Deep learning for visual recognition at pixel, object, and image levels

Kuen, Jason Wen Yong

2019

Kuen, J. W. Y. (2019). Deep learning for visual recognition at pixel, object, and image levels.
Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/107566>

<https://doi.org/10.32657/10356/107566>

**DEEP LEARNING FOR VISUAL RECOGNITION AT
PIXEL, OBJECT, AND IMAGE LEVELS**

JASON KUEN WEN YONG

SCHOOL OF ELECTRICAL & ELECTRONIC ENGINEERING

2019

Deep Learning for Visual Recognition at Pixel, Object, and Image Levels

Jason Kuen Wen Yong

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

2019

To my parents and brother for their love and support

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

16 August 2019

.....

Date

Jason Kuen

.....

Jason Kuen Wen Yong

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

16/8/2019.

Date

Professor Tan Yap Peng

Author Attribution Statement

This thesis contains material from 4 papers published in the following peer-reviewed conference proceedings where I was the first and/or corresponding author.

Chapter 3 is published as Jason Kuen, Zhenhua Wang, Gang Wang. “Recurrent Attentional Networks for Saliency Detection”. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016).

The contributions of the co-authors are as follows:

- I came up with the initial idea, wrote the experimental code, and ran all experiments.
- Dr. Zhenhua Wang, Dr. Gang Wang, and I contributed to brainstorming and idea discussions.
- I prepared the paper and conference rebuttal drafts. Dr. Zhenhua Wang and Dr. Gang Wang reviewed them and provided recommendations accordingly.
- Dr. Zhenhua Wang assisted in the development of the figures in the paper.

Chapter 4 is published as Jason Kuen, Federico Perazzi, Zhe Lin, Jianming Zhang, Yap-Peng Tan. “Scalable Object Detection with Auto-Encoder Weight Transfer Network”. *IEEE International Conference on Computer Vision (ICCV)* (2019).

The contributions of the co-authors are as follows:

- I came up with the initial idea, wrote the experimental code, and ran all experiments.
- Dr. Federico Perazzi, Dr. Zhe Lin, Dr. Jianming Zhang, Prof. Yap-Peng Tan, and I contributed to brainstorming and idea discussions.
- I prepared the paper and conference rebuttal drafts. Dr. Federico Perazzi, Dr. Zhe Lin, and Prof. Yap-Peng Tan reviewed them and provided recommendations accordingly.

Chapter 5 is published as Jason Kuen, Xiangfei Kong, Gang Wang, Yap-Peng Tan. “DelugeNets: Deep Networks with Efficient and Flexible Cross-layer Information Inflows”. *International Conference on Computer Vision Workshop (ICCVW) on Compact and Efficient Feature Representation and Learning in Computer Vision (CEFRL)* (2017).

The contributions of the co-authors are as follows:

- I came up with the initial idea, wrote the experimental code, and ran all experiments.
- Dr. Xiangfei Kong assisted in checking for algorithm correctness and bugs in the experimental code.
- Dr. Xiangfei Kong, Dr. Gang Wang, Prof. Yap Peng Tan, and I contributed to brainstorming and idea discussions.
- I prepared the paper and conference rebuttal drafts. Dr. Xiangfei Kong, Dr. Gang Wang, and Prof. Yap Peng Tan reviewed them and provided recommendations accordingly.

Chapter 6 is published as Jason Kuen, Xiangfei Kong, Zhe Lin, Gang Wang, Jianxiong Yin, Simon See, Yap-Peng Tan. “Stochastic Downsampling for Cost-Adjustable Inference and Improved Regularization in Convolutional Networks”. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2018).

The contributions of the co-authors are as follows:

- I came up with the initial idea, wrote the experimental code, and ran all experiments.
- Dr. Xiangfei Kong, Dr. Zhe Lin, Dr. Gang Wang, Professor Yap Peng Tan, and I contributed to brainstorming and idea discussions from algorithm perspectives.
- Mr. Jianxiong Yin, Dr. Simon See, and I contributed to brainstorming and idea discussions from *computer systems* and computational perspectives.

- I prepared the paper and conference rebuttal drafts. Dr. Xiangfei Kong, Dr. Zhe Lin, Dr. Gang Wang, Mr. Jianxiong Yin, Dr. Simon See and Prof. Yap Peng Tan reviewed them and provided recommendations accordingly.
- Dr. Xiongfei Kong assisted in writing parts of the introduction section in the paper.

16 August 2019

.....

Date

Jason Kuen

.....

Jason Kuen Wen Yong

Acknowledgements

I would first like to extend my gratitude to my supervisor Professor Tan Yap Peng for his support, guidance, and encouragement. His advice and suggestions on academic research and writing have benefited me tremendously, not to mention the strong backing from him for the many decisions I have had in the PhD study. I am also grateful to my former supervisor, Dr. Gang Wang who was instrumental in helping me to kick-start my research career. I appreciate his continued support and encouragement after his departure from the school.

I would like to thank my research collaborators whom I collaborated with at NTU: Dr. Xiangfei Kong, Dr. Zhenhua Wang, Dr. Jianlou Si, Mr. Ping Hu, and Mr. Jiuxiang Gu. Thanks to my industrial research collaborators at Adobe Research: Dr. Zhe Lin, Dr. Jianming Zhang, Dr. Federico Perazzi, Mr. Mingyang Ling, Dr. Xiaohui Shen, not forgetting my Nvidia collaborators: Mr. Jianxiong Yin, and Dr. Simon See. I appreciate the effort and time they spent in providing support, discussing ideas, and reviewing paper drafts among others in the research collaborations.

Thanks to my colleagues at ROSE Lab: Dr. Bing Shuai, Dr. Amir Shahroudy, Dr. Jun Liu, Dr. Abrar Abdul Nabi, Mr. Jianhan Mei, Ms. Kamila Abdiyeva, Mr. Yasin Yazıcı, Dr. Zenghai Chen, Dr. Ting Liu, Mr. Xingxing Wang, Dr. Rahul Rama Varior, Mr. Henghui Ding, Dr. Jinghua Wang, and Mr. Jianda Chen. I appreciate the conversations and idea-exchanging moments that we had. I would like to express my gratitude to the ROSE Lab staffs: Ms. Qian Wang, Mr. Joseph Lim, and Dr. Dennis Sng for providing a conducive lab environment for research.

Finally, many thanks to my parents and my brother. This thesis would not have been possible without their continued support, encouragement, and unconditional love.

Contents

Statement of Originality

Supervisor Declaration Statement

Authorship Attribution Statement

Acknowledgements

Summary

List of Figures

List of Tables

1	Introduction	1
1.1	Deep Learning	1
1.2	Visual Recognition	3
1.3	Motivations	5
1.4	Major Contributions	8
1.5	Thesis Organization	10
2	Literature Review	11
2.1	Deep Learning	11
2.1.1	Multi-layer Perceptron (MLP)	13
2.1.2	Convolutional Neural Network (CNN)	14
2.1.3	Recurrent Neural Network (RNN)	19
2.2	Saliency Detection (Pixel-level Recognition)	21
2.2.1	Saliency Detection Methods	22
2.2.2	Fully Convolutional Networks	23
2.2.3	Attention Mechanism in Neural Networks	23
2.3	Object Detection (Object-level Recognition)	24
2.3.1	Object Detection Networks	24
2.3.2	Scaling up Object Detection	26
2.4	Image Classification (Image-level Recognition)	28
2.4.1	Very Deep Networks	29

2.4.2	Separable Convolutions	31
2.4.3	Cost-adjustable Inference	32
2.4.4	Stochastic Regularization	32
2.4.5	Multiscale Training	33
3	Recurrent Attentional Networks for Saliency Detection	35
3.1	Introduction	35
3.2	Proposed Method	37
3.2.1	Deconvolutional Networks for Salient Object Detection . . .	38
3.2.2	Attentional Inputs and Outputs with Spatial Transformer .	40
3.2.3	Recurrent Attentional Networks for Saliency Refinement . .	42
3.3	Implementation Details	47
3.4	Saliency Training Datasets	48
3.5	Experiments	49
3.5.1	Datasets and Evaluation Metrics	50
3.5.2	Comparison with Baseline Methods	51
3.5.3	Comparison with State-of-the-art Methods	53
3.5.4	Comparison with Recent Recurrent Network-based Methods	55
3.6	Concluding Remarks	56
4	Scalable Object Detection with Auto-Encoder Weight Transfer Network	57
4.1	Introduction	57
4.2	Weight Transfer Network	59
4.2.1	Preliminaries	59
4.2.2	Normalizations	62
4.3	Autoencoder Weight Transfer Network	65
4.4	Experiments	69
4.4.1	Implementation Details	69
4.4.1.1	Training and evaluation datasets for seen classes $\check{D}$	69
4.4.1.2	Evaluation datasets for novel classes $\check{N}$	69
4.4.1.3	Classification network (CLN)	70
4.4.1.4	Detection Network (DEN)	70
4.4.1.5	WTN	71
4.4.2	Comparison with related methods	72
4.4.3	Analysis	74
4.4.3.1	Local neighbourhood preservation	74
4.4.3.2	Normalizations in WTN ⁺	76
4.4.3.3	Choice of feature normalization	77
4.4.3.4	Computational efficiency	78
4.5	Concluding Remarks	78
5	DelugeNets: Deep Networks with Efficient and Flexible Cross-Layer Connections	81

5.1	Introduction	81
5.2	Deluge Networks	85
5.2.1	Composite Layer	85
5.2.2	Cross-layer Depthwise Convolutional Layers	87
5.2.2.1	Advantages	88
5.2.3	Block Transition	90
5.3	Experiments	90
5.3.1	CIFAR-10 and CIFAR-100	91
5.3.1.1	Datasets	91
5.3.1.2	Implementation	91
5.3.1.3	Results	92
5.3.1.4	Parameter efficiency	93
5.3.1.5	Computational complexity	93
5.3.1.6	Ablation study	94
5.3.1.7	Cross-layer connectivity	96
5.3.2	ImageNet	97
5.3.2.1	Implementation	97
5.3.2.2	Results	98
5.4	Memory Usage	100
5.5	Concluding Remarks	100
6	Stochastic Downsampling for Cost-Adjustable Inference and Improved Regularization	103
6.1	Introduction	103
6.2	Preliminaries: Conventional CNNs with Fixed Downsampling Points	106
6.3	Stochastic Downsampling Point	107
6.3.1	Downsampling Operation	108
6.3.2	Downsampling Points	109
6.3.3	Downsampling Ratios	110
6.3.4	Training with SDPoint	111
6.3.5	Regularization from Weight Sharing	112
6.4	Cost-adjustable Inference	113
6.4.1	Instance-Specific Batch Normalization	113
6.5	Experiments	114
6.5.1	CIFAR	116
6.5.1.1	Baseline Comparison	116
6.5.1.2	State-of-the-art Comparison	118
6.5.1.3	Stochastic Training and Inference	119
6.5.2	ImageNet	119
6.5.2.1	Ablation Study	121
6.5.2.2	State-of-the-art Comparison	122
6.5.2.3	Analysis	123
6.6	Concluding Remarks	125

7 Conclusion and Recommendations	127
7.1 Conclusion	127
7.2 Recommendations for further research	130
Author Publications	133
Bibliography	135

Summary

Deep (machine) learning in recent years has significantly increased the predictive modeling strengths for applications in the areas of computer vision, speech recognition, and natural language processing. Being heavily data-driven, conventional general-purpose deep learning methods mainly rely on training data to learn task-specific data representations. In this thesis, we focus on leveraging novel learning algorithms in deep learning to tackle visual recognition challenges at the three key levels – namely pixel, object, and image levels. Throughout the thesis, we demonstrate how each and every of our proposed methods can address a specific visual recognition problem.

Firstly, we apply deep learning to improve the saliency detection performance of deep networks. A coarse saliency map of the input image’s salient object is initially predicted by a convolutional-deconvolutional network (or equivalently a fully convolutional network). Following that, the coarse saliency map is sequentially refined by a separate network called Recurrent Attentional Convolutional-Deconvolutional Network (RACDNN) that shares its network weights across its sequential steps. At every sequential (temporal) step, the same RACDNN is applied to a newly selected/attended image sub-region and it performs saliency map refinement for that particular sub-region. Despite having many learnable network parameters and no access to additional training data, RADCNN can be trained well and it outperforms the “single-step” (coarse saliency detection) baseline and state-of-the-art saliency detection methods by large margins, on several saliency detection benchmark datasets.

Secondly, we introduce auto-encoder weight transfer networks (AE-WTN⁺), a deep model for scaling up object detection. Conventional WTN transfers object class knowledge from the classification weights of a pretrained large-scale image classification network, to the classification weights of an object detection network.

However, they suffer from under-fitting and are not able to preserve the rich object class information of novel classes. In this work, we propose a new WTN method – AE-WTN⁺ which makes use of normalization techniques to ease network optimization. Inputs and intermediate features are normalized to improve trainability for reduced under-fitting. More importantly, a novel auto-encoding (reconstruction) loss is proposed to encourage information preservation of all classes in the weight predictions of AE-WTN⁺. This enables the predicted weights of AE-WTN⁺ to maintain the class discriminability and neighborhood relationships contained in the pre-trained large-scale classification weights. Experiments on large-scale detection datasets validate the effectiveness of AE-WTN⁺ on both seen and novel classes.

Thirdly, we improve the efficiency of cross-layer network connections for very deep convolutional network in the context of image classification. This gives rise to a new convolutional network architecture called Deluge Networks (DelugeNets) which pass the output (feature activations) of any preceding composite network layer as direct inputs to all succeeding composite layers. For the incoming feature activations of preceding composite layers, the succeeding composite layers first apply rather lightweight cross-layer depthwise convolutional weights on them for feature aggregation. The heavy lifting is carried out by the last convolutional weight layer in each preceding composite layer that performs cross-channel convolutions. The proposed technique enables unimpeded information flow among many layers in very deep DelugeNets with great efficiency. Experiments on small-scale and large-scale datasets show that DelugeNets perform competitively to state-of-the-art deep convolutional networks at lower computational and parameter costs.

Fourthly, we propose Stochastic Downsampling (SDPoint), a method to overcome a major limitation suffered by conventional deep convolutional networks – their computational costs during inference are fixed and cannot be changed. SDPoint downsamples the feature maps at a randomly selected point (layer index) in the

Summary

network hierarchy during training. The many downsampling configurations known as SDPoint instances each entail a unique downsampling point and downsampling ratio. The same convolutional network weights are shared across the different SDPoint instances that downsample feature maps differently. During inference, the computational cost of a network trained with SDPoint can be instantaneously adjusted to fit a given computational budget, by manually selecting an appropriate SDPoint instance. Additionally, sharing weights between different feature map scales provides significant regularization benefits, making the networks trained with SDPoint less sensitive to scales and more competent. In image classification experiments, SDPoint demonstrates significant cost-accuracy and computational advantages, over independently-trained (non-weight-shared) baselines with various inference costs.

List of Figures

1.1	Illustrations of a typical traditional machine learning or predictive modelling pipeline (Top) and a pipeline where deep learning is involved (Bottom). In the traditional pipeline, the sensory data sequentially undergoes a number of distinct processing steps (e.g., feature extraction, feature selection, model selection & training) before predictions can be made. Whereas, in the deep learning pipeline, all the intermediate processing steps are handled by a single deep neural network model.	2
1.2	Examples of visual recognition tasks from the three recognition levels. The leftmost column is saliency detection, a pixel-level recognition task. All image pixels should be “recognized” and have output/prediction values (bottom part). The middle column is object detection, an instance-level recognition task. The objective is to recognize object instances and the outputs are represented by bounding boxes. Lastly, in the rightmost column, the task is image classification (image-level recognition) where the outputs are the predicted image-level semantic classes/categories.	5
2.1	Multi-layer perceptron (MLP) network architecture (left) and its basic network building block (right).	12
2.2	ReLU and Leaky ReLU activation functions.	13
2.3	Applying convolutional weights to input image and intermediate feature maps in a CNN.	15
2.4	A typical CNN architecture with multiple stages and pooling layers between the stages.	16
2.5	Average and max pooling operations. The different colours correspond to different pooling windows. Average pooling computes the mean of values in each of the pooling windows, while max pooling obtains the max value.	17
2.6	Input images that maximally activate some feature channels in a CNN. It can be seen that the lower layers learn to detect lower-level visual concepts, and vice versa. They are borrowed from the paper of Mahendran and Vedaldi [1].	18
2.7	A simple RNN example showing how weight sharing is carried out for input-to-feature and feature-to-feature transformations, at different time steps.	19

2.8	The differences in terms of (ground-truth) saliency maps between eye fixation-based saliency detection and segmentation-based saliency detection. The eye fixation’s saliency map consists of blurry and somehow vague Gaussian blobs which are imprecise at local pixel level.	22
2.9	On the left, the object detector fails to detect the not-so-common object classes (<i>wheelchair</i> , <i>stethoscope</i>), due to the small sizes of mainstream object detection datasets. The second detector takes into account an extra knowledge source (albeit without bounding box annotations) to learn knowledge about the not-so-common classes during training, and is able to perform detection for those classes. .	28
2.10	A visual comparison between normal (mixed-channel) spatial convolution and depthwise convolution.	31
3.1	An example of applying recurrent attention-based saliency refinement to an initial saliency map produced by convolutional-deconvolutional network. Compared to the initial saliency map, the refined saliency map has significantly sharper edges and preserves more object details.	36
3.2	A generic convolutional-deconvolutional network for saliency detection.	38
3.3	To map the input feature maps U to output feature maps V , spatial transformer transforms output point coordinates on V to sampling point coordinates on U	40
3.4	Overall architecture of our Recurrent Attentional Convolutional-Deconvolutional Network (RACDNN).	43
3.5	Precision-recall curves, with average precisions.	49
3.6	Qualitative saliency results of some evaluated images. From the leftmost column: input image, saliency ground-truth, the saliency output maps of our proposed method (CNN-DecNN + RACDNN) with mean-shift post-processing, MCDL [2], MDF [3], RRWR [4], BSCA [5], DRFI [6], RBD [7], DSR [8], MC [9], and HS [10].	53
4.1	Comparison between network architectures of WTN and WTN ⁺ . The green rectangles correspond to layers with learnable parameters, whereas the grey rectangles are layers without learnable parameters.	62
4.2	A Group Normalization (GN) [11] layer splits the input channels into a number of channel groups (each group is represented by a distinct colour). The top GN splits the channels into 5 groups, while the bottom one splits them into 3 groups. For any GN configuration shown here, all the channel groups share the same number of channels, although this is not a strict requirement.	64
4.3	Autoencoder-WTN ⁺ (AE-WTN ⁺) that retains rich class information of all classes in the latent space W_{DEN} and effectively exploits novel class knowledge during training.	67

4.4	Qualitative results of the proposed AE-WTN ⁺ . The detection boxes with blue labels are seen object classes $\check{D}$, while the boxes with red labels are novel classes $\check{N}$. Our method can handle a variety of novel classes and concepts, while performing well on seen classes. . .	75
4.5	The overlapping count (vertical axis) between CLN's nearest neighbours and the nearest neighbours obtained by the WTN model of interest (either AE-WTN ⁺ , WTN ⁺ , or WTN), given varying numbers of nearest neighbours (horizontal axis).	76
5.1	Deluge Network components: (a) a composite layer, (b) a block transition component, and (c) a block. Red-coloured arrows indicate 1×1 cross-layer depthwise convolutions.	83
5.2	Cross-layer Depthwise Convolution. The columns correspond to feature channel indices, and the rows correspond to preceding <i>composite layer</i> indices.	86
5.3	Layer-wise L2-norms of cross-layer depthwise convolution weights. Each of the 3 columns corresponds to a different block transition stage in the networks. Vertical axes indicate the indices of the preceding <i>composite layers</i> , and horizontal axes indicate normalized L2-norm values. The longer the horizontal bar of a <i>composite layer</i> , the larger its contribution.	95
6.1	WRNs' and DenseNetBC's cost-error plots on CIFAR-10 (Top) and CIFAR-100 (Bottom). It is observed that models trained with SD-Point consistently outperform their non-SDPoint counterparts, given the same computational budgets.	115
6.2	PreResNets' [12] cost-error plots on ImageNet. PreResNet-d101 (SDPoint) only has to be trained once (as a single model), while the baseline models (without SDPoint) has to be trained separately with huge training and storage costs.	121
6.3	ResNeXts' [13] cost-error plots on ImageNet. Like Figure 6.2, any of ResNeXt-d101-c32 (SDPoint..) only has to be trained once (as a single model).	122
6.4	Some Imagenet validation examples grouped according to the minimum inference costs (FLOPs) required by ResNeXt-d101-c32 (with SDPoint) to classify them correctly, in terms of top-5 accuracy. The ground-truth label names are shown below their corresponding images.	124

List of Tables

3.1	F-measure scores (F-M) and Mean Absolute Errors (MAE) (compared with baseline methods).	52
3.2	F-measure scores (F-M) and Mean Absolute Errors (MAE) (compared with state-of-the-art methods).	52
3.3	Comparison with deep learning-based methods. *MDF is trained on a subset of HKUIS, and then evaluated on the remaining HKUIS samples.	55
3.4	Comparison with very recent recurrent network-based methods.	56
4.1	The <i>train</i> and <i>test</i> phases of object detector DEN with Autoencoder-WTN ⁺ (AE-WTN ⁺). Train phase: Before training DEN, we take the final classification FC layer’s weights of a pre-trained CLN and discard its remaining layers. AE-WTN ⁺ is then trained simultaneously with DEN to transform weights from CLN to DEN through the shared classes $\check{S}$. AE-WTN ⁺ has a reconstruction loss to reconstruct the weights of CLN’s classes $\check{C}$. The “other detection classes” ($\check{D} \setminus \check{S}$) are trained normally as conventional classification weights. Test phase: During testing, CLN’s weights of both the novel classes $\check{N}$ and shared classes $\check{S}$ can be adapted for use in DEN through the trained AE-WTN ⁺ . Given that, DEN is able to perform detection for classes $\check{N}$ in addition to the other training classes.	60
4.2	Comparison with weight transfer-related methods on evaluation datasets – OI-500 (seen classes), OI-57 (novel classes), and VG-200 (novel classes).	71
4.3	Ablation study on WTN ⁺ architecture.	77
4.4	<i>Means</i> and <i>variances</i> of post-ReLU activation norms.	78
4.5	Training time and memory usage.	78
5.1	CIFAR-10 and CIFAR-100 test errors (percentage) of existing models and DelugeNets.	89
5.2	Comparison with ResNet-like baselines on CIFAR-100 test errors. The fourth column reports the performance differences (PDiff) between baselines and DelugeNets.	94
5.3	ImageNet validation errors (single 224×224 crops).	99

6.1	Progression of feature map spatial sizes during training of a (Left) conventional CNN, (Right) with SDPoint. The <i>costs</i> here refer to computational costs measured in numbers of floating-point operations (FLOPs).	105
6.2	CIFAR-10 and CIFAR-100 validation errors (%). The GFLOPs with 2 values separated by “/” are for CIFAR-10 and CIFAR-100 respectively.	118
6.3	<i>Deterministic</i> and <i>stochastic</i> inference results of training WRN-d28-w10 [14] with different stochastic training methods on CIFAR-100. .	118
6.4	ImageNet top-1 and top-5 validation errors (%), with model parameter numbers and giga-FLOPs (GFLOPs).	120

Chapter 1

Introduction

1.1 Deep Learning

In the areas of pattern recognition and machine learning, the traditional techniques rely on hand-crafted or manually designed features (e.g., SIFT [15] & HoG [16] for computer vision, MFCC [17] for speech recognition) which serve to distinguish between data examples at high levels. The design and extraction of hand-crafted features are largely separated from learning-based techniques in the traditional predictive modelling pipeline. While hand-crafted features have achieved successes, they necessitate the expensive design effort and expertise of domain experts. Furthermore, hand-crafted features are designed for very specific predictive modelling tasks and purposes, and they generally work poorly for out-of-scope tasks. Deep (machine) learning [18] resolves such limitations associated with hand-crafted features by taking a data-driven approach instead to learn features directly from data to solve machine learning problems. Figure 1.1 illustrates the differences between the traditional predictive modelling pipeline that involves hand-crafted feature extraction and the typical pipeline for deep learning approaches.

FIGURE 1.1: Illustrations of a typical traditional machine learning or predictive modelling pipeline (Top) and a pipeline where deep learning is involved (Bottom). In the traditional pipeline, the sensory data sequentially undergoes a number of distinct processing steps (e.g., feature extraction, feature selection, model selection & training) before predictions can be made. Whereas, in the deep learning pipeline, all the intermediate processing steps are handled by a single deep neural network model.

More specifically, deep learning techniques perform hierarchical representation learning directly on the raw sensory data using multiple network layers in deep neural networks. In deep learning, the feature/representation learning and modelling (e.g., supervised learning) are unified through a single machine learning model. Since deep learning entails task-driven or task-specific representation learning, it is generally perceived that increasing the amounts and diversity of training data [19] is one of the key factors to more effective deep learning models. The learned task-specific data representations have enabled many deep learning approaches to achieve human-level [20] and even superhuman [21] performance on a wide range of artificial intelligence perception tasks.

Fundamentally, deep learning approaches are built upon the well-established concepts of *network depth* [22] and *weight sharing* [23, 24]. Network depth refers to the multilayer-ness of deep network architectures that enables learning of increasingly more abstract and rich data representations. Whereas, weight sharing helps to avoid learning redundant weights by allowing deep network weights to be shared among local parts of data, thus boosting model regularization. Depth and weight

sharing form the cornerstones of conventional deep learning models such as multilayer perceptron (MLP) [25], convolutional neural network (CNN) [26–28], and recurrent neural network (RNN) [29, 30].

Although the above-mentioned conventional deep learning models have achieved successes in many problems and applications, they are *general-purpose* methods which mainly rely on the available training data to learn task-specific representations to solve problems. However, increasing the amounts and quality of training data is an extremely resource-inefficient way to improve deep learning, since it is highly labour-intensive and it puts excessive burden on the training time. Beyond training data, the conventional deep learning methods do not adequately utilize other non-training-data priors and insights to tackle some specific problems and challenges at hand. Thus, it is highly desirable to consider improvements in the learning algorithm aspects of deep learning such as network architectures [31], network layers & components [32–34], loss functions [35], regularization [36], learning paradigms, and etc. The advancements and customisations in such fundamental aspects of deep learning can lead to more effective and possibly more efficient representation learning for the tasks at hand.

The primary focus of this thesis is on leveraging the strengths of deep learning, and designing novel deep learning algorithms directed towards several challenging problems in a number of visual recognition tasks. The next section gives a brief introduction of visual recognition along with the mainstream visual recognition tasks at pixel, object, and image levels considered in the thesis.

1.2 Visual Recognition

Visual recognition is one of the major focus areas of computer vision that revolves around methods and applications that deal with recognition of semantic entities

found in visual data. Static images and videos are examples of visual data. This thesis focuses primarily on static natural images. Semantic entities can be represented according to some high-level semantic concepts such as objects (e.g., dog, person, car), stuffs [37] (e.g., sky, grass, road), scenes [38, 39] (e.g., restaurant, museum, park). In visual recognition for static images, recognition tasks can be roughly categorised into three different recognition levels – namely **pixel**, **object**, and **image** levels. Some visual examples of recognition tasks at these three levels are presented in Figure 1.2. The following outlines what visual recognition tasks at the three levels aim to achieve:

- *Pixel-level*: Recognizing the semantic class/classes (commonly objects, stuffs) locally for every pixel of the input images.
- *Object-level*: Recognizing the semantic class/classes for all object instances belonging to a kind of semantic concept (commonly object semantics), present in the images.
- *Image-level*: Recognizing the semantic class/classes (commonly objects, scenes) associated with the images at the global image level.

In this thesis, we employ deep learning to tackle challenging problems in visual recognition tasks at the three recognition levels. For pixel-level visual recognition, we focus on improving the performance of deep networks for pixelwise *saliency detection* (**Chapter 3**). Saliency detection refers to the visual recognition task of distinguishing salient and non-salient image pixels given an input image. For object-level recognition, we focus on object detection (**Chapter 4**) and address the problem of scaling up the number of object classes handled by object detectors, without having access to bounding box annotations for a large number of classes. Lastly, for image-level recognition, we focus on two image-level object classification problems. The first is to tackle the efficiency and effectiveness issues of training

FIGURE 1.2: Examples of visual recognition tasks from the three recognition levels. The leftmost column is saliency detection, a pixel-level recognition task. All image pixels should be “recognized” and have output/prediction values (bottom part). The middle column is object detection, an instance-level recognition task. The objective is to recognize object instances and the outputs are represented by bounding boxes. Lastly, in the rightmost column, the task is image classification (image-level recognition) where the outputs are the predicted image-level semantic classes/categories.

very deep (> 200 -layer) networks for image-level classification (**Chapter 5**). The other is to address the computational requirement inflexibility suffered by visual recognition-based deep networks during test-time (**Chapter 6**), using an image-level classification task as a medium to gauge the performance.

1.3 Motivations

In the field of visual recognition, hand-crafted visual feature descriptors were the de-facto standard until they were overtaken by deep learning methods [18] in recent years. Deep learning methods outperform hand-crafted features because in deep learning, highly task-specific visual data representations are learned directly from the relevant training data and annotations to tackle certain visual recognition tasks. This thesis is motivated by the representational and task performance merits of deep learning to tackle visual recognition problems using deep learning.

Conventionally, the successes of deep learning approaches in machine learning problems depend primarily on the given training data (e.g., quality, scale, variety) to perform well. As such, deep learning is particularly well-suited for visual recognition tasks due to the large amounts of visual data (and its corresponding annotations) available. Although improving (e.g., employing better sensory input devices, enlarging the scale, enhancing the diversity) the data is a straightforward and sure-fire way to improve the performance of deep learning approaches, the overemphasis on training data is poorly aligned with human’s visual perception systems that have significantly less overwhelming “training data” requirements. Besides, there are certain beneficial qualities and capabilities we wish to impart to deep neural networks which cannot possibly be achieved by solely improving the training data and employing off-the-shelf (generic) deep learning. In this thesis, we focus on the non-training-data aspects of deep learning and propose novel deep learning algorithms to address some specific problems and challenges in the visual recognition tasks considered:

- The standard network architecture for pixel-level recognition – convolutional-deconvolutional networks (or fully convolutional networks) can be directly adapted to predict pixelwise saliency maps in an end-to-end and efficient manner, taking entire images as inputs and producing saliency maps as outputs. However, the saliency map outputs produced by convolutional-deconvolutional networks tend to be coarse and the fine details of the salient objects are lost. This is due to the general tendency for convolutional-deconvolutional networks to focus primarily on the global visual information and not being sensitive to the local details contained in the input images. To better capture the fine details, can we perform saliency detection at more local levels?
- In object detection (object-level recognition task), the existing training datasets cover a much smaller numbers (at most hundreds) of object classes, compared

to large-scale image-level recognition datasets which can reach up to several thousands of classes. The image-level training datasets can be utilized to scale up the number of detection classes in partially supervised settings when training object detectors. Despite that, such large-scale training datasets are very computationally heavy to obtain, store, and train on. Can we bypass the large-scale datasets and instead consider transferring the rich object class information learned by and compressed in easily available pre-trained image classification networks to object detection networks?

- Additive skip (residual) connections help to alleviate the trainability issues of very deep convolutional networks on image classification tasks, but they impose a strong rigidity in terms of the cross-layer connections between neural network layers over long depth distances. The cross-layer connection rigidity limits the representational capabilities of very deep neural networks. At the other extreme, using many independent full convolutional layers to provide very strong cross-layer connection flexibility brings excessive computational burdens. Is there a way to achieve a good balance between flexibility and efficiency for the cross-layer connections in very deep neural networks?
- Generally, once a deep convolutional network for image classification or another visual recognition task has been trained, its computational cost during test-time cannot be changed/adjusted. This severely limits the usefulness of convolutional networks in real-world applications and systems where the computational resource availabilities vary in time. One straightforward solution is to independently train many convolutional network models with varied computational costs, but that would be unreasonably computationally heavy. Can we develop a technique for convolutional networks that permits instantaneously adjustable computational costs of a single model during test-time, without the need of re-training or fine-tuning?

1.4 Major Contributions

In this thesis, to address the aforementioned challenging problems in visual recognition, we propose several task-specific deep learning methods. The main contributions of this thesis are ordered by chapters and summarized as follows:

- (a) To address the coarseness and missing of fine detail problems faced by deep saliency detection network, we propose Recurrent Attentional Convolutional-Deconvolutional Network (RACDNN). Coarse saliency maps are firstly generated by a conventional convolutional-deconvolutional network. Given the coarse saliency maps, RADCNN learns to sequentially attend to important image sub-regions, and performs saliency refinement for the attended sub-regions using recurrent network. Through the sequential refinement steps, fine object details are recovered and clearer/sharper saliency maps are obtained. Experiments on several saliency detection benchmark datasets using standard evaluation metrics validates the effectiveness of RACDNN. (Chapter 3)
- (b) We propose Autoencoder Weight Transfer Networks (AE-WTN⁺) to expand the number of object classes handled by object detection networks. This is achieved by transferring object class knowledge from a pre-trained large-scale image classification networks in an effective and efficient manner using an improved variant of WTN. Input and feature normalizations are added to the WTN to tackle underfitting issues suffered by conventional WTN. To further improve detection performance on novel classes during training, a reconstruction loss is proposed to encourage WTN to preserve rich class information during weight transfer, resulting in AE-WTN⁺. Experiments on large-scale detection datasets demonstrate the significant advantages of AE-WTN⁺ over existing related methods on the detection tasks involving seen and unseen (novel) object classes. (Chapter 4)

- (c) We propose cross-layer depthwise convolutions to achieve flexible yet efficient cross-layer connections in very deep convolutional networks. Deluge Networks (DelugeNets) are a variant deep convolutional network architecture that utilizes cross-layer depthwise convolutions. A cross-layer depthwise convolutional layer learns minimal depthwise weights for flexible cross-layer connections, while relying on the final 1×1 convolutional weights of the preceding composite layers for learning complicated cross-channel correlations and/or relationships. This simplified method of aggregating features from multiple preceding layers provides a good trade-off between connection flexibility and computational efficiency. Experimentally, DelugeNets performs competitively to state-of-the-art very deep networks on image classification tasks at much lower computational and memory requirements. (Chapter 5)
- (d) To enable cost-adjustable inference in convolutional networks, we propose Stochastic Downsampling (SDPoint) that downsamples feature maps of a randomly chosen “point”/location and a randomly chosen downsampling ratio in the network hierarchy during training. When the downsampling happens at an early point in the network hierarchy, the computational cost of executing the remaining network layers becomes cheaper. SDPoint shares the whole convolutional network weights across different downsampling choices. Thus, the network weights are learned to adapt to the different scales of the differently downsampled feature maps in the network. This provides significant regularization benefits to the convolutional networks. During inference, one may manually handpick any downsampling configuration of the same network model that fits a particular computational budget. In experimental evaluations on image classification, SDPoint strongly outperforms a variety of cost-adjustable baseline and state-of-the-art methods in terms of cost-accuracy performance. (Chapter 6)

1.5 Thesis Organization

The rest of this thesis is organized as follows:

- In Chapter 2, we review the fundamentals of deep learning, visual recognition tasks, and the existing works on deep learning related to the relevant visual recognition topics.
- In Chapter 3, we present Recurrent Attentional Convolutional-Deconvolutional Network (RACDNN) for improved pixel-level saliency detection.
- In Chapter 4, we present Autoencoder Weight Transfer Networks (AE-WTN⁺) for scaling up the number of object classes in object-level object detection.
- In Chapter 5, we present cross-layer depthwise convolutions along with very deep Deluge Networks (DelugeNets) for learning efficient and flexible cross-layer connections for image-level classification.
- In Chapter 6, we present Stochastic Downsampling (SDPoint) to enable cost-adjustable inference and to improve regularization in convolutional networks for image-level classification.
- In Chapter 7, we conclude this thesis and give concrete recommendations for further research.

Chapter 2

Literature Review

2.1 Deep Learning

Deep learning is a branch of machine learning centered around deep neural networks that are able to learn deep hierarchical representations with multiple levels of abstractions [18] to tackle a wide range of challenging predictive modelling and machine learning problems. Deep neural networks are able to perform complicated modelling tasks mainly because they are universal function approximators [40], and the complexity of the function that a deep neural network can compute grows exponentially with the *depth* of the network [41].

Generally, deep neural networks are equipped with large amounts of trainable network weights and excessive learning capacities which allow them to “memorize” [42] training data and are prone to overfitting issues. One crucial aspect of deep learning is *regularization*. Regularization dampens the representational and fitting strengths of deep neural networks to achieve better generalization. Majority of the regularization methods proposed to alleviate the overfitting issues of deep neural networks work either by introducing noise or random variations [27, 43–45] to the

FIGURE 2.1: Multi-layer perceptron (MLP) network architecture (left) and its basic network building block (right).

training processes or by constraining the fitting strengths of neural networks based on certain desirable generalization priors [46–48].

In addition to constraining deep neural networks through the abovementioned regularization methods, *weight sharing* is another crucial aspect of deep learning that focuses on constraining the learning processes and solution spaces of deep neural networks. In a weight-shared neural network, the same network weights/parameters are shared among different parts of the data. In general, weight sharing techniques effectively reduce the numbers of trainable network weights to counter overfitting, while striving to maintain the necessary representational strengths of deep neural networks. Modern deep neural network architectures employ one or more forms of weight sharing to attain strong generalization and performance.

We first introduce multi-layer perceptron [25] (MLP), an early and fundamental deep neural network architecture which is the precursor to the recent deep architectures. Following that, we review and discuss recent deep network architectures, convolutional neural networks (CNN) and recurrent neural networks (RNN) which are the current standard in deep learning.

FIGURE 2.2: ReLU and Leaky ReLU activation functions.

2.1.1 Multi-layer Perceptron (MLP)

A multi-layer perceptron (MLP) [25] network consists of *fully-connected* weight layers (with real-valued weights) stacked in a sequential manner. In a fully-connected (FC) layer, there are feedforward dense connections (weights) that connect all input dimensions to the entire output dimensions. An example of MLP network architecture is shown in Figure 2.1. The weights of an FC layer are in the form of (2D) matrix where one dimension corresponds to the input, while the other corresponds to the output. Whereas, data examples (inputs and outputs) are represented by 1D vectors. To obtain features or output of an input example, matrix multiplication is performed between the input vector and weight matrix. During training, the weights of all FC layers are jointly tuned (also known as *learning*) to maximize/minimize some training objectives. The number of network layers of the network is referred to as *depth*, and a *deep* neural network contains many layers. By treating each network layer as a function and the entire MLP network as a composition of functions, errors from the training loss function and output layer can be passed to the earlier layers using the chain rule of calculus. This is known as *backpropagation* [49] training, which has been the de facto standard for computing gradients of weights in deep neural networks.

The essence of deep learning lies in the *depth* of the deep neural networks. In deep

neural networks, the *depth* is established by the *non-linearity* which is needed to model non-linear input-output relationships and/or decision boundaries. Thus, a non-linear (element-wise) activation function is generally applied to the outputs of FC layers or other weight layers. Without any non-linear activation functions, the multiple weight matrices of multiple layers of a MLP can be readily collapsed into a single weight matrix that results in only linear transformation. A non-linear activation function is usually implemented as a non-linear activation layer as illustrated in the right of Figure 2.1. The commonly used activation functions in recent years are Rectified Linear Unit (ReLU) [50] function and Leaky Rectified Linear Unit (Leaky ReLU) [51] function. Figure 2.2 shows the plots (shapes) of these two activation functions. ReLU and Leaky ReLU are linear (identity) for all positive input values, allowing gradients to pass through much easier compared to traditional saturating activation functions such as hyperbolic tangent and sigmoid. In the remaining thesis chapters, we stick to either ReLU and Leaky ReLU in the neural networks in our proposed methods.

The FC layers (with 2D weight matrices) in MLP are at the core of weight-sharing network architectures such as convolutional networks (CNN) and recurrent networks (RNN). Instead of applying large FC layer/layers to the entire data (as in MLP), a weight-sharing network applies (and shares the weights of) smaller FC layer/layers to the local pieces of the data.

2.1.2 Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) [26–28] are neural networks that share network weights across local input image and feature map patches. It is one of the most effective machine learning model for computer vision and has also obtained successes in speech recognition [52] and natural language processing [53]. All the remaining thesis chapters involve CNNs to great extents.

FIGURE 2.3: Applying convolutional weights to input image and intermediate feature maps in a CNN.

While CNN and MLP have a high degree of commonality (e.g., real-valued network weights, non-linear activation function, training algorithm), they differ significantly in terms of both how data (input, intermediate features) is represented and how network weights are applied to the input. A visual example of the data representation and weight application in CNN is provided in Figure 2.3. In contrast to MLP’s vector-based data examples, data examples in CNN are represented by 3D tensors¹. One of the 3 dimensions is for feature channels (e.g., RGB channels in input images), and the other two are for spatial width and height. Whereas, for MLP, an image data example has to be flattened into a 1D vector, ignoring its spatial information. In CNN, the network weights (known as convolutional weights) are 4D tensors ($\# \text{output channels} \times \# \text{input channels} \times \# \text{input height} \times \# \text{input width}$)² with small spatial support (e.g., 3×3). CNN’s convolutional weights are shared densely across all locations in the feature maps or input image. As shown by Figure 2.3, convolutional weights are applied to inputs (via matrix multiplication) from the small windows. For a small-window input, its output is a multi-channel vector at a single “pixel” location (where the two arrows are pointing at) on the output feature maps. There are such small windows for every spatial location on the same 3D input/features maps. In MLP, there would be only a single large window that covers the entire input for applying weights.

A typical CNN architecture (Figure 2.4) consists of multiple *network stages*, where

¹tensors are a generalization of scalars, vectors, and matrices.

²‘#’ refers to ‘number of’

FIGURE 2.4: A typical CNN architecture with multiple stages and pooling layers between the stages.

each stage handles a specific feature map resolution and specific number of feature channels. The feature map resolutions are gradually decreased and the feature channel numbers are increased, going from one stage to the next. In every network stage, there are multiple repeated (the number of repetitions is a hyperparameter) basic building blocks stacked sequentially, resembling MLP's architecture introduced in Chapter 2.1.1. The only difference between MLP and CNN in terms of basic building block (right of Figure 2.1) is that CNN uses convolutional weight layer and MLP uses FC weight layer. As in MLP, non-linear activation layers (e.g., ReLU) are required in CNN for learning non-linear data representations. Between the network stages are pooling operations [32] that downsample the feature maps using some weight-free statistical aggregation methods (e.g., mean, max). Simple examples of *mean/average* and *max* pooling operations are shown in Figure 2.5. While here we show pooling operations with non-overlapping pooling windows, the windows can also be overlapping. In Chapter 6, we propose a downsampling method in CNN that downsamples at not just between network stages, but at any

FIGURE 2.5: Average and max pooling operations. The different colours correspond to different pooling windows. Average pooling computes the mean of values in each of the pooling windows, while max pooling obtains the max value.

network layers.

By virtue of such a stage-wise architectural design, CNN is able to learn hierarchical visual representations effectively. The differences among network stages in the numbers of feature channels and feature map resolutions force CNN to learn features in a hierarchical manner, where the features learned in each stage (or a stage's layer) can strongly relate to a specific level of visual abstraction [1, 54]. The lower layers in a CNN detect low-level visual patterns that normally include edges and colour blobs. The higher layers of a CNN are focused on detecting high-level features such as object parts (e.g., body parts of an animal) or even whole objects (e.g., the whole animal). To understand what CNN has learned, input images that maximally activate certain feature channels can be generated by optimizing the input pixels [1, 54]. Some examples of inputs that maximally activates certain features in different convolutional layers (conv1, conv2, conv3, ...) in a CNN are provided in Figure 2.6.

Since the majority of visual recognition problems are not sensitive to the absolute locations (translation symmetry, shift invariance) of objects or other semantic entities, weight sharing in CNNs allows detection of visual features or patterns at any

FIGURE 2.6: Input images that maximally activate some feature channels in a CNN. It can be seen that the lower layers learn to detect lower-level visual concepts, and vice versa. They are borrowed from the paper of Mahendran and Vedaldi [1].

local parts of or locations on the image and feature maps. This is a form of prior knowledge encoded in the design of CNNs. It relieves the great burden of CNNs to discover local image structure and to learn that there are regularities among the local patches. As demonstrated by Lin et al. [55], a deep MLP network without weight sharing performs significantly worse than a CNN. Commonly, weight sharing is applied to and enforced on all CNN layers except for the final classification fully-connected (FC) layer.

FIGURE 2.7: A simple RNN example showing how weight sharing is carried out for input-to-feature and feature-to-feature transformations, at different time steps.

2.1.3 Recurrent Neural Network (RNN)

Recurrent neural networks (RNN) [29, 30] have been successful in sequence modelling tasks, especially natural language processing tasks. Similar to CNN, RNN applies weight sharing but along the temporal dimension of sequential data rather than spatial dimension as in CNN. We adopt RNN in Chapter 3 to sequentially process different image sub-regions.

As illustrated in Figure 2.7, the input-to-feature weights of a RNN are shared across different (sequentially processed) input segments of the data. While, the *hidden layer*'s weights of RNNs learn a “time step-to-time step” transition mapping that effectively retains and aggregates useful contextual information from past input for use in later sequential steps. The hidden layer's transition mapping weights are also shared across time steps. This is known as *memory* in RNNs. Karpathy et al.

[56] demonstrate that qualitative visualizations can reveal the existence of long-range context dependencies modelling in RNNs. There are many existing works that improve the memory capability in RNNs, notably Long-Short Term Memory (LSTM) [57] that incorporates advanced information gates (shared across time steps) to enable sequential information to pass smoothly across many time steps, while alleviating vanishing and exploding gradient issues associated with conventional RNNs. Gated Recurrent Unit (GRU) [58] simplifies the gating mechanism of LSTM without sacrificing the long-term memory capability of LSTM. Conventionally, the weights in RNN are 2D weight matrices and have full feedforward weight connectivity (fully-connected/FC) between all input and output dimensions similar to MLP architecture.

The different input segments that share the same RNN weights correspond to different (sequence) positions or time steps of the sequential data. A typical or vanilla RNN makes use of fully-connected (FC) weights (which are 2D real-valued matrices) for its input-to-hidden and hidden-to-hidden mappings, mirroring that of MLP (Chapter 2.1.1). In contrast, our work in Chapter 3 proposes a recurrent network with fully convolutional (Chapter 2.1.2) weights, applying weight sharing across iterative steps and spatial locations. The different sequential input segments are vectors which similar to the data structure considered by MLP. The homogeneity and regularities among the sequential data segments allows for effective weight sharing. Without weight sharing, a RNN becomes an ordinary MLP that has separate weights for every time step's input, therefore requiring an explosive number of trainable weights. Compared to feedforward networks that do not have weight sharing, the weight sharing in RNNs allows them to be unconstrained by the number of sequential steps. This is beneficial to RNNs in sequence modeling tasks where the data examples have variable sequence lengths. For example, character-based sequences (words) have different character lengths. For the tasks where the different input instances of the incoming sequence data arrive at the same time, RNNs can

be trained and used in bidirectional [59, 60] manners. Bidirectional RNN apply the same RNN weights to process the sequence data in two temporal directions (the default direction and its reverse). This helps RNNs to better capture long-distant dependencies and to further strengthen weight sharing in RNNs.

Our thesis contributions (in the remaining chapters) focus on improving the learning algorithms of deep learning methods built upon the abovementioned deep learning architectures (MLP, CNN, & RNN), to tackle challenging problems in visual recognition. The above-mentioned deep neural networks are general-purpose methods which can be applied generically to most problems with little to no modifications, and their learned deep representations are mainly driven by the available training data. Instead, in this thesis, we aim to introduce novel and improved learning algorithms for deep learning to specifically address certain challenges and problems encountered in visual recognition tasks.

2.2 Saliency Detection (Pixel-level Recognition)

There are two major saliency detection [61, 62] tasks – one is to predict eye fixation-based saliency maps, and the other is to perform pixel-level salient object segmentation. Eye fixation-based saliency detection tries to predict the intensities and rough positions of eye fixations of humans on images. In the latter task – segmentation-based saliency detection, the objective is to determine the salient pixels in images. The salient (*positive*) pixels are expected to be pixels of the salient object/objects in a given input image. The differences between the two are highlighted in Figure 2.8. In Chapter 3, we focus on segmentation-based saliency detection due to the stricter pixel-level precision requirements that resemble those of other popular pixel-level visual recognition tasks (e.g., semantic segmentation [63], scene segmentation [64]).

FIGURE 2.8: The differences in terms of (ground-truth) saliency maps between eye fixation-based saliency detection and segmentation-based saliency detection. The eye fixation’s saliency map consists of blurry and somehow vague Gaussian blobs which are imprecise at local pixel level.

2.2.1 Saliency Detection Methods

Traditional saliency detection methods can be coarsely categorized into bottom-up and top-down methods. Bottom-up methods [65–71] make use of level local visual cues like color, contrast, orientation and texture. Top-down methods [72–74] are based on high-level task-specific prior knowledge. Deep learning-based saliency detection methods [2, 3, 75, 76] have also been proposed. Instead of manually defining and tuning saliency-based features, these methods can learn both low-level features and high-level semantics useful for saliency detection straight from raw images and ground-truth annotations.

These existing deep learning-based saliency detection methods [2, 3, 75, 76] do not learn to perform saliency detection end-to-end (network input: whole image, network output: final saliency map), unlike the proposed method in Chapter 3 that effectively leverage weight sharing to learn to detect saliency end-to-end from merely raw image pixels and saliency ground-truths. Wang et al. [75] learn patch-based saliency using handcrafted features, and employ another neural network to aggregate local saliency detections to detect saliency globally. Both Zhao et al. [2] and Li and Yu [3] similarly create superpixel representations of input images

using offline non-learning approaches first before training deep networks for saliency predictions.

2.2.2 Fully Convolutional Networks

In Chapter 3, our proposed method is built on a base fully convolutional network (FCN) [77] architecture (also known as convolutional-deconvolutional network in Chapter 3). FCNs are a specialized variant of CNNs designed for pixelwise prediction tasks. FCNs differ from conventional classification CNNs by not down-sampling/collapsing the spatially informative feature maps to non-spatial feature vectors. Such an architectural modification enables stronger preservation of useful spatial information and details of the input, improving the pixelwise prediction output quality.

2.2.3 Attention Mechanism in Neural Networks

Attention models [78] are a variant of neural networks that employ attention mechanisms (inspired by human visual processing system) to overcome the inabilities of traditional neural networks to selectively focus on important parts of the input. They are often used with recurrent neural networks to achieve sequential attention. Mnih et al. [79] formulate a recurrent attention model that surpasses CNN on some image classification tasks. Ba et al. [80] extend the work of Mnih et al. [79] by making the model deeper and apply it to multi-object classification task. To overcome the training difficulty of recurrent attention model, Gregor et al. [81] propose a differentiable attention mechanism and apply it to generative image generation and image classification. Jaderberg et al. [82] propose a differentiable and efficient sampling-based spatial attention mechanism, in which any spatial transformation can be used. Unlike the above works [79–81] which mostly use small attention networks for low-resolution digit classification task, the attention mechanism used

in our work in Chapter 3 is much more advanced, as it is coupled with a large convolutional-deconvolutional network for dense pixelwise saliency refinement.

2.3 Object Detection (Object-level Recognition)

In instance-level object detection [83], one or more instances belonging to some known object classes present in the input image are detected and labelled using square bounding boxes. Pascal VOC [63] and COCO [84] are the two mainstream object detection datasets in recent years. Pascal VOC [63] includes 20 object classes, while COCO [84] covers 80 object classes. These object class numbers are relatively small compared to image classification datasets (e.g., ImageNet [85]) which can reach thousands of object classes. Object detection datasets are smaller in scales because they are considerably harder to annotate by human annotators. When the object class labels have complex label relationships (e.g., overlapping, child-parent), one notable annotation issue is the annotation subjectivity – for the same object/objects, different annotators annotate differently. To overcome annotation difficulties, some existing works propose a sophisticated design of annotation system [86] and the use of pre-trained object detectors [87] to provide automated annotation assistance. In Chapter 4, we propose an effective and efficient weight transfer method to scale up the numbers of object classes handled by object detectors, in the absence of large-scale object detection datasets.

2.3.1 Object Detection Networks

Over the years, several convolutional network-based object detection frameworks and architectures have been proposed: R-CNN [88], Fast R-CNN [89], Faster R-CNN [90], R-FCN [91], SSD [92], YOLO [93, 94], FPN [95]. They can be roughly categorized into single-shot detectors [91–94] which predict detection boxes from

feature maps directly, and two-shot detectors [88–90] which first generate object proposals and then perform spatial extraction of feature maps based on the proposals for further predictions. These approaches have improved object detection from an algorithmic perspective and in a fully supervised setting. In the work in Chapter 4, we adopt Faster R-CNN [90] because its box-level *classification head* learns just a single set of classification weights, resembling image-level classification (source task) networks. This allows a smoother knowledge transfer from classification task to detection task, compared to using single-shot detection networks which learn multiple sets of classification weights for different anchor boxes.

R-CNN [88] is one of the earliest works that adapt convolutional networks for object detection. In a R-CNN, object proposals are obtained using generic object proposal methods [96, 97] and images cropped according to the object proposals are passed into a CNN (conventional architecture) for classifications. Fast R-CNN [89] extends R-CNN by cropping proposal-specific feature maps generated by a CNN instead of cropping on raw images. This allows Fast R-CNN [89] to achieve greater detection speeds and accuracy compared to R-CNN [88]. Faster R-CNN [90] is the third extension of the R-CNN series that learns a region proposal network (RPN) concurrently with classification network for a more integrated object detection solution. With RPN, Faster R-CNN [90] bypasses the need to use generic object proposal methods [96, 97], and is able to generate proposals more relevant to the training dataset/task. As a result, the number of object proposals required by Faster R-CNN [90] is significantly lower than those of R-CNN [88] and Fast R-CNN [89], thus greatly boosting the detection speed and efficiency. Faster R-CNN [90] has gained much attention and becomes one of the highest-performing object detectors in recent years. In Chapter 4, we adopt Faster R-CNN [90] as the base detection framework for our large-scale detection method, due to its stronger detection performance compared to the single-stage detectors which we review in the following paragraph.

The other detection methods – R-FCN [91], SSD [92], YOLO [93], and RetinaNet [95] are strictly single-stage object detectors that prioritize detection speed over accuracy. R-FCN [91] adopts FCNs and positive-sensitive score maps to carry out single-stage efficient detection. Whereas, SSD [92] aggregates (multi-scale) feature maps of multiple CNN layers to efficiently detect objects of multiple scales. YOLO [93] achieves single-stage object detection by dividing the image into a fixed-size grid and predicting multiple object bounding boxes per grid cell. Motivated by the ineffectiveness of single-stage detectors to handle hard examples during training, Lin et al. [98] introduce “focal” loss to exponentially grow the training losses of misclassified examples.

2.3.2 Scaling up Object Detection

Object-levels annotations are time-consuming and tedious to collect, especially when the number of classes is large. With a large number of classes, it is very challenging to obtain accurate and complete annotations due to complex overlapping meanings of classes. Thus, several approaches attempt to scale up the number of object classes handled by object detectors using image-level annotations. Transferring knowledge from image classification to object detection is an active research area tackling the lack of bounding box annotations of the target datasets and/or object classes. As shown via a visual comparison in Figure 2.9, the variety and number of semantic classes handled by an object detector can be scaled by exploiting other knowledge source such as image-level datasets. These knowledge transfer-based method for scaling up object detection can be divided into two categories: weakly-supervised and partially-supervised approaches.

Weakly-supervised methods typically rely only on a image-level classification dataset and leverage class agnostic box proposals or prior object knowledge to build object detectors. For example, Uijlings *et al.* [99] perform multiple instance learning

with knowledge transfer (source dataset with bounding boxes) to produce boxes for the target training dataset. In [100], a weakly-supervised object detector is trained on a weakly-labelled web dataset to generate pseudo ground-truths for the target detection task. [101] combines region-level semantic similarity and common-sense information learned from some external knowledge bases to train the detector with just image-level labels. More closely related to our work are weight adaptation methods [102–104] that fine-tune classification networks and learn detection-specific bias vectors to adapt the networks for detection. These adaptation-based methods assume the classification power of the network is well-preserved (e.g., using R-CNN [88]) when transferred to the detection task. This restricts them from being effectively applied to recent detection methods (e.g., Faster R-CNN [90], feature pyramid network [95]) that significantly modify the backbone network structure. Whereas, our method proposed in Chapter 4 is not restricted by such constraints. In general, classification weight-based knowledge transfer [105] can be applied to any recent detection frameworks [90, 92, 94].

On the other hand, partially supervised approaches employ weak labels, i.e. image level annotations, as well as bounding box-level annotations. E.g., YOLO-9000 [94] extend the detector’s class coverage by concurrently training on bounding box-level data and image-level data, such that the image-level data contribute only to classification loss. By decoupling the detection network into two branches (positive-sensitive & semantic-focused), R-FCN-3K [101] is able to scale detection up to 3000 classes despite being trained on limited bounding box annotations for several object classes. In contrast to these, we focus on large-scale object detection without having access to additional data (classification) sources during the training. An well-trained image classification network possesses sufficiently rich semantic knowledge about the large-scale dataset’s categories and the information is compressed in weights of its classification layers. We demonstrate in Chapter 4 that such weights can effectively be exploited to help build an object detector handling a large number

FIGURE 2.9: On the left, the object detector fails to detect the not-so-common object classes (*wheelchair*, *stethoscope*), due to the small sizes of mainstream object detection datasets. The second detector takes into account an extra knowledge source (albeit without bounding box annotations) to learn knowledge about the not-so-common classes during training, and is able to perform detection for those classes.

of categories.

2.4 Image Classification (Image-level Recognition)

Image classification [27, 28] is one of the fundamental computer vision tasks that is generally about classifying images into one or more semantic classes. It is one of the most significant exemplars of image-level visual recognition. The recent advances and happenings in image classification, particularly the large-scale ImageNet [85] challenge, have inspired and invigorated many other computer vision and visual recognition applications. In image classification, the goal is to recognize some semantic content (e.g., object classes [85], scene categories [39, 106]) represented

by the input images, with object recognition being the most common. Of all visual recognition tasks, image classification has gained the most traction because manual data annotation (by human workers) is more convenient and less costly at the image-level. Thus, image classification datasets tend to be large-scale and are more likely to cover a plethora of diverse visual and semantic concepts. Furthermore, in deep transfer learning [107], the performance gains achieved by new methods and algorithms proposed on image classification can be translated to non-trivial improvements in a wide range of other computer vision tasks. Most research efforts in image classification tend to place great emphasis on designing deeper and more effective deep network architectures [31] that better leverage the huge amounts of training data available.

ImageNet [85] is a large image classification dataset containing images belonging to 1,000 object classes, and has been the standard benchmark dataset for image classification networks. In Chapter 5 and 6, we rigorously evaluate our proposed methods – DelugeNets and SDPoint respectively on the ImageNet [85] dataset. SUN [106] and Places [39] datasets are both image classification datasets for scene recognition. Instead of having object class labels, the images in SUN [106] and Places [39] datasets are annotated with scene category labels (e.g., restaurant, living room).

2.4.1 Very Deep Networks

Developing methods for training very deep neural networks is a rather significant research topic that has received much attention over the years. It is predominantly tied with image classification tasks, largely due to their high semantic comprehensiveness (e.g., large numbers of object classes) and data volumes (i.e., big data reduces overfitting risks). Lee et al. [108] incorporate classification losses into intermediate hidden layers, allowing unimpeded supervised signals to reach the layers.

In a similar spirit as [108], GoogleNets [109] and Inception [110] models attach auxiliary classifiers to a few intermediate layers to encourage feature discriminativeness in lower network layers. DelugeNets proposed in Chapter 5, by contrast, can readily backpropagate the supervised signals to earlier layers without relying on additional losses, due to connections supporting flexible information inflows from preceding to succeeding layers.

There is another stream of works focusing on improving the information flows between layers of very deep networks, which is also the focus of our work. Highway Networks [111, 112] make use of a Long-Short-Term-Memory (LSTM [57])-inspired gating mechanism to control information flow from linear and nonlinear pathways. Through appropriately learned gating functions, information can flow in an unimpeded manner across many layers. He et al. [20] propose Residual Networks (ResNets) which compute the residual (additive) functions of the outputs of linear and nonlinear pathways, without complex gating mechanisms. ResNets have shown to tackle well the vanishing gradient and network degradation problems that occur in very deep networks. The pre-activation variants of ResNet (ResNet-v2) [12] normalize incoming activations at the beginnings of residual blocks to improve information flow and regularization.

Although ResNets [12, 20] have obtained reasonable successes, their representational capabilities are still limited by their residual connections which can be seen as rigid cross-layer network connections. Huang et al. [113] propose Densely Connected Networks (DenseNets) as a more flexible alternative to ResNets. DenseNets [113] rely on weight&computational-heavy (full) convolutional layers to learn flexible cross-layer connections, imposing very heavy computational burdens. In Chapter 5, by virtue of weight sharing, we develop a very deep network architecture called DelugeNet that is driven by flexible yet much more efficient cross-layer connections.

FIGURE 2.10: A visual comparison between normal (mixed-channel) spatial convolution and depthwise convolution.

2.4.2 Separable Convolutions

Separable convolutions have been adopted to construct efficient convolutional networks. Earlier works [114, 115] compress convolutional networks by finding low-rank approximation of convolutional layers of pre-trained networks. Network-in-network [116] employs 1×1 pointwise (cross-channel) convolutional layers to enrich representation learning in an efficient manner. 1×1 pointwise convolutions are generally coupled with other convolution variants (e.g., spatial convolutions) to achieve separable convolutions. Flattened convolutional networks [117] are equipped with one-dimensional convolutional filters of 3 dimensions (channel, horizontal, and vertical) which are processed sequentially and trained from scratch. For maximal channel-spatial separability, a conventional convolutional layer can be replaced with depthwise separable convolution (spatial depthwise convolution followed by a 1×1 pointwise convolution), as demonstrated by Xception [34]. A visual comparison between normal spatial convolution and depthwise convolution is given in Figure 2.10. In contrast to these existing works which mainly deal with channel-spatial separability, the work in Chapter 5 focuses on “cross-layer”-channel separability, to achieve efficient and flexible cross-layer connections in deep networks.

2.4.3 Cost-adjustable Inference

It is important to have CNN models that allow instantaneous adjustment of their computational costs during test-time, to resolve the variable (computational) resource availability issues in real-world applications. One representative method to achieve cost-adjustable inference is to train intermediate classifiers [108, 118, 119] which branch out of intermediate network layers. A lower inference cost can be attained by *early-exiting*, based on the intermediate classifiers’ output *confidence* [118] or *entropy* [119] threshold. The lower the threshold is, the lower the inference cost will be, and vice versa. In [118], intermediate softmax classifiers are trained (second stage) after the base network has been completely trained (first stage). The downside of [118] is that the intermediate classifier losses are not back-propagated for fine-tuning the base network weights. To make the networks more aware of intermediate classifiers, BranchyNet [119] has intermediate classifiers (each with more layers per branch than [118]) and final classifier trained jointly, using a weighted sum of classification losses. Unlike these works, our Stochastic Downsampling (SDPoint) method proposed in Chapter 6 relies on the same final classifier for different inference costs. FractalNets [120] which are CNNs designed to have many parallel subnetworks or “paths” which can be stochastically dropped for regularization during training. For cost-adjustable inference, some FractalNet’s paths can be left out. But the path-dropping regularization gives inconsistent/marginal improvements if data augmentation is used.

2.4.4 Stochastic Regularization

Our SDPoint work in Chapter 6 that stochastically downsamples feature maps is closely related to stochastic regularization methods which apply certain stochastic operations to network training for regularization. Dropout [43] drops network activations, while DropConnect [44] drops network weights. Stochastic Depth [121]

allows nonlinear residual building blocks to be dropped during training. Swapout [45] allows elementwise Bernoulli random variables, for each of the residual network function separately. These four methods are similar in the way that during inference, all stochastically dropped elements (activations, weight, residual blocks) are to be present. For any of the methods, its different stochastic instances seen during training have rather comparable forward pass computational costs, making them unfit for cost-adjustable inference. However, cost-adjustable inference can be achieved by SDPoint, for its stochastic instances have different computational costs.

2.4.5 Multiscale Training

Multiscale training of CNNs, first introduced by [122] is quite similar to cost-adjustable inference method SDPoint proposed in Chapter 6. In the training of [122], the network is trained with 224×224 and 180×180 images alternatively (one scale per epoch). The same idea has also been applied to CNN training for other tasks [94, 123]. While multiscale training downsamples the input images to different sizes, SDPoint only downsamples feature maps (at feature level). Downsampling at feature level encourages earlier network layers to learn to better preserve information, to compensate for loss of spatial information caused by stochastic downsampling later. This does not apply to multiscale training, where the input images are downsampled through interpolation operations which happen before network training takes place.

Another research direction is on exploiting weight sharing for improving CNN scale invariance at the feature level. For an input image or feature map, Kanazawa et al. [124] simultaneously downsample and upsample it to obtain multiple scales of the input (original, downsampled, upsampled), and apply the same convolutional weights to them. [124] is related to the proposed SDPoint method (Chapter 6).

The method of [124] is computationally heavy (as the same convolutional weights need to be applied to all input scales simultaneously) and cannot achieve cost-adjustable inference, whereas SDPoint actually brings down computational costs (whether in training or inference) and enables cost-adjustable inference.

Chapter 3

Recurrent Attentional Networks for Saliency Detection

3.1 Introduction

Saliency detection refers to the challenging computer vision task of identifying salient objects in imagery and segmenting the object boundaries. Despite that it has been studied for years, saliency detection still remains an unsolved research problem due to its tough goal to model high-level subjective human perceptions. Recently, saliency detection methods have received considerable amount of attention, due to the wide and growing range of applications facilitated by it. Some notable applications of saliency detection are object recognition [125], visual tracking [126], and image retrieval [127].

Traditionally, methods in saliency detection leverage low-level saliency priors such as contrast prior and center prior to model and approximate human saliency. However, such low-level priors can hardly capture high-level information about the objects and their surroundings: the traditional methods are still very far away from how saliency works in the context of human perceptions. To incorporate high-level

visual concepts into a saliency detection framework, it is natural to consider **convolutional neural networks (CNN)**. For a lot of computer vision tasks [28], CNNs have been shown to be remarkably effective. It is also the first learning algorithm to achieve human-competitive performances [128] in large-scale image classification task, which is a high-level vision task like saliency detection. Although there have been works on developing CNNs for visual saliency modelling, they either focus on predicting eye fixations [129], or applying CNNs to predict just the saliency value of visual sub-units (e.g. superpixels) independently [2]. Besides, conventional CNNs downsize feature maps over multiple convolutional and pooling layers and lose detailed information for our problem of densely segmenting salient objects.

FIGURE 3.1: An example of applying recurrent attention-based saliency refinement to an initial saliency map produced by convolutional-deconvolutional network. Compared to the initial saliency map, the refined saliency map has significantly sharper edges and preserves more object details.

Inspired by the success of **convolutional-deconvolutional network (CNN-DecNN)** in semantic segmentation [130], in this chapter, we adapt the network to detect salient objects in an end-to-end fashion. For this framework, the input is an image, and the output is its corresponding saliency map. A deconvolutional network (DecNN) is a variant of CNN that performs convolution and unpooling to produce dense pixel-precise outputs. However, CNN-DecNN works poorly for objects of multiple scales [77, 130] due to the fixed-size receptive fields. To

overcome this limitation, we propose a **recurrent attentional convolutional-deconvolutional network (RACDNN)** to refine the saliency maps generated by CNN-DeCNN. RACDNN uses spatial transformer and recurrent network units to iteratively attend to flexibly-sized image sub-regions, and refines the saliency predictions on those sub-regions. As shown in Figure 3.1, RACDNN can perform saliency detection at finer scales due to its ability to attend to smaller sub-regions. Another advantage of RACDNN is that the attended sub-regions in the previous iterations can provide contextual information for the saliency refinement of the sub-region in the current iteration. For example, in Figure 3.1, RACDNN can make use of the more visible front legs of the deer to help at refining the saliency values of the less-visible back legs.

We perform experiments on several challenging saliency detection benchmark datasets, and compare the proposed method with state-of-the-art saliency detection methods. Experimental results show the effectiveness of our proposed method.

3.2 Proposed Method

In this section, we describe our proposed saliency detection method in detail. In our method, initial saliency maps are first generated by a convolutional-deconvolutional network (CNN-DecNN) which takes entire images as input, and outputs saliency maps. The saliency maps are then refined iteratively via another CNN-DecNN operated under a recurrent attentional framework. Unlike the initial saliency map prediction which is done through single feedforward passes on the entire images, the saliency refinement is done locally on selected image sub-regions in a progressive way. At every processing iteration, the recurrent CNN-DecNN attends to an image sub-region, through the use of a spatial transformer-based attention mechanism. The attentional saliency refinement helps to alleviate the inability of CNN-DecNN to deal with multiscale saliency detection. In addition, the sequential nature of the

attention enables the network to exploit contextual patterns from past iterations to enhance the representation of the attended sub-region, hence to improve the saliency detection performance.

FIGURE 3.2: A generic convolutional-deconvolutional network for saliency detection.

3.2.1 Deconvolutional Networks for Salient Object Detection

Conventionally, CNNs downsize feature maps over multiple convolutional and pooling layers, to construct spatially compact image representations. Although these spatially compact feature maps are well-suited for whole-image classification tasks, they tend to produce very coarse outputs when being applied for dense pixelwise prediction tasks (e.g., semantic segmentation). To tackle dense prediction tasks in the multi-layered convolutional learning setting, one can append a deconvolutional network (DecNN) to a CNN as shown in [130]. In such a convolutional-deconvolutional (CNN-DecNN) framework, the CNN learns globally meaningful representations, while the DecNN upsizes feature maps and learns increasingly localized representations. Unlike the work of [130], we preserve the spatial information of CNN’s output (the input to DecNN) by using only convolutional layers. In practice, we find that preserving such spatial information works better than without preserving it. This is because the preserved spatial information provides a

good head start for DecNN to gradually introduce more spatial information to the feature maps. A generic network architecture of CNN-DecNN is shown in Figure 3.2.

A DecNN is almost identical to conventional CNNs except for a few minor differences. Firstly, in deconvolutional networks, convolution operations are often carried out in such a way that the resulting feature maps retain the same spatial sizes as those of the input feature maps. This is done by adding appropriate zero paddings beforehand. Secondly, the pooling operators adopted by CNNs are substituted with unpooling operators in DecNNs. Given input feature maps, unpooling operators work by upsizing the feature maps, contrary to what pooling operators achieve. A few variants of unpooling methods [130, 131] have been proposed previously to tackle several computer vision tasks involving spatially large and dense outputs. In this chapter, we employ the simple unpooling method demonstrated in [131], whereby each block (with spatial size 1×1) in the input feature maps is mapped to the top left corner of a blank output block with spatial size $k \times k$. This effectively increases the spatial size of the whole feature maps by a factor of k .

In the processing pipeline of CNN-DecNN for saliency detection, the CNN first transforms the input image x to a spatially compact hidden representation z , as $z = \text{CNN}(x)$. Then, z is transformed to a raw saliency map r through the DecNN, as $r = \text{DecNN}(z)$. To obtain the final saliency map $\bar{S}$ that lies within the probability range of $[0, 1]$, we perform $\bar{S} = \sigma(r)$, passing the raw saliency map r into element-wise sigmoid activation function $\sigma(\cdot)$. Given the ground-truth saliency map $\bar{G}$, the loss function of CNN-DecNN for saliency detection is the binary cross-entropy between $\bar{G}$ and $\bar{S}$. The resulting network can be trained in end-to-end fashion to perform saliency detection. Although CNN-DecNN can achieve pixel-wise labelling, it works poorly for objects of multiple scales [77, 130] due to the fixed-size receptive fields used. Furthermore, long-distance contextual information which is important for saliency detection, cannot be well captured by the

locally applied convolution filters in DecNN. To address these issues, we propose a recurrent attentional network that iteratively attends to image sub-regions (of unconstrained scale and location) for saliency refinement, which is described in the next two subsections.

FIGURE 3.3: To map the input feature maps U to output feature maps V , spatial transformer transforms output point coordinates on V to sampling point coordinates on U .

3.2.2 Attentional Inputs and Outputs with Spatial Transformer

To realize the attention mechanism for saliency refinement, we adopt the spatial transformer network proposed in [82]. Spatial transformer is a sub-differentiable sampling-based neural network which spatially transforms its input feature maps (may also be images), resulting in an output feature maps that is an attended

region of the input feature maps. Due to its differentiability, spatial transformer is relatively easier to train compared to some non-differentiable neural network-based attention mechanisms [79, 80] proposed recently.

Spatial transformer achieves spatial attention by mapping an input feature map¹ $U \in \mathbb{R}^{A \times B \times C}$ into an output feature map $V \in \mathbb{R}^{A' \times B' \times C}$. V can have spatial sizes different from U , but they must share the same number of channels C since we consider only spatial attention. Given U , spatial transformer first computes the transformation matrix τ that determines how the point coordinates in V are transformed to those in U . An example of V-to-U coordinate-wise transformation is shown in Figure 3.3. A wide range of transformation types are supported by spatial transformer. For simplicity, we restrict the transformation to a basic form of spatial attention, involving only isotropic scaling and translation. The affine transformation matrix τ with just isotropic scaling and translation is given as

$$\tau = \begin{bmatrix} a_s & 0 & a_{tx} \\ 0 & a_s & a_{ty} \\ 0 & 0 & 1 \end{bmatrix} \quad (3.1)$$

where a_s , a_{tx} , and a_{ty} are the scaling, horizontal translation, and vertical translation parameters respectively. Aligning with the recent works [79–81] in recurrent visual attention modelling, the parameters deciding where the attention takes place (in our case, τ) is produced by the localization network $f_{loc}(\cdot)$. More details on $f_{loc}(\cdot)$ will be introduced later. Subsequently, the transformation matrix τ is applied to the regular coordinates of V to obtain sampling coordinates. Based on the sampling coordinates, V is formed by sampling feature map points from U using bilinear interpolation.

¹*Feature map* is used in a generic manner here to refer to any data that can be directly represented in the form of feature map including input images and output saliency maps. In this chapter, we apply spatial transformers to input images and output saliency maps respectively.

Generally, attention mechanisms are applied only to input images. However, our saliency refinement method (see Chapter 3.2.3) via DecNN demands that the input and output ends point to the same image sub-region. To this end, we propose an inverse spatial transformer which can map refined saliency output back to the same sub-region attended at input end. Assuming that τ is the transformation matrix for the input end, the inverse spatial transformer takes the inverse of τ as the output transformation matrix τ^{-1} :

$$\tau^{-1} = \begin{bmatrix} 1/a_s & 0 & -a_{tx}/a_s \\ 0 & 1/a_s & -a_{ty}/a_s \\ 0 & 0 & 1 \end{bmatrix} \quad (3.2)$$

3.2.3 Recurrent Attentional Networks for Saliency Refinement

Recurrent neural networks (RNN) [132] are a class of neural networks developed for modelling the sequential dependencies between sub-instances of sequential data. In RNN, the hidden state h_i at time step or iteration i is computed as a non-linear function of the input and the previous iteration's hidden state h_{i-1} . Given an input x_i at iteration i , the hidden state h_i of a RNN is formulated as:

$$h_i = \phi(W_I x_i + W_R h_{i-1} + b) \quad (3.3)$$

where W_I and W_R are the learnable weights for input-to-hidden and hidden-to-hidden connections respectively, while b is a bias term, and $\phi(\cdot)$ is a nonlinear activation function. By explicitly making the current hidden state h_i dependable on the previous hidden state h_{i-1} , RNN is able to encode contextual information gained from past iterations for use in future iterations. As a result, a more powerful representation h_i can be learned.

FIGURE 3.4: Overall architecture of our Recurrent Attentional Convolutional-Deconvolutional Network (RACDNN).

In this work, we combine the recurrent computational structure of RNN with CNN-DecNN as well as the spatial transformer attention mechanism, to establish the **re-current attentional convolutional-deconvolutional networks (RACDNN)**.

As illustrated in Figure 3.4, given an initial saliency map produced by the initial CNN-DecNN, RACDNN iteratively uses spatial transformer to attend to a sub-region, and applies its CNN-DecNN to perform saliency refinement for the attended sub-region, by learning powerful context-aware features using RNN.

At every computational iteration i , RACDNN first receives an attended input x_i from the full input image x as follows:

$$x_i = ST(x, \tau_i) \quad (3.4)$$

where $ST(\cdot)$ is a spatial transformer function which produces an output image sampled from the input image, given the transformation matrix τ_i . τ_i is computed at the previous iteration $i - 1$ through the localization network $f_{loc}(\cdot)$. Then, RACDNN uses a recurrent-based CNN, CNN_r , to encode the attended input x_i into a spatially-compact hidden representation z_i . CNN_r is similar to CNN except that CNN_r is used in the recurrent setting, and all recurrent instances of CNN_r share the same network parameters. To form the recurrent hidden state h_i^1 of iteration i , the representation z_i is combined with the hidden state h_{i-1}^1 of the previous iteration:

$$z_i^1 = CNN_r(x_i) \quad (3.5)$$

$$h_i^1 = \phi(W_I^1 * z_i^1 + W_R^1 * h_{i-1}^1 + b^1) \quad (3.6)$$

where W_I^1 is the convolution filters for input-to-hidden connections, W_R^1 is the convolution filters for hidden-to-hidden connections between any two consecutive iterations, b^1 is a bias term. As in RNN, the hidden-to-hidden connections allow

contextual information gathered at previous iterations to be passed to the future iterations. Since RACDNN is attentional, the already attended sub-regions can help to guide saliency refinement for the upcoming sub-regions. This is beneficial for the task of saliency detection, as the saliency of an object is highly dependent on its surrounding regions. Different from conventional RNNs that use matrix product (fully-connected network layers) for both input-to-hidden and hidden-to-hidden connections, these connections in our method are convolution operations (convolutional layers) as in [133]. By using recurrent connections that are convolutional, we can preserve the spatial information of hidden representation h_i^1 . As mentioned in Chapter 3.2.1, preserving the spatial information of hidden representation between CNN and DecNN is favourable for DecNN's upsizing-related operations.

After obtaining h_i^1 , we can then perform saliency refinement on initial saliency maps using $DecNN_r$. The initial saliency maps are generated by the global CNN-DecNN in single forward passes. Instead of replacing the values of initial saliency map with the output of RACDNN at each iteration, the initial saliency map r_0 is refined cumulatively for N number of iterations. At iteration i , the saliency map r_i is refined as

$$r_i = r_{i-1} + ST(DecNN_r(h_{i-1}^1), \tau_i^{-1}) \quad (3.7)$$

Before being added to r_i , the saliency output of $DecNN_r$ is spatially transformed back to the attended sub-region using inverse spatial transformer (ST). For the unattended regions, the saliency refinement values are set as zero and thus those regions do not affect r_i . After N number of iterations, as in Chapter 3.2.1, sigmoid activation function $\sigma(\cdot)$ is applied to r_N , resulting in the final saliency map $\bar{S}_r$.

Besides saliency refinement outputs, at every iteration, RACDNN should generate τ to determine which sub-region to attend to in the next iteration. A simple way

to achieve that is by simply treating h_i^1 as input to a fully-connected network-based regressor. However, to model the sequential dependencies between attended locations, such a simplistic approach is insufficient. This is because h_i^1 should focus mainly on modelling contextual dependencies for saliency refinement, not multiple kinds of dependency. To better model locational dependencies, we propose to add another recurrent layer to RACDNN. The hidden state of the second recurrent layer at iteration i is denoted by h_i^2 and it is formulated as

$$h_i^2 = \phi(W_I^2 h_i^1 + W_R^2 h_{i-1}^2 + b^2) \quad (3.8)$$

where the weights W_I^2, W_R^2 and bias b^2 are semantically the same as their counterparts in the first recurrent layer in Equation (5). The input of the second recurrent layer is the output of the first recurrent layer, making the RACDNN a stacked recurrent network. Considering the nature of the regression task, we use only fully-connected layers for both recurrent input and hidden connections in the second recurrent layer. Finally, given h_i^2 , a $f_{loc}(\cdot)$ can be used to regress the transformation matrix for the next iteration $i + 1$:

$$\tau_{i+1} = f_{loc}(h_i^2) = \phi(W_{loc^2} \phi(W_{loc^1} h_i^2)) \quad (3.9)$$

W_{loc^1} and W_{loc^2} are respectively the weight matrices of the first and second layers of the two-layered fully-connected network $f_{loc}(\cdot)$ used in our work.

In RACDNN, the hidden representations (h_0^1, h_0^2) at the 0-th iteration are provided by a CNN (sharing the same architectural properties as CNN_r) which accepts the whole image region as input. Observing the full image region at the 0-th iteration helps RACDNN to better decide which sub-regions to attend subsequently.

Similar to the CNN-DecNN used for saliency detection, the loss function of RAD-CNN is the binary cross-entropy between the final saliency output $\bar{S}_r$ and the

ground-truth saliency map $\bar{G}$. Since every component in RADCNN is differentiable, errors can be backpropagated to all network layers and parameters of RADCNN, making it trainable with any gradient-based optimization methods (e.g., gradient descent). W_I^1 , W_R^1 , b^1 , W_I^2 , W_R^2 , b^2 , W_{loc^1} , W_{loc^2} , and the network weights in CNN_r and $DecNN_r$ are learnable parameters in RADCNN.

3.3 Implementation Details

For initial saliency detection, we use a CNN-DecNN independent of the CNN-DecNN used in the saliency refinement stage. The CNN part is initialized from the weights of VGG-CNN-S [134], a relatively powerful CNN model pre-trained on ImageNet dataset. VGG-CNN-S consists of 5 convolutional layers and 3 fully-connected layers. We discard the fully-connected layers of VGG-CNN-S and retain only its convolutional and pooling layers for network initialization. The CNN accepts 224×224 RGB images as inputs, and it outputs a 7×7 feature maps with 256 feature channels. The DecNN part of the initial CNN-DecNN is a network with 3 convolutional layers (5×5 kernel size, 1×1 stride, 2×2 zero paddings), and there is an unpooling layer before each convolutional layer. To increase the representational capability of the DecNN without adding too many weight parameters, we append a layer convolution layer with 1×1 convolution kernel, to each DecNN convolutional layer. At the end of the initial CNN-DecNN, the DecNN outputs a 56×56 saliency map. The output size of 56×56 achieves a good balance between computational complexity and saliency pixels details. For performance evaluation, the 56×56 saliency map is resized to the input image's original size. The initial CNN-DecNN is trained with Adam [135] in default learning settings.

As mentioned previously, the CNN_r and $DecNN_r$ used in RADCNN are trained and executed independently of those in the initial CNN-DecNN. On the other hand, $DecNN_r$ is initialized using the pre-trained weights of DecNN of the initial

CNN-DecNN. In the recurrent layers of RACDNN, rectified linear unit (ReLU) is employed as the non-linear activation $\phi(\cdot)$. The feature maps of the hidden state h_i^1 (the first recurrent layer of RACDNN) is of size 7×7 and has 256 feature channels. For the second recurrent layer’s hidden state h_i^2 , the feature representation is a 512-dimensional vector. The weight parameters W_{loc^1} and W_{loc^2} of $f_{loc}(\cdot)$ are 512×256 and 256×3 matrices respectively. The number of recurrent iterations of RACDNN (inclusive of the 0-th iteration) is set to 9 for all saliency detection experiments. RACDNN is trained using RMSProp [136] with an initial learning rate of 0.0001. The learning rate is reduced by an order of magnitude whenever validation performance stops improving. During training, gradients are hard-clipped to be within the range of $[-5, 5]$ as a way to mitigate the gradient explosion problem which occurs when training recurrent-based networks. To speed up training and improve training convergence, we apply Batch Normalization [137] to all weight layers (except for recurrent hidden-to-hidden connections) in both the initial CNN-DecNN and RADCNN.

Most of the saliency detection methods employ object segmentation techniques which can output image segments with consistent saliency values within each segment. Furthermore, the edges of the output segments are sharp. To achieve similar effects, we apply a mean shift-based segmentation method [138, 139] to the outputs of RACDNN as a post-processing step.

3.4 Saliency Training Datasets

Learning-based methods require a big amount of training samples to generalize to new examples well. However, most of the saliency detection datasets are too small. It is not possible to train the deep models well if the experimental evaluations are done in such a way that each dataset is split into training, testing and validation sets

in proportions. Here, we follow the dataset procedure in one recent deep learning-based saliency detection work [2]. We train the deep models (initial CNN-DecNN and RADCNN) in our proposed method on saliency datasets different from the datasets used for experimental evaluations. The training datasets we use are: DUT-OMRON [140], NJU2000 [141], RGBD Salient Object Detection dataset [142], and ImageNet segmentation dataset [143]. The data samples in these datasets reach a total number of 12,430, which is roughly the size of the dataset (with 10,000 samples) used in [2]. We randomly split the combined datasets into 10,565 training samples and 1865 validation samples. Although the training set is considered large in saliency detection context, it is still small for deep learning methods, and may cause overfitting. Thus, we apply data augmentation in the form of cropping, translation, and colour jittering on the training samples.

3.5 Experiments

FIGURE 3.5: Precision-recall curves, with average precisions.

3.5.1 Datasets and Evaluation Metrics

We evaluate our proposed on a number of challenging saliency detection datasets: **MSRA10K** [70] is by far the largest publicly available saliency detection dataset, containing 10,000 annotated saliency images. **THUR15K** [144] has 6,232 images which belong to five object classes of “butterfly”, “coffee mug”, “dog jump”, “giraffe”, and “plane”. It is challenging because some of its images do not contain any salient object. **HKUIS** [3] is a recently released saliency detection dataset with 4,447 annotated images. **ECSSD** [10] is a challenging saliency detection dataset with many semantically meaningful but structurally complex images. It contains 1,000 images. **SED2** [145] is a small saliency dataset having only 100 images. For each image, there are two salient objects.

We evaluate the proposed method based on precision-recall curves, which is the most commonly used evaluation metric for saliency detection. The saliency output is thresholded at integer values within the range of $[0, 255]$. At each threshold value, the binarized saliency output is compared to the binary ground-truth mask to obtain a pair of precision-recall values. Another popular evaluation metric for saliency detection is F-measure, which is a combination of precision and recall values. Following the recent saliency detection benchmark paper [62], we use a weighted F-measure F_β that favours precision more than recall: $\frac{(1+\beta_2)\text{Precision} \times \text{Recall}}{\beta_2 \text{Precision} + \text{Recall}}$, where β_2 is set as 0.3. The reported F_β is the maximum F-measure computed from all precision-recall pairs, which is a good summary of detection performance according to [62].

Even though F-measure is the most commonly used evaluation metric for saliency detection, it is not comprehensive enough as it does not consider true negative saliency labelling. To have a more comprehensive experimental evaluation, we consider another evaluation metric known as Mean Absolute Error (MAE) adopted by [62]. MAE is given by: $\frac{1}{\mathcal{W} \times \mathcal{H}} \sum_{n=1}^{\mathcal{W}} \sum_{m=1}^{\mathcal{H}} |\bar{S}(n, m) - \bar{G}(n, m)|$, where $\mathcal{W}$ and $\mathcal{H}$

are width and height of saliency map; $\bar{S}$ is the real-valued saliency map output normalized to the range of $[0, 1]$, and $\bar{G}$ is the saliency ground-truth. Saliency map binarization is not needed in MAE as it measures the mean of absolute differences between ground-truth saliency pixels and given saliency pixels.

3.5.2 Comparison with Baseline Methods

To highlight the advantages of recurrent attention mechanism in the proposed network RACDNN, we use CNN-DecNN as one of the baseline methods in our experiments. Compared to the proposed method, the baseline CNN-DecNN has no recurrent attention mechanism to perform iterative saliency refinement. The other baseline method is a CNN-DecNN paired with a non-recurrent attentional convolutional-deconvolutional network (NACDNN) in place of RACDNN. NACDNN is a RACDNN variant whose layers h^1 and h^2 are made non-recurrent. By removing the recurrent connections, NACDNN cannot learn context-aware features useful for saliency refinement despite having attention mechanism. At each computational iteration, NACDNN works almost like a CNN-DecNN except that it has a localization network $f_{loc}(\cdot)$ that accepts CNN's output as input and outputs spatial transformation matrix.

To compare the proposed method with baseline methods, we use F-measure and MAE as evaluation metrics. The F-measure scores and Mean Square Errors (MAEs) for comparisons with the baselines are shown in Table 3.1. On all the five datasets and two evaluation metrics, the proposed method achieves better results than both the baseline methods. This shows that the RACDNN can help to improve the saliency map outputs of CNN-DecNN, using a recurrent attention mechanism to alleviate the scale issues of CNN-DecNN, and to learn region-based contextual dependencies not easily modelled by mere convolutional and deconvolutional network operations. The second baseline method NRACDNN that has attention mechanism

	MSRA10K			THUR15K			HKUIS			ECSSD			SED2		
in %	F-M	MAE		F-M	MAE		F-M	MAE		F-M	MAE		F-M	MAE	
CNN-DecNN	87.91	7.03		69.28	10.42		82.48	8.10		85.72	8.72		82.79	9.29	
+ NRACDNN	88.62	6.85		70.39	10.46		83.74	7.88		86.65	8.43		83.99	9.30	
+ RACDNN	89.98	6.02		71.12	9.04		85.57	7.03		87.81	8.12		85.35	9.29	

TABLE 3.1: F-measure scores (**F-M**) and Mean Absolute Errors (**MAE**) (compared with baseline methods).

	MSRA10K			THUR15K			HKUIS			ECSSD			SED2		
in %	F-M	MAE		F-M	MAE		F-M	MAE		F-M	MAE		F-M	MAE	
RRWR [4]	84.92	12.36		59.99	17.77		71.28	17.18		74.70	18.51		77.98	16.08	
BSCA [5]	85.88	12.52		60.94	18.24		71.89	17.48		76.03	18.32		78.25	15.79	
DRFI [6]	88.07	11.82		67.02	15.03		77.31	13.45		78.70	16.59		83.86	12.70	
RBD [7]	85.59	10.80		59.62	15.04		72.29	14.24		71.79	17.33		82.96	12.97	
DSR [8]	83.46	12.07		61.07	14.19		73.47	14.22		73.69	17.29		78.90	14.01	
MC [9]	84.76	14.51		60.96	18.38		72.34	18.40		74.18	20.37		77.10	17.96	
HS [10]	84.49	14.86		58.54	21.78		70.76	21.50		73.04	22.83		80.37	11.18	
Ours	89.98	6.02		71.12	9.04		85.57	7.03		87.81	8.12		85.35	9.29	

TABLE 3.2: F-measure scores (**F-M**) and Mean Absolute Errors (**MAE**) (compared with state-of-the-art methods).

performs better than the non-attentional first baseline. However, due to the lack of recurrent connections, NRACDNN is inferior to RACDNN because it does not exploit contextual information from past iterations for saliency refinement.

FIGURE 3.6: Qualitative saliency results of some evaluated images. From the leftmost column: input image, saliency ground-truth, the saliency output maps of our proposed method (CNN-DecNN + RACDNN) with mean-shift post-processing, MCDL [2], MDF [3], RRWR [4], BSCA [5], DRFI [6], RBD [7], DSR [8], MC [9], and HS [10].

3.5.3 Comparison with State-of-the-art Methods

In addition to the baseline methods, we compare the proposed method “CNN-DecNN + RACDNN” with several state-of-the-art saliency detection methods: RRWR [4], BSCA [5], DRFI [6], RBD [7], DSR [8], MC [9], and HS [10]. DRFI, RBD, DSR, MC, and HS are the top-performing methods evaluated in [62], while RRWR and BSCA are two very recent saliency detection works. To obtain the results for these methods, we run the original codes provided by the authors with recommended parameter settings. The precision-recall curves are given in Figure 3.5. We compute the curves based on the saliency maps generated by the proposed method. In overall, the proposed method “CNN-DecNN + RACDNN” performs better than the evaluated state-of-the-art methods. Especially in datasets with complex scenes (ECSSD & HKUIS), the performance gains of the proposed method over the state-of-the-art methods are more noticeable.

We also compare the proposed method “CNN-DecNN + RACDNN” with the state-of-the-art methods in terms of F-measure scores and Mean Square Errors (MAEs) (Table 3.2). In these evaluation metrics, its performance gains over the other methods are very significant. For the HKUIS and ECSSD dataset, the F-measure improvements of the proposed method over the next top-performing method DRFI are more than 5%. The proposed method also pushes down the MAEs on these challenging datasets by a large margin.

Besides quantitative results, we show some qualitative results in Figure 3.6. The proposed method “CNN-DecNN + RACDNN” can better detect multiple intermingled salient objects, as shown in the second image with a dog and a rabbit. Our method is the only one that can detect both objects well. The success of our method on this image is attributed to the attention mechanism that allows it to attend to different object regions for local refinement, making it is less likely to be negatively affected by distant noises and other objects. However, the proposed method tends to fail to detect salient objects which are mostly made up of background-like colours and textures (e.g., sky: third image, soil: fourth image). We conjecture that this is caused by the local context utilized by RACDNN that is incapable of helping it to distinguish the confusing background-like salient objects from the non-salient backgrounds. E.g., in the third image, the local patches of the white (left) horse are likely to be confused as background patches.

To further evaluate the proposed method “CNN-DecNN + RACDNN”, we compare it with two recent deep learning-based saliency detection methods (MCDL [2] and MDF [3]) on HKUIS, ECSSD, and SED2 datasets. We use the trained models provided by the authors. The F-measure scores and MAEs are given in Table 3.3, showing that the proposed method is comparable to both MCDL and MDF in terms of F-measure, but outperforming them in terms of MAEs.

	HKUIS		ECSSD		SED2	
in %	F-M	MAE	F-M	MAE	F-M	MAE
MCDL [2]	80.85	9.13	83.74	10.20	81.37	11.45
MDF [3]	86.01*	12.93*	83.06	10.81	86.23	11.18
Ours	85.57	7.03	87.81	8.12	85.35	9.29

TABLE 3.3: Comparison with deep learning-based methods. *MDF is trained on a subset of HKUIS, and then evaluated on the remaining HKUIS samples.

3.5.4 Comparison with Recent Recurrent Network-based Methods

We compare our method with very recent saliency detection methods [146–148] that utilize recurrent neural networks in Table 3.4. These methods came out much later after the publication of our work. RFCN [146] uses a fully convolutional network architecture with recurrent feedback to refine saliency outputs. DGRL [147] treats each residual block in ResNet [20] as a recurrent unit, and the residual blocks refine features for saliency detection in a recurrent fashion. PAGRN [148] aggregates features from the main CNN using spatial and channel attention mechanisms, while multi-path recurrent connections are established between different network stages of the main CNN to enhance global context information utilization. While these recent methods are recurrent network-based, in general they avoid processing local patches in the recurrent steps. Instead, they perform recurrent refinement globally with some sort of soft attention mechanisms. The comparison table is given in. Note that RFCN [146] employs pretrained VGG-16 [149] as backbone network, PAGRN [148] employs VGG-19 [149], and [147] employs ResNet-50 [20]. These backbone networks are a lot more sophisticated and deeper (16,19,50 layers respectively) than the one (VGG-S [134]) used in our method which has only 8 layers. Despite using a much weaker backbone network, we find that our proposed method still performs fairly close to and not too far off from the recent methods.

	HKUIS		ECSSD		THUR15K	
in %	F-M	MAE	F-M	MAE	F-M	MAE
RFCN [146]	85.65	5.47	87.13	6.68	-	-
DGRL [147]	88.2	3.7	90.3	4.5	71.6	7.7
PAGRN [148]	88.6	4.8	89.1	6.4	72.9	7.0
Ours	85.57	7.03	87.81	8.12	71.12	9.04

TABLE 3.4: Comparison with very recent recurrent network-based methods.

3.6 Concluding Remarks

In this chapter, we introduce a novel method of using recurrent attention and convolutional-deconvolutional network to tackle the saliency detection problem. The proposed method has been shown to be very effective experimentally. The generality of the proposed method implies that it can be adapted for other vision tasks that require pixel-wise prediction [77, 131].

Chapter 4

Scalable Object Detection with Auto-Encoder Weight Transfer Network

4.1 Introduction

State-of-the-art object detectors [90, 150] are typically trained on a dataset with a large number of box annotations. Large-scale datasets such as COCO [84], Pascal VOC [151] and OpenImages [152] provide a great number of bounding boxes, but the number of annotated object classes is often very limited. The reason is that scaling the number of bounding-boxes can be semi-automated, e.g. [152], while increasing the number of classes requires a significant amount of human labour. On the other hand, image-level labels such as those available in classification datasets are much easier to collect as they do not require costly bounding box annotations. As a consequence, several works investigated the training of object detectors in a weakly-supervised regime, using only image-level labels. These methods leverage a wide variety of classes available in classification datasets or image tags found

in social networks [153] but neglect the spatial information available in object detection datasets.

In contrast, partial supervised methods [105] employ both types of annotations. While existing methods [94, 103, 104, 154] that transfer knowledge from a classification network to a detection network with partial supervision achieve higher accuracy than weakly-supervised methods [154–156], they incur a significant computational overhead during training and testing. The overhead comes either from joint training of the two networks [94, 154], or from performing forward passes of the classification network during testing [103, 104]. Furthermore, joint-training methods often require storage-intensive, large-scale classification datasets to be present while training the detection network.

To overcome these limitations, we propose a novel approach to transfer discriminative semantic knowledge from classification to detection with a non-linear weight-transfer network (WTN) [105]. Given a set of common classes annotated for both tasks, we learn a function, the weight-transfer network, that maps weights at the fully-connected layer of the classification network to those of the object detection network. Once trained, WTN is used to extend the number of classes recognized by the object detector via transferring weights of unseen classes from the classification network. This strategy is advantageous because it only adds little computational and memory overheads to training and no overhead to inference at all.

Compared to the vanilla WTN model [105], we introduce two key components to our weight-transfer network. First, we insert normalization layers to account for the different amplitude of the classification weights. Most importantly, we replace the multilayer perceptron with an autoencoder. The latent space of the autoencoder corresponds to the classification weights of the object detector and therefore is trained with object-level supervision. The reconstruction loss between the input and output of the autoencoder is essential to retain information in weights of all

the classes, while the detection network’s classification loss facilitates the learning of discriminative embedding of the class weights.

Extensive experimentation on the Open Images [152] dataset demonstrates that the proposed method significantly outperforms existing partially-supervised detection approaches on challenging detection tasks involving novel object classes. Moreover, due to the auxiliary regularization effect brought by the reconstruction loss of autoencoder-WTN⁺, our proposed method even recovers the detection performance loss of existing WTN on seen classes.

In summary, contributions of this chapter are as follows:

- We address the under-fitting issue of WTN by introducing input and feature normalization schemes. The resulting model WTN⁺ achieves improved detection performance over the vanilla WTN.
- We propose our main model, autoencoder-WTN⁺, that can better preserve semantic knowledge of all object classes, while learning to generate discriminative classification weights for the detection network.
- We verify the effectiveness of our method with extensive evaluations using large-scale datasets with millions of images and several hundreds of object classes.

4.2 Weight Transfer Network

4.2.1 Preliminaries

We consider the setting where we have a classification network CLN that handles object classes $\check{C}$, and a detection network DEN that handles object classes $\check{D}$. The

TABLE 4.1: The *train* and *test* phases of object detector DEN with Autoencoder-WTN⁺ (AE-WTN⁺). **Train phase:** Before training DEN, we take the final classification FC layer’s weights of a pre-trained CLN and discard its remaining layers. AE-WTN⁺ is then trained simultaneously with DEN to transform weights from CLN to DEN through the shared classes $\tilde{S}$. AE-WTN⁺ has a reconstruction loss to reconstruct the weights of CLN’s classes $\tilde{C}$. The “other detection classes” ($\tilde{D} \setminus \tilde{S}$) are trained normally as conventional classification loss. **Test phase:** During testing, CLN’s weights of both the novel classes $\tilde{N}$ and shared classes $\tilde{S}$ can be adapted for use in DEN through the trained AE-WTN⁺. Given that, DEN is able to perform detection for classes $\tilde{N}$ in addition to the other training classes.

number of classes handled by CLN is much greater than the number of classes handled by DEN, i.e. $|\check{C}| \gg |\check{D}|$. The goal of our approach is to expand the number of classes handled by DEN through partial supervision, where we transfer weight knowledge from CLN (source task) to DEN (target task). We make use of the final fully-connected (FC) layer weights of the CLN that has been pre-trained on a large scale image classification dataset. The final FC layer weights can be seen as a form of *semantic embeddings* comprising rich knowledge about the object classes and the complex class relationships. Furthermore, pre-trained large-scale image classification networks are very accessible and many are shared publicly.

Classification knowledge from CLN is transferred to DEN using a weight transfer network (WTN) through the object classes shared ($\check{S}$) between the two tasks: $\check{S} = \check{C} \cap \check{D}$. WTN is a neural network that works as a class-generic function $T()$ used to transform per-class classification weight vectors $W_{\check{C}} = [w_{\check{C}}^1, w_{\check{C}}^2, \dots, w_{\check{C}}^{|\check{C}|}]$ from CLN to DEN's classification weights $W_{\check{D}} = [w_{\check{D}}^1, w_{\check{D}}^2, \dots, w_{\check{D}}^{|\check{D}|}]$ as follows: $W_{\check{D}} = T(W_{\check{C}})$.

WTN is trained simultaneously with DEN on detection dataset with classes $\check{D}$. The gradients of WTN's network parameters come from DEN's box-level classification loss ℓ_{cls} . Before training WTN and DEN, we 'freeze' $W_{\check{C}}$ (taken from pre-trained CLN). While $\check{S}$ rely on WTN, for the DEN's classes which are not part of $\check{S}$ (i.e., $\check{D} \setminus \check{S}$), we train their weights as in conventional detection network. To obtain DEN's classification score predictions, we simply perform matrix multiplication between DEN's box-level visual features and WTN's predicted weights, just like how it works for conventional classification weights. Conventionally, WTN is based on a two-layer multi-layer perceptron (MLP) architecture.

Due to the class-genericness of WTN, WTN is able to carry out effective *inductive learning* [157]. In other words, despite that only classes $\check{S}$ are seen by WTN and DEN during training, during testing WTN (and the DEN model that incorporates WTN) can work reasonably well with classes $\check{N}$ of CLN, where $\check{N} = \check{C} \setminus \check{S}$.

FIGURE 4.1: Comparison between network architectures of WTN and WTN⁺. The green rectangles correspond to layers with learnable parameters, whereas the grey rectangles are layers without learnable parameters.

4.2.2 Normalizations

Commonly, in large-scale classification datasets, the class distribution is unbalanced and this has strong implications in how the classification weights of CLN are trained. E.g., in one large-scale CLN, we discover that the ‘highest-norm’ class has a weight vector *norm* that is 28 times that of the ‘lowest-norm’ class. Besides,

a *class-generic non-linear* WTN naturally cannot adapt and learn as well as (conventional) *class-specific linear* classification weights, for loss minimization. These pose challenges to the training and optimization of WTN. Empirically, we find that a DEN equipped with existing WTN performs inferiorly to a conventional DEN on a detection task for classes $\check{D}$.

We introduce a new variant of WTN – WTN⁺ to counter underperformance and ease optimization, drawing from the recent findings and advancements in normalization techniques [11, 137] for neural networks. The model architectural differences between WTN and WTN⁺ are illustrated in Fig. 4.1. Standard normalization is applied to the input weights $W_{\check{C}}$ to enable different input channels to contribute comparably to the prediction of $W_{\check{D}}$, in order to curb the overdominance/underdominance of certain classes. Let v_j denote the weights of j -th feature/channel of $W_{\check{C}}$, we normalize v_j by: $\frac{v_j - \mu(v_j)}{\sigma(v_j)}$, where $\mu(\cdot)$ and $\sigma(\cdot)$ are the *mean* and *standard deviation* functions respectively.

Group Normalization (GN) [11] layer, known for its strong optimization benefits, is added to normalize intermediate features to encourage good gradient flows for easier network optimization. As demonstrated in Fig. 4.2, a GN layer first splits the incoming feature channels into multiple channel groups before performing standard normalization independently for each channel group. Since WTN is trained from scratch with randomly-initialized weights, the channels (before training) are considered unordered and thus they can be split into groups arbitrarily. This is a common practice used effectively in depthwise convolution [34] and grouped convolution [13] to split channels into groups in an arbitrary fashion. Typically, the channel groups have the same number of channels, and the main hyperparameter of GN is the number of groups to split the incoming channels into. Let $Z \in \mathbb{R}^{B \times K}$ denote the incoming feature activations, where B is the data batch size and K is the total number of feature channels. Using a specified number of groups M (GN’s main hyperparameter), the channel splitting of GN is performed

as follows: $Z = [Z_1 \in \mathbb{R}^{B \times \frac{K}{M}}, \dots, Z_j \in \mathbb{R}^{B \times \frac{K}{M}}, \dots]$. For the j -th group's feature activations Z_j , we compute its per-batch-example statistics – *mean* $\mu(Z_j) \in \mathbb{R}^{B \times 1}$ and *standard deviation* $\sigma(Z_j) \in \mathbb{R}^{B \times 1}$, to perform same-group normalization independently¹ for each batch example. The normalized Z_j is denoted as $\hat{Z}_j \in \mathbb{R}^{B \times \frac{K}{M}}$, where $\hat{Z}_j = \frac{Z_j - \mu(Z_j)}{\sigma(Z_j)}$. The complete output $\hat{Z}$ is obtained by concatenating the normalized groups: $\hat{Z} = [\hat{Z}_{v_j} \in \mathbb{R}^{B \times \frac{K}{M}}]$. Finally, to learn more flexibly, GN typically applies a per-channel linear transformation layer with learnable multiplicative and additive parameters ($\gamma \in \mathbb{R}^{1 \times K}$ and $\beta \in \mathbb{R}^{1 \times K}$) to $\hat{Z}$ as follows: $\gamma\hat{Z} + \beta$.

FIGURE 4.2: A Group Normalization (GN) [11] layer splits the input channels into a number of channel groups (each group is represented by a distinct colour). The top GN splits the channels into 5 groups, while the bottom one splits them into 3 groups. For any GN configuration shown here, all the channel groups share the same number of channels, although this is not a strict requirement.

These small yet crucial modifications of incorporating input-level and feature-level normalizations are the key to training highly effective WTN.

¹The per-batch-example normalization scheme makes GN insensitive to the training batch size, in contrast to the more widely-used Batch Normalization [137] that relies on batch statistics.

4.3 Autoencoder Weight Transfer Network

During the training of DEN and WTN, WTN (including WTN⁺) is merely concerned with classes $\check{S}$. Only $\check{S}$ contribute to the training gradients and losses. The novel object classes $\check{N}$ are unknown to and unconsidered by WTN. The lack of knowledge of the entire class population of $\check{C}$ limits WTN's capability to effectively model the *good* classification space originally attained by the pre-trained CLN for handling large number of classes. We hypothesize that by letting WTN have a narrow view of the class population, its modelling capability (relating to $\check{N}$ specifically) is severely underexploited and this compromises the performance of WTN on classes $\check{N}$.

To this end, we introduce Autoencoder-WTN⁺ (AE-WTN⁺) – novel WTN variant that attempts to preserve knowledge on all of classes $\check{C}$ contained in pre-trained $W_{\check{C}}$, while learning a discriminative WTN function to achieve good detection performance. AE-WTN⁺ is an autoencoder with both encoder and decoder networks. AE-WTN⁺ is built on top of WTN⁺. The encoder network shares the same architecture as WTN⁺'s, while the decoder network (with separate network layers/-parameters) is the mirrored version of the encoder. Following existing WTN, the encoder network works as a function $T()$ to predict $W_{\check{D}}$ given $W_{\check{C}}$ as input. During training, gradients are propagated from DEN's loss to the encoder network. The *train* and *test* phases of DEN + AE-WTN⁺ are illustrated in Fig. 4.1. The network architecture of AE-WTN⁺ is shown in Fig. 4.3.

AE-WTN⁺ is trained with an additional autoencoder-based training loss – *reconstruction loss* [158] that forces the decoder network to predict (or reconstruct) the original inputs, from the output activations of the encoder network. Let $T()$ denote the encoder network and $G()$ denote the decoder network, the reconstruction is predicted as follows: $\hat{w}_{\check{C}} = G(T(w_{\check{C}}))$; $\forall w_{\check{C}} \in W_{\check{C}}$. Here, we adopt smooth L1 loss [89] as the reconstruction loss to minimize the difference between the predicted

reconstructions and the original inputs ($W_{\check{C}}$):

$$\ell_{\text{rc}} = \begin{cases} 0.5(\hat{w}_{\check{C}} - w_{\check{C}})^2, & \text{if } |\hat{w}_{\check{C}} - w_{\check{C}}| < 1 \\ |\hat{w}_{\check{C}} - w_{\check{C}}| - 0.5, & \text{otherwise} \end{cases} \quad (4.1)$$

Note that we apply reconstruction loss to all CLN classes $\check{C}$ (i.e., $\check{S} \cap \check{N}$), rather than just shared classes $\check{S}$. On the other hand, the detection loss (box-level classification) only cares about classes $\check{S}$ and “other” detection classes. With such formulation, we perform multi-task training based on the following mixture of training losses (excluding Region Proposal Network’s [90]: $\ell_{\text{cls}} + \ell_{\text{reg}} + \alpha \ell_{\text{rc}}$, where ℓ_{reg} is box regression loss and α is the loss scaling hyperparameter.

Reconstruction loss penalizes intermediate network activations which do poorly to reconstruct the original weights W_{CLN} . Since AE-WTN⁺’s output $W_{\check{D}}$ (weights for DEN) is a form of intermediate network activations, they are affected by the reconstruction loss and are expected to retain original class information greatly for reconstruction purpose. In contrast, existing WTN (or even WTN⁺) is solely driven by DEN’s classification loss (which may not be optimal for model generalization) and is not compelled to retain more of potentially useful class information. Reconstruction-based information preservation has been shown to help neural networks achieve better local optima [159, 160] in supervised learning. By complementing CLN’s classification loss with a reconstruction loss, AE-WTN⁺ is able to learn a non-linear mapping that achieves a good balance between class/class discriminability and class information retainment. We find that this has a regularization effect on AE-WTN⁺ and it helps improve generalization performance on the fully-annotated object classes ($\check{D} \cap \check{S}$) seen during training. This observation is aligned with the findings of [159, 160] that supervised learning can be improved with autoencoders. While we apply reconstruction loss to all classes including $\check{N}$ (which do not have supervised annotations), [159, 160] apply the loss to only input

examples with supervised annotations. Our work also resembles semi-supervised learning where reconstruction loss (autoencoder) [161, 162] is used as an auxiliary loss to exploit unlabeled data (in this work, class $\check{N}$ are unlabeled) to improve model performance and generalization.

FIGURE 4.3: Autoencoder-WTN⁺ (AE-WTN⁺) that retains rich class information of all classes in the latent space W_{DEN} and effectively exploits novel class knowledge during training.

During the training of existing WTN, $W_{\check{C},\check{N}}$ the weights of novel classes $\check{N}$, contained in $W_{\check{C}}$, is not utilized. And, classes $\check{N}$ do not contribute to the training. Deep neural networks are generally known to eliminate task-irrelevant information of the inputs through training [163, 164]. Thus, it is likely that WTN learns to “dismiss” some class information about classes $\check{N}$ that is unimportant to classes $\check{S}$ but is useful for the detection of classes $\check{N}$. The reconstruction loss of AE-WTN⁺ addresses such a shortcoming of existing WTN by explicitly involving the novel object classes $\check{N}$. The rich class information in $W_{\check{C},\check{N}}$ (which is potentially beneficial to AE-WTN⁺’s test-performance on classes $\check{N}$) is preserved in the intermediate network activations of AE-WTN⁺.

With AE-WTN⁺, DEN is encouraged to learn to output RoI features that can interact reasonably well with a more class-diverse *weight space* (output space of AE-WTN⁺) contributed and established by both the shared ($\check{S}$) and novel ($\check{N}$) object classes. This is in contrast to the *weight space* learned by WTN⁺/WTN which is solely optimized for just the shared ($\check{S}$) object classes. Although the most optimal RoI features for classes $\check{N}$ cannot be learned due to the lack of training box annotations, using AE-WTN⁺ to improve the feature generality and class diversity enables DEN’s RoI features to better generalize to the $\check{N}$ classes, compared to non-AE WTN variants. This resembles the zero-shot recognition method that jointly learns image and text embeddings to enable the visual model (CNN) to generalize to unseen concepts [165].

4.4 Experiments

4.4.1 Implementation Details

4.4.1.1 Training and evaluation datasets for seen classes $\check{D}$

We use the official training and validation dataset (referred to as **OI-500**) [152] from Open Images V4 Challenge which contains 500 object classes for training and evaluating DEN on classes $\check{D}$. The object classes in Open Images dataset are hierarchically organized and many classes are not mutually exclusive. Open Images’ official evaluation metric [152], a custom version of “Average Precision (AP) @ 0.5 IoU threshold” or AP_{50} is used for evaluation on the validation set provided. We use the same Open Images training set to train baseline Faster RCNN and our WTN-based models for fair comparisons on novel classes $\check{N}$.

4.4.1.2 Evaluation datasets for novel classes $\check{N}$

To evaluate DEN’s performance on novel classes $\check{N}$, we employ two evaluation datasets. The first evaluation set (**OI-57**) is a subset of Open Images V4 *complete*/non-challenge dataset containing 57 *novel* object classes and 31,061 images. The second evaluation set (**VG-200**) is set as a subset of Visual Genome [166] dataset containing 24,690 images spanning 200 high-frequency object classes which are *novel* to DEN. We adopt the same AP_{50} metric for **OI-57**. Since many object instances in Visual Genome dataset are not annotated at all, we follow the practice of [167] by using *Average Recall*/ $AR_{50}@100$ detections per image to gauge the detection performance of DEN on this evaluation set.

4.4.1.3 Classification network (CLN)

Prior to training WTN and DEN, a pre-trained large-scale CLN model has to be acquired. We use a publicly available² ResNet-101 pre-trained on Open Images v2 [152] with 5000 object classes. It is trained with multi-label (sigmoid) classification loss given the multi-label nature of the dataset. Training resolution is 299×299 . The model is trained asynchronously with 50 GPU workers and batch size 32 for 620K training steps. Incoming features to the final classification layer is 2048-dimensional.

4.4.1.4 Detection Network (DEN)

The DEN architecture in this chapter is a Faster R-CNN [90] with a backbone integrating ResNet-50 [20] and Feature Pyramid Network (FPN) [95]. ResNet-50 backbone is pre-trained on ImageNet-1k [85] dataset, and its BN parameters are frozen during training of DEN. The box-level head (for box classification and regression) is a 2-layer multi-layer perceptron (MLP) with a 2048-dimensional feature and output channels. DEN is trained with mini batches of 8 images (2 images/GPU) for a total of 180K iterations. We optimize the network using SGD with momentum of 0.9 and initial learning rate of 2×10^{-2} . The network is regularized with weight decay of 1×10^{-4} . We stick closely to the original training loss functions of Faster R-CNN except for the classification loss which we replace with sigmoid binary cross-entropy, taking Open Images' class hierarchy and multilabel nature into account. The training class labels are expanded [168] according to the hierarchy tree [152] given.

²github.com/openimages/dataset/blob/master/tools/classify_oidv2.py

	OI-500 (Seen)	OI-57 (Novel)	VG-200 (Novel)
Method	AP_{50}	AP_{50}	AR_{50}
Faster R-CNN [90]	59.55	-	-
Faster R-CNN (NN)	-	28.09	49.39
LSDA [102]	59.44	25.89	51.14
LSDA (Visual Transfer) [103]	59.44	26.43	53.03
ZSD [167] with CLN weights	47.37	34.63	38.04
ZSD [167] with fastText [170]	58.39	29.51	35.09
WTN [105]	52.87	34.94	41.91
WTN ⁺			
► default model	58.82	39.28	65.60
▷ 5× weight decay	58.46	40.79	65.87
▷ activity regularizer [171]	55.86	33.47	36.26
▷ Dropout [43]	57.14	40.09	65.52
▷ reduced capacity	58.80	37.81	63.16
AE-WTN ⁺	59.59	41.07	66.75

TABLE 4.2: Comparison with weight transfer-related methods on evaluation datasets – OI-500 (seen classes), OI-57 (novel classes), and VG-200 (novel classes).

4.4.1.5 WTN

By default, WTN variants have input/feature/output channels of 2048. For Group Normalization (GN) layer in WTN⁺ and AE-WTN⁺, we follow the same “number of groups”/#groups hyperparameter, which is set to 32 as found to be a good choice by [11]. WTN networks are trained from scratch simultaneously with DEN using AdamW [169] using default hyperparameters and weight decay of 1×10^{-4} . For AE-WTN⁺, we find that the performance of AE-WTN⁺ is relatively insensitive to the loss scaling hyperparameter α (except for extremely small or large values), and we set α to 20 throughout the experiments.

4.4.2 Comparison with related methods

To validate the effectiveness of our proposed AE-WTN⁺ model, we experimentally compare it with existing weight transfer-related methods described in the following. Note that all these methods use the same Faster R-CNN detection framework and a ResNet-50 backbone.

- **Faster R-CNN**: Vanilla Faster R-CNN [90] performs fully supervised learning on seen classes. In contrast to WTN, vanilla Faster R-CNN learns conventional classification weights which are both linear³ and class-specific⁴. To detect novel classes, we employ the nearest-neighbour approach (*NN*), taking the detections of nearest seen classes.
- **LSDA** [102]: LSDA adapts CLN’s weights for detection task by learning additive class-specific biases. To make predictions for a novel class during test-time, the biases of nearest classes are averaged and added to CLN’s weight vector. The visual similarity transfer variant [103] is also included.
- **ZSD** [167]: ZSD performs zero-shot detection through pre-trained word embeddings. In a joint visual-word embedding setting, the detector learns to output visual embeddings in the words’ embedding space. Here, two kinds of embeddings are considered – CLN’s weights and fastText [170].
- **WTN** [105]: This corresponds to the standard (existing) WTN model that makes use of neither normalization techniques nor reconstruction loss.
- **WTN⁺ variants**: Since the reconstruction loss of AE-WTN⁺ can be seen as a regularizer, we compare it with several WTN⁺ variants regularized with increased weight decay ($5\times$) [47], activity regularizer (0.01) [171], Dropout

³These weights are considered linear because they are not transformed using non-linear WTN.

⁴WTN’s network parameters/weights are class-generic and applicable to all classes in CLN, while vanilla Faster R-CNN learns classification weights which are absolutely class-specific.

(0.3) [43] on intermediate activations, and reduced network capacity (halving the number of channels in hidden layer).

The results are given in Table 4.2. We use ResNet-50 as the backbone for the vanilla Faster RCNN detector, and its AP_{50} on OI-500 is 59.55% which is mildly worse than the 60.0% achieved by the state-of-the-art SE-ResNeXt-101 detector [168]. Overall, WTN methods outperform the non-WTN methods by large margins on the novel classes (OI-57 and VG-200), due to the powerful weight transfer function learned by WTN that can generalize to many classes. Among the WTN methods, AE-WTN⁺ achieves the best results.

WTN and WTN⁺ suffer from the weakened detection performance on OI-500 (seen classes $\check{D}$) compared with the vanilla Faster R-CNN detector that it is built upon. In other words, switching to WTN from *conventional classification weights* decreases performance on the seen classes. This phenomenon has been observed by prior works [102, 172] attempting to scale object detection with weak or partial supervision. By integrating autoencoder into WTN (AE-WTN⁺), the seen-class detection performance can be recovered. It is extremely challenging to train conventional WTN from scratch. The reconstruction loss (which is more easily optimized than detection loss) encourages AE-WTN⁺ to output weights highly representative of original CLN weights, thus providing a good initialization to attain better local optima. Similar to prior works that find reduced supervised training loss with autoencoder [159, 160], we find that the box-level classification training loss ℓ_{cls} on seen classes attained by AE-WTN⁺ (0.5572) is lower than that of WTN⁺ (0.5754).

Moreover, the reconstruction loss explicitly involves novel classes $\check{N}$ during training and forces AE-WTN⁺ to preserve rich class information of the novel classes in the latent and output spaces. It also encourages visual features learned by DEN to be more “generic” (less specific to classes $\check{D}$ in the detection dataset) in order to accommodate to the many classes represented by AE-WTN⁺. Therefore,

the detector equipped with AE-WTN⁺ shows improved (absolute) performances of 1.8% and 1.1% over WTN⁺ on OI-57 and VG-200 respectively. Compared to other existing regularization techniques applied to WTN⁺, AE-WTN⁺ performs better consistently across all datasets. This provides confirmation that the advantages of the reconstruction loss cannot be simply replicated by other regularizers that do not leverage the rich class information contained in CLN’s weights.

Qualitative results. We provide in Fig. 4.4 some qualitative results obtained by our proposed AE-WTN⁺ detector on test images of Open Images [152] and Visual Genome [166] datasets. Only the classes with the highest scores are shown, and novel classes compete with seen classes for the same bounding box. Remarkably, the detector can detect quite a number of novel classes at greater confidence than seen classes, despite not seeing them during training.

4.4.3 Analysis

4.4.3.1 Local neighbourhood preservation

To better understand the implications of the reconstruction loss on local neighbourhood preservation of AE-WTN⁺, we compute the overlapping count between nearest neighbours (NN) obtained by CLN’s weights and the output weights of the WTN model of interest (AE-WTN⁺, WTN⁺, or WTN), varying the number of neighbours (a standard hyperparameter of *nearest neighbour* approach) for all methods. This study is performed on 20 randomly sampled object classes and the counts are averaged across those classes. The sampled classes and nearest neighbours are among the 5,000 classes of CLN. The findings are presented in Fig. 4.5. E.g., at 100 neighbours, AE-WTN⁺’s output weights and CLN’s weights have an average of 48.25 overlapping neighbours, while WTN⁺ and WTN have 38.0 and 31.95 overlapping neighbours respectively. As shown, AE-WTN⁺ consistently reaches greater numbers of overlapping neighbours (with CLN’s neighbours) than

FIGURE 4.4: Qualitative results of the proposed AE-WTN⁺. The detection boxes with blue labels are **seen** object classes $\tilde{D}$, while the boxes with red labels are **novel** classes $\tilde{N}$. Our method can handle a variety of novel classes and concepts, while performing well on seen classes.

WTN⁺ and WTN do, indicating that AE-WTN⁺ can better preserve the local neighbourhood relationships of classes. Noticeably, the gap widens as the number of nearest neighbours increases.

FIGURE 4.5: The overlapping count (vertical axis) between CLN’s nearest neighbours and the nearest neighbours obtained by the WTN model of interest (either AE-WTN⁺, WTN⁺, or WTN), given varying numbers of nearest neighbours (horizontal axis).

4.4.3.2 Normalizations in WTN⁺

As explained in Sec. 4.2.2, we introduce normalization techniques to WTN (thus forming WTN⁺) to tackle the under-fitting issues suffered by the existing WTN model. Here, we perform ablation study to understand how the performance changes when ablating the normalization techniques. The results of the ablation study are provided in Table 4.3. We find that using vanilla WTN without any normalization gives the worst performance and it is worse by large margins than the other “partially/fully-normalized” ablation models. Adding input (standard) normalization somehow improves the detection performances, while using feature normalization (Group Normalization [11]) alone also provides some performance gains. Combining the two (WTN⁺), we obtain the best results for both seen and

novel classes. Furthermore, we observe worse training losses with standard non-normalized WTN compared with WTN^+ , implying that model under-fitting is the inherent cause of WTN’s under-performance. This reaffirms the importance of normalization techniques which are known to be crucial for neural network optimization.

$\text{WTN} \rightarrow \text{WTN}^+$		OI-500 (Seen)	OI-57 (Novel)	VG-200 (Novel)
Input Norm.	Group Norm.	AP_{50}	AP_{50}	AR_{50}
✗	✗	52.87	34.94	41.91
✓	✗	57.60	37.27	54.19
✗	✓	54.60	35.84	58.55
✓	✓	58.82	39.28	65.60

TABLE 4.3: Ablation study on WTN^+ architecture.

4.4.3.3 Choice of feature normalization

GN [11] is chosen over the commonly-used BatchNorm (BN) [137] for feature normalization in WTN^+ because we find that BN is less robust towards novel-class inputs which do not have detection annotations/loss. The poor robustness of BN against unseen inputs has also been identified by [173]. Particularly, we find that the post-ReLU activation (L2) norms of WTN^+ with BN have an unusually large *variance* for novel classes. It is $70\times$ (or $\frac{7.117}{0.104}$) that of shared classes, despite allowing BN to normalize over all classes in training. Such unstable activations are not encountered by the detection network during training. This causes WTN^+ ’s predicted weights for novel classes to interact poorly with image-region features at test time, resulting in unreliable class-score predictions. Table 4.4 shows the L2 norm *means* & *variances* of using GN and BN respectively.

	mean		variance	
	shared cls.	novel cls.	shared cls.	novel cls.
GN [11]	1.838	1.784	0.091	0.093
BN [137]	1.379	2.627	0.104	7.117

TABLE 4.4: Means and variances of post-ReLU activation norms.

4.4.3.4 Computational efficiency

Computational efficiency is a major concern in the training and/or deployment of object detectors, especially for large-scale detectors. In Table 4.5, we show the per-iteration training time (in milliseconds/*ms*) and single-GPU memory usage of training with different models. Overall, the WTN models add very little computational costs on top of Faster R-CNN’s. It is noteworthy that, at just a small increase in computational cost over the previous method, AE-WTN⁺ is able to improve the detection performance on both seen and novel classes. During testing, the weights can be transformed offline with WTN/WTN⁺/AE-WTN⁺ to reach vanilla Faster R-CNN’s efficiency.

	Faster R-CNN	WTN	WTN⁺	AE-WTN⁺
Time (ms)	365	371	379	401
Mem. (GB)	4.11	4.15	4.19	4.26

TABLE 4.5: Training time and memory usage.

4.5 Concluding Remarks

Training large-scale object detectors are extremely resource-demanding. In this chapter we introduce an efficient WTN approach to scale up object detection and propose novel methods to push the limits of WTN through normalization techniques and autoencoder-based reconstruction loss. The reconstruction loss adopted by AE-WTN⁺ effectively improves its capability to retain and exploit the semantically-rich class information (of all classes) learned by the pre-trained CLN.

This leads to improved regularization of DEN and better detection performances on both seen and novel classes.

Chapter 5

DelugeNets: Deep Networks with Efficient and Flexible Cross-Layer Connections

5.1 Introduction

Deep learning methods [174, 175], particularly convolutional neural networks (CNN) [176] have revolutionized the field of computer vision. CNNs are integral components of many recent computer vision techniques which spread across a diverse range of vision application areas [28]. Hence, developing more sophisticated CNNs has been a prime research focus. Over the years, many variants of CNN architectures have been proposed. Some works focus on improving the activation functions [33, 128], and some focus on increasing the heterogeneity of convolutional filters within the same layers [109, 110]. Lately, the idea of improving CNNs by greatly deepening them has gained much traction, following the immense successes of Residual Networks (ResNets) [12, 20] in image classification.

ResNets make use of residual connections to support relatively unobstructed information flows (shortcut) between layers. Each succeeding layer receives the sum of all its preceding layers’¹ outputs as input. Compared to traditional non-residual deep networks, outputs of preceding layers in ResNets can reach succeeding layers with minimal obstructions, even if the preceding layer and succeeding layer are separated by a very long layer-distance. However, the cross-layer connections between preceding and succeeding layers of ResNet are fixed and not “selective”, and therefore the succeeding layers are not able to prioritize or deprioritize output channels of certain preceding layers. Instead, the outputs of preceding layers are lumped together via simplistic additive operation, making it very tough for succeeding layers to perform layer-wise information selection. The inflexibility of residual connections also hinders the ability of ResNets to learn cross-layer interactions and correlations.

Densely connected networks (DenseNets) [113] aim to overcome this drawback of ResNets, by having convolutional layers to consider an extra dimension – the depth/cross-layer dimension, in addition to the spatial and feature channel dimensions used in regular convolutions. In DenseNets, the input feature maps to succeeding layers are concatenations of preceding layers’ outputs, rather than simple summations. Hence, when applying convolution operations on the concatenated feature maps, the convolutional filters have to learn spatial, cross-channel, and cross-layer correlations altogether, entailing heavy amounts of parameters (filter width $\times$ filter height $\times$ # input channels $\times$ # output channels $\times$ # preceding layers) and computations. DenseNet-BC [113] was recently introduced as a more efficient variant of DenseNet, in which the filters consider just cross-channel and cross-layer correlations. Despite that, considering that DenseNets’ composite layers receive inputs from several dozens of preceding layers, the computation and parameter requirements are still high.

¹The unit layer in ResNet, ResNet-like, and DenseNet models refers to a *composite layer* formed by several basic layers. See Chapter 5.2.1.

FIGURE 5.1: Deluge Network components: (a) a composite layer, (b) a block transition component, and (c) a block. **Red**-coloured arrows indicate 1×1 cross-layer depthwise convolutions.

To counter excessive computational complexity and parameter growth, DenseNet models are specifically configured to have much lower output *width* (between 12 and 48 output channels) at each layer, compared to typical image classification CNNs. However, it is crucial to have considerable network width as contended by [14], and decreasing output width too much is harmful to network’s representational power. Furthermore, by visualizing DenseNet’s weight norms, Huang et al. [113] showed that the features of preceding *composite layers* get reused directly by the succeeding *composite layers* in a rather infrequent manner. Yet, these “diminished” features have to be processed by relatively expensive convolution operations in DenseNets.

Thus, in this chapter, we propose a new class of CNNs called Deluge Networks (DelugeNets) which enable **flexible cross-layer connections** yet have **regular output width** in each composite layer. As a result of using regular output width, the information inflows from preceding layers to succeeding layers in DelugeNets are massive, in contrast to the lesser information inflows in DenseNets. DelugeNets are inspired by separable convolutions [34, 114, 116, 117]. The efficiency of convolutions can be improved by separating the combined dimensions involved, resulting in separable convolutions. DelugeNets are designed such that the depth/cross-layer dimension is processed independently of the rest (channel and spatial dimensions), using a novel variant of convolutional layer called *cross-layer depthwise convolutional layer* (see Figure 5.2) as described in Chapter 5.2.2. Cross-layer depthwise convolutional layers handle only **cross-layer** interactions and correlations, without getting burdened by other dimensions. They facilitate cross-layer connections in DelugeNets in a very efficient and effective manner. Experiments show the superior performances of DelugeNets in terms of classification accuracy, parameter efficiency, and more remarkably computational complexity.

5.2 Deluge Networks

Similar to existing CNNs (ResNet [12, 20], VGGNet [149], and AlexNet [27]), DelugeNets gradually decrease spatial sizes and increase feature channels of feature maps from bottom to top layers, with a linear classification layer attached to the end. The layers operating on the same feature map dimensions can be grouped to form a *block*. In DelugeNets, the input to a particular layer comes from **all of its preceding layers** of the same *block*. There is no information directly flowing from other *blocks*. Within any *block*, the cross-layer information flows through connections established by the cross-layer depthwise convolutions (see Chapter 5.2.2). For transition to the next *block* as described in Chapter 5.2.3, we perform cross-layer depthwise convolution followed by strided spatial convolution to obtain feature map of matching dimensions. The structure of *block* in DelugeNets is illustrated in Figure 5.1(c), with individual layers separated by vertical dashed lines.

5.2.1 Composite Layer

In CNNs, a *layer* often refers to a *composite layer* of several basic layers such as Rectified Linear Unit (ReLU), Convolutional (Conv), Batch Normalization (BN) layers. Inspired by [12], we use the bottleneck-kind of *composite layer* BN-ReLU-Conv-BN-ReLU-Conv-BN-ReLU-Conv in DelugeNets, as illustrated in Figure 5.1(a). This kind of *composite layer* is designed to improve parameter efficiency in deep networks, by employing 1×1 spatial convolutional layers at the beginning to reduce channel dimension, and at the end to increase channel dimension. In the ResNet models proposed by [12], the base channel dimensions are increased by 4 times. We however only increase them by 2 times in this chapter, for the reason that we can allocate more computational and parameter budgets to train deeper DelugeNets.

Such a *composite layer* has also shown to work very well for very deep neural networks which combine information from multiple sources, such as ResNets and the proposed DelugeNets. The primary reason that it works well is that combined multi-source information is normalized via BN layers before it is passed into the upcoming weight (convolutional) layers. This reduces internal covariate shift and regularizes the model more effectively [12] than just passing unnormalized multi-source information to the weight layers.

FIGURE 5.2: Cross-layer Depthwise Convolution. The columns correspond to feature channel indices, and the rows correspond to preceding *composite layer* indices.

5.2.2 Cross-layer Depthwise Convolutional Layers

To facilitate efficient and flexible cross-layer information inflows, in this chapter, we develop a cross-layer depthwise convolution method. A cross-layer depthwise convolutional layer concatenates the channels of feature map outputs of many layers, and then applies “channel,spatial”-independent filters to the concatenated channels. Equipped with such filters, DelugeNets are able to process the depth/layer dimension independently of the rest (channel and spatial dimensions), as mentioned in Chapter 5.1. Figure 5.2 gives a graphical illustration of cross-layer depthwise convolution operation.

Cross-layer depthwise convolutional layers facilitate the inflows of information from preceding *composite layers* to succeeding *composite layers*. Suppose that ℓ denotes the layer of an arbitrary *composite layer*, and $h_{\ell-i}^c \in \mathbb{R}$ denotes the c -th channel of the preceding $(\ell - i)$ -th *composite layer*’s output. And, there are N number of preceding *composite layers*, as well as one preceding block transition output $\mathbf{h}_0$. The c -th channel of the input $\mathbf{x}_\ell$ to *composite layer* ℓ is:

$$x_\ell^c = \sum_{i=1}^{N+1} w_{\ell-i}^c h_{\ell-i}^c + b_\ell^c \quad (5.1)$$

where $w_{\ell-i}^c \in \mathbb{R}$ and $b_\ell^c \in \mathbb{R}$ are the filter weights and bias respectively, for each channel. We streamline the equation by not having spatial location-related notations, and the weights and biases are assumed to be shared across all 1×1 spatial locations (spatially independent) in the input feature maps as mentioned earlier.

The parameter cost of adding cross-layer depthwise convolutional layer to any existing network architecture is relatively low compared to other network parameters. For an arbitrary *composite layer* in the network, the number of additional parameters is merely $N \times M + 1$, where M is the number of feature channels. Experimentally, we find that these extra parameters on average make up about 3% of entire

model parameters. In terms of computational complexity (measured in floating-point operation numbers or FLOPs), cross-layer depthwise convolutions on average cost 3% more, compared to baseline models without such convolutions. DenseNets [113], on the other hand, require heavy amounts of computations and parameters to connect to preceding layers, through cross-layer output concatenations followed by 3×3 or 1×1 spatial convolutions.

5.2.2.1 Advantages

Cross-layer depthwise convolutional layers are beneficial because they encourage features generated by a preceding *composite layers* to be taken as inputs for many times by the succeeding layers (feature reuse [177]). We conjecture that this leads to parameter efficiency because it alleviates the need to redundantly learn filters which generate the same features in succeeding layers, in case those features are needed again later. Furthermore, in conventional ReLU-based convolutional networks, features that get turned off by ReLU activation functions (at the beginnings of blocks) cannot be recovered by other network parts or layers. In DelugeNets, via the use of cross-layer depthwise convolutional layers, output of a preceding *composite layer* can be transformed differently for each succeeding *composite layer* to serve as input. Consequently, input features that get turned off at the beginning of certain succeeding *composite layers* may be active in others.

In CNNs, the filter weights are shared across many spatial locations in the feature maps. The weight sharing mechanism acts as an effective regularizer. Similar to the CNN’s weight sharing mechanism, the same features in DelugeNets’ preceding *composite layers* are shared by the succeeding *composite layers*. As a result, the weights of composite layers in DelugeNets become more regularized. Based on such consideration, we allocate more model parameters to the spatially smaller network blocks, by setting the number of *composite layer* in succeeding block to be larger

Model	#Params	Depth	GigaFLOPs	CIFAR-10	CIFAR-100
Highway Network [111]	-	-	-	7.60	32.24
FractalNet [120]	38.6M	20	-	4.59	22.85
ResNet [20]	1.7M	164	0.51	5.93	25.16
ResNet [20]	16.1M	1001	2.36	7.61	27.82
ResNet with ELU [178]	-	110	0.51	5.62	26.55
ResNet with Identity Mappings [12]	1.7M	164	0.51	5.46	24.33
ResNet with Identity Mappings [12]	19.4M	1001	2.36	4.62	22.71
ResNet with Swapout [45]	7.4M	32	2.16	4.76	22.72
ResNet with Stochastic Depth [121]	1.7M	110	0.51	5.23	24.98
ResNet with Stochastic Depth [121]	10.2M	1202	2.84	4.91	-
Wide-ResNet (04×width) [14]	8.7M	40	2.60	4.53	21.18
Wide-ResNet (08×width) [14]	11.0M	16	3.10	4.27	20.43
Wide-ResNet (10×width) [14]	36.5M	28	10.49	4.00	19.25
DenseNet ($k = 12$) [113]	7.0M	100	3.65	4.10	20.20
DenseNet ($k = 24$) [113]	27.2M	100	14.56	3.74	19.25
DenseNet-BC ($k = 24$) [113]	15.3M	250	10.09	3.62	17.60
DenseNet-BC ($k = 40$) [113]	25.6M	190	18.67	3.46	17.18
DelugeNet-146	6.7M	146	1.43	3.98	19.72
DelugeNet-218	10.0M	218	2.13	3.88	19.31
Wide-DelugeNet-146	20.2M	146	4.31	3.76	19.02

TABLE 5.1: CIFAR-10 and CIFAR-100 test errors (percentage) of existing models and DelugeNets.

than preceding block. The motivation behind this is to achieve **lower computational complexity** (since smaller feature maps are computationally cheaper to process), relying more on feature sharing through cross-layer feature reuse, and less on weight-sharing across spatial locations, for regularization. Such a parameter allocation strategy differs greatly from that of ResNets. In ResNets, many layers and parameters are allocated for blocks with spatially large feature maps to regularize filters better, and less for blocks with spatially small feature maps to reduce overfitting (see Chapter 5.3.2).

Besides encouraging feature reuse, cross-layer depthwise convolutional layers are advantageous from the perspective of gradient flow. The gradient flows in DelugeNets are regulated by multiplicative interactions with the filter weights in cross-layer depthwise convolutional layers, such that the *composite layers* all receive unique backpropagated gradient signals even if they come from the same block. This is not true for ResNet models, in which the *composite layers* within the same block receive identical backpropagated gradient signals, due to simple addition (residual) operation.

5.2.3 Block Transition

Different network blocks operate on feature maps of different spatial and channel dimensions. For block transition, there is a need to transform the feature map to match the spatial and channel dimensions of next block. In ResNet-like models, block transition can be done with either 1×1 strided convolution, or strided average pooling with channel padding. These block transition designs aim to preserve the information from previous block by having only minimal transformation as well as dismissing any non-linear activation function. Such block transition designs are suboptimal for DelugeNets because they allow direct information flow from just the last *composite layer* of the previous block, and they conceivably hinder the information flows from other *composite layers*.

To this end, we propose a new block transition component which has a cross-layer depthwise convolutional layer followed by 3×3 spatial convolutional layer. The cross-layer depthwise convolutional layer allows direct information inflow from all *composite layers* from the previous block, therefore summarizing the outputs of all *composite layers* of previous block. Then, the 3×3 strided spatial convolutional layer (see Figure 5.1(b)) transforms the summarized feature map to have matching spatial and channel dimensions. 3×3 strided convolutional layer is chosen over 1×1 strided convolutional layer as the latter wastes the features it receives, for many of the feature map’s spatial locations, while the former does not. Similar to the block transition designs in ResNets, we do not add non-linear activation functions after the spatial convolutional layer.

5.3 Experiments

To rigorously validate the effectiveness of DelugeNets, we evaluate DelugeNets on 3 image classification datasets with varied degrees of challengingness and scales:

CIFAR-10 [179], CIFAR-100 [179], ImageNet [85].

5.3.1 CIFAR-10 and CIFAR-100

5.3.1.1 Datasets

CIFAR-10 and CIFAR-100 are 2 subsets of the Tiny Images dataset [180] annotated to serve as image classification datasets. There are 50,000 training images and 10,000 testing images for each of the 2 CIFAR datasets. For pre-processing, we subtract channel-wise means from the images, and divide them by channel-wise standard deviations. During training, data augmentation is carried out moderately as in [14, 113], with horizontal flipping and random crops taken from images padded by 4 pixels on each side. For all CIFAR-based models, the training is carried out using a single GPU.

5.3.1.2 Implementation

A total of 3 different DelugeNet models are implemented and evaluated on CIFAR datasets. Similar to [12, 14], all the 3 DelugeNet models have 3 *blocks* – the first block works on spatially 32×32 feature maps, followed by 16×16 and 8×8 feature maps for second and third blocks respectively. They vary only in terms of numbers of *composite layers* and feature channel dimensions for the 3 blocks. To minimize manual tuning of architectural hyperparameters, we design different DelugeNet models based on a simple principle that follows the parameter allocation scheme mentioned in Chapter 5.2.2 – the second block has 2 times the **numbers of *composite layers*** and **feature channel dimension** (width) of the first block, the third block has 2 times of the second's, and so on:

- **DelugeNet-146** has base feature channel dimensions (widths) of $\{32, 64, 128\}$, and *composite layer* counts of $\{8, 16, 24\}$, for its 3 blocks (in sequential ordering) respectively.
- **DelugeNet-218** shares the same base widths as DelugeNet-146, but it comes with larger *composite layer* counts of $\{12, 24, 36\}$ which make it a much deeper model.
- **Wide-DelugeNet-146** is a $1.75\times$ wider variant of DelugeNet-146, having base widths of $\{56, 112, 224\}$, while the *composite layer* counts remain the same.

The proposed models (DelugeNet-146, DelugeNet-218, and Wide-DelugeNet-146) for the 2 CIFAR datasets differ only in the numbers of output labels (10 and 100). To train the models, we run Stochastic Gradient Descent (SGD) over a total of 300 training epochs, with Nestorov Momentum [181] and weight decay [47] rate of $1e-4$. As most of the existing models we compare with in this chapter do not use any dropout-like regularization, we do not use any either, for fairer comparison. The starting learning rate is 0.1, and it is decayed by factor of 0.1 at epoch 150 and 225. We set the mini-batch size as 64. All DelugeNet model parameters are initialized using He’s initialization method [128], and they are trained using the same settings. The training settings are in fact identical to the settings employed in [113] to train DenseNets.

5.3.1.3 Results

The top-1 classification errors achieved by the DelugeNets and existing models on both CIFAR datasets are presented in Table 5.1. The results of existing models are obtained directly from their respective papers. As shown in Table 5.1, DelugeNets can benefit from “deepening” (DelugeNet-218) and “widening” (Wide-DelugeNet-146).

5.3.1.4 Parameter efficiency

DelugeNets are able to perform well despite requiring much lower numbers of learnable parameters/weights compared to existing models. The parameter efficiencies of Delugenets are second only to DenseNet-BCs [113] which aggressively compress and reduce feature channels to save trainable parameters. Notably, DelugeNet-218 performs competitively to Wide-ResNet ($10\times\text{width}$), on both CIFAR-10 and CIFAR-100 datasets, with merely 10M weights compared to 36.5M weights in Wide-ResNet. Besides, Wide-DelugeNet-146 achieves CIFAR classification errors comparable to those of DenseNet ($k = 24$), with 7M fewer parameters.

5.3.1.5 Computational complexity

In addition to parameter numbers, we report the model complexities of DelugeNets and several other comparable models (Wide-ResNets, DenseNets, and DenseNet-BCs), in terms of floating-point operation (FLOP in giga prefix unit, Giga/GFLOP) numbers. We find that in overall DelugeNets have significantly fewer model complexities than other models. Surprisingly, DelugeNet-218 requires just 1/5 of the FLOPs required by Wide-ResNet ($10\times\text{width}$) to achieve similar classification errors. Although DelugeNets cannot exactly match or outperform DenseNet-BC, they (DelugeNets) can achieve appreciable classification errors which are rather close to those of DenseNet-BCs, at fractions of DenseNet-BCs' complexity costs. The lower model complexities of DelugeNets are attributed to the parameter allocation scheme (Chapter 5.2.2) as well as the capability of cross-layer depthwise convolutions at alleviating overfitting, even when spatially smaller network blocks have more parameters/layers than their spatially larger counterparts.

Model	#Params	GFLOPs	Error	PDiff
ResNet-like baseline				
- 1x1 conv shortcut	6.15M	1.33	21.14	-
- 3x3 conv shortcut	6.50M	1.39	20.84	+0.30
DelugeNet-146	6.69M	1.43	19.72	+1.12
ResNet-like baseline				
- 1x1 conv shortcut	9.26M	1.98	20.78	-
- 3x3 conv shortcut	9.55M	2.05	20.31	+0.47
DelugeNet-218	10.00M	2.13	19.31	+1.00
ResNet-like baseline				
- 1x1 conv shortcut	18.76M	4.05	19.79	-
- 3x3 conv shortcut	19.82M	4.25	19.98	-0.19
Wide-DelugeNet-146	20.19M	4.31	19.02	+0.77

TABLE 5.2: Comparison with ResNet-like baselines on CIFAR-100 test errors. The fourth column reports the performance differences (**PDiff**) between baselines and DelugeNets.

5.3.1.6 Ablation study

In this work, we propose cross-layer depthwise convolutional layer and a new kind of block transition design with 3×3 spatial convolution, which differentiate DelugeNets from existing networks. To better understand the contributions of these components, we construct ResNet-like baselines on the 3 proposed DelugeNet models. There are 2 types of baselines for each of the DelugeNet models: The **first baseline** has all of its cross-layer depthwise convolutions replaced by residual connections. Alternatively, the residual connections can be seen as cross-layer depthwise convolutional layers, whose weights are fixed as ones as pointed in Chapter 2.4.1. The **second baseline** is similar to the first one except that it is equipped with 3×3 convolutional shortcuts for block transitions, similar to our proposed block transition design. Other than those mentioned, all aspects of the baselines and their corresponding DelugeNets are the same, including training settings. We evaluate the baselines on CIFAR-100. The results are shown in Table 5.2.

Block transitions with 3×3 convolutional shortcuts can mildly improve the performances of DelugeNet-146’s and DelugeNet-218’s baselines. However, there is slight overfitting ($19.79\% \rightarrow 19.98\%$) from adding 3×3 convolutional shortcuts to Wide-DelugeNet-146’s baseline. The overfitting issue is greatly eased by having cross-layer depthwise convolutions in Wide-DelugeNet-146. As evidenced by the significant performance improvements (about 1%) of DelugeNets over the baselines, the biggest contributor is cross-layer depthwise convolutional layer. Yet, the parameter costs incurred by adding these layers are very marginal. The smallest DelugeNet model, DelugeNet-146 (19.72%) with just 6.69M parameters and complexity of 1.43 GFLOPs, suprisingly outperforms the biggest ResNet-like baseline (19.79%) with 18.76M parameters and complexity of 4.05 GFLOPs. Furthermore, just tiny increases in complexity (about 3%) are needed by cross-layer depthwise convolutions to achieve considerable performance gains. These findings reaffirm the advantages of the proposed cross-layer depthwise convolutional layer for deep networks.

FIGURE 5.3: Layer-wise L2-norms of cross-layer depthwise convolution weights. Each of the 3 columns corresponds to a different block transition stage in the networks. Vertical axes indicate the indices of the preceding *composite layers*, and horizontal axes indicate normalized L2-norm values. The longer the horizontal bar of a *composite layer*, the larger its contribution.

5.3.1.7 Cross-layer connectivity

For better understanding of cross-layer depthwise convolutional layers, we compute layer-wise L2-norms of the cross-layer depthwise convolutional filter weights of DelugeNet-218, on CIFAR-10 and CIFAR-100. We provide visualizations in Figure 5.3. The weight’s L2-norms are normalized² by dividing them with the maximum layer-wise L2-norms of every block. We consider only cross-layer depthwise convolutional layers in the **Block Transition 1** (from Block 1 to Block 2), **Block Transition 2** (from Block 2 to Block 3), and the cross-layer depthwise convolutional layer (from Block 3 to classification) **before classification layer**. These are the cross-layer depthwise convolutional layers with the highest numbers cross-layer connections in the networks.

Generally, all the preceding *composite layers* contribute reasonably, with a few dominating. The weights (initialized uniformly) are no longer uniform for all layers in the trained models, being different from the connection rigidity exhibited by ResNets. For first and second block transitions, the last *composite layers* always contribute the most, somehow approximating the behaviours of conventional neural networks where all incoming information comes solely from the layer just before the current layer. On the other hand, for the cross-layer depthwise convolutional layer (before classification layer) connected to the third network block, the early *composite layers* generally contribute the most, and the final *composite layer* contributes moderately. We reckon that the features computed by the earlier *composite layers* are fairly ready for classification, and the subsequent *composite layers* just refine them further. Such phenomenon has also been observed in ResNets [182], where upper layers could be deleted without hurting performance much. In addition, we notice that some *composite layers* in the first block of DelugeNet-218 (CIFAR-100)

²The relative (as opposed to absolute) L2-norm values are sufficient, since BN layers follow cross-layer depthwise convolutional layers.

hardly have any contributions to Block Transition 1. This observation may suggest that *layer sparsity* can be potentially exploited for training DelugeNets.

5.3.2 ImageNet

Dataset: ImageNet (1000 classes) dataset [85] is the most widely used large-scale image classification dataset in recent years. We report the results for validation images. We follow the data augmentation scheme adopted in GoogleNet/Inception [109, 110] and ResNet-v2 [12] with the following augmentation techniques: scale [27] & aspect ratio [109] augmentation, PCA-based lighting augmentation [27], photometric distortions [183], and horizontal flipping. The images are normalized by subtracting them from channel-wise means and dividing them by channel-wise standard deviations.

5.3.2.1 Implementation

We implement and evaluate 3 different DelugeNet models on ImageNet dataset. Similar to ResNet models [12], before being passed to the first block, the feature map (after first layer) is downsampled to spatial dimensions of 56×56 via max-pooling. We set the base feature channel dimensions (widths) of all ImageNet-based DelugeNet models to be identical to those of ResNets' [12] – $\{64, 128, 256, 512\}$. Most of the network architectural details follow ResNets' closely, and they are not necessarily optimal for DelugeNets. Moreover, we emphasize on great simplicity when choosing the *composite layer* counts for DelugeNets, setting the number of *composite layers* in each block to be larger than or equal to that of its preceeding block. This is in contrast to the carefully tuned *composite layer* counts [12] (e.g., $\{3, 4, 23, 3\}$, $\{3, 8, 36, 3\}$) in ResNets. The specifications of DelugeNet models are as follows:

DelugeNet-92 has *composite layer* counts of $\{7,7,8,8\}$, for its 4 blocks (in sequential ordering) respectively. **DelugeNet-104** and **DelugeNet-122** are two deeper DelugeNet models, with *composite layer* counts of $\{7,8,9,10\}$ and $\{7,9,11,13\}$ respectively.

The ImageNet-based models are initialized similarly to the CIFAR models. Training is carried out with SGD over a total of 100 training epochs, with Nestorov Momentum [181] and weight decay rate [47] of $1e-4$. We start with learning rate of 0.1, and decay it by factor of 0.1 at the end of every thirty epoch. The training mini-batch size is 256. In view of large model and image sizes, we train the models in multi-GPU mode with 8 GPUs, splitting each mini-batch into 8 portions. These are standard training settings and similar to those used to train ImageNet-based ResNets in existing works [12, 184].

5.3.2.2 Results

The top-1 and top-5 classification errors achieved by DelugeNets on ImageNet validation dataset are presented in Table 5.3, along with the numbers of floating-point operations (GigaFLOPs/GFLOPs) required by the models to process one image. For comparison, we include the results of ResNet-v2 [12], Stochastic Depth [121], [14], and DenseNet [113]. We also report the actual inference time in milliseconds (ms) using Nvidia V100 GPU on a single image.

DelugeNet-92 with merely 43.4M parameters outperform ResNet-101 (top1 +0.39%, top5 +0.18%) and even ResNet-152 (top1 +0.11%, top5 +0.13%). Besides, with 25.5M less parameters and about half (11.8 GFLOPs) of the Wide-ResNet-50’s FLOPs (22.8 GFLOPs), DelugeNet-92 performs comparably to Wide-ResNet-50. Both deeper models DelugeNet-104 and DelugeNet-122 further push down the classification errors substantially. Remarkably, DelugeNet-122 attains classification errors comparable to ResNet-200’s, despite needing just about half (15.2 GFLOPs) of

Model	#Params	GFLOPs	Inference Time (ms)	Top-1	Top-5
ResNet-101 [12]	44.6M	15.7	70.6	22.44	6.21
ResNet-152 [12]	60.3M	23.1	73.3	22.16	6.16
ResNet-200 [12]	64.8M	30.1	99.6	21.66	5.79
ResNet-152 (Stochastic Depth) [121]	60.3M	23.1	73.3	23.38	6.92
Wide-ResNet-50 [14]	68.9M	26.2	22.8	21.9	6.03
DenseNet-161 [113]	28.7M	15.5	86.7	22.33	6.15
DelugeNet-92	43.4M	11.8	54.8	22.05	6.03
DelugeNet-104	51.4M	13.2	68.3	21.86	5.98
DelugeNet-122	63.6M	15.2	75.5	21.53	5.86

TABLE 5.3: ImageNet validation errors (single 224×224 crops).

the computations required by ResNet-200 (30.1 GFLOPs). With the flexible cross-layer connections established by cross-layer depthwise convolutions, DelugeNet-122 is robust against the overfitting issue caused by allocating more parameters to the spatially smaller blocks. Moreover, DelugeNet-122 outperforms DenseNet-161 given similar model complexities. It is worth pointing out that DenseNet-161 [113] is the largest DenseNet model reported for ImageNet dataset that could fit into 4 GPU cards for training.

Given similar or considerably lower model budgets (GFLOPs, number of parameters, inference time), DelugeNets are able to surpass ResNets, although DelugeNets use a very simple heuristic of setting the *composite layer* counts. While it is prohibitively hard to train larger DenseNet models with as many parameters and similar performance as large ResNet models such as ResNet-200, we are able to train a large DelugeNet, DelugeNet-122 that performs competitively to large ResNet-200 at a lower computational cost. In the next section, we discuss the GPU memory consumptions (during training) of DelugeNets and DenseNets to further showcase the computational resource advantage of DelugeNets.

5.4 Memory Usage

In ResNets, the residual (addition) operation allows memory buffers to be shared or reused across consecutive composite layers. However, for DelugeNets and DenseNets [113], the output activations and gradients of the last convolutional layer of every composite layer have to be retained persistently during training. For instance, when doing training ($\hat{\mathbf{T}}$) and inference ($\hat{\mathbf{I}}$) with Wide-DelugeNet-146 on CIFAR-100 (batch size of 32), the occupied GPU memory is roughly $\{\hat{\mathbf{T}}: 2.8\text{G}, \hat{\mathbf{I}}: 0.61\text{G}\}$, while its ResNet-baseline counterpart only requires $\{\hat{\mathbf{T}}: 1.3\text{G}, \hat{\mathbf{I}}: 0.44\text{G}\}$. The gap is smaller during inference ($1.5\times$) than in training ($2.2\times$). On the other hand, DenseNet($k = 24$), DenseNet-BC($k = 24$), and DenseNet-BC($k = 40$) require $\{\hat{\mathbf{T}}: 6.3\text{G}, \hat{\mathbf{I}}: 0.87\text{G}\}$, $\{\hat{\mathbf{T}}: 8.6\text{G}, \hat{\mathbf{I}}: 0.75\text{G}\}$, and $\{\hat{\mathbf{T}}: 8.4\text{G}, \hat{\mathbf{I}}: 1.1\text{G}\}$ respectively. Wide-DelugeNet-146 is more memory-efficient than DenseNet($k = 24$), while DenseNet-BCs are very memory-costly.

5.5 Concluding Remarks

We extend depthwise convolutional layers to *cross-layer depthwise convolutional layers*, which facilitate cross-layer connections in our proposed DelugeNets. The cross-layer information inflows in DelugeNets are **flexible** (cross-layer depthwise convolutional filters are learned) yet **massive** (output widths of composite layers are regular). Experiments indicate that DelugeNets are quite comparable to state-of-the-art models in terms of accuracies, and yet DelugeNets have lower model complexities. This suggests that DelugeNets may be potentially useful in energy-efficient deep learning. DelugeNets outperform ResNets across different datasets with varying scales of image-level classes and data examples. Furthermore, DelugeNets are more applicable and accessible than DenseNets since we can train

DelugeNets at a wide range of model sizes even for very large model sizes but doing so is not feasible with highly-expensive DenseNets. In a nutshell, DelugeNets achieve a very good balance between ResNets (connecting directly to just the previous layer's activations with an addition operation) and DenseNets (aggregating all preceding layers' activations with learnable full connection weights) which can be seen as the two extremes of the cross-layer connectivity spectrum.

Chapter 6

Stochastic Downsampling for Cost-Adjustable Inference and Improved Regularization

6.1 Introduction

Convolutional networks (CNNs) [28] have greatly accelerated the progress of many computer vision areas and applications in recent years. Despite their powerful visual representational capabilities, CNNs are bottlenecked by their immense computational demands. Recent CNN architectures such as Residual Networks (ResNets) [12, 20] and Inception [185] require billions of floating-point operations (FLOPs) to perform inference on just one single input image. Furthermore, as the amount of visual data grows, we need increasingly higher-capacity (thus higher complexity) CNNs which have shown to better utilize these large visual data compared to their lower-capacity counterparts [19].

There have been works which tackle the efficiency issues of deep CNNs, mainly by lowering numerical precisions (quantization) [186–188], pruning network weights

[189–193], or adopting separable convolutions [13, 34, 114]. These methods result in more efficient models which have fixed inference costs (measured in floating-point operations or FLOPs). Models with fixed inference costs cannot work effectively in certain resource-constrained vision systems, where the computational budget that can be allocated to CNN inference depends on the real-time resource availability. When the system is lower in resources, it is preferable to allocate a lower budget for more efficient or cheaper inference, and vice versa. Moreover, in some cases, the exact inference budget cannot be known beforehand during training time.

As a simple solution to such a concern, one could train several CNN models such that each has a different inference cost, and then select the one that matches the given budget at inference time. However, it is extremely time-consuming to train many models, not to mention the computational storage required to store the weights of many models. In this work, we focus on CNNs whose computational costs are dynamically adjustable at inference time. A CNN with cost-adjustable inference only has to be trained once, and it allows users to control the trade-off of inference cost against network accuracy/performance. The different inference instances (each with different inference cost) are all derived from the **same model weights**.

For cost-adjustable inference in CNNs, we propose a novel training method – Stochastic Downsampling Point (SDPoint). A SDPoint instance is a network configuration consisting of a unique downsampling point (layer index) in the network layer hierarchy as well as a unique downsampling ratio. As illustrated in Figure 6.1, at every training iteration, a SDPoint instance is randomly selected (from a list of instances), and downsampling happens based on the downsampling point and ratio of that instance. The earlier the downsampling happens, the lower the total computational costs will be, given that spatially smaller feature maps are cheaper to process.

TABLE 6.1: Progression of feature map spatial sizes during training of a (**Left**) conventional CNN, (**Right**) with SDPoint. The *costs* here refer to computational costs measured in numbers of floating-point operations (FLOPs).

During inference, a SDPoint instance can be deterministically handpicked (among the SDPoint instances seen during training) to match the given inference budget. Existing approaches [118–120] to achieve cost-adjustable inference in CNNs work by evaluating just subparts of the network (e.g., skipping layers or skipping sub-paths), and therefore not all network weights are utilized during cheaper inference. In contrast to existing approaches, SDPoint makes full use of all network weights regardless of the inference costs, thus making better use of network representational capacity. Moreover, the (scale-related) weight sharing across the SDPoint instances (each with a different downsampling and downsampling ratio) provides significant improvement in terms of model regularization. On top of these advantages, SDPoint is architecture-neutral, and it adds no weight or training overheads. We carry out experiments on image classification with a variety of recent network architectures to validate the effectiveness of SDPoint in terms of cost-accuracy performances and regularization benefits.

6.2 Preliminaries: Conventional CNNs with Fixed Downsampling Points

Conventionally, downsampling of feature maps happens in CNNs at several predefined fixed locations/points in the layer hierarchy, depending on the architectural designs. For example, in ResNet-50, spatial pooling (happens after the first ReLU layer, and after the last residual block) and strided convolutions (or convolution with strides > 1 which happens right after the 3rd, 7th, and 13th residual blocks) are used to achieve downsampling. Between these downsampling layers are *network stages*. Downsampling in CNNs trades low-level spatial information for richer high-level semantic information (needed for high-level visual tasks such as image classification) in a gradual fashion.

During network inference, these fixed downsampling points have to be followed exactly as how they are configured during training, for optimal accuracy performance. In this work, we go beyond fixed downsampling points – we develop a novel stochastic downsampling method named Stochastic Downsampling Point (SDPoint) which does not restrict downsampling to happen every time at same fixed points in the layer hierarchy. The proposed method is complementary to the fixed downsampling points in existing network architectures, and do not replace them. SDPoint can be simply plugged into existing network architectures, and no major architectural modifications are required.

6.3 Stochastic Downsampling Point

A Stochastic Downsampling Point (SDPoint) instance has a unique downsampling point $p \in \mathbb{Z}$ and a unique downsampling ratio $r \in \mathbb{R}$ which are stochastically/randomly selected during network training. A p and a r are stochastically selected at the beginning of each network training iteration, and downsampling occurs to the selected point (based on the selected ratio) for all samples in the current training mini-batch. The downsampling points and a downsampling ratios will be discussed more thoroughly in the upcoming sections. Downsampling is performed by a downsampling function $D(\cdot)$ which makes use of some downsampling operations. When the selected point falls at the lower layer in the layer hierarchy, the downsampling happens earlier (in the forward propagation), causing quicker loss of spatial information in the feature maps, but more computation savings. Conversely, spatial information can be better preserved at higher computational costs, if the stochastic downsampling happens later.

SDPoint can effectively turn the feature map spatial sizes right before prediction layers to be different from original sizes, and this could cause shape incompatibility between the prediction layer weights (as well as labels) and the convolutional

outputs (before prediction layers). To prevent this, we preserve the feature map spatial size in the last network stage, regardless of stochastic downsampling taking place or not, by adjusting convolution strides and/or pooling sizes accordingly. For example, in image classification networks, we consider the global average pooling layer [116] and the final classification layer to be the last network stage. Therefore, regardless of the spatial size (variable due to SDPoint) of the incoming feature maps, we globally pool them to have spatial size of 1×1 .

6.3.1 Downsampling Operation

As discussed in Sect. 6.2, the downsampling operation employed in $D(\cdot)$ can be either pooling [32] (*average* or *max* variations) or strided convolution. We opt for average pooling (the corresponding downsampling function is denoted as $D_{\text{avg}}(\cdot)$), rather than strided convolutions or max pooling for several reasons. Strided convolutions are the preferred way to do downsampling in recent network architectures, because they add extra parameters (convolution weights) and therefore improving the representational capability. In this work, we want to rule out the possible performance improvements from increase in weight numbers (rather than the SDPoint itself). Moreover, strided convolutions with integer-valued strides cannot work well with arbitrary downsampling ratios (see Sect. 6.3.3). On the other hand, average pooling is preferred over max pooling in this chapter due to the fact that max pooling itself is a form of non-linearity. Using max pooling as the downsampling operation could either push for a greater non-linearity in the network (positive outcome) which is unfair to the baselines, or could exacerbate the vanishing gradient problem [194] commonly associated with deep networks (negative outcome).

Besides, the effectiveness of average pooling has been validated through its extensive and critical roles in recent CNN architectures (e.g., global average pooling [20, 116], DenseNets' transition [113]). Particularly, due to its scale insensitivity,

average pooling allows ResNet models pre-trained on one input size to be effectively evaluated/applied on other input sizes with favourable results [12]. In addition, average pooling is known to be more robust than max pooling against outliers and noise [195]. It is also a conceptually simple method that has been empirically verified to outperform max pooling consistently over a range of CNN network architectures [196].

6.3.2 Downsampling Points

At every training iteration, a downsampling point p for a SDPoint instance can be drawn from a discrete uniform distribution on a set of predefined downsampling point indices $P = \{0, 1, 2, \dots, \mathcal{N}-1, \mathcal{N}\}$, with $\mathcal{N} + 1$ number of points. In this work, the downsampling point candidates are the points between two consecutive CNN “basic building blocks”, mirroring the placements of *fixed downsampling* layers in conventional CNNs. We keep the original network (without stochastic downsampling) as an instance by assigning the index $p = 0$ to it, so that we can perform full-cost inference later. Let $F(\cdot)$ denote the function carried out by the i -th basic building block, w_i denote the network weights involved in the block. For a given input x_i and downsampling ratio r , the downsampling is carried out as following:

$$y_i = D_{\text{avg}}(F(x_i; w_i); s_i, r) \quad (6.1)$$

to obtain the output y_i . The downsampling switch denoted as $s_i \in \{\text{True}, \text{False}\}$ is turned on if $p = i$.

For non-residual CNNs (e.g., VGG-Net [149]), the basic building block comprises 3 consecutive *convolutional*, *Batch Normalization* (BN) [137], *non-linear activation* layers. On the other hand, for residual networks, residual blocks are considered as the basic building blocks. The downsampling point p can be stochastically selected to be any point between any 2 basic building blocks in the network, where

downsampling happens. Since a residual block involves two streams of information – (i.) the identity skip connection and (ii.) the non-linear function consisting of several network layers, we apply stochastic downsampling function $D_{\text{avg}}(\cdot)$ to the point right after the residual addition operation. We also experiment with Densely Connected Networks (DenseNets) [113] in this chapter. For DenseNets, the SDPoint downsampling points are the points right behind each block concatenation operation, mirroring the *fixed downsampling* in DenseNets.

In principle, each mini-batch sample could have its unique downsampling point p_i (for stronger stochasticity), but due to practical reasons (e.g., training efficiency, ease of implementation), we resort to using the same p_i for all samples in a mini-batch. While it is possible to have more than one downsampling points in each training iteration, the number of possible combinations or SDPoint instances would become excessively large. Some instances would deviate too much from the original network, in terms of computational cost and accuracy performance. Moreover, the final feature map resolution (spatial size) goes down exponentially with the increase of the number of stochastic downsampling points. For instance, with one stochastic downsampling points and 0.5 downsampling ratio, the final feature map’s *spatial area* is 0.25 $((0.5)^2)$ that of the original area. It becomes 0.0625 $((0.5 \times 0.5)^2)$ with two stochastic downsampling points, and 0.015625 $((0.5 \times 0.5 \times 0.5)^2)$ with three. In such cases, sharing the weights of the entire network is implausible given such vast differences in the feature map resolutions. Therefore, we opt for single **stochastic downsampling point** in this work.

6.3.3 Downsampling Ratios

We consider a set of downsampling ratios R , which the SDPoint instance can stochastically draw a downsampling ratio r from, for use at current training iteration. Similar to Sect. 6.3.2, downsampling ratios are drawn according to discrete

uniform distributions. The ratios cannot be too low that they hamper the training convergence (due to weight-sharing unfeasibility). And, we consider only a small number of downsampling ratios in R to prevent an excessive number of SDPoint instances, which would cause great difficulty in experimentally evaluating all SDPoint instances for cost-adjustable inference. A recent experimental study [31] on CNNs finds that it is sufficient to make qualitative conclusions about optimal network structure that hold for the full-sized (224×224 image resolution) ImageNet [85] classification task, by using just 128×128 (roughly half the original resolution) input images. Conceivably, the same network structure/architecture that works well with a certain image resolution is likely to work well with a resolution double/half of that. Motivated by the above-mentioned heuristics and experimental finding, we come up with the downsampling ratio set $R = \{0.5, 0.75\}$. The same ratios have also been used by [123] for “multiscale-input” semantic segmentation. The same hyperparameter R is used across all experiments in this chapter.

Downsampling with such fractional downsampling ratios cannot be trivially achieved with integer-valued pooling hyperparameters. For example, pooling a 28×28 feature map to a 21×21 one (with r of 0.75 and minimal overlaps) cannot be easily done by tuning just the pooling size and stride. To this end, we adopt a spatial pooling strategy (which works along with the pooling choice in Sect. 6.3.1) akin to that of Spatial Pyramid Pooling [122] that generates fixed-length representation via adaptive calculations of pooling sizes and strides.

6.3.4 Training with SDPoint

SDPoint gives rise to a new training algorithm for CNNs. The training algorithm consolidating all the previously introduced SDPoint concepts is given in Algorithm 1. $F(\cdot)$ denotes the generic nonlinear building network block in CNNs. For simplicity’s sake, we omit the other network layers which are not basic building blocks

Algorithm 1 : Training with SDPoint

```

1:  $P = \{0, 1, 2, \dots, \mathcal{N}-1, \mathcal{N}\}$  ▷ Downsampling Points
2:  $R = \{0.5, 0.75\}$  ▷ Downsampling Ratios
3: while given a training mini-batch  $x$  do
4:   Randomly draw  $p$  from  $P$ 
5:   Randomly draw  $r$  from  $R$ 
6:    $x_1 = x$ 
7:   for  $i \in \{1, 2, \dots, \mathcal{N}-1, \mathcal{N}\}$  do ▷ Forward pass
8:     if  $i == p$  then  $s_i = \text{True}$  else  $s_i = \text{False}$ 
9:      $x_{i+1} = D_{\text{avg}}(F(x_i; w_i); s_i, r)$ 
10:  end for
11:  Compute  $loss$  with  $x_{\mathcal{N}+1}$ 
12:  Backward pass
13:  Weight updates
14: end while

```

– typically the starting and ending layers. In a nutshell, Algorithm 1 shows that whenever a building block index i is equal to the downsampling point p , the downsampling switch s is turned on. Stochastic downsampling then happens to the output of i -th building block, with the stochastic downsampling ratio r . It is important to point out that the (stochastic) downsampling does not happen, if p is drawn to be 0, allowing the network to work in its original “unadulterated” form.

6.3.5 Regularization from Weight Sharing

SDPoint can be seen as a regularizer for CNNs. When stochastic downsampling takes place, the receptive field size becomes larger and it causes a sudden shrinkage of spatial information in the feature maps. The network has to learn to adapt to such variations during training, and perform **weight sharing** across the downsampled feature maps and the originally sized feature maps (when $p = 0$). In addition to robustness in terms of receptive field size and spatial shrinkage, SDPoint also necessitates the convolutional layers to accommodate for different “*padded pixel* to *non-padded pixel*” ratios. For example, applying a 3×3 convolutional filter (weights) (with zero-padding of 1) to a 8×8 feature map gives a padded-pixel

ratio of 0.44, compared to 0.56 ratio resulted from applying the same filter to 6×6 feature map. Zero-padded pixels are quite similar to the *zero-ed out* activations caused by Dropout [43], in the sense that they both are missing values. Thus, a higher padded-pixel ratio is akin to having a higher number of *dropped-out* activations, vice versa. This form of variation provides further regularization boost. Experimentally, we find that even with the use of heavy data augmentation – such as “scale + aspect ratio” augmentation [109, 185], SDPoint can still help.

6.4 Cost-adjustable Inference

A network that can perform inference at different computational costs depending on the user requirements, is considered to be capable of *cost-adjustable inference*. Opting for a lower inference cost usually results in a lower prediction accuracy, and vice versa. SDPoint naturally supports cost-adjustable inference, given that SDPoint instances have varying computational costs, given the different downsampling point locations and downsampling ratios. More importantly, the instances have all been trained to minimize the same prediction loss, and this helps them to work relatively well for inference. During inference, one may handpick a SDPoint instance (with its downsampling point p and downsampling ratio r) to make the inference cost fit a particular inference budget.

6.4.1 Instance-Specific Batch Normalization

As mentioned in Sect. 6.3, SDPoint instances are trained in such a way that every training mini-batch and iteration shares the same SDPoint instance. For a SDPoint instance, the prediction and loss minimization during training are based on the Batch Normalization (BN) statistics (means and standard deviations) of that particular instance. Therefore, using the BN statistics accumulated over

many training iterations (and thus many different SDPoint instances) for inference causes inference-training “mismatch”. A similar form of inference-training “mismatch” caused by BN statistics has also been observed by [45] in another context. The BN statistics required for one SDPoint instance should differ from that of another instance. When using the same (accumulated) BN statistics to perform cost-adjustable inference, the inference accuracies could be jeopardized.

To address the “mismatch” issue, we compute SDPoint instance-specific BN statistics, and use them for cost-adjustable inference. Disentangling the different SDPoint instances by unsharing BN statistics makes the inference more accurate. The computational storage overhead resulted from *instance-specific BN* statistics is relatively low, as BN statistics of some earlier layers can be shared among certain SDPoint instances that downsample at later layers.

6.5 Experiments

Experiments are carried out on image classification tasks to evaluate SDPoint. We consider image classification datasets with varying dataset scales in terms of numbers of categories/classes and sample counts: CIFAR-10 [179] (50k training images, 10k validation images, 10 classes), CIFAR-100 [179] (50k training images, 10k validation images, 100 classes), ImageNet [85] (1.2M training images, 50k validation images, 1000 classes). For inference cost comparison, we measure the model costs in terms of floating-point operation numbers (FLOPs) needed for forward propagation of single image. We treat *addition* and *multiplication* as two separate operations.

FIGURE 6.1: WRNs' and DenseNetBC's cost-error plots on CIFAR-10 (**Top**) and CIFAR-100 (**Bottom**). It is observed that models trained with SDPoint consistently outperform their non-SDPoint counterparts, given the same computational budgets.

6.5.1 CIFAR

For CIFAR-10 and CIFAR-100, the baseline architectures are Wide-ResNet [14] (WRN-d28-w10 and WRN-d40-w4) and DenseNetBC-d40-g60 [113]. ‘d’, ‘w’, ‘g’ stand for the network depth, widen factor of WRN, and growth rate of DenseNetBC, respectively. The training hyperparameters (e.g., learning rates, schedules, batch sizes, augmentation) follow the ones in original papers, except for training epoch numbers which we fix to 400 for all. The original learning rate schedules still apply (e.g., learning rates are dropped at 50% and 75% of total number of training epochs). The numbers of SDPoint downsampling points ($\mathcal{N}$) for {WRN-d28-w10, WRN-d40-w4, and DenseNetBC-d40-g60} are {12, 18, 12} respectively. As mentioned in Sect. 6.3.3, the downsampling ratios are drawn uniformly from $R = \{0.5, 0.75\}$.

6.5.1.1 Baseline Comparison

We compare SDPoint with some baseline methods related to ours, in terms of cost-adjustable inference performance. The classification error-cost performance plots on CIFAR-10 and CIFAR-100 are shown in Figure 6.1. Note that for SDPoint and baseline methods, not all instances of the same model appear on the plots; if a higher-cost instance performs worse than any lower-cost instance, it is not shown. Each model (evaluated on a dataset) is trained only once to obtain its cost-error plot. In the cost-error plot, for each cost (FLOP) point on the horizontal axis, we report the error achieved by the best-performing SDPoint instance that has a cost (FLOPs) equal to or lower than the particular horizontal-axis point we are looking at.

1. **Early-Exits (EE):** We train models based on the WRN with intermediate classifiers (branches) which allow early-exits (EE), following the design of

BranchyNet [119]. Each *network stage* in the main network has two evenly spaced branches, and the branches each have single-repetition of building block per *branch network stage*. The blocks in the branches follow the same hyperparameters (e.g., #channels) as the blocks in the original network. For cost-adjustable inference, we evaluate every branch, and make all samples “exit” at the same branch. The early-exit models have considerably more weights than both the baseline models and SDPoint-based models. We conjecture that the relatively worse performance of EE is due to lack of full network weight utilization. Also, EE forces CNN features to be classification-ready in early stage, thus causing higher layers to rely heavily on the classification-ready features, instead of learning better features on their own.

2. **Multiscale Training (MS):** Multiscale (MS) training is a baseline method inspired by [94, 122, 123]. The input images are downsampled using bilinear interpolations, to an integer-valued size randomly chosen from sizes ranging from half (16×16) to full size (32×32), with step size of 1 pixel. This is done for every training iteration, similar to SDPoint. The number of “instances” (16) resulted from multiscale training is close to the downsampling point numbers of applying SDPoint to WRNs and DenseNetBC(s). Also, the ranges of cost-adjustable inference costs among them are comparable. Instance-specific BN statistics are applied. The cost-adjustable performance of MS consistently trails behind that of SDPoint, as input downsampling causes more drastic information loss than feature map downsampling (see Chapter 2.4.5).

3. **Uniform Batch Normalization (UBN):** To validate the effectiveness of SDPoint instance-specific BN, we show the results of a SDPoint baseline whose BN statistics are averaged from many training iterations, and are uniform for all of its instances. There are consistent classification performance

Model	# Params	GFLOPs	CIFAR-10	CIFAR-100
ResNeXt-d29-c08 [13]	34.4M	10.8	3.65	17.77
ResNeXt-d29-c16 [13]	68.1M	21.4	3.58	17.31
DenseNetBC-d250-g24 [113]	15.3M	10.1	3.62	17.60
DenseNetBC-d190-g40 [113]	25.6M	18.7	3.46	17.18
WRN-d40-w4 [14]	8.9M	2.6	4.29	20.78
WRN-d40-w4 [14] with SDPoint	8.9M	2.5/2.6	3.73 (↓ 0.56)	19.55 (↓ 1.23)
WRN-d28-w10 [14]	36.5M	10.5	3.84	18.51
WRN-d28-w10 [14] with SDPoint	36.5M	6.5/10.1	3.35 (↓ 0.49)	17.53 (↓ 0.98)
DenseNetBC-d40-g60 [113]	4.3M	3.6	3.99	20.00
DenseNetBC-d40-g60 [113] with SDPoint	4.3M	2.7/3.6	3.39 (↓ 0.60)	19.25 (↓ 0.75)

TABLE 6.2: CIFAR-10 and CIFAR-100 validation errors (%). The GFLOPs with 2 values separated by “/” are for CIFAR-10 and CIFAR-100 respectively.

Model	Error(%) / Deterministic	Error(%) / Stochastic
WRN-d28-w10	18.51	-
*with Dropout [43]	18.05	17.98
*with Swapout [45], $Linear(1, 0.5)$	20.55	18.68
*with Swapout [45], $Linear(1, 0.8)$	19.21	18.65
*with SDPoint	17.53	17.20

TABLE 6.3: *Deterministic* and *stochastic* inference results of training WRN-d28-w10 [14] with different stochastic training methods on CIFAR-100.

gaps between using UBN statistics and instance-specific BN statistics, suggesting that it is preferable to keep instance-specific statistics for inference.

We compare the different cost-adjustable methods in terms model storage requirements. On CIFAR-100 dataset with WRN-28-10, both MS and SDPoint take about 146M of storage space, while EE takes 148M due to additional classifier weights.

6.5.1.2 State-of-the-art Comparison

Table 6.2 reports the CIFAR validation results of state-of-the-art (SOTA) ResNeXt [13] and DenseNetBC [113] models, for comparison with ours. For each SDPoint-enabled model, we show the results (giga-FLOPs, classification errors) from the

best-performing SDPoint instance among its instances. Notably, WRN-d28-w10 with SDPoint is competitive to SOTA models on CIFAR-100, and it outperforms them on CIFAR-10. Overall, SDPoint considerably improves classification performance without bringing in additional parameters and computational costs, unlike the SOTA models which require about $2\times$ model complexity to attain slight improvements. In fact, the best SDPoint-enabled models on CIFAR-10 have reduced inference costs (FLOPs). We reckon that a prolonged preservation of spatial details (i.e., no early downsampling) in CNN feature maps is not crucial to a dataset with relatively low label complexity such as CIFAR-10. This reveals a drawback of current practice of using CNNs in “one-size-fits-all” fashion.

6.5.1.3 Stochastic Training and Inference

Since our method is related to the stochastic training family, we experimentally compare SDPoint with Dropout [43] and Swapout [45] (using 2 linear decay rules suggested), with WRN-d28-w10 as the baseline and CIFAR-100 dataset. The results are given in Table 6.3. Stochastic inference was proposed by [45] as a way to tackle the poor interaction of Swapout/Dropout with BatchNorm. For fair comparison, we report also the results of performing stochastic inference (50 trials). In both *deterministic* and *stochastic* inference settings, we find that SDPoint outperforms the rest.

6.5.2 ImageNet

We consider ResNeXt-d101-c32 [13] and PreResNet-d101 [12] as baseline architectures. ‘c’ stands for ResNeXt’s cardinality. With SDPoint, there are 33 downsampling points ($\mathcal{N}$) per model. We train the models on ImageNet-1k [85] training set, and evaluate them on the validation set (224×224 center crops). All models

Model	# Params	GFLOPs	Top-1	Top-5
ResNeXt-d101-c64 [13]	~89M	~32	20.4	5.3
DenseNetBC-d264 [197]	~73M	~26	20.4	-
ResNeXt-d101-c32 [13]	44.3M	16.0	21.2	5.6
ResNeXt-d101-c32 [13] with Stochastic Depth [121]	44.7M	15.7	22.2 (↑ 1.0)	5.9 (↑ 0.3)
ResNeXt-d101-c32 [13] with SDPoint	44.3M	16.0	20.4 (↓ 0.8)	5.3 (↓ 0.3)
PreResNet-d101 [12]	44.7M	15.7	22.0	6.1
PreResNet-d101 [12] with Stochastic Depth [121]	44.7M	15.7	22.8 (↑ 0.8)	6.4 (↑ 0.3)
PreResNet-d101 [12] with SDPoint	44.7M	15.7	21.4 (↓ 0.6)	5.6 (↓ 0.5)
PreResNet-d101 [12] with SACT [198]	45.0M	11.1	24.4	7.2
PreResNet-d101 [12] with SDPoint	44.7M	7.7	24.3	7.2

TABLE 6.4: ImageNet top-1 and top-5 validation errors (%), with model parameter numbers and giga-FLOPs (GFLOPs).

are trained using training hyperparameters and “scale + aspect ratio” augmentation [109] similar to [13]. Note that we do not allocate more training epochs to models with SDPoint. The cost-error plots are given in Figure 6.2 and 6.3, for PreResNet-d101 and ResNeXt-d101-c32 respectively, along with some **fixed-cost** & **carefully designed**¹ baseline models from the same architecture families. The cost-error plots are obtained as in 6.5.1.1. Overall, models trained with SDPoint can roughly match the performance of baseline models in the lower-cost range, and surpass them in the upper-cost range. Notably, to obtain cost-error plots, SDPoint-enabled models only have to be trained once. The baseline models are **trained separately**, resulting in a huge total number of epochs ($\#models \times \#epochs$ per model) and storage cost.

¹model hyperparameters are carefully chosen by the authors [12, 13] to optimize accuracy performances under some budget constraints.

FIGURE 6.2: PreResNets’ [12] cost-error plots on ImageNet. PreResNet-d101 (SDPoint) only has to be trained once (as a single model), while the baseline models (without SDPoint) has to be trained separately with huge training and storage costs.

6.5.2.1 Ablation Study

We study the effects of choice of SDPoint downsampling points and downsampling ratios on cost-adjustable inference performance. For this, we train a ResNeXt-d101-c32 with default SDPoint hyperparameters (downsampling points at the end of every residual block, downsampling ratios of $\{0.5, 0.75\}$), as well as 2 baseline models with (i) downsampling points at the end of every other residual block dubbed *alternate* (ii) downsampling ratio of just $\{0.75\}$ dubbed *075*. They are shown on Figure 6.3. Either removing the 0.5 downsampling ratio or alternating blocks for downsampling gives worse results, due to reduced stochasticity (and regularization strengths).

FIGURE 6.3: ResNeXts’ [13] cost-error plots on ImageNet. Like Figure 6.2, any of ResNeXt-d101-c32 (SDPoint..) only has to be trained once (as a single model).

6.5.2.2 State-of-the-art Comparison

We compare our models with SOTA ResNeXt-d101-c64 [13] and DenseNetBC-d264-g48 [197] models in Table 6.4. SDPoint pushes the top-1 and top-5 validation errors of ResNeXt-d101-c32 down to 20.4% and 5.3% respectively, which are (previously) only attainable by SOTA models with roughly $2\times$ inference costs and parameter counts. We also display the results (and mean FLOPs) of Spatially Adaptive Computation Time (SACT) [198] paired with PreResNet-d101, and compare it to a SDPoint instance of our PreResNet-d101 that achieves similar classification errors. SDPoint merely needs 69% of FLOPs needed by SACT to achieve similar results. SACT saves computation by skipping layers (and network parameters) for certain locations in feature maps according to learned policy and inputs, while SDPoint downsamples feature maps to save computation (but makes full use of network

parameters & capacity during inference). We contend that in cost-accuracy trade-off for inference, reducing feature map spatial sizes is less harmful to accuracy than skipping network parameters/layers. As a regularization study, we train the 2 baseline models with Stochastic Depth [121] (using suggested decay rule), and find that they considerably degrade the classification performance unlike SDPoint.

6.5.2.3 Analysis

We provide some analyses of ResNeXt-d101-c32 (trained with SDPoint on ImageNet) regarding certain aspects of downsampling and SDPoint.

Cost-dependent misclassifications: We group ImageNet validation images (which are correctly classified with full inference cost) according to the minimum inference costs required to classify them correctly, and present some examples on Figure 6.4. More difficult examples that require higher inference costs (9.9, 16.0 GFLOPs) to be classified correctly, generally have size-dominant interfering objects/scenes (e.g., **hair dryer**, **cab**, **caldron**, **cock**, **tench**), in contrast to the easier examples (4.3 GFLOPs). Intuitively, pooling-based downsampling causes more information loss to smaller objects than to larger (size-dominant) objects, especially when it occurs at some early layer, where the semantic/context information is still relatively weak to distinguish objects of interest from interfering objects. So, for those difficult examples, it makes sense to preserve spatially informative object details longer in the CNN layer hierarchy, and downsample the feature maps only after they are semantically rich enough.

Scale sensitivity: Training CNNs with SDPoint involves stochastic downsampling of intermediate feature maps, which we hypothesize to be beneficial for scale sensitivity/invariance, as mentioned in Chapter 6.3.5. To validate this hypothesis, we vary the *pre-cropping*² sizes of ImageNet validation images in the range of

²It is a standard practice [12, 13, 20, 113] to resize images to a shorter side of 256 (*pre-cropping* size) before doing 224×224 center-cropping.

FIGURE 6.4: Some Imagenet validation examples grouped according to the **minimum** inference costs (FLOPs) required by ResNeXt-d101-c32 (with SDPoint) to classify them correctly, in terms of top-5 accuracy. The ground-truth label names are shown below their corresponding images.

256, ..., 352 with step size of 16, resulting in 7 *pre-cropping* sizes. For every *pre-cropping* size, 224×224 center image regions are cropped out for evaluation. The models involved are SDPoint-enabled ResNeXt-d101-c32, and the baseline without SDPoint. We compute the mean of all pairwise cosine similarities (a total of 21 pairs) resulted from the different *pre-cropping* sizes, in terms of **ImageNet 1k-class probability scores**. This is done for entire ImageNet validation set. The pairwise cosine-similarity mean obtained for baseline model is **0.944**, while for the SDPoint-enabled model, it is **0.961**. A higher cosine similarity is a strong indicator of the model being less sensitive to scales. This demonstrates that SDPoint can indeed benefit CNNs, in terms of scale sensitivity.

6.6 Concluding Remarks

We propose Stochastic Downsampling Point (SDPoint), a novel approach to train CNNs by downsampling intermediate feature maps. At no extra parameter/weight and training costs, SDPoint facilitates effective cost-adjustable inference and greatly improves network regularization (thus accuracy performance). Through experiments, we additionally find out that SDPoint can help to identify more optimal (yet less costly) sub-networks, sort input examples by various levels of classification difficulties (Figure 6.4), and make CNNs less scale-sensitive.

Chapter 7

Conclusion and Recommendations

7.1 Conclusion

This thesis is centered around deep learning (deep neural networks) and their applications in visual recognition tasks. We propose novel learning algorithms in deep learning directed towards specific challenging problems encountered in visual recognition tasks at three recognition levels (pixel-level, object-level, and image-level). These visual recognition problems cannot be tackled effectively using off-the-shelf (general-purpose) deep learning methods nor they can be addressed by merely improving the training data. For all the visual recognition challenges that we tackle using deep learning, we perform extensive and rigorous experimental evaluations to validate the effectiveness of our proposed methods. Generally, our proposed methods are either competitive (at lower computational costs) or superior to strong baseline methods and state-of-the-art approaches.

Chapter 3 proposes a Recurrent Attentional Convolutional-Deconvolutional (RACDNN) network architecture that combines the complementary strengths of

CNN and RNN for saliency detection. The initial coarse saliency map that focuses on mostly global visual information is generated by an initial convolutional-deconvolutional network. At every sequential iteration, RACDNN uses a visual attention mechanism to attend to a particular image sub-region, and performs saliency refinement using convolutional-deconvolutional layers for that particular sub-region. Different image sub-regions (recurrent instances) are sequentially and iteratively processed with the same RACDNN weights. This greatly improves sample efficiency for effective training of RACDNN, without considering additional training data. As each iteration only focuses on an image sub-region, more local details of the salient objects can be captured. The results are saliency maps which are sharper and more detailed in terms of salient object shapes and boundaries, compared to those output by the globally-oriented convolutional-deconvolutional baseline.

In Chapter 4, we address the challenging problem of efficiently transferring knowledge from image classification to scale up object detection, using Auto-encoder Weight Transfer Network (AE-WTN⁺). AE-WTN⁺ tackles the weaknesses of conventional WTN pertaining to its training difficulty (under-fitting) and underuse the class information in the pre-trained classification weights. To tackle the under-fitting and ease optimization, normalization techniques are proposed to normalize the inputs and intermediate features of WTN. Whereas, to improve the class information utilization particularly for novel classes, a novel auto-encoding method for WTN is proposed. AE-WTN⁺ employs auto-encoding/reconstruction loss to reconstruct input weights from its weight predictions to preserve class information for all classes. Through experiments on large-scale detection datasets, AE-WTN⁺ demonstrates performance improvements over conventional WTN and baselines on novel classes while maintaining the performance on seen classes.

Chapter 5 focuses on improving the efficiency of flexible cross-layer connections in very deep neural networks for image classification. This is achieved using cross-layer

depthwise convolutional layers. For a preceding *composite layer* in the network, its succeeding *composite layers* learn lightweight layer-specific depthwise (channel-independent) weights, while the heavy cross-channel information mixing is handled by the 1×1 convolutional layers (with smaller numbers of channels) of the network building blocks. The deep network architecture that integrates with cross-layer depthwise convolutions — Deluge Networks (DelugeNets) can propagate information (feature activations, gradients) with a greater balance between connection flexibility and efficiency, compared to existing very deep network architectures. Experimentally, DelugeNets achieve results that are competitively to existing very deep network architectures, at considerably reduced computational and memory costs.

In Chapter 6, we introduce Stochastic Downsampling (SDPoint), a CNN training method to enable cost-adjustable inference and to enhance regularization, in the context of image classification. During training, SDPoint randomly selects a downsampling point/layer (in the network architecture) and a downsampling ratio, to downsample the corresponding feature maps. Different downsampling configurations called SDPoint instances entail differently downsampled feature maps, but they share the same network weights. This enables efficient network weight storage (i.e., only the weights of one network model have to be stored), and provides significantly regularization advantages due to the necessity of the weights to learn to work well with various SDPoint instances. For cost-adjustable inference, SDPoint bypasses the need to train multiple independent network models with different inference costs. During inference, one can manually handpick the SDPoint instance whose computational cost best fits a particular computational budget (that could be unknown beforehand during the training stage). Experimentally, we find that SDPoint is significantly superior to strong cost-adjustable baselines and state-of-the-art approaches, in terms of cost-accuracy performance and training time.

7.2 Recommendations for further research

While the contributions in this thesis have unlocked many previously unexplored or underexplored potentialities of deep learning for tackling a range of challenging visual recognition problems, there are still many that are worth considering for future works and extensions.

- Although the recurrent-attentional networks work in Chapter 3 is introduced to improve saliency detection performance, this work can be potentially extended and adapted to other pixel-level recognition tasks such as semantic segmentation [64] and image super-resolution [199] which share the similar issues of losing of local image details. Besides, it may be plausible to incorporate adaptive computation methods [198, 200] to the iterative refinement framework, for the network to learn to adaptively select a particular number of refinement iterations based on the given input, for enhanced computational efficiency.
- In Chapter 4, the knowledge (object class weights) transfer is only carried out between two visual recognition levels – transferring knowledge from image classification (image level) to object detection (object level). It would be interesting to consider knowledge transfer across the three recognition levels. Taking into account the different object class numbers (scales) of the recognition tasks (image classification: large-scale [85], object detection: medium-scale [152], semantic segmentation: small-scale [64]), one plausible idea is multi-level weight transfer which is to transfer weight knowledge from image classification network to object detection network, and then to transfer from the two above-mentioned networks to a pixel-level segmentation network.

The work in Chapter 4 focuses on weight transfer for just the final classification layer’s weights. There are other CNN layers in the network that may

contain useful knowledge for the downstream task. As a future work, it would be exciting to consider weight transfer through other earlier CNN layers.

- For Chapter 5's work, the cross-layer connections despite being efficient, are still relatively heavy compared to deep networks with only rigid non-flexible cross-layer connections. Many of the cross-layer connections potentially have negligible effects on the network. It would be interesting to explore the feasibility of weight/filter pruning [191–193] approaches in the context of (already quite efficient) cross-layer connections, to prune the insignificant cross-layer connection weights to further improve efficiency.
- Although the Stochastic Downsampling (SDPoint) method proposed in Chapter 6 allows cost-adjustable inference in convolutional neural networks, the (intended) inference costs have to be manually handpicked or specified by the users. The networks trained with SDPoint cannot optimally decide which downsampling configuration to use based on the given input image. It would be interesting to improve SDPoint such that during inference the networks can adaptively decide how to downsample given an input image; downsampling should happen earlier for easier inputs, and vice versa.

As mentioned in Chapter 6.3.2, having multiple stochastic downsampling could provide a lot more cost-accuracy choices but such strategy is expected to work poorly with the current full weight sharing mechanism. It would be interesting in future research to explore softer forms of weight sharing (e.g., partial weight sharing) that may work effectively for multiple stochastic downsampling.

- This thesis leverages deep learning to tackle the challenges in visual recognition independently for each recognition level. To achieve higher levels of visual intelligence in machines, it is imperative to unify the all levels of visual recognition under a single yet significantly larger and more complex visual recognition task/problem. It is crucial that in future research, there should

be explorations on how to define such a unified visual recognition problem, design effective evaluation methodologies, and develop learning models (potentially leveraging multi-task learning [201–204]) that are effective.

Author Publications

- **Jason Kuen**, Federico Perazzi, Zhe Lin, Jianming Zhang, Yap-Peng Tan. “Scalable Object Detection with Auto-Encoder Weight Transfer Network”. *IEEE/CVF International Conference on Computer Vision (ICCV)*, **Oral Presentation** (2019).
- **Jason Kuen**, Xiangfei Kong, Zhe Lin, Gang Wang, Jianxiong Yin, Simon See, Yap-Peng Tan. “Stochastic Downsampling for Cost-Adjustable Inference and Improved Regularization in Convolutional Networks”. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, **Poster Presentation** (2018).
- **Jason Kuen**, Xiangfei Kong, Gang Wang, Yap-Peng Tan. “DelugeNets: Deep Networks with Efficient and Flexible Cross-layer Information Inflows”. *International Conference on Computer Vision Workshop (ICCVW) on Compact and Efficient Feature Representation and Learning in Computer Vision (CEFRL)*, **Oral Presentation** (2017).
- **Jason Kuen**, Zhenhua Wang, Gang Wang. “Recurrent Attentional Networks for Saliency Detection”. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, **Poster Presentation** (2016).
- Ping Hu, Gang Wang, Xiangfei Kong, **Jason Kuen**, Yap-Peng Tan. “Motion-Guided Cascaded Refinement Network for Video Object Segmentation”. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, **Poster Presentation** (2018).

- Jianlou Si, Honggang Zhang, Chun-Guang Li, **Jason Kuen**, Xiangfei Kong, Alex Kot, Gang Wang. “Dual Attention Matching Network for Context-Aware Feature Sequence based Person Re-Identification”. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, **Poster Presentation** (2018).
- Jiuxiang Gu*, Zhenhua Wang*, **Jason Kuen**, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, Tsuhan Chen. “Recent Advances in Convolutional Neural Networks”. *Pattern Recognition (PR)* (2017).

Bibliography

- [1] Aravindh Mahendran and Andrea Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision*, 120(3):233–255, 2016. , [17](#), [18](#)
- [2] Rui Zhao, Hongsheng Li, and Xiaogang Wang. Saliency Detection by Multi-Context Deep Learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1265–1274, 2015. , [22](#), [36](#), [49](#), [53](#), [54](#), [55](#)
- [3] Guanbin Li and Yizhou Yu. Visual Saliency Based on Multiscale Deep Features. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015. , [22](#), [50](#), [53](#), [54](#), [55](#)
- [4] Changyang Li, Yuchen Yuan, Weidong Cai, Yong Xia, and David Dagan Feng. Robust Saliency Detection via Regularized Random Walks Ranking. In *IEEE Conference on Computer Vision and Pattern Recognition*, page 3166, jun 2015. , [52](#), [53](#)
- [5] Yao Qin, Huchuan Lu, Yiqun Xu, and He Wang. Saliency Detection via Cellular Automata. In *IEEE Conference on Computer Vision and Pattern Recognition*, number i, pages 1–11, jun 2015. , [52](#), [53](#)
- [6] Jingdong Wang, Huaizu Jiang, Zejian Yuan, Ming-ming Cheng, and Yihong Gong. Salient Object Detection : A Discriminative Regional Feature Integration Approach Salient Object Detection : A Discriminative Regional Feature

- Integration Approach. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2013. , [52](#), [53](#)
- [7] Wangjiang Zhu. Saliency Optimization from Robust Background Detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, IEEE Conference on Computer Vision and Pattern Recognition, pages 2814–2821, 2014. , [52](#), [53](#)
- [8] Xiaohui Li, Huchuan Lu, Lihe Zhang, Xiang Ruan, and Ming Hsuan Yang. Saliency detection via dense and sparse reconstruction. In *IEEE International Conference on Computer Vision*, ICCV '13, pages 2976–2983, 2013. , [52](#), [53](#)
- [9] Bowen Jiang, Lihe Zhang, Huchuan Lu, Chuan Yang, and Ming-Hsuan Yang. Saliency detection via absorbing markov chain. In *IEEE International Conference on Computer Vision*, ICCV '13, pages 1–16, 2013. , [52](#), [53](#)
- [10] Qiong Yan, J Shi, L Xu, and J Jia. Hierarchical Saliency Detection on Extended CSSD. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016. , [50](#), [52](#), [53](#)
- [11] Yuxin Wu and Kaiming He. Group Normalization. In *European Conference on Computer Vision*, 2018. , [63](#), [64](#), [71](#), [76](#), [77](#), [78](#)
- [12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity Mappings in Deep Residual Networks. In *European Conference on Computer Vision*, 2016. , [30](#), [81](#), [85](#), [86](#), [89](#), [91](#), [97](#), [98](#), [99](#), [103](#), [109](#), [119](#), [120](#), [121](#), [123](#)
- [13] Saining Xie, Ross Girshick, and Piotr Doll. Aggregated Residual Transformations for Deep Neural Networks. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 5987–5995. IEEE, 2017. , [63](#), [104](#), [118](#), [119](#), [120](#), [122](#), [123](#)
- [14] Sergey Zagoruyko and Nikos Komodakis. Wide Residual Networks. In *British Machine Vision Conference*, 2016. , [84](#), [89](#), [91](#), [98](#), [99](#), [116](#), [118](#)

- [15] David G Lowe et al. Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision*, volume 99, pages 1150–1157, 1999. [1](#)
- [16] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *IEEE International Conference on Computer Vision*, volume 1, pages 886–893. IEEE Computer Society, 2005. [1](#)
- [17] Beth Logan et al. Mel frequency cepstral coefficients for music modeling. In *International Symposium on Music Information Retrieval*, volume 270, pages 1–11, 2000. [1](#)
- [18] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015. [1](#), [5](#), [11](#)
- [19] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Re-visiting unreasonable effectiveness of data in deep learning era. In *IEEE International Conference on Computer Vision*, pages 843–852. IEEE, 2017. [2](#), [103](#)
- [20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, jun 2016. [2](#), [30](#), [55](#), [70](#), [81](#), [85](#), [89](#), [103](#), [108](#), [123](#)
- [21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics*, pages 4171–4186, 2019. [2](#)
- [22] Shizhao Sun, Wei Chen, Liwei Wang, Xiaoguang Liu, and Tie-Yan Liu. On the depth of deep neural networks: A theoretical view. In *AAAI Conference on Artificial Intelligence*, 2016. [2](#)

- [23] Siamak Ravanbakhsh, Jeff Schneider, and Barnabas Poczos. Equivariance through parameter-sharing. In *International Conference on Machine Learning*, pages 2892–2901, 2017. [2](#)
- [24] Qianli Liao and Tomaso Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *arXiv preprint arXiv:1604.03640*, 2016. [2](#)
- [25] Hecht-Nielsen. Theory of the backpropagation neural network. In *International 1989 Joint Conference on Neural Networks*, pages 593–605 vol.1, 1989. [3](#), [12](#), [13](#)
- [26] Yann LeCun. Generalization and network design strategies. *Connectionism in Perspective*, pages 143–155, 1989. [3](#), [14](#)
- [27] Alex Krizhevsky and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, pages 1–9, 2012. [11](#), [28](#), [85](#), [97](#)
- [28] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2017. [3](#), [14](#), [28](#), [36](#), [81](#), [103](#)
- [29] Zachary C. Lipton, John Berkowitz, and Charles Elkan. A Critical Review of Recurrent Neural Networks for Sequence Learning. *arXiv preprint arXiv:1506.00019*, 2015. [3](#), [19](#)
- [30] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Annual Conference of the International Speech Communication Association*, pages 1045–1048, 2010. [3](#), [19](#)

- [31] Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. Systematic evaluation of CNN advances on the ImageNet. *Computer Vision and Image Understanding*, 2017. [3](#), [29](#), [111](#)
- [32] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A Theoretical Analysis of Feature Pooling in Visual Recognition. In *International Conference on Machine Learning*, pages 111–118, 2010. [3](#), [16](#), [108](#)
- [33] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical Evaluation of Rectified Activations in Convolutional Network. In *ICML Deep Learning Workshop*, 2015. [81](#)
- [34] Francois Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2017. [3](#), [31](#), [63](#), [84](#), [104](#)
- [35] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*, 2017. [3](#)
- [36] Jan Kukačka, Vladimir Golkov, and Daniel Cremers. Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686*, 2017. [3](#)
- [37] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. Coco-stuff: Thing and stuff classes in context. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1209–1218, 2018. [4](#)
- [38] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *Advances in Neural Information Processing Systems*, pages 487–495, 2014. [4](#)
- [39] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 Million Image Database for Scene Recognition. *IEEE*

- Transactions on Pattern Analysis and Machine Intelligence*, 40(6):1452–1464, 2018. [4](#), [28](#), [29](#)
- [40] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. [11](#)
- [41] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. In *International Conference on Machine Learning*, pages 2847–2854, 2017. [11](#)
- [42] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2016. [11](#)
- [43] Ilya Sutskever, Geoffrey Hinton, Alex Krizhevsky, and Ruslan R Salakhutdinov. Dropout : A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014. [11](#), [32](#), [71](#), [73](#), [113](#), [118](#), [119](#)
- [44] Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, number 1, pages 109–111, 2013. [32](#)
- [45] Saurabh Singh, Derek Hoiem, and David Forsyth. Swapout: Learning an ensemble of deep architectures. In *Advances in Neural Information Processing Systems*, pages 28–36, 2016. [11](#), [33](#), [89](#), [114](#), [118](#), [119](#)
- [46] Michael Cogswell, Faruk Ahmed, Ross Girshick, Larry Zitnick, and Dhruv Batra. Reducing Overfitting in Deep Networks by Decorrelating Representations. *International Conference on Learning Representations*, 2016. [12](#)

-
- [47] JE Moody and SJ Hanson. A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems*, pages 1–8, 1995. [72](#), [92](#), [98](#)
- [48] Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. Regularizing Neural Networks by Penalizing Confident Output Distributions. *International Conference on Learning Representations Workshop*, 2017. [12](#)
- [49] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985. [13](#)
- [50] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep Sparse Rectifier Neural Networks. In *Journal of Machine Learning Research*, number 15, pages 315–323, 2011. [14](#)
- [51] Andrew L. Maas, Awni Y. Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*, volume 28, page 6, 2013. [14](#)
- [52] Ossama Abdel-hamid, Hui Jiang, and Gerald Penn. Applying Convolutional Neural Networks concepts to hybrid NN-HMM model for speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, number May, pages 4277–4280. IEEE, 2012. [14](#)
- [53] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. *arXiv preprint arXiv:1803.01271*, 2018. [14](#)
- [54] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014. [17](#)

- [55] Zhouhan Lin, Roland Memisevic, and Kishore Konda. How far can we go without convolution: Improving fully-connected networks. *International Conference on Learning Representations Workshop*, 2016. [18](#)
- [56] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. In *International Conference on Learning Representations*, 2016. [20](#)
- [57] Sepp Hochreiter and Jurgen Schmidhuber. Long Short-term Memory. *Neural computation*, 9(8):1735–1780, 2016. [20](#), [30](#)
- [58] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. 2014. [20](#)
- [59] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997. [21](#)
- [60] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6):602–610, 2005. [21](#)
- [61] Yin Li, Xiaodi Hou, Christof Koch, James M Rehg, and Alan L Yuille. The Secrets of Salient Object Segmentation Supplementary Materials. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 280–287, 2014. [21](#)
- [62] Ali Borji, Dicky N Sihite, and Laurent Itti. Salient Object Detection : A Benchmark. *IEEE Transactions on Image Processing*, 2015. [21](#), [50](#), [53](#)
- [63] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The PASCAL Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, 88(2):303–338, 2012. [21](#), [24](#)

- [64] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 633–641, 2017. [21](#), [130](#)
- [65] Itti Laurent, Koch Christof, and Niebur Ernst. A Model of Saliency-Based Visual Attention for Rapid Scene Analysis Laurent. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1575–1580, 1998. [22](#)
- [66] Jonathan Harel, Christof Koch, and Pietro Perona. Graph-based visual saliency. In *Advances in Neural Information Processing Systems*, pages 545–552, 2006.
- [67] Xiaodi Hou, Liqing Zhang, and Dongchuan Road. Saliency Detection: A Spectral Residual Approach. In *IEEE Conference on Computer Vision and Pattern Recognition*, number C, pages 2005–2008. IEEE, 2008.
- [68] Radhakrishna Achantay, Sheila Hemamiz, Francisco Estraday, and Sabine Süssstrunk. Frequency-tuned salient region detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, volume 2009 IEEE, pages 1597–1604. IEEE, 2009.
- [69] Tie Liu, Zejian Yuan, Jian Sun, Jingdong Wang, Nanning Zheng, Xiaoou Tang, and Heung-Yeung Shum. Learning to Detect a Salient Object. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(2):1–8, 2010.
- [70] Ming-Ming Cheng, Niloy J Mitra, Xiaolei Huang, Philip H S Torr, and Shi-Min Hu. Global Contrast Based Salient Region Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):569–582, 2015. [50](#)
- [71] Ran Margolin. What Makes a Patch Distinct ? In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1139–1146, 2013. [22](#)

-
- [72] Lingyun Zhang, Matthew H. Tong, Tim K. Marks, Honghao Shan, and Garrison W. Cottrell. SUN: A Bayesian framework for saliency using natural statistics. *Journal of Vision*, 8(7):32, 2008. [22](#)
- [73] Jimei Yang and Ming-Hsuan Yang. Top-Down Visual Saliency via Joint CRF and Dictionary Learning. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 39, pages 576–588. IEEE, 2017.
- [74] Tilke Judd, Krista Ehinger, Aude Oliva, Frédo Durand, and Antonio Torralba. Using machine learning to predict where people look. In *IEEE International Conference on Computer Vision*, number Iccv, pages 2106–2113. IEEE, 2009. [22](#)
- [75] Lijun Wang, Huchuan Lu, Xiang Ruan, and Ming-hsuan Yang. Deep Networks for Saliency Detection via Local Estimation and Global Search. In *IEEE Conference on Computer Vision and Pattern Recognition*, jun 2015. [22](#)
- [76] Eleonora Vig, Michael Dorr, and David Cox. Large-Scale Optimization of Hierarchical Features for Saliency Prediction in Natural Images. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2798–2805. IEEE, 2014. [22](#)
- [77] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015. [23](#), [36](#), [39](#), [56](#)
- [78] Feng Wang and David M. J. Tax. Survey on the attention based RNN model and its applications in computer vision. *arXiv preprint arXiv:1601.06823*, 2016. [23](#)

- [79] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent Models of Visual Attention. In *Advances in Neural Information Processing Systems*, pages 2204–2212, 2014. [23](#), [41](#)
- [80] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple Object Recognition with Visual Attention. In *International Conference on Learning Representations*, 2015. [23](#), [41](#)
- [81] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. DRAW: A Recurrent Neural Network For Image Generation. In *International Conference on Machine Learning. JMLR Workshop and Conference Proceedings*, 2015. [23](#), [41](#)
- [82] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015. [23](#), [40](#)
- [83] Shivang Agarwal, Jean Ogier Du Terrail, and Frédéric Jurie. Recent Advances in Object Detection in the Age of Deep Convolutional Neural Networks. *arXiv preprint arXiv:1809.03193*, 2018. [24](#)
- [84] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollar, and C Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *European Conference on Computer Vision*, 2016. [24](#), [57](#)
- [85] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Fei-Fei Li. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 2015. [24](#), [28](#), [29](#), [70](#), [91](#), [97](#), [111](#), [114](#), [119](#), [130](#)

- [86] Hao Su, Jia Deng, and Li Fei-Fei. Crowdsourcing Annotations for Visual Object Detection. In *AAAI Conference on Artificial Intelligence*, volume 1, pages 40–46, 2012. [24](#)
- [87] Dim P. Papadopoulos, Jasper R. R. Uijlings, Frank Keller, and Vittorio Ferrari. We don’t need no bounding-boxes: Training object class detectors using only human verification. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 854–863, 2016. [24](#)
- [88] Ross Girshick, Jeff Donahue, Trevor Darrell, U C Berkeley, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2–9, 2012. [24](#), [25](#), [27](#)
- [89] Ross Girshick. Fast R-CNN. In *IEEE International Conference on Computer Vision*, 2015. [24](#), [25](#), [65](#)
- [90] Shaoqing Ren, Kaiming He, and Ross Girshick. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *Advances in Neural Information Processing Systems*, (6):1–9, 2015. [24](#), [25](#), [27](#), [57](#), [66](#), [70](#), [71](#), [72](#)
- [91] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: Object Detection via Region-based Fully Convolutional Networks. In *Advances in neural information processing systems*, pages 379–387, 2016. [24](#), [26](#)
- [92] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single Shot Multi-Box Detector. In *European Conference on Computer Vision*, pages 21–37. Springer, 2016. [24](#), [26](#), [27](#)

- [93] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. [24](#), [26](#)
- [94] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016. [24](#), [27](#), [33](#), [58](#), [117](#)
- [95] Tsung-Yi Lin, Piotr Dollar, Ross B Girshick, Kaiming He, Bharath Hariharan, and Serge J Belongie. Feature Pyramid Networks for Object Detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2017. [24](#), [26](#), [27](#), [70](#)
- [96] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective Search for Object Recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013. [25](#)
- [97] C Lawrence Zitnick and Piotr Doll. Edge Boxes : Locating Object Proposals from Edges. In *European conference on computer vision*, pages 391–405. Springer, 2014. [25](#)
- [98] Tsung-yi Lin, Ross Girshick, and Piotr Doll. Focal Loss for Dense Object Detection. *IEEE International Conference on Computer Vision*, 2017. [26](#)
- [99] Jasper Uijlings, Stefan Popov, and Vittorio Ferrari. Revisiting knowledge transfer for training object class detectors. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018. [26](#)
- [100] Qingyi Tao, Hao Yang, and Jianfei Cai. Zero-Annotation Object Detection with Web Knowledge Transfer. In *European Conference on Computer Vision*, 2018. [27](#)

- [101] Bharat Singh, Hengduo Li, Abhishek Sharma, and Larry S. Davis. R-FCN-3000 at 30fps: Decoupling Detection and Classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1081–1090, 2018. [27](#)
- [102] Judy Hoffman, Sergio Guadarrama, Eric Tzeng, Ronghang Hu, Jeff Donahue, Ross Girshick, Trevor Darrell, and Kate Saenko. LSDA: Large Scale Detection Through Adaptation. In *Advances in Neural Information Processing Systems*, pages 3536–3544, 2014. [27](#), [71](#), [72](#), [73](#)
- [103] Yuxing Tang, Josiah Wang, Boyang Gao, Emmanuel Dellandréa, Robert Gaizauskas, and Liming Chen. Large scale semi-supervised object detection using visual and semantic knowledge transfer. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2119–2128, 2016. [58](#), [71](#), [72](#)
- [104] Yuxing Tang, Josiah Wang, Boyang Gao, Emmanuel Dellandrea, Robert Gaizauskas, and Liming Chen. Large Scale Semi-Supervised Object Detection Using Visual and Semantic Knowledge Transfer. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 40(12):2119–2128, 2016. [27](#), [58](#)
- [105] Ronghang Hu, Piotr Dollár, Kaiming He, Trevor Darrell, and Ross Girshick. Learning to Segment Every Thing. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018. [27](#), [58](#), [71](#), [72](#)
- [106] Jianxiong Xiao, Krista A. Ehinger, James Hays, Antonio Torralba, and Aude Oliva. SUN Database: Exploring a Large Collection of Scene Categories. *International Journal of Computer Vision*, 119(1):3–22, 2016. [28](#), [29](#)
- [107] Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016. [29](#)

-
- [108] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-Supervised Nets. In *Artificial Intelligence and Statistics (AISTATS)*, pages 1–10, 2015. [29](#), [30](#), [32](#)
- [109] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Inception-v1: Going deeper with convolutions. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 07-12-June, pages 1–9, 2015. [30](#), [81](#), [97](#), [113](#), [120](#)
- [110] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016. [30](#), [81](#), [97](#)
- [111] Rupesh Kumar Srivastava, Klaus Greff, and Jurgen Schmidhuber. Training Very Deep Networks. In *Advances in Neural Information Processing Systems*. 2015. [30](#), [89](#)
- [112] Klaus Greff, Rupesh K. Srivastava, and Jürgen Schmidhuber. Highway and Residual Networks learn Unrolled Iterative Estimation. In *International Conference on Learning Representations*, 2017. [30](#)
- [113] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. DenseNet: Densely Connected Convolutional Networks. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2017. [30](#), [82](#), [84](#), [88](#), [89](#), [91](#), [92](#), [93](#), [98](#), [99](#), [100](#), [108](#), [110](#), [116](#), [118](#), [123](#)
- [114] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up Convolutional Neural Networks with Low Rank Expansions. In *British Machine Vision Conference*, 2014. [31](#), [84](#), [104](#)

- [115] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014. [31](#)
- [116] Min Lin, Qiang Chen, and Shuicheng Yan. Network In Network. In *International Conference on Learning Representations*, 2014. [31](#), [84](#), [108](#)
- [117] Jonghoon Jin, Aysegul Dunder, and Eugenio Culurciello. Flattened Convolutional Neural Networks for Feedforward Acceleration. In *International Conference on Learning Representations Workshop*, 2015. [31](#), [84](#)
- [118] Sam Leroux, Steven Bohez, Tim Verbelen, Bert Vankeirsbilck, Pieter Simoens, and Bart Dhoedt. Resource-constrained classification using a cascade of neural network layers. In *International Joint Conference on Neural Networks*, volume 2015-Septe, 2015. [32](#), [106](#)
- [119] Surat Teerapittayanon, Bradley McDanel, and H T Kung. BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks. In *International Conference on Pattern Recognition*, 2016. [32](#), [117](#)
- [120] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. FractalNet: Ultra-Deep Neural Networks without Residuals. *International Conference on Learning Representations*, 2017. [32](#), [89](#), [106](#)
- [121] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Weinberger. Deep Networks with Stochastic Depth. In *European Conference on Computer Vision and Pattern Recognition*, 2016. [32](#), [89](#), [98](#), [99](#), [120](#), [123](#)
- [122] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual recognition. In *European Conference on Computer Vision*, 2014. [33](#), [111](#), [117](#)

-
- [123] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs, Atrous Convolution, and Fully Connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018. [33](#), [111](#), [117](#)
- [124] Angjoo Kanazawa, Abhishek Sharma, and David Jacobs. Locally Scale-Invariant Convolutional Neural Networks. *NIPS 2014 Workshop on Deep Learning*, pages 1–11, 2014. [33](#), [34](#)
- [125] Zhixiang Ren, Shenghua Gao, Liang-Tien Chia, and Ivor Wai-Hung Tsang. Region-Based Saliency Detection and Its Application in Object Recognition. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(5): 769–779, 2014. [35](#)
- [126] Ali Borji, Simone Frntrop, Dicky N. Sihite, and Laurent Itti. Adaptive object tracking by learning background context. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 23–30. IEEE, 2012. [35](#)
- [127] Tao Chen, Ming-ming Cheng, Ping Tan, Ariel Shamir, and Shi-min Hu. PhotoSketch : Internet Image Montage. *ACM Transactions on Graphics*, 28 (5):124, 2009. [35](#)
- [128] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *IEEE International Conference on Computer Vision*, 2015. [36](#), [81](#), [92](#)
- [129] Nian Liu, Junwei Han, Dingwen Zhang, Shifeng Wen, and Tianming Liu. Predicting Eye Fixations using Convolutional Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 362–370, jun 2015. [36](#)

- [130] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning Deconvolution Network for Semantic Segmentation. In *IEEE International Conference on Computer Vision*, 2015. [36](#), [38](#), [39](#)
- [131] Jost Tobias Springenberg. Learning to Generate Chairs with Convolutional Neural Networks ". In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5–8, 2015. [39](#), [56](#)
- [132] Elman, J. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. [42](#)
- [133] Pedro H. O. Pinheiro and Ronan Collobert. Recurrent Convolutional Neural Networks for Scene Parsing. In *International Conference on Machine Learning*, pages 82–90, 2014. [45](#)
- [134] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the Devil in the Details: Delving Deep into Convolutional Nets. In *British Machine Vision Conference*, 2014. [47](#), [55](#)
- [135] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2014. [47](#)
- [136] Tijmen Tieleman, Geoffrey E. Hinton, Nitish Srivastava, and Kevin Swersky. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012. [48](#)
- [137] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*, pages 448–456, 2015. [48](#), [63](#), [64](#), [77](#), [78](#), [109](#)

-
- [138] Simone Frintrop, Thomas Werner, and German Martin Garcia. Traditional Saliency Reloaded : A Good Old Model in New Shape. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2–6, jun 2015. [48](#)
- [139] German M. Garcia, Ekaterina Potapova, Thomas Werner, Michael Zillich, Markus Vincze, and Simone Frintrop. Saliency-based object discovery on RGB-D data with a late-fusion approach. In *IEEE International Conference on Robotics and Automation*, volume 2015-June, pages 1866–1873, 2015. [48](#)
- [140] Chuan Yang, Lihe Zhang, Huchuan Lu, Xiang Ruan, and Ming-hsuan Yang. Saliency Detection via Graph-Based Manifold Ranking. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3166–3173. IEEE, 2013. [49](#)
- [141] Ran Ju, Yang Liu, Tongwei Ren, Ling Ge, and Gangshan Wu. Depth-aware salient object detection using anisotropic center-surround difference. *Image Communication*, 38:115–126, 2015. [49](#)
- [142] Houwen Peng, Bing Li, Weihua Xiong, Weiming Hu, and Rongrong Ji. Rgb-d salient object detection: a benchmark and algorithms. In *European Conference on Computer Vision*, pages 92–109, 2014. [49](#)
- [143] Matthieu Guillaumin, Daniel Küttel, and Vittorio Ferrari. ImageNet Auto-Annotation with Segmentation Propagation. *International Journal of Computer Vision*, 110(3):328–348, 2014. [49](#)
- [144] Ming Ming Cheng, Niloy J. Mitra, Xiaolei Huang, and Shi Min Hu. SalientShape: Group saliency in image collections. *Visual Computer*, 30(4):443–453, 2014. [50](#)
- [145] S Alpert, M Galum, R Basri, and Achi Brandt. Image Segmentation by Probabilistic Bottom-Up Aggregation and Cue Integration. In *IEEE Computer*

- Society Conference on Computer Vision and Pattern Recognition*, volume 34, pages 1–8, 2007. [50](#)
- [146] Linzhao Wang, Lijun Wang, Huchuan Lu, Pingping Zhang, and Xiang Ruan. Salient object detection with recurrent fully convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(7):1734–1746, 2018. [55](#), [56](#)
- [147] Tiantian Wang, Lihe Zhang, Shuo Wang, Huchuan Lu, Gang Yang, Xiang Ruan, and Ali Borji. Detect globally, refine locally: A novel approach to saliency detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3127–3135, 2018. [55](#), [56](#)
- [148] Xiaoning Zhang, Tiantian Wang, Jinqing Qi, Huchuan Lu, and Gang Wang. Progressive attention guided recurrent network for salient object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 714–722, 2018. [55](#), [56](#)
- [149] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*, 2015. [55](#), [85](#), [109](#)
- [150] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *IEEE International Conference on Computer Vision*. IEEE, 2017. [57](#)
- [151] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. [57](#)
- [152] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Mallocci, Tom

- Duerig, and Vittorio Ferrari. The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale. *arXiv:1811.00982*, 2018. [57](#), [59](#), [69](#), [70](#), [74](#), [130](#)
- [153] Dhruv Mahajan, Ross B. Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. In *European Conference on Computer Vision*, 2018. [58](#)
- [154] Dong Li, Jia-Bin Huang, Yali Li, Shengjin Wang, and Ming-Hsuan Yang. Weakly Supervised Object Localization with Progressive Domain Adaptation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3512–3520, 2016. [58](#)
- [155] Hakan Bilen and Andrea Vedaldi. Weakly supervised deep detection networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2846–2854, 2016.
- [156] Hakan Bilen, Marco Pedersoli, and Tinne Tuytelaars. Weakly supervised object detection with convex clustering. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1081–1089, 2015. [58](#)
- [157] Ramon Lopez De Mantaras and Eva Armengol. Machine learning from examples: Inductive and lazy methods. *Data & Knowledge Engineering*, 25(1-2):99–123, 1998. [61](#)
- [158] Geoffrey E Hinton and Richard S Zemel. Autoencoders, minimum description length and helmholtz free energy. In *Advances in Neural Information Processing Systems*, pages 3–10, 1994. [65](#)

- [159] Lei Le, Andrew Patterson, and Martha White. Supervised autoencoders: Improving generalization performance with unsupervised regularizers. In *Advances in Neural Information Processing Systems*, pages 107–117, 2018. [66](#), [73](#)
- [160] Yuting Zhang, Kibok Lee, and Honglak Lee. Augmenting supervised neural networks with unsupervised objectives for large-scale image classification. In *International Conference on Machine Learning*, pages 612–621, 2016. [66](#), [73](#)
- [161] Junbo Zhao, Michael Mathieu, Ross Goroshin, and Yann Lecun. Stacked what-where auto-encoders. *International Conference on Learning Representations*, 2016. [67](#)
- [162] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, pages 3546–3554, 2015. [67](#)
- [163] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2015. [68](#)
- [164] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017. [68](#)
- [165] Jimmy Lei Ba, Kevin Swersky, Sanja Fidler, et al. Predicting deep zero-shot convolutional neural networks using textual descriptions. In *IEEE International Conference on Computer Vision*, pages 4247–4255, 2015. [68](#)
- [166] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Li Fei-Fei. Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations. *International Journal of Computer Vision*, 2017. [69](#), [74](#)

-
- [167] Ankan Bansal, Karan Sikka, Gaurav Sharma, Rama Chellappa, and Ajay Divakaran. Zero-Shot Object Detection. In *European Conference on Computer Vision*, 2018. [69](#), [71](#), [72](#)
- [168] Takuya Akiba, Tommi Kerola, Yusuke Niitani, Toru Ogawa, Shotaro Sano, and Shuji Suzuki. PFDet: 2nd Place Solution to Open Images Challenge 2018 Object Detection Track. *arXiv preprint arXiv:1809.00778*, 2018. [70](#), [73](#)
- [169] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. [71](#)
- [170] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. In *Association for Computational Linguistics Conference*, pages 427–431, 2017. [71](#), [72](#)
- [171] Stephen Merity, Bryan McCann, and Richard Socher. Revisiting activation regularization for language rnns. *arXiv preprint arXiv:1708.01009*, 2017. [71](#), [72](#)
- [172] Bingyi Kang, Zhuang Liu, Xin Wang, Fisher Yu, Jiashi Feng, and Trevor Darrell. Few-shot object detection via feature reweighting. *arXiv preprint arXiv:1812.01866*, 2018. [73](#)
- [173] Angus Galloway, Anna Golubeva, Thomas Tanay, Medhat Moussa, and Graham W Taylor. Batch Normalization is a Cause of Adversarial Vulnerability. In *International Conference on Machine Learning Workshop*, 2019. [77](#)
- [174] Y. Bengio. Learning Deep Architectures for AI. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2009. [81](#)
- [175] Jurgen Schmidhuber. Deep Learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. [81](#)

-
- [176] LeCun Y, Bottou L, Bengio Y, and Haffner P. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. [81](#)
- [177] Andy Hess. Exploring feature reuse in densenet architectures. *arXiv preprint arXiv:1806.01935*, 2018. [88](#)
- [178] Anish Shah, Eashan Kadam, Hena Shah, Sameer Shinde, and Sandip Shingade. Deep Residual Networks with Exponential Linear Unit. In *International Symposium on Computer Vision and the Internet*, pages 59–65, 2016. [89](#)
- [179] Alex Krizhevsky and Geoffrey Hinton. Learning Multiple Layers of Features from Tiny Images. Technical report, 2009. [91](#), [114](#)
- [180] Antonio Torralba, Rob Fergus, and William T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, nov 2008. [91](#)
- [181] George E. Dahl, Tara N. Sainath, and Geoffrey E. Hinton. On the Importance of Initialization and Momentum in Deep Learning. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, number 28, pages 8609–8613, 2013. [92](#), [98](#)
- [182] Andreas Veit, Michael Wilber, and Serge Belongie. Residual Networks Behave Like Ensembles of Relatively Shallow Networks. In *Advances in Neural Information Processing Systems*, 2016. [96](#)
- [183] Andrew G. Howard. Some Improvements on Deep Convolutional Neural Network Based Image Classification. *arXiv preprint arXiv:1312.5402*, 2013. [97](#)

-
- [184] Sam Gross and Michael Wilber. Training and investigating residual nets. *Facebook AI Research, CA.[Online]. Available: <http://torch.ch/blog/2016/02/04/resnets.html>*, 2016. 98
- [185] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In *AAAI Conference on Artificial Intelligence*, 2017. 103, 113
- [186] Itay Hubara, Daniel Soudry, and Ran El Yaniv. Binarized Neural Networks. In *Advances in Neural Information Processing Systems*, 2016. 103
- [187] Ali Farhadi Mohammad Rastegari, Vicente Ordonez, Joseph Redmon. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *European Conference on Computer Vision*, 2016.
- [188] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained Ternary Quantization. In *International Conference on Learning Representations*, 2017. 103
- [189] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both Weights and Connections for Efficient Neural Networks. In *Advances in Neural Information Processing Systems*, 2015. 104
- [190] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning Filters for Efficient ConvNets. In *International Conference on Learning Representations*, 2017.
- [191] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 5687–5695, 2017. 131

- [192] Yihui He, Xiangyu Zhang, and Jian Sun. Channel Pruning for Accelerating Very Deep Neural Networks. In *IEEE International Conference on Computer Vision*, volume 2017-Octob, pages 1398–1406, 2017.
- [193] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *IEEE International Conference on Computer Vision*, 2017. [104](#), [131](#)
- [194] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. *Master’s thesis, Institut für Informatik, Technische Universität, Munchen*, 91: 1–71, 1991. [108](#)
- [195] Alexandre Araujo, Benjamin Negrevergne, Yann Chevaleyre, and Jamal Atif. Training compact deep learning models for video classification using circulant matrices. In *European Conference on Computer Vision Workshops*, pages 0–0, 2018. [109](#)
- [196] Faraz Saeedan, Nicolas Weber, Michael Goesele, and Stefan Roth. Detail-preserving pooling in deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 9108–9116, 2018. [109](#)
- [197] Geoff Pleiss, Danlu Chen, Gao Huang, Tongcheng Li, Laurens van der Maaten, and Kilian Q. Weinberger. Memory-Efficient Implementation of DenseNets. *arXiv preprint arXiv:1707.06990*, 2017. [120](#), [122](#)
- [198] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry P Vetrov, and Ruslan Salakhutdinov. Spatially Adaptive Computation Time for Residual Networks. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2017. [120](#), [122](#), [130](#)
- [199] Wenming Yang, Xuechen Zhang, Yapeng Tian, Wei Wang, and Jing-Hao Xue. Deep learning for single image super-resolution: A brief review. *arXiv preprint arXiv:1808.03344*, 2018. [130](#)

-
- [200] Alex Graves. Adaptive Computation Time for Recurrent Neural Networks. *arXiv preprint arXiv:1603.08983*, 2016. [130](#)
- [201] Sebastian Ruder. An Overview of Multi-Task Learning in Deep Neural Networks. *arXiv preprint arXiv:1706.05098*, 2017. [132](#)
- [202] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarczyk, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.
- [203] Lukasz Kaiser, Aidan N Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit. One model to learn them all. *arXiv preprint arXiv:1706.05137*, 2017.
- [204] Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv:1806.08730*, 2018. [132](#)