UNIVERSITY OF CALIFORNIA,
IRVINE

Training Efficient Neural Network Models via Pruning and Knowledge Distillation

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Mathematics

by

Biao Yang

Dissertation Committee:
Professor Jack Xin, Chair
Professor Long Chen
Professor Yifeng Yu

2022

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

I would like to thank my advisor, Professor Jack Xin for his patience, profound advice, and continuous support during my Ph.D. study. He not only inspired my research interest and guided me with his immense knowledge and plentiful experience, but also encouraged me a lot in my daily life.

I would also like to thank Dr. Yingyong Qi, Dr. Shuai Zhang, and Dr. Jiancheng Lyu for their treasured support and guidance during my research. I wish to thank my collaborator Fanghui Xue. It was my great honor to work with them.

I would like to thank all my defense committee members, Professor Long Chen and Professor Yifeng Yu as well as my advancement committee members, Professor Hongkai Zhao and Professor Lizhi Sun for their time and expertise.

I would like to extend my appreciation to Donna McConnel, Aubrey Rudd, and Kate Lynch for their help with all the paperwork and registration; Professor Patrick Guidotti for being my provisional advisor; Professor Katya Krupchyk and Professor Roman Vershynin for their guidance in courses.

Finally, I would like to express my appreciation to my family and my friends. It would be impossible for me to finish my study without their sincere love and great support.

I would like to thank all my funding supports. My research was partially supported by Professor Jack Xin's NSF grant IIS-1632935, DMS-1854434, DMS-1952644, a Qualcomm Faculty Award, and Qualcomm AI Research.

Formal acknowledgments:
Portion of the text of Chapter 2 is a reprint of the material as it appears in IEEE [69]. Dr. Shuai Zhang, Dr. Jiancheng Lyu, and Dr. Yingyong Qi provided contributions to enlightening discussions and helpful suggestions. Professor Jack Xin provided the concept and directed and supervised research which forms the basis for the dissertation. The work was partially supported by NSF grant IIS-1632935, DMS-1854434, a Qualcomm Faculty Award, and Qualcomm AI Research.

Portion of the text of Chapter 3 is a reprint of the material as it appears in Springer Nature Switzerland [70]. The co-authors of the original article are acknowledged as follows. Fanghui Xue contributed to some of the experiments and the presentation of content. Dr. Shuai Zhang and Dr. Jiancheng Lyu contributed to discussions and suggestions about the project. Professor Jack Xin and Dr. Yingyong Qi directed and supervised research which forms the basis for the dissertation. The work was partially supported by NSF grants DMS-1854434, DMS-1952644, and a Qualcomm Faculty Award.

VITA

Biao Yang

EDUCATION

Doctor of Philosophy in Mathematics **2022**
University of California, Irvine *Irvine, California*

Master of Science in Applied Mathematics **2014**
State University of New York. Stony Brook *Stony Brook, New York*

Bachelor of Science in Mathematics **2013**
Nankai university *Tianjin, Tianjin*

RESEARCH EXPERIENCE

Graduate Research Assistant **2018–2022**
University of California, Irvine *Irvine, California*

TEACHING EXPERIENCE

Teaching Assistant **2018–2022**
University of California, Irvine *Irvine, California*

REFEREED CONFERENCE PUBLICATIONS

Channel Pruning for Deep Neural Networks via a Relaxed Group-wise Splitting Method **Aug 2019**
2019 Second International Conference on Artificial Intelligence for Industries

Improving Efficient Semantic Segmentation Networks by Enhancing Multi-Scale Feature Representation via Resolution Path Based Knowledge Distillation and Pixel Shuffle **Oct 2021**
16th International Symposium on Visual Computing

ABSTRACT OF THE DISSERTATION

Training Efficient Neural Network Models via Pruning and Knowledge Distillation

By

Biao Yang

Doctor of Philosophy in Mathematics

University of California, Irvine, 2022

Professor Jack Xin, Chair

A relaxed group-wise splitting method (RGSM) is developed and evaluated for channel pruning of deep neural networks. Experiments with VGG-16 and ResNet-18 architectures on CIFAR-10/100 image data show that RGSM can achieve much higher channel sparsity than group Lasso method, while keeping comparable accuracy.

Multi-resolution paths and multi-scale feature representation are key elements of semantic segmentation networks. We develop two techniques for efficient networks based on the recent FasterSeg network architecture. One is to use a state-of-the-art high resolution network (e.g. HRNet) as a teacher to distill a light weight student network. Due to dissimilar structures in the teacher and student networks, distillation is not effective to be carried out directly in a standard way. To solve this problem, we introduce a tutor network with an added high resolution path to help distill a student network which improves FasterSeg student while maintaining its parameter/FLOPs counts. The other finding is to replace standard bilinear interpolation in the upscaling module of FasterSeg student net by a depth-wise separable convolution and a Pixel Shuffle module which leads to 1.9% (1.4%) mIoU improvements on low (high) input image sizes without increasing model size.

A Fast Feature Affinity loss is developed for intermediate feature knowledge distillation. It requires less computational cost as well as storage cost. Experiments with modified Efficient-

Net architectures on CIFAR-100 data show that both Feature Affinity loss and Fast Feature Affinity loss improve the accuracy of the network and have close performance.

A compact DETR based architecture is proposed for human-only detection. By replacing the backbone of DETR with MobileNet-V3 and shrink the decoder layer, we first obtain a baseline model. Then we replace the transformer encoder with convolutional encoder. And experiments show that convolutional based encoders have better performance, but less FLOPs and parameters.

Chapter 1

Introduction

1.1 Developing Efficient Neural Network

Although the concept of backpropagation [31, 17, 41] and multilayer neural network [29] did exist over 50 years ago, it was until in 1986 when David Rumelhart, Geoffrey Hinton, and Ronald Williams sparked the interest in training neural networks by backpropagation in [57]. However limited by the computing power and the design of the networks [25, 20] during that time, most researchers shifted their interest toward other machine learning algorithms like Support Vector Machines and Random Forests. But the foundation of deep learning was still slowly built over the next decade. Many well-known neural networks were created including first backpropagation trained Convolutional neural network (CNN) [35], LeNet [36], long short-term memory recurrent neural nets (LSTMs) [26] . After 2010, with cheaper GPUs, faster computing power and larger data, it becomes possible to train deep neural networks. In 2012, the famous convolutional neural network AlexNet [34], developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [58] and led the second place over 10% accuracy. The great success of

AlexNet inspired more and more researchers to engage in deep neural network. And deep neural network (DNN) has made great achievement in many fields such as computer vision [34, 61], speech recognition [23], and natural language processing [13, 15].

Generally, a neural network can be considered as a composite function with images, videos, sound or text input and certain output based on its task. For example, the output will be the probability for each category in image classification task, a pixel-wise labeled image in semantic segmentation task, and translated text in machine translation task. Now given the neural network architecture, training data, and a task, a loss function of the network's output and the target is introduced to measure the performance of the model. Then training such a model can be treated as a minimization problem of the high dimensional non-convex loss function. Surprisingly, such complex networks are usually trained well by the simple stochastic gradient descent (SGD).

Although people have not fully understood the neural network from a theoretical perspective [38, 4, 60], it does not prevent researchers from developing many advanced neural network architectures. Nowadays, DNNs have occupied most of the leaderboards on different public datasets. From AlexNet (Parameters: 60M) [34], VGG-19 (Parameters: 144M) [61] to ResNext-101 (Parameters: 829M) [49], people proposed larger and more complicated neural networks on the journey of searching more powerful and better performance models. However, these neural networks often require higher computational cost and memory, which might result in tens of or hundreds of GPU hours training time and longer inference time. As DNN models are more exposed to personal computers and edge devices such as smartphones or IoT devices, it has become a critical issue of improving training efficiency, reducing model size and computational latency while maintaining the performance on resource limited devices.

Besides designing more efficient models like MobileNet [28], ShuffleNet [77], AutoShuffleNet [48], and EfficientNet [64], model compression has been widely studied to solve the issue in the past decades. Typically, model compression can be categorized into three methods: Quantization, Pruning, and Knowledge Distillation.

Quantization method [51, 11, 72, 30, 12] quantizes full precision floating point model weights into lower bit floating point numbers. At the lower bitwidth representation, one can directly reduce the model size and improve inference speed with the help of hardware and software accelerators such as s NVIDIA's Tensor Core [52], AMD Matrix Core [1], Intel MKL-DNN [50], and NVIDIA TensorRT [53]. Quantizing a 32-bits floating-point model to 16-bits floating point model is often considered as a "free" quantization since it directly reduces the model size by half without much loss of accuracy. And quantizing model to 8-bits or even lower is still an active topic in model compression.

In Pruning [16, 46, 68, 37, 47], the unimportant weights or structures of models are removed. Based on removing individual weights or groups of weights, pruning method is divided into unstructured pruning and structured pruning. For unstructured pruning , individual weights in the neural are removed by setting them to 0, which turns the dense tensors into sparse tensors. And with certain software, like the Neural Magic Inference Engine, one can run the pruned networks much faster. For structured pruning, group of weights are literally removed, such as channels or filters. This will result in direct compression and speedup of the models.

Knowledge distillation [24, 3, 71, 56, 66] is the process of transferring knowledge from a larger teacher model to a smaller student model. The larger teacher network usually has better performance but more computational cost and slower inference rate. And by utilizing the learning framework, one can enhance the performance of student network while keeps its efficiency.

Our work focuses on improving network efficiency via different model compression methods. We first introduce a structured pruning method via regularization. Then we introduce an intermediate knowledge distillation framework and a structural design technique for efficient networks based on a multi-resolution paths network architecture. We further improve the efficiency of the intermediate knowledge distillation method making it suitable for high resolution features. In the last chapter, we present our current work on transformer, where we design a light transformer model for human detection.

1.2 Relaxed Group-wise Splitting Method

Deep neural networks have achieved great successes in many fields in the past decades. The high performance of networks relies on their millions or even billions of parameters. In resource limited situations however, light weight networks are desirable for which pruning methods have been actively studied. One approach to reduce the redundancy of networks is the pruning method which removes parameters from an existing network.

In network pruning [46], a major line of work is on structured pruning [68] via group sparsity penalties, most notably Glasso [74]. Besides direct implementation in gradient descent [68], primal-dual like approaches can bring additional efficiency in pruning. In [76], the alternating direction method of multipliers (ADMM) is applied for unstructured weight pruning. In [16], a relaxed variable splitting method (RVSM) is proposed for unstructured sparsity and its convergence is analyzed in a regression problem. In RVSM, thresholding and gradient descent are efficiently integrated to handle non-smooth (discontinuous) penalties for network training. The RVSM is much simpler than ADMM, and so is more computationally appealing for deep network training.

In this work, we propose a Relaxed Group-wise Splitting Method (RGSM) extending the Relaxed Variable Splitting Method (RVSM)[16] to group sparsification of network weights, especially focusing on channel pruning. Channel sparsity yields highly compact network as zeroing out channels also reduce filters and network complexity. The RGSM utilizes the thresholding formulas of group-Lasso (GLasso) [74], and group-ℓ_0. Moreover, we also found that blending RGSM with the direct GLasso [68] can help zero out small weights more effectively than each individual method. Our main contributions are:

- formulation of RGSM for structured network pruning;
- general applicability of RGSM for discontinuous penalty ℓ_0 and others with closed form proximal operators;
- blending RGSM and direct GLasso [68] into an efficient group sparsity method.

1.3 Improving Efficient Semantic Segmentation Networks by Enhancing Multi-Scale Feature Representation via Resolution Path Based Knowledge Distillation and Pixel Shuffle

Semantic segmentation is concerned with pixel-wise classification of images and has been studied as a long-standing problem in computer vision, see [42, 65] and references therein. Predictions are first made at a range of scales, and are then combined with averaging/pooling or an attention layer. A class of efficient networks (called FasterSeg) have been recently constructed [8] based on differentiable neural architecture search [43] of a supernet and a subsequent knowledge distillation [24] to generate a smaller student net with 3.4M parameters and 27G FLOPs on full resolution (1024×2048) image input [1].

We are interested in distilling such a light weight Student Net from a high performance Teacher Net which we choose as HRNet-OCR [75] in this work. Our motivation is to im-

[1]This model was searched on our machine based on source code from FasterSeg [8].

prove the FasterSeg Student Net while maintaining its size and FLOPs by enhancing the resolutions of its multi-scale feature maps and their combinations for better prediction. The HRNet has about 10% higher accuracy than the FasterSeg Teacher Net on Cityscapes dataset [10]. Specifically, we first add a higher resolution path to the FasterSeg Student Net architecture and train it through distilling HRNet predictions. Then we let this high resolution path guide the prediction of the lower resolution paths in FasterSeg Student Net through a feature affinity (FA) matrix. At inference, the high resolution path is absent hence its role is virtual and does not add computational overheads on the Student Net. Though knowledge distillation at intermediate level was known in FitNets [56], the knowledge passing across multi-resolution paths for semantic segmentation appears new. In addition, we improve the inaccurate interpolation treatment in FasterSeg's feature fusion module by a combination of depth-wise separable convolutions and Pixel Shuffle (PS) technique [59], resembling the efficient operations in Shufflenets [77, 48] for regular image classification task.

Our main contributions are:

- introducing a novel *teacher-tutor-student framework* to enhance multi-resolution paths by path-wise knowledge distillation with application to Faster-Seg Student Net while keeping computational costs invariant, which utilizes the intermediate feature information from the tutor model;
- improving multi-scale feature map fusion by depth-wise separable convolution and Pixel Shuffle techniques to gain 1.9% (1.4%) validation accuracy in mIoU on low (high) resolution input images from Cityscapes dataset, at reduced computational costs.

1.4 Generalized Feature Affinity Loss

The teacher-student framework training with feature affinity loss can help smaller networks learning pixel-wise cosine similarity information of the intermediate feature maps, which has been verified in Dual Super-Resolution Learning for Semantic Segmentation [67] and Chapter 3. However, the heavy computational burden during computing the FA loss limits its application to high resolution feature maps. For feature maps of size $C \times H \times W$, the computational cost and memory space of FA loss are $O(N^2C)$ and $O(N^2)$ respectively, where $N = HW$. In [67], to reduce the high computational cost and memory overheads, they subsample the the pairs of pixels to its 1/8, which may drop some key pixels during the training.

In the corresponding chapter, we introduce an efficient Feature Affinity loss named Fast Feature Affinity (FFA) loss. Benefiting from sampling, we greatly reduce the computational cost and the memory space of FA loss to $O(NC)$ and $O(N)$. In our FFA loss, we generate a random vector in each batch during the training and compute a loss based on projected vector by the affinity matrix. Instead of explicitly calculating the affinity matrix, the FFA loss can be viewed as computing the loss along a random dimension in each batch. So the large computational cost is distributed into batches. And under expectation, we theoretically prove that our FFA loss will be equal to or close to the original FA loss. We also perform numerical experiments on modified EfficientNet [64] on CIFAR-100 [33] data, and the FFA loss has comparable performance as FA loss. Experiments on inference time indicate that FFA loss is much more computational efficient on high resolution features.

Our main contributions are:

- improving the efficiency of Feature Affinity loss by sampling, which largely reduce the computational cost and memory space needed in calculation of the loss;

- theoretically proving the FA loss and our FFA loss are equivalent under expectation, and numerically showing that they have close performance on EfficientNet on CIFAR-100 dataset.

1.5 Light DETR for Human Detection

Object detection is a computer vision technique that works to predict a set of bounding boxes along with category labels for objects of interest. The technique is widely used in tasks such as vehicle counting, face detection, face recognition, and image annotation. Methods for object detection are typically split into neural network based and non-neural network based. And modern neural network approaches can be generally categorized into two: two-stage methods and one-stage methods. In the two-stage approaches, models will first generate region proposals by select search [19, 18] or regional proposal network [55, 5]. Then a separate network will label a class or refine the region by bounding box regression for each region proposal. On the contrary, the one-stage methods will directly make predictions without pre-generated region proposals. Without region proposal stage, these networks pre-design anchors or window centers that help assign the boxes. Classical one-stage methods include YOLO [54] and SSD [44]. One-stage methods are usually faster and simpler at a potential expense of drop of the performance.

However, making predictions on a large set of proposals, anchors or window centers require postprocessing such as non-maximum suppression (NMS) to remove duplicate predictions and anchor-based coordinate decoding which makes these methods inefficient. A recent work DEtection TRansformer (DETR) [6] propose a new pipeline in object detection. DETR directly addresses the prediction problem without requirement of any postprocessing like most architectures do. The design of DETR is simple with a convolutional neural network backbone, a transformer encoder-decoder and a feed forward network head. And in 2021, Machine Learning Research at Apple introduces their compact framework HyperDETR [2] for on-device panoptic segmentation for camera. Modified on DETR model, the HyperDETR is compact and efficient enough to execute on-device with reasonable accuracy.

Inspired by all these works, we propose a compact DETR based model named Light DETR. The network is designed for human-only detection for targets less than 5. Light DETR has MobileNet-V3 as its backbone and a SOTA convolutional based encoder. We experimentally show that convolutional encoder can achieve better performance and be more efficient than original transformer encoder on the selected COCO dataset. On the human-only dataset, Light DETR with VAN block encoder has close performance, but only about 2/3 FLOPs and parameters than Light DETR with transformer encoder.

Chapter 2

Relaxed Group-wise Splitting Method

2.1 Preliminaries

In this section we first describe the structure of a convolutional neural network and the way it works. Then we further explain the concept of channel pruning.

2.1.1 Convolutional Neural Network(CNN)

Consider a simple Convolutional Neural Network(CNN) with L layers as shown in Figure 2.1. Without considering batches, the model takes an input $x^0 \in \mathbb{R}^{3\times H\times W}$. In the l-th layer the output feature map x^l is computed as

$$x^l_{i,:,:} = \delta(W^l_{i,:,:,:} \odot x^{l-1} + b^l_i),\ i = 1, 2, ..., C^l_o$$

with input feature map x^{l-1} and then the result is passed to the next layer. Here $\odot$ is the convolution operation if the layer is a convolutional layer and regular matrix multiplication is the layer is a dense layer. δ is the activation function, and a common option is the Rectified Linear Unit function $\delta(x) = ReLU(x) = \mathbb{1}_{x\geq 0}(x)$.

For a convolutional layer, the kernel weight in l-th layer W^l is of the size $C_o^l \times C_i^l \times k_h^l \times k_w^l$, here C_o^l, C_i^l are the number of output channels and input channels of l-th layer respectively, k_h^l, k_w^l are the height and width of the kernel size. b^l is the bias vector with shape C_o^l. We usually name $W_{i,:,:,:}^l$ the i-th filter and $W_{:,i,:,:}^l$ the i-th channel of the kernel in l-th layer. The convolution operation is to slide a filter $W_{i,:,:,:}^l$ across the feature map and the elementwise multiplication is performed between the filter and its mapping feature map. Then all the results are summed with the bias to give us a one-depth channel convoluted feature output. The above computation for i-th filter centering at (p, q) of the input feature map can be formularized as below.

$$\tilde{x}_{i,p,q}^l = \sum W_{i,:,:,:}^l * x_{:,p-\lfloor k_h^l/2\rfloor:p+\lfloor k_h^l/2\rfloor,q-\lfloor k_w^l/2\rfloor:q+\lfloor k_w^l/2\rfloor}^{l-1} + b_i^l$$

With repeating the process among all the C_o^l filters and stacking the output features together, we will obtain the C_o^l-depth channel output feature map before the activation function.

Let ϕ be a loss function which is used to measure the performance of neural network $h(\cdot, w)$ depending on the task and the model weights $w = \{W^l, b^l | l = 1, ..., L\}$. Now, given input data X_i with corresponding labels Y_i, $i = 1, 2, ..., m$, an empirical risk minimization problem is $\hat{f}_{X,Y}(w) = \sum_{i=1,...,m} \phi(h(X_i; w), Y_i)$. Considering the population loss function $f(w) = \mathbb{E}_{X,Y}\hat{f}_{X,Y}(w)$, with Stochastic gradient descent (SGD), the weights w can be updated by:

$$w^t = w^{t-1} - \eta \nabla f(w^t) \tag{2.1}$$

where t is the iteration number, η is the learning rate.

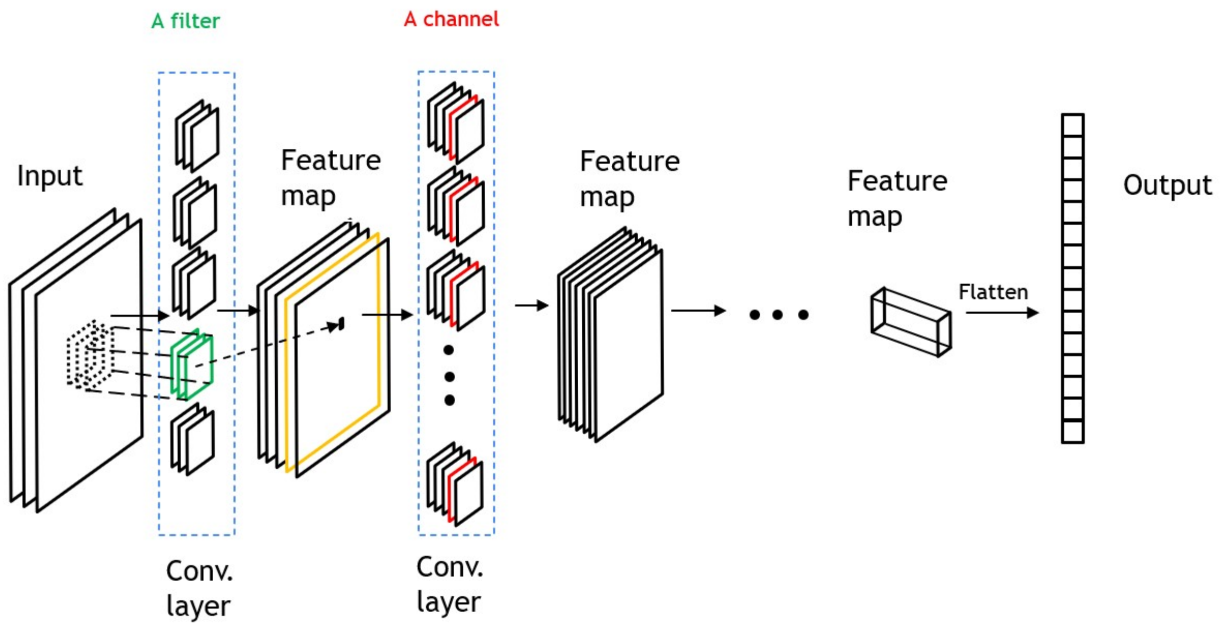


Figure 2.1: Example of a Convolutional Neural Network.

2.1.2 Channel Pruning

Channel pruning which belongs to structured pruning focuses on zeroing out channels of weights in the convolutional layers. As shown in Figure 2.1, if the channel in the second convolutional layer (red color) is set to be all zeros, then the corresponding channel (yellow color) of the input feature map for the second layer has no effect on the output result of the layer. Accordingly, the filter (green color) in the first layer which generates the channel of the input feature map for the second layer can be removed as well. So in channel pruning, removing channels can not only reduce the number of channels in the current layer, but also reduce the number of filters in the previous layer. As a result, the pruning method will lead to direct compression and speedup without dedicated hardware or libraries.

2.2 Relaxed Group-wise Splitting Method

We solve for the explicit group-wise threshholding formulas in Section 2.2.1 and then propose our Relaxed Group-wise Splitting Method in Section 2.2.2.

2.2.1 Group-wise Thresholding Formulas

Let $w = \{w_1, ..., w_g, ..., w_G\}$ be the grouped weights of convolutional layers of a deep network, where G is the total number of groups. Let I_g be the indices of w in group g. In the channel pruning case, $\{w_g : g = 1, ..., G\}$ are grouped weights of channels in convolutional layers. The GLasso penalty [74] is defined as:

$$\|w\|_{GL} := \sum_{g=1}^{G} \|w_g\|_2. \tag{2.2}$$

We obtain the GLasso proximal operator of (2.2) by solving:

$$y^* = \text{argmin}_y \lambda \, \|y\|_{GL} + \frac{1}{2}\|y - w\|_2^2, \tag{2.3}$$

or Group-wise:

$$y_g^* = \text{argmin}_{y_g} \lambda \, \|y_g\|_2 + \frac{1}{2}\sum_{i \in I_g} |y_{g,i} - w_{g,i}|^2, \; g = 1, 2, ..., G, \tag{2.4}$$

If $y_g^* \neq 0$, the objective function of (2.4) is differentiable and setting the gradient to zero gives:

$$y_{g,i} - w_{g,i} + \frac{\lambda}{\|y_g\|_2} y_{g,i} = 0, \; \forall i \in I_g$$

or

$$(1 + \frac{\lambda}{\|y_g\|_2})y_{g,i} = w_{g,i}, \ \forall i \in I_g, \tag{2.5}$$

which implying:

$$(1 + \frac{\lambda}{\|y_g\|_2})\|y_g\|_2 = \|w_g\|_2.$$

Then we should have:

$$\|y_g^*\|_2 = \|w_g\|_2 - \lambda, \text{ if } \|w_g\|_2 > \lambda, \tag{2.6}$$

Otherwise, the critical equation does not hold and $y_g^* = 0$. Then based on (2.5) and (2.6), the minimal point formula is:

$$y_{g,i}^* = \begin{cases} \frac{\|w_g\|_2 - \lambda}{\|w_g\|_2} w_{g,i} & , \text{if } \|w_g\|_2 > \lambda \\ 0 & , \textit{otherwise.} \end{cases}$$

The result can be written as a soft-thresholding operation:

$$y_g^* := \text{Prox}_{GL,\lambda}(w_g) = \frac{\max(\|w_g\|_2 - \lambda, 0)}{\|w_g\|_2} w_g = \frac{S_\lambda(\|w_g\|_2)}{\|w_g\|_2} w_g. \tag{2.7}$$

Similarly, the group-ℓ_0 penalty (G-ℓ_0) is:

$$\|w\|_{G\ell_0} := \sum_{g=1}^{G} \mathbb{1}_{\|w_g\|_2 \neq 0}. \tag{2.8}$$

In this case, to obtain the G-ℓ_0 proximal (projection) operator, equations (2.3) and (2.4) are replaced by:

$$y^* = \text{argmin}_y \lambda \, \|y\|_{G\ell_0} + \frac{1}{2}\|y - w\|_2^2, \tag{2.9}$$

and

$$y_g^* = \text{argmin}_y \lambda \, \mathbb{1}_{\|y_g\|_2 \neq 0} + \frac{1}{2}\sum_{i \in I_g} |y_{g,i} - w_{g,i}|^2, \;\; g = 1, 2, ..., G, \tag{2.10}$$

If $y_g = 0$, the objective equals $\frac{1}{2}\|w_g\|_2^2$. So if $\lambda \geq \frac{1}{2}\|w_g\|_2^2$, $y_g = 0$ is a minimal point. If $\lambda < \frac{1}{2}\|w_g\|_2^2$, $y_g = w_g$ gives minimal value λ. Thus the solution for (2.10) is the hard-thresholding operation:

$$y_g^* := \text{Prox}_{G\ell_0,\lambda}(w_g) = w_g \mathbb{1}_{\|w_g\|_2 > \sqrt{2\lambda}}. \tag{2.11}$$

2.2.2 Relaxed Group-wise Splitting Method (RGSM)

To sparsify network weights channel-wise, we add group sparsity to the population loss function f. Consider the minimization problem:

$$l(w) = f(w) + \lambda\|w\|. \tag{2.12}$$

where $\|w\|$ is the GLasso or G-ℓ_0 regularization. Inspired by the unstructured sparsifying method, Relaxed Variable Splitting Method [16], we first relaxed (2.12) into an equation of two variables

$$l(w, u) = f(w) + \lambda\|u\|.$$

and consider its augmented Lagrangian

$$\mathcal{L}_\beta(w,u) = f(w) + \lambda\|u\| + \frac{\beta}{2}\|w-u\|_2^2. \tag{2.13}$$

To minimize (2.13), we alternately update w and u. Noticing that $\mathcal{L}_\beta(w,u)$ is differentiable in w, we simply use SGD to update w. For the update on u, it is equivalent to minimize $\lambda\|u\| + \frac{\beta}{2}\|w-u\|_2^2$ which has solved in Section 2.2.1. Now we have extended RVSM [16] to our RGSM by modifying gradient descent (2.1) into:

$$\begin{aligned} u_g^t &= \mathrm{Prox}_{\lambda/\beta}(w_g^t), \quad g = 1, \cdots, G, & (2.14)\\ w^{t+1} &= w^t - \eta\,\nabla f(w^t) - \eta\,\beta\,(w^t - u^t), & (2.15) \end{aligned}$$

Here $\mathrm{Prox}_{\lambda/\beta}(\cdot)$ is one of the two Group-wise thresholding operators in (2.7) or (2.11). λ and β are two positive hyperparameters.

In the experiments, we find that blending our RGSM with GLasso [68] can imrpove the performance as well as the stability of training. Let $\lambda_1 = \lambda$ and λ_2 be the GLasso *blending parameter*. So the general RGSM is summarized in Algorithm 1. If $\lambda_2 = 0$, Algorithm 1 is called Relaxed Group-wise Splitting method (RGSM). And if $\lambda_2 \neq 0$, it becomes Relaxed Group-wise Splitting method blending with GLasso, (RGSM+GL) for short. The RGSM can be RGSM(GL) or RGSM(G-ℓ_0) depending on using (2.2) or (2.8). The output u^t ($t = max\_epoch$) gives the pruned weights. The $\{w^t\}$'s are auxiliary weights to help compute $\{u^t\}$'s.

Algorithm 1 Relaxed Group-wise Splitting Method

```
Input: $\beta, \lambda_1, \lambda_2, max_{epoch}, max_{batch}$
Output: $u^{max_{epoch}}$
Define: objective function $\rho(w) = l(w) + \lambda_2 \|w\|_{GL}$
Initialize: $w^0$
Define: $u^0$
for $g = 1, 2, ..., G$ do
    $u_g^0 = \mathrm{Prox}_{\lambda_1}(w_g^0)$
end
for $t = 1, 2, ..., max_{epoch}$ do
    for $batch = 1, 2, ..., max_{batch}$ do
        $w^{t+1} \leftarrow w^t - \eta \nabla \rho(w^t) - \eta \beta (w^t - u^t)$
        for $g = 1, 2, ..., G$ do
            $u_g^{t+1} \leftarrow \mathrm{Prox}_{\lambda_1}(w_g^t)$
        end
    end
end
```

2.3 Experiments and Results

We compare Algorithm 1 with GLasso [68] on CIFAR-10 [32] dataset through VGG-16[61] and ResNet-18 [22], and on CIFAR-100 [33] through ResNet-18. In training, λ_1 controls the threshold, and is found to be larger for RGSM(G-ℓ_0) to be effective.

2.3.1 VGG-16 on CIFAR-10

We train VGG-16 model in 200 epochs, and use SGD as optimizer with momentum 0.9, weight decay 5e-4 and initial learning rate 0.1. The learning rate decays by a factor of 0.1 at the 100th and 160th epochs. We apply Algorithm 1 to pruning convolutional layers of the model. The sparsity is measured as the percentage of all channels with ℓ_2-norm less than 10^{-15}. Table 2.1 shows that both RGSM(GL) and blended RGSM(GL) with Glasso (GL) achieve higher channel sparsity than GL while maintaining the original network accuracy. And Figure 2.2 shows the number of channels of each layer in VGG-16 trained on CIFAR-10

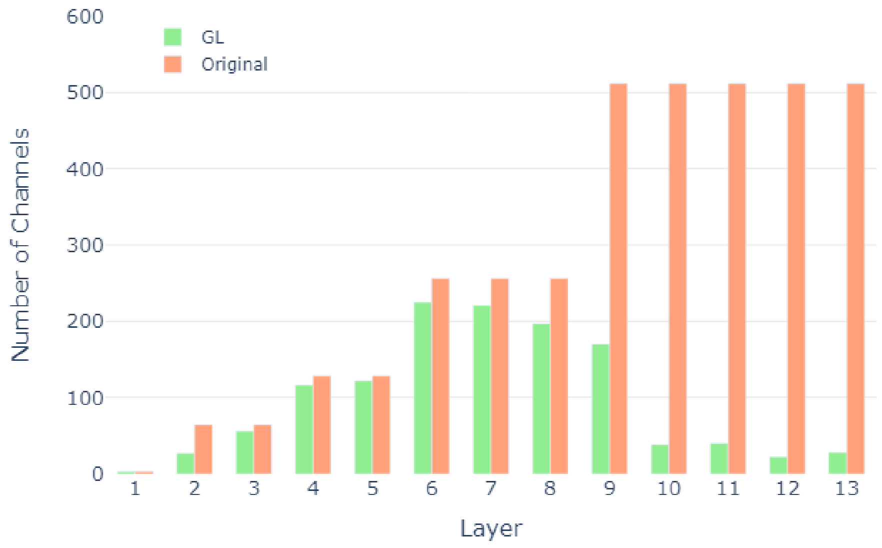
Table 2.1: Accuracy (Acc. %) and Sparsity (Sp. %) of VGG-16 on CIFAR-10.

Model	$\boldsymbol{\beta}$	$\boldsymbol{\lambda_1}$	$\boldsymbol{\lambda_2}$	***Acc.***	***Channel Sp.***	***Weight Sp.***
Original	0	0	0	93.94	0	0
GL	0	0	1e-4	93.62	65.9	74.1
RGSM(GL)	1	1e-3	0	93.68	69.0	77.7
RGSM(GL)+GL	1	1e-3	1e-6	93.61	70.1	78.8
RGSM(G-ℓ_0)	1	4e-2	0	93.77	67.8	76.6
RGSM(G-ℓ_0)+GL	1	4e-2	1e-6	93.64	70.1	78.9

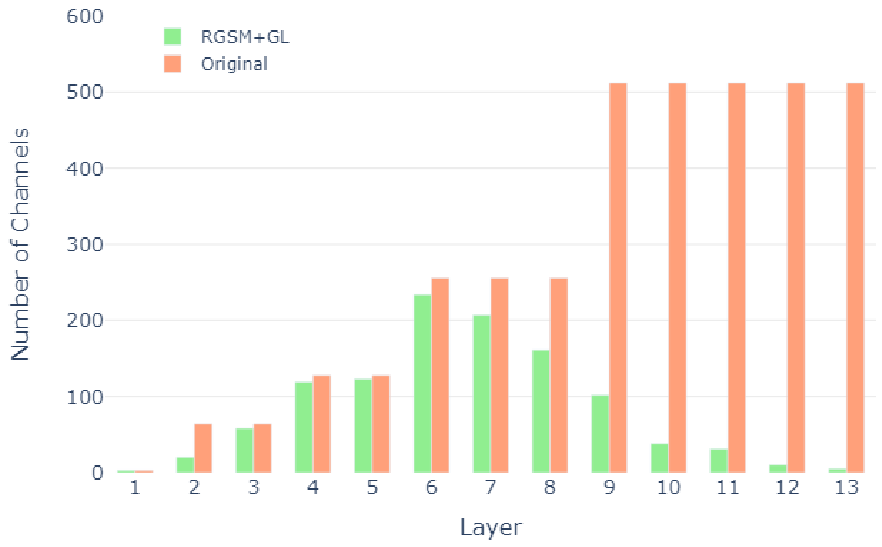
with only GL and our blended RGSM(GL) with GL. Both methods tend to prune channels at the last several layers, where these layers tend to extract high level feature information.

2.3.2 ResNet-18 on CIFAR-10 & CIFAR-100

We implement Algorithm 1 on CIFAR-10 and CIFAR-100 with ResNet-18 under the same training condition as VGG-16. In Table 2.2 and 2.3 , the blended RGSM(G-ℓ_0) and GL garners the highest sparsity under 1% loss of the original accuracy. This can be explained by the observation: while the splitting procedure zeros out channels with ℓ_2-norm under certain threshold, the blended GLasso helps promote channel differences so more channels with small ℓ_2-norm appear. Figure 2.3 and Figure 2.4 show the number of channels of each layer in ResNet-18 trained on CIFAR-10 and CIFAR-100 with only GL and our blended RGSM(GL) with GL. Most pruned channels appear at the last several layers as experiment results for VGG-16. Surprisingly on CIFAR-10, our RGSM(GL)+GL prunes all channels at the last 2 layers, which means convolutional layers are not necessary in these layers.

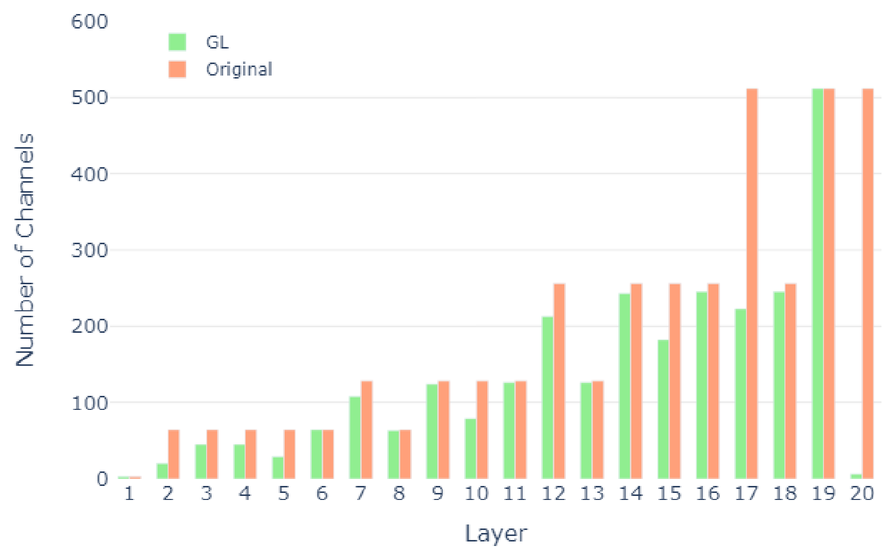


(a) Before and after GL pruning on CIFAR-10.

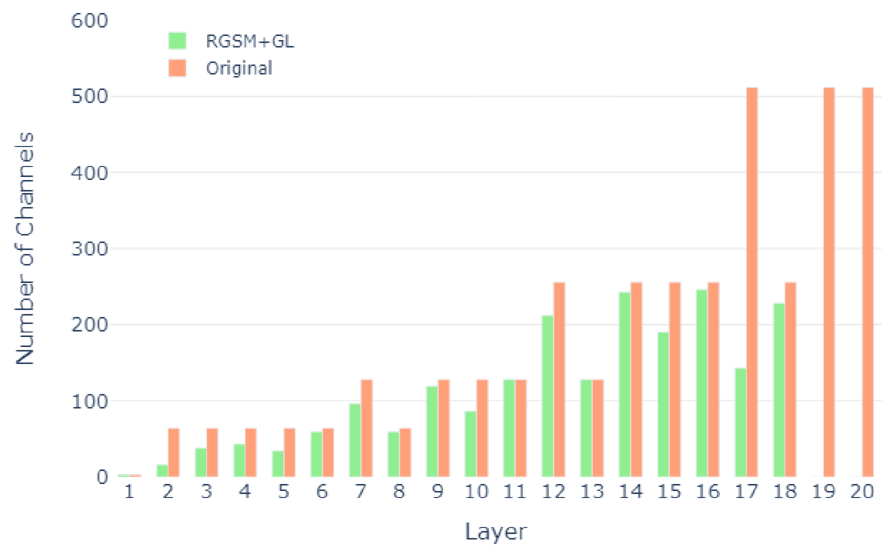


(b) Before and after RGSM(GL)+GL pruning on CIFAR-10.

Figure 2.2: Layer-wise channel numbers in VGG-16 before and after pruning on CIFAR-10.

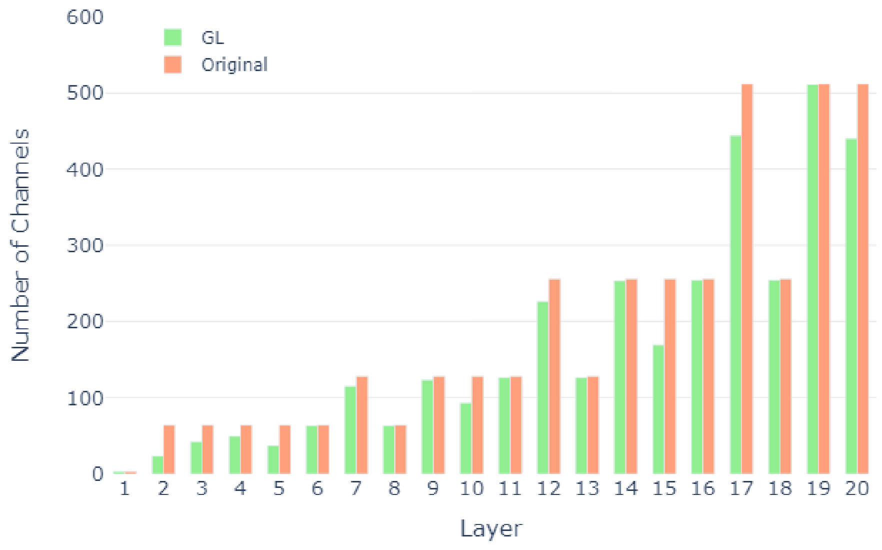


(a) Before and after GL pruning on CIFAR-10.

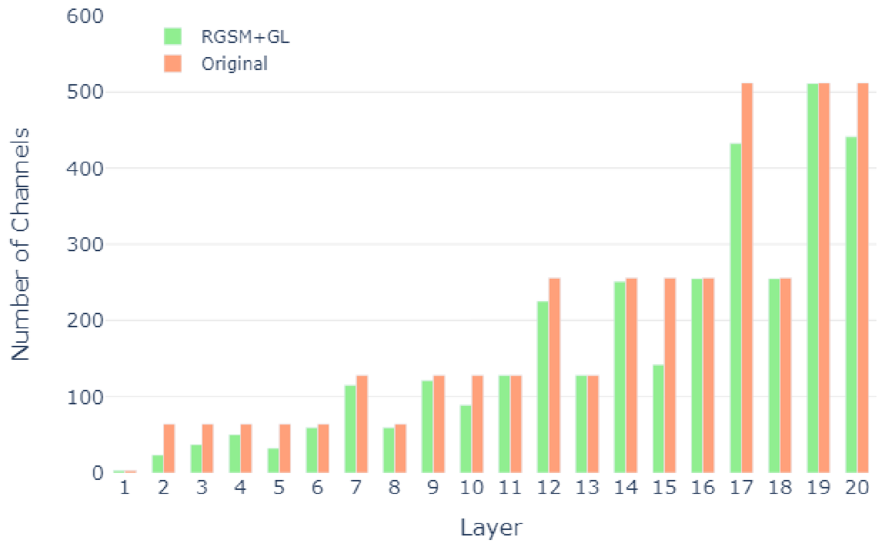


(b) Before and after RGSM(GL)+GL pruning on CIFAR-10.

Figure 2.3: Layer-wise channel numbers in ResNet-18 before and after pruning on CIFAR-10.



(a) Before and after GL pruning on CIFAR-100.



(b) Before and after RGSM(GL)+GL pruning on CIFAR-100.

Figure 2.4: Layer-wise channel numbers in ResNet-18 before and after pruning on CIFAR-100.

Table 2.2: Accuracy (Acc. %) and Sparsity (Sp. %) of ResNet-18 on CIFAR-10.

Model	$\boldsymbol{\beta}$	$\boldsymbol{\lambda_1}$	$\boldsymbol{\lambda_2}$	***Acc.***	***Channel Sp.***	***Weight Sp.***
Original	0	0	0	94.97	0	0
GL	0	0	1e-4	95.13	29.7	37.4
RGSM(GL)	1	1e-3	0	94.74	45.8	61.4
RGSM(GL)+GL	1	1e-3	5e-6	94.74	46.1	62.1
RGSM(G-ℓ_0)	1	1e-2	0	95.19	35.9	44.3
RGSM(G-ℓ_0)+GL	1	1e-3	5e-6	94.87	49.7	62.8

Table 2.3: Accuracy (Acc. %) and Sparsity (Sp. %) of ResNet-18 on CIFAR-100.

Model	$\boldsymbol{\beta}$	$\boldsymbol{\lambda_1}$	$\boldsymbol{\lambda_2}$	***Acc.***	***Channel Sp.***	***Weight Sp.***
Original	0	0	0	77.76	0	0
GL	0	0	1e-4	77.52	11.2	9.5
RGSM(GL)	1	1e-3	0	77.03	11.1	9.5
RGSM(GL)+GL	1	1e-3	5e-6	77.47	12.7	10.7
RGSM(G-ℓ_0)	0.1	5e-2	0	76.93	19.7	15.3
RGSM(G-ℓ_0)+GL	0.1	5e-2	1e-6	76.88	20.3	16.6

2.4 Conclusion

We propose the Relaxed Group-wise Splitting Method (RGSM) which is developed for structured channel pruning. It outperforms GLasso in the number of pruned channels while maintaining network accuracy. The blended ℓ_0-version, viz. RGSM(G-ℓ_0)+GL, achieve most channel sparsity while keeping loss of accuracy under one percent for pruning ResNet-18 on both CIFAR-10 and CIFAR-100. The blending of group-wise splitting and GLasso is found to be effective. While splitting zeros out channels with ℓ_2-norm under certain threshold, GLasso helps create channel differences.

Chapter 3

Improving Efficient Semantic Segmentation Networks by Enhancing Multi-Scale Feature Representation via Resolution Path Based Knowledge Distillation and Pixel Shuffle

3.1 Overview

Semantic segmentation has been studied for decades. Recent lines of research include hierarchical architecture search, knowledge distillation (introduced in [24] for standard classification), and two-stream methods. Among large capacity models are Autodeeplabs ([42] and references therein), high resolution net (HRNet [75]), zigzag net [39] and hierarchical multi-scale attention network [65]. Among the light weight models are FasterSeg [8], and BiSenet

[73]. In Gated-SCNN [63], a high level stream on region masks guides the low level stream on shape features for better segmentation. A gated structure connects the intermediate layers of the two streams, and resulted in 2% mask (mIoU) gain over DeepLabV3+ [7] on Cityscapes dataset [10]. In [67], a dual super-resolution learning framework is introduced to produce high resolution representation on low resolution input. A 2% gain in mIoU over various baseline models is accomplished on Cityscapes data. A feature affinity function is used to promote cooperation of the two super-resolution networks, one on semantic segmentation, the other on single image super-resolution.

Though knowledge distillation at an intermediate level [56] has been known conceptually, how to set it up in the multi-resolution paths for semantic segmentation networks is not much studied. In part, a choice of corresponding locations in the Teacher Net and Student Net depends on network architecture. This is what we set out to do on FasterSeg Student Net.

Besides the conventional upscaling methods like bilinear interpolation, Pixel Shuffle (PS) is widely adopted in various multi-resolution image processing tasks. In the super-resolution task [59], an artful PS operator has been applied to the output of the convolutional layer in the low resolution, and hence has reduced the computational complexity. This technique is inherited by [67] for the super-resolution semantic segmentation, boosting the performance of the model with low resolution input.

3.2 Search and Training in FasterSeg

The FasterSeg search space consists of multi-resolution branches with searchable down-sampling-path from high resolution to low resolution, and searchable operations in the cells (layers) of the branches. Each cell (layer) contains 5 operations: skip connect, 3×3 Conv,

3×3 Conv $\times 2$, Zoomed 3×3 Conv and Zoomed 3×3 Conv $\times 2$. The "Zoomed Conv" contains bilinear down-sampling, 3×3 Conv and bi-linear up-sampling.

Below we recall FasterSeg's architecture search and training procedure [8] in order to introduce our proposed path-wise distillation. In the search stage, the overall optimization objective is:

$$L = L_{seg}(M) + \lambda \, Lat(M) \tag{3.1}$$

where $Lat(M)$ is the latency loss of the supernet M; L_{seg} is the supernet loss containing cross-entropy of logits and targets from different branches. Note that the branches come from resolutions: 1/8, 1/16, 1/32, 1/8+1/32 and 1/16+1/32, as well as losses from different expansion ratios (max, min, random and architecture parameter ratio).

The architecture parameters (α, β, γ) are in a differentiable computation graph, and optimized by gradient descent. The α is for operations in each cell, β for down-sampling weight, and γ for expansion ratio. Following [43], the training dataset is randomly half-split into two disjoint sets Train-1 and Train-2. Then the search follows the first order DARTS [43]:

1) Update network weights W on Train-1 by gradient: $\nabla_w L_{seg}(M|W, \alpha, \beta, \gamma)$.
2) Update architecture α, β, γ on Train-2 by gradient:

$$\nabla_{\alpha,\beta,\gamma} L_{seg}(M|W, \alpha, \beta, \gamma) + \lambda \cdot \nabla_{\alpha,\beta,\gamma} LAT(M|W, \alpha, \beta, \gamma).$$

For teacher-student co-searching with two sets of architectures (α_T, β_T) and $(\alpha_S, \beta_S, \gamma_S)$, the first order DARTS [43] becomes:

1) Update network weights W by $\nabla_w L_{seg}(M|W, \alpha_T, \beta_T)$ on Train-1,
2) Update network weights W by $\nabla_w L_{seg}(M|W, \alpha_S, \beta_S, \gamma_S)$ on Train-1,

3) Update architecture α_T, β_T by $\nabla_{\alpha,\beta,\gamma} L_{seg}(M|W, \alpha_T, \beta_T)$ on Train-2,

4) Update architecture α_S, β_S, γ_S by $\nabla_{\alpha,\beta,\gamma} L_{seg}(M|W, \alpha_S, \beta_S, \gamma_S)$
$+\lambda \nabla_{\alpha,\beta,\gamma} LAT(M|W, \alpha_S, \beta_S, \gamma_S)$ on Train-2.

The next step is to train the weights to obtain final models.

Step 1): train Teacher Net by cross-entropy to compute the losses between logits of different resolutions and targets.

$$loss_T = CE(pred8_T, target) + \lambda CE(pred16_T, target) + \lambda CE(pred32_T, target) \quad (3.2)$$

where $predn_T$ is the prediction of the $1/n$ resolution path of the Teacher Net.

Step 2): train Student Net with an extra distillation loss between logits of resolution 1/8 from Teacher Net and logits of resolution 1/8 from Student Net.

$$\begin{aligned} Loss_S &= CE(pred8_S, target) + \lambda CE(pred16_S, target) \\ &+\lambda CE(pred32_S, target) + KL(pred8_T, pred8_S). \end{aligned} \quad (3.3)$$

Here $predn_S$ is the prediction of $1/n$ resolution path of the Student Net.

3.3 Resolution Path Based Distillation

In Section 3.3.1, we first show how to build a FasterSeg Student Net with 1/4 resolution path. Then we introduce feature affinity loss in Section 3.3.2, with search and training details in Section 3.3.3.

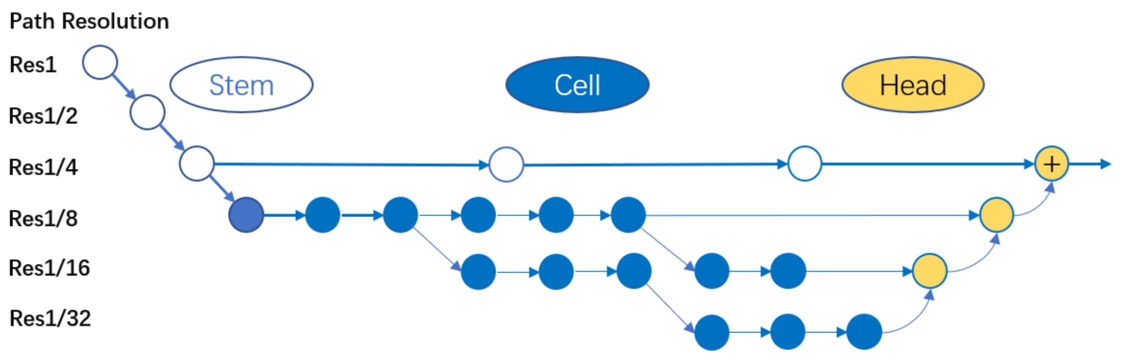


Figure 3.1: Tutor net: a FasterSeg form of Student Net with 1/4 resolution path added.

3.3.1 Tutor Model with 1/4 Resolution Path

We build a FasterSeg Student Net ("student tutor") with 1/4 resolution path, see in Figure 3.1. The 1/4 resolution path follows the 1/4 resolution stem and contains 2 basic residual $2\times$ layers also used in other stems. The 1/4 path then merges with the 1/8 resolution path by interpolating output of the 1/8 resolution path to the 1/4 resolution size, refining and adding. The model is trained with HRNet-OCR as the Teacher Net.

In Figure 3.1, the stem is a layer made up of Conv and Batch normalization and ReLU. The cell is mentioned in Section 3.2. And the head is to fuse outputs of different resolution paths together. In 1/4 path, its output is directly added to the output of 1/8 path. While for the other paths, the outputs are concatenated with up-sampled output of lower resolution paths, and then go through a 3×3 Conv.

3.3.2 Feature Affinity Loss

Feature Affinity (FA) loss [67] is a measure to overcome dimension inconsistency in comparing two feature maps by aggregating information from all channels. Consider a feature map with dimensions (H, W, C), which contains $H \times W$ vectors (in pixel direction) of length C. A pixel of a feature map F is denoted by F_i, where $1 \leq i \leq H \times W$. The affinity of two pixels is reflected in the affinity matrix with entries being the pairwise normalized inner product:

$$S_{ij} = (\frac{F_i}{\|F_i\|_p})^T \cdot (\frac{F_j}{\|F_j\|_p}). \tag{3.4}$$

In other words, the affinity matrix consists of cosine similarities of all pixel-pairs. In order to make sure their consistency in the spacial dimension, we need to interpolate the student affinity matrix to the same dimensional size of the teacher affinity matrix. Let the two affinity matrices for the Teacher and Student Nets be denoted by S^t_{ij} and S^s_{ij}. Then the Feature Affinity (FA) loss is defined as [67]:

$$L_{fa} = \frac{1}{H^2W^2}\|S^t - S^s\|_q. \tag{3.5}$$

where q is not necessarily the dual of p, and the norm is entry-wise q-norm. Here we choose $p = 2, q = 1$, and consider adding the FA loss (3.5) to the distillation objective for gradient descent.

Let $S^{1/n}$ be the output of the $1/n$ resolution path of Student Net after passing to ConvNorm layers, $T^{1/n}$ be the output of the $1/n$ resolution path of Teacher Net. We introduce a path-wise FA loss as:

$$FA\_loss = L_{fa}(S^{1/8}, T^{1/4}) + \lambda L_{fa}(S^{1/16}, T^{1/16}) + \lambda L_{fa}(S^{1/32}, T^{1/32}), \tag{3.6}$$

where λ balances the FA losses on different paths. In our experiment, $\lambda = 0.8$ is chosen to weigh a little more on the first term for the 1/8 path of the student net to mimic the 1/4

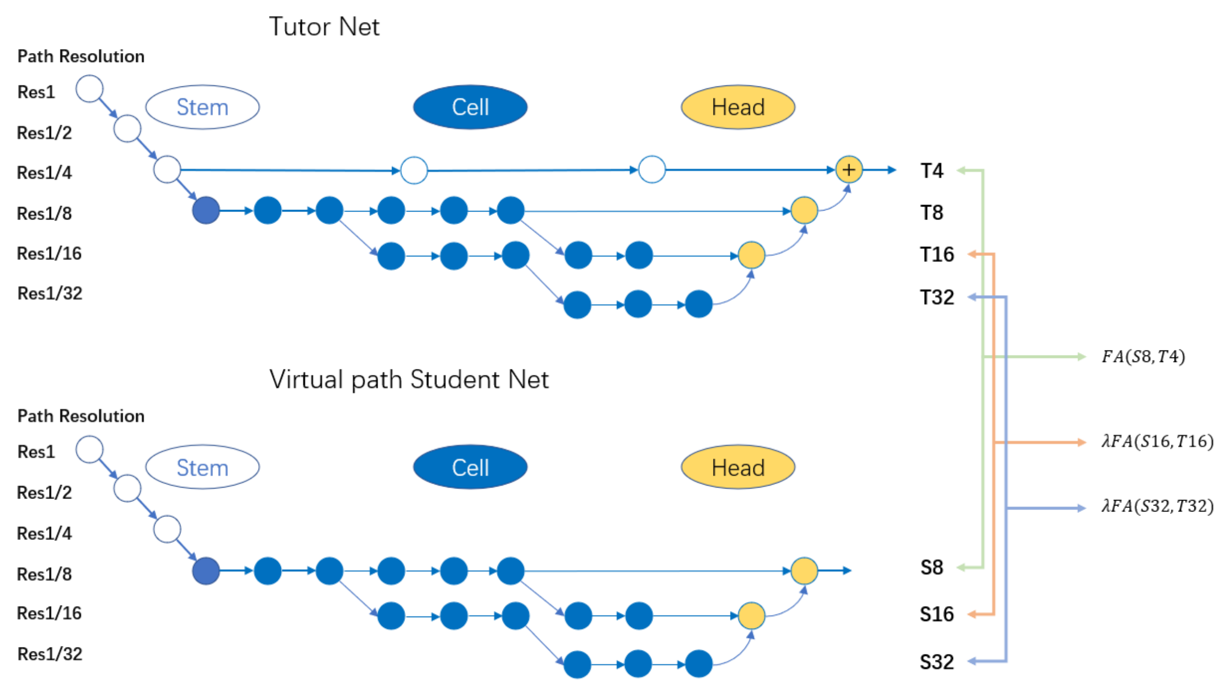


Figure 3.2: Student Net with virtual 1/4 path (lower) distilled from tutor net (upper).

path of the tutor net. Figure 3.2 illustrates how our path-wise FA loss is constructed for distillation learning in the training process.

3.3.3 Teacher Net Guided Student Net Search and Training

We summarize our search and training steps below.

Search:

FasterSeg searches its own teacher model, which takes time and may not be most ideal. Instead, we opt for a state-of-the-art model as teacher to guide the search of a light weight Student Net. In our experiments, the Teacher Net is HRNet-OCR [65]. To shorten inference time, we set it back to the original HRNet [62]. The search objective is:

$$L = L_{seg}(M) + \lambda_1 \, Lat(M) + \lambda_2 \, Dist(M, HR). \tag{3.7}$$

The $L_{seg}(M)$ and $Lat(M)$ are same as in Equation (3.1) of FasterSeg, with the added third term to narrow the distance between Student Net and the Teacher Net HR.

By first order DARTS [43] on the randomly half split training datasets (Train-1 and Train-2), we have:

1) Update network weights W by $\nabla_w L_{seg}(M|W, \alpha, \beta, \gamma)$ on Train-1

2) Update architecture α, β, γ by $\nabla_{\alpha,\beta,\gamma} L(M|W, \alpha, \beta, \gamma)$ on Train-2.

Training:

1) The baseline model is a FasterSeg student model distilled from HRNet. The training objective is:

$$\begin{aligned} Loss_S &= CE(pred8_S, target) + \lambda CE(pred16_S, target) \\ &+ \lambda CE(pred32_S, target) + KL(pred_{HR}, pred8_S). \end{aligned} \tag{3.8}$$

Here $pred_{HR}$ is the prediction of the HRNet and $predn_S$ is the same notation as in FasterSeg training.

2) For the Student Net with virtual 1/4 path, we first train a FasterSeg Student Net with additional 1/4 resolution path as in Section 3.3.1. This more accurate yet also heavier temporary Student Net serves as a “tutor” for the final Student Net.

Similar to FasterSeg, we only use the outputs of the 1/4, 1/16 and 1/32 resolution paths. The output of the 1/8 path is fused with that of the 1/4 path for computational efficiency and memory savings. The resulting loss is:

$$\begin{aligned} Loss_{Tu} &= CE(pred4_{Tu}, target) + \lambda CE(pred16_{Tu}, target) \\ &+ \lambda CE(pred32_{Tu}, target) + KL(pred_{HR}, pred4_{Tu}). \end{aligned} \tag{3.9}$$

Here Tu stands for the "tutor" model with 1/4 resolution path.

Next we distill the true Student Net from the "tutor net" by minimizing the the loss function:

$$\begin{aligned} Loss_S &= cFA\_loss + CE(pred8_S, target) + \lambda CE(pred16_S, target) \\ &+ \lambda CE(pred32_S, target) + KL(pred4_{Tu}, pred8_S), \end{aligned} \tag{3.10}$$

where $predn_{Tu}$ is the prediction of the tutor model.

3.4 Experiment Results

In Section 3.4.1, we introduce the dataset and our computing environment. We present experimental results and analysis in Section 3.4.2, and analyze FA loss in Section 3.4.3.

3.4.1 Dataset and Implementations

We use the Cityscapes [10] dataset for training and validation. There are 2975 images for training, 500 images for evaluation, and 1525 images for testing. The class mIoU (mean Intersection over Union) is the accuracy metric.

Our experiments are conducted on Quadro RTX 8000. Our environment is CUDA 11.2 and CUDNN 7.6.5, implemented on Pytorch.

3.4.2 Experimental Results and Analysis

We perform our method on both low resolution (256×512) input and high resolution (512×1024) input. The different resolution here means the cropping size of the raw image input. The evaluation is on the original image resolution of 1024×2048. And we use only the train dataset for training and perform validation on validation (Val) dataset.

During the search process, we set pretrain epochs as 20, number of epochs as 30, batch size as 6, learning rate as 0.01, weight decay as 5×10^{-4}, initial latency weight λ_1 as 10^{-2}, distillation coefficient λ_2 as 1.

We first train a baseline student model with total epochs as 500, batch size as 10, learning rate as 0.012, learning rate decay as 0.990, weight decay as 10^{-3}. The baseline model is comparable to FasterSeg's student [8] in performance.

Then we train our 1/4 path tutor model for low (high) resolution input as a student initialized from the baseline model above with HRNet as teacher. The total number of epochs is 350 (400 for high), batch size is 10, learning rate is 0.0026 (0.003 for high), learning rate decay is 0.99, and weight decay is 10^{-3}.

Finally, we distill the student model with virtual 1/4 path from the 1/4 path tutor model with 400 epochs, batch size 10, learning rate 0.003, learning rate decay 0.99, weight decay 10^{-3} and coefficient c for FA loss as 1.

The results are in Table 3.1 where the baseline Student Net has 60.1% (71.1%) mIoU for low (high) resolution input. The tutor net with 1/4 path has 63.6% (73.03%) mIoU for low (high) resolution input. The Student Net with virtual 1/4 path increases the accuracy of the baseline Student Net to 62.2% (72.3%) mIoU for low (high) resolution input. While we have trained the Student Net for different input sizes (256×512 and 512×1024), the inferences are made on the full resolution (1024×2048). For all of our experiments, FLOPs is also computed on the full resolution (1024×2048) images, regardless of the training input size. The improvement by the student net with virtual 1/4 path over the baseline student net are illustrated through images in Figure 3.3. In the rectangular regions marked by dashed red lines, more pixels are correctly labeled by the student net with virtual 1/4 path. On test dataset, the baseline Student Net has 57.7% (69.3%) mIoU for low (high) resolution input, and the virtual 1/4 path Student Net has 60.2% (69.7%) mIoU for low (high) resolution input.

We show ablation experimental results for low resolution input in Table 3.2. It contains student nets with virtual 1/4 path trained from scratch and different coefficient c settings for the FA loss in Equation (3.10). Both fine tuning and FA loss contribute to the improvement of the accuracy.

To further understand our teacher-tutor-student distillation framework, we studied direct student net distillation from HRNet with the help of FA loss. However, the mIoU is lower than that from the above teacher-tutor-student distillation framework. This might be due to the more disparate architectural structures between the teacher model and the FasterSeg form of the student net. We notice that our improvement is lower for the high resolution input. This may be due to reaching the maximal capability of the student net. In our experiment, we only have 2 layers in the 1/4 resolution path of the student net. For more gain in mIoU, adding more layers in the path is a viable approach to be explored in the future.

Table 3.1: Tutor/student net with virtual 1/4 path for low/high image input sizes on Cityscapes Val dataset.

Input size	256×512	512×1024	Param/FLOPs
baseline student net	60.1	71.1	3.4M/27G
tutor net w. 1/4-path	63.6	73.0	3.9M/100G
student net (w. virtual 1/4-path)	62.2	72.3	3.4M/27G

Table 3.2: Student nets with virtual 1/4 path on low input size images of Val dataset.

Input size	256×512
baseline student net	60.1
student net (w. virtual 1/4-path) from scratch	59.9
student net (w. virtual 1/4-path) $c = 0$	60.9
student net (w. virtual 1/4-path) $c = 1$	62.2

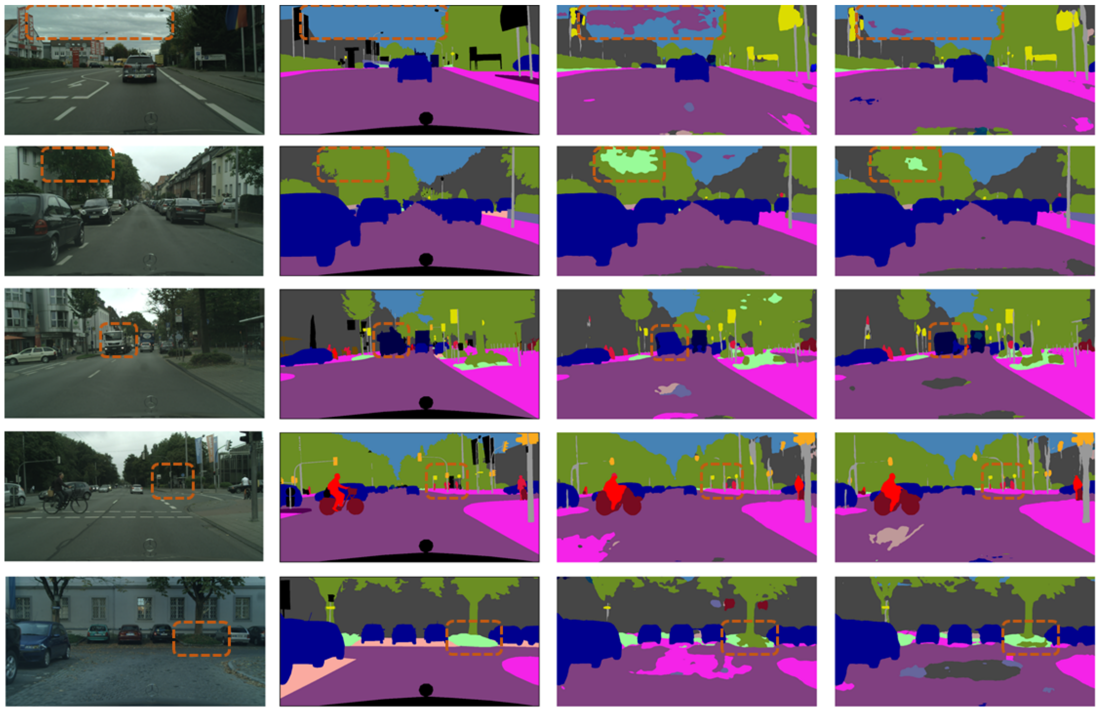


Figure 3.3: Baseline student net improved by student net with virtual 1/4-path (pixels in rectangular regions). The 4 columns (left to right) display input images, true labels, output labels of the baseline net and the student net with virtual 1/4-path resp. The 5 example images are taken from the validation dataset.

3.4.3 Analysis of FA loss

In [67], the authors added a ConvNorm layer right after the extraction of feature maps to adjust their distributions before computing FA loss. We found that it would be better not add extra layers to both the tutor and student models. Adding such a ConvNorm Layer to the student model would be enough. By checking the affinity matrices with and without ConvNorm layers, we observed that given the tutor model, adding such extra layers to both student and tutor models before computing FA loss forces the entries of affinity matrices all close to 1 (a trivial way to reduce FA loss). This phenomenon is illustrated in Figure 3.4. And we also find that enlarging the coefficient c in FA loss will help the model converge faster but might cause overfitting.

Theoretically, given F_i and F_j which are two pixels of feature map F, if a ConvNorm layer is applied to F before passing to Equation (3.4) with $p = 2$, we have

$$S_{ij} = (\frac{g(F_i)}{\|g(F_i)\|_2})^T \cdot (\frac{g(F_j)}{\|g(F_j)\|_2}). \tag{3.11}$$

where the g is the ConvNorm layer which contains a 1×1 Convolutional layer, a BatchNorm layer and a ReLU layer. For a vector $(x \in \mathbb{R}^C) \sim \mathcal{D}$,

$$g(x) = \delta\big(\frac{(Ax + b) - \mathbb{E}x}{var(x)}\gamma + \beta\big) \tag{3.12}$$

with learnable parameters $A \in \mathbb{R}^{C \times C}$, $b \in \mathbb{R}^C$, and $\gamma, \beta \in \mathbb{R}$, and $\delta(\cdot)$ is the ReLU activation function. Assume that we have $F_i, F_j \overset{iid}{\sim} \mathcal{N}(\mu, \sigma^2 I)$, then we show that there would be trivial choices of learnable parameters in g such that the S_{ij} in Equation (3.11) would be close to 1 in high probability.

Now consider an all-ones matrix $A = (1)_{C \times C}$, $b = \mu$, and $\gamma = \sigma^2$, then for $x \sim \mathcal{N}(\mu, \sigma^2 I)$

Equation (3.12) becomes

$$\begin{aligned} g(x) &= \delta\big(\frac{(Ax+b)-\mathbb{E}x}{var(x)}\gamma+\beta\big) \\ &= \delta\big(\frac{(Ax+\mu)-\mu}{\sigma^2}\sigma^2+\beta\big) \\ &= \delta\big(Ax+\beta\big) \\ &= \delta\big((\sum_k x_k+\beta)1_C\big) \end{aligned}$$

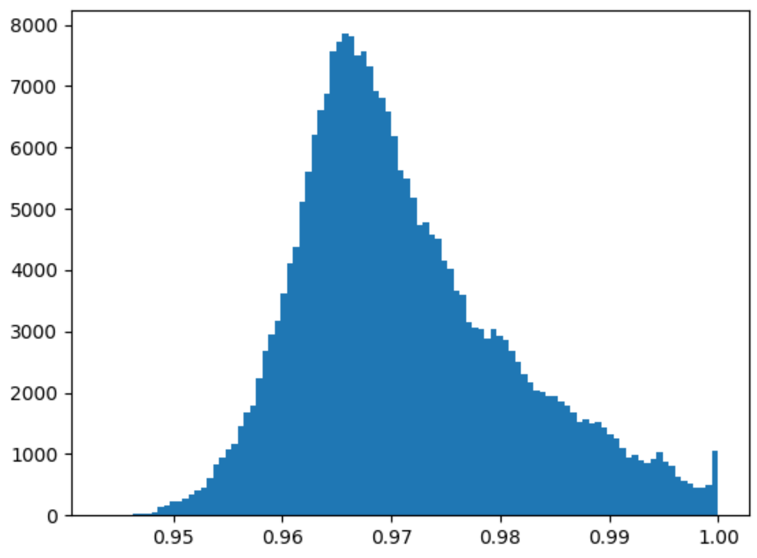
where 1_C is a all-ones vector of length C. Noting that $(\sum_k x_k+\beta)\sim\mathcal{N}(C\mu+\beta, C\sigma^2)$, by choosing a large enough β, with high probability we have $(\sum_k x_k+\beta)>0$ and so is $g(x)>0$. In Equation (3.11), $g(F_i)=\delta\big((\sum_k F_{ik}+\beta)1_C\big)$ and $g(F_j)=\delta\big((\sum_k F_{jk}+\beta)1_C\big)$ are both positive vector with high probability, hence

$$S_{ij}=(\frac{(\sum_k F_{ik}+\beta)1_C}{\|(\sum_k F_{ik}+\beta)1_C\|_2})^T\cdot(\frac{(\sum_k F_{jk}+\beta)1_C}{\|(\sum_k F_{jk}+\beta)1_C\|_2})=1$$

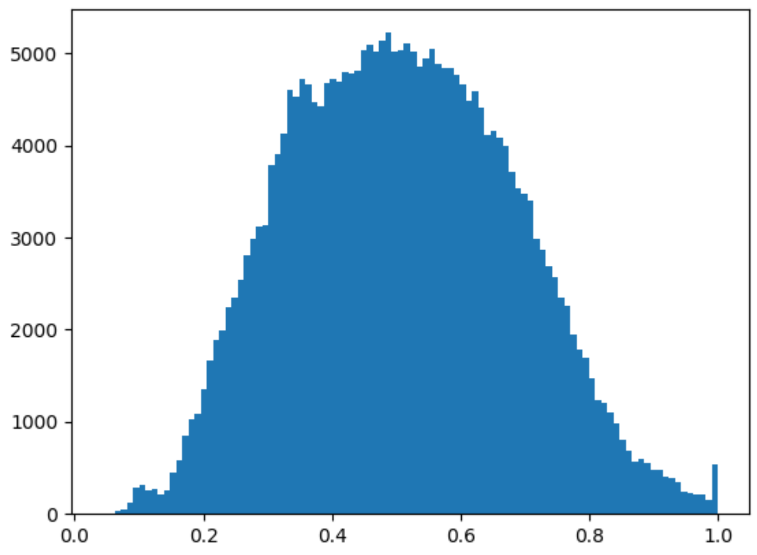
with high probability.

In the case that $F_i, F_j \overset{iid}{\sim} \mathcal{D}$ with $\mathcal{D}$ being a positive distribution, the same strategy still works. And by choosing a all-ones matrix $A=(1)_{C\times C}$, $b=\mathbb{E}F_i$, $\beta>0$, and $\gamma=var(F_i)$, then $g(F_i)=\delta\big((\sum_k F_{ik}+1)1_C\big)=(\sum_k F_{ik}+1)1_C$ and $g(F_j)=\delta\big((\sum_k F_{jk}+1)1_C\big)=(\sum_k F_{jk}+1)1_C$. Then similiar argument will guarantee $S_{ij}=1$.

Thus adding ConvNorm layers to both teacher and the student nets before computing FA loss might not help to transfer knowledge as there are trivial settings of ConvNorm which forces affinity scores S_{ij} of both nets all close to 1 independently of the models.



(a) Histogram of S_{ij} with ConvNorm layers added to feature maps prior to computing FA loss from Equation (3.10).



(b) Histogram of S_{ij} from Equation (3.10).

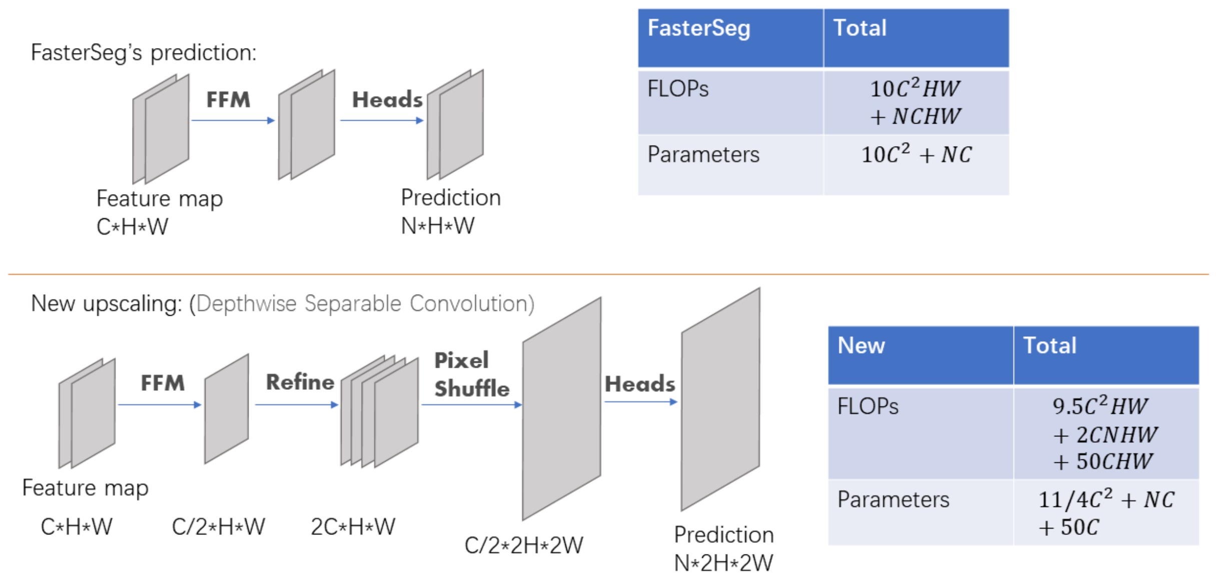
Figure 3.4: Histograms of entries in affinity matrix entries in tutor-student distillation learning.

3.5 Pixel Shuffle Prediction Module

In this section, we develop a specific technique for FasterSeg student prediction module, see Figure 3.5. FasterSeg [8] uses Feature Fusion Module (FFM) to fuse two feature maps from different branches, with the outcome of size $C \times H \times W$, which is passed through the Head Module to generate the prediction map of size $19 \times H \times W$. Afterward, a direct interpolation by a factor of 8 up-scales the prediction map to the original input size. This treatment makes FasterSeg fast however at an expense of accuracy.

We discovered an efficient improvement by generating a larger prediction map without introducing too many parameters and operations. The idea is to adopt depth-wise separable convolution [9] to replace certain convolution operations in FasterSeg. First, we reduce the channel number of FFM by half and then use Refine Module (group-wise convolution) to raise the channel number. Finally, we apply Pixel Shuffle on the feature map to give us a feature map of size $C/2 \times 2H \times 2W$. Let us estimate the parameters and operations of FFM and Heads focusing on the convolution operations. The FFM is a 1×1 convolution with C^2 parameters and C^2HW operations. The Head contains a 3×3 Conv and a 1×1 Conv, whose parameters are $9\,C^2 + C\,N$ and the operations are $9\,C^2HW + CNHW$. In total, there are $10\,C^2HW + NCHW$ and $10\,C^2 + NC$ operations. Similarly in our structure, FFM consumes $C^2/2$ parameters and $(C^2/2)HW$ operations, Refine costs $50C$ parameters and $50CHW$ operations, the Head takes $9\,(C/2)^2 + (C/2)N$ parameters and $9(C/2)^2 2H2W + (C/2)N2H2W$ operations. In total, there are $(11/4)\,C^2 + NC/2 + 50\,C$ parameters and $9.5\,C^2HW + 2\,CNHW + 50\,CHW$ operations. If C is large enough, our proposed structure has fewer parameters and operations to produce a larger prediction map.

In our experiments, we replace the Prediction Module of FasterSeg model with our Pixel Shuffle Prediction Module, keeping all the other choices in training the same. As shown in Table 3.3, our proposed model has 0.1 MB fewer parameters, yet has achieved 1.9 % (1.4 %)



FasterSeg	Total
FLOPs	$10C^2HW + NCHW$
Parameters	$10C^2 + NC$

New	Total
FLOPs	$9.5C^2HW + 2CNHW + 50CHW$
Parameters	$11/4C^2 + NC + 50C$

Figure 3.5: Proposed Pixel Shuffle Prediction Module vs. that of FasterSeg [8].

Table 3.3: Comparison of validation mIoUs of our proposed up-scaling method with depth-wise separable convolution (dep. sep. conv) and Pixel Shuffle (PS) on low/high input image sizes vs. those of the FasterSeg student (original net) [8] implemented on our local machine.

Input size	256×512	512×1024	Param/FLOPs
original net	60.1	71.1	3.4 MB/27 GB
our net with dep. sep. conv & PS	62.0	72.5	3.3 MB/27 GB

mIoU improvement for low (high) resolution input images.

3.6 Conclusion

We present a teacher-tutor-student resolution path based knowledge distillation framework and apply it to FasterSeg Student Net [8] guided by HRnet. While preserving parameter sizes and FLOPs counts, our method improves mIoU by 2.1% (1.2%) on low (high) input image sizes on Cityscapes dataset. We design a depth-wise separable convolution and Pixel Shuffle technique in the resolution upscaling module which improves FastserSeg's Student Net by 1.9% (1.4%) on low (high) input image sizes with slightly lower (same) parameter

(FLOPs) count.

Chapter 4

Generalized Feature Affinity Loss

4.1 Overview

In Equation (3.4), the affinity matrices of the teacher and the student nets' are computed by pixel-wise cosine similarity distance. And minimizing the Feature Affinity (FA) loss [67] which measures the difference between two nets' affinity matrices transfers pixel-wise relation from the teacher net to the student net. And in Section 3.4.3, we analyze that adding ConvNorm layers to both teacher and student nets before computing FA loss might not help to transfer knowledge.

In this Chapter, we will first discuss about the computational cost of FA loss in Section 4.2, and then propose a method to reduce the computational cost named as Fast Feature Affinity (FFA) loss in Section 4.3. And the numerical experiment results are in Section 4.4.

4.2 Computational Cost of Feature Affinity Loss

We currently ignore the batch size and consider the FA loss for two feature map with same dimensions (H, W, C). In Equation (3.4), the feature maps are reshaped to matrices of size $HW \times C := N \times C$. The matrices are normalized in row dimension which costs $O(NC)$ operations and then two affinity matrices are generated by standard matrix multiplication of the normalized matrices and their transposes respectively with $O(N^2C)$ operations needed. In Equation (3.5), the average of 1-norm of the difference of two $N \times N$ matrices are calculated, thus there are $O(N^2)$ operations needed. So in all, the number of total operations for FA loss is $O(N^2C)$. And $N \times N$ memory space is needed to store the matrices.

The number of operations and memory space grows quadratically as total number of pixels in the feature map. If FA loss is implemented on lower level feature maps of a neural network with a high resolution image input, the computational cost of FA loss would be beyond tolerance.

4.3 Fast Feature Affinity Loss

To reduce the computational and storage cost of FA loss, we propose a Fast Feature Affinity Loss, which requires only $O(NC)$ operations and $O(N)$ memory space.

Consider reshaped feature maps F^1 and F^2 of size $HW \times C = N \times C$. Let $\tilde{F^1}$ and $\tilde{F^2}$ be normalized F^1 and F^2 with 2-norm respectively, i.e. $\tilde{F_i^1} = F_i^1/\|F_i^1\|_2$ and $\tilde{F_i^2} = F_i^2/\|F_i^2\|_2$ for $i = 1, 2, ..., N$. Then by Equation (3.4) and (3.5), we have the original FA loss to be

$$L_{fa}(F1, F2; q) = \|\tilde{F^1}\tilde{F^1}^T - \tilde{F^2}\tilde{F^2}^T\|_q. \tag{4.1}$$

Here we currently do not consider the scaling coefficient. Now if let $A = \tilde{F^1}\tilde{F^1}^T$ and $B =$

$\tilde{F^2}\tilde{F^2}^T$, Equation (4.1) can be written as

$$L_{fa}(F1, F2; q) = \|A - B\|_q. \tag{4.2}$$

4.3.1 Fast Feature Affinity Loss

We introduce a random vector $x \sim \mathcal{N}(0, I_N)$, then we define our Fast Feature Affinity Loss as

$$\begin{aligned} L_{ffa}(F1, F2; q) &= \|(\tilde{F^1}\tilde{F^1}^T - \tilde{F^2}\tilde{F^2}^T)x\|_q \\ &= \|(A - B)x\|_q. \end{aligned} \tag{4.3}$$

And during the training, the x is randomly generated in each batch.

Since $(\tilde{F^1}\tilde{F^1}^T - \tilde{F^2}\tilde{F^2}^T)x = \tilde{F^1}(\tilde{F^1}^T x) - \tilde{F^2}(\tilde{F^2}^T x)$, if we first compute the terms in the parentheses on the right hand side of the equation, then the computational cost for this step is only $O(NC)$. And memory space is $O(N)$ as only vectors of length at most N are needed to store the intermediate results.

4.3.2 Analysis of Fast Feature Affinity Loss

We will show that Fast Feature Affinity Loss and Feature Affinity Loss are equivalent by taking expectation. Hence by taking enough samples of x, Fast Feature Affinity Loss will have comparable performance to Feature Affinity Loss.

Case: q=2

Noting that for the entry-wise norm and a matrix M with number of total entries m, we should have $\sqrt{m}\|M\|_2 \geq \|M\|_1 \geq \|M\|_2$. When m is fixed, minimizing $\|M\|_2$ can also result in minimizing $\|M\|_1$. So we can consider minimizing FA loss with $q = 2$.

When $q = 2$, the original FA loss is

$$\begin{aligned} L_{fa}(F1, F2; 2) &= \|A - B\|_2 \\ &= \sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N} |a_{ij} - b_{ij}|^2}. \end{aligned} \tag{4.4}$$

And

$$\begin{aligned} L_{fa}^2(F1, F2; 2) &= \|A - B\|_2^2 \\ &= \sum_{i=1}^{N}\sum_{j=1}^{N} |a_{ij} - b_{ij}|^2. \end{aligned} \tag{4.5}$$

Now for our L_{ffa} in Equation (4.3), the equation becomes

$$\begin{aligned} L_{ffa}(F1, F2; 2) &= \|(A - B)x\|_2 \\ &= \sqrt{\sum_{i=1}^{N}(\sum_{j=1}^{N} |a_{ij} - b_{ij}|x_j)^2}. \end{aligned} \tag{4.6}$$

And

$$\begin{aligned}
L^2_{ffa}(F1,F2;2) &= \|(A-B)x\|_2^2 \\
&= \sum_{i=1}^{N}(\sum_{j=1}^{N}|a_{ij}-b_{ij}|x_j)^2. \qquad (4.7)
\end{aligned}$$

If we take the expectation on x for Equation (4.7), we have

$$\begin{aligned}
\mathbb{E}_x L^2_{ffa}(F1,F2;2) &= \mathbb{E}_x \sum_{i=1}^{N}(\sum_{j=1}^{N}|a_{ij}-b_{ij}|x_j)^2 \\
&= \mathbb{E}_x \sum_{i=1}^{N}(\sum_{j=1}^{N}|a_{ij}-b_{ij}|^2 x_j^2 + 2\sum_{j\neq k}|a_{ij}-b_{ij}||a_{ik}-b_{ik}|x_j x_k) \\
&= \mathbb{E}_x \sum_{i=1}^{N}\sum_{j=1}^{N}|a_{ij}-b_{ij}|^2 x_j^2 + 2\sum_{i=1}^{N}\sum_{j\neq k}|a_{ij}-b_{ij}||a_{ik}-b_{ik}|x_j x_k \\
&= \sum_{i=1}^{N}\sum_{j=1}^{N}|a_{ij}-b_{ij}|^2 \mathbb{E}_x x_j^2 + 2\sum_{i=1}^{N}\sum_{j\neq k}|a_{ij}-b_{ij}||a_{ik}-b_{ik}|\mathbb{E}_x x_j x_k \\
&= \sum_{i=1}^{N}\sum_{j=1}^{N}|a_{ij}-b_{ij}|^2 = L^2_{fa}(F_1,F_2;2). \qquad (4.8)
\end{aligned}$$

Thus FFA loss is the same as FA loss under the expectation with $q=2$.

Case: q=1

When $q = 1$, the original FA loss is

$$\begin{aligned} L_{fa}(F1, F2; 1) &= \|A - B\|_1 \\ &= \sum_{i=1}^{N} \sum_{j=1}^{N} |a_{ij} - b_{ij}|. \end{aligned} \tag{4.9}$$

Now for our L_{ffa} in Equation (4.3), the equation becomes

$$\begin{aligned} L_{ffa}(F1, F2; 1) &= \|(A - B)x\|_1 \\ &= \sum_{i=1}^{N} \Big| \sum_{j=1}^{N} (a_{ij} - b_{ij}) x_j \Big|. \end{aligned} \tag{4.10}$$

If we take the expectation on x for Equation (4.10), we have

$$\begin{aligned} \mathbb{E}_x L_{ffa}(F1, F2; 1) &= \mathbb{E}_x \sum_{i=1}^{N} \Big| \sum_{j=1}^{N} (a_{ij} - b_{ij}) x_j \Big| \\ &= \sum_{i=1}^{N} \mathbb{E}_x \Big| \sum_{j=1}^{N} (a_{ij} - b_{ij}) x_j \Big|. \end{aligned} \tag{4.11}$$

Since $x \sim \mathcal{N}(0, I_N)$, we have $\sum_{j=1}^{N} (a_{ij} - b_{ij}) x_j \sim \mathcal{N}(0, \sum_{j=1}^{N} (a_{ij} - b_{ij})^2)$. Given a normal distributed random variable $Y \sim \mathcal{N}(0, \sigma^2)$, we have $\mathbb{E}|Y| = \sigma\sqrt{\frac{2}{\pi}}$, where $|Y|$ is also known as having a folded normal distribution. So Equation (4.11) equals

$$\begin{aligned}\mathbb{E}_x L_{ffa}(F1, F2; 1) &= \sum_{i=1}^{N} \mathbb{E}_x \Big| \sum_{j=1}^{N} (a_{ij} - b_{ij}) x_j \Big| \\ &= \sum_{i=1}^{N} \left(\sqrt{\sum_{j=1}^{N} (a_{ij} - b_{ij})^2} \sqrt{\frac{2}{\pi}} \right) \\ &= \sqrt{\frac{2}{\pi}} \sum_{i=1}^{N} \sqrt{\sum_{j=1}^{N} (a_{ij} - b_{ij})^2}. \end{aligned} \tag{4.12}$$

The expectation can be viewed as a norm of blending 1-norm with 2-norm on $A - B$. Thus FFA loss with $q = 1$ is close to FA loss with $q = 1$.

4.4 Numerical Experiments

We conduct our experiments on CIFAR-100 [33] dataset with modified EfficientNet [64]. We adopt ArcFace loss [14] and Label Refinery [3] in training the model. The batch size is 128, and the total number of epochs is 70. We choose a simple learning rate strategy, by setting learning rate to be 0.07 at epochs 1-25, 0.007 at epochs 26-50, 0.0007 at epochs 51-65 and 0.00007 at epochs 66-70. Our environment is CUDA 11.2 and CUDNN 7.6.5, implemented on Pytorch with Quadro RTX 8000.

We first train a EfficientNet-B3 model. Then we use the model as the teacher net to train a smaller network EfficientNet-B0 by minimizing KL-divergence between B3's and B0's output logits as our benchmark. Next besides keeping the KL-divergence loss, we add FA or FFA loss on the intermediate features from the B3 and B0 models to the final loss function in the training. Here we do not add extra layers before implementing the FA or FFA loss as the models only differ in the size and task is simpler. The FA or FFA loss is added before the

MBConvBlock with output channel number of 112, output spatial size of 16×16.

In our experiments, we use the FA loss and FFA loss with different scaling coefficient as below

$$L_{fa}(F1, F2) = \begin{cases} \frac{1}{N^2} \|\tilde{F^1}\tilde{F^1}^T - \tilde{F^2}\tilde{F^2}^T\|_1, \text{ if } q = 1 \\ \frac{1}{N} \|\tilde{F^1}\tilde{F^1}^T - \tilde{F^2}\tilde{F^2}^T\|_2, \text{ if } q = 2 \end{cases}$$

and

$$L_{ffa}(F1, F2) = \begin{cases} \frac{1}{N^2} \|(\tilde{F^1}\tilde{F^1}^T - \tilde{F^2}\tilde{F^2}^T)x\|_1, \text{ if } q = 1 \\ \frac{1}{N} \|(\tilde{F^1}\tilde{F^1}^T - \tilde{F^2}\tilde{F^2}^T)x\|_2, \text{ if } q = 2 \end{cases}$$

And Table 4.1 shows that nets trained with FA loss or FFA loss outperform nets trained without these intermediate feature information over 1.0% on average. And our FFA loss has similar performance as the original FA loss in both $q = 1$ and $q = 2$ cases.

We theoretically analyze the FLOPs of both extra losses in our case with $q = 1$. For FA loss, the estimated FLOPs in feature normalizing is $2 \times 3 \times 112 \times 16^2 = 172,032$, in computing affinity matrices is $2 \times 2 \times 16^2 \times 16^2 \times 112 = 29,360,128$, and in calculating the final loss is $2 \times 16^2 \times 16^2 = 131,072$. So the total FLOPs for FA loss is around $29,663,232$. Similarly, for FFA loss, the estimated FLOPs in feature normalizing remains the same, the estimated FLOPs in computing the products is about $2 \times 2 \times 16^2 \times 112 + 2 \times 2 \times 16^2 \times 112 = 229,376$, and the FLOPs in computing the final loss is $2 \times 16^2 = 512$. Thus the total FLOPs for FFA loss is approximately $401,920$. The FLOPs needed in FFA loss is much less than the FLOPs in FA loss.

We also measure these two losses' inference time in a separate experiment in the same environment. By fixing the batch size to be 10 and the channel number to be 100, we generate features of size $H \times H$ with H ranging from 10 to 1000 of step size 10. And for

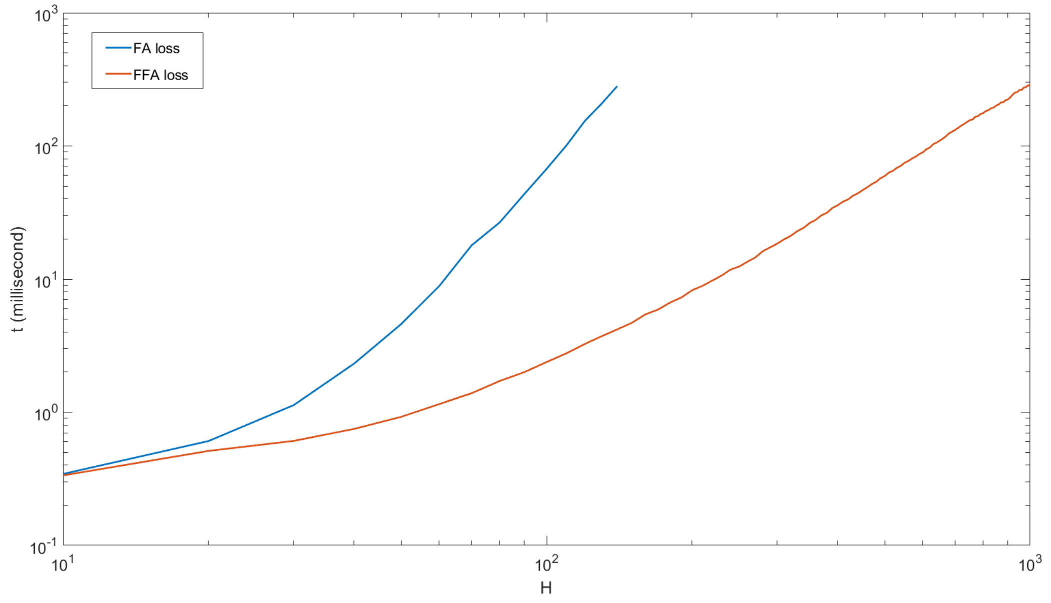
Models	Exp1 (%)	Exp2 (%)	Exp3 (%)	Ave (%)	Param/FLOPs
B3	80.24	80.24	80.24	80.24	10.5M/0.97G
B0	75.62	76.00	76.02	75.88	4.1M/0.38G
B0+FA($q = 1$)	77.17	77.56	77.61	77.45	4.1M/0.38G
B0+FFA($q = 1$)	77.32	77.52	77.63	77.49	4.1M/0.38G
B0+FA($q = 2$)	76.78	77.54	77.72	77.35	4.1M/0.38G
B0+FFA($q = 2$)	77.27	77.29	77.31	77.29	4.1M/0.38G

Table 4.1: EfficientNet on Cifar-100 dataset with KL-divergence and with/without FA loss or FFA loss. B3 (B0) stands for modified EfficientNet-B3 (B0). FA (FFA) means FA (FFA) loss is applied in the training. And Column 2-4 contain 3 independent experiments' results and Column 5 lists the average accuracy for each method.

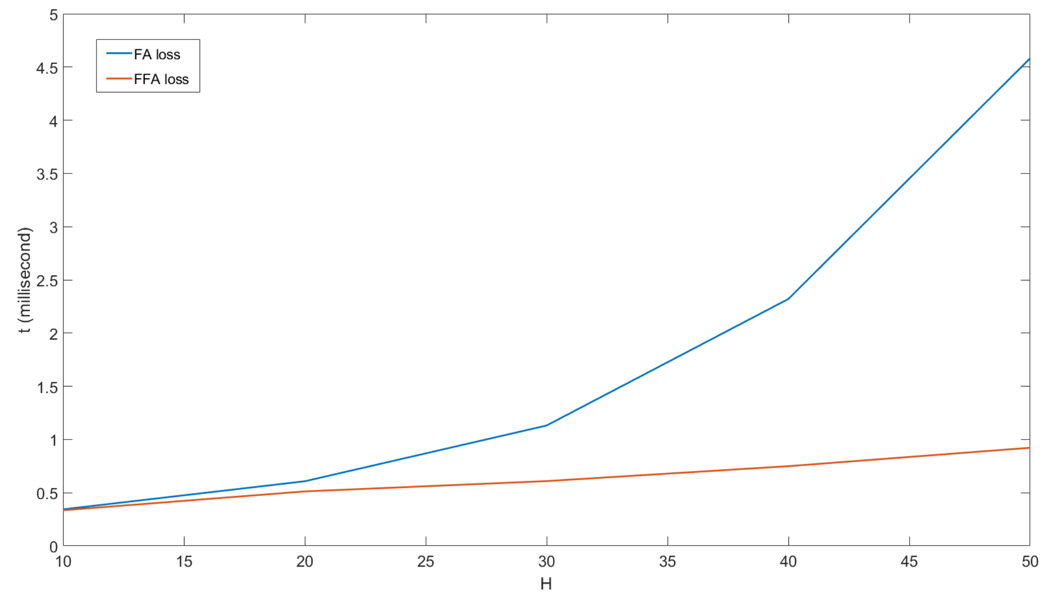
different spatial size H, we record the time of 50 running of FA loss and FFA loss respectively and take their average. For FA loss, the maximum spatial size for the features is 140, while larger size will run out of GPU memory on our machine. We draw a log-log plot to describe the result in Figure 4.1a. Although difference in inference rate between FA loss and FFA loss is negligible for small spatial size H (See Figure 4.1b), the speedup is obvious for larger spatial size. And more importantly, FFA loss can handle features with spatial size greater than 140×140, but FA loss can not.

4.5 Conclusion

We propose the Fast Feature Affinity Loss which is developed for knowledge distillation on intermediate feature maps. It maintains the performance of the Feature Affinity Loss, while largely reducing both computational cost and computing memory space. We also theoretically prove that under expectation our Fast Feature Affinity Loss will have similar behavior as the Feature Affinity Loss.



(a) Log-log plot for inference time of FA loss and FFA loss.



(b) Zoomed in plot for inference time of FA loss and FFA loss.

Figure 4.1: Plots for inference time of FA loss and FFA loss.

Chapter 5

Light DETR for Human Detection

5.1 Light DETR model

Light DETR is a light neural network model suitable for human-only detection task. Inspired by DETR [6], we adopt their novel detection pipeline and modify their transformer encoder-decoder architecture into convolutional encoder and transformer decoder. In Section 5.1.1, we briefly recall the architecture of DETR. And we introduce our convolutional based encoder in Section 5.1.2.

5.1.1 DETR architecture

The DETR architecture can be generally divided into 3 parts: a CNN backbone, a transformer encoder-decoder, and a simple feed forward network (FFN) which makes the final detection prediction. The architecture is well summarized in the Figure 5.1.

The original DETR [6] uses ResNet-50 and ResNet-101 [22] as the backbone. The backbone aims to extract feature representation. In the encoder-decoder module, DETR has a

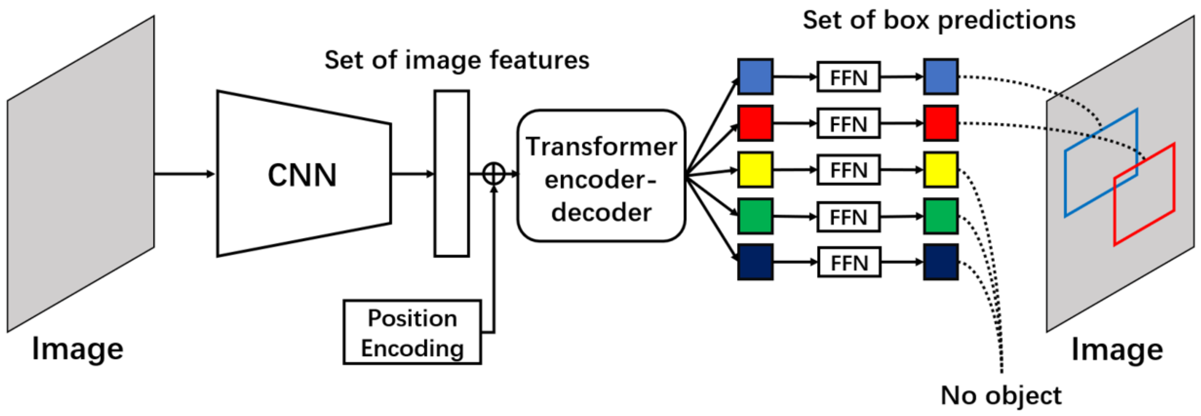


Figure 5.1: DETR architecture.

conventional Transformer encoder which contains a multi-head self-attention module and a feed forward network (FFN). Positional embedding is also included during the computation to keep the position information. And a standard Transformer decoder which transforms N object queries into output embeddings is adopted in DETR. The N output embeddings are then fed to FFN and turned into N predictions. The structure of the Transformer encoder-decoder is depicted in Figure 5.2.

5.1.2 Light DETR architecture

As we are more interested in object detection in daily situation, we constrain the number of prediction slots in original DETR model from default configuration of 100 to 5. We then replace the ResNet-50 and ResNet-101 [22] model with a much more compact model, MobileNet-V3 Large [27], in the backbone. The lighter backbone greatly reduces the computational cost and number of parameters of the model, while it remains acceptable detection performance for our target images. We next shrink the number of decoder layers from 6 to 2 to further compress our model. Now our analysis of the computational cost and the number of parameters indicates that there are 2.95M parameters and 2.74G FLOPs in the backbone,

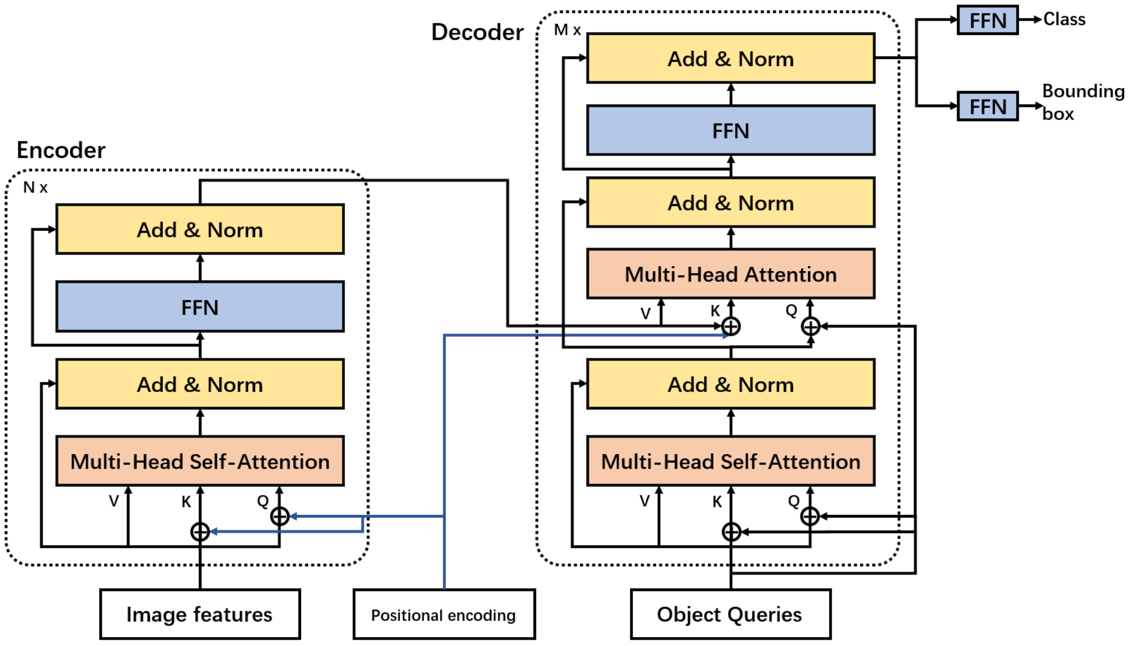
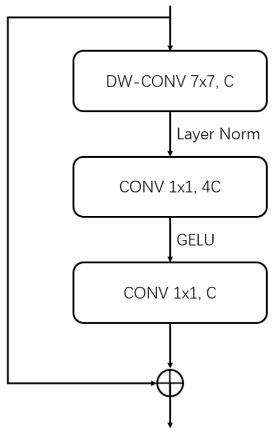


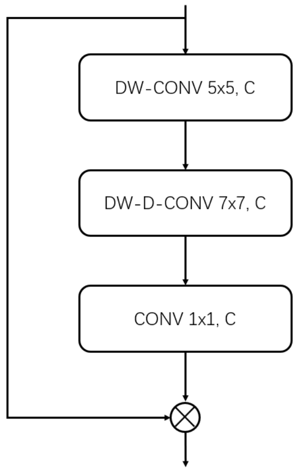
Figure 5.2: Architecture of DETR's transformer.

7.86M parameters and 6.12G FLOPs in the encoder, and only 3.14M parameters and 0.18G FLOPs in the decoder. To further improve the performance, we replace the transformer encoder with two recent convolutional based attention block: ConvNeXt block [45] and Visual Attention Network (VAN) block [21]. To our understanding, convolutional based networks are more efficient in extracting feature representation while transformer based networks are better at decoding.

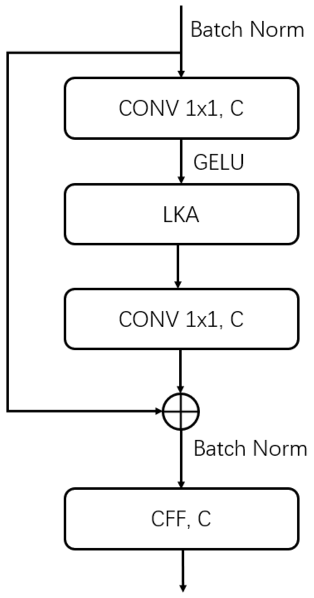
The architecture of two different encoders are described in the Figure 5.3. In the figure, CONV is the convolutional layer, DW-CONV is depth-wise convolution, and DW-D-CONV means dilated depth-wise convolution. CFF means convolutional feed-forward network. For example, CONV 1×1, C means that the corresponding module is a convolutional layer with kernel size 1×1 and input channel number C.



(a) Architecture of ConvNeXt Block.



(b) Large Kernel Attention (LKA).



(c) Architecture of Visual Attention Network (VAN) block
.

Figure 5.3: Light DETR Encoders. CONV is the convolutional layer, DW-CONV is depth-wise convolution, and DW-D-CONV means dilated depth-wise convolution. CFF means convolutional feed-forward network.

Table 5.1: Experiments of Light DETR on person-only data.

Backbone	Encoder	GFLOPS	#params	AP	APs	APm	APl
MobileNetV3	DeTR	9.0	14.0M	53.8	12.9	51.0	72.9
MobileNetV3	ConvNext block [45]	5.0	9.3M	51.0	11.3	46.5	70.5
MobileNetV3	VAN block [21]	5.7	10.6M	54.0	14.0	51.1	73.1
VAN	VAN block [21]	13.0	11.7M	57.7	19.1	55.4	75.3

5.2 Experiment Results

We perform our numerical experiments on object detection task on our self-selected person-only COCO dataset [40]. The person-only data is a subset of the COCO dataset, consisting of images with person annotations only. We have further restricted the number of people to be less than 5, and remove all the crowds of people. The training setting is the same as in DETR. We first train Light DETR with MobileNet-V3 Large backbone as baseline. Then we test models with transformer encoder replaced by convolutional encoder. We also test Light DETR model with VAN backbone and VAN block encoder. All experiments are performed on Pytorch with Quadro RTX 8000 under CUDA 11.2 and CUDNN 7.6.5 environment. The results are list in the Table 5.1.

The ConvNext encoder reduces the FLOPs and parameters almost by half with a 2.8% drop of AP. And to our surprise, the VAN encoder has a slightly better performance than Light DETR with original encoder, but has only around 2/3 FLOPs and parameters. VAN backbone and VAN encoder can bring much higher accuracy, but also increase the FLOPs.

5.3 Conclusion

We present Light DETR, a compact transformer object detection model based on DETR. The model achieves reasonable results on a selected human-only data of COCO dataset. And experiments on different encoder reveal that convolutional based encoders outperform the original transformer encoder in DETR with less FLOPs and parameters and better performance.

Bibliography

[1] AMD. Amd cdna architecture. `https://www.amd.com/system/files/documents/amd-cdna-whitepaper.pdf`. Accessed: 26 April 2022.

[2] M. L. R. at Apple. On-device panoptic segmentation for camera using transformers. `https://machinelearning.apple.com/research/panoptic-segmentation#figure3l`. Accessed: 26 April 2022.

[3] H. Bagherinezhad, M. Horton, M. Rastegari, and A. Farhadi. Label refinery: Improving imagenet classification through label progression.

[4] P. Baldi and R. Vershynin. The capacity of feedforward neural networks. *CoRR*, abs/1901.00434, 2019.

[5] Z. Cai and N. Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[6] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020.

[7] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *ECCV*, 2018.

[8] W. Chen, X. Gong, X. Liu, Q. Zhang, Y. Li, and Z. Wang. Fasterseg: Searching for faster real-time semantic segmentation. *ICLR, 2020; arXiv 1912.10917*, 2019.

[9] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of IEEE CVPR*, July 2017.

[10] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, , and B. Schiele. The cityscapes dataset for semantic urban scene understanding. *CVPR*, 2016.

[11] M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[12] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[13] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier. Language modeling with gated convolutional networks. *CoRR*, abs/1612.08083, 2016.

[14] J. Deng, J. Guo, N. Xue, and S. Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4685–4694, 2019.

[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[16] T. Dinh and J. Xin. Convergence of a relaxed variable splitting method for learning sparse neural networks via l1, l0, and transformed-l1 penalties. *Intelligent Systems Conference (IntelliSys).*

[17] S. E. Dreyfus. The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, 5:30–45, 1962.

[18] R. Girshick. Fast r-cnn. *CoRR*, abs/1504.08083, 2015.

[19] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 00, pages 580–587, June 2014.

[20] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In G. Gordon, D. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[21] M.-H. Guo, C.-Z. Lu, Z.-N. Liu, M.-M. Cheng, and S.-M. Hu. Visual attention network, 2022.

[22] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[23] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[24] H. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *NeurIPS-Worskop*, 2014.

[25] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(2):107–116, 1998.

[26] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[27] A. Howard, M. Sandler, G. Chu, L. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam. Searching for mobilenetv3. *CoRR*, abs/1905.02244, 2019.

[28] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017. cite arxiv:1704.04861.

[29] A. G. Ivakhnenko. Polynomial theory of complex systems. *IEEE transactions on Systems, Man, and Cybernetics*, (4):364–378, 1971.

[30] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[31] H. J. Kelley. Gradient theory of optimal flight paths. *ARS Journal*, 30:947–954, 1960.

[32] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 (canadian institute for advanced research).

[33] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-100 (canadian institute for advanced research).

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[35] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

[36] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[37] N. Lee, T. Ajanthan, and P. Torr. SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY. In *International Conference on Learning Representations*, 2019.

[38] N. Lei, D. An, Y. Guo, K. Su, S. Liu, Z. Luo, S.-T. Yau, and X. Gu. A geometric understanding of deep learning. *Engineering*, 6(3):361–374, 2020.

[39] D. Lin, D. Shen, S. Shen, Y. Ji, D. Lischinski, D. Cohen-Or, and H. Huang. Zigzagnet: Fusing top-down/bottom-up context for object segmentation. *CVPR*, 2019.

[40] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: Common objects in context, 2014. cite arxiv:1405.0312Comment: 1) updated annotation pipeline description and figures; 2) added new section describing datasets splits; 3) updated author list.

[41] S. Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16:146–160, 1976.

[42] C. Liu, L. Chen, F. Schroff, H. Adam, H. Wei, A. Yuille, and F. Li. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. *CVPR, 2019; arXiv:1901.02985v2*, 2019.

[43] H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. *ICLR, 2019; arXiv preprint arXiv:1806.09055*, 2018.

[44] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *ECCV (1)*, volume 9905 of *Lecture Notes in Computer Science*, pages 21–37. Springer, 2016.

[45] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. A convnet for the 2020s, 2022.

[46] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning, 2018.

[47] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5068–5076, 2017.

[48] J. Lyu, S. Zhang, Y. Qi, and J. Xin. Autoshufflenet: Learning permutation matrices via an exact Lipschitz continuous penalty in deep convolutional neural networks. *KDD*, 2020.

[49] D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. van der Maaten. Exploring the limits of weakly supervised pretraining. In V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, editors, *Computer Vision – ECCV 2018*, pages 185–201, Cham, 2018. Springer International Publishing.

[50] I. MKL-DNN. Intel(r) math kernel library for deep neural networks (intel(r) mkl-dnn). `https://oneapi-src.github.io/oneDNN/v0/index.html`. Accessed: 26 April 2022.

[51] M. Nagel, M. van Baalen, T. Blankevoort, and M. Welling. Data-free quantization through weight equalization and bias correction. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1325–1334, 2019.

[52] NVIDIA. Nvidia h100 tensor core gpu architecture overview. `https://resources.nvidia.com/en-us-tensor-core`. Accessed: 26 April 2022.

[53] NVIDIA. Tensorrt documentation: Working with int8. `https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#working-with-int8`. Accessed: 26 April 2022.

[54] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection, 2015. cite arxiv:1506.02640.

[55] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[56] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets, ICLR, 2015.

[57] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536, 1986.

[58] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[59] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883, 2016.

[60] R. Shwartz-Ziv and N. Tishby. Opening the black box of deep neural networks via information. *CoRR*, abs/1703.00810, 2017.

[61] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[62] K. Sun, Y. Zhao, B. Jiang, T. Cheng, B. Xiao, D. Liu, Y. Mu, X. Wang, W. Liu, and J. Wang. High-resolution representations for labeling pixels and regions. *arXiv: 1904.04514*, 2019.

[63] T. Takikawa, D. Acuna, V. Jampani, and S. Fidler. Gated-scnn: Gated shape cnns for semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5229–5238, 2019.

[64] M. Tan and Q. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019.

[65] A. Tao, K. Sapra, and B. Catanzaro. Hierarchical multi-scale attention for semantic segmentation. *arXiv: 2005.10821*, 2020.

[66] Y. Tian, D. Krishnan, and P. Isola. Contrastive representation distillation. In *International Conference on Learning Representations*, 2020.

[67] L. Wang, D. Li, Y. Zhu, L. Tian, and Y. Shan. Dual super-resolution learning for semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3774–3783, 2020.

[68] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. H. Li. Learning structured sparsity in deep neural networks. In *NIPS*, 2016.

[69] B. Yang, J. Lyu, S. Zhang, Y. Qi, and J. Xin. Channel pruning for deep neural networks via a relaxed groupwise splitting method. In *2019 Second International Conference on Artificial Intelligence for Industries (AI4I)*, pages 97–98, 2019.

[70] B. Yang, F. Xue, Y. Qi, and J. Xin. Improving efficient semantic segmentation networks by enhancing multi-scale feature representation via resolution path based knowledge distillation and pixel shuffle. In G. Bebis, V. Athitsos, T. Yan, M. Lau, F. Li, C. Shi, X. Yuan, C. Mousas, and G. Bruder, editors, *Advances in Visual Computing*, pages 325–336, Cham, 2021. Springer International Publishing.

[71] J. Yim, D. Joo, J. Bae, and J. Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[72] P. Yin, S. Zhang, J. Lyu, S. Osher, Y. Qi, and J. Xin. Binaryrelax: A relaxation approach for training deep neural networks with quantized weights. *SIAM Journal on Imaging Sciences*, 11(4):2205–2223, 2018.

[73] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang. BiSenet: Bilateral segmentation network for real-time semantic segmentation. ECCV, 2018.

[74] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society Series B*, 68:49–67, 02 2006.

[75] Y. Yuan, X. Chen, and J. Wang. Object-contextual representations for semantic segmentation. *arXiv preprint arXiv:1909.11065v5*, ECCV, 2020.

[76] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *ECCV*, 2018.

[77] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CVPR*, 2017.