsilon \in (0, 1]$ is a parameter controlling the amount of contribution from each frame. After normalization, we then use $\{w_m\}$ as the weights to compute the verification score $S$. If $S > 0$, we determine that image patch $B_i$ belongs to the background. Otherwise, it is classified as the animal.

Figure 2.5: Work-flow of animal verification: (a) original camera-trap images; (b) foreground segmentation using ensemble graph cuts; (c)-(d): cross-frame patch verification for foreground object 1-3, respectively. To save space, foreground objects 4-5 are ignored in this figure.
2.3.4 Learning the Animal-Background Verification Model

In this section, we discuss how to train the verification model $V(B_i, O_{i,m})$ to determine if two collocated image patches from the camera-trap image sequence are similar to each other. The major challenges are: (1) this verification model should be able to accommodate large background variations between $B_i$ and $O_{i,m}$, which include background texture dynamics, displace of objects (such as tree leaves or branches), changes of lighting conditions, moving shadows or sun spots, etc. This requires that the image feature description should be invariant under these types of changes. (2) Unlike other image verification tasks, such as face verification, our animal-background verification needs to deal with image patches with a wide ranges of sizes and aspect ratios since they are directly generated by the ensemble graph cuts.

To address this challenge, we propose the following image processing and feature representation schemes for animal-background verification. As is shown in Fig. 2.7, our scheme has three major steps: (1) background modulation, (2) fine-tuned deep convolution neural network (DCNN) features; (3) adaptive histogram of oriented gradients (HOG) extraction and Fisher vector encoding.

(1) **Background modulation.** We recognize that, in camera-trap images, the background is highly cluttered and has large variations across frames due to background motion. To minimize this background interference and improve the animal-background verification performance, we propose to modulate the original image patch using the background model. We call this pre-processing step as Mean Subtracted Difference Enhancement (MSDE). In MSDE, the pair of image patches are first converted to the YCbCr color space. We then perform mean centering (scale normalization) to the absolute difference between foreground and reference background images on Y (luminance) components with all negative values clamped to zero. Note that this truncation must be coupled with mean subtracted difference image because we will normalize the entire image to 8-bit range $[0, 255]$ to stretch the contrast between object and background. Fig. 2.6 shows three examples
of the MSDE images.

Figure 2.6: Background modulation using MSDE. (a) Animal Patches; (b) Reference backgrounds; (c) Processed patches using MSDE.

(2) **DCNN features learned from the camera-trap images.** Recent results on image classification, scene recognition, fine-grain object recognition and detection, and image retrieval have demonstrated that the generic image descriptors learned from deep convolutional neural networks (DCNN) are very effective [7]. A deep convolutional neural network (DCNN) consists of multiple convolutional layers and then followed by one or more fully connected layers. Pooling methods are often used in DCNN to achieve translation invariant properties for DCNN features. In this work, we follow the architecture described by Krizhevsky et al. [7] using the caffe [56] implementation. We use the pre-trained DCNNs model by Girshick et al. [9] on large auxiliary ILSVRC 2012 dataset and perform fine-tuning on our annotated camera-trap images. During fine-tuning, we replace the 21-way
soft max output classification layer with a two-way animal-background layer. The rest five convolutional and three fully-connected layers remain the same. The input image is resized to $227 \times 227$ regardless of its original size and aspect ratio to fit in the CNN input layer. We recognize that this will change the aspect ratios of the input images and create image distortion. However, since both image patches, $P_k$ and $O_{km}$ under the verification task are experiencing the same distortion, the impact on the verification performance by this distortion is very limited. The output DCNN feature is a 4096-dimensional vector.

![Proposed feature extraction module](image)

Figure 2.7: Proposed feature extraction module: (a) pre-processing; (b) deep CNN feature extraction; (c) adaptive size HOG Fisher vector descriptor extraction.

(3) **HOG with Fisher Vector encoding.** DCNN features, self-learned from training data, provide an efficient and generic low to mid-level features that are rich enough to describe image details such as edge, shape, color distribution, etc. However, it often requires a large amount of data for training and fine-tuning to achieve desired performance levels. In practice, the amount of available training data is often limited. In this work, we explore a mixture of self-learned DCNN features and hand-crafted image features, aiming to enhance the performance of DCNN features with specially designed features for our animal-background verification. Specifically, we propose to use Fisher vector-encoded HOG (histogram of oriented gradients) features. We use HOG since it is insensitive to illumination change and is able to capture local textures. We partition the input image into the $20 \times 20$ blocks without overlapping (the input patches to our animal-background verification module will have different sizes and aspect ratios). Within each block, we extract its HOG feature.
We observe that the original HOG feature are not able to accommodate large background variations and texture dynamics. To address this issue, we propose to encode the HOG features using Fisher Vectors (FV) [57] to extract high-level invariant statistical features. The FV encoding aggregates a large set of HOG vectors into a high-dimensional vector representation by fitting a parametric generative model, e.g. the Gaussian Mixture Model (GMM) to these features, and then encoding the derivatives of the log-likelihood of the model with respect to its parameter [57]. Unlike the BoW model which approximates a feature using a pre-trained codebook, Fisher vector successfully retains extra information by soft assignment to the Gaussian components. Let $\gamma_t(k)$ be the soft assignment of the descriptor $d_t$ to the Gaussian component $k$ where $w_k$, $\mu_k$ and $\sigma_k$ are respectively the mixture weight, mean vector and covariance matrix of Gaussian $k$:

$$\gamma_t(k) = \frac{w_k \mu_k(d_t)}{\sum_{j=1}^{K} w_j \mu_j(d_t)}, \quad (2.7)$$

$G_{\mu,k}^M$ and $G_{\sigma,k}^M$ are descriptor gradients with respect to $\mu_k$ and $\sigma_k$ of the component $k$. $G_{\mu,k}^M$ and $G_{\sigma,k}^M$ can be computed as:

$$G_{\mu,k}^M = \frac{1}{T \sqrt{w_k}} \sum_{t=1}^{T} \gamma_t(k) (\frac{m_t - \mu_k}{\sigma_k}), \quad (2.8)$$

$$G_{\sigma,k}^M = \frac{1}{T \sqrt{2w_k}} \sum_{t=1}^{T} \gamma_t(k) [\frac{(m_t - \mu_k)^2}{\sigma_k^2} - 1]. \quad (2.9)$$

We concatenate $G_{\mu,k}^M$ and $G_{\sigma,k}^M$ vectors for $k = 1, 2,...,K$, resulting in a $2 \times 16 \times K$ dimensional feature, where 16 is the dimensionality of our local descriptor.

### 2.3.5 Animal-Background Verification Model Training

The FV-encoded HOG feature is a 1280-dimension vector according to our experimental setup. After normalization, we concatenate it with the 4096-dimension DCNN feature,
resulting in a 5376-dimension feature vector for the image pair under verification. To generate the training data, we have manually labeled 800 camera-trap image sequences (about 8000 images in total), placing bounding boxes around animals. For each animal patch $B_i^+$ inside the bounding box, we find their collocated image patches in the same sequence $\{O_{i,m}^+\}$ and make sure that all $O_{i,m}^+$ are from the background. Each pair of $B_i^+$ and $O_{i,m}^+$ will constitute a positive sample for training our verification model. To construct negative samples, we randomly generate bounding boxes within the background regions. For each background patch $B_i^-$, we extract it collocated patches $\{O_{i,m}^-\}$ within the background regions. Each pair of $B_i^-$ and $O_{i,m}^-$ will constitute a negative sample. The next step is to combine Fisher HOG feature with DCNN feature, followed by a classifier. Here one option is to implement a customized adaptive HOG layer in neural network, a concatenate layer to combine features with original DCNN feature, followed by a softmax loss layer. Considering SVM and softmax layer provide very close performance as classifier [31], and Fisher-HOG are not highly parallelizable, we propose to use external concatenation to train the pairwise verification model $V(B_i, O_{i,m})$ using linear SVM (Support Vector Machine).

### 2.4 Experimental Results

#### 2.4.1 Datasets

In this work, we use our Camera_trap and popular CDnet dataset for evaluation purposes. We establish the Camera-Trap dataset for performance evaluation of animal detection from highly cluttered natural scenes. We made this dataset publicly available for peer researchers: http://videonet.ece.missouri.edu/cameratrap/. In our on-going work on automated large-scale wildlife monitoring, we have collected over 1 million camera-trap images of wildlife species. This dataset consists of 800 camera-trap image sequences with 23 animal species, in both daytime color and nighttime infrared formats. These are very challenging videos
with highly cluttered and dynamic natural scenes. Spatial resolutions of the images vary from $1920 \times 1080$ to $2048 \times 1536$, and are considered as very high resolution images. The length of each sequence varies from 10 to more than 300 frames depending on the duration of the animal activities. For each image, we have manually labeled the bounding box of each animal in the scene. An overall of 6493 ground-truth foreground bounding-boxes are recorded and archived in the dataset. Ground-truth labels are stored in a plain text file, each row corresponding to one image. Fig.2.8 shows some example images from the dataset.

We also include Change Detection dataset as a reference dataset since its popular in motion detection community. More details will be covered in Section 2.4.3.

For performance evaluations, we plan to use the following performance metrics as in the CDNet [58] dataset: 1) Recall $\frac{TP}{(TP + FN)}$; 2) Precision $\frac{TP}{(TP + FP)}$; and 3) F-measure

$$F = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}} = \frac{2TP}{2TP + FN + FP}$$ (2.10)

where $TP$ is the number of true positives, $FP$ is the number of false positives, and $FN$ is the number of false negatives. True detections are those which have Intersection-over-Union(IoU) $\geq 0.5$ with ground-truth bounding boxes.
2.4.2 Experimental Setup

For background modeling with HOG and BoW, we use a patch size of $16 \times 16$. The codebook size is set to be 128. All region candidates with $\geq 0.5$ intersection-over-union (IoU) overlap with ground-truth boxes are treated as true positive patches, the rest are regarded as false positives. However, to avoid mixing possible animal object patch with negative samples, in practice, we only select $\text{IoU} = 0$ patches as negatives for training. Thanks to pre-trained object classification model, we can easily achieve more than 97% training accuracy in less than 12 hours on an Nvidia GTX Titan X GPU. We use 50% (400 sequences) as training data and the rest for testing in the following experiments. We also use Pascal VOC 2007 and 2012 detection dataset [59] as auxiliary data to train a generic object model to be used on CDnet 2014 dataset [58] since it contains objects such as cars and boats. To use Pascal dataset, we randomly sample background regions with zero overlapping to any foreground region, marked as 0. And we use 20 original classes objects as positive samples, marked as 1. For all verification models in experiments, we finetune a 2-way classification model, with a batch size 128, learning rate 0.01, momentum 0.9 and a decay of 0.0005. We continue training with a maximum iteration 40000 on camera-trap, and 80000 on Pascal VOC, respectively. When training accuracy is higher than 98% and stopped climbing for a while, we stop the training to prevent over-fitting.

2.4.3 Experimental Results

(1) Qualitative evaluations. Fig. 2.9 shows some example experimental results on four sequences from different species, which are Red Fox, Great Tinamou, European Hare and Paca, respectively. Row (a) is the original image from the camera-trap. Due to space limitations, we only include 3 out 10 images here. Row (b) shows the segmentation results after the video object graph cut. The animal body boundaries (shapes) are not very accurate and there a significant amount of incorrect segmentation results. After several iterations of
Figure 2.9: Examples of segmentation and verification results on the Camera Trap dataset with four (A-D) sequences: (a) original image in sequences; (b) initial Segmentation foreground map; (c) iterative ensemble object cut results; (d) joint object verification results; (e) Animal segmentation results using the proposed method. The required output is animal bounding boxes, however, the foreground masks are still available from graph-cut results.

Table 2.1: Performance comparison on Object Proposals in the Camera Trap dataset with other methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>80% Coverage</th>
<th>90% Coverage</th>
<th>Best Coverage (Maximum %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selective Search [33]</td>
<td>2829.7</td>
<td>5903.5</td>
<td>13882 (99.8%)</td>
</tr>
<tr>
<td>GOP [60]</td>
<td>2489.1</td>
<td>3984.6</td>
<td>9874 (98.2%)</td>
</tr>
<tr>
<td>MOP [61]</td>
<td>335.8</td>
<td>482.3</td>
<td>891.7 (96.7%)</td>
</tr>
<tr>
<td>FCOP [62]</td>
<td>132.8</td>
<td>384.2</td>
<td>393.1* (90.4%)</td>
</tr>
<tr>
<td>STODP [63]</td>
<td>146.0</td>
<td>352.8</td>
<td>418.5 (91.9%)</td>
</tr>
<tr>
<td>Proposed</td>
<td><strong>95.4</strong></td>
<td><strong>237.3</strong></td>
<td>626.9 (93.1%)</td>
</tr>
</tbody>
</table>

Cross-frame information fusion and graph cuts by utilizing existing background information, better results are achieved in row (c). We can see that false positive patches caused by background variations, such as shadows, waving leaves, and moving clouds, are still in the segmentation results. Row (d) shows the final results after animal-background verification. These false positive patches have been successfully removed. Row (e) shows the animal pixels with row (d) as the mask. We can see that the proposed method is able to achieve very accurate and reliable segmentation of the foreground animals in dynamic scenes by preserving true positives and filtering out false positives. Fig. 2.10 shows some examples of animal segmentation from highly cluttered and dynamic natural scenes.
Figure 2.10: Examples of final animal object detection and segmentation results.

(2) Quantitative Results. We first compare the performance of our animal object proposal method using iterative embedded graph-cut with the following state-of-the-art object proposal methods: Spatial-Temporal Object Detection Proposals (STODP) [63], Fully Connected Object Proposals for Video Segmentation (FCOP) [62], and Learning to Segment Moving Object in Videos (MOP) [61]. For reference purposes, we also compare the performances of single frame object proposal techniques, as proposed in Geodesic Object Proposals [60] and Selective Search [33]. The latter is very popular and is used in RCNN and fast-RCNN as the enabling proposal method. Table 2.1 provides the average number of proposal bounding box required by each method in order to cover 80%, 90% and the most ground-truth animal objects, respectively. Here, coverage is the percentage of ground-truth bounding boxes which have $IoU \geq 0.5$ with any box in detection proposal list. The best coverage rate indicates the capability of detecting all objects in every frame. Improving the
Table 2.2: Performance comparison on Camera_trap dataset. Metrics showing average recalls.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Agouti</td>
<td>0.7382</td>
<td>0.7239</td>
<td>0.8088</td>
<td>0.8105</td>
<td>0.8218</td>
<td>0.8364</td>
</tr>
<tr>
<td>Collared Peccary</td>
<td>0.8436</td>
<td>0.8516</td>
<td>0.8838</td>
<td>0.8865</td>
<td>0.9049</td>
<td>0.9202</td>
</tr>
<tr>
<td>Paca</td>
<td>0.7799</td>
<td>0.7658</td>
<td>0.797</td>
<td>0.8055</td>
<td>0.7946</td>
<td>0.8226</td>
</tr>
<tr>
<td>Red Brocket Deer</td>
<td>0.7772</td>
<td>0.7905</td>
<td>0.8492</td>
<td><strong>0.8794</strong></td>
<td>0.8587</td>
<td>0.8723</td>
</tr>
<tr>
<td>White-nosed Coati</td>
<td>0.8221</td>
<td>0.8016</td>
<td>0.8739</td>
<td>0.8883</td>
<td>0.8893</td>
<td>0.8993</td>
</tr>
<tr>
<td>Spiny Rat</td>
<td>0.6908</td>
<td>0.7016</td>
<td>0.7729</td>
<td>0.7924</td>
<td>0.789</td>
<td>0.8092</td>
</tr>
<tr>
<td>Ocelot</td>
<td>0.7935</td>
<td>0.7893</td>
<td>0.8592</td>
<td>0.8796</td>
<td>0.8732</td>
<td>0.8855</td>
</tr>
<tr>
<td>Red Squirrel</td>
<td>0.7978</td>
<td>0.7761</td>
<td>0.8682</td>
<td>0.8901</td>
<td>0.8839</td>
<td>0.8914</td>
</tr>
<tr>
<td>Common Opossum</td>
<td>0.7395</td>
<td>0.7582</td>
<td>0.8187</td>
<td>0.8456</td>
<td>0.8263</td>
<td>0.8623</td>
</tr>
<tr>
<td>Bird spec</td>
<td>0.5505</td>
<td>0.4968</td>
<td>0.6083</td>
<td>0.6188</td>
<td>0.6515</td>
<td>0.6717</td>
</tr>
<tr>
<td>Great Tinamou</td>
<td>0.6964</td>
<td>0.7247</td>
<td>0.8282</td>
<td>0.8473</td>
<td>0.8546</td>
<td>0.8699</td>
</tr>
<tr>
<td>White-tailed Deer</td>
<td>0.7847</td>
<td>0.8165</td>
<td>0.8251</td>
<td>0.8549</td>
<td>0.8403</td>
<td>0.8611</td>
</tr>
<tr>
<td>Mouflon</td>
<td>0.7788</td>
<td>0.7743</td>
<td>0.8197</td>
<td>0.8395</td>
<td>0.8429</td>
<td>0.8782</td>
</tr>
<tr>
<td>Red Deer</td>
<td>0.8555</td>
<td>0.8642</td>
<td>0.8792</td>
<td><strong>0.9052</strong></td>
<td>0.898</td>
<td>0.9008</td>
</tr>
<tr>
<td>Roe Deer</td>
<td>0.8353</td>
<td>0.8548</td>
<td>0.8853</td>
<td>0.8968</td>
<td>0.8956</td>
<td>0.9076</td>
</tr>
<tr>
<td>Wile Boar</td>
<td>0.8013</td>
<td>0.8553</td>
<td>0.8732</td>
<td>0.9018</td>
<td>0.8922</td>
<td>0.907</td>
</tr>
<tr>
<td>Red Fox</td>
<td>0.676</td>
<td>0.6548</td>
<td>0.7388</td>
<td>0.7682</td>
<td>0.7765</td>
<td>0.7933</td>
</tr>
<tr>
<td>European Hare</td>
<td>0.6695</td>
<td>0.6561</td>
<td>0.7862</td>
<td>0.7892</td>
<td>0.7983</td>
<td>0.8283</td>
</tr>
<tr>
<td>Wood Mouse</td>
<td>0.7176</td>
<td>0.6815</td>
<td>0.7972</td>
<td>0.8136</td>
<td>0.8098</td>
<td>0.8357</td>
</tr>
<tr>
<td>Coiban Agouti</td>
<td>0.6678</td>
<td>0.6915</td>
<td>0.7982</td>
<td>0.8046</td>
<td>0.8121</td>
<td>0.8221</td>
</tr>
<tr>
<td>Average</td>
<td>0.7587</td>
<td>0.7674</td>
<td>0.8251</td>
<td>0.8493</td>
<td>0.8417</td>
<td>0.8597</td>
</tr>
</tbody>
</table>

Coverage is hard and costly, which often results in a massive amount of proposal detections. We can see that our proposed method is much more efficient than existing methods at the coverage rates of 80% and 90%, and find a good trade-off between the number of proposals (which affects the subsequent verification time) and the coverage rate. The single-frame based methods, such as the Selective search and GOP often produce a large amount of proposals. The limitation of proposed IEC method is its weakness in detecting slow moving object, which could be neglected in motion triggered dataset such as camera_trap. In return, IEC is exceptionally good at filtering out non-candidate object proposals, which is a crucial for accurate animal detection.

Table 2.2, Table 2.3 and Table 2.4 provides quantitative recall, precision and F-score comparisons on our Camera_trap dataset, respectively. We compare our proposed method with the following methods: (1) EC, Ensemble graph Cuts developed in our previous work [55]; (2) YOLO, a fast unified object detection algorithm which detect objects in single inference [64]; (3) Fast-RCNN [32]; (4) Faster-RCNN [12], an improved version of Fast-
Table 2.3: Performance comparison on Camera Trap dataset. Metrics showing average Precisions.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>voc07+voc12</td>
<td>voc07+voc12</td>
<td>voc07+voc12</td>
<td>voc07+voc12</td>
<td>camera-trap</td>
<td>camera-trap</td>
</tr>
<tr>
<td>Agouti</td>
<td></td>
<td>0.7632</td>
<td>0.7593</td>
<td>0.742</td>
<td>0.7514</td>
<td>0.7875</td>
<td>0.8244</td>
</tr>
<tr>
<td>Collared Peccary</td>
<td></td>
<td>0.8209</td>
<td>0.8359</td>
<td>0.8015</td>
<td>0.8094</td>
<td>0.7682</td>
<td>0.8152</td>
</tr>
<tr>
<td>Paca</td>
<td></td>
<td>0.7969</td>
<td>0.8169</td>
<td>0.8039</td>
<td>0.8289</td>
<td>0.8122</td>
<td>0.8333</td>
</tr>
<tr>
<td>Red Brocket Deer</td>
<td></td>
<td>0.8563</td>
<td>0.8915</td>
<td>0.8517</td>
<td>0.8879</td>
<td>0.8658</td>
<td>0.8867</td>
</tr>
<tr>
<td>White-nosed Coati</td>
<td></td>
<td>0.8059</td>
<td>0.8314</td>
<td>0.7899</td>
<td>0.7952</td>
<td>0.803</td>
<td>0.8221</td>
</tr>
<tr>
<td>Spiny Rat</td>
<td></td>
<td>0.7539</td>
<td>0.7642</td>
<td>0.7193</td>
<td>0.7314</td>
<td>0.7604</td>
<td>0.7756</td>
</tr>
<tr>
<td>Ocelot</td>
<td></td>
<td>0.7918</td>
<td>0.8192</td>
<td>0.7726</td>
<td>0.7952</td>
<td>0.8011</td>
<td>0.8154</td>
</tr>
<tr>
<td>Red Squirrel</td>
<td></td>
<td>0.7345</td>
<td>0.7682</td>
<td>0.7328</td>
<td>0.7437</td>
<td>0.7638</td>
<td>0.7727</td>
</tr>
<tr>
<td>Common Opossum</td>
<td></td>
<td>0.7816</td>
<td>0.8164</td>
<td>0.7951</td>
<td>0.8155</td>
<td>0.8023</td>
<td>0.8205</td>
</tr>
<tr>
<td>Bird spec</td>
<td></td>
<td>0.6527</td>
<td>0.7465</td>
<td>0.6412</td>
<td>0.6619</td>
<td>0.6898</td>
<td>0.7228</td>
</tr>
<tr>
<td>Great Tinamou</td>
<td></td>
<td>0.789</td>
<td>0.8349</td>
<td>0.8035</td>
<td>0.8148</td>
<td>0.8313</td>
<td>0.8441</td>
</tr>
<tr>
<td>White-tailed Deer</td>
<td></td>
<td>0.8218</td>
<td>0.8432</td>
<td>0.8303</td>
<td>0.8792</td>
<td>0.8551</td>
<td>0.8671</td>
</tr>
<tr>
<td>Mouflon</td>
<td></td>
<td>0.7594</td>
<td>0.8448</td>
<td>0.7692</td>
<td>0.7846</td>
<td>0.7922</td>
<td>0.8107</td>
</tr>
<tr>
<td>Red Deer</td>
<td></td>
<td>0.7947</td>
<td>0.8214</td>
<td>0.7963</td>
<td>0.7991</td>
<td>0.8234</td>
<td>0.8391</td>
</tr>
<tr>
<td>Roe Deer</td>
<td></td>
<td>0.7969</td>
<td>0.8391</td>
<td>0.7793</td>
<td>0.7925</td>
<td>0.8022</td>
<td>0.8218</td>
</tr>
<tr>
<td>Wile Boar</td>
<td></td>
<td>0.7863</td>
<td>0.8417</td>
<td>0.7965</td>
<td>0.805</td>
<td>0.8131</td>
<td>0.8282</td>
</tr>
<tr>
<td>Red Fox</td>
<td></td>
<td>0.6471</td>
<td>0.7349</td>
<td>0.6752</td>
<td>0.6849</td>
<td>0.7056</td>
<td>0.7358</td>
</tr>
<tr>
<td>European Hare</td>
<td></td>
<td>0.7156</td>
<td>0.7514</td>
<td>0.7391</td>
<td>0.7485</td>
<td>0.753</td>
<td>0.772</td>
</tr>
<tr>
<td>Wood Mouse</td>
<td></td>
<td>0.7094</td>
<td>0.7539</td>
<td>0.7293</td>
<td>0.7336</td>
<td>0.7493</td>
<td>0.7632</td>
</tr>
<tr>
<td>Coiban Agouti</td>
<td></td>
<td>0.7316</td>
<td>0.7815</td>
<td>0.749</td>
<td>0.7598</td>
<td>0.7732</td>
<td>0.7778</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td>0.7824</td>
<td>0.8315</td>
<td>0.7801</td>
<td>0.7886</td>
<td>0.8017</td>
<td>0.8209</td>
</tr>
</tbody>
</table>

RCNN which propose candidates using fully convolutional network; (5) the IEC+DCNN method which performs animal-background verification using the DCNN features only; and (6) the proposed method that uses joint DCNN and FV-HOG features. Note that models in (2)-(4) are trained on Pascal VOC dataset (convolutional layers are pre-trained on imagenet [65] for better convergence) and fine-tuned on camera-trap dataset. The last column shows the performance of the method developed in this work which uses combined FV-HOG and DCNN features for animal-background verification. We can see that the animal-background verification significantly improves the performance, either with FV-HOG or DCNN features. The verification module can significantly reduce the false positive rates while maintaining high true-positive rates. The average F-Score is 83.98%, almost 7% higher than the original 77.03%. The average precision is also improved by a large margin from 78.24% to 82.09%. We also conclude that it is more efficient to combine FV-HOG and DCNN features and they can enhance the performance of each other. The proposed framework outperforms the fine-tuned Faster-RCNN [12] by up to 4%. With limited training
Table 2.4: Performance comparison on Camera Trap dataset. Metrics showing average F-scores.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Agouti</td>
<td>0.7505 0.7436</td>
<td>0.7783 0.7825</td>
<td>0.8043</td>
<td>0.831</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
<tr>
<td>Collared Peccary</td>
<td>0.8321 0.8246</td>
<td>0.8455 0.8546</td>
<td>0.831</td>
<td>0.831</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
<tr>
<td>Paca</td>
<td>0.7883 0.7816</td>
<td>0.8004 0.8145</td>
<td>0.8035</td>
<td>0.828</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
<tr>
<td>Red Brocket Deer</td>
<td>0.8148 0.8241</td>
<td>0.8568 0.8803</td>
<td>0.8622</td>
<td>0.8795</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
<tr>
<td>White-nosed Coati</td>
<td>0.814 0.8348</td>
<td>0.8398 0.8415</td>
<td>0.8439</td>
<td>0.859</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
<tr>
<td>Spiny Rat</td>
<td>0.721 0.7282</td>
<td>0.7485 0.7503</td>
<td>0.7745</td>
<td>0.7921</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
<tr>
<td>Ocelot</td>
<td>0.7926 0.7844</td>
<td>0.8048 0.8117</td>
<td>0.8356</td>
<td>0.849</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
<tr>
<td>Red Squirrel</td>
<td>0.7648 0.7486</td>
<td>0.7892 0.7962</td>
<td>0.8194</td>
<td>0.8278</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
<tr>
<td>Common Opossum</td>
<td>0.76 0.7782</td>
<td>0.8071 0.8286</td>
<td>0.8142</td>
<td>0.8409</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
<tr>
<td>Bird spec</td>
<td>0.5973 0.5543</td>
<td>0.6367 0.6488</td>
<td>0.6701</td>
<td>0.6963</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
<tr>
<td>Great Tinamou</td>
<td>0.7398 0.7581</td>
<td>0.8143 0.8185</td>
<td>0.8428</td>
<td>0.8568</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
<tr>
<td>White-tailed Deer</td>
<td>0.8028 0.8147</td>
<td>0.847 0.8672</td>
<td>0.8476</td>
<td>0.8641</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
<tr>
<td>Mouflon</td>
<td>0.769 0.7498</td>
<td>0.7962 0.8067</td>
<td>0.8168</td>
<td>0.8431</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
<tr>
<td>Red Deer</td>
<td>0.824 0.8345</td>
<td>0.8397 0.8416</td>
<td>0.8591</td>
<td>0.8689</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
<tr>
<td>Roe Deer</td>
<td>0.8157 0.8354</td>
<td>0.8254 0.8435</td>
<td>0.8463</td>
<td>0.8626</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
<tr>
<td>Wile Boar</td>
<td>0.7937 0.8491</td>
<td>0.8312 0.8477</td>
<td>0.8508</td>
<td>0.8658</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
<tr>
<td>Red Fox</td>
<td>0.6612 0.6814</td>
<td>0.7162 0.7211</td>
<td>0.7394</td>
<td>0.7634</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
<tr>
<td>European Hare</td>
<td>0.6918 0.6815</td>
<td>0.7573 0.7604</td>
<td>0.775</td>
<td>0.7992</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
<tr>
<td>Wood Mouse</td>
<td>0.7135 0.6981</td>
<td>0.7681 0.7692</td>
<td>0.7784</td>
<td>0.7978</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
<tr>
<td>Coiban Agouti</td>
<td>0.6982 0.7204</td>
<td>0.7582 0.7685</td>
<td>0.7921</td>
<td>0.7993</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>0.7703 0.7515</td>
<td>0.7937 0.8043</td>
<td>0.8212</td>
<td>0.8398</td>
<td>0.8646</td>
<td>0.803</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.4: Performance comparison on Camera Trap dataset. Metrics showing average F-scores.

data and highly dynamic and cluttered background, it is easy to explain that cross-frame information fusion could compensate for the inevitable noise introduced during learning and fine-tuning, thus improves the performance of learned models.

In order to evaluate the impact of cross-frame validation, we adjust the number of pairs, i.e., $M$ in Eq. (2.5) to be used in the verification phase. A special case is when $M = 0$, all cross-frame validation is disabled, and features are only extracted on a single patch without MSDE. From Fig. 2.11, we can see that, with more frames being included in the verification, the overall F-Score is being improved significantly. Our proposed method outperforms Fast-RCNN and Faster-RCNN using only one pair, which proved the effectiveness of cross-frame validation in improving the final performance.

We also monitored the processing time of our system with different experimental choices, as shown in Table 2.6. The hardware choice is Intel Xeon E5-1603v3 @2.8GHz (4C4T) and Nvidia Titan X. The results indicate that even with longer processing time of our iterative graph-cut method, it significantly reduced the number of verification candidates, thus
Figure 2.11: Effect of number of frames used in validation on the overall F-score performance.

in return not only benefit to overall performance but also ran faster.

(3) Performance evaluation as segmentation task. We also evaluate our proposed method on Change Detection 2014 dataset [58], using three categories: dynamic background, low frame-rate, and intermittent object motion. These selected categories best fit the complex scenes in the wild, thus are ideal for comparison with other approaches. Note that CDnet dataset is purely for motion detection and does not provide any training data. In order to make our patch verification model to recognize objects such as cars and boats in CDnet, we need extra training data. In experiments, we evaluate the models trained on camera-trap, Pascal VOC 2007+2012 [59] and both, respectively. Table 2.5 shows the performance comparisons on CDnet 2014 dataset. By comparing with state-of-art approaches, we observed that no single algorithm could handle all scenes well enough especially in different categories. Our method, though require off-line processing and auxiliary training data, is
Table 2.5: Performance comparison on Change-Detection 2014 dataset using three categories.

<table>
<thead>
<tr>
<th></th>
<th>Dynamic Background</th>
<th>Low Frame-rate</th>
<th>Intermittent Object Motion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Training data</td>
<td>avg Re</td>
<td>avg Prec</td>
</tr>
<tr>
<td>AMBER [66]</td>
<td>0.9177</td>
<td>0.799</td>
<td>0.8436</td>
</tr>
<tr>
<td>EFIC [67]</td>
<td>0.6667</td>
<td>0.6849</td>
<td>0.5779</td>
</tr>
<tr>
<td>CEFIC [68]</td>
<td>0.6556</td>
<td>0.6993</td>
<td>0.5627</td>
</tr>
<tr>
<td>CwistarDRP [69]</td>
<td>0.8291</td>
<td>0.8723</td>
<td>0.8487</td>
</tr>
<tr>
<td>SSBS</td>
<td>0.7875</td>
<td>0.9185</td>
<td>0.8391</td>
</tr>
<tr>
<td>FTSG [70]</td>
<td>0.8691</td>
<td>0.9129</td>
<td>0.8792</td>
</tr>
<tr>
<td>PAWCS [71]</td>
<td>0.8868</td>
<td>0.9038</td>
<td>0.8938</td>
</tr>
<tr>
<td>IUTIS-3 [72]</td>
<td>0.8778</td>
<td>0.8239</td>
<td>0.896</td>
</tr>
<tr>
<td>IUTIS-5 [72]</td>
<td>0.8636</td>
<td>0.9324</td>
<td>0.9239</td>
</tr>
<tr>
<td>This work</td>
<td>camera-trap</td>
<td>0.8593</td>
<td>0.9014</td>
</tr>
<tr>
<td>This work</td>
<td>voc07+voc12</td>
<td>0.8859</td>
<td>0.9425</td>
</tr>
<tr>
<td>This work</td>
<td>trap+voc07+12</td>
<td>0.8764</td>
<td>0.9401</td>
</tr>
</tbody>
</table>

Figure 2.12: Segmentation results comparison with state-of-art methods on Change Detection dataset 2014. (a)-(c) Original frame, region of interest and ground-truth mask. (d) FTSG [70]. (e) IUTIS-5 [72]. (f) PAWCS [71]. (g) Superpixel Strengthen Background Subtraction. (h) IEC without verification model. (i) Proposed method.

very robust in maintaining relatively good recall while preserve very high precision. For intermittent object motion category, our method achieved weaker performance compared with the other two categories due to the incapability to detect object candidates at the first stage, i.e., iterative graph-cut, however, our verification model still achieved high precision by rejecting most non-object detections. As is shown in Fig 2.12, our method successfully addressed the false alarms that are inevitably spotted by various algorithms due to the complex scenes. By comparing columns (d) to (i), we can see the two-step propose and reject scheme can alleviate the trade-off problem between recall and precision. The generic object based verification model is highly robust, effective, and self-adaptive to various com-
Table 2.6: Average processing time per image in second with various experimental choices.

<table>
<thead>
<tr>
<th>Experimental choices</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Selective Search</td>
<td>✓ ✓ ✓</td>
</tr>
<tr>
<td>Use Iterative Graph-Cut</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Run CNN feature extractor on GPU</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Run CNN feature extractor on CPU</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Run CNN using large batch</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
<tr>
<td>Proposal generation</td>
<td>0.75 0.75 0.75 1.03 1.03 1.03</td>
</tr>
<tr>
<td>Verification</td>
<td>8.94 3.68 2.92 4.19 1.01 0.54</td>
</tr>
<tr>
<td>Total</td>
<td>9.69 4.43 3.67 5.22 2.04 1.57</td>
</tr>
</tbody>
</table>

plex scenes. We also noticed that mixing camera-trap with Pascal VOC dataset generates slightly inferior results. Considering the entirely different types of scenes between camera-trap and Pascal VOC, and the large volume of Pascal VOC dataset itself, we recognize that using Pascal VOC dataset is good enough to train a generic purpose objectness model to be used in CDnet dataset which has similar human surrounded environments.

2.5 Conclusion

In this work, we have successfully developed an animal object segmentation scheme for highly dynamic and cluttered scenes using iterative ensemble graph cuts followed by joint object verification. Our extensive experimental results and performance comparisons over a diverse set of challenging videos with dynamic scenes, including the Camera Trap dataset and the Change Detection Challenge Dataset, demonstrated that the proposed video object segmentation framework outperforms various state-of-the-art algorithms [73, 74].

Major Contributions. The major contributions of this work are: (1) we have developed a new and effective framework for animal detection and segmentation in highly cluttered natural scenes. It couples ensemble foreground-background graph cuts segmentation with foreground-background verification, to achieve high true-positive rates while significantly reducing the false positive rates. (2) These two components compliment with each other to achieve accurate and reliable moving object detection and segmentation. (3) We have de-
veloped a compound feature which combines the FV-encoded HoG feature and the DCNN feature, to achieve significantly improved performance.
Chapter 3

Knowledge Projection for Effective Design of Thinner and Faster Deep Neural Networks

3.1 Introduction

Recently, large neural networks have demonstrated extraordinary performance on various computer vision and machine learning tasks. Visual competitions on large datasets such as ImageNet [8] and MS COCO [75] suggest that wide and deep convolutional neural networks tend to achieve better performance, if properly trained on sufficient labeled data with well-tuned hyper-parameters, at the cost of extremely high computational complexity. Over-parameterization in large networks seems to be beneficial for the performance improvement [76, 77], however, the requirements for large sets of labeled data for training and high computational complexity pose significant challenges for us to develop and deploy deep neural networks in practice.

First, low power devices such as mobile phones, cloud based services with high throughput demand, and real-time systems, have limited computational resources, which requires that the network inference or testing should have low computational complexity. Besides
the complexity issue, a large network often consumes massive storage and memory bandwidth. Therefore, smaller and faster networks are often highly desired in real-world applications. Recently, great efforts have been made to address the network speed issue. A variety of model compression approaches [78, 79, 80, 81, 82] were proposed to obtain faster networks that mimic the behavior of large networks. Second, in practical applications, we often have access to very limited labeled samples. It is very expensive to obtain human labeled ground-truth samples for training. In some applications domains, it is simply not feasible to accumulate enough training examples for deep networks [83, 84, 85, 86].

Interestingly, these two problems are actually coupled together. The network capacity is often positively correlated to its task complexity. For instance, we would expect a small network classifier of two classes (e.g., dog and cat) to achieve a similar level of accuracy as a significantly larger network for tens of thousand classes of objects. Existing solutions to obtaining a fast network on new tasks is often based on a two-step approach: train the network on a large dataset, then apply model compression or distillation to the network after fine-tuning or transfer learning on the new dataset [82]. Each step is performed separately and they are not jointly optimized. Therefore, how to jointly address the problems of network compression, speed up, and domain adaptation becomes a very important and intriguing research problem.

A successful line of work [87, 82, 88, 89, 90] suggest that cumbersome large neural networks, despite their redundancy, have very robust interpretation of training data. By switching learning targets from labels to interpreted features in small networks, we have observed not only speed-ups but also performance improvements. Inspired by this phenomenon, we are interested to explore if this interpretation power is still valid across different (at least similar) domains, and to what extent of performance a newly trained student network can achieve with the help of a large model pre-trained on different datasets.
3.2 Related Work

Large neural networks have demonstrated extraordinary performance on various computer vision and machine learning tasks. During the past a few years, researchers have been investigating how to deploy these deep neural networks in practice. There are two major problems that need to be carefully addressed: the high computational complexity of the deep neural network and the large number labeled samples required to train the network [91, 92]. Our work is closely related to domain adaptation and model compression, which are reviewed in this section.

To address the problem of inadequate labeled samples for training, methods for network domain adaptation [93, 85, 94] have been developed, which enable learning on new domains with few labeled samples or even unlabeled data. Transfer learning methods have been proposed over the past several years, and we focus on supervised learning where a
small amount of labeled data is available. It has been widely recognized that the difference in the distributions of different domains should be carefully measured and reduced [94]. Learning shallow representation models to reduce domain discrepancy is a promising approach, however, without deeply embedding the adaptation in the feature space, the transferability of shallow features will be limited by the task-specific variability. Recent transfer learning method coupled with deep networks can learn more transferable representations by embedding domain adaptations in the architecture of deep learning [95] and outperforms traditional methods by a large margin. Tzeng et al. [86] optimizes domain invariance by correcting the marginal distributions during domain adaptation. The performance has been improved, but only within a single layer. Within the context of deep feed-forward neural networks, fine-tune is an effective and overwhelmingly popular method [96, 45]. Feature transferability of deep neural networks has been comprehensively studied in [97]. It should be noted that this method does not apply directly to many real problems due to insufficient labeled samples in the target domain. There are also some shallow architectures [98, 99] in the context of learning domain-invariant features. Limited by representation capacity of shallow architectures, the performance of shallow networks is often inferior to that of deep networks [94].

With the dramatically increased demand of computational resources by deep neural networks, there have been considerable efforts to design smaller and thinner networks from larger pre-trained network in the literature. A typical approach is to prune unnecessary parameters in trained networks while retaining similar outputs. Instead of removing close-to-zero weights in the network, LeCunn et al. proposed Optimal Brain Damage (OBD) [78] which uses the second order derivatives to find trade-off between performance and model complexity. Hassibi et al. followed this work and proposed Optimal Brain Surgeon (OBS) [79] which outperforms the original OBD method, but was more computationally intensive. Han et al. [100] developed a method to prune state-of-art CNN models without loss of accuracy. Based on this work, the method of deep compression [80] achieved better
network compression ratio using ensembles of parameter pruning, trained quantization and Huffman coding, achieved 3 to 4 times layer-wise speed up and reduced the model size of VGG-16 [19] by 49 times. This line of work focuses on pruning unnecessary connections and weights in trained models and optimizing for better computation and storage efficiency.

Various factorization methods have also been proposed to speed up the computation-intensive matrix operations which are the major computation in the convolution layers. For example, methods have been developed to use matrix approximation to reduce the redundancy of weights. Jenderberg et al. [81] and Denton et al. [101] use SVD-based low rank approximation. For example, Gong et al. [102] use a clustering-based product quantization to reduce the size of matrices by building an indexing. Zhang et al. [103] successfully compressed very deep VGG-16 [19] to achieve 4 times speed up with 0.3% loss of accuracy based on Generalized Singular Value Decomposition and special treatment on non-linear layers. This line of approaches can be configured as data independent processes, but fine-tuned with training data to improve the performance significantly. In contrast to off-line optimization, Ciresan et al. [104] trained a sparse network with random connections, providing good performance with better computational efficiency than densely connected networks.

Rather than pruning or modifying parameters from existing networks, there has been another line of work in which a smaller network is trained from scratch to mimic the behavior of a much larger network. Starting from the work of Bucila et al. [87] and Knowledge Distillation (KD) by Hinton et al. [82], the design of smaller yet efficient networks has gained a lot of research interest. Smaller networks can be shallower (but much wider) than the original network, performing as well as deep models, as shown by Ba and Caruna in [105]. The key idea of knowledge distillation is to utilize the internal discriminative feature that is implicitly encoded in a way not only beneficial to original training objectives on source training dataset, but also has a side-effect of eliminating incorrect mappings in networks. It has been demonstrated in [82] that small networks can be trained to generalize
in the same way as large networks with proper guidance. FitNets [88] achieved better compression rate than knowledge distillation by designing a deeper but much thinner network using trained models. The proposed hint-based training is one step further beyond knowledge distillation which uses a finer network structure. Nevertheless, training deep networks has proven to be challenging [106]. Significant efforts have been devoted to alleviate this problem. Recently, adding supervision to intermediate layers of deep networks is explored to assist the training process [107, 108]. These methods assume that source and target domains are consistent. It is still unclear whether the guided training is effective when the source and target domains are significantly different. DenseNet [16] is a successful tweak which helps deeper network to converge by concatenated links rather than summation (ResNet) so links of various depths can contribute to the back-propagation. However, training thin and deep networks directly is still an unsolved problem, especially when we tackle with very limited training data.

3.3 Methods

In this work, we consider a unique setting of the problem. We use a large network pre-trained on a large dataset (e.g., the ImageNet) to guide the training of a thinner and faster network on a new smaller dataset with limited labeled samples, involving adaptation over different data domains and model compression at the same time.

3.3.1 Knowledge Projection Network

In this section, we present the proposed Knowledge Projection Network (KPN). We start with the KPN architecture and then explain the knowledge projection layer design. A multi-path multi-stage training scheme coupled with iterative pruning for projection route selection is developed afterwards.
Overview

Enlightened by the teacher-student paradigm described in FitNets [88], KPN is built on connected teacher-network pair while the original hard regressor was replaced by a link of projection matrices. An example pipeline of KPN is illustrated in Figure 3.2. Starting from a large teacher network pre-trained on a large dataset, a student network is designed to predict desired outputs for the target problem with guidance from the teacher network. The student network uses similar building blocks as the teacher network, such as Residual [15], Inception [109] Dense Connections [16] or stacks of plain layers [19], sub-sampling and BatchNorm [110] layers. The similarity in baseline structure ensures smooth transferability. Note that the convolution layers consume most of the computational costs. Their complexity can be modeled by the following equation

\[ C = \sum_{i=1}^{N-1} C_i \cdot H_i \cdot W_i \cdot C_{i+1} \cdot K_i^H \cdot K_i^W, \quad (3.1) \]

where the computational cost is multiplicatively related to the number of input \( C_i \) and

Figure 3.2: KPN architecture. Solid arrows showing the forward data-flow, dotted arrows showing the paths for gradients.
output channels $C_{i+1}$, the spatial size of input feature map $H_i \cdot W_i$ where $H_i$ and $W_i$ are the height and width of the feature map at the $i$-th layer, and kernel size $K_i^H \cdot K_i^W$. The student network is designed to be thinner (in terms of filter channels) but deeper to effectively reduce network capacity while preserves enough representation power [105, 88]. We depict the convolutional blocks in Figure 3.3 that are used to build the thin student networks.

In contrast to standard convolutional layers, a squeeze-then-expand [111, 14] structure is effective in reducing the channel-wise redundancy by inserting spatially narrow ($1 \times 1$) convolutional layers between $3 \times 3$ standard convolutional layers. We denote this structure as bottleneck Type A and extend it to a more compact squeeze-expand-squeeze shape, namely bottleneck Type B (bottleneck structure used in ResNet [15]). With (3.1), we can calculate the proportional layer-wise computation cost for the standard convolutional layer, bottleneck Type A and B, respectively. For simplicity, feature map dimensions are denoted in capital letters, and we use identical size for kernel height and width, denoted as $K$, without loss of generality:

$$
\begin{align*}
C \times H \times W & \xrightarrow{3 \times 3 \text{ Conv}} C' \times H \times W \\
C \times H \times W & \xrightarrow{1 \times 1 \text{ Conv}} X \times H \times W \\
X \times H \times W & \xrightarrow{3 \times 3 \text{ Conv}} C' \times H \times W \\
C \times H \times W & \xrightarrow{1 \times 1 \text{ Conv}} X \times H \times W \\
X \times H \times W & \xrightarrow{1 \times 1 \text{ Conv}} C' \times H \times W
\end{align*}
$$

Figure 3.3: Left: Standard $3 \times 3$ Convolutional layer. Middle: Bottleneck type A. Right: Bottleneck type B. $H$ and $W$ are feature spatial height and width, $C, X, C'$ are input, reduced and output channels for this building block, respectively. For simplicity, batch-norm and activation layers are omitted in this figure.
\[ C_{\text{standard}} = C \cdot H \cdot W \cdot C' \cdot K^2, \quad (3.2) \]

\[ C_{\text{TypeA}} = C \cdot H \cdot W \cdot X + X \cdot H \cdot W \cdot C' \cdot K^2, \quad (3.3) \]

\[ C_{\text{TypeB}} = C \cdot H \cdot W \cdot X + X^2 \cdot H \cdot W \cdot K^2 + X \cdot H \cdot W \cdot C'. \quad (3.4) \]

Combining (3.2), (3.3) and (3.4), we define the reductions in computation for Type A and B as

\[ \frac{C_{\text{TypeA}}}{C_{\text{standard}}} = \frac{X}{C'} \cdot K^2 + \frac{X}{C} \approx \frac{X}{C}, \quad (3.5) \]

\[ \frac{C_{\text{TypeB}}}{C_{\text{standard}}} = \frac{X}{C'} \cdot K^2 + \frac{X^2}{C} \cdot C' + \frac{X \cdot K^2}{C} \approx \frac{X^2}{C} \cdot C'. \quad (3.6) \]

Bottleneck structures A and B can effectively reduce the computational cost while preserve the dimension of feature map and receptive field, and the layer-wise reduction is controlled by \( X \). For example, by cutting the bottleneck channels by half, i.e., \( X = \frac{C}{2} \), we have the approximate reduction rate \( \frac{1}{2} \) for Type A, \( \frac{1}{4} \approx \frac{1}{8} \) for Type B. In practice, the output channel \( C' \) is equal to or larger than input channel \( C \): \( C' \in [C, 2C] \). We replace standard convolutional layers by bottleneck structures A and B in the teacher network according to computational budget and constitute corresponding student network. Layer-wise width multipliers \( \alpha = \frac{X}{C} \) are the major contributor to model reduction. We use smaller \( \alpha \) in deep layers where the feature is sparse and computational expensive layers where the gain is significant. The bottleneck structure A is used when we want to match similar non-linearity between teacher and student network. The flexibility of bottleneck structures and elastic value range of \( \alpha \) ensured we have enough degrees of freedom controlling the student network capacity. In our KPN, the student network is trained by optimizing the following joint...
loss function:

$$W^*_s = \arg \min_{W_s} \lambda \cdot L_{KP}(W_s, W_k) + L_p(W_s) + R,$$  \hspace{1cm} (3.7)

where $L_{KP}$ and $L_p$ are loss from the knowledge projection layer and problem specific loss, respectively. For example, for the problem-specific loss, we can choose the cross-entropy loss in many object recognition tasks. $\lambda$ is the weight parameter decaying during training, $W_k$ is the trained teacher network, $R$ is a $L_2$ regularization term, and $W^*_s$ is the trained parameters in the student network. Unlike traditional supervised training, the knowledge projection loss $L_{KP}$ plays an important role in guiding the training direction of KPN, which will be discussed in more detail in the following section.

**Knowledge Projection Layer Design**

In this work, the pre-trained teacher network and the student network analyze the input image simultaneously. We expect student to learn as much task relevant knowledge as possible from teacher network while preserve the regularization to strip unnecessary regression target since we are training on a new task domain. To use the teacher network to guide the student network, we propose to map the feature $F_T$ of size $N$ learned at one specific layer of the teacher network into a feature vector $F_S$ of size $M$ and inject it into the student network to guide its training process. For the mapping, we choose linear projection

$$F_S = \mathcal{P} \cdot F_T,$$  \hspace{1cm} (3.8)

where $\mathcal{P}$ is an $N \times M$ matrix, $F_T$ and $F_S$ are outputs without non-linearity of teacher and student intermediate layers, respectively. In deep convolutional neural networks, this linear projection matrix $\mathcal{P}$ can be learned by constructing a convolution layer between the teacher and student network. Specifically, we use a convolutional layer to bridge teacher’s knowledge layer and student’s injection layer. A knowledge layer is defined as the output
of a teacher’s hidden convolutional layer responsible for guiding the student’s learning process by regularizing the output of student’s injection convolutional layer. Let $O^t_h$, $O^t_w$ and $O^t_c$ be the spatial height, spatial width, and number of channels of the knowledge layer output in the teacher network, respectively. Let $O^s_h$, $O^s_w$ and $O^s_c$ be the corresponding sizes of student’s injection layer output, respectively. Note that there are a number of additional layers in the student network to further analyze the feature information acquired in the inject layer and contribute to the final network output. We define the following loss function:

$$\mathcal{L}_{KP(W_s, W_k)} = h[\mu(x; W_k)] \cdot |r[\mu(x; W_k); W_{KP}] - v[x; W_s]|,$$  \hspace{1cm} (3.9)

$$h(x) = \begin{cases} 1, & \text{if } x \geq 0, \\ \eta, & \text{otherwise} \end{cases}$$  \hspace{1cm} (3.10)

where $\mu$ and $v$ represent the deep nested functions (stacks of convolutional operations) up to the knowledge and injection layer with network parameters $W_k$ and $W_s$, respectively. $r[.]$ is the knowledge projection function applied on $\mu[.]$ with parameter $W_{KP}$ which is another convolution layer in this work. $\mu$, $v$ and $r$ must be comparable in terms of spatial dimensionality. A Leaky-Relu activation function $h$ is applied to loss for sake of soft regression purpose while incorporating the effects of following Relu activations.

The knowledge projection layer is designed as a convolutional operation with a $1 \times 1$ kernel in the spatial domain. As a result, $W_{KP}$ is a $O^t_c \times O^s_c \times 1 \times 1$ tensor. As a comparison, a fully connected adaptation layer will require $O^t_h \times O^t_w \times O^t_c \times O^s_h \times O^s_w \times O^s_c$ parameters which is not feasible in practice especially when the spatial size of output is relatively large in the early layers. Using the convolutional adaptation layer is not only beneficial for lower computational complexity, but also provides a more natural way to filter distinctive channel-wise features from the knowledge layers while preserve spatial consistency. The
output of the knowledge projection layer will guide the training of student network by generating a strong and explicit gradient applied to backward path to the injection layer in the following form

$$\Delta W_{s,i} = -\lambda \cdot \frac{\partial L_{KP}}{\partial W_{s,i}}, \quad (3.11)$$

where \( W_{s,i} \) is the weight matrix of injection layer in student network. Note that in (3.9), \( h[\mu(x; W_k)] \) is applied to \( L_{KP} \) with respect to the hidden output of knowledge projection layer as a relaxation term. For negative responses from \( \mu(x; W_k) \), \( L_{KP} \) is effectively reduced by the slope factor \( \eta \), which is set to 0.25 by cross-validation. In practice, performance is not sensitive to slope value \( \eta \) if \( \eta \geq 0.1 \), and an automatic solution PReLU [112] is also available as a substitution. Overall, \( L_{KP} \) acts as a relaxed \( L1 \) loss. Compared to \( L2 \) loss, \( L_{KP} \) is more robust to outliers, but still has access to finer level representations in \( r[\mu(x; W_k); W_{KP}] \). Unlike the traditional hard logistic loss used in Knowledge Distillation [82] and FitNets [88], the unique setting of projection layer can benefit from the backward path starting from ground-truth of target dataset. The regression loss of knowledge projection is selective, which is optimized for given dataset.

**Multi-Path Multi-Stage Training**

In the student network, layers after the injection layer are responsible for adapting the projected feature to the final network output. This adaptation must be memorized throughout the training process. Those network layers before the injection layer aim to learn distinctive low-level features. Therefore, in our KPN framework, the student network and knowledge projection layer are randomized and trained in two stages: **initialization stage** and end to end **joint training stage**.

In the initialization stage, Path ① in Figure 3.2 is disconnected, i.e. the knowledge projection layer together with the lower part of student network is trained to adapt the in-
Figure 3.4: Candidate Routes of Knowledge Projection. Candidate routes are paths from teacher’s knowledge layer to student’s injection layer. Only one route will survive after iterative pruning.

Intermediate output of teacher’s knowledge layer to the final target by minimizing $\mathcal{L}_p$, which is the loss for target task, e.g., softmax or linear regression loss. The upper part of student network is trained sorely by minimizing $\mathcal{L}_{KP}$. In this stage, we use the projection matrix as an implicit connection between upper and lower parts in the student network. The upper student network layers are always optimized towards features interpreted by the projection matrix, and have no direct access to targets. This strategy prevents the student network from over-fitting quickly during the early training stage which is very hard to correct afterwards.

After the initialization stage, we then disconnect Path ① and reconnect Path ②, the training now involves jointly minimizing the objective function described in (3.7). Using
the results from stage 1 as the initialization, the joint optimization process aims to establish smooth transitions inside the student network from the input to the final output. The loss $\mathcal{L}_{KP}$ injected into the student network continues to regularize the training process. In this way, the student network is trained based on a multi-loss function which has been used in the literature to regulate deep networks [113].

**Iterative Pruning for Projection Route Selection**

One important question in knowledge projection between the teacher and student networks is to determine which layers from the teacher network should be chosen as the knowledge layer and which layers from the students should be chosen for the injection layer. In this work, we propose to explore an iterative pruning and optimization scheme to select the projection route.

Assume that the teacher network $N_t$ and the student network $N_s$ have $L_t$ and $L_s$ layers, respectively. Candidate projection routes are depicted in Figure 3.4. We use only convolution layers as candidates for the knowledge and injection layers. To satisfy the constraints on spatial size and receptive field, candidate knowledge projection routes are computed and denoted as $R_{i,j} \in \mathcal{G}$, where $i$ is the index of knowledge layer in the teacher network, $j$ is the index of injection layer in the student network, and $\mathcal{G}$ is the set of all candidate routes. We follow the procedure for computing the center of receptive field in [114] for calculating the size of receptive field in layer $L$:

$$S_L = \sum_{p=1}^{L} \prod_{q=1}^{p-1} S_q (F_p - 1),$$  

(3.12)

where $S_q$ and $F_p$ are the layer-wise stride and kernel size, assuming they are identical along $x$ and $y$ directions for simplicity. Routes with constrained receptive filed are kept after calculation with a small tolerance $\beta = 0.2$: 

49
(1 − β) · S_i ≤ S_j ≤ (1 + β) · S_i. \hspace{1cm} (3.13)

For example, in Figure 3.4, we have

\{R_{1,3}, R_{1,4}, R_{2,5}\} \subset \mathcal{G} \hspace{1cm} (3.14)

and the rest routes in this figure are not valid due to mismatched spatial shapes. The idea of iterative pruning for the projection route selection is to traverse all possible routes with same training hyper-parameters, and determine the best route for knowledge-injection pair on-the-fly. Specifically, we randomly initialize |\mathcal{G}| KPNs according to each \(R_{i,j}\).

Each KPN stores a student network \(W_s\), knowledge projection parameter \(W_{KP}\) and routing \(R_{i,j}\), teacher network \(W_t\) is shared across all KPNs to save computation and memory overhead. The target is to find the KPN setting with minimum joint loss

\(\{W'_s, W'_{KP}, R'_{i,j}\} = \arg\min_{\{W_s, W_{KP}, R_{i,j}\}} (\lambda \cdot \mathcal{L}_{KP} + \mathcal{L}_p)\). \hspace{1cm} (3.15)

We assume that the pre-trained teacher network \(W_t\) is responsible for guiding the training of a specifically designed student network \(W_s\) which satisfies the computational complexity requirement. According to (3.13), we can generate a list \(\mathbb{L}\) of candidate KPNs. Each KPN is a copy of the designed student network \(W_s\) with different projection routing \(R_{i,j}\) and corresponding parameters \(W_{KP}\). Within a period of \(k\) epochs, the KPNs are optimized separately using Stochastic Gradient Descend to minimize the joint loss described in (3.15). Note that even though the optimization target is a joint loss, as depicted in Fig. 3.2, the upper and bottom layers of the student network are receiving different learning targets from the teacher network and dataset distribution, respectively. At the end of \(k\) epochs, the joint loss of each KPN computed on the validation dataset is used to determine which KPN to prune. The same procedure is applied on the remaining KPNs in the list \(\mathbb{L}\) iteratively. This iterative pruning procedure is summarized in Algorithm 1:
Algorithm 1: Iterative pruning algorithm for projection route selection.

**Input:** List $\mathbb{L}$ of KPNs, as in form $\{W_{s,n}, W_{KP,n}, R_{i,n,j,n}\}$, where $n = 1, \ldots, |G|$, and teacher network $W_t$

**Output:** $W_s^\ast$, $W_{KP}^\ast$ and $R_{i,j}^\ast$

1. Configure all KPNs as initialization stage.
2. while $|\mathbb{L}| > 1$ do
   3. for $k$ epochs do
      4. for Batch $x$ in Data do
         5. Forward teacher: $y_t \leftarrow \mu(x; W_k)$;
         6. for $\{W_s, W_{KP}, R_{i,j}\} \in \mathbb{L}$ do
            7. Forward-backward w.r.t. $W_s, W_{KP}$;
         end
      end
   8. end
   9. $\{W_s', W_{KP}', R_{i,j}'\} \leftarrow \arg \min (\lambda \cdot \mathcal{L}_{KP} + \mathcal{L}_p)$;
   10. Remove $\{W_s', W_{KP}', R_{i,j}'\}$ in $\mathbb{L}$;
11. end
12. return $\{W_s^\ast, W_{KP}^\ast, R_{i,j}^\ast\}$ in $\mathbb{L}$;

Only one KPN will survive after the iterative pruning process. We continue the multi-stage training with or without adjusting the batch-size depending on the released memory size after sweeping out bad KPNs. The stopping criteria can either be plateau of validation accuracy or a pre-defined end epoch.

### 3.4 Experimental Results

In this section, we provide comprehensive evaluations of our proposed method using three groups of benchmark datasets. Each group consists of two datasets, the large dataset $D_l$ used to train the teacher network and the smaller dataset $D_s$ used to train the student network. The motivation is that, in practical applications, we often need to learn a network to recognize or classify a relatively small number of different objects and the available training dataset is often small. We also wish the trained network to be fast and efficient. The large dataset is often available from existing research efforts, for example, the ImageNet. Both the large and the small datasets have the same image dimensions so that pre-trained
models are compatible with each other in terms of shape. We use the existing teacher network model already trained by other researchers on the public dataset $D_t$. We compare various algorithms on the benchmark dataset $D_s$ where state-of-the-art results have been reported. Performance reports on small datasets are rare, thus we choose existing large famous benchmark datasets in following experiments, and aggressively reduce the size of training set to simulate the shortage of labeled data in real world scenarios.

### 3.4.1 Network Training

We have implemented our KPN framework using the MXNet [115], a deep learning framework designed for both efficiency and flexibility. The dynamically generated computational graph in MXNet allows us to modify network structures during run time. The KPNs are trained on NVidia Titan X 12GB with CUDNN v5.1 enabled. Batch-sizes vary from 16 to 128 depending on the KPN group size. For all experiments, we train using the Stochastic Gradient Descend (SGD) with momentum 0.9 and weight decay 0.0001 except the knowledge projection layers. The weight decay for all knowledge projection layers is 0.001 in the initialization stage and 0 for the joint training stage. 40% of iterations are used for the initialization stage, and the rest goes to be joint training stage. The weight controller parameter $\lambda$ for joint loss is set to be 0.6, and gradually decays to 0. The pruning frequency is 10000 and we also randomly revoke the initialization stage during joint training stage, to repetitively adjusting network guidance strength.

For fine-tuning, we test with a wide variety of experimental settings. Starting from pre-trained networks, we adjust the last layer to fit to the new dataset, and randomly initialize the last layer. The reshaped network is trained with standard back-propagation with respect to labels on the new dataset, and unfreeze one more layer from the bottom one at a time. The best result from all configurations was recorded. To make sure all networks are trained using the optimal hyper-parameter set, we extensively try a wide range of learning rates, and repeat experiments on the best parameter set for at least 5 times. The average perfor-
mance of the best 3 runs out of 5 will be reported. Data augmentation is limited to random horizontal flip if not otherwise specified.

Table 3.1: CIFAR-10 accuracy and network capacity comparisons with state-of-the-art methods. Results using randomly sampled subsets from training data are also reported. Number of network parameters are calculated based on reports in related work.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Accuracy with Different $S_T$</th>
<th>$N_{Para}$</th>
<th>$N_{MA}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>50000</td>
<td>5000</td>
<td>1000</td>
</tr>
<tr>
<td>Maxout [116]</td>
<td>90.18</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FitNets-11 [88]</td>
<td>91.06</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FitNets [88]</td>
<td>91.61</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GP CNN [117]</td>
<td>93.95</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ALL-CNN-C [118]</td>
<td>92.7</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Good Init [119]</td>
<td>94.16</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ResNet-50 slim</td>
<td>87.53</td>
<td>71.92</td>
<td>55.86</td>
</tr>
<tr>
<td>ResNet-38</td>
<td>90.86</td>
<td>75.28</td>
<td>61.74</td>
</tr>
<tr>
<td>ResNet-38 fine-tune</td>
<td>91.15</td>
<td>89.61</td>
<td>86.26</td>
</tr>
<tr>
<td>Our method</td>
<td>92.37</td>
<td>90.35</td>
<td>88.73</td>
</tr>
</tbody>
</table>

3.4.2 Results on the reduced CIFAR-10 Dataset

We first evaluate the performance of our method on the CIFAR-10 dataset guided by a teacher network pre-trained on CIFAR-100 dataset. The CIFAR-10 and CIFAR-100 datasets [120] have 60000 $32 \times 32$ color images with 10 and 100 classes, respectively. Typically they were both split into 50K-10K sets for training and testing, we choose different sizes $S_T$ of the training set and list the accuracy to simulate smaller datasets. To validate our approach, we trained a 38-layer Resnet on the CIFAR-100 as reported in [15], and use it to guide a 50-layer but significantly slimmer Resnet on the CIFAR-10. We augment the data using random horizontal flip and color jittering. Table 3.1 summarizes the results, with comparisons against the state-of-the-art results which cover a variety of optimization techniques including Layer-sequential unit-variance initialization [119], pooling-less [118], generalized pooling [117] and maxout activation [116]. For network complexity, we compute its number of model parameters $N_{Para}$ and the number of multiplication and
additions $N_{MA}$ needed for the network inference. It should be noted that for methods in the literature we do not have their accuracy results on down-sized training sets.

We do not apply specific optimization techniques used in the state-of-the-art methods due to some structures not reproducible in certain conditions. To compare, we trained a standard 38-layer Residual Network, a 50-layer slimmer version of ResNet (each convolutional layer is half the capacity of the vanilla ResNet) and a fine-tuned model of 38-layer ResNet (from CIFAR-100) on CIFAR-10 with different amount of training samples. With all 50000 training data, our proposed method outperforms direct training and best fine-tuning results and still match the state-of-the-art performance. We believe the performance gain specified in [117, 119] can be also applied to our method, i.e., ensemble of multiple techniques could achieve better performance. The proposed KPN method has improved the accuracy by up to 1.2% while significantly reducing the network size by about 11 times, from 3.1M network parameters to 273K parameters. It also demonstrated strong robustness against aggressive reduction of labeled training samples.

### 3.4.3 Results on the reduced Pascal VOC 07 Dataset

Table 3.2: PASCAL VOC 2007 test object classification performances comparison. Results using randomly sampled subsets from training data are also reported. Number of convolution layer parameters are listed for fair comparison based on reports in related work.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Accuracy at Different $S_T$</th>
<th>$N_{Para}$</th>
<th>$N_{MA}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5011</td>
<td>1000</td>
<td>200</td>
</tr>
<tr>
<td>Chatfield et al. [121]</td>
<td>82.4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>VGG16+SVM [19]</td>
<td>89.3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>VGG19+SVM [19]</td>
<td>89.3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>HCP-VGG [122]</td>
<td>90.9</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FisherNet-VGG16 [123]</td>
<td><strong>91.7</strong></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>VGG16 standard BP</td>
<td>83.5</td>
<td>65.2</td>
<td>&lt;30</td>
</tr>
<tr>
<td>Fine-tune VGG16 last layer (softmax)</td>
<td>89.6</td>
<td>87.4</td>
<td>85.7</td>
</tr>
<tr>
<td>Fine-tune VGG16 2+ learnable layers</td>
<td>90.2</td>
<td>86.3</td>
<td>82.8</td>
</tr>
<tr>
<td>Our method</td>
<td><strong>91.2</strong></td>
<td><strong>88.4</strong></td>
<td><strong>86.5</strong></td>
</tr>
</tbody>
</table>
We evaluate the proposed method on the PASCAL Visual Object Classes Challenge (VOC) dataset [10] with a VGG-16 model [19] pre-trained on the ILSVRC 2012 dataset [8]. The pre-training usually takes several weeks, thus we downloaded and converted the teacher network from the Caffe model available online. We compare our method with state-of-the-art results obtained on this dataset in the literature, including the VGG16+SVM method [19], the segment hypotheses based multi-label HCP-VGG method [122], and the FisherNet-VGG16 method [123] which encodes CNN feature with fisher vector. These papers have reported results on the original whole dataset with 5011 images. To test the learning capability of the network on smaller datasets with reduced samples, we also implement the fine-tuning method. We try different combination of network update scheme and learning parameters and use the best result for performance comparison with our method. We conducted our experiments on the entire training set with 5011 images and test set
Table 3.3: Test error rate comparisons between experimental settings and baseline methods.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Error Rates at Different $S_T$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>19280</td>
</tr>
<tr>
<td>Deep CNN [124]</td>
<td>13.5%</td>
</tr>
<tr>
<td>Deep Siamese CNN [124]</td>
<td>8.0%</td>
</tr>
<tr>
<td>Large CNN standard BP</td>
<td>9.3%</td>
</tr>
<tr>
<td>Small CNN standard BP</td>
<td>12.1%</td>
</tr>
<tr>
<td>Fine-tuned from MNIST</td>
<td>6.8%</td>
</tr>
<tr>
<td>Our method</td>
<td>5.9%</td>
</tr>
</tbody>
</table>

with 4952 images. In addition, we randomly sample 50 and 10 images from each class, generating two small datasets with 1000 and 200 training images, respectively. The results are summarized in Table 3.2. We list the test accuracy of the network for each configuration. We compute the corresponding complexity of the network, including the number of model parameters $N_{Para}$ and the number of multiplication and additions $N_{MA}$. It should be noted that for methods in the literature we do not have their accuracy results on down-sized training sets. It can be seen that our proposed method outperforms standard training and fine-tuning by a large margin while reducing the model size by 2 times and improving the inference speed by 4.6 times.

### 3.4.4 Results on the Omniglot Dataset

We are interested in how the proposed KPN method works on very small datasets, for example, the Omniglot handwritten recognition dataset. The MNIST [6] is a famous handwritten digits dataset, consists of 60000 training images and 10000 test images, 28x28x1 in size, organized into 10 classes. The Omniglot [124] is a similar but much smaller dataset, containing 1623 different handwritten characters from 50 alphabets. Each of the 1623 characters was drawn online via Amazon’s Mechanical Turk by 20 different people. All images are binarized and resized to $28 \times 28 \times 1$ with no further data augmentation. We use all 70000 images from MNIST for training a 5-layer Maxout convolutional model as the
Table 3.4: Network configurations for extensive benchmarks on Omniglot dataset. N denotes slim network with N layers, similarly, N layer slimmer network is denoted as N-. Note that 1 × 1 adaptive convolutions for residual modules are not included in this table.

<table>
<thead>
<tr>
<th># Layers</th>
<th>50</th>
<th>50-</th>
<th>50-</th>
<th>44-</th>
<th>44-</th>
<th>38-</th>
<th>38-</th>
<th>32-</th>
<th>32-</th>
<th>26-</th>
<th>26-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv 3 x 3 / s1</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>ResConv 3 x 3 / s2</td>
<td>32 x 16</td>
<td>32 x 16</td>
<td>32 x 16</td>
<td>32 x 14</td>
<td>32 x 12</td>
<td>16 x 12</td>
<td>32 x 10</td>
<td>16 x 10</td>
<td>32 x 8</td>
<td>16 x 8</td>
<td></td>
</tr>
<tr>
<td>ResConv 3 x 3 / s1</td>
<td>64 x 16</td>
<td>64 x 16</td>
<td>64 x 16</td>
<td>64 x 14</td>
<td>64 x 12</td>
<td>32 x 12</td>
<td>64 x 10</td>
<td>32 x 10</td>
<td>64 x 8</td>
<td>32 x 8</td>
<td></td>
</tr>
<tr>
<td>Conv 3 x 3 / s1</td>
<td>128 x 16</td>
<td>128 x 16</td>
<td>128 x 16</td>
<td>128 x 14</td>
<td>128 x 12</td>
<td>64 x 12</td>
<td>128 x 10</td>
<td>64 x 10</td>
<td>128 x 8</td>
<td>64 x 8</td>
<td></td>
</tr>
</tbody>
</table>

teacher network $N_t$ as proposed in [116]. We report experimental results of various algorithms across a wide range of number of training examples, from 19280 to merely 1000, shown in Table 3.3. Note that we use class dependent shuffling to randomly select training subsets, which is critical to avoid unbalanced class distribution in Omniglot due to the limited number of samples for each class. We can see that the proposed KPN is able to reduce the error rate by 1.1-1.3%. Table 3.3 also provides some interesting insights of how models are transferred to different tasks. First, the fine-tuning methods are all affected by the number of learnable parameters and training samples. Smaller training set will result in significant over-fitting, thus breaking the fragile co-adaptation between layers. If the training set is large enough, the number of learnable parameters are positively related to the performance. This phenomenon is also discussed in [97], where transferring knowledge from the pre-trained model to an exactly same network is extensively tested.

3.4.5 Algorithm Parameter Analysis

In this section, we study how the performance of the our method is impacted by the selection of major parameters.

(1) **Trade-off between Performance and Efficiency.** To evaluate how the size of network affects the performance, we measure the test accuracy, number of parameters, and network speed up ratio of various student networks on the CIFAR-10 dataset. Figure 3.6 shows the results. Student networks are designed based on a multi-layer Resnet denoted as $N$- or $N$- -, where $N$ is the number of layers, - and - - indicate it’s a slim or slimmer
Figure 3.6: Network capacity and performance analysis. Top: test accuracies with proposed KPN and normal training with standard back-propagation; Middle: number of parameters ($\times 10^3$), note that the y-axis is in logarithmic scale; Bottom: actual inference speed up ratio with respect to Resnet-50. Network notations: $i$ is teacher network, $N$- denotes slim network with $N$ layers, similarly, $N$ layer slimmer network is denoted as $N$- -.

version of Resnet. The detailed network configurations are listed in Table 3.4. As expected, deeper and slimmer networks are more difficult to train with limited training data. However, with proposed method enabled, the depth is beneficial, and networks are less suffered from performance drop. Impressively, we could obtain a model which is 34 times faster using less than 2% parameters, with about 3% accuracy loss, compared to the teacher network.

(2) **Analysis of Iterative Pruning for Automatic Route Selection.** The knowledge projection route is critical for the network training and test performance. Intuitively, the projection route should not be too shallow or too deep. Shallow layers may contain only low-level texture features, while deep layers close to output may be too task specific. To
Figure 3.7: Iterative pruning analysis. Top: occurrences of projection route $t-s$ over 32 standalone tests. Bottom: mean classification error of projection route $t-s$ by disabling iterative pruning. $t-s$: network with knowledge layer $t$ from teacher to injection layer $s$ from student.

study how the iterative pruning works during training, we record the pruning results and compare them with respect to manually defined projection routes, shown in Figure 3.7. We can see that the statistics of survival projection routes is highly correlated to the training accuracy, which is evaluated by manually defining projection route from $t$ to $s$ and disabling iterative pruning during training. The result also indicates that choosing the middle layers for projection is potentially better. Reducing the size of training data also affects the pruning results. This might relate to the difficulty of fitting knowledge projection layer to the target domain when very limited data is presented. As a result, projection layers tend to appear more on very deep layers close to the output, so that the penalty from adaptation loss will not dominate. The bottom line is, even though the iterative pruning method is a random optimization process, it is reliably producing satisfactory results.
3.4.6 Discussion and Future Work

Our KPN is designed in a highly modular manner. The training of projection layers is removed during actual network testing, and the network capacity is highly configurable for performance/speed trade-off. This KPN method can be easily extended to other problems such as object detection, object segmentation, and pose estimation by replacing softmax loss layer used in the classification problems. A lot of existing work can also benefit from our method, e.g., [125, 126, 127]. Since the deployed network is a pure standard network, another research direction is to apply KPN as a building block in traditional model compression techniques to reshape the network in a new perspective. Although we have focused on the advantage of KPN with thinner networks on smaller datasets, there are potential benefits to apply KPN on large network and relatively large datasets, for example, performance oriented situations where speed is not an issue.

3.5 Conclusion

We have developed a novel knowledge projection framework for deep neural networks the address the issues of domain adaptation and model compression in training simultaneously [128]. We exploit the distinctive general features produced by the teacher network trained on large dataset, and use a learned matrix to project them into domain relevant representations to be used by the student network. A smaller and faster student network is trained to minimize joint loss designed for domain adaptation and knowledge distillation simultaneously. Extensive experimental results have demonstrated that our unified training framework provides an effective way to obtain fast high-performance neural networks on small datasets with limited labeled samples.
Chapter 4

Progressive Deep Convolutional Neural Network

4.1 Introduction

Researchers in earlier years tend to design larger, deeper networks simply because they have capabilities to learn more complex tasks, ignoring computation budget and putting all stress on computational resources. This used to be a very good direction and helped pushing the limit of parallel computation and even distributed computation. However, in order to harvest the achievements in many areas, it is inevitable to consider a faster model for deployment. Selecting the right deep model for a specific task is hard. Deploying same model on different platforms is even challenging. To fulfill the requirements of accuracy and speed in various circumstances, the current strategy is to design different networks that fit in each category. It is extremely inefficient and time consuming considering the current training protocol usually involves a fresh training on huge datasets such as ImageNet [8] in order to obtain satisfactory performances. Hyper-parameter tuning is a repetitive procedure during this period, which is inefficient in both time and computational resources.

We are inspired by the idea ”Progressive decoding” in image compression domain. Pro-
gressive encoded JPEG [129], usually adopted in websites, are images encoded in multiple passes of progressively higher detail. The decoded image will present an animation effect from blur to finer details. This effect reduces the response time of loading each image and is able to control the quality of decoded image.

Deep CNNs are expensive to inference. The computational cost is a concept similar to bit budget in image compression. In our design, a progressive network is able to decompose the computation serially and expected to produce better accuracy while spending more and more computations on same input. In deployment, an endpoint inference device can determine the computational cost at runtime, thus substitute a flexible way to compose deep models literally everywhere, either on powerful workstation or edge devices.

Diving deep into the problem, we notice that, the inference structures, execution procedures, and computational complexity of existing deep neural networks, once trained, are fixed and remain the same for all test images, no matter how much they have been optimized speed-wise. In this work, our goal is to develop a new progressive framework for deep neural network such that a single model can be evaluated at different performance levels with different computational complexity requirements. This single-model-variable-complexity property is very important in practice. We recognize that different images have different complexity levels of visual representation and different difficulty levels for visual recognition. For simple images with low visual recognition complexity, we can easily classify the image or recognize the object with simple networks at very low complexity. For example, it will be very easy to detect a person standing in front of a clean background or classify if it is male or female. For harder images, we will need to extract sophisticated visual features using more layers of network representation to gain sufficient visual discriminative power so that the object can be successfully distinguished from those in other classes.

To validate this observation, we conducted the following interesting experiment. We collected 16 different pre-trained deep neural networks, ranging from very low-complexity
Figure 4.1: **Left**: Validation results of different models, images are sorted by average top-1 accuracies across models. **Right**: Individual image difficulties across different models.

MobileNet [20], to very high-complexity DenseNet201 [130]. We use these networks to classify the images in the ILSVRC2012 [8] 50k validation set. Fig. 4.1 (left) shows the classification results. Each row corresponds to a specific network. The horizontal axis represents the index of the test image. A blue line indicates that the image is successfully classified by the network or ranked first in all images. A magenta line indicates that the correct result is within the top 2 to 5 categories. A yellow line indicates that the correct result is beyond the top 5 results. As summarized in Fig. 4.1, we can see that 38% of images are always correctly classified by all networks, no matter how simple the network is. These are the so-called simple images with low visual recognition complexity. This suggests that, if we can successfully identify those set of simple images, we can save a lot of computational resources by choosing simple networks to analyze them. More excitingly, if we are able to model or predict the visual recognition complexity of the input image and if we are able to establish a progressive network, we can then adapt the network complexity during run-time according to the visual recognition complexity of the input image. This will allow us to save the computational resources significantly.

Let us look at one more example. Fig. 4.2 shows two images, a simple Ice Cream image and a hard Siamese Cat image with occlusion. We choose 5 networks with different computational complexity. The most complex network is the DenseNet201 [130] labeled
Figure 4.2: Each prediction result is produced by pre-trained model of different size. Top row: hard example which requires full network inference. Bottom row: easy example that is predicted confidently by even tiny models. Best viewed in color.

with 100%. The complexity levels of the other four networks are shown in approximate relative percentage in Fig. 4.2. For example, the inference cost of the first network is about 5% of DenseNet201. From this experiment, we can see that, for the simple Ice Cream image, the confidence score for Ice Cream is much higher than other object categories, even with very simple networks. However, for the hard Siamese Cat image, the score distribution is more uniform for low-complexity networks. As the network becomes more complex and more powerful, the classifier is more and more confident about the result since the score for the correct label is getting much higher than other categories.

These two experiments strongly suggest that it is highly desirable to establish a progressive deep neural network structure which is able to adapt its network inference structure and computational complexity to images with different visual recognition complexity. This progressive structure should be able to scale up its discriminative power and achieve higher recognition capability by activating and executing more analysis units of the deep neural networks and accumulating more visual evidences for more challenging vision analysis tasks or image sets.
4.2 Related Work

Deep learning frameworks such as Theano [131], Caffe [56], Torch [132], MXNet [115], Tensorflow [133] and CNTK [134] are the work horses of deep learning projects. These frameworks use namely computational graphs to dispatch computation workloads (e.g. deep neural networks) that define the orders of computations that are required to be performed to multi-core CPUs/GPUs that can fully utilize the parallel computational resources. This, in turn has led to a proliferation of libraries making it easier to train and deploy such models, by expressing them in terms of differentiable data-flow graphs over tensors. The recent achievements on deep convolutional neural networks are basically coupled with these deep learning frameworks, as well as high performance GPUs. Static computational graph is extremely fast and easier to optimize for sake of performance, however, it’s allocated prior of real tasks, thus is not flexible enough to handle dynamic situations where some parts of the computational graph is conditional. They also have difficulties to implement batching for various inputs with different shapes which is crucial for performance. TF-Fold [135] introduce a technique called dynamic batching, which not only batches together operations between different input graphs of dissimilar shape, but also between different nodes within a single input graph. The result is a static graph with support for dynamic inputs. This emulates a dynamic computational graph using existing static graph frameworks.

Nevertheless, the new deep learning architectures have moved beyond the original and mostly unchanged deep learning frameworks. This makes implementation of new models difficult. As most of the existing deep learning frameworks were designed for image processing using convolutional neural networks, their methods for abstracting data structures and training models are not optimal for implementing the newer generation of deep learning models. In addition, most existing frameworks use a domain specific language for representing deep learning models, along with an interpreter to translate them into a data structure stored in memory. Therefore, developers using these frameworks cannot use standard programming language debuggers—a significant problem as debugging is a ma-
JOR part of developing and tuning deep learning models. Later on, a fully imperative and
dynamic way of composing neural networks are available as a form of dynamic computa-
tional graph in Chainer [136], Dynet [137], Pytorch and Gluon. Dynamic computational
graph is a define-and-run mechanism that record forwarding path and apply reversed auto-
differentiation according to chain rule as back-propagation. Based on these properties, we
could generate a lot more sophisticated structures and even composing a random structure
at runtime.

This is work is closely related to complexity control / optimization and confidence
analysis of deep neural networks.

4.2.1 Complexity Optimization of Deep Neural Networks.

Deep neural networks often involves high computational complexity. A number of methods
have been developed to accelerate inference speed of deep neural networks, or reduce its
computational resource requirement so that they can operate on lower-end devices, such
as CPUs, embedded processors and mobile devices. Research community has spotted this
problem in early years. Vanhoucke et al. [138] explored the aligned memory and SIMD
operations to speed up CNN inference on CPUs. Mathieu et al. [139] discovered that
convolutional operations can be accelerated in Fourier domain using FFT. Linear matrix
factorization methods, proposed by Denton et al. [101] and Jaderberg et al. [81], produced
up to two times speed up with little compromise of classification accuracies. Denil et al.
[76] lead the researches of model pruning by showing heavily over-parameterized neural
network parameters. Gong et al. [102] and Wu [140] applied k-means scalar quantization
to pre-trained parameters. Significant speed up can be achieved by 8-bit integer and 16-bit
floating point quantization as shown in [141]. Parameter pruning approaches [142, 143, 79]
can be used to reduce network complexity. Low rank factorization and decomposition
[144], transferred learning and compact convolutional filter learning [145, 146] Unlike in-
place model acceleration, another line of work, such as Knowledge Distillation [147, 105,
have been developed to train a new student network by distill prior knowledges from teach networks.

In the meantime, speed oriented network designs from scratch have gained more attentions. MobileNet [20] explores the usage of depth-wise convolution throughout the entire network and outperforms existing models under the same computational constraint. ShuffleNet [21] couples depth-wise convolution and grouped convolution with channel shuffle mechanism to attain outstanding performance. In the latest MobileNet-v2 [148], an inverted residual bottleneck structure is introduced as the building block. We are observing combinations of successful designs and more efficient network building blocks available today.

NoScope [149] tackled the problem of very expensive surveillance video object detection by using a shallow and fast CNN as a early estimator and dispatcher. Only complex scenes with significant inter-frame changes will require a full run of deep object detection network, therefore higher analysis speed can be achieved. All of these networks aim to optimize the computational complexity and inference speed of deep neural networks, often at the cost of degraded visual recognition performance. Their inference structures, execution procedures, and computational complexity, once trained, are fixed and remain the same for all test images. They are not able to adapt their network inference structure and complexity to different resource supplies and input images.

4.2.2 Confidence Analysis for Deep Neural Networks.

Calibration and confidence scores have been studied in various contexts in recent years, especially the confidence measurement of very deep neural networks due to their overwhelmingly success. These uncertainties are formalized as probability distributions over either the model parameters, or model outputs, respectively. Researchers have recognized that the decision scores of existing deep neural networks are poorly calibrated [150]. For example, higher scores often do not correspond to better or closer samples. [151] argues
that probability scores generated by \textit{softmax} should not be considered as confidence score or distance measure directly since they are based on the $l_2$ norm of \textit{pre-softmax} inputs. To address this issue, various methods using scalar, vector, matrix and binning methods to calibrate the confidence scores produced by the \textit{softmax} function. Gaussian density modeling is proposed in [151] as a post prediction calibration using prior information of the training data. Pereyra et al. [152] proposed a penalty based overconfident prediction calibration as a form of regularization. The open set deep network approach in [153] attempts to measure uncertainty contributed from unknown categories. It should be noted that these methods are based on statistical modeling, being optimized on the entire validation set, and therefore not suitable for confidence analysis for each individual input image.

There exist uncertainly measurements for individual samples. For example, Gal et al. [154] has implemented a randomized dropout network [155] to estimate the uncertainty level of network prediction. Similarly, Lakshminarayanan et al. [156] use ensembles of networks to obtain uncertainty estimates. Kendall et al. [157] combine this approach with a model that outputs a predictive mean and variance for each data point. Additionally, neural networks can be used in conjunction with Bayesian models that output complete distributions. An example is shown in deep kernel learning [158], which combines deep neural networks with Gaussian processes on classification and regression problems. However, these methods introduces extra computational cost which is against the purpose of our work.

Fig. 4.3 provides an overview of the proposed framework for deep neural networks. In this work, we divide network into $N$ stages with $N$ deep neural network units, $U_n$, $1 \leq n \leq N$. Each unit $U_n$ consists of a set of network layers, including convolution, pooling, ReLU, etc. The output of unit $U_n$ is a feature vector $F_n$. At decision stage 1, we use the feature $F_1$ to generate the decision output, i.e., the classification result for the input image, using an evaluation network $E_n$. The evaluation network consists of a set of network layers, including convolution, pooling, fully connected layers, and softmax layers. At
stage 2, feature $F_1$ generated from unit $U_1$ and feature $F_2$ generated from unit $S_2$ are fused together using a fusion network to produce the fused feature $F_2$, which will be used by the evaluation network $E_2$ to produce the decision output. The fusion network consists of feature concatenating followed by convolution layers for normalization. The fused feature $F_2$ produced at stage 2 will be forwarded to stage $S_3$. At stage $N$, the network unit $S_N$ will produce feature $F_N$, which will be fused with $F_{N-1}$ from previous layers to produce the fused feature $F_N$. We can see that the proposed progressive deep neural network is able to accumulate and fuse more and more visual features, scale up its visual recognition power by activating more network units and certainly consuming more computational resources. Certainly, the network inference can be terminated at any stage.

Let $O_n$ be the output result produced by the evaluation network $E_n$. Certainly, at later stages, the progressive network is able to accumulate more visual features or evidences for classification, its classification accuracy or visual recognition power will be higher, and the uncertainty for decision will be lower. So, we need a carefully designed module, called Confidence Analysis and Decision Policy (CADP) network to analyze the output results $O_n$ from each stage and its previous stage. It will decide if the current decision is reliable enough with early termination of the inference process or we need to proceed to the next stage to gather more visual evidence. In this work, the CADP network is realized by a recurrent neural network (RNN) learned from the training data.
The task the CADP network is two-fold: (1) it needs to generate the decision of classification or other visual recognition tasks at the current stage $S_n$ using the evaluation results $\{O_m | 1 \leq m \leq n\}$ from the current and previous stages. (2) It needs to learn an optimal decision policy for early termination such that the overall computational complexity is minimized while maintaining the state-of-the-art classification accuracy achieved by existing non-progressive networks. Let $x_k, 1 \leq k \leq K$ be the input image. We define

$$E(x_k, n) = \begin{cases} 1, & \text{if } x_k \text{ is correctly classified at stage } n, \\ 0, & \text{if } x_k \text{ is wrongly classified at stage } n. \end{cases}$$ \hspace{1cm} (4.1)$$

We denote the decision policy in the CADP network by $\pi[\cdot]$. The CADP network decides that image $x_k$ be terminated at stage $\pi[x_k]$. Let $C(n)$ be the computational complexity of stage $S_n$. Then the computational complexity for evaluating the input image $x_k$ will be

$$C(x_k, \pi) = \sum_{m=1}^{\pi[x_k]} C(m).$$ \hspace{1cm} (4.2)$$

The overall accuracy for all test images will be given by

$$P(\pi) = \frac{1}{K} \cdot \sum_{k=1}^{K} E(x_k, \pi[x_k]).$$ \hspace{1cm} (4.3)$$

Therefore, the optimal policy to be learned by the CADP network aims to minimize the overall complexity while maintaining the target accuracy:

$$\pi = \arg \min_{\pi} C(x_k, \pi) = \sum_{m=1}^{\pi[x_k]} C(m),$$ \hspace{1cm} (4.4)$$

s.t. \quad P(\pi) = \frac{1}{K} \cdot \sum_{k=1}^{K} E(x_k, \pi[x_k]) \geq P_T.$$

In the following sections, we will present our progressive deep neural network design and explain our method to learn the CADP network.
4.3 Progressive Deep Neural Network Design

The concept of progressive inference is different from traditional network inference. It must be able to produce a sequence of complete prediction results. Early stage of the network should have small computational complexity. Besides, the features and results from previous stages should be reused and accumulated. As discussed in [148], the overall computation required by standard convolutional layers in a CNN is given by:

\[ C = \sum_{i=0}^{L} K_i^2 \times M_i \times N_i \times F_i^2, \] (4.5)

where \( K_i, F_i, M \) and \( N \) are kernel size, input feature map size, input and output channels of layer \( i \), respectively. To change the values of \( K_i \) and \( F_i \), we can choose different building blocks, such as \textit{residual} [15] and \textit{dense} [130]. In complexity-progressive network design, we focus on the rest two sets of parameters: \textit{channels}(\( M, N \)) and \textit{layers}(\( L \)), which dominate the overall complexity since their values are typically very large.

It can be seen that these two sets of parameters represent two different dimensions, corresponding to two different dimensions for network partition: horizontal and vertical partitions. This leads to two different structures, parallel and serial structures, for progress deep neural network design, which will be explained in the following.

(A) \textbf{Parallel structures.} In the parallel structure for deep neural networks, we partition the network into multiple stages by reducing the input and output channel size, \( M \) and \( N \). Let \( r \) be the down-sampling ratio of \( M \) and \( N \). The complexity of convolution layers can be then reduced by \( r^2 \) according to Eq. (4.5). As shown in Fig. 4.4 (left), at stage \( S_n \), we use a thin network with small input and output channel sizes. The depth of the network \( L \) could be as large as the original non-progressive network. We use existing building blocks developed in the literature. Available choices are residual [15], residual bottleneck [159], dense [130], inception [109], inception-residual [160] and NasNet [161] blocks. Similar to [162], a \textit{Reduction block} contains stride 2 convolution or down-sampling layers used
Figure 4.4: ProgNet prototype structures of parallel(left) and serial(right) for image classification. Spatial reduction cells are omitted in figures for simplicity.

to reduce spatial size by a factor of 2. A normal block keeps spatial dimension intact. In this parallel structure, the input image is analyzed by different network units with different channel sizes. The features generated by different units are fused and accumulated together using a concatenate operator.

(B) Serial structures. One limitation of the parallel structure is that the width of each unit or branch cannot be reduced to arbitrary numbers. In our experiments, 4 and 8 are the minimum effective width of each unit on the CIFAR [120] and ImageNet [8] datasets, respectively, in order to maintain sufficient representation capacity. The serial structure partitions the network along the dimension of layers $L$. As shown in Fig. 4.4 (right), we extract features from different layers of the network, apply global pooling to them, and use a fully connected layer to generate the output decision. Also, this feature is concatenated with features extracted from next layers to be used for decision at the next stage. We can see that in this serial structure, the complexity of different stages increases linearly with the
Designing and successfully training the progressive network structure is a challenging task. Specifically, we need to make sure: (1) the overall accuracy at stage $S_n$ achieved by the evaluation network $E_n$ is increasing with $n$. Otherwise, additional computational computational resources have been wasted. (2) When we apply the full complexity, i.e., evaluate each input using the whole network, we need to make sure that it is more complexity-accuracy effective than existing state-of-the-art networks.

Following the work in [7, 19] for multi-class classification, we use the Cross Entropy loss as our joint loss function:

$$L = \sum_{m=1}^{M} w_m \cdot \mathcal{L}_{\text{CrossEntropy}}(y_m, \hat{y})$$  \hspace{1cm} (4.6)

where $w_m$ and $y_m$ are weight and output from stage $m$, respectively. $\hat{y}$ is the ground-truth label. If not otherwise required by targeting application, we treat each stage with uniform weights ($\vec{w} = \vec{u}$), i.e. outputs from all stages are equally important.

### 4.4 Confidence Analysis and Decision Policy Learning

In the previous section, we have introduced the ProgNet that can perform network prediction at a sequence of stages. At each stage, ProgNet needs to determine if the current evaluation output is reliable or confident enough and if it is necessary to proceed to the next stage for accumulate more visual evidences. During our experiments, we found out that the decisions at different stages are inter-dependent with each other. Specifically, the current stage needs to examine the evaluation results in all previous stages for effective decisions. To address issue, we propose to design a recurrent neural network (RNN) to learn the confidence analysis and decision policy (CADP). As illustrated in Fig. 4.5, the RNN CADP network uses the pre-softmax outputs in the evaluation networks of all previous stages to
learn the confidence estimation and decision policy at the current stage.

More specifically, for each input (usually a mini-batch of images), a RNN controller takes as input the $n$-class pre-softmax output from the current CNN classifier, and produces $n + 1$ outputs, with $n$ new categorization results and 1 post-sigmoid confidence estimation. Post-sigmoid confidence score is compared with user defined threshold $t$ in range $(0, 1)$ to determine whether output is emitted directly from RNN classification results, otherwise another stage of CNN-RNN-decision is required.

Optimizing the RNN controller is the most challenging problem in this work. For each image, with a user-defined threshold $t \in (0, 1)$, the objective of CADP network is to solve the following optimization problem: minimizing the overall error rate subjected to a computational complexity constraint:

$$
\begin{align*}
\text{minimize}_{x} & \quad \mathcal{L}_{\text{CADP}} = \int_{0}^{1} \sum_{i=1}^{M} c_i \cdot \prod_{j=1}^{i} H(z_j - t) dt \\
& + \lambda \cdot \int_{0}^{1} (1 - \hat{y}_p) dt, \\
\end{align*}
$$

$$
\begin{align*}
p = \arg \max_{i=0}^{M-1} [\text{where}(H(z_i - t) = 1 \land H(z_{i+1} - t) = 0)],
\end{align*}
$$

Figure 4.5: Early termination control during inference by RNN controller.
subject to \[ 0 < z_i < 1, \quad z_i \in \mathbb{R} \] \hspace{1cm} (4.9)

where \( M \) is the number of stages and \( C_i \) is the normalized computational cost of \( i \)-th DNN unit. \( C_i \) is a constant number once ProgNet is composed. \( H(\cdot) \) is Heaviside(unit) step function which is 1 for positive inputs and 0 for negative inputs. \( \hat{y}_p \) is the correctness score (1 if correctly classified, 0 otherwise) of the \( p \)-th stage which is the last rejected stage by \( t \). Finding \( p \) is equivalent to finding the first index where \( (H(z_i - t) = 1) \) and \( (H(z_{i+1} - t) = 0) \) as shown in Eq. (4.8). Without loss of generality, we approximate the confidence integral \( \int_0^1 (\cdot) dt \) using summation over discrete samples of \( T \) within the range of \((0, 1)\). The above optimization problem becomes:

\[
L_{conf} = \sum_{t \in T} \left[ \sum_{i=0}^{M} c_i \cdot \prod_{j=0}^{i} H(z_j - t) + \lambda \cdot \sum_{t \in T} (1 - \hat{y}_p) \right].
\hspace{1cm} (4.10)

Combining standard cross entropy loss and \( L_{conf} \), we have the following optimization objective function:

\[
\text{minimize}_{w^*, b^*} \quad - \sum y_k \cdot \log \hat{y}_k + \alpha \cdot \sum_{t \in T} \left[ \sum_{i=0}^{M} c_i \cdot \prod_{j=0}^{i} H(z_j - t) \right] + \lambda \cdot \sum_{t \in T} (1 - \hat{y}_p),
\hspace{1cm} (4.11)
\]

where \( w^* \) and \( b^* \) are weights and bias of the RNN controller, respectively. \( y_k \) is the ground-truth of \( k \)-th image, \( \alpha \) is the hyper-parameter controlling weights of classification and confidence losses. While it is possible to directly optimize Eq. (4.11) with constraints Eq. (4.9) using the method in [163], we found that it is more efficient and robust to convert the
problem into:

\[
\minimize_{z_j \in \mathbb{R}} \sum_{t \in T} \sum_{i=0}^{M} c_i \cdot \prod_{j=0}^{i-1} H(z_j - t) + \lambda \cdot \sum_{t \in T} (1 - \hat{y}_p), \tag{4.12}
\]

\[
\minimize_{w^*, b^*} -\sum y_k \cdot \log \hat{y}_k + \alpha \cdot \sum_{i=0}^{M} |z_i - z_i^*|, \tag{4.13}
\]

where \(z^*\) is the optimal confidence score. The problem in Eq. (4.12) can be solved with the Constrained Optimization by Linear Approximation algorithm [164]. Eq. (4.13) can be solved using back-propagation with a standard stochastic gradient descend (SGD).

Note that the desired \(\hat{y}_p\) in Eqs. (4.10), (4.11), and (4.12) is the output from the RNN classifier, while it is possible to update \(\hat{y}_p\) after each batch or epoch, it is a very expensive process. In this work we are using the outputs from the evaluation networks for the first 80% controller training epochs. We then update using the latest \(\hat{y}_p\) from the controller output, and continue training controller for the rest 20% epochs. We implement the RNN controller using a three-layer LSTM, stacking 3 LSTM cells with 2 Dropout [155] layers in between. Before each forwarding step, internal states of controller are initialized with zero inputs. More training details will be provided in Section 4.5.

### 4.5 Experimental Results

In this section, we evaluate our proposed ProgNet using the CIFAR-10 [120] and ImageNet(ILSVRC2012) [8] datasets. On both datasets, our goal is to train a single ProgNet model which provides progressive complexity-accuracy scalability while outperforming existing state-of-the-art networks in terms of complexity-accuracy performance. All ProgNet models are trained on AWS P3 8x large instances \(^1\) with 4 Tesla V100 GPUs. Testings and run-time benchmarks are executed on local workstation with 1 Intel Xeon(R) CPU E5-1620

\(^1\)https://aws.amazon.com/cn/ec2/instance-types/p3/
Table 4.1: Network Configurations for CIFAR-10 and ImageNet Datasets.

<table>
<thead>
<tr>
<th>Multiplier</th>
<th>CIFAR-10</th>
<th>ImageNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 × 32</td>
<td>3 × 3 conv, stride 2</td>
<td>7 × 7 conv, stride 2</td>
</tr>
<tr>
<td>16 × 16</td>
<td>[res × 2] - [3 × 3 s2 max pool]</td>
<td>3 × 3 max pool, stride 2</td>
</tr>
<tr>
<td>8 × 8</td>
<td>[res × 3] - [3 × 3 s2 max pool]</td>
<td>[d × 7]-[d, stride 2, fc]</td>
</tr>
<tr>
<td>4 × 4</td>
<td>[res × 3] - [global avg pool]</td>
<td>[d × 11]-[d, stride 2, fc]</td>
</tr>
<tr>
<td>1 × 1</td>
<td>fc × 4</td>
<td>fc</td>
</tr>
</tbody>
</table>

(A) Network Configurations. Both parallel and serial structures of the ProgNet are flexible and highly expandable. In this work, we conducted extensive experiments using three different network settings for CIFAR-10 and two different settings for ImageNet, which are summarized in Table 4.1.

(B) Network Training and Inference. The base classifier and LSTM controller in ProgNet are trained separately using SGD optimization. For the base network, we use a batch size of 256. The number of training epochs for CIFAR-10 and ImageNet are 350 and 180, respectively. Following [130, 162, 161], we use an initial learning rate 0.1, weight decay 0.0001, and momentum 0.9. The learning rate is lowered by a factor of 10 at 25%, 50% and 80% of total epochs. Parameters with the best mean accuracy of all DNN units are saved as our best model. We then start training the LSTM controller using this best model.

We sample the early termination threshold $T = \{x \times 0.1 \mid x = 1, 2, ..., 9\}$. 0 and 1.0 are

---

2https://mxnet.incubator.apache.org/tutorials/gluon/gluon.html
Figure 4.6: Error rate versus actual average inference speed on CIFAR-10 validation set. Black squares indicate results from previously published networks running on our machine.

excluded because 0 are considered as no network inference and 1.0 stands for full network inference. The controllers are optimized using SGD with a learning rate 0.5, weight decay 0 and momentum 0 for all experiments.

To evaluate the impact of RNN controllers during inference, we conducted experiments using the following two modes: dynamic and fixed. In the dynamic mode, users can specify the confidence $T$ as early termination threshold. This is the desired behavior in this work. For comparison, we allow the ProgNet to follow preset stage setting and run in fixed mode. Once set, The ProgNet acts as a non-progressive network.

In our experiments, we evaluate two different progressive structure: parallel and serial, with different stages, such as 4, 6, and 9 stages. We also evaluate two different modes, dynamic and fixed, for the CADP controller. For example, in our figures, the notation p6-dynamic indicates that the network has a parallel structure with 6 stages and a dynamic CADP mode. s9-fixed indicates that the network has a serial structure with 9 stages and a fixed CADP mode.
Table 4.2: Fully inferred ProgNet performance and other state-of-the-art models on CIFAR-10 classification. MACC stands for the number of composite Multiply-Accumulation operations. Error rates reported in parentheses of ProgNets are post-controller results.

<table>
<thead>
<tr>
<th>model</th>
<th># params</th>
<th>MACC</th>
<th>error rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DenseNet (L = 40, k = 12) [130]</td>
<td>1.0M</td>
<td>5.34M</td>
<td>5.24</td>
</tr>
<tr>
<td>DenseNet(L = 100, k = 12) [130]</td>
<td>7.0M</td>
<td>35.6M</td>
<td>4.10</td>
</tr>
<tr>
<td>DenseNet (L = 100, k = 24) [130]</td>
<td>27.2M</td>
<td>138.4M</td>
<td>3.74</td>
</tr>
<tr>
<td>ResNet-20 [15]</td>
<td>0.28M</td>
<td>41.8M</td>
<td>8.73</td>
</tr>
<tr>
<td>ProgNet-p4-residual-full</td>
<td>3.3M</td>
<td>38.57M</td>
<td>4.92(4.77)</td>
</tr>
<tr>
<td>ProgNet-p6-residual-full</td>
<td>4.43M</td>
<td>51.57M</td>
<td>4.53(4.39)</td>
</tr>
<tr>
<td>ProgNet-s9-dense-full</td>
<td>1.15M</td>
<td>47.98M</td>
<td>4.23(4.08)</td>
</tr>
</tbody>
</table>

4.5.1 Experimental Results on the CIFAR-10 Dataset

In our CIFAR-10 experiments, we follow previous practice [15] to augment the data. We pre-process the training images by converting them to float32, followed by up-sampling and random cropping to 32 × 32. Random horizontal flipping is applied during training as a common strategy. The validation data is processed by converting to float32 without augmentation. We set the batch size to be 50. For a given early termination threshold $T$, the confidence analysis and decision policy network in ProgNet will decide when to stop the network inference for the input image. Let $C(T)$ be the corresponding average network inference time of ProgNet. We also record the overall error rate by $P(T)$. Fig. 4.6 shows the $P(T)$ curves the following network settings: p6-dynamic, p4-dynamic, and s9-dynamic. We show the results on CPU (left) and GPU (right). We include results for ProgNet with fixed controller modes (shown in solid boxes). For comparison, we also include the complexity-accuracy results for state-of-the-art networks on the CIFAR-10 dataset: resnet20, resnet10, densenet100/12, densenet100/24. It can be seen that our proposed ProgNet outperforms existing networks in the complexity-accuracy performance using one single model. As we increase the number of stages, we can achieve a large complexity-accuracy scalability range. The parallel structure is slightly better than the serial structure.
Figure 4.7: Top-1 accuracy versus actual average inference speed on the ImageNet validation set. Black squares represent results from previously published networks.

### 4.5.2 Experimental Results on the ImageNet

Following the same procedures outlined in existing papers [159, 130, 21, 162, 161], we conduct the experiments of ProgNet training and testing on the ILSVRC 2012 [8]. We use an image size $224 \times 224$. Training images are augmented with random cropping with min/max aspect ratio of $3/4$ and $4/3$. Random horizontal flipping is used in training. For training and validation, pixel means of $[123, 117, 104]$ are subtracted from images and then

<table>
<thead>
<tr>
<th>Model</th>
<th>Top-1</th>
<th>Params</th>
<th>MACC</th>
<th>CPU</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18 [15]</td>
<td>68.0</td>
<td>11.69M</td>
<td>1.83G</td>
<td>52.37ms</td>
<td>0.54ms</td>
</tr>
<tr>
<td>ResNet-50 [15]</td>
<td>75.4</td>
<td>25.26M</td>
<td>3.87G</td>
<td>103.00ms</td>
<td>1.86ms</td>
</tr>
<tr>
<td>DenseNet-121 [130]</td>
<td>74.9</td>
<td>7.98M</td>
<td>3.08G</td>
<td>142.73ms</td>
<td>1.95ms</td>
</tr>
<tr>
<td>MobileNet-v1 [20]</td>
<td>70.6</td>
<td>4.2M</td>
<td>575M</td>
<td>41.98ms</td>
<td>0.59ms</td>
</tr>
<tr>
<td>MobileNet-v2 [148]</td>
<td>71.7</td>
<td>3.4M</td>
<td>300M</td>
<td>64.78ms</td>
<td>0.89ms</td>
</tr>
<tr>
<td>ShuffleNet(1.5) [21]</td>
<td>69.0</td>
<td>2.9M</td>
<td>292M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NasNet-A [161]</td>
<td>74.0</td>
<td>5.3M</td>
<td>564M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ProgNet-p4-dynamic</td>
<td>73.9</td>
<td>13.12M</td>
<td>1.87G</td>
<td>89.03ms</td>
<td>1.17ms</td>
</tr>
<tr>
<td>ProgNet-s6-dynamic</td>
<td>74.6</td>
<td>14.3M</td>
<td>1.31G</td>
<td>85.31ms</td>
<td>1.23ms</td>
</tr>
<tr>
<td>ProgNet-p4-fixed @t=0.5</td>
<td>71.9</td>
<td>13.12M</td>
<td>-</td>
<td>47.3ms</td>
<td>0.67ms</td>
</tr>
<tr>
<td>ProgNet-p4-fixed @t=0.6</td>
<td>72.5</td>
<td>13.12M</td>
<td>-</td>
<td>59.8ms</td>
<td>0.81ms</td>
</tr>
<tr>
<td>ProgNet-s6-fixed @t=0.3</td>
<td>67.1</td>
<td>14.3M</td>
<td>-</td>
<td><strong>23.2ms</strong></td>
<td><strong>0.31ms</strong></td>
</tr>
</tbody>
</table>
normalized by standard deviations of [58.4, 57.12, 57.38]. As in [15, 159, 162, 165, 166], we use 1.28 million images for training and 50000 images for testing.

Similar to the above CIFAR-10 experiments, we record the complexity-accuracy curve \( P(T) \) for different ProgNet structures (parallel and serial) with different stages (4, and 6 stages) using different controller modes (dynamic and fixed). Fig. 4.7 shows these curves for CPU (left) and GPU (right). We also include complexity-accuracy results achieved by existing networks, including resnet18, resnet50, densenet121, mobilenet-v1, mobilenet-v2. Table 4.3 summarizes the complexity-accuracy performance of ProgNet in comparison with existing networks. We include results on the number of model parameters, number of MACC (Multiply-Accumulation) operations, and running times on CPU and GPU. It can be seen that our proposed ProgNet outperforms existing networks in the complexity-accuracy performance by a large margin using one single model. For the same complexity, both ProgNet variants outperform the MobileNet [20, 148], which has been significantly optimized, by 2% to 3% in classification accuracy. For the same accuracy, the ProgNet-p4-dynamic model is able to achieve 20% less inference time than MobileNet-v2.

In our ProgNet design, the confidence analysis and decision policy (CADP) network plays a critical role since it controls whether the next network stage should be activated for the input image or its inference should be early terminated. This has a direct impact on the complexity-accuracy performance of our ProgNet. To further understand the behavior and capability of the CADP controller, we implement a random controller in which the network inference of an input image is terminated at a random stage. We then run the experiment using this random controller on the ImageNet dataset for 1000 times to simulate partial brute-force search for the best control policy. In Fig. 4.8, one dot represents the complexity-accuracy result for one experimental run. The solid curve represents our CADP controller. We can see that the CADP is very effective, outperforming static control.
Figure 4.8: Complexity-accuracy performance comparison between our CADP controller and random termination.

Table 4.4: Video Frame Classification Results on Sports-1M Dataset

<table>
<thead>
<tr>
<th>Models</th>
<th>Clip Hit@1</th>
<th>Video Hit@1</th>
<th>Video Hit@5</th>
<th>Total Inference time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Frame [167]</td>
<td>41.1</td>
<td>59.3</td>
<td>77.7</td>
<td>-</td>
</tr>
<tr>
<td>Single-Frame + Multires [167]</td>
<td>42.4</td>
<td>60.0</td>
<td>78.5</td>
<td>-</td>
</tr>
<tr>
<td>Slow Fusion [167]</td>
<td>41.9</td>
<td>60.9</td>
<td>80.2</td>
<td>-</td>
</tr>
<tr>
<td>MobileNetv2 [148]</td>
<td>43.8</td>
<td>62.2</td>
<td>83.4</td>
<td>3.46 hour</td>
</tr>
<tr>
<td>ProgNet-s6-full</td>
<td><strong>44.9</strong></td>
<td><strong>63.0</strong></td>
<td><strong>83.9</strong></td>
<td>7.5 hour</td>
</tr>
<tr>
<td>ProgNet-s6-dynamic</td>
<td>44.2</td>
<td>62.5</td>
<td>82.9</td>
<td><strong>1.55 hour</strong></td>
</tr>
</tbody>
</table>

4.5.3 Experimental Results on the Sports-1M Dataset

The Sports-1M dataset [167] contains 1,133,158 sports videos which have been annotated with 487 labels. The amount of video frames in Sports-1M dataset is enormous, therefore making it extremely sensitive to inference speed.

We fine-tune our models on Sports-1M dataset which were originally pre-trained on ImageNet. In order to compare the overall performance and inference speed with modern networks designed for efficiency, we also fine-tuned the latest MobileNet-v2 [148] on Sports-1M with same methodology.

To solve the problem that occasionally there are more than one labels for a video frame,
we modified the objective of CADP network by replacing sparse label with a vector label in Eq. 4.8.

\[
p = \arg \mathop{\min}_{i=0}^{M-1} \left[ \text{where} \left( \vec{H}(\vec{z}_i - t) = 1 \land \vec{H}(\vec{z}_{i+1} - t) = 0 \right) \right], \quad (4.14)
\]

To present fair qualitative results on this dataset, we followed exactly the same prediction methods described in [167]. We randomly sampled 20 clips from each video and present each clip to the network 4 times with different crops and horizontal flips. The prediction results of different categories are averaged to produce the global estimation of the entire video clip. That is in total \(218667 \times 20 \times 4 = 17493360\) inferences, that is more than 17 million predictions to make, not to mention we are sub-sampling the video clips. Limited by the vast amount of data, we only tested MobileNet-v2 and ProgNet-s6 models. We show the performance comparison on Sports-1m in Table. 4.4. Not surprisingly, ProgNet-s6 achieves slightly better classification performance than the MobileNetv2, which outperforms the baselines reported in [167] by a large margin. It is also notably that running in dynamic prediction mode, ProgNet-s6 is 5 times faster than fully inference mode, with a minor accuracy drop. Compared to MobileNetv2, ProgNet-s6-dynamic has better accuracy and achieved more than 2 times speed up ratio. ProgNet can significant amount of time spent on inference.

4.6 Discussion

In addition to classification results, we also conducted comprehensive studies into ProgNets.
Figure 4.9: Termination performance of ProgNet-s6-dense on ILSVRC12 validation set. For reference, prob is using raw softmax probability for decision by comparing maximum class probability with threshold $T$, and random uses a Bernoulli distribution with 0.5 probability as decision rule.

**Termination by LADP Controller.**

LSTM controllers in ProgNets are responsible for terminating executions when intermediate confidences exceed threshold. Therefore performance of controllers are critical to ProgNets. Termination In Fig. 4.9, comparisons of recall/precision performance between LSTM controller and raw softmax probabilities are shown as ROC curves. It is obvious that our controller outperforms naive softmax probability by a large margin, indicate it is more reliable and efficient to apply LSTM controller even taken into consideration the extra cost of controller. We also inspected emitted image distribution at all stages of ProgNet-s6-dense model at threshold 0.5, depicted in Fig. 4.10. Easy images which are successfully ranked as top-1 (as in Fig. 4.1) are terminated by early stages. In contrast, hard examples are more likely to retain until the last stage, indicating controller is effective distinguish between intermediate outputs.
Analysis of Easy/Hard classification examples.

We briefly discussed the idea of classification difficulty in Section ??, which is measured by an image’s cross-model classification accuracy. Back to the original problem, we are curious to know how easy and hard images behave so different using the same neural network? We show randomly selected sample images from the easiest and the most difficult splits of ImageNet validation set in Fig. 4.11. By carefully observing the properties of im-

Figure 4.10: Distribution of terminated extremely easy and hard images in each stage of ProgNet-s6-dense model on ILSVRC12 validation set.

Figure 4.11: Easy(left) vs. Hard(right) examples in ImageNet. Visually there is no significant difference between easy and hard examples. **Best viewed in color.**
ages belonging to the extreme easy and hard sets, we cannot simply conclude any pattern to distinguish between them. This observation supports our preliminary analysis that there is no simple early termination mechanism without joint observations of independent visual features. CADP controller is not meant to be the optimal solution but fits best in the context of currently available architectures.

**Subnets Divisions.**

Superficially, it looks like convolutional classifiers in ProgNets parallel type are simply separable CNNs which are later concatenated together. However, subnets may play different roles during joint training phase. Fig. 4.12 depict most activated CIFAR-10 [120] validation images by ProgNet-p6-residual convolutional filters from first and last stages. We can observe that stage 0 filters are focusing on class specific patterns, a classical pattern in deep layers [118]. Interestingly, last stage filters show entirely different attributes. Some of the outputs in the last stage are no longer activated by unique patterns but instead activated by background. This phenomenon indicates late stage features are more context activated providing that unique patterns are sufficient because features from different stages are always concatenated before N-way classification layer. It is still unknown whether this phenomenon happens only if models are saturated in late stages because we did not spot similar results on ImageNet models.

### 4.7 Conclusion

In this work, we have successfully established a progressive deep neural network structure which is able to adapt its network inference structure and computational complexity to images with different visual recognition complexity. This progressive structure is able to scale up its discriminative power and achieve higher recognition capability by activating and executing more analysis units of the deep neural networks for more difficult vision
Figure 4.12: Most activated CIFAR-10 validation images from different stages in ProgNet-p6-residual. Left: most activated images of stage 0(first) 17th layer filters. Left: most activated images of stage 5(last) 17th layer filters. **Best viewed in color.**

analysis tasks or image sets. We have developed a multi-stage progressive structure, called ProgNet [168], for deep neural networks, with each stage being separately evaluated for output decision and all stages being activated in a sequential manner with progressively increased complexity and visual recognition power. We developed a recurrent neural network method to learn the confidence analysis and decision policy for early termination. Our extensive experimental results on the CIFAR-10 and ImageNet datasets demonstrated that the proposed ProgNet is able to obtain more than 10 fold complexity scalability while achieving the state-of-the-art performances with a single network model.
Chapter 5

Real-time Deep Object Detection Models on Edge Devices

5.1 Introduction

Object detection is one of the most important and enabling steps in various computer vision tasks. There is an increasing demand to deploy high performance object detection models to edge devices that are widely diverged in architectures. Unlike the majority of image classification models which consist of convolution, batch normalization [110], non-linear activation layers, object detection models are more complex and usually require customized operators to process prioritized bounding box representations. Mapping such workloads to a variety of hardwares, including embedded CPUs, mobile GPUs, FPGAs and ASICs(e.g., the TPUs [169]) is significantly more challenging.

Current deployment process is strongly obscured by large gap between research platforms and real end user hardwares. To embrace highest efficiency of development and training, it is a common strategy for researchers to develop models on GPU workstations and train on GPU clusters with current mainstream deep learning frameworks, such as TensorFlow, Caffe, MXNet, and PyTorch. These frameworks provide high performance
operators (e.g., Convolution, Fully-connected, BatchNorm) out of the box on CPU and
GPU (more specifically Nvidia GPUs) with native hardware level optimized computing li-
braries. For example, Intel’s MKLDNN and Nvidia’s CuDNN for selective X86 CPUs and
Nvidia GPUs, respectively. These deep learning frameworks also provide APIs for adding
customized operators written in C/C++, even Python for fast prototyping. Given these flex-
ible and convenient developing platform, we are observing skyrocketing performance gains
of object detection models. Figure 5.1 illustrates the object detection performance improve-
ment in the recent three years. Note that COCO dataset is extremely difficult dataset for
object detection, which contains near real world scenarios and has tiny objects in those
scenes.

Despite the notable achievements in object detection, deploying such models is still
cumbersome. As depicted in Figure 5.2, various hardware targets diverge in terms of mem-
ory structures, compute functional units and therefore require explicit handling of instruc-
tion scheduling. Implementing the simplest add(+) operation for maximum throughput
requires knowledge of the underlying hardware architectures. This divergence is introduc-
ing significant amount of replicated re-engineering work in the process of deploying state
of the art models to products.

Figure 5.1: COCO object bounding box detection mean average precision leader in recent years,
comparing with baseline.
To this end, we have investigated the challenges involved in object detection model deployment and developed a solution to accelerate the process and reduce the duplicate engineering efforts given emerging new hardwares. We first summarize the challenges in Section 5.2, and propose the methods in Section 5.3. We then provide experimental results in Section 5.4, and conclude the work in Section 5.5.

5.2 Challenges

In this section, we survey the difficulties of deploying efficient deep object detection models on edge devices, from both hardware and software scopes. In this work, we cover four categories of hardware devices: CPU, GPU, FPGA, ASIC, and mainly focus on mobile CPU(e.g., ARM cores) and GPUs(e.g., Mali or PowerVR).

5.2.1 Hardware limitations

Modern deep object detection networks rely on base feature extraction backbone which are usually pre-trained on ImageNet image classification tasks. This step enables generic image feature distillation and is preferable in various vision tasks. Therefore the early stages of object detection networks are identical to image classification networks. One
Table 5.1: Theoretical GFLOP Estimations of popular Convolutional Neural Networks with resolution $512 \times 512$.

<table>
<thead>
<tr>
<th>Model</th>
<th>Theoretical GFLOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet [7]</td>
<td>8.6</td>
</tr>
<tr>
<td>ResNet-50 [15]</td>
<td>40.0</td>
</tr>
<tr>
<td>DenseNet-201 [16]</td>
<td>43.4</td>
</tr>
<tr>
<td>VGG-16 [19]</td>
<td>122.8</td>
</tr>
<tr>
<td>MobileNet [20]</td>
<td>3.0</td>
</tr>
</tbody>
</table>

The notable difference is that input image has significant higher resolution (e.g., $512 \times 512$ comparing with typical $224 \times 224$ for image classification). This is beneficial for neural networks to locate relatively tiny objects which will otherwise being too small to be effectively represented on feature maps. This difference, however, requires more than 4 times the computational cost merely for feature extraction, not to mention the extra architectures required for bounding boxes detection task. Table 5.1 shows theoretical computational requirements of popular convolutional neural networks with input size $512 \times 512$, denoted by Gig FLOP. A modern mobile CPU on mainstream smart phones in 2017 is able to process at roughly 100 GFLOP per second. Given this number and network GFLOP estimations, we are concluding that no smart phone CPU is able to run deep object detection models backed by these base networks at real time (25+ FPS). Mobile GPUs can deliver higher FLOPs and is more suitable for parallel execution of deep neural networks.

Extra cautious should be used to accommodate the performance gap between the network requirements and hardware limitations. As we will cover in the following sections, there are non-trivial operations appended to the base network and post-processing steps. It is not possible to fully utilize the hardware in the entire pipeline.

5.2.2 Missing Operator Support

Due to the previously mentioned divergence in hardware architectures, customized operators for object detection models are always unavailable every-time a new hardware be-
Table 5.2: Operators used by SSD Object Detection Network with ResNet 50 layer backbone.

<table>
<thead>
<tr>
<th>Count</th>
<th>Operator</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>73</td>
<td>Convolution</td>
<td>Convolution</td>
</tr>
<tr>
<td>57</td>
<td>Activation</td>
<td>ReLU Activation</td>
</tr>
<tr>
<td>61</td>
<td>BatchNorm</td>
<td>Batch Normalization</td>
</tr>
<tr>
<td>18</td>
<td>Flatten</td>
<td>Collapse to 2D matrix</td>
</tr>
<tr>
<td>16</td>
<td>elemwise_add</td>
<td>Element-wise addition</td>
</tr>
<tr>
<td>13</td>
<td>transpose</td>
<td>Transpose matrix</td>
</tr>
<tr>
<td>6</td>
<td>_contrib_MultiBoxPrior</td>
<td>Custom op: generating default bounding boxes</td>
</tr>
<tr>
<td>3</td>
<td>Concat</td>
<td>Matrix concatenation</td>
</tr>
<tr>
<td>2</td>
<td>Reshape</td>
<td>Change matrix shape</td>
</tr>
<tr>
<td>1</td>
<td>Pooling</td>
<td>Global Average Pooling</td>
</tr>
<tr>
<td>1</td>
<td>softmax</td>
<td>Softmax</td>
</tr>
<tr>
<td>1</td>
<td>_contrib_MultiBoxDetection</td>
<td>Custom op: generating bounding box outputs</td>
</tr>
</tbody>
</table>

comes is the targeting platform. Targeting mainstream contexts (e.g., CPU and CUDA), it is relatively effortless to port existing code between deep learning frameworks, e.g., from tensorflow to torch. However, intensive work is still needed to port CUDA to FPGA and definitely require rewritten by domain experts.

Table 5.2 summarizes the operators and usage in a typical Single-shot Multibox Object Detection model [13]. Note that there are custom operators which are not portable for new hardwares.

5.2.3 Post-processing

All modern object detection models follow a pipeline in three steps: (1) proposing a search space of box windows (exhaustive by sliding window or sparser using proposals), (2) scoring the window with classifiers and guide the spatial offset by regressors, and (3) merging windows that might belong to the same object. This last stage is commonly referred to as “non-maximum suppression” (NMS) [170]. The de facto standard for NMS is a simple hand-crafted test time post-processing, which we call GreedyNMS. The algorithm greedily selects high scoring detections and deletes close-by less confident neighbors since they are
Non-maximum Suppression is a commonly used post-processing step to remove duplicate detection. Likely to cover the same object with different bounding boxes results. This algorithm is simple, fast, and surprisingly competitive compared to proposed alternatives. Soft-NMS [171], a minor tweak to GreedyNMS which takes context into consideration, proved to achieve better performance. Despite the effectiveness, the effort to bring NMS and similar post-processing into object detection models is rarely seen in research community. The reasons are two fold: (1) It is difficult to beat the simple greedy algorithm. (2) The GreedyNMS itself is efficient and proved to be effective. However, when it comes to brand new hardware platforms which has poor support for general operators, it requires extra efforts to implement efficient GreedyNMS on edge devices, especially when the new architecture is not fully revealed and explored.

5.3 Methods

To alleviate the problems mentioned in Section 5.2 and to optimize for optimal execution efficiency on various edge devices, we propose a novel pipeline for deep object detection model deployment. We first explore a truly end-to-end neural network for object detection, which embed NMS functionality into network structure. We then introduce operator decomposition, in order to divide and conquer the composite operators which are complex
and non-universal. We also use on device auto tuning mechanism to achieve optimal kernel locally.

5.3.1 Truly End-to-end Neural Network for Object Detection

The main intuition to replace GreedyNMS post-processing is to fully utilize the highly optimized computational kernels for standard neural network operators, e.g., convolution and fully-connected. These common operators are guaranteed to be highly optimized by both hardware design and low-level software support. Despite being a key ingredient of object detection models, NMS algorithm has received relatively little attention compared to network design and optimization improvements. Inspired by Jan et al.’s work [170], we recognize that neural networks can play an important role in truly end to end object detection system, which will benefit to versatile detection model deployment.

Independent processing of image regions leads to overlapping detection with similar scores. This is not a faulty but actually a requirement of robust prediction functions: similar inputs shall lead to similar outputs. A detector that outputs only one high scoring detection per object thus has to be also conditioned on other detections: multiple detections on the same object should be processed jointly, so the detector can tell there are repeated detections and only one of them should receive a high score and the rest should be suppressed thereafter. Nowadays the loss function encourage multiple high score detections covering the same object, this is against the target of NMS.

From this analysis we observe two key ingredients are necessary in order to train a truly end to end detector that produce exactly one confident detection per real object: (1) The loss function must penalize multiple detection over single object, without sacrifice recall rate. (2) Joint processing of neighboring detections is required to gather necessary local information.
Training procedure

Typical object detectors are trained as a combination of classifiers and bounding box regressors. For classifiers that make predictions of foreground/background, the labels are generated from the Intersection-over-Union(IoU) overlap between reference anchor boxes and object annotations. When doing so the spatial relation between detector outputs and the annotations is being neglected and little work has addressed this. In this work, we follow YOLOv3 [172] and adapted the idea in Tnet [173] that smoothly decrease the weight for positive samples, in order to force classifiers learn their neighbors, and make decisions jointly.

Training Loss

The training objective is to reduce the score of all detections that belong to the same object, except the best one of them. To that end, we match every annotation to all predictions that overlap at least 0.5 IoU and choose the maximum scoring detection among them as the one positive training sample. This yields a label $y_p$ for every location $p$ in the input grid. Since the number of background detections (negative samples) are much more than positive samples, it is necessary to weight the loss terms to balance the two. We can describe the training loss with a standard object detector loss and a weighted logistic loss

$$\mathcal{L}(x) = \frac{1}{\sum_{p \in G} w_{y_p}} \sum_{p \in G} w_{y_p} \log(1 + \exp(-y_p f(x_p))) + \sum_{z \in (x,y,h,w) | t_z - \sigma(b_z)| + \mathcal{L}_{\text{crossentropy}}}$$

Since the input and output are one-to-one mapped, it is straightforward to train a convolutional network to minimize the loss. The major difference between our E2E network and Tnet is the combined training loss. In Tnet, it does not take into consideration the intermediate feature presentations, where our E2E network fully utilize.
Table 5.3: Comparison between E2E net and Tnet with Faster-RCNN on COCO person detection.

<table>
<thead>
<tr>
<th>Methods</th>
<th>All $AP_{0.5}$ $AP_{0.95}^{0.5}$</th>
<th>Occlusion [0, 0.5) $AP_{0.5}$ $AP_{0.95}^{0.5}$</th>
<th>Occlusion [0.5, 1] $AP_{0.5}$ $AP_{0.95}^{0.5}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>GreedyNMS val</td>
<td>65.6 35.6</td>
<td>65.2 35.2</td>
<td>35.3 12.1</td>
</tr>
<tr>
<td>Tnet [173] val</td>
<td>66.8 36.5</td>
<td>66.4 36.2</td>
<td>36.3 12.8</td>
</tr>
<tr>
<td>E2E val</td>
<td><strong>67.2</strong> <strong>37.1</strong></td>
<td><strong>67.1</strong> <strong>36.8</strong></td>
<td><strong>36.7</strong> <strong>13.0</strong></td>
</tr>
<tr>
<td>GreedyNMS test</td>
<td>65.0 35.5</td>
<td>61.8 33.8</td>
<td>30.3 11.0</td>
</tr>
<tr>
<td>Tnet [173] test</td>
<td>65.9 36.2</td>
<td>66.1 35.7</td>
<td>32.9 11.6</td>
</tr>
<tr>
<td>E2E test</td>
<td><strong>66.8</strong> <strong>36.6</strong></td>
<td><strong>66.9</strong> <strong>36.5</strong></td>
<td><strong>34.1</strong> <strong>12.2</strong></td>
</tr>
</tbody>
</table>

Training Parameters

The model is trained on Darknet-53 [172] which is pre-trained ImageNet. The additional layers are initialized randomly with Xavier. We use learning rate $10^{-4}$ with linear warm-up from $10^{-6}$ in the first 5000 iterations. We use standard weight decay of $10^{-5}$, a momentum of 0.9 and gradient clipping of 10 with Stochastic Gradient Descend (SGD).

Performance Evaluation

To evaluate the NMS performance, we experimentally evaluate the proposed architecture on the COCO dataset. To compare with related work, we follow the same ”person” category in COCO dataset. All results are measured in average precision (AP), which is the area under the recall-precision curve. The overlap threshold (for matching detections to objects) is 0.5 IoU (denoted as $AP_{0.5}$). COCO also uses stricter criteria to encourage better localization quality, one such metric averages AP evaluated over several overlap criteria in the range [0.5, 0.95] in 0.05 increments, which we denote by $AP_{0.95}^{0.5}$. The AP results are listed in Table 5.3. Our E2E network achieved on average better results than GreedyNMS with on par speed. We consider these encouraging results, confirming that indeed the convolutional network is capable of producing similar and even better performance than greedy algorithm based NMS functions.
5.3.2 Operator Decomposition

In Section 5.2.2 we discussed the weakness of current deep object detectors which require significant amount of customized operators. To alleviate this problem, we propose to use graph level operator decomposition. More specifically, operator decomposition is a process that analysis the internal operations applied in custom operators, and use standard math/neural network operators to substitute the chunky operator. For example, given a typical sigmoid operator which does the following math operation

\[
S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}
\]  

(5.2)

Anytime sigmoid function is not implemented, as long as exponential is available, we can substitute sigmoid as the example in Fig. 5.4. After we rewrite all customized operator with standard math operators in graph level, an SSD network now consists of standard operators only, which is listed in Table 5.4. It will significantly reduce the porting effort to any new hardware with fundamental operator supports.

5.3.3 Graph-level Optimization

Graph representation is an computational flow of necessary operators that are grouped by human logic at operator level. This is a result of high level presentation designed by human. In deep learning frameworks, we work with high level operators such as convolution layers and activation layers. It is optimal for us to dispatch modules with different functionalities.
Table 5.4: Operators used by De-composed SSD Object Detection Network with ResNet 50 layer backbone.

<table>
<thead>
<tr>
<th>Count</th>
<th>Operator</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>173</td>
<td>slice_axis</td>
<td>Slicing channel</td>
</tr>
<tr>
<td>98</td>
<td>Scalar</td>
<td>Scalar operations (addition, subtraction, etc.)</td>
</tr>
<tr>
<td>86</td>
<td>Concat</td>
<td>Matrix concatenation</td>
</tr>
<tr>
<td>73</td>
<td>Convolution</td>
<td>Convolution</td>
</tr>
<tr>
<td>61</td>
<td>BatchNorm</td>
<td>Batch Normalization</td>
</tr>
<tr>
<td>57</td>
<td>Activation</td>
<td>ReLU Activation</td>
</tr>
<tr>
<td>24</td>
<td>Elemwise</td>
<td>Element-wise operations (addition, subtraction, etc.)</td>
</tr>
<tr>
<td>21</td>
<td>Reshape</td>
<td>Change matrix shape</td>
</tr>
<tr>
<td>12</td>
<td>Flatten</td>
<td>Collapse to 2D matrix</td>
</tr>
<tr>
<td>12</td>
<td>transpose</td>
<td>Transpose matrix</td>
</tr>
<tr>
<td>6</td>
<td>Broadcast</td>
<td>Broadcasting operations (addition, subtraction, etc.)</td>
</tr>
<tr>
<td>2</td>
<td>exp</td>
<td>Exponential</td>
</tr>
<tr>
<td>2</td>
<td>where</td>
<td>Branch selection</td>
</tr>
<tr>
<td>1</td>
<td>Pooling</td>
<td>Global Average Pooling</td>
</tr>
<tr>
<td>1</td>
<td>softmax</td>
<td>Softmax</td>
</tr>
</tbody>
</table>

However, underlying hardware treat computational flow in a entirely different way. For example, addition operation followed by a non-linear activation function is usually divided into two operators. On many devices, it is more efficient to fuse these two operations in favor of hardware restrictions (cache and bandwidth). This fusing strategy is illustrated in Figure 5.5.

Similarly, operators can be split into multiple operators. Unlike manual operator de-

![Figure 5.5: Graph-level operator fusing strategy.](image-url)
composition discussed in previous section, this splitting step, as well as operator fusing, are executed in graph level systematically with optimization constraints.

5.3.4 On-device Optimization

Different devices diverge in their architecture and computational capabilities, therefore it is crucial to use a technology named on-device optimization to tune for real performance peaks. We rely on TVM [1] to set up on-device Remote Procedure Call (RPC) server and remotely connect to device to run and benchmark our models. The optimization is done by first generate a large algorithm search space by combining different loop and tiling flavors, caching strategies, grouping choices, etc. Once search space is generated, we remotely apply exhaust search all available solutions and benchmark for the on-device performance by compiling and executing the lowered models. The search space can be reused by all operators, and can be expanded with further development.

5.4 Experimental Results

We experiment with OpenCL and Metal on Radeon Pro 455 and iPhone 8 Plus with A11 CPU, respectively. For performance comparison, we also benchmarked the converted mod-
els on Nvidia GTX1070ti GPU with Pascal Architecture, Intel HD530 Graphic Card, Intel i7-6850HQ CPU. The optimal kernels are written to configuration and benchmarked again off the device.

We illustrate the average inference speed in Figure 5.7. The detailed numbers are listed in Table 5.5. Note that From them we can clearly observe that despite iPhone which is not able to run standard SSD [13] (not implemented), we achieved 1.5-3.0 speed up on average by applying our fully end-to-end and graph-level optimization as well as on-device optimization techniques, without any loss of accuracy. We also list real world inference FPS on device, depicted in Figure 5.8. It is worth noted that by our deployment optimization, we achieve working model on iOS devices, without manually writing any iOS specific code and achieved 10FPS near real time detection speed. It is capable to analysis every two frames in real time video records, and we predict this number is going to be rated at fully real-time given further optimization and performance bump in the next year.
Table 5.5: Object detection model real on-device inference speed. All measures are average in milliseconds. (++) indicate that it’s fully end-to-end and optimized model.

<table>
<thead>
<tr>
<th>Devices</th>
<th>Runtime</th>
<th>MobileNet-SSD</th>
<th>MobileNet-SSD++</th>
<th>Speed Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nvidia GTX1070ti</td>
<td>CUDNN</td>
<td>10.2</td>
<td>8.3</td>
<td>1.22x</td>
</tr>
<tr>
<td>AMD Radeon Pro 455</td>
<td>OpenCL</td>
<td>37.2</td>
<td>33.1</td>
<td>1.12x</td>
</tr>
<tr>
<td>AMD Radeon Pro 455</td>
<td>Metal</td>
<td>26.6</td>
<td>22.0</td>
<td>1.21x</td>
</tr>
<tr>
<td>Intel HD530</td>
<td>Metal</td>
<td>198.2</td>
<td>89.4</td>
<td>2.22x</td>
</tr>
<tr>
<td>CPU(<a href="mailto:I7-6850HQ@2.7G">I7-6850HQ@2.7G</a>)</td>
<td>LLVM</td>
<td>216.2</td>
<td>134.2</td>
<td>1.61x</td>
</tr>
<tr>
<td>Apple A11 chip (@2.39G)</td>
<td>Metal</td>
<td>-</td>
<td>98.4</td>
<td>Inf</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nvidia GTX1070ti</td>
<td>CUDNN</td>
<td>21.5</td>
<td>15.4</td>
<td>1.40x</td>
</tr>
<tr>
<td>AMD Radeon Pro 455</td>
<td>OpenCL</td>
<td>216.9</td>
<td>153.2</td>
<td>1.42x</td>
</tr>
<tr>
<td>AMD Radeon Pro 455</td>
<td>Metal</td>
<td>161.1</td>
<td>132.8</td>
<td>1.21x</td>
</tr>
<tr>
<td>Intel HD530</td>
<td>Metal</td>
<td>482.5</td>
<td>163.5</td>
<td>2.95x</td>
</tr>
<tr>
<td>CPU(<a href="mailto:I7-6850HQ@2.7G">I7-6850HQ@2.7G</a>)</td>
<td>LLVM</td>
<td>430.5</td>
<td>241.7</td>
<td>1.78x</td>
</tr>
<tr>
<td>Apple A11 chip (@2.39G)</td>
<td>Metal</td>
<td>-</td>
<td>179.4</td>
<td>Inf</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResNet-SSD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResNet-SSD++</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.8: Inference FPS comparison.
5.5 Conclusion

We have presented a fully end-to-end object detector based on deep neural networks. The E2E object detector is capable to process realtime videos on laptops without powerful GPUs, and enabled accurate object detection tasks on low power edge devices such as mobile phones. More importantly, this object detection model fully decoupled engineering efforts and therefore is a perfect fit for emerging new edge devices, despite their unique hardware architectures. Thanks to flexible on-device optimization scheme, it is guaranteed to achieve highly optimized execution kernels on a variety of devices. Comparing to current intensive manual work, it generates comparable performance while significantly reduce human labors.
Chapter 6

Summary and concluding remarks

This dissertation summarizes my research during my Ph.D study, covering object detection model in sequential images, model compression, progressive model inference, and on-device model optimization and deployment.

In Chapter 2, we have successfully developed an animal object segmentation scheme for highly dynamic and cluttered scenes using iterative ensemble graph cuts followed by joint object verification. Our extensive experimental results and performance comparisons over a diverse set of challenging videos with dynamic scenes, including the Camera Trap dataset and the Change Detection Challenge Dataset, demonstrated that the proposed video object segmentation framework outperforms various state-of-the-art algorithms.

In Chapter 3, we have developed a novel knowledge projection framework for deep neural networks the address the issues of domain adaptation and model compression in training simultaneously. We exploit the distinctive general features produced by the teacher network trained on large dataset, and use a learned matrix to project them into domain relevant representations to be used by the student network. A smaller and faster student network is trained to minimize joint loss designed for domain adaptation and knowledge distillation simultaneously. Extensive experimental results have demonstrated that our unified training framework provides an effective way to obtain fast high-performance neural networks on
small datasets with limited labeled samples.

In Chapter 4, we have successfully established a progressive deep neural network structure which is able to adapt its network inference structure and computational complexity to images with different visual recognition complexity. This progressive structure is able to scale up its discriminative power and achieve higher recognition capability by activating and executing more analysis units of the deep neural networks for more difficult vision analysis tasks or image sets. We have developed a multi-stage progressive structure, called ProgNet, for deep neural networks, with each stage being separately evaluated for output decision and all stages being activated in a sequential manner with progressively increased complexity and visual recognition power. We developed a recurrent neural network method to learn the confidence analysis and decision policy for early termination. Our extensive experimental results on the CIFAR-10 and ImageNet datasets demonstrated that the proposed ProgNet is able to obtain more than 10 fold complexity scalability while achieving the state-of-the-art performances with a single network model.

And finally in Chapter 5, we revisit the problem of deploying high performance object detector models to real hardware devices, especially low power edge devices. We have presented a fully end-to-end object detector based on deep neural networks. The end-to-end object detector is capable to process real-time videos on laptops without powerful GPUs, and enabled accurate object detection tasks on low power edge devices such as mobile phones. More importantly, this object detection model fully decoupled engineering efforts and therefore is a perfect fit for emerging new edge devices, despite their unique hardware architectures.
Bibliography


[31] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hier-
archies for accurate object detection and semantic segmentation. In Computer Vision 
and Pattern Recognition, 2014.


[33] Jasper RR Uijlings, Koen EA van de Sande, Theo Gevers, and Arnold WM Smeul-
ders. Selective search for object recognition. International journal of computer 

[34] Ming Ming Cheng, Ziming Zhang, Wen Yan Lin, and Philip Torr. Bing: Binarized 
normed gradients for objectness estimation at 300fps. In Computer Vision and Pat-

[35] Zoran Zivkovic. Improved adaptive gaussian mixture model for background sub-

[36] Antoine Monnet, Anurag Mittal, Nikos Paragios, and Visvanathan Ramesh. Back-
ground modeling and subtraction of dynamic scenes. In Computer Vision, 2003. Pro-


[38] Nuria M Oliver, Barbara Rosario, and Alex P Pentland. A bayesian computer vision 
system for modeling human interactions. Pattern Analysis and Machine Intelligence, 


[55] Xiaobo Ren, Tony X. Han, and Zhihai He. Ensemble video object cut in highly
dynamic scenes. *Proceedings of International Conference on Computer Vision and

[56] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross
Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture

[57] Bingpeng Ma, Yu Su, and Frédéric Jurie. Local descriptors encoded by fisher vectors

Changedetection. net: A new change detection benchmark dataset. In *2012 IEEE
Computer Society Conference on Computer Vision and Pattern Recognition Work-

[59] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisser-


[61] Katerina Fragkiadaki, Pablo Arbeláez, Panna Felsen, and Jitendra Malik. Learning
to segment moving objects in videos. In *Proceedings of the IEEE Conference on

Fully connected object proposals for video segmentation. In *Proceedings of the


[70] Rui Wang, Filiz Bunyak, Guna Seetharaman, and Kannappan Palaniappan. Static and moving object detection using flux tensor with split gaussian models. In *Com-


[87] Cristian Bucilua, Rich Caruana, and Alexandru Niculescu-Mizil. Model compres-
sion. In Proceedings of the 12th ACM SIGKDD international conference on Knowl-


Zhi Zhang was born in Zhejiang Province, China. He received the B.S. degree from the School of Electronic and Information Engineering in 2012. He started M.S. research in University of Missouri in the same year, under the advisory of Dr. Zhihai He. He received M.S. degree in computer engineering in 2014 and continued research as doctorate student. In 2017, he had amazing internship at Amazon Web Services as applied scientist. His research interests include computer vision, machine learning and deep learning.

He will join Amazon after graduation, to continue his work on developing state-of-the-art computer vision algorithms and products. He is dedicated to bridge the latest research to ease the daily life in everywhere.