HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF ELECTRONICS AND ELECTRICAL ENGINEERING

THESIS

BACHELOR GRADUATION

Research:

PAT: PIXEL-WISE ADAPTIVE TRAINING FOR LONG-TAILED SEGMENTATION

Student:	DO HOANG KHOI
	Advanced Program 01 - K65
Supervisor:	DR. NGUYEN VIET DUNG

Hanoi, October 21, 2024

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF ELECTRONICS AND ELECTRICAL ENGINEERING

THESIS

BACHELOR GRADUATION

Research:

PAT: PIXEL-WISE ADAPTIVE TRAINING FOR LONG-TAILED SEGMENTATION

Student:	DO HOANG KHOI
	Advanced Program 01 - K65
Supervisor:	DR. NGUYEN VIET DUNG
Reviewers:	

Hanoi, October 21, 2024

ASSESSMENT
(For the supervisor)

Supervisor: Nguyen Viet Dung
Student name: Do Hoang Khoi Student ID: 20200332
Subject: Pat: Pixel-wise Adaptive Training for Long-tailed Segmentation

. . . .

Choose the appropriate scores for students following the criteria below:
Very poor (1); Poor (2); Average (3); Good (4); Excellent (5)

The combination of theory and practice (20)						
1	Clearly state the urgency and importance of the subject, the problems and assumptions (include purposes and relevance) as well as the scope of application of the thesis.	1	2	3	4	5
2	Update the latest research results (national/international).	1	2	3	4	5
3	Clearly state the study/problem-solving methodology in detail.	1	2	3	4	5
4	Have stimulation/experimental results and describe obtained results in detail.	1	2	3	4	5
Ability to analyze and evaluate the results (15)						
5	Clear working plan, including objectives and methodology based on systemically theoretical study results.	1	2	3	4	5
6	Results are presented logically and easy to understand; all results are satisfactorily analyzed and assessed.	1	2	3	4	5
7	In the conclusion section, the author specifies the differences (if any) between the results obtained and the initial objectives while providing arguments to propose possible solutions in the future.	1	2	3	4	5
Writing skill (10)						
8	The thesis is presented in a prescribed format with a logical and nice structure of chapters (Tables, and figures are clear with captions, are numbered and explained or mentioned in the thesis; has alignments, has spaces after full stops and commas, etc.), has a chapter introductions and conclusions, listed references and citations following regulations.	1	2	3	4	5
9	Excellent writing skill (right syntax, scientific style, logical reasoning, appropriate vocabularies, etc.).	1	2	3	4	5
Science research achievements (5) *(choose 1 in 3 options)*						
10a	Had the published or accepted scientific articles/3rd prize at School level at student science research conference or higher/3rd scientific prize (international/national) or higher/registered patents.	5				
10b	Reported at School-level board in student science research conference but not achieved 3rd prize or higher/Achieved a consolation prize at other nationally or internationally specialized competitions such as TI contest	2				
10c	No achievement in science research activity.	0				
Total score		/50				
Total score on scale of 10						

Other comments *(Supervisor comments on student's work attitude and spirit)*

....

....

....

....

....

....

Date: ... / ... / 2023

Supervisor

(Signature)

PREFACE

The decade of deep learning (DL) has witnessed an unprecedented surge in real-world applications, igniting a fervent pursuit of solutions that transcend the confines of traditional datasets. These datasets, meticulously curated with balanced classes and faithful representations of test set distributions, often fall short of encapsulating the full spectrum of real-world scenarios. Consequently, the imperative emerges to cultivate robust algorithms capable of navigating the complexities of imbalanced datasets and unforeseen testing conditions, characterized by distributional shifts between training and testing environments. Label prior and non-semantic likelihood discrepancies loom large among these shifts, as extensively discussed in contemporary literature.

In response to these challenges, researchers have endeavored to fortify DL algorithms against the perils of long-tailed rare categories in segmentation tasks. Strategies such as resampling, data augmentation, logit adjustment, and domain adaptation have been vigorously pursued. However, the persistent imbalance among classes within samples poses a formidable hurdle, with existing methodologies proving insufficient in addressing the intricacies of long-tailed segmentation. Moreover, the absence of research addressing imbalances within mask representations further exacerbates the problem.

In light of these dilemmas, this thesis embarks on a comprehensive exploration of long-tailed segmentation, unraveling nuanced insights into imbalanced mask representations and model uncertainty. Building upon these insights, the thesis introduces Pixel-wise Adaptive Training (PAT) as a pioneering solution to the long-standing challenges of long-tailed rare category problems in segmentation. PAT comprises two pivotal contributions: Class-wise Gradient Magnitude Homogenization and Pixel-wise Class-Specific Loss Adaptation (PCLA). These innovations not only address imbalanced learning stemming from disparities in object sizes and pixel-wise loss contributions but also offer a paradigm shift in mitigating the adverse effects of model uncertainty.

Through meticulous experimentation and analysis, this thesis endeavors to shed light on the efficacy and practical implications of PAT in the realm of long-tailed segmentation. By elucidating the intricacies of imbalanced learning and proposing novel methodologies to mitigate its impact, this work aims to propel the field towards more robust and equitable solutions, poised to tackle the challenges of real-world applications head-on.

In this thesis, I want to give a special thanks to Dr. Viet Dung Nguyen, who has worked with me for nearly 6 years on the road of professional research. He has supported me not only with domain knowledge in the related field, and research skills but also a good computing resource for conducting research.

I also want to give a special thanks to Dr. Minh Duong Nguyen, from Pusan National University, who has been my best collaborator on the road of professional research. Mr. Duong was the person who helped me so much to dive deep into the theoretical understanding of Deep Learning and Optimization.

Besides, I want to give thanks to Professor Tran Hoang Nguyen from the University of Sydney, who helped me a lot in this research, Professor Pham Thanh Long from the University of Warwick, who gives a lot of experience in research and emotion-controlling, and Professor Pham Quoc Viet from Trinity College of Dublin who helped me a lot revising research papers.

Finally, I want to give a special thanks to the ScaleMind team, a research team founded by me under the BioSignal and Image Analysis lab of Dr. Viet Dung Nguyen. The team includes Hoang Minh Thao, Ngo Anh Huyen, Nguyen Phuong Linh, Duong Le Minh, Ngo Thi Thu Ha, Tran Minh Hieu, Mai Tat Chuyen, Nguyen Nhat Nguyen Duong, Dang Thi Thanh Huyen, Trinh Khanh Ly, Ong Thi Huyen Trang. They are all big talents in my team, who force me not to stay but to keep improving myself all day. They are also a big motivation for me to find a way to be a better version of myself not only in research but also in daily life. At the end of my undergraduate journey, I thank them so much for believing in me and giving me a big and responsible chance to be their leader.

PLEDGE

My name is DO HOANG KHOI, student code 20200332, from Advanced Electronics 01, cohort 65. My supervisor is Dr. Viet Dung Nguyen. I guarantee that the whole information provided, and references are followed by the terms and conditions of Intellectual Property. All the references are listed clearly. I take full responsibility for the content written in this thesis.

Hanoi, October 21, 2024

Pledger

Khoi

Do Hoang Khoi

TABLE OF CONTENTS

ABBREVIATIONS

Aug	Data augmentation
CD	Classifier Design
CNN	Convolutional Neural Network
CRF	Conditional Random Fields
CRL	Class Rectification Loss
CSL	Class-sensitive learning
DA	Domain Adaptation
DCL	Dynamic Curriculum Learning
DL	Deep Learning
DT	Decoupled Training
Ensemble	Ensemble learning based methods
FA	Feature Augmentation
SA	Samling Augmentation
FTL	Feature Transfer Learning
GAN	Generative adversarial networks
HFL	Hierarchical Feature Learning
IEM	Inflated Episodic Memory
LA	Logits adjustment
LMLE	Large Margin Local Embedding
LSTM	Long Short Term Memory
MetaSAug	Meta Semantic Augmentation
MLP	Multilayer Perceptron
M2m	Major-to-Minor translation
OFA	Online Feature Augmentation
OLTR	Open Long-Tailed Recognition
PaCo	Parametric Contrastive Learning
PAT	Pixel-wise Adaptive Training
PCLA	Pixel-wise class-specific loss adaptation
PPVs	Pixel-wise predicting vectors
RL	Representation Learning
RNN	Recurrent Neural Network
RSG	Rare-class Sample Generator
TL	Transfer learning
ViT	Vision Transformer

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Beyond class frequency, we recognize the impact of class-wise relationships among various class-specific predictions and the imbalance in label masks on long-tailed segmentation learning. To address these challenges, we propose an innovative Pixel-wise Adaptive Training (PAT) technique tailored for long-tailed segmentation. PAT has two key features: 1) class-wise gradient magnitude homogenization, and 2) pixel-wise class-specific loss adaptation (PCLA). First, the class-wise gradient magnitude homogenization helps alleviate the imbalance among label masks by ensuring *equal consideration of the class-wise impact on model updates*. Second, PCLA tackles the detrimental impact of both rare classes within the long-tailed distribution and inaccurate predictions from previous training stages by *encouraging learning classes with low prediction confidence* and *guarding against forgetting classes with high confidence*. This combined approach fosters robust learning while preventing the model from forgetting previously learned knowledge. PAT exhibits significant performance improvements, surpassing the current state-of-the-art by 2.2% in the NyU dataset. Moreover, it enhances overall pixel-wise accuracy by 2.85% and intersection over union value by 2.07%, with a particularly notable declination of 0.39% in detecting rare classes compared to Balance Logits Variation, as demonstrated on the three popular datasets, i.e., OxfordPetIII, CityScape, and NYU. The code is available at https://github.com/KhoiDOO/ibla.

CHAPTER 1. INTRODUCTION

The integration of deep learning (DL) into real-world applications has sparked a renewed and widespread enthusiasm for research that extends beyond meticulously crafted datasets, which typically feature balanced classes and accurate representations of the test set distribution. However, it remains highly doubtful that it is feasible to foresee and construct datasets that consistently encompass all potential scenarios. Therefore, it is of paramount importance to explore and develop robust algorithms that are capable of performing well on imbalanced datasets and unforeseen circumstances during the testing phase. These challenges can be classified as distributional shifts between the training and testing conditions. Notably, these shifts include label prior shift and non-semantic likelihood shift, as discussed in the literature [11].

Many researchers attempt to propose robust algorithms against long-tailed rare categories in segmentation via resampling [12, 13], data augmentation [8, 14], logits adjustment (LA) [15, 16, 17, 18, 19], domain adaptation (DA) [20, 21, 22]. However, the imbalance among classes inside samples in segmentation remains a critical issue. Existing efforts predominantly concentrate on addressing sample imbalance within classes, with limited research devoted to long-tailed segmentation. The data augmentation and sampling method prove inadequate as it can neither augment nor sample classes within masks. Additionally, designing model architectures for this purpose is highly resource-intensive and computationally demanding.

Recognizing these challenges, we delve into existing research on long-tailed segmentation and uncover intriguing insights. **1) Imbalanced mask representations**: beyond the difficulties posed by rare objects, imbalanced mask representations occur when some masks dominate the learning process, leading to a bias towards recognizing dominant classes and neglecting minority classes. **2) Model uncertainty and degradation**: models facing uncertainty often produce low-precision channel-wise logits, leading to biased gradient updates. These updates favor incorrect label predictions and ignore progress toward the true labels, further degrading performance.

Building upon these insights, we introduce Pixel-wise Adaptive Training (PAT), a novel approach for addressing long-tailed rare category problems in segmentation. PAT comprises two key contributions: **1) Class-wise Gradient Magnitude Homogenization**: We address the imbalanced learning caused by differences in object size across classes by *dividing the loss by the corresponding class mask's size*.

This effectively equalizes the influence of each class on the learning process. **2) Pixel-wise Class-Specific Loss Adaptation (PCLA)**: This component focuses on pixel-wise predicting vectors (PPVs) within each pixel (see Fig. 4.3). By examining the PPVs, we can evaluate how individual channels influence the learning process by analyzing the logit predictions. Specifically, by employing an inverted softmax function, we can strike a balance between two factors: the presence of long-tailed rare objects and the impact of insufficient loss contribution on the joint loss function. This equilibrium enables us to determine coefficients that prioritize learning in classes with rare objects or those with minimal contributions to the joint loss induced by the wrong predictions of the model from previous low-performance learning progress. Consequently, we address the issue of imbalanced learning stemming from both current training samples and previously learned imbalances in the models.

CHAPTER 2. RELATED WORKS

There are three main categories: class re-balancing, information augmentation, and module improvement. In this table, “CSL" indicates class-sensitive learning; “LA" indicates logit adjustment; “TL" represents transfer learning; “Aug" indicates data augmentation; “RL" indicates representation learning; “CD" indicates classifier design, which seeks to design new classifiers or prediction schemes for long-tailed recognition; “DT" indicates decoupled training, where the feature extractor and the classifier are trained separately; “Ensemble" indicates ensemble learning based methods. In addition, “Target Aspect" indicates from which aspect an approach seeks to resolve the class imbalance (refers to Table 2.1).

2.0.1 Re-sampling

Conventional deep network training often relies on random sampling, which overlooks class imbalances in long-tailed learning, resulting in biased models. Re-sampling methods like Dynamic Curriculum Learning (DCL), Long-tailed Object Detector with Classification Equilibrium (LOCE), and VideoLT dynamically adjust sampling rates based on model performance. Meta-learning-based approaches like Balanced Meta-softmax and Feature Augmentation and Sampling Adaptation (FASA) estimate optimal sampling rates through meta-learning. Recommendations include using progressively-balanced sampling when label frequencies are known or utilizing training statistics for real-world applications. These methods provide a foundation for designing multi-level re-sampling strategies to tackle complex imbalance scenarios ([23], [24], [25], [26], [27], [28]).

2.0.2 Class-sensitive Learning

Conventional deep network training with softmax cross-entropy loss ignores class imbalance, leading to uneven gradients across classes. Class-sensitive learning aims to adjust training loss values for different classes to mitigate this imbalance. This approach includes re-weighting strategies, where loss values are adjusted using various methods such as weighted softmax, class-balanced loss, and balanced softmax based on training label frequencies or prediction hardness. Additionally, methods like Focal loss use class prediction hardness for re-weighting, while Meta-Weight-Net learns class weights from data. These techniques aim to rebalance training effects and address class imbalance ([29], [30], [31], [32], [33], [34], [35], [36], [37], [26], [38], [39], [40]).

2.0.3 Logit Adjustment

Logit adjustment addresses class imbalance by modifying prediction logits in biased deep models. Recent studies have analyzed logit adjustment using training label frequencies, demonstrating its Fisher consistency in minimizing average per-class error. RoBal adjusts the cosine classifier based on training label frequencies, but these methods falter when label frequencies are unknown. UNO-IC learns logit offsets from a balanced meta-validation set, while DisAlign employs adaptive calibration functions for logit adjustment. LADE adjusts model outputs based on test label frequencies, making it suitable for agnostic test class distributions, though its practicality is limited by the availability of test label frequencies ([41], [42], [43], [44], [45], [46]).

2.0.4 Data Augmentation

Data Augmentation, a technique to expand and enhance datasets through predefined transformations, plays a crucial role in mitigating class imbalance in long-tailed learning. Two main approaches have been explored: transfer-based augmentation and non-transfer augmentation. In transfer-based augmentation, knowledge from head classes is transferred to augment tail-class samples. For example, Major-to-Minor translation (M2m) translates head-class samples to augment tail classes, while methods like Feature Transfer Learning (FTL) and LEAP enhance tail-class feature spaces based on head-class knowledge of intra-class variance. Rare-class Sample Generator (RSG) dynamically estimates feature centers to augment tail samples, and Online Feature Augmentation (OFA) combines class-specific features of tail-class samples with class-agnostic features from head-class samples. Non-transfer augmentation focuses on adapting conventional methods to long-tailed problems. Classic techniques like SMOTE and recent approaches like MiSLAS and Remix leverage data mixup to address class imbalance. Alternatively, methods like FASA and Meta Semantic Augmentation (MetaSAug) generate new data features based on class-wise Gaussian priors or semantic directions estimated from class-conditional statistics, effectively augmenting tail classes ([47], [48], [49], [50], [51], [52], [53], [54], [55], [56], [27], [57])

2.0.5 Representation Learning

Long-tailed learning methods enhance representation learning through metric learning, prototype learning, and sequential training paradigms. In metric learning, techniques like Large Margin Local Embedding (LMLE) utilize quintuplet sampling to learn discriminative feature spaces, maintaining both intra-cluster and inter-class margins. Class Rectification Loss (CRL) addresses sample differences between head and tail classes by constructing more hard-pair triplets for tail classes. Range loss innovates representation learning by considering overall distances among all sample pairs within mini-batches, mitigating data number imbalance across classes. Contrastive learning approaches such as KCL and Parametric Contrastive learning (PaCo) aim to learn balanced feature spaces, while DRO-LT extends this with distribution robust optimization for improved robustness to distribution shift. Prototype learning methods like Open Long-Tailed Recognition (OLTR) and Inflated Episodic Memory (IEM) learn class-specific feature prototypes to handle long-tailed recognition, with IEM dynamically updating memory blocks for better representation of real data distribution. Sequential training methods such as Hierarchical Feature Learning (HFL) and Unequal-training hierarchically cluster objects or treat head-class and tail-class subsets differently to gradually transfer knowledge and enhance inter-class discrimination ([58], [59], [60], [61], [62], [63], [64], [65], [66], [67], [68]).

Table 2.1 Existing deep long-tailed learning methods in the top-tier conferences.

Method	Year	Class Re-balancing			Augmentation		Module Improvement				Target Aspect
		Re-sampling	CSL	LA	TL	Aug	RL	CD	DT	Ensemble	
LMLE [58]	2016						✓				feature
HFL [67]	2016						✓				feature
Focal loss [39]	2017		✓								objective
Range loss [60]	2017						✓				feature
CRL [59]	2017						✓				feature
MetaModelNet [69]	2017				✓						
DSTL [70]	2018				✓						
DCL [23]	2019	✓									sample
Meta-Weight-Net [40]	2019		✓								objective
LDAM [71]	2019		✓								objective
CB [37]	2019		✓								objective
UML [72]	2019		✓								feature
FTL [50]	2019				✓	✓					feature
Unequal-training [68]	2019						✓				feature
OLTR [65]	2019						✓				feature
Balanced Meta-Softmax [26]	2020	✓	✓								sample, objective
Decoupling [73]	2020	✓	✓				✓	✓	✓		feature, classifier
LST [74]	2020	✓			✓						sample
Domain adaptation [75]	2020		✓								objective
Equalization loss (ESQL) [29]	2020		✓								objective
DBM [76]	2020		✓								objective
Distribution-balanced loss [77]	2020		✓								objective
UNO-IC [44]	2020			✓							prediction
De-confound-TDE [78]	2020			✓				✓			prediction
M2m [49]	2020				✓	✓					sample
LEAP [51]	2020				✓	✓	✓				feature
OFA [53]	2020				✓	✓			✓		feature
SSP [79]	2020				✓		✓				feature
LFME [80]	2020				✓					✓	sample, model
IEM [66]	2020						✓				feature
Deep-RTC [81]	2020							✓			classifier
SimCal [28]	2020								✓	✓	sample, model
BBN [82]	2020									✓	sample, model
BAGS [83]	2020									✓	sample, model
VideoLT [25]	2021	✓									sample
LOCE [24]	2021	✓	✓								sample, objective
DARS [84]	2021	✓	✓		✓						sample, objective
CReST [85]	2021	✓			✓						sample
GIST [86]	2021	✓			✓			✓			classifier
FASA [27]	2021	✓				✓					feature
Equalization loss v2 [87]	2021		✓								objective
Seesaw loss [88]	2021		✓								objective
ACSL [89]	2021		✓								objective
IB [90]	2021		✓								objective
PML [91]	2021		✓								objective
VS [38]	2021		✓								objective
LADE [46]	2021		✓	✓							objective, prediction
RoBal [43]	2021		✓	✓				✓			objective, prediction
DisAlign [45]	2021		✓	✓					✓		objective, classifier
MiSLAS [55]	2021		✓			✓			✓		objective, feature, classifier
Logit adjustment [42]	2021			✓							prediction
Conceptual 12M [92]	2021				✓						
DiVE [93]	2021				✓						
MosaicOS [94]	2021				✓						
RSG [52]	2021				✓	✓					feature
SSD [95]	2021				✓				✓		
RIDE [96]	2021				✓					✓	model
MetaSAug [57]	2021					✓					sample
PaCo [62]	2021						✓				feature
DRO-LT [64]	2021						✓				feature
Unsupervised discovery [97]	2021						✓				feature
Hybrid [63]	2021						✓				feature
KCL [61]	2021						✓		✓		feature
DT2 [98]	2021								✓		feature, classifier
LTML [99]	2021									✓	sample, model
ACE [100]	2021									✓	sample, model
ResLT [101]	2021									✓	sample, model
SADE [102]	2021									✓	objective, model

3.1 Deep Neural Network

Linearity entails the *weaker* presumption of *monotonicity*, meaning that any increase in a feature must either consistently lead to an increase in the model's output (if the corresponding weight is positive) or consistently result in a decrease in the model's output (if the corresponding weight is negative). This assumption is sometimes reasonable. For instance, when predicting whether an individual will repay a loan, it is plausible to assume that, all other factors being equal, an applicant with a higher income would always be more likely to repay than one with a lower income. While this relationship is monotonic, it is unlikely to be linearly related to the probability of repayment. An increase in income from 0 to 50,000 probably results in a more significant increase in the likelihood of repayment than an increase from 1 to 1.05 million. One way to address this issue is to postprocess the outcome to make linearity more plausible, such as by using the logistic map (and thus the logarithm of the probability of outcome).

The limitations of linear models can be overcome by incorporating one or more hidden layers. The simplest method to achieve this is by stacking multiple fully connected layers sequentially. Each layer passes its output to the next layer until the final outputs are generated. The first $L-1$ layers can be considered the representation, while the final layer acts as the linear predictor. This structure is commonly known as a *multilayer perceptron*, often abbreviated as *MLP*.

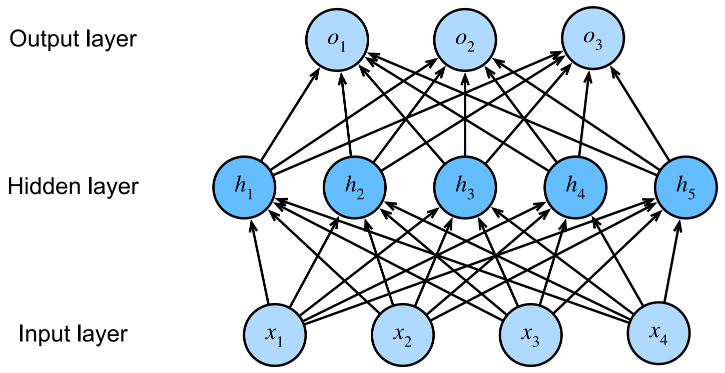


Figure 3.1 An MLP with a hidden layer of five hidden units [4].

Figure 3.1 illustrates an MLP with four inputs and three outputs, featuring a hidden layer composed of five hidden units. As the input layer does not perform any computations, generating outputs with this network necessitates executing the calculations for both the hidden and output layers. Consequently, this MLP is considered to have two layers. It is important to note that both layers are fully connected. Each input affects every neuron in the hidden layer, and each of these neurons subsequently affects every neuron in the output layer.

3.1.1 Architecture Design

The matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$ represents a minibatch of n examples, with each example containing d inputs (features). For an MLP with one hidden layer comprising h hidden units, the outputs of the hidden layer, referred to as *hidden representations*, are represented by $\mathbf{H} \in \mathbb{R}^{n\times h}$. Given that both the hidden and output layers are fully connected, the hidden-layer weights are denoted by $\mathbf{W}^{(1)} \in \mathbb{R}^{d\times h}$ with biases $\mathbf{b}^{(1)} \in \mathbb{R}^{1\times h}$, and the output-layer weights by $\mathbf{W}^{(2)} \in \mathbb{R}^{h\times q}$ with biases $\mathbf{b}^{(2)} \in \mathbb{R}^{1\times q}$. This configuration allows for the calculation of the outputs $\mathbf{O} \in \mathbb{R}^{n\times q}$ of the one-hidden-layer MLP as follows:

$$\begin{aligned} \mathbf{H} &= \mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}, \\ \mathbf{O} &= \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}. \end{aligned} \tag{3.1}$$

Note that with the introduction of the hidden layer, the model must track and update additional sets of parameters. What is the benefit of this addition? The hidden units described above are defined by an affine transformation of the inputs, and the outputs (pre-softmax) are simply an affine transformation of the hidden units. An affine transformation of an affine transformation remains an affine transformation. Furthermore, a linear model was already able to represent any affine function.

To see this more formally, the hidden layer can be eliminated in the definition above, resulting in an equivalent single-layer model with parameters $\mathbf{W} = \mathbf{W}^{(1)}\mathbf{W}^{(2)}$ and $\mathbf{b} = \mathbf{b}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}$:

$$\mathbf{O} = (\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})\mathbf{W}^{(2)} + \mathbf{b}^{(2)} = \mathbf{X}\mathbf{W}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)} = \mathbf{X}\mathbf{W} + \mathbf{b}. \tag{3.2}$$

To fully harness the power of multilayer architectures, an essential component is required: a nonlinear *activation function* σ, which is applied to each hidden unit following the affine transformation. A common example is the ReLU (rectified linear unit) activation function $\sigma(x) = \max(0, x)$, which operates elementwise on its arguments. The outputs produced by the activation function $\sigma(\cdot)$ are referred to as *activations*. In general, once activation functions are incorporated, it is no longer feasible to reduce the MLP to a linear model:

$$\mathbf{H} = \sigma(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}), \tag{3.3}$$

$$\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}. \tag{3.4}$$

Since each row in $\mathbf{X}$ represents an example in the minibatch, we define the nonlinearity σ to operate on its inputs in a row-wise manner, meaning it processes one example at a time. This notation is similarly used for softmax when denoting:

3.1.2 Optimization Algorithm

1) Forward Propagation

The procedure of *forward propagation* (or *forward pass*) involves the sequential computation and storage of intermediate variables and outputs within a neural network, advancing from the input layer to the output layer. This section provides a detailed explanation of the steps involved in a neural network with one hidden layer, describing each step precisely. While this might appear thorough, as the funk legend James Brown famously said, one must "pay the cost to be the boss."

For the sake of clarity and brevity, let us assume that the input example is denoted as $\mathbf{x} \in \mathbb{R}^d$, and that the hidden layer lacks a bias term. Therefore, the intermediate variable is defined as:

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x}, \tag{3.5}$$

where $\mathbf{W}^{(1)} \in \mathbb{R}^{h \times d}$ represents the weight parameter for the hidden layer. After applying the activation function ϕ, the resulting hidden activation vector, which has a length of h, is expressed as:

$$\mathbf{h} = \phi(\mathbf{z}). \tag{3.6}$$

The output from the hidden layer, denoted as $\mathbf{h}$, acts as an intermediate variable. If we consider that the parameters of the output layer consist only of the weight matrix $\mathbf{W}^{(2)} \in \mathbb{R}^{q \times h}$, the output layer variable is calculated as a vector of length q:

$$\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}. \tag{3.7}$$

Following this, given a loss function represented by l and a label for a single example denoted as y, the loss term for that individual data example is calculated as:

$$L = l(\mathbf{o}, y). \tag{3.8}$$

Moreover, integrating the ℓ_2 regularization, which will be explained in detail later and represented by the hyperparameter λ, the regularization term can be formulated as:

$$s = \frac{\lambda}{2}\left(\|\mathbf{W}^{(1)}\|_{\mathrm{F}}^2 + \|\mathbf{W}^{(2)}\|_{\mathrm{F}}^2\right), \tag{3.9}$$

Here, the Frobenius norm of the matrix is essentially the ℓ_2 norm applied after the matrix has been flattened into a vector. Consequently, the regularized loss of the model for a given data example is established as:

$$J = L + s. \tag{3.10}$$

Throughout the subsequent discourse, J is referred to as the *objective function.*

2) Backpropagation

Backpropagation involves computing the gradients of neural network parameters. In essence, it entails traversing the network in reverse order, from the output to the input layer, applying the chain rule from calculus. The algorithm stores any intermediate variables (partial derivatives) needed during the gradient computation for specific parameters. Let's consider functions $Y = f(X)$ and $Z = g(Y)$, where the input and output, X, Y, Z, are tensors of arbitrary shapes. Applying the chain rule, the derivative of Z concerning X can be calculated as:

$$\frac{\partial Z}{\partial X} = \text{prod}\left(\frac{\partial Z}{\partial Y}, \frac{\partial Y}{\partial X}\right). \tag{3.11}$$

In this context, the prod operator performs multiplication on its arguments following necessary operations like transposition and swapping input positions. This operation is straightforward for vectors, where it simplifies to matrix-matrix multiplication. For tensors of higher dimensions, the appropriate operation is employed. Utilizing the prod operator streamlines the notation, removing unnecessary complexity.

The parameters of the simple network with a single hidden layer are denoted by $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$. The gradients $\partial J/\partial \mathbf{W}^{(1)}$ and $\partial J/\partial \mathbf{W}^{(2)}$ are computed through backpropagation. This involves applying the chain rule and calculating the gradient of each intermediate variable and parameter in sequence. Unlike forward propagation, where calculations proceed from input to output, in backpropagation, the order is reversed to start with the outcome of the computational graph and work backward towards the parameters. Initially, the gradients of the objective function $J = L + s$ with respect to the loss term L and the regularization term s are computed:

$$\frac{\partial J}{\partial L} = 1 \quad \text{and} \quad \frac{\partial J}{\partial s} = 1. \tag{3.12}$$

Moving forward, we determine the gradient of the objective function with respect to the output layer variable $\mathbf{o}$ using the chain rule:

$$\frac{\partial J}{\partial \mathbf{o}} = \text{prod}\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \mathbf{o}}\right) = \frac{\partial L}{\partial \mathbf{o}} \in \mathbb{R}^q. \tag{3.13}$$

Next, the gradient of the objective function concerning the variable of the output layer $\mathbf{o}$ is computed according to the chain rule:

$$\frac{\partial J}{\partial \mathbf{o}} = \text{prod}\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \mathbf{o}}\right) = \frac{\partial L}{\partial \mathbf{o}} \in \mathbb{R}^q. \tag{3.14}$$

Next, the gradients of the regularization term concerning both parameters are calculated:

$$\frac{\partial s}{\partial \mathbf{W}^{(1)}} = \lambda \mathbf{W}^{(1)} \quad \text{and} \quad \frac{\partial s}{\partial \mathbf{W}^{(2)}} = \lambda \mathbf{W}^{(2)}. \tag{3.15}$$

Now the gradient $\partial J/\partial \mathbf{W}^{(2)} \in \mathbb{R}^{q \times h}$ of the model parameters closest to the output layer is calculated. Using the chain rule yields:

$$\frac{\partial J}{\partial \mathbf{W}^{(2)}} = \text{prod}\left(\frac{\partial J}{\partial \mathbf{o}}, \frac{\partial \mathbf{o}}{\partial \mathbf{W}^{(2)}}\right) + \text{prod}\left(\frac{\partial J}{\partial s}, \frac{\partial s}{\partial \mathbf{W}^{(2)}}\right) = \frac{\partial J}{\partial \mathbf{o}} \mathbf{h}^\top + \lambda \mathbf{W}^{(2)}. \tag{3.16}$$

To obtain the gradient concerning $\mathbf{W}^{(1)}$, the backpropagation continues along the output layer to the hidden layer. The gradient with respect to the hidden layer output $\partial J/\partial \mathbf{h} \in \mathbb{R}^h$ is given by

$$\frac{\partial J}{\partial \mathbf{h}} = \text{prod}\left(\frac{\partial J}{\partial \mathbf{o}}, \frac{\partial \mathbf{o}}{\partial \mathbf{h}}\right) = \mathbf{W}^{(2)\top} \frac{\partial J}{\partial \mathbf{o}}. \tag{3.17}$$

Since the activation function, ϕ, applies elementwise, calculating the gradient $\partial J/\partial \mathbf{z} \in \mathbb{R}^h$ of the intermediate variable $\mathbf{z}$ requires that the elementwise multiplication operator, denoted by $\odot$, is used:

$$\frac{\partial J}{\partial \mathbf{z}} = \text{prod}\left(\frac{\partial J}{\partial \mathbf{h}}, \frac{\partial \mathbf{h}}{\partial \mathbf{z}}\right) = \frac{\partial J}{\partial \mathbf{h}} \odot \phi'(\mathbf{z}). \tag{3.18}$$

Finally, the gradient $\partial J/\partial \mathbf{W}^{(1)} \in \mathbb{R}^{h\times d}$ of the model parameters closest to the input layer is obtained. According to the chain rule, we have

$$\frac{\partial J}{\partial \mathbf{W}^{(1)}} = \text{prod}\left(\frac{\partial J}{\partial \mathbf{z}}, \frac{\partial \mathbf{z}}{\partial \mathbf{W}^{(1)}}\right) + \text{prod}\left(\frac{\partial J}{\partial s}, \frac{\partial s}{\partial \mathbf{W}^{(1)}}\right) = \frac{\partial J}{\partial \mathbf{z}}\mathbf{x}^\top + \lambda \mathbf{W}^{(1)}. \tag{3.19}$$

3) Training Procedure

As described in Algorithm 1, the training process involves multiple epochs of minibatch stochastic gradient descent. This algorithm is widely used for training deep learning models due to its efficiency in handling large datasets and its ability to converge to a good solution over time.

The algorithm begins by taking as input the dataset $\mathcal{D}$, which contains the training examples, the number of epochs E, which determines how many times the algorithm will iterate over the entire dataset, the minibatch size B, which specifies the number of examples in each minibatch, and the learning rate α, which controls the step size of the parameter updates.

During each epoch, the dataset is shuffled to ensure that the model sees the examples in a different order in each iteration. This helps prevent the model from overfitting to the order of the examples in the dataset.

For each minibatch in the shuffled dataset, the algorithm performs the following steps:

1. **Forward Propagation:** The input minibatch $\mathbf{X}$ is passed through the neural network to compute the predicted output $\mathbf{O}$. This involves computing the activations of the hidden layer $\mathbf{H}$ using the Rectified Linear Unit (ReLU) activation function, and then computing the output layer activations using the softmax function.

2. **Backward Propagation:** The gradients of the loss function concerning the output layer activations and hidden layer activations are computed. These gradients are then used to update the model parameters using gradient descent.

3. **Parameter Updates:** The weights and biases of both the output and hidden layers are updated using the computed gradients and the learning rate α.

This process is repeated for each minibatch in the dataset for the specified number of epochs E. By iteratively updating the model parameters based on small batches of data, the algorithm gradually converges toward a solution that minimizes the chosen loss function.

The use of minibatch stochastic gradient descent allows the algorithm to efficiently handle large datasets that may not fit into memory all at once. Additionally, by updating the parameters based on only a subset of the data at a time, the algorithm is less likely to get stuck in local minima and is more likely to find a good solution that generalizes well to unseen data.

Algorithm 1 Training Algorithm for Deep Learning Model

```
1: Input:
2: Minibatch of examples $\mathbf{X} \in \mathbb{R}^{n \times d}$ with $n$ examples and $d$ features.
3: Hidden layer weights $\mathbf{W}^{(1)} \in \mathbb{R}^{d \times h}$ and biases $\mathbf{b}^{(1)} \in \mathbb{R}^{1 \times h}$.
4: Output layer weights $\mathbf{W}^{(2)} \in \mathbb{R}^{h \times q}$ and biases $\mathbf{b}^{(2)} \in \mathbb{R}^{1 \times q}$.
5: Number of epochs $E$.
6: Learning rate $\alpha$.
7: Output: Updated weights and biases for both layers.
8: for $epoch = 1$ to $E$ do
9:     Forward Propagation:
10:    Compute hidden layer activations:
11:    $\mathbf{H} \leftarrow \mathrm{ReLU}(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})$
12:    Compute output layer activations:
13:    $\mathbf{O} \leftarrow \mathrm{softmax}(\mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)})$
14:
15:    Backward Propagation:
16:    Compute gradients of the loss function for output layer activations:
17:    $\frac{\partial J}{\partial \mathbf{O}} \leftarrow \mathbf{O} - \mathbf{Y}$
18:    Compute gradients of the loss function with respect to hidden layer activations:
19:    $\frac{\partial J}{\partial \mathbf{H}} \leftarrow \frac{\partial J}{\partial \mathbf{O}} \mathbf{W}^{(2)^\top}$
20:    Update output layer weights and biases:
21:    $\mathbf{W}^{(2)} \leftarrow \mathbf{W}^{(2)} - \alpha \frac{\partial J}{\partial \mathbf{W}^{(2)}}$
22:    $\mathbf{b}^{(2)} \leftarrow \mathbf{b}^{(2)} - \alpha \frac{\partial J}{\partial \mathbf{b}^{(2)}}$
23:    Update hidden layer weights and biases:
24:    $\mathbf{W}^{(1)} \leftarrow \mathbf{W}^{(1)} - \alpha \frac{\partial J}{\partial \mathbf{W}^{(1)}}$
25:    $\mathbf{b}^{(1)} \leftarrow \mathbf{b}^{(1)} - \alpha \frac{\partial J}{\partial \mathbf{b}^{(1)}}$
26: end for
```

3.2 Deep Convolutional Neural Network

3.2.1 Constraining the MLP

To commence, contemplate an MLP operating on two-dimensional images denoted by **X** as its input and their immediate hidden representations, **H**, represented analogously as matrices (manifesting as two-dimensional tensors in code), where both $\mathbf{X}$ and $\mathbf{H}$ exhibit identical shapes. Let this concept sink in. Now envisage that not only the inputs but also the hidden representations possess spatial arrangement.

Let $[\mathbf{X}]_{i,j}$ and $[\mathbf{H}]_{i,j}$ signify the pixel situated at coordinates (i, j) within the input image and the hidden representation, correspondingly. Consequently, to ensure each hidden unit captures input from every input pixel, transition from utilizing weight matrices (as conventionally practiced in MLPs) to representing parameters as fourth-order weight tensors denoted by W. Assuming $\mathbf{U}$ encompasses biases, the fully connected layer can be formally delineated as:

$$\begin{aligned}[\mathbf{H}]_{i,j} &= [\mathbf{U}]_{i,j} + \sum_k \sum_l [\mathsf{W}]_{i,j,k,l} [\mathbf{X}]_{k,l} && (3.20)\\ &= [\mathbf{U}]_{i,j} + \sum_a \sum_b [\mathsf{V}]_{i,j,a,b} [\mathbf{X}]_{i+a,j+b}. && (3.21)\end{aligned}$$

The transition from W to V is merely superficial at this point, as there exists a direct mapping between coefficients in both fourth-order tensors. Simply rearrange the subscripts (k, l) so that $k = i + a$ and $l = j + b$. In simpler terms, define $[\mathsf{V}]_{i,j,a,b} = [\mathsf{W}]_{i,j,i+a,j+b}$. The indices a and b traverse both positive and negative offsets, encompassing the entire image. For any given position (i, j) within the hidden representation $[\mathbf{H}]_{i,j}$, calculate its value by summing over pixels in $\mathbf{X}$, centered around (i, j), and weighted by $[\mathsf{V}]_{i,j,a,b}$.

Now contemplate the total parameter count necessary for a single layer with this parametrization: a 1000×1000 image (1 megapixel) is mapped to a 1000×1000 hidden representation. This entails 10^{12} parameters, significantly surpassing current computational capabilities.

3.2.2 Translation Invariance

The primary principle to consider is that of translational invariance. This principle suggests that a shift in the input $\mathbf{X}$ should correspondingly shift the hidden representation $\mathbf{H}$. This outcome is only attainable if V and $\mathbf{U}$ do not depend on (i, j). Consequently, it implies that $[\mathsf{V}]_{i,j,a,b} = [\mathbf{V}]_{a,b}$ and $\mathbf{U}$ remains constant, represented as u. Thus, the definition for $\mathbf{H}$ can be simplified as follows:

$$[\mathbf{H}]_{i,j} = u + \sum_a \sum_b [\mathbf{V}]_{a,b} [\mathbf{X}]_{i+a,j+b}. \qquad (3.22)$$

This process is termed a *convolution*. The pixels surrounding the position (i, j) at $(i + a, j + b)$ are multiplied by coefficients $[\mathbf{V}]_{a,b}$ to calculate the value

$[\mathbf{H}]_{i,j}$. It's noteworthy that $[\mathbf{V}]_{a,b}$ necessitates notably fewer coefficients compared to $[\mathsf{V}]_{i,j,a,b}$, as it's no longer contingent on the specific image location. Consequently, the parameter count reduces from 10^{12} to a more manageable 4×10^6, retaining the dependence on $a, b \in (-1000, 1000)$.

3.2.3 Locality

The second principle we consider is that of locality. As discussed earlier, the premise is that it shouldn't be necessary to search far from position (i, j) to gather pertinent information for assessing what's happening at $[\mathbf{H}]_{i,j}$. This suggests that beyond a certain range $|a| > \Delta$ or $|b| > \Delta$, the value of $[\mathbf{V}]_{a,b}$ ought to be set to zero. In other words, $[\mathbf{H}]_{i,j}$ can be reformulated as:

$$[\mathbf{H}]_{i,j} = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} [\mathbf{V}]_{a,b} [\mathbf{X}]_{i+a,j+b}. \tag{3.23}$$

This adjustment decreases the parameter count from 4×10^6 to $4\Delta^2$, where Δ typically remains below 10. Consequently, another four orders of magnitude are shaved off the parameter count. It's worth noting that Equation 3.2.3 depicts what's termed a *convolutional layer*. Convolutional neural networks (CNNs) belong to a specific category of neural networks incorporating convolutional layers. In the deep learning research realm, $\mathbf{V}$ is often referred to as a *convolution kernel*, a *filter*, or simply the layer's *learnable parameters*.

Previously, representing a single layer in an image-processing network might have demanded billions of parameters. Now, typically, only a few hundred parameters suffice, without altering the dimensions of either the inputs or the hidden representations. However, this drastic parameter reduction comes with a trade-off: the features become translation invariant, and the layer can only consider local information when determining each hidden activation's value. All learning relies on imposing inductive bias. When this bias aligns with reality, it yields sample-efficient models that generalize effectively to unseen data. Conversely, if these biases don't correspond with reality, such as if images aren't actually translation invariant, the models may struggle to even fit the training data.

This significant parameter reduction leads to the final goal: deeper layers should capture larger and more intricate aspects of an image. This can be accomplished by interleaving nonlinearities and convolutional layers repeatedly.

3.2.4 Convolutions

Equation 3.2.3 is referred to as a convolution. In mathematics, the *convolution* between two functions, say $f, g : \mathbb{R}^d \to \mathbb{R}$, is defined as

$$(f * g)(\mathbf{x}) = \int f(\mathbf{z}) g(\mathbf{x} - \mathbf{z})\, d\mathbf{z}. \tag{3.24}$$

This formulation quantifies the intersection between f and g as one function is "mirrored" and displaced by $\mathbf{x}$. In the case of discrete entities, the integral transforms into a summation. For instance, for vectors drawn from the collection of square-summable infinite-dimensional vectors with the index spanning across $\mathbb{Z}$, the specification is as follows:

$$(f * g)(i) = \sum_a f(a) g(i - a). \tag{3.25}$$

For two-dimensional tensors, a corresponding sum involves indices (a, b) for f and $(i - a, j - b)$ for g, respectively:

$$(f * g)(i, j) = \sum_a \sum_b f(a, b) g(i - a, j - b). \tag{3.26}$$

This equation bears a resemblance to Equation 3.23, albeit with a significant deviation. Instead of employing $(i+a, j+b)$, it opts for the difference. Nevertheless, this discrepancy is mainly superficial, as the notation between Equation 3.23 and Equation 3.26 can always be aligned. The initial definition in Equation 3.23 more precisely characterizes a *cross-correlation*.

3.2.5 Channels

The current approach encounters a significant issue. Up to this point, the conversation has conveniently overlooked the fact that images consist of three channels: red, green, and blue. In essence, images aren't two-dimensional entities but rather third-order tensors, delineated by height, width, and channel, for instance, with dimensions of $1024 \times 1024 \times 3$ pixels. While the initial two dimensions pertain to spatial relationships, the third dimension assigns a multidimensional representation to each pixel location. Thus, X is indexed as $[\mathsf{X}]_{i,j,k}$. Consequently, the convolutional filter must adapt accordingly. Instead of $[\mathbf{V}]_{a,b}$, it now becomes $[\mathsf{V}]_{a,b,c}$.

Furthermore, just as the input constitutes a third-order tensor, it's advantageous to similarly frame the hidden representations as third-order tensors H. In essence, rather than having a singular hidden representation corresponding to each spatial location, an entire array of hidden representations corresponding to each spatial location is desired. These hidden representations can be envisioned as comprising numerous two-dimensional grids stacked atop each other. They're occasionally referred to as *channels* or *feature maps*, as each furnishes a spatialized set of learned features for the subsequent layer. Conceptually, at lower layers proximate to inputs, some channels might specialize in edge recognition, while others might identify textures.

To accommodate multiple channels in both inputs (X) and hidden representations (H), a fourth coordinate is introduced to V: $[\mathsf{V}]_{a,b,c,d}$. Consolidating all these considerations yields the ensuing expression:

$$[\mathsf{H}]_{i,j,d} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} \sum_{c} [\mathsf{V}]_{a,b,c,d} [\mathsf{X}]_{i+a,j+b,c}, \tag{3.27}$$

The parameter d denotes the output channels in the hidden representations H. The subsequent convolutional layer will accept a third-order tensor, H, as input. Due to its versatility, Equation 3.2.5 is adopted as the definition of a convolutional layer for multiple channels, where V serves as a kernel or filter of the layer.

Numerous operations remain to be addressed. For instance, it's crucial to determine the method of aggregating all the hidden representations into a singular output, such as identifying whether there's a Waldo *anywhere* in the image. Additionally, decisions must be made regarding efficient computation, combining multiple layers, selecting appropriate activation functions, and making judicious design choices to produce networks that perform effectively in practical scenarios.

3.3 Attention Mechanism

During the initial surge of deep learning advancements, the focal points revolved around architectures like the multilayer perceptron, convolutional networks, and recurrent networks. Surprisingly, despite nearly three decades passing, the foundational model architectures driving many breakthroughs in deep learning during the 2010s remained remarkably unchanged. While numerous methodological innovations entered the mainstream, such as ReLU activations, residual layers, batch normalization, dropout, and adaptive learning rate schedules, the fundamental architectures were essentially scaled-up versions of traditional concepts.

Despite numerous alternative proposals, models resembling classical convolutional neural networks continued to maintain their *state-of-the-art* status in computer vision, while models akin to Sepp Hochreiter's original LSTM recurrent neural network design dominated various natural language processing applications. Perhaps up to that juncture, the rapid advancement of deep learning seemed primarily propelled by advancements in available computational resources (courtesy of GPU-driven parallel computing innovations) and the abundance of massive data resources (owing to inexpensive storage and Internet services). While these factors likely remained the primary drivers behind the technology's increasing prowess, we are now witnessing a significant shift in the dominant architectural landscape.

The central innovation behind the Transformer model is the *attention mechanism*, initially conceived as an augmentation for encoder-decoder RNNs employed in sequence-to-sequence tasks like machine translation [103]. Early sequence-to-sequence models for machine translation [104] compressed the entire input by the encoder into a single fixed-length vector for the decoder's consumption. The notion behind attention was to allow the decoder to revisit the input sequence at each step, dynamically focusing on particular parts of the input sequence during decoding. Bahdanau's attention mechanism provided a straightforward way for the decoder to dynamically *attend* to different input segments during each decoding step. Essentially, the encoder could represent the full original input sequence length, and during decoding, the decoder could receive, as input (via a control mechanism), a context vector comprising a weighted sum of the input representations at each time step. These weights determine the degree to which each context emphasizes each input token, with the crucial aspect being the differentiability of the weight assignment process, enabling it to be learned alongside other neural network parameters.

Initially, attention mechanisms proved highly successful enhancements to recurrent neural networks dominating machine translation tasks. These models outperformed original encoder-decoder sequence-to-sequence architectures. Additionally, researchers noted interesting qualitative insights emerging from attention-weight patterns. In translation tasks, attention models often assign high weights to cross-lingual synonyms when generating corresponding words in the target language. For instance, when translating "my feet hurt" to "j'ai mal au pieds", the network might assign high attention weights to the "feet" representation when generating the French word "pieds". While these insights led to claims about attention models providing "interpretability", the precise meaning of attention weights i.e., how if at all, they should be *interpreted* remains a somewhat nebulous research area.

However, attention mechanisms soon emerged as significant concerns beyond their utility as encoder-decoder recurrent neural network enhancements and their purported utility in highlighting salient inputs. [105] introduced the Transformer architecture for machine translation, abandoning recurrent connections entirely and relying instead on well-structured attention mechanisms to capture all input-output token relationships. The architecture performed exceptionally well, with the Transformer becoming prevalent in most state-of-the-art natural language processing systems by 2018. Concurrently, the prevailing practice in natural language processing shifted towards pre-training large-scale models on vast generic background corpora, optimizing them using self-supervised pretraining objectives, and subsequently fine-tuning them with available downstream data. The gap between Transformers and traditional architectures widened significantly when applied in this pretraining paradigm, aligning with the ascendancy of Transformers and the advent of such large-scale pre-trained models, sometimes dubbed foundation models [106].

3.3.1 Queries, Keys, and Values

All the networks discussed so far fundamentally rely on the input having a specified size. For instance, the images in ImageNet have dimensions of 224×224 pixels, and CNNs are optimized for this dimension. Similarly, in natural language processing, the input size for RNNs is defined and fixed. Handling variable size is managed by sequentially processing one token at a time or by utilizing specially designed convolution kernels [107]. However, this strategy can cause significant issues when the input varies in size and information content during text transformation [104]. Specifically, for lengthy sequences, it becomes challenging to keep track of all the information the network has processed or generated. Even explicit tracking heuristics proposed by [108] provide limited advantage.

To further illustrate this concept, consider databases. At their most basic, they are collections of keys (k) and values (v). For example, a database $\mathcal{D}$ might contain tuples {("Zhang", "Aston"), ("Lipton", "Zachary"), ("Li", "Mu"), ("Smola", "Alex"), ("Hu", "Rachel"), ("Werness", "Brent")}, where the last name is the key and the first name is the value. Various operations can be executed on $\mathcal{D}$, such as an exact query (q) for "Li," which would return the value "Mu." If ("Li", "Mu") is not a record in $\mathcal{D}$, no valid answer exists. If approximate matches are allowed, ("Lipton", "Zachary") might be retrieved instead. This simple example illustrates several valuable concepts:

- Queries (q) can be formulated to work on (k, v) pairs such that they remain valid regardless of the database size.

- The same query can produce different responses depending on the database contents.
- The code for handling a large state space (the database) can be quite simple (e.g., exact match, approximate match, top-k).
- There is no need to reduce or simplify the database to make the operations effective.

This discussion leads to one of the most intriguing concepts introduced in deep learning in the past decade: the *attention mechanism* [103]. Define $\mathcal{D} \overset{\text{def}}{=} \{(\mathbf{k}_1, \mathbf{v}_1), \dots (\mathbf{k}_m, \mathbf{v}_m)\}$ as a database of m tuples of *keys* and *values*. Let $\mathbf{q}$ be a *query*. Then we can define the *attention* over $\mathcal{D}$ as

$$\text{Attention}(\mathbf{q}, \mathcal{D}) \overset{\text{def}}{=} \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i, \tag{3.28}$$

where $\alpha(\mathbf{q}, \mathbf{k}_i) \in \mathbb{R}$ $(i = 1, \dots, m)$ are scalar attention weights. The operation itself is typically referred to as *attention pooling*. The name *attention* derives from the fact that the operation pays particular attention to the terms for which the weight α is significant (i.e., large). As such, the attention over $\mathcal{D}$ generates a linear combination of values contained in the database. Several special cases exist:

where $\alpha(\mathbf{q}, \mathbf{k}_i) \in \mathbb{R}$ $(i = 1, \dots, m)$ are scalar attention weights. The operation itself is typically referred to as *attention pooling*. The term *attention* comes from the fact that the operation emphasizes the terms for which the weight α is significant (i.e., large). Thus, the attention over $\mathcal{D}$ produces a linear combination of values in the database. Several special cases exist:

- The weights $\alpha(\mathbf{q}, \mathbf{k}_i)$ are nonnegative. In this case, the output of the attention mechanism lies within the convex cone spanned by the values $\mathbf{v}_i$.
- The weights $\alpha(\mathbf{q}, \mathbf{k}_i)$ form a convex combination, i.e., $\sum_i \alpha(\mathbf{q}, \mathbf{k}_i) = 1$ and $\alpha(\mathbf{q}, \mathbf{k}_i) \geq 0$ for all i. This is the most common setting in deep learning.
- Exactly one of the weights $\alpha(\mathbf{q}, \mathbf{k}_i)$ is 1, while all others are 0. This is similar to a traditional database query.
- All weights are equal, i.e., $\alpha(\mathbf{q}, \mathbf{k}_i) = \frac{1}{m}$ for all i. This is equivalent to averaging across the entire database, also called average pooling in deep learning.

A common strategy for ensuring that the weights sum up to 1 is to normalize them via

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \frac{\alpha(\mathbf{q}, \mathbf{k}_i)}{\sum_j \alpha(\mathbf{q}, \mathbf{k}_j)}. \tag{3.29}$$

Specifically, to guarantee that the weights are nonnegative, exponentiation can be employed. This allows for the selection of any function $a(\mathbf{q}, \mathbf{k})$, followed by the application of the softmax operation used in multinomial models as follows:

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_j \exp(a(\mathbf{q}, \mathbf{k}_j))}. \tag{3.30}$$

This operation is readily implemented in all deep learning frameworks. It is differentiable and its gradient never vanishes, which are beneficial properties for a model. However, the attention mechanism described above is not the sole option. For example, a non-differentiable attention model can be developed and trained using reinforcement learning techniques. Training such a model is challenging, though. Therefore, most contemporary attention research adheres to the framework depicted in Figure 3.2. The discussion centers on this category of differentiable mechanisms.

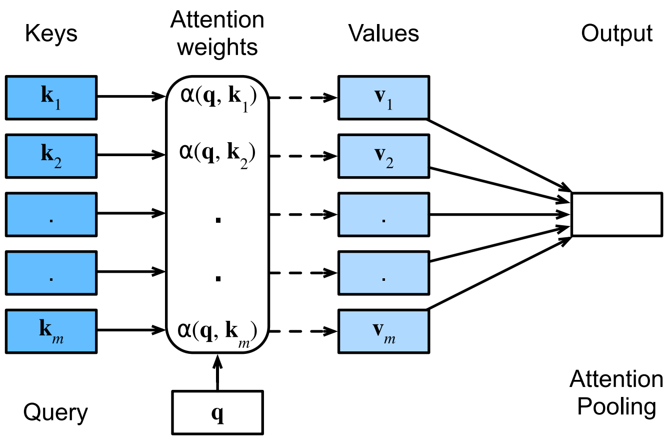


Figure 3.2 The attention mechanism computes a linear combination over values $\mathbf{v}_i$ via attention pooling, where weights are derived according to the compatibility between a query $\mathbf{q}$ and keys $\mathbf{k}_i$ [4].

3.3.2 Attention Pooling by Similarity

With the primary elements of the attention mechanism now introduced, let's apply them in a traditional context, specifically regression and classification through kernel density estimation ([109, 110]). This diversion offers extra background information and can be bypassed if desired. At their essence, Nadaraya–Watson estimators utilize a similarity kernel $\alpha(\mathbf{q}, \mathbf{k})$ that connects queries $\mathbf{q}$ to keys $\mathbf{k}$. Some typical kernels include

$$\alpha(\mathbf{q}, \mathbf{k}) = \exp\left(-\frac{1}{2}\|\mathbf{q} - \mathbf{k}\|^2\right) \qquad \text{Gaussian;} \tag{3.31}$$

$$\alpha(\mathbf{q}, \mathbf{k}) = 1 \text{ if } \|\mathbf{q} - \mathbf{k}\| \leq 1 \qquad \text{Boxcar;} \tag{3.32}$$

$$\alpha(\mathbf{q}, \mathbf{k}) = \max(0, 1 - \|\mathbf{q} - \mathbf{k}\|) \qquad \text{Epanechikov.} \tag{3.33}$$

There are numerous other options available. All kernels are heuristic in nature and can be adjusted. For example, the width can be fine-tuned, not only globally but also on a per-coordinate basis. Nonetheless, they all result in the following equation for both regression and classification:

$$f(\mathbf{q}) = \sum_i \mathbf{v}_i \frac{\alpha(\mathbf{q}, \mathbf{k}_i)}{\sum_j \alpha(\mathbf{q}, \mathbf{k}_j)}. \tag{3.34}$$

For (scalar) regression with observations $(\mathbf{x}_i, y_i)$ representing features and labels, $\mathbf{v}_i = y_i$ are scalars, $\mathbf{k}_i = \mathbf{x}_i$ are vectors, and the query $\mathbf{q}$ indicates the new location for evaluating f. For (multiclass) classification, one-hot encoding of y_i is used to obtain $\mathbf{v}_i$. A notable feature of this estimator is that it requires no training. Moreover, if the kernel is sufficiently narrowed as the data volume increases, the method is consistent ([111]), meaning it converges to an optimal statistical solution. Let's begin by examining some kernels.

3.3.3 Attention Scoring Function

Let's review the attention function (without exponentiation) from the Gaussian kernel for a moment:

$$a(\mathbf{q}, \mathbf{k}_i) = -\frac{1}{2}\|\mathbf{q} - \mathbf{k}_i\|^2 = \mathbf{q}^\top \mathbf{k}_i - \frac{1}{2}\|\mathbf{k}_i\|^2 - \frac{1}{2}\|\mathbf{q}\|^2.$$

First, note that the final term depends on $\mathbf{q}$ only. As such it is identical for all $(\mathbf{q}, \mathbf{k}_i)$ pairs. Normalizing the attention weights to 1, as is done in (3.30), ensures that this term disappears entirely. Second, note that both batch and layer normalization (to be discussed later) lead to activations that have well-bounded, and often constant, norms $\|\mathbf{k}_i\|$. This is the case, for instance, whenever the keys $\mathbf{k}_i$ were generated by a layer norm. As such, we can drop it from the definition of a without any major change in the outcome.

Last, we need to keep the order of magnitude of the arguments in the exponential function under control. Assume that all the elements of the query $\mathbf{q} \in \mathbb{R}^d$ and the key $\mathbf{k}_i \in \mathbb{R}^d$ are independent and identically drawn random variables with zero mean and unit variance. The dot product between both vectors has zero mean and a variance of d. To ensure that the variance of the dot product still remains 1 regardless of vector length, we use the *scaled dot product attention* scoring function. That is, we rescale the dot product by $1/\sqrt{d}$. We thus arrive at the first commonly used attention function that is used, e.g., in Transformers [105]:

$$a(\mathbf{q}, \mathbf{k}_i) = \mathbf{q}^\top \mathbf{k}_i / \sqrt{d}. \tag{3.35}$$

Note that attention weights α still need normalizing. We can simplify this further via (3.30) by using the softmax operation:

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \mathrm{softmax}(a(\mathbf{q}, \mathbf{k}_i)) = \frac{\exp(\mathbf{q}^\top \mathbf{k}_i / \sqrt{d})}{\sum_{j=1} \exp(\mathbf{q}^\top \mathbf{k}_j / \sqrt{d})}. \tag{3.36}$$

As it turns out, all popular attention mechanisms use the softmax, hence we will limit ourselves to that in the remainder of this chapter.

3.3.4 The Bahdanau Attention Mechanism

When addressing machine translation, an encoder-decoder architecture was developed for sequence-to-sequence learning using two RNNs [104]. Specifically, the RNN encoder converts a variable-length sequence into a fixed-size context variable. Subsequently, the RNN decoder produces the output (target) sequence token by token, relying on the generated tokens and the context variable.

Refer to 3.3 with additional details. Traditionally, in an RNN, all pertinent information about a source sequence is encapsulated into an internal *fixed-dimensional* state representation by the encoder. This state is then utilized by the decoder as the sole source of information for generating the translated sequence. In essence, the sequence-to-sequence model considers the intermediate state as a sufficient statistic of the input string.

While this approach is reasonable for shorter sequences, it becomes impractical for longer ones, such as entire book chapters or even lengthy sentences. Eventually, the intermediate representation will lack the capacity to retain all crucial information from the source sequence. Consequently, the decoder will struggle to translate lengthy and intricate sentences. One of the earliest encounters with this

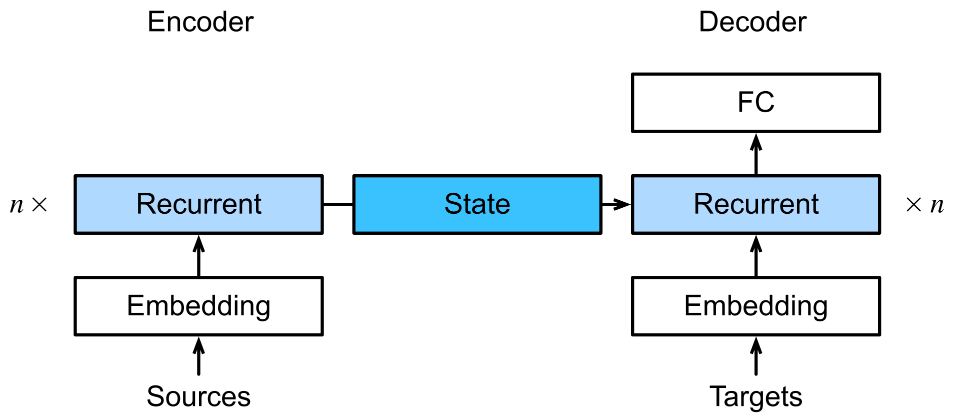


Figure 3.3 Sequence-to-sequence model. The state, as generated by the encoder, is the only piece of information shared between the encoder and the decoder [4].

challenge was documented by [112], who attempted to devise an RNN for generating handwritten text. Given the arbitrary length of the source text, they devised a differentiable attention mechanism to align text characters with the much lengthier pen trace, enabling alignment in only one direction. This concept draws inspiration from decoding techniques in speech recognition, such as hidden Markov models [113].

Building upon the notion of learning to align, [103] introduced a differentiable attention mechanism *without* the constraint of unidirectional alignment. When predicting a token, if not all input tokens are pertinent, the model selectively aligns (or attends) to segments of the input sequence deemed relevant to the current prediction. This alignment is then leveraged to update the current state before generating the next token. Despite its seemingly innocuous description, the *Bahdanau attention mechanism* has arguably emerged as one of the most influential concepts in deep learning over the past decade, spawning architectures like Transformers [105] and numerous related innovations.

3.3.5 Multi-head Attention

In practical applications, when faced with the same set of queries, keys, and values, it might be advantageous for the model to integrate insights from diverse behaviors of the identical attention mechanism. This could involve capturing dependencies across various ranges (e.g., shorter-range versus longer-range) within a sequence. Hence, enabling the attention mechanism to concurrently utilize distinct representation subspaces of queries, keys, and values could prove beneficial.

To achieve this, instead of conducting a singular attention pooling, queries, keys, and values undergo individual linear projections, each learned independently.

These h projected queries, keys, and values are then concurrently inputted into attention pooling. Eventually, the outputs of the h attention poolings are concatenated and further processed through another learned linear projection to generate the ultimate output. This architecture is referred to as *multi-head attention*, where each of the h attention pooling outcomes constitutes a *head* [105]. Employing fully connected layers to implement learnable linear transformations, Figure 3.4 illustrates multi-head attention.

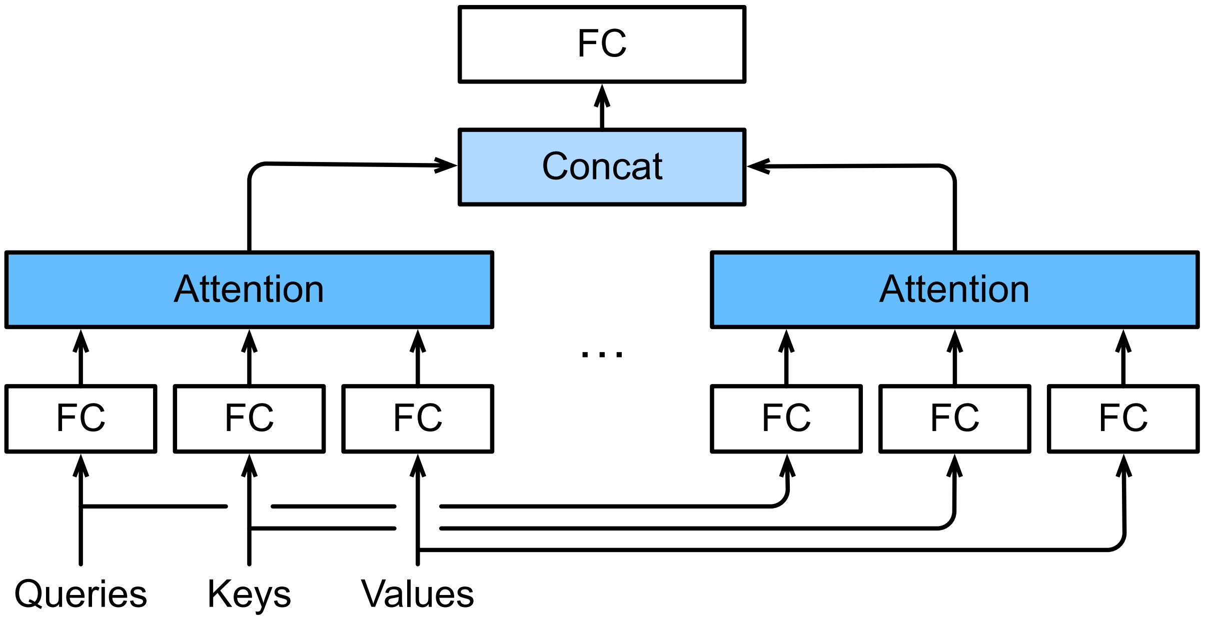


Figure 3.4 Multi-head attention, where multiple heads are concatenated and then linearly transformed [4].

Before detailing the implementation of multi-head attention, it is crucial to establish the mathematical formalism of this model. For a given query $\mathbf{q} \in \mathbb{R}^{d_q}$, key $\mathbf{k} \in \mathbb{R}^{d_k}$, and value $\mathbf{v} \in \mathbb{R}^{d_v}$, each attention head $\mathbf{h}_i$ $(i = 1, \ldots, h)$ is computed as

$$\mathbf{h}_i = f(\mathbf{W}_i^{(q)}\mathbf{q}, \mathbf{W}_i^{(k)}\mathbf{k}, \mathbf{W}_i^{(v)}\mathbf{v}) \in \mathbb{R}^{p_v}, \tag{3.37}$$

where $\mathbf{W}_i^{(q)} \in \mathbb{R}^{p_q \times d_q}$, $\mathbf{W}_i^{(k)} \in \mathbb{R}^{p_k \times d_k}$, and $\mathbf{W}_i^{(v)} \in \mathbb{R}^{p_v \times d_v}$ represent learnable parameters, and f denotes the attention pooling mechanism, such as additive attention or scaled dot product attention. The output of multi-head attention is obtained by another linear transformation using learnable parameters $\mathbf{W}_o \in \mathbb{R}^{p_o \times hp_v}$ applied to the concatenation of h heads:

$$\mathbf{W}_o \begin{bmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_h \end{bmatrix} \in \mathbb{R}^{p_o}. \tag{3.38}$$

Based on this design, each head may attend to different parts of the input, allowing more sophisticated functions than the simple weighted average to be expressed.

3.3.6 Self-Attention and Positional Encoding

In the realm of deep learning, sequences are often encoded using CNNs or RNNs. With attention mechanisms in mind, consider inputting a sequence of tokens into an attention mechanism, where each token has its own set of queries, keys, and values at every step. When determining the representation of a token at the next layer, it can attend (via its query vector) to any other token, based on their key vectors. By computing the compatibility scores between all query-key pairs, a representation can be calculated for each token by forming a weighted sum over the other tokens.

Since every token attends to every other token (in contrast to decoder steps attending to encoder steps), such architectures are commonly referred to as *self-attention* models [114, 105], and alternatively known as *intra-attention* models [115, 116, 117]. This section explores sequence encoding using self-attention, including considerations for the sequence order.

Given a sequence of input tokens $\mathbf{x}_1, \ldots, \mathbf{x}_n$, where each $\mathbf{x}_i \in \mathbb{R}^d$ $(1 \leq i \leq n)$, self-attention produces an output sequence of the same length $\mathbf{y}_1, \ldots, \mathbf{y}_n$, where

$$\mathbf{y}_i = f(\mathbf{x}_i, (\mathbf{x}_1, \mathbf{x}_1), \ldots, (\mathbf{x}_n, \mathbf{x}_n)) \in \mathbb{R}^d \tag{3.39}$$

Following the definition of attention pooling, the code snippet below demonstrates the computation of self-attention using multi-head attention for a tensor with dimensions (batch size, number of time steps or sequence length in tokens, d). The resulting tensor maintains the same shape.

It proves beneficial to compare architectures for transforming a sequence of n tokens into another sequence of equal length, where each input or output token is represented by a d-dimensional vector. Specifically, we examine the architectures of CNNs, RNNs, and self-attention, considering their computational complexity, sequential operations, and maximum path lengths. Sequential operations hinder parallel computation, while shorter paths between any combination of sequence positions facilitate the learning of long-range dependencies within the sequence.

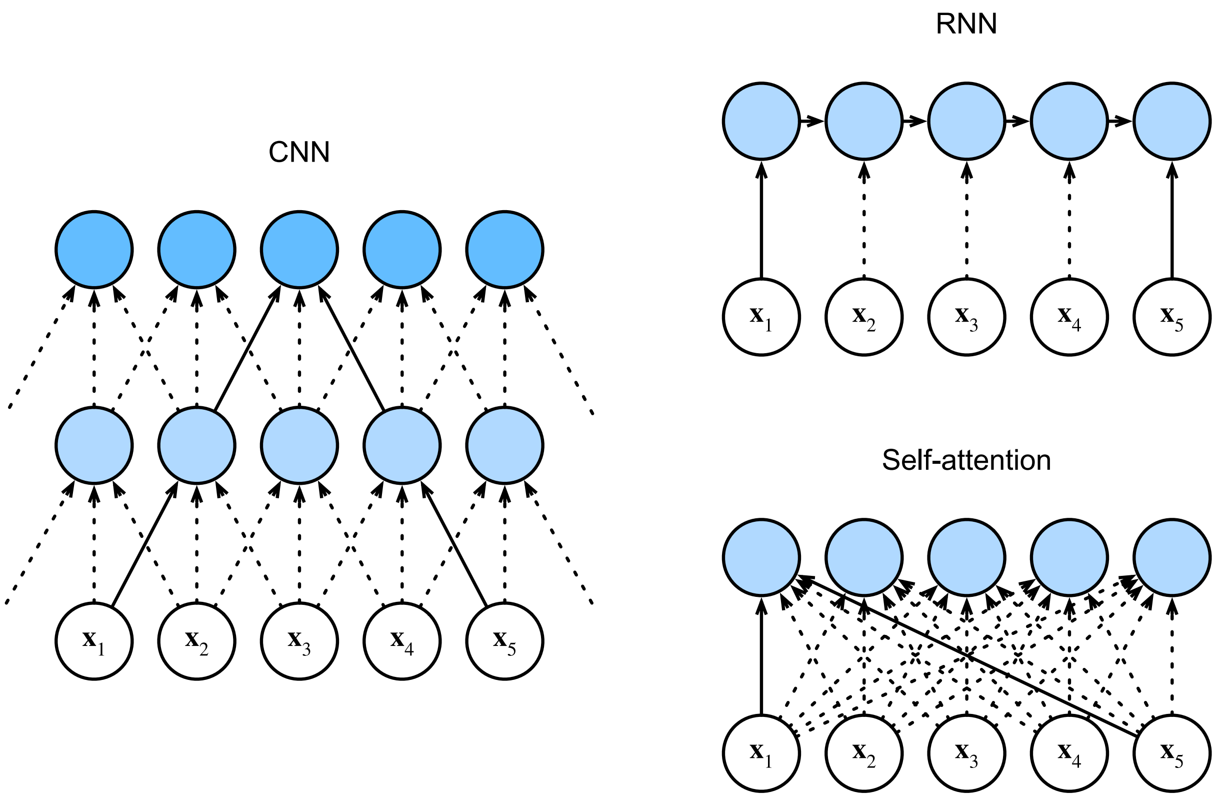


Figure 3.5 Comparing CNN (padding tokens are omitted), RNN, and self-attention architectures [4].

Any textual sequence can be treated as a "one-dimensional image," akin to how one-dimensional CNNs handle local features like n-grams in text. For a sequence of length n, suppose a convolutional layer with a kernel size of k and both input and output channels set to d. The computational complexity of this layer is $\mathcal{O}(knd^2)$. Illustrated in Figure 3.5, CNNs exhibit a hierarchical structure, resulting in $\mathcal{O}(1)$ sequential operations and a maximum path length of $\mathcal{O}(n/k)$. For instance, in Figure 3.5, tokens $\mathbf{x}_1$ and $\mathbf{x}_5$ fall within the receptive field of a two-layer CNN with a kernel size of 3.

When updating RNN hidden states, the multiplication of the $d \times d$ weight matrix and the d-dimensional hidden state has a computational complexity of $\mathcal{O}(d^2)$. With a sequence length of n, the computational complexity of the recurrent layer becomes $\mathcal{O}(nd^2)$. As depicted in Figure 3.5, there are $\mathcal{O}(n)$ sequential operations, which cannot be parallelized, and the maximum path length remains $\mathcal{O}(n)$.

In self-attention, the queries, keys, and values are all $n \times d$ matrices. Considering scaled dot product attention, where an $n \times d$ matrix is multiplied by a $d \times n$ matrix, followed by the multiplication of the resulting $n \times n$ matrix by an

$n \times d$ matrix, the self-attention exhibits a computational complexity of $\mathcal{O}(n^2 d)$. As demonstrated in Figure 3.5, each token directly connects to every other token through self-attention. Thus, computations can be parallelized with $\mathcal{O}(1)$ sequential operations, and the maximum path length is also $\mathcal{O}(1)$.

In summary, both CNNs and self-attention allow for parallel computation, with self-attention boasting the shortest maximum path length. However, the quadratic computational complexity concerning the sequence length renders self-attention impractical for extremely long sequences.

3.4 Segmentation Model Architecture

In the field of computer vision, segmentation models play a crucial role by providing pixel-level understanding of images, which is essential for numerous applications including medical imaging, autonomous driving, and scene understanding. These models aim to partition an image into segments, or regions, that correspond to different objects or parts of objects, thereby enabling a detailed analysis of the visual content. Traditional segmentation approaches relied heavily on handcrafted features and algorithms like thresholding, region growing, and edge detection, which, while effective to some extent, often struggled with complex scenes and varied object appearances.

The advent of deep learning, particularly convolutional neural networks (CNNs), has revolutionized image segmentation by allowing models to learn hierarchical features directly from data, significantly improving accuracy and robustness. Among the most influential architectures is the Fully Convolutional Network (FCN), which replaces fully connected layers with convolutional ones, enabling dense predictions at the pixel level. This architecture has inspired numerous variants and improvements, such as U-Net, originally designed for biomedical image segmentation, which introduces a symmetric U-shaped structure with skip connections that help recover fine details by combining low-level and high-level features.

Another notable advancement is the development of the Mask R-CNN, which extends Faster R-CNN by adding a branch for predicting segmentation masks, thus integrating object detection and instance segmentation into a unified framework. More recent approaches leverage transformer-based models, like the Vision Transformer (ViT) and its derivatives, which capture long-range dependencies through self-attention mechanisms, thereby enhancing the segmentation performance for complex scenes. Additionally, models such as DeepLab employ atrous convolutions (also known as dilated convolutions) to capture multi-scale contextual information without reducing the resolution of feature maps, addressing the challenge of segmenting objects at multiple scales.

Techniques like Conditional Random Fields (CRFs) are often integrated as post-processing steps to refine the segmentation boundaries by enforcing spatial consistency. Moreover, the rise of generative adversarial networks (GANs) has introduced new avenues for semi-supervised and unsupervised segmentation, where the discriminator network helps ensure the plausibility of segmentations even with limited labeled data. The continuous evolution of segmentation models is driven by the ever-growing datasets and computational resources, enabling the training of deeper and more sophisticated networks.

These advancements not only push the boundaries of segmentation accuracy but also expand the applicability of these models to diverse domains, from real-time video segmentation for augmented reality applications to high-precision tissue segmentation in medical diagnostics. As the field progresses, challenges such as handling occlusions, achieving real-time performance, and generalizing to unseen data remain active areas of research, ensuring that segmentation models will continue to be a focal point of innovation in computer vision.

3.4.1 SegNet

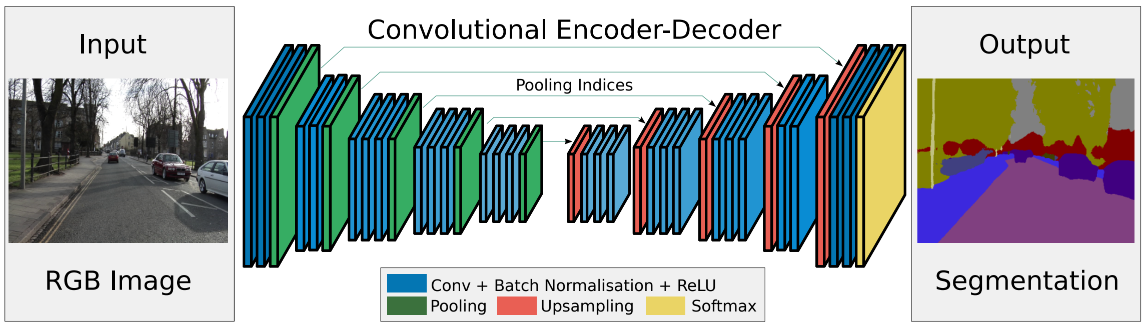


Figure 3.6 An illustration of the SegNet architecture[5]. There are no fully connected layers and hence it is only convolutional. A decoder upsamples its input using the transferred pool indices from its encoder to produce a sparse feature map(s). It then performs convolution with a trainable filter bank to densify the feature map. The final decoder output feature maps are fed to a soft-max classifier for pixel-wise classification.

SegNet comprises an encoder network and a corresponding decoder network, followed by a final pixel-wise classification layer, as depicted in Fig. 3.6. The encoder network encompasses 13 convolutional layers, mirroring the initial 13 layers of the VGG16 network devised for object classification. Consequently, the training process can commence with weights pre-trained for classification on extensive datasets [118]. To maintain higher-resolution feature maps at the deepest encoder output, the fully connected layers are discarded, resulting in a significant reduction in the number of parameters in the SegNet encoder network (from 134M to 14.7M) compared to other recent architectures [119]. Each encoder layer corresponds to a decoder layer, resulting in a decoder network with 13 layers. The ultimate decoder output is fed into a multi-class softmax classifier to generate class probabilities for each pixel independently.

In the encoder network, each convolutional layer convolves with a filter bank to generate a set of feature maps, followed by batch normalization [120] and rectified linear activation (ReLU) $max(0, x)$. Subsequently, max-pooling with a 2×2 window and stride 2 (non-overlapping window) is applied, reducing the output by a factor of 2. Max-pooling achieves translation invariance over minor spatial shifts in the input image, providing a broader input image context for each pixel in the feature map. However, successive layers of max-pooling and sub-sampling sacrifice spatial resolution for enhanced translation invariance, which is detrimental to boundary delineation crucial for segmentation. Hence, it's essential to preserve boundary information in the encoder feature maps before sub-sampling. If memory constraints during inference are not an issue, storing all encoder feature maps (after sub-sampling) is feasible. However, practical applications often face memory limitations. To address this, we propose a more efficient storage method, retaining only the max-pooling indices, i.e., the locations of the maximum feature value in each pooling window, for each encoder feature map. This can be accomplished using 2 bits for each 2×2 pooling window, significantly reducing memory consumption compared to storing feature maps in floating-point precision. Although this lower memory storage incurs a slight loss of accuracy, it remains suitable for practical applications.

In the decoder network, each decoder utilizes the memorized max-pooling indices from the corresponding encoder feature maps to upsample its input feature maps, producing sparse feature maps. This decoding process, illustrated in Fig. 3.7, is followed by convolution with a trainable decoder filter bank to generate dense feature maps, which then undergo batch normalization. Notably, the decoder corresponding to the first encoder, closest to the input image, produces a multi-channel feature map, despite its encoder input having 3 channels (RGB), unlike other decoders in the network. The high-dimensional feature representation at the final decoder output is fed into a trainable softmax classifier, classifying each pixel independently. The softmax classifier output is an image of probabilities with K channels, where K is the number of classes. The predicted segmentation corresponds to the class with the maximum probability at each pixel.

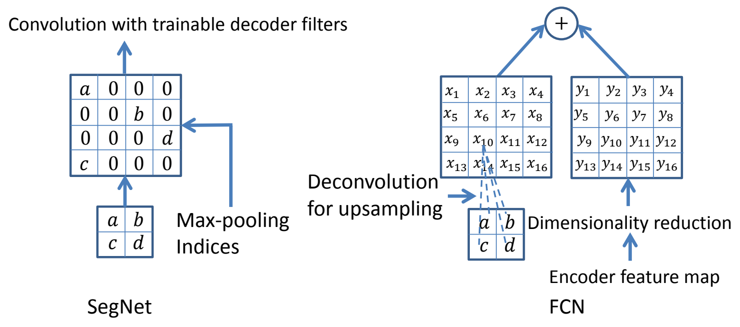


Figure 3.7 a, b, c, d represent values within a feature map[5]. In SegNet, the max-pooling indices are employed for upsampling (without learning) the feature map(s), which are then convolved with a trainable decoder filter bank. Conversely, FCN achieves upsampling by learning to deconvolve the input feature map and adding the corresponding encoder feature map to generate the decoder output. This feature map corresponds to the output of the max-pooling layer (comprising sub-sampling) in the corresponding encoder. Notably, FCN does not incorporate trainable decoder filters.

3.4.2 UNET

The architecture of the network is depicted in Fig. 3.8. It comprises a contracting path (left side) and an expansive path (right side). The contracting path adheres to the typical convolutional network structure. It involves iteratively applying two 3x3 convolutions (without padding), each succeeded by a rectified linear unit (ReLU), and a 2x2 max-pooling operation with a stride of 2 for downsampling. With each downsampling iteration, the number of feature channels doubles. Each step in the expansive path entails upsampling the feature map, followed by a 2x2 convolution (termed "up-convolution") to halve the number of feature channels, concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU. Cropping is necessary due to the loss of border pixels in every convolution. In the final layer, a 1x1 convolution is employed to map each 64-component feature vector to the desired number of classes. Overall, the network comprises 23 convolutional layers.

To ensure smooth tiling of the output segmentation map, it is crucial to choose the input tile size such that all 2x2 max-pooling operations are applied to a layer with an even x- and y-size.

The U-Net architecture revolutionized biomedical image segmentation and has since been applied across various domains. Presented in the 2015 paper "U-Net: Convolutional Networks for Biomedical Image Segmentation" by Olaf Ronneberger, Philipp Fischer, and Thomas Brox, it was tailored to achieve precise segmentation with limited training data, a common scenario in medical imaging. Its hallmark U-shaped design integrates down-sampling and up-sampling paths,

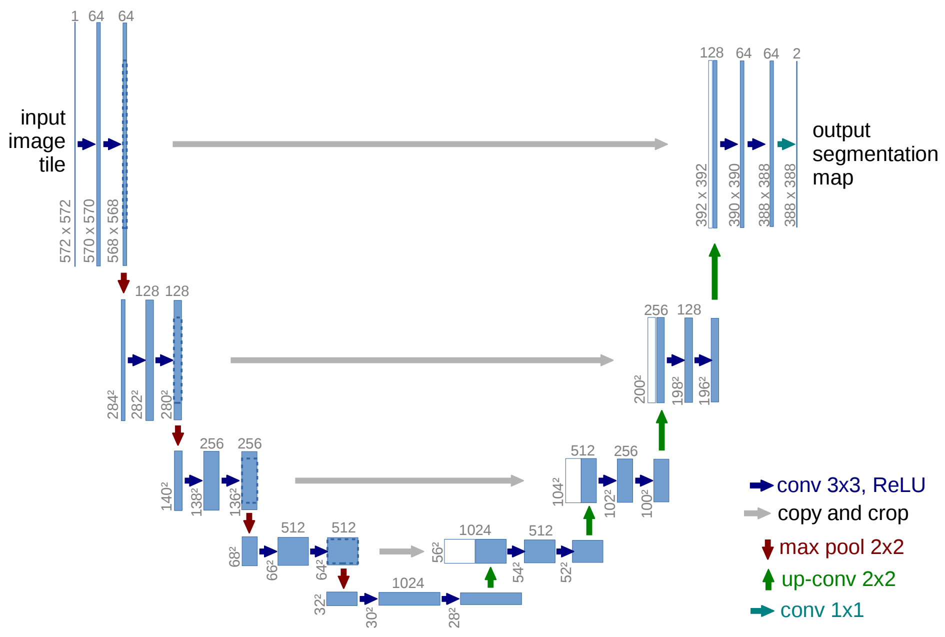


Figure 3.8 U-net architecture (illustrated for a resolution of 32x32 pixels at the lowest scale). Each blue rectangle represents a multi-channel feature map, with the number of channels indicated at the top. The x-y dimensions are displayed at the bottom left corner of each rectangle. White rectangles indicate duplicated feature maps. Arrows indicate the various operations.

facilitating context capture and accurate localization.

Comprising a contracting path and an expansive path, the U-Net model mirrors traditional convolutional networks. The contracting path employs convolutional and pooling layers to decrease image dimensions while enhancing feature map depth. Each step involves dual convolutional layers with ReLU activations followed by max-pooling, halving the image size.

Conversely, the expansive path employs up-convolutions to restore spatial dimensions and fuse high-resolution features from the contracting path via skip connections. These connections allow the model to leverage fine-grained information from earlier layers. Concatenating feature maps from both paths ensures consideration of global and local features during reconstruction, yielding high-resolution segmentation maps with precise object delineation.

A key advantage of U-Net is its efficacy with limited datasets, crucial in medical imaging where large annotated datasets are scarce due to privacy concerns and annotation complexity. U-Net addresses this challenge through data augmentation techniques such as elastic deformations and intensity variations, bolstering generalization to unseen data. Its symmetrical architecture and skip connections enable robust performance with scant training samples.

U-Net has excelled in diverse biomedical tasks including cell tracking, liver segmentation, and brain tumor delineation. Its success spurred variants like 3D U-Net for volumetric data and attention U-Net incorporating attention mechanisms for complex scenes.

Beyond biomedicine, U-Net finds applications in satellite image segmentation, autonomous driving, and agriculture. In satellite imaging, it segments land cover types and identifies changes over time. In autonomous driving, it delineates road lanes, vehicles, and pedestrians. In agriculture, it aids in crop segmentation, disease detection, and yield estimation, enhancing farming efficiency.

CHAPTER 4. METHODOLOGY

4.1 Problem Statement

Consider $(x, y) \sim P(\mathcal{X}, \mathcal{Y})$, where $x \in \mathbb{R}^{N\times C\times H\times W}$ and $y \in \mathbb{R}^{N\times L\times H\times W}$ are input data and corresponding ground truth, respectively. C, L are the number of image channels and categories, respectively. N, H, and W are the total number of training samples, height, and width of the image, separately. The segmentation problem is represented in Eq. (4.40).

$$\mathcal{L}(x, y) = \frac{1}{B}\sum_{i=0}^{B-1}\sum_{l=0}^{L-1}\mathcal{L}_l(x_i, y_i), \tag{4.40}$$

where $\mathcal{L}_l(x_i, y_i) = -\log\left(\frac{\exp\{\hat{x}_i^l\}}{\sum_{l'=0}^{L-1}\exp\{\hat{x}_i^{l'}\}}\right)y_i^l$ denotes the class-wise loss on sample $x_i \in \mathbb{R}^{C\times H\times W}$ and its corresponding ground truth $y_i \in \mathbb{R}^{L\times H\times W}$. $\hat{x}_i^l, y_i^l \in \mathbb{R}^{H\times W}$ are the predicted mask and the ground-truth of channel l (which represents class l), respectively.

4.2 Imbalance among label masks

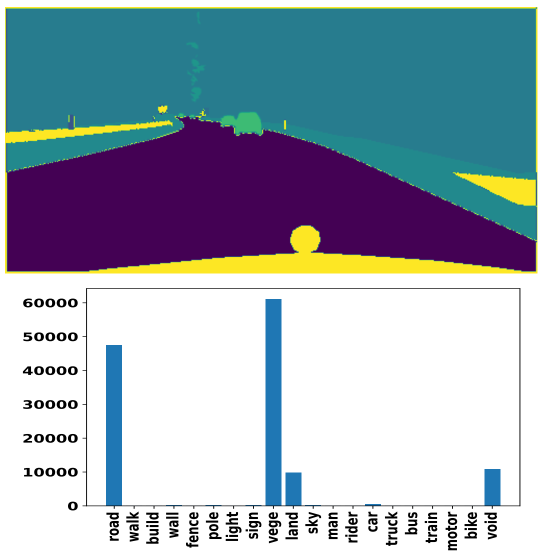
One major challenge in image segmentation is the class imbalance in label masks (see Fig. 4.1). Larger masks contribute more significantly to the loss of function than smaller masks, leading to a bias towards dominant classes. Specifically, we come over the class-wise loss component, which can be represented as:

$$\begin{aligned}\mathcal{L}_l(x_i, y_i) &= -\sum_{j=0}^{HW}\log\left(\frac{\exp\{\hat{x}_{i,j}^l\}}{\sum_{l'=0}^{L-1}\exp\{\hat{x}_{i,j}^{l'}\}}\right)y_{i,j}^l \\ &= -\sum_{j=0}^{HW}\log\left(\frac{\exp\{\hat{x}_{i,j}^l\}}{\sum_{l'=0}^{L-1}\exp\{\hat{x}_{i,j}^{l'}\}}\right)\mathbb{I}(y_{i,j}^l = 1) = S_i^l \times \bar{\ell}_l(x_i, y_i),\end{aligned} \tag{4.41}$$

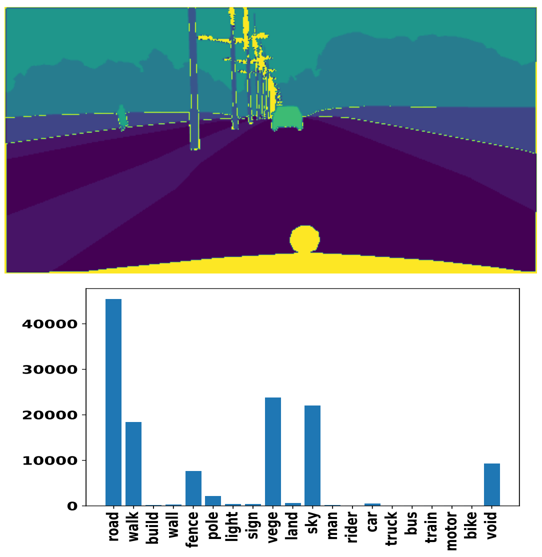
where S_i^l and $\bar{\ell}_l(x_i, y_i)$ denote the size of label l mask and the cross-entropy value on class l for image i, respectively, where $S_i^l = \sum_{j=0}^{HW}\mathbb{1}(y_{i,j}^l = 1)$. While the traditional approach is rooted in classification problems, in segmentation tasks, the loss is adjusted based on the mask size S_i^l. Consequently, to ensure uniformity in gradient magnitude, we diminish the loss by the label mask size of each instance. This adjustment guarantees that all class-specific loss pixels receive equal consideration within the collective loss function.

4.3 Pixel-wise Adaptive Traning with Loss Scaling

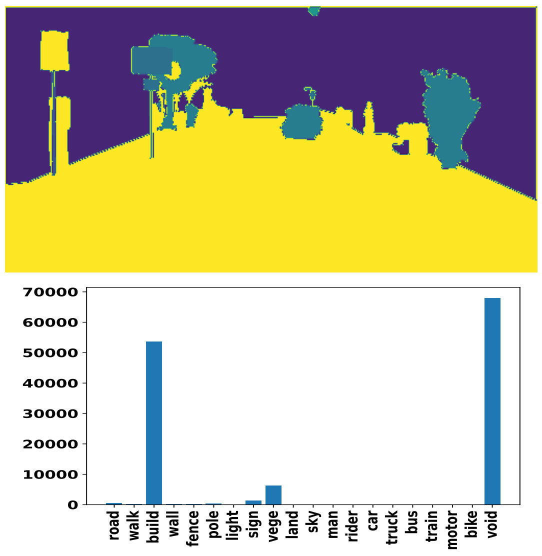
The summary of the methodology is shown in Fig. 4.2. To design an adaptive pixel-wise loss scaling, we first decompose the conventional segmentation function



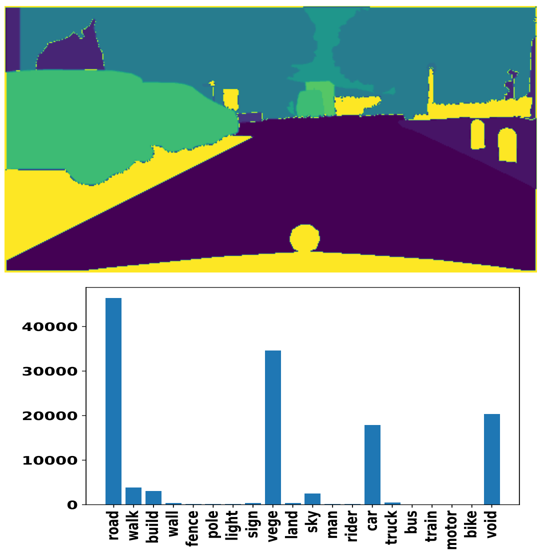
(a)



(b)



(c)



(d)

Figure 4.1 Quantitative analysis on the imbalance in mask size among classes. The vertical axis illustrates the mask size calculated by the total number of pixels, which are associated with the corresponding mask. The horizontal axis shows different masks that potentially appear in the ground truth. 4.1a) While road and vegetation take 50000 pixels and 60000 pixels, respectively, cars account for around 1000 pixels. 4.1b) Not only road and vegetation but also walking and sky take roughly 99% proportion, compared to cars. 4.1c) Void and build take nearly 70000 pixels and 60000 pixels, under 1000 pixels are accounted by sign, and no car appears. 4.1d) Cars take roughly 20000 pixels, though their masks' sizes are much bigger than the ones in Figs. 4.1a and 4.1b.

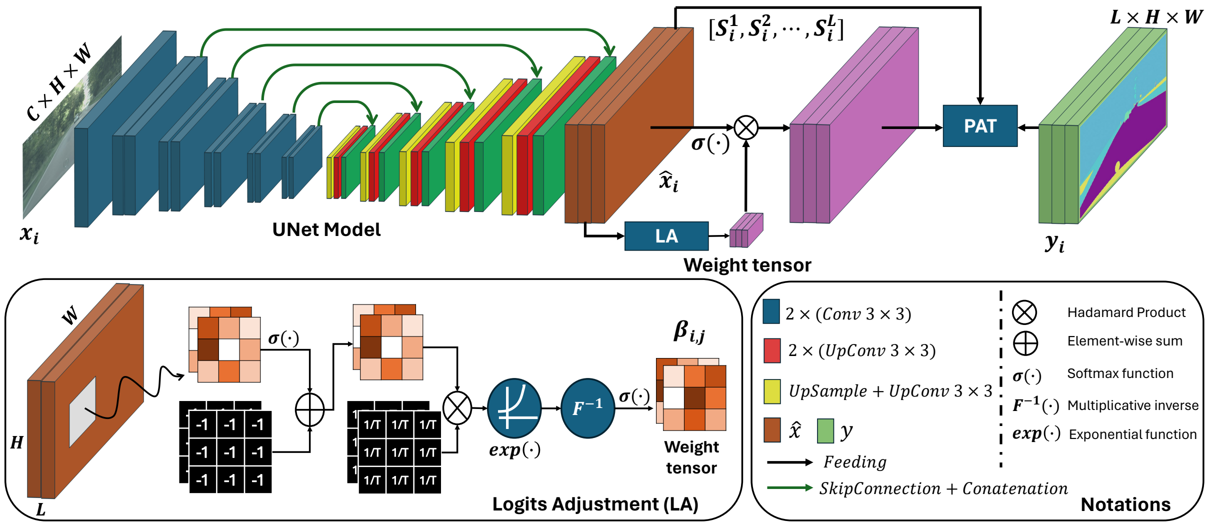

$C \times H \times W$
x_i
UNet Model
$\hat{x}_i$
$\sigma(\cdot)$
$[S_i^1, S_i^2, \cdots, S_i^L]$
PAT
$L \times H \times W$
y_i
LA
Weight tensor
W
H
L
$\sigma(\cdot)$
$exp(\cdot)$
F^{-1}
$\beta_{i,j}$
Weight tensor
Logits Adjustment (LA)
$2 \times (Conv\ 3 \times 3)$
$2 \times (UpConv\ 3 \times 3)$
$UpSample + UpConv\ 3 \times 3$
$\hat{x}$ y
Feeding
SkipConnection + Conatenation
Hadamard Product
Element-wise sum
$\sigma(\cdot)$ Softmax function
$F^{-1}(\cdot)$ Multiplicative inverse
$exp(\cdot)$ Exponential function
Notations

Figure 4.2 Overall methodology. **1) Training procedure**: an input image x_i is fed into a typical encoder-decoder model architecture to produce output $\hat{x}_i$. This prediction's logits are then adjusted to create a weight tensor, which has the same size as $\hat{x}_i$. The normalized $\hat{x}_i$ is multiplied by the weight tensor to balance the dominant logits. Finally, the loss value is obtained using the proposed PAT loss function. **2) Logits Adjustment**: The logits vector is normalized by the Softmax function, added by a tensor of -1, and scaled by the exponential function to find the inverse dominant coefficients $\beta_{i,j}$. Then, these coefficients are normalized into a range of $[0, 1]$ to form the weight tensor.

into a pixel-wise function (refer to Eq. (4.42)), which is derived from Eq. (4.40).

$$\mathcal{L}(x, y) = -\frac{1}{B}\sum_{i=0}^{B-1}\sum_{j=1}^{HW}\left[\sum_{l=0}^{L-1} y_{i,j}^l \log\left(\frac{\exp\{\hat{x}_{i,j}^l\}}{\sum_{l'=0}^{L-1}\exp\{\hat{x}_{i,j}^{l'}\}}\right)\right], \tag{4.42}$$

where $\hat{x}_{i,j}^l$ is the logits prediction of sample i at pixel j with regard to category l.

In segmentation problems, the class is considered by a composition of many pixels over the masks. We hypothesize that the learning in each class may occur diversely according to the classification of different pixels. Therefore, we propose the pixel-wise adaptive (PAT) loss via the pixel-wise adaptive coefficient set $\beta_{i,j} \in \mathbb{R}^L = \left[\beta_{i,j}^0, \beta_{i,j}^1, \cdots, \beta_{i,j}^{L-1}\right]$.

$$\mathcal{L}(x, y) = -\frac{1}{B}\sum_{i=0}^{B-1}\sum_{j=1}^{HW}\left[\sum_{l=0}^{L-1} \beta_{i,j}^l \times \frac{y_{i,j}^l}{S_i^l} \log\left(\frac{\exp\{\hat{x}_{i,j}^l\}}{\sum_{l'=0}^{L-1}\exp\{\hat{x}_{i,j}^{l'}\}}\right)\right]. \tag{4.43}$$

Our key idea is to control the PAT loss using $\beta_{i,j}$. Essentially, $\beta_{i,j}$ represents a tensor with dimensions identical to the logits $\hat{x}_{i,j}$. Through the pixel-wise multiplication, $\beta_{i,j}$ effectively modulates the pixel-wise loss components. Calculation of

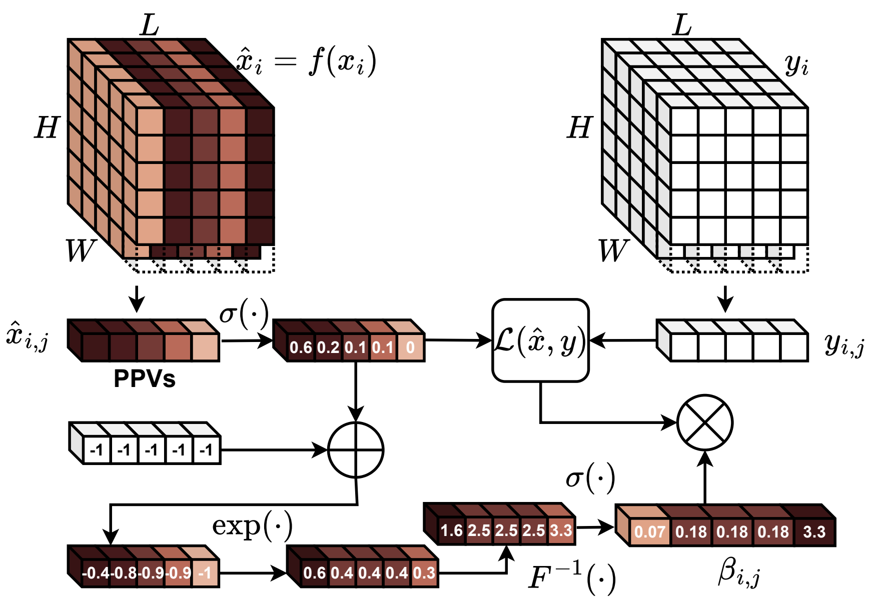


Figure 4.3 Illustration the PAT procedure of adjusting the logits' value to tackle the imbalance in dominant probability from categories whose big mask size.

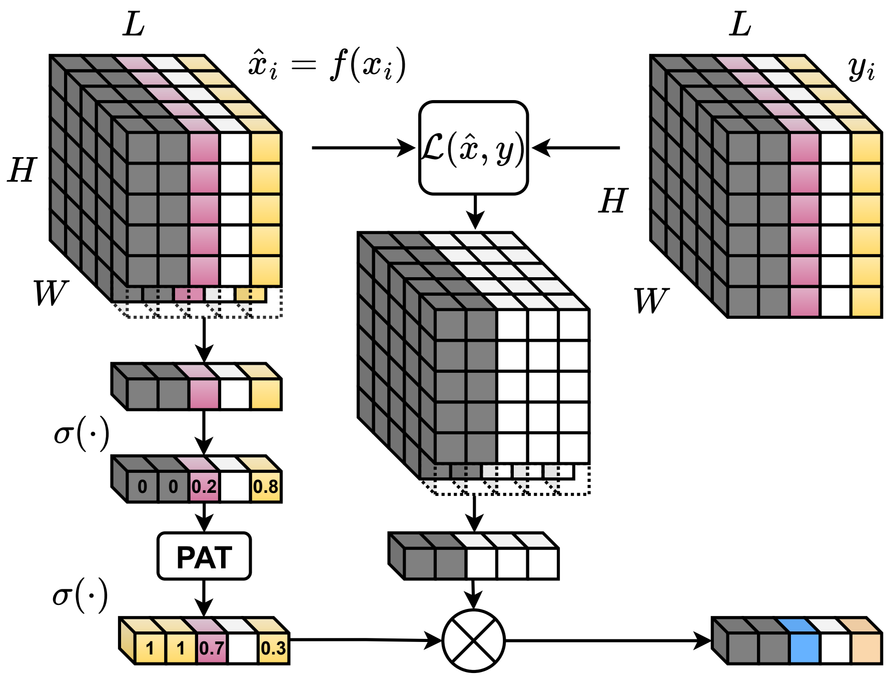


Figure 4.4 The process of adaptive gradient scaling in PAT. Specifically, the channels with no mask can easily be adapted. Therefore, the problem of adaptive gradient scaling can be reduced to two cases.

$\beta_{i,j}$ based on the logits $\hat{x}_{i,j}$ is as follows:

$$\beta_{i,j} = \frac{1}{\exp\{(p(\hat{x}_{i,j}) - 1 + \epsilon)/T\}}, \tag{4.44}$$

where $p(\hat{x}_{i,j}) = \left[\exp\{\hat{x}^{l}_{i,j}\}/\sum_{l'=0}^{L-1}\exp\{\hat{x}^{l'}_{i,j}\}\right]_{l=0}^{L-1}$ indicates the output of the softmax

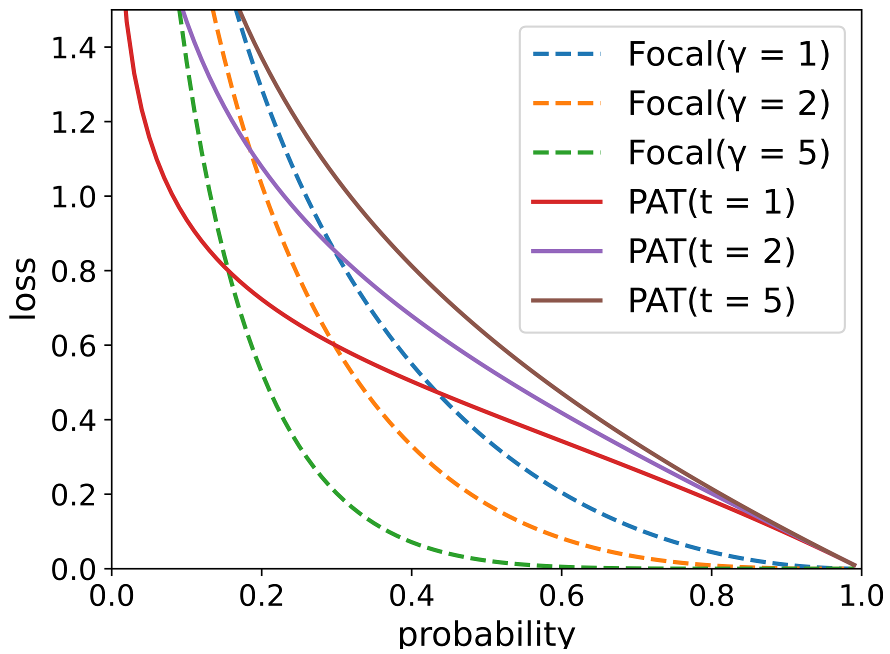


Figure 4.5 In addition to Fig. 4.3, Fig. 4.5 shows the difference in scaling coefficient between PAT and Focal[7], that PAT (smooth lines) (i) puts a higher weight on low confidence pixel and (ii) keeps low scaling coefficients for high confidence pixels. Otherwise, Focal (dash line), puts zero scalarization on well-classified pixels that may cause forgetfulness of frequent or big mask size categories.

activation which normalize the logits vector elements into probability range of $[0, 1)$. We define T as a temperature coefficient and ϵ as an arbitrary constant. By tuning the $\beta_{i,j}$ according to each pixel-wise vector $\hat{x}_{i,j} = \{\hat{x}^l_{i,j} |\ l \in \{0, \ldots, L-1\}\}$, our proposed coefficient hinges on two key concepts: firstly, equalizing the loss value across various logits, and secondly, preventing negative transfer in well-classified results. Further analysis is presented in Section 5.2. Additionally, in Eq. (4.43), we normalize the loss across all components by dividing by S^l_i. This approach allows us to penalize loss components that have a dominant size relative to others, thereby facilitating the homogenization of class-wise gradient magnitudes.

5.1 Generalization of PAT on special case

In numerous scenarios, $\beta_{i,j}$ may encounter near-zero logit values, potentially causing value explosions. This occurrence can lead to computational errors in practice. To mitigate this issue, we introduce temperature coefficients T and a constant ϵ, effectively preventing the value explosion of $\beta_{i,j}$. Moreover, near-zero logit values are often associated with the absence of label masks. Consequently, through the computation of the joint loss function, pixel-level loss values are frequently nullified to 0 rather than undergoing explosion (see Fig. 4.4).

5.2 Analysis on the PAT to the logits imbalance

Experimentally, Focal loss performance is lower than current approaches, through its simple and optimized implementation. To have a comprehensive understanding of PAT robustness to the imbalance rare object segmentations, we compare the loss value of PAT and Focal[7] at different logit probabilities in Fig. 4.5, yielding two significant observations.

Table 5.1 Adaptive loss value comparison between Focal loss and PAT loss functions with different adaptive coefficients: $\gamma \in \{2, 5\}$ and $T \in \{2, 5\}$. According to the analysis, when the logits prediction approach 1 indicates a well-classified case, the adaptive loss value of Focal loss is zero, which potentially makes the gradient trajectory unstable.

$p(\hat{x}_{i,j})$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Focal ($\gamma = 2$)	1.87	1.03	0.59	0.33	0.17	**0.08**	**0.03**	**0.01**	**0.0**
PAT ($T = 2$)	1.47	1.08	0.85	0.68	0.54	0.42	0.31	0.2	0.1
Focal ($\gamma = 5$)	1.36	0.53	0.2	0.07	0.02	**0.01**	**0.0**	**0.0**	**0.0**
PAT ($T = 5$)	1.92	1.37	1.05	0.81	0.63	0.47	0.34	0.21	0.1

Firstly, by parameterizing the loss function with the PAT scaling factor, we observe more balanced learning across classes with varying logit probabilities. Secondly, in contrast to Focal, we notice that losses associated with high-probability logits are zero-weighted (refers to Table 5.1). This phenomenon can prevent positive transfer on well-classified samples. In comparison, the PAT-scaling parameterized loss function fosters equitable learning while maintaining loss information for high-probability logits, thus allowing the training process to retain valuable knowledge from well-classified samples.

5.3 Analysis on time and space complexity

Table 5.2 theoretically suggests that traditional Cross-Entropy, Focal, and PAT have the lowest complexity in both time and space, compared to LDAM and BLV, which require extensive supporting tensors (Δ_y, $\delta(\sigma)$, and c) to manipulate margin probability distribution. To conclude, PAT fulfills all requirements indicated in Section . PAT can tackle long-tailed rare object segmentation, especially in detecting objects with a small portion accounting in the mask while maintaining the performance of high-confidence classes(refers to Section 4.3, Section 5.2, and Figure 4.5). PAT acquired low-cost computation compared to the current state-of-the-art by integrating a simple weighting mechanism requiring neither further supporting tensors nor calculation on them.

Table 5.2 Theoretical time ($\mathcal{O}$) and space ($\mathcal{V}$) complexity comparison between state-of-the-arts and PAT. Denote $\mathcal{F}$ as loss formula, γ is scale coefficient of Focal[7], z_y, z_i are logits of channel y and i, respectively, σ is standard deviation for BLV[8], and δ is a distribution generator.

Method	$\mathcal{F}$	$\mathcal{O}$	$\mathcal{V}$
CE	$-\log(p(x))$	$\mathcal{O}(BCHW)$	$\mathcal{O}(BCHW)$
Focal	$-(1-p(x))^\gamma \log(p(x))$	$\mathcal{O}(BCHW)$	$\mathcal{O}(BCHW)$
LDAM	$-\log(\frac{\exp(z_y-\Delta_y)}{\sum_i \exp(z_i-\Delta_i)})$	$\mathcal{O}(2BCHW)$	$\mathcal{O}(2BCHW)+\mathcal{O}(2C)$
BLV	$-\log(\frac{\exp(z_y+c_y\delta(\sigma))}{\sum_i \exp(z_i+c_i\delta(\sigma))})$	$\mathcal{O}(2BCHW)$	$\mathcal{O}(3BCHW)+\mathcal{O}(2C)$
PAT	$-\exp\{(1-p(x))/T\}\log(p(x))$	$\mathcal{O}(BCHW)$	$\mathcal{O}(BCHW)$

5.4 Analysis on the PAT to the gradient magnitude homogenization

Magnitude differences of the gradients across tasks may lead to a subset of tasks dominating the total gradient, and therefore to the model prioritizing them over the others [121]. This phenomenon happens obviously in the segmentation task. To have a comprehensive understanding, from Eq. (4.41) we consider the gradient across classes as follows:

$$\nabla\mathcal{L}_l(x,y) = S_i^l \cdot \nabla\bar{\ell}_l(x_i,y_i). \tag{5.45}$$

Therefore, the gradient norm proportion between classes l and l' is as follows:

$$\frac{\|\nabla \mathcal{L}_l(x, y)\|}{\|\nabla \mathcal{L}_{l'}(x, y)\|} = \frac{S_i^l \cdot \|\nabla \bar{\ell}_l(x_i, y_i)\|}{S_i^{l'} \cdot \|\nabla \bar{\ell}_{l'}(x_i, y_i)\|} \tag{5.46}$$

As the mask size S_i^l becomes divergent across classes in one task, the gradient magnitude is diverse, thereby biasing the learning towards the subset of classes with dominant mask sizes. By applying the normalization via mask size S_i^l, as mentioned in Eq. (4.43), we can have the model update focus on the labels' canonical loss $\bar{\ell}_l(x_i, y_i)$. This approach bears resemblance to gradient magnitude normalization techniques discussed in [121, 122], facilitating more balanced contributions among classes during learning. Nevertheless, by normalizing directly according to the mask size, we can notably decrease computational complexity compared to utilizing gradient norms, as seen in the previously mentioned studies.

We conducted experiments on three popular datasets: OxfordPet[1], CityScapes[2], and NyU[3] whose frequency of classes is considerably sample-wise imbalanced (refers to Section 6.1). The training, validating, and testing ratios are 0.8, 0.1, and 0.1, respectively. We compare PAT with Focal[7], Class Balance Loss (CB)[123], the combination of CB and Focal (CBFocal)[123], Balance Meta Softmax (BMS)[124], Label Distribution Aware Margin Loss (LDAM)[16], and Balance Logits Variation (BLV)[8] evaluated by mean Intersection over Union (mIoU %), pixel accuracy (Pix Acc %), and Dice Error (Dice Err). The number of rounds of all experiments is fixed to 30000 rounds.

We tuned the hyperparameter of each loss function to find the best case. Specifically, 1) Focal and the combination of Class Balance and Focal are trained with different values of $\gamma \in \{0.5, 1, 2, 3, 4, 5\}$[7, 123]. 2) In the LDAM, we set the parameter μ of 0.5 as the default setting in[125] and trained with different scale $s \in \{10, 20, 30, 50\}$. 3) For the BLV, we applied types of distribution: Gaussian, Uniform, and Xaviver along with different standard deviation $\sigma \in \{0.5, 1, 2\}$. 4) Hyper-parameter tuning is conducted on PAT with values of $T \in \{5, 10, 20, 50\}$ (refer to Section 6.2.2).

6.1 Comparisons to state-of-the-arts (SOTA)

In Table 6.1, the PAT outperforms the others in almost all settings from 0.09% to 2.2% in mIoU and from 0.07% to 0.36% in pix acc, respectively. While the dice error of BLV and PAT are identically 0.23 in the Oxford dataset, PAT's dice error in CityScapes and NyU datasets (two higher number of categories dataset) decreased around 0.02 and 0.39, respectively. Regarding Fig. 6.1, models trained by PAT can segment objects with mIoU of 76.22% and pix acc of 85.80%, increasing 2.2% and 0.36%, respectively. In detail, these figures illustrate visually how PAT can balance the class-wise consideration.

First, PAT can enhance models on detecting long-tailed rare objects by putting a higher weight on the class-wise loss function (refer to Figs. 4.3, 4.5). In the second column of Fig. 6.1 (Munich domain), under a long-tailed scenario where most parts of the image are road and sky, while the building owns just a small proportion, the model trained by PAT can segment fully the road and sky as well as most of the building. This phenomenon is also true with models trained on OxfordPetIII (refer to Fig. 6.2).

Table 6.1 Overall Performance of 8 baselines (i.e. Vanilla Softmax, Focal, Class Balance Loss, Class Balance Focal) and proposed method among three different scenarios including OxfordPetIII, CityScape, and NyU datasets. In detail, the bold number indicates the best performance, while ↑ and ↓ show "higher is better" and "lower is better", respectively. Note that all experiments shown in this table are trained using SegNet[5] architecture.

Method	Dataset								
	OxfordPetIII[1]			CityScape[2]			NyU[3]		
	mIoU↑	Pix Acc↑	Dice Err↓	mIoU↑	Pix Acc↑	Dice Err↓	mIoU↑	Pix Acc↑	Dice Err↓
CE	76.14	91.21	**0.23**	73.83	85.31	0.66	18.05	53.50	1.80
Focal ($\gamma = 2$)	75.76	91.17	0.30	74.02	85.44	0.56	15.14	51.07	1.75
CB	76.60	90.90	0.20	72.26	81.33	0.69	18.56	52.17	1.94
CBFocal ($\gamma = 2$)	76.02	90.54	0.26	71.17	80.70	0.72	17.31	50.46	1.76
BMS	13.22	25.45	1.28	8.15	11.80	3.08	12.27	22.84	2.40
LDAM ($\mu = 0.5$, $s = 20$)	75.43	90.97	0.78	74.80	85.20	2.27	19.59	52.59	2.30
BLV (Gaussian, $\sigma = 0.5$)	76.24	91.22	**0.23**	74.21	85.37	0.53	18.37	52.62	1.90
PAT (Ours) ($T = 20$)	**76.69** ↑ **0.09**	**91.28** ↑ **0.07**	**0.23**	**76.22** ↑ **2.2**	**85.80** ↑ **0.36**	**0.51** ↓ **0.02**	**21.41** ↑ **2.85**	**55.57** ↑ **2.07**	**1.36** ↓ **0.39**

Second, as aforementioned, the proposed method can tackle underlying issues in detecting long-tailed rare objects by using scalarization coefficients without forgetting the well-classified categories (refer to Fig. 4.5). Visualization performance in Berlin and Leverkusen (refer to Fig. 6.1), are two typical examples. Other SOTAs tend to misclassify the sky and the road, even though the sky and road account for a considerable proportion. This phenomenon is not an exception, which also appears in the Leverkusen example, where other SOTAs misclassify car masks, which take 40% in mask size of the whole portion.

6.1.1 OxfordPetIII Dataset[1].

Oxford dataset contains 37 categories of dogs and cats with roughly 200 images for each type. The mask ground truth of each image includes three classes: background, boundary, and main body of the animals. All images are collected with a high resolution of 640×340 pixels, which is then resized to 256×256 pixels. In this dataset, the biggest obstacle is to segment the boundary of the animal which is very small and not easily distinguishable. In training the image is preprocessed by diving the max value of that image. The labels, on the other hand, are converted into tensors of one-hot vectors at pixel levels.

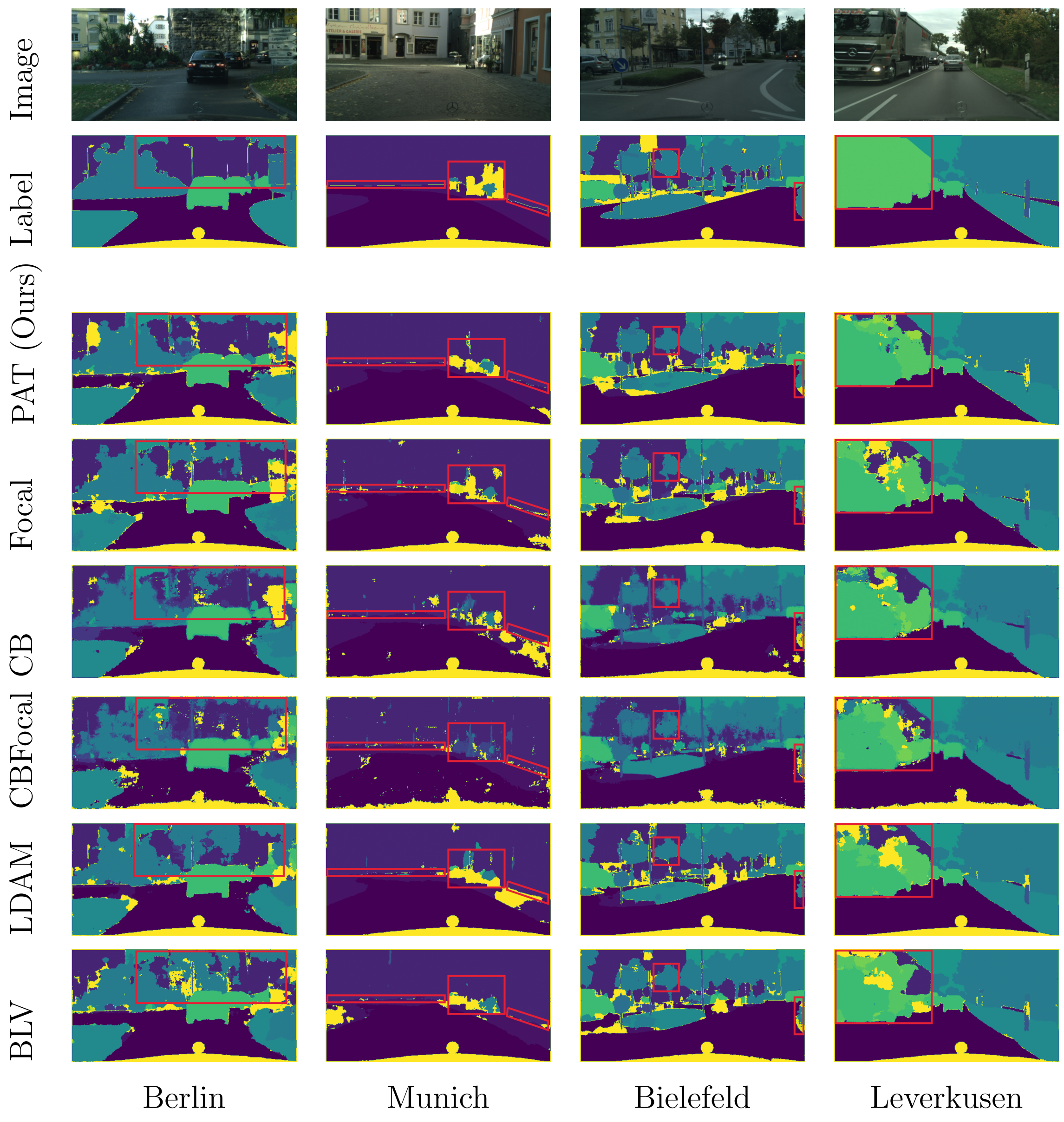


Figure 6.1 Segmentation visualization of models trained by the proposed method PAT and other baselines on the CityScapes dataset. The two first rows are the original image on the test set and its corresponding ground truth, respectively. The third row illustrates the performance in visualization of the proposed method. We sample four typical examples from four corresponding different domains containing Berlin, Munich, Bielefeld, and Leverkusen.

6.1.2 CityScapes Dataset[2].

CityScapes dataset contains roughly 5000 images with a high resolution of 1024×512 pixels, Note that adjusting the image size does not change the proportions among classes. The ground truth in this dataset is labeled in fine mode, in which there are no boundaries among classes. This dataset faced a big issue in the imbalance among classes (refers to Fig. 4.1). There are potentially many categories that only take a small account of the mask as well as a large number of ones that are not included.

6.1.3 NyU Dataset[3].

The NyU dataset contains roughly 1000 samples, with 1000 different categories. Each sample contains an image and label whose size of 340×256 pixels. As in the CityScapes dataset, NyU also faces a big challenge in segmenting long-tailed rare objects. In the preprocessing stage, the image is normalized into the range of $[0, 1]$, and its corresponding ground truth is converted into a tensor of one-hot vectors at pixel levels.

6.2 Ablation studies

6.2.1 Model integration analysis.

To guarantee the proposed method's adaptability and independence in different deep architecture designs[16, 126, 22, 17, 18, 19], we analyze the performance on 3 additional different model structures containing UNet[6], Attention UNet[127] (AttUNet), Nested UNet[128] (UNet++), DeepLabV3[129] (DLV3), and DeepLabV3+[10] (DLV3+). The hyperparameter settings are presented in Table 6.1. The quantitative performance results of this ablation test are shown in Table 6.2. In detail, we compare the performance among methods conducted in each model. The corresponding colors of UNet, AttUNet, UNet++, DLV3, and DLV3+ are green, blue, yellow, red, and purple, respectively, which indicate the best cases.

According to Table 6.2, PAT outperforms the other baselines in CityScapes experiments, which is the highest number of classes dataset. Especially in Unet++ experiments, the mIoU and the pix acc of CityScapes are above 75% and 85%, compared to under the two aforementioned values in BLV[8] and LDAM[125]. Although the Class-Balance loss function tends to work well with the OxfordPetIII dataset (3 distinct categories), which reaches 78.71% and 76.60% of mIoU (refer to Tables 6.16.2) trained by UNet and SegNet, respectively, CB exhibit low performance on datasets with higher number of categories.

LDAM and BLV, otherwise, achieve higher performance as opposed to PAT from 0.01% to 0.04% in the NyU dataset. In detail, LDAM achieves 55.22% of pix acc using AttUNet and BLV reaches 54.67% and 54.93% of pix acc using UNet and UNet++, respectively. However, performance in mIoU of PAT outperforms LDAM and BLV from 1% to 3% (refer to Table 6.2). mIoU is more crucial than pix acc, which not only indicates the correctness but also the completeness of segmented objects[130]. PAT, otherwise, outperforms other baselines in both the CityScapes dataset and the NyU dataset on various types of model architecture, which achieve

Table 6.2 Quantitative comparisons between baselines and the proposed method in three datasets and three model architecture designs: UNet, Attention UNet, and Nested UNet whose encoder is ResNet101[9]. Note that the green, blue, yellow, red and purple colors indicate the outperforming cases trained on UNet, AttUNet, UNet++, DLV3, and DLV3+ respectively.

Method	Model	Encoder	OxfordPetIII		CityScapes		NyU	
			mIoU↑	Pix Acc↑	mIoU↑	Pix Acc↑	mIoU↑	Pix Acc↑
	UNet		78.61	92.40	74.08	83.63	15.36	50.97
	AttUnet		78.83	92.52	73.28	87.33	16.46	51.35
CE	UNet++	ResNet101	78.18	92.30	72.91	82.87	16.48	51.42
	DLV3		79.06	93.72	77.42	92.36	21.74	57.22
	DLV3+		79.82	94.23	77.73	92.94	22.34	57.86
	UNet		78.27	92.24	73.53	83.69	15.42	51.03
Focal	AttUnet		78.31	92.30	73.93	83.10	16.46	51.32
($\gamma = 2$)	UNet++	ResNet101	77.78	92.15	74.21	83.52	16.72	51.41
	DLV3		79.53	93.98	77.91	92.78	22.01	57.64
	DLV3+		79.97	94.67	78.18	93.06	22.35	57.92
	UNet		**78.71**	91.99	73.58	83.19	20.18	54.91
	AttUnet		79.26	92.31	74.47	83.76	19.07	54.66
CB	UNet++	ResNet101	78.51	91.96	74.85	83.66	19.17	53.23
	DLV3		79.20	93.87	77.43	92.52	21.86	57.28
	DLV3+		79.83	94.31	77.77	93.12	22.60	57.96
	UNet		78.13	91.61	73.74	83.01	17.82	54.14
CBFocal	AttUnet		78.82	92.02	73.42	83.18	18.76	54.26
($\gamma = 2$)	UNet++	ResNet101	77.76	91.54	72.58	82.45	17.09	53.02
	DLV3		79.39	94.02	77.78	92.68	22.01	57.71
	DLV3+		80.41	94.65	78.21	93.32	22.93	58.40
	UNet		78.68	**92.44**	73.42	83.81	18.04	54.33
LDAM	AttUnet		78.71	92.51	73.73	83.46	19.81	**55.22**
($\mu = 0.5$,	UNet++	ResNet101	78.06	92.20	73.55	84.64	19.04	52.99
$s = 20$)	DLV3		79.62	94.13	77.87	92.79	22.28	57.85
	DLV3+		80.26	94.77	78.40	93.55	23.04	58.63
	UNet		78.25	92.25	74.72	84.74	18.32	**54.67**
BLV	AttUnet		78.97	92.46	74.45	84.56	19.45	54.72
(Gaussian,	UNet++	ResNet101	78.37	92.23	74.93	84.86	19.81	**54.93**
$\sigma = 0.5$)	DLV3		80.13	94.32	78.04	93.04	22.40	57.99
	DLV3+		80.17	94.52	78.42	93.59	23.16	58.71
	UNet		78.63	92.27	**74.85**	**85.51**	**21.18**	54.22
PAT (Ours)	AttUnet		**79.37**	**92.72**	**74.57**	**85.56**	**21.44**	54.41
($T = 20$)	UNet++	ResNet101	**79.14**	**92.51**	**75.24**	**85.80**	**20.66**	54.86
	DLV3		**80.34**	**94.16**	**78.48**	**93.10**	**22.66**	**58.41**
	DLV3+		**80.42**	**94.82**	**78.63**	**94.01**	**23.72**	**59.09**

74.85%, 74.57%, and 75.24% of mIoU and 21.18%, 21.44%, and 20.66% of mIoU on UNet, AttUNet, and UNet++, respectively. Figure 6.2 illustrates that PAT can segment the object completely, compared to other baselines that tend to misclassify the objects' boundary.

6.2.2 Temperature configurations.

We perform various experiments of PAT with different values of temperature T, which are $\{5, 10, 20, 50\}$ (refers to Table 6.3). This ablation test analyzes how temperature parameter T affects the performance of the segmentation model. To make a fair comparison, we conduct all experiments with three related datasets as mentioned in Section 5.4 with three different types of model architecture including SegNet, UNet, and DLV3+.

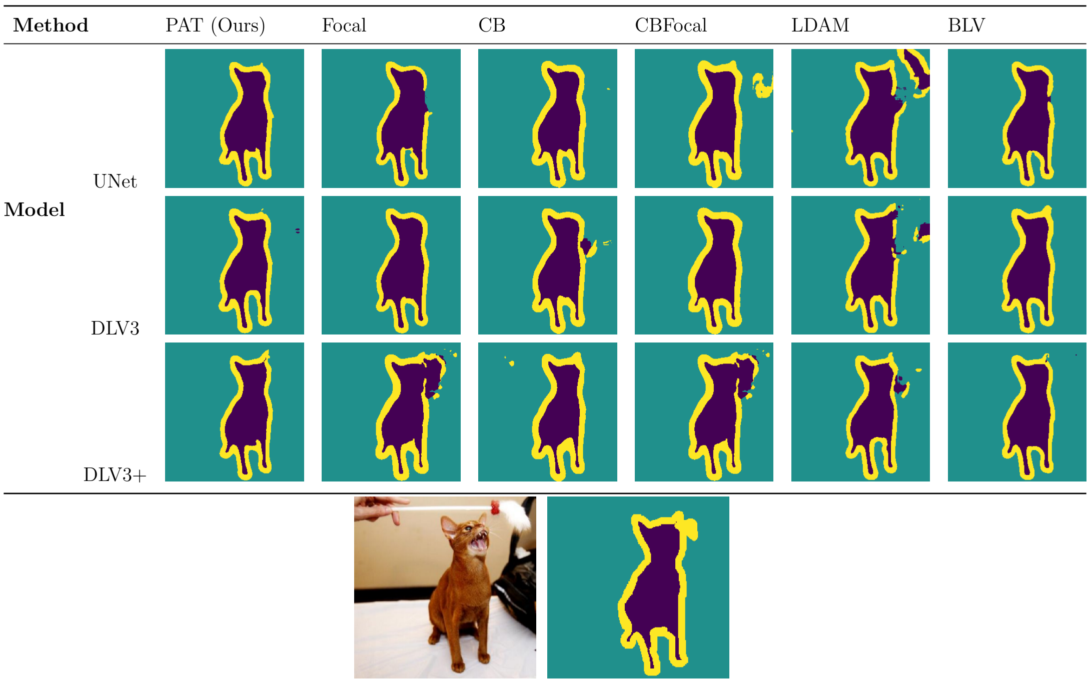
Table 6.3 Quantitative Ablation Results of various values of temperature parameter T. The experiments are conducted using two typical models including UNet[6] and SegNet[5].

Model	T	**OxfordPetIII**		**CityScapes**		**NyU**	
		mIoU↑	Pix Acc↑	mIoU↑	Pix Acc↑	mIoU↑	Pix Acc↑
SegNet	$T = 5$	76.83	91.42	73.86	84.82	19.35	53.39
	$T = 10$	76.69	91.40	74.76	85.14	19.49	54.43
	$T = 20$	76.69	91.28	**76.22**	**85.80**	**21.41**	**55.57**
	$T = 50$	**76.87**	**91.49**	76.21	85.76	**21.41**	54.29
UNet	$T = 5$	78.58	91.42	74.17	84.82	18.23	52.05
	$T = 10$	78.74	92.35	74.57	85.34	19.49	54.13
	$T = 20$	**78.63**	92.27	**74.85**	85.51	21.18	54.22
	$T = 50$	78.54	**92.29**	74.29	**85.56**	**21.32**	**54.26**
DLV3+	$T = 5$	81.97	94.49	78.42	93.66	23.58	58.9
	$T = 10$	82.35	94.81	78.61	93.76	23.65	59.04
	$T = 20$	82.42	94.82	78.63	94.01	**23.72**	59.09
	$T = 50$	**82.58**	**94.97**	**79.02**	**94.45**	23.32	**59.59**

CityScapes dataset whose number of classes is much higher, which is 20, compared to 3 in OxfordPetIII, is indicated to be more sensitive to the temperature parameter T. According to Table 6.3, The mIoU and pix acc of CityScapes increase from 73.86% to 76.22% and from 84.82% to 85.76%, respectively using SegNet model architecture. SegNet model performance in the NyU dataset, on the other hand, achieves from 19.35% to 21.41% in mIoU and from 53.39% to 54.29% in pix acc. The UNet model is not an exception, whose performance on CityScapes and NyU increases 0.7% in mIoU, 1.7% in pix acc, and 3.1% in mIoU, 2.2% in pix acc, respectively. These quantitative results suggest that the temperature parameter T is needed to make the proposed method robust to different datasets whose different numbers of classes.

6.2.3 Class-wise performance evaluation

We investigate the class-wise model performance based mIoU metric on CityScapes Dataset using DeepLabV3+ model architecture (refers to Table 6.4). Table 6.4 suggests that PAT can improve the model performance on both head and tail classes, compared to the current state-of-the-art.



(a) Image (b) Label

Figure 6.2 Visualization of segmentation performance on OxfordPetIII dataset trained by different model architectures including UNet, AttUNet, and NestUNet.

Table 6.4 Class-wise experimental evaluation on CityScapes[2] Dataset using DLV3+[10] based mIoU metric. Note that bold and underlined number indicates the highest and second-highest performance cases.

Method	**Road**	**S.Walk**	**Build.**	**Wall**	**Fence**	**Pole**	**Light**	**Sign**	**Veget.**	**Terrain**	**Sky**	**Person**	**Rider**	**Car**	**Truck**	**Bus**	**Train**	**Motor**	**Bike**	**Void**	**mIoU**
CE	99.11	78.60	92.97	63.32	59.07	61.34	64.07	73.82	94.10	52.56	95.60	78.40	56.13	94.89	84.84	85.69	82.17	70.44	68.16	97.79	77.73
Focal	99.48	78.34	92.89	63.62	58.91	61.09	64.45	73.78	94.44	53.04	96.09	78.51	56.55	94.66	85.32	85.52	81.90	70.21	68.34	97.71	78.18
CB	99.33	78.67	92.95	63.58	59.24	61.27	64.02	73.59	94.05	52.70	95.14	78.45	55.96	94.79	85.03	85.55	82.25	70.71	68.97	97.69	77.77
CBFocal	99.37	78.78	93.22	63.28	59.29	61.27	64.54	74.02	94.47	52.55	96.06	78.80	56.40	95.03	85.3	85.84	82.48	70.62	68.90	98.12	78.21
LDAM	99.47	79.60	93.09	63.54	59.73	61.19	65.42	74.40	94.22	52.28	**96.76**	78.93	56.51	95.17	85.68	85.98	82.54	**71.30**	68.40	97.84	78.40
BLV	99.12	**79.97**	93.44	63.12	59.49	**61.86**	**65.52**	74.69	**95.03**	53.13	96.49	**79.95**	57.36	95.31	86.39	86.35	83.57	71.14	68.83	98.71	78.42
PAT	**99.76**	79.92	**93.89**	**64.23**	**60.11**	61.11	65.28	**75.63**	94.95	**54.38**	95.64	79.19	**57.60**	**95.80**	**86.52**	**86.37**	**84.32**	70.23	**68.69**	**99.15**	78.63

6.2.4 Training utilization.

Owing to the demand of taking full advantage of big data which is not only a large scaled number of samples but also high pixel resolution[131], a method that is low cost in both computation and memory usage is essential. To investigate the performance of different methods, we use three metrics including the average training time (seconds/epoch), the average memory acquisition, and the average GPU utilization. We calculate these metrics in each epoch and then take the average value once the training is done.

Fig. 6.3 suggests that PAT (pink circle), which includes the LA and PAT can adapt to a wide range of hardware specifications. While the proposed method acquires roughly 15GB, recent methods (i.e. BLV, LDAM) acquire nearly 17GB and 20GB in the OxfordPetIII and NyU datasets, respectively. In the CityScapes dataset, the proposed method is one of the three lowest GPU-utilized methods, along with the vanilla cross-entropy loss function, which also refers to the lowest time-consumed method.

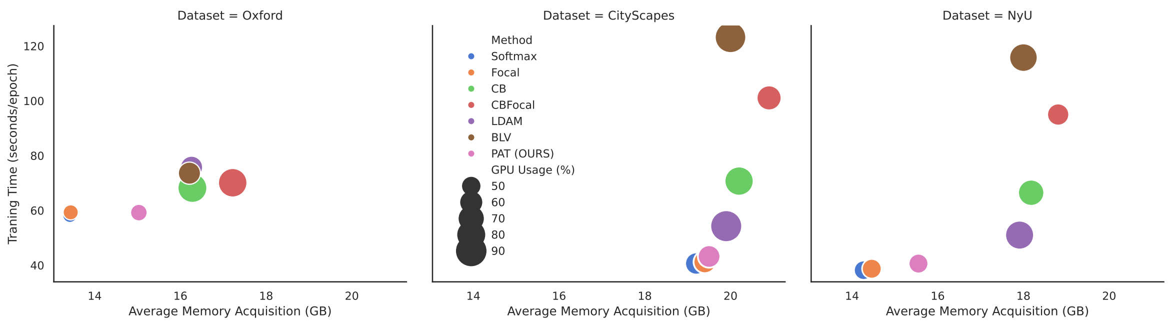


Figure 6.3 Performance comparison between baselines and our proposed method in three different scenarios containing OxfordPetIII, CityScapes, and NyU. Performance metrics include Training Time (seconds/epoch), Average Memory Acquisition shown in Gigabyte (GB) units, and the GPU Utilization proportion (%).

6.3 Limitations

There are two main types of imbalance: class imbalance (some classes appear more common than others) and size imbalance (some objects account for more proportion than others in one mask)[130]. PAT can tackle the issues of detecting long-tailed rare objects as well as keeping good performance on well-classified and large-proportion categories. However, several key areas still require improvement.

First, the amount of memory acquired by PAT is still larger than the Focal and CE (refers to Fig. 6.3). This high computation and memory is owing to the large number of calculations needed for the exponential function, though it is a smooth and differentiated function along with its ability to put a high scalarization on low confidence classes.

Second, sharing a common challenge with other techniques, PAT is notably susceptible to domain shift. This means that logits differ significantly across different data domains, making it difficult to determine the optimal scaling loss coefficient. Applying the model trained on one domain to others can result in performance degradation. Future work could address this issue by incorporating domain generalization techniques[132, 133, 134].

CONCLUSION

General Conclusion

In this paper, we addressed the challenges of long-tailed segmentation learning by introducing a novel Pixel-wise Adaptive Training (PAT) technique. Our approach specifically targets the imbalance in label masks and the detrimental impact of class-wise relationships among various class-specific predictions. The PAT technique comprises two key components: class-wise gradient magnitude homogenization and pixel-wise class-specific loss adaptation (PCLA). Through these components, PAT ensures equal consideration of class-wise impacts on model updates and encourages learning from classes with low prediction confidence while guarding against forgetting classes with high confidence. Our extensive experiments demonstrate that PAT significantly improves performance, surpassing current state-of-the-art methods by 2.2% on the NYU dataset, enhancing overall pixel-wise accuracy by 2.85%, and improving intersection over union by 2.07%, while achieving a notable reduction in rare class detection error by 0.39% on the OxfordPetIII, CityScape, and NYU datasets.

Development Orientation

There are several potential avenues for future work to further advance the capabilities of long-tailed segmentation learning. First, exploring the integration of PAT with other adaptive learning techniques could provide insights into synergistic effects and further performance enhancements. Second, extending the evaluation to a broader range of datasets, including those from different domains and with varying levels of class imbalance, would help in understanding the generalizability and robustness of the PAT approach. Third, investigating the application of PAT in real-time segmentation tasks could be beneficial for scenarios requiring rapid adaptation to new data. Finally, incorporating self-supervised or semi-supervised learning paradigms could reduce the reliance on labeled data, making PAT applicable in more diverse and data-scarce environments.

REFRENCE

[1] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar, "Cats and dogs," in *CVPR*, 2012.

[2] 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," in *CVPR*, 2016.

[3] P. K. Nathan Silberman, Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012.

[4] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning.* Cambridge University Press, 2023.

[5] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE TPAMI*, 2017.

[6] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, 2015.

[7] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *ICCV*, 2017, pp. 2999–3007.

[8] Y. Wang, J. Fei, H. Wang, W. Li, T. Bao, L. Wu, R. Zhao, and Y. Shen, "Balancing logit variation for long-tailed semantic segmentation," in *CVPR*, 2023, pp. 19 561–19 573.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

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

[11] A. Gupta, P. Dollar, and R. Girshick, "Lvis: A dataset for large vocabulary instance segmentation," in *CVPR*, June 2019.

[12] J. Bai, Z. Liu, H. Wang, J. Hao, Y. FENG, H. Chu, and H. Hu, "On the effectiveness of out-of-distribution data in self-supervised long-tail learning." in *The Eleventh International Conference on Learning Representations*, 2023.

[13] B. Dong, P. Zhou, S. Yan, and W. Zuo, “LPT: Long-tailed prompt tuning for image classification,” in *The Eleventh International Conference on Learning Representations*, 2023.

[14] T. Perrett, S. Sinha, T. Burghardt, M. Mirmehdi, and D. Damen, “Use your head: Improving long-tail video recognition,” in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[15] Y.-Y. He, P. Zhang, X.-S. Wei, X. Zhang, and J. Sun, “Relieving long-tailed instance segmentation via pairwise class balance,” in *CVPR*, 2022, pp. 6990–6999.

[16] K. P. Alexandridis, J. Deng, A. Nguyen, and S. Luo, “Long-tailed instance segmentation using gumbel optimized loss,” in *ECCV*, 2022, pp. 353–369.

[17] J. Wang, W. Zhang, Y. Zang, Y. Cao, J. Pang, T. Gong, K. Chen, Z. Liu, C. C. Loy, and D. Lin, “Seesaw loss for long-tailed instance segmentation,” in *CVPR*, 2021.

[18] Y. Li, T. Wang, B. Kang, S. Tang, C. Wang, J. Li, and J. Feng, “Overcoming classifier imbalance for long-tail object detection with balanced group softmax,” in *CVPR*, 2020, pp. 10 991–11 000.

[19] Y. Wang, J. Fei, H. Wang, W. Li, T. Bao, L. Wu, R. Zhao, and Y. Shen, “Balancing logit variation for long-tailed semantic segmentation,” in *CVPR*, 2023.

[20] C. Zhang, T.-Y. Pan, T. Chen, J. Zhong, W. Fu, and W.-L. Chao, “Learning with free object segments for long-tailed instance segmentation,” in *ECCV*, 2022, pp. 655–672.

[21] R. Li, S. Li, C. He, Y. Zhang, X. Jia, and L. Zhang, “Class-balanced pixel-level self-labeling for domain adaptive semantic segmentation,” in *CVPR*, 2022, pp. 11 593–11 603.

[22] Y. Zang, C. Huang, and C. Change Loy, “Fasa: Feature augmentation and sampling adaptation for long-tailed instance segmentation,” in *ICCV*, 2021.

[23] Y. Wang, W. Gan, J. Yang, W. Wu, and J. Yan, “Dynamic curriculum learning for imbalanced data classification,” in *International Conference on Computer Vision*, 2019, pp. 5017–5026.

[24] C. Feng, Y. Zhong, and W. Huang, "Exploring classification equilibrium in long-tailed object detection," in *International Conference on Computer Vision*, 2021.

[25] X. Zhang, Z. Wu, Z. Weng, H. Fu, J. Chen, Y.-G. Jiang, and L. Davis, "Videolt: Large-scale long-tailed video recognition," in *International Conference on Computer Vision*, 2021.

[26] R. Jiawei, C. Yu, X. Ma, H. Zhao, S. Yi *et al.*, "Balanced meta-softmax for long-tailed visual recognition," in *Advances in Neural Information Processing Systems*, 2020.

[27] Y. Zang, C. Huang, and C. C. Loy, "Fasa: Feature augmentation and sampling adaptation for long-tailed instance segmentation," in *International Conference on Computer Vision*, 2021.

[28] T. Wang, Y. Li, B. Kang, J. Li, J. Liew, S. Tang, S. Hoi, and J. Feng, "The devil is in classification: A simple framework for long-tail instance segmentation," in *European Conference on Computer Vision*, 2020.

[29] J. Tan, C. Wang, B. Li, Q. Li, W. Ouyang, C. Yin, and J. Yan, "Equalization loss for long-tailed object recognition," in *Computer Vision and Pattern Recognition*, 2020, pp. 11 662–11 671.

[30] T.-I. Hsieh, E. Robb, H.-T. Chen, and J.-B. Huang, "Droploss for long-tail instance segmentation," in *AAAI Conference on Artificial Intelligence*, vol. 35, no. 2, 2021, pp. 1549–1557.

[31] C. Elkan, "The foundations of cost-sensitive learning," in *International Joint Conference on Artificial Intelligence*, 2001.

[32] Z.-H. Zhou and X.-Y. Liu, "Training cost-sensitive neural networks with methods addressing the class imbalance problem," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 1, pp. 63–77, 2005.

[33] P. Zhao, Y. Zhang, M. Wu, S. C. Hoi, M. Tan, and J. Huang, "Adaptive cost-sensitive online classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 2, pp. 214–228, 2018.

[34] Y. Zhang, P. Zhao, J. Cao, W. Ma, J. Huang, Q. Wu, and M. Tan, "Online adaptive asymmetric active learning for budgeted imbalanced data," in *SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2768–2777.

[35] Y. Zhang, P. Zhao, S. Niu, Q. Wu, J. Cao, J. Huang, and M. Tan, “Online adaptive asymmetric active learning with limited budgets,” *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[36] Y. Sun, M. S. Kamel, A. K. Wong, and Y. Wang, “Cost-sensitive boosting for classification of imbalanced data,” *Pattern Recognition*, vol. 40, no. 12, pp. 3358–3378, 2007.

[37] Y. Cui, M. Jia, T.-Y. Lin, Y. Song, and S. Belongie, “Class-balanced loss based on effective number of samples,” in *Computer Vision and Pattern Recognition*, 2019, pp. 9268–9277.

[38] G. R. Kini, O. Paraskevas, S. Oymak, and C. Thrampoulidis, “Label-imbalanced and group-sensitive classification under overparameterization,” in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 18 970–18 983.

[39] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection,” in *International Conference on Computer Vision*, 2017, pp. 2980–2988.

[40] J. Shu, Q. Xie, L. Yi, Q. Zhao, S. Zhou, Z. Xu, and D. Meng, “Meta-weight-net: Learning an explicit mapping for sample weighting,” *Advances in Neural Information Processing Systems*, 2019.

[41] F. Provost, “Machine learning from imbalanced data sets 101,” in *AAAI Workshop on Imbalanced Data Sets*, vol. 68, no. 2000, 2000, pp. 1–3.

[42] A. K. Menon, S. Jayasumana, A. S. Rawat, H. Jain, A. Veit, and S. Kumar, “Long-tail learning via logit adjustment,” in *International Conference on Learning Representations*, 2021.

[43] T. Wu, Z. Liu, Q. Huang, Y. Wang, and D. Lin, “Adversarial robustness under long-tailed distribution,” in *Computer Vision and Pattern Recognition*, 2021, pp. 8659–8668.

[44] J. Tian, Y.-C. Liu, N. Glaser, Y.-C. Hsu, and Z. Kira, “Posterior re-calibration for imbalanced datasets,” in *Advances in Neural Information Processing Systems*, 2020.

[45] S. Zhang, Z. Li, S. Yan, X. He, and J. Sun, “Distribution alignment: A unified framework for long-tail visual recognition,” in *Computer Vision and Pattern Recognition*, 2021, pp. 2361–2370.

[46] Y. Hong, S. Han, K. Choi, S. Seo, B. Kim, and B. Chang, "Disentangling label distribution for long-tailed visual recognition," in *Computer Vision and Pattern Recognition*, 2021.

[47] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *arXiv:1712.04621*, 2017.

[48] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019.

[49] J. Kim, J. Jeong, and J. Shin, "M2m: Imbalanced classification via major-to-minor translation," in *Computer Vision and Pattern Recognition*, 2020.

[50] X. Yin, X. Yu, K. Sohn, X. Liu, and M. Chandraker, "Feature transfer learning for face recognition with under-represented data," in *Computer Vision and Pattern Recognition*, 2019, pp. 5704–5713.

[51] J. Liu, Y. Sun, C. Han, Z. Dou, and W. Li, "Deep representation learning on long-tailed data: A learnable embedding augmentation perspective," in *Computer Vision and Pattern Recognition*, 2020.

[52] J. Wang, T. Lukasiewicz, X. Hu, J. Cai, and Z. Xu, "Rsg: A simple but effective module for learning imbalanced datasets," in *Computer Vision and Pattern Recognition*, 2021, pp. 3784–3793.

[53] P. Chu, X. Bian, S. Liu, and H. Ling, "Feature space augmentation for long-tailed data," in *European Conference on Computer Vision*, 2020.

[54] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-smote: a new over-sampling method in imbalanced data sets learning," in *International Conference on Intelligent Computing*, 2005, pp. 878–887.

[55] Z. Zhong, J. Cui, S. Liu, and J. Jia, "Improving calibration for long-tailed recognition," in *Computer Vision and Pattern Recognition*, 2021.

[56] H.-P. Chou, S.-C. Chang, J.-Y. Pan, W. Wei, and D.-C. Juan, "Remix: Rebalanced mixup," in *European Conference on Computer Vision Workshop*, 2020, pp. 95–110.

[57] S. Li, K. Gong, C. H. Liu, Y. Wang, F. Qiao, and X. Cheng, "Metasaug: Meta semantic augmentation for long-tailed visual recognition," in *Computer Vision and Pattern Recognition*, 2021, pp. 5212–5221.

[58] C. Huang, Y. Li, C. C. Loy, and X. Tang, "Learning deep representation for imbalanced classification," in *Computer Vision and Pattern Recognition*, 2016.

[59] Q. Dong, S. Gong, and X. Zhu, "Class rectification hard mining for imbalanced deep learning," in *International Conference on Computer Vision*, 2017, pp. 1851–1860.

[60] X. Zhang, Z. Fang, Y. Wen, Z. Li, and Y. Qiao, "Range loss for deep face recognition with long-tailed training data," in *International Conference on Computer Vision*, 2017, pp. 5409–5418.

[61] B. Kang, Y. Li, S. Xie, Z. Yuan, and J. Feng, "Exploring balanced feature spaces for representation learning," in *International Conference on Learning Representations*, 2021.

[62] J. Cui, Z. Zhong, S. Liu, B. Yu, and J. Jia, "Parametric contrastive learning," in *International Conference on Computer Vision*, 2021.

[63] P. Wang, K. Han, X.-S. Wei, L. Zhang, and L. Wang, "Contrastive learning based hybrid networks for long-tailed image classification," in *Computer Vision and Pattern Recognition*, 2021, pp. 943–952.

[64] D. Samuel and G. Chechik, "Distributional robustness loss for long-tail learning," in *International Conference on Computer Vision*, 2021.

[65] Z. Liu, Z. Miao, X. Zhan, J. Wang, B. Gong, and S. X. Yu, "Large-scale long-tailed recognition in an open world," in *Computer Vision and Pattern Recognition*, 2019, pp. 2537–2546.

[66] L. Zhu and Y. Yang, "Inflated episodic memory with region self-attention for long-tailed visual recognition," in *Computer Vision and Pattern Recognition*, 2020, pp. 4344–4353.

[67] W. Ouyang, X. Wang, C. Zhang, and X. Yang, "Factors in finetuning deep model for object detection with long-tail distribution," in *Computer Vision and Pattern Recognition*, 2016, pp. 864–873.

[68] Y. Zhong, W. Deng, M. Wang, J. Hu, J. Peng, X. Tao, and Y. Huang, "Unequal-training for deep face recognition with long-tailed noisy data," in *Computer Vision and Pattern Recognition*, 2019, pp. 7812–7821.

[69] Y.-X. Wang, D. Ramanan, and M. Hebert, "Learning to model the tail," in *Advances in Neural Information Processing Systems*, 2017.

[70] Y. Cui, Y. Song, C. Sun, A. Howard, and S. Belongie, “Large scale fine-grained categorization and domain-specific transfer learning,” in *Computer Vision and Pattern Recognition*, 2018, pp. 4109–4118.

[71] K. Cao, C. Wei, A. Gaidon, N. Arechiga, and T. Ma, “Learning imbalanced datasets with label-distribution-aware margin loss,” in *Advances in Neural Information Processing Systems*, 2019.

[72] S. Khan, M. Hayat, S. W. Zamir, J. Shen, and L. Shao, “Striking the right balance with uncertainty,” in *Computer Vision and Pattern Recognition*, 2019, pp. 103–112.

[73] B. Kang, S. Xie, M. Rohrbach, Z. Yan, A. Gordo, J. Feng, and Y. Kalantidis, “Decoupling representation and classifier for long-tailed recognition,” in *International Conference on Learning Representations*, 2020.

[74] X. Hu, Y. Jiang, K. Tang, J. Chen, C. Miao, and H. Zhang, “Learning to segment the tail,” in *Computer Vision and Pattern Recognition*, 2020.

[75] M. A. Jamal, M. Brown, M.-H. Yang, L. Wang, and B. Gong, “Rethinking class-balanced methods for long-tailed visual recognition from a domain adaptation perspective,” in *Computer Vision and Pattern Recognition*, 2020, pp. 7610–7619.

[76] D. Cao, X. Zhu, X. Huang, J. Guo, and Z. Lei, “Domain balancing: Face recognition on long-tailed domains,” in *Computer Vision and Pattern Recognition*, 2020, pp. 5671–5679.

[77] T. Wu, Q. Huang, Z. Liu, Y. Wang, and D. Lin, “Distribution-balanced loss for multi-label classification in long-tailed datasets,” in *European Conference on Computer Vision*, 2020, pp. 162–178.

[78] K. Tang, J. Huang, and H. Zhang, “Long-tailed classification by keeping the good and removing the bad momentum causal effect,” in *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[79] Y. Yang and Z. Xu, “Rethinking the value of labels for improving class-imbalanced learning,” in *Advances in Neural Information Processing Systems*, 2020.

[80] L. Xiang, G. Ding, and J. Han, “Learning from multiple experts: Self-paced knowledge distillation for long-tailed classification,” in *European Conference on Computer Vision*, 2020, pp. 247–263.

[81] T.-Y. Wu, P. Morgado, P. Wang, C.-H. Ho, and N. Vasconcelos, "Solving long-tailed recognition with deep realistic taxonomic classifier," in *European Conference on Computer Vision*, 2020, pp. 171–189.

[82] B. Zhou, Q. Cui, X.-S. Wei, and Z.-M. Chen, "Bbn: Bilateral-branch network with cumulative learning for long-tailed visual recognition," in *Computer Vision and Pattern Recognition*, 2020, pp. 9719–9728.

[83] Y. Li, T. Wang, B. Kang, S. Tang, C. Wang, J. Li, and J. Feng, "Overcoming classifier imbalance for long-tail object detection with balanced group softmax," in *Computer Vision and Pattern Recognition*, 2020, pp. 10 991–11 000.

[84] R. He, J. Yang, and X. Qi, "Re-distributing biased pseudo labels for semi-supervised semantic segmentation: A baseline investigation," in *International Conference on Computer Vision*, 2021.

[85] C. Wei, K. Sohn, C. Mellina, A. Yuille, and F. Yang, "Crest: A class-rebalancing self-training framework for imbalanced semi-supervised learning," in *Computer Vision and Pattern Recognition*, 2021.

[86] B. Liu, H. Li, H. Kang, G. Hua, and N. Vasconcelos, "Gistnet: a geometric structure transfer network for long-tailed recognition," in *International Conference on Computer Vision*, 2021.

[87] J. Tan, X. Lu, G. Zhang, C. Yin, and Q. Li, "Equalization loss v2: A new gradient balance approach for long-tailed object detection," in *Computer Vision and Pattern Recognition*, 2021, pp. 1685–1694.

[88] J. Wang, W. Zhang, Y. Zang, Y. Cao, J. Pang, T. Gong, K. Chen, Z. Liu, C. C. Loy, and D. Lin, "Seesaw loss for long-tailed instance segmentation," in *Computer Vision and Pattern Recognition*, 2021.

[89] T. Wang, Y. Zhu, C. Zhao, W. Zeng, J. Wang, and M. Tang, "Adaptive class suppression loss for long-tail object detection," in *Computer Vision and Pattern Recognition*, 2021, pp. 3103–3112.

[90] S. Park, J. Lim, Y. Jeon, and J. Y. Choi, "Influence-balanced loss for imbalanced visual classification," in *International Conference on Computer Vision*, 2021.

[91] Z. Deng, H. Liu, Y. Wang, C. Wang, Z. Yu, and X. Sun, "Pml: Progressive margin loss for long-tailed age classification," in *Computer Vision and Pattern Recognition*, 2021, pp. 10 503–10 512.

[92] S. Changpinyo, P. Sharma, N. Ding, and R. Soricut, “Conceptual 12m: Pushing web-scale image-text pre-training to recognize long-tail visual concepts,” in *Computer Vision and Pattern Recognition*, 2021.

[93] Y.-Y. He, J. Wu, and X.-S. Wei, “Distilling virtual examples for long-tailed recognition,” in *International Conference on Computer Vision*, 2021.

[94] C. Zhang, T.-Y. Pan, Y. Li, H. Hu, D. Xuan, S. Changpinyo, B. Gong, and W.-L. Chao, “Mosaicos: A simple and effective use of object-centric images for long-tailed object detection,” in *International Conference on Computer Vision*, 2021.

[95] T. Li, L. Wang, and G. Wu, “Self supervision to distillation for long-tailed visual recognition,” in *International Conference on Computer Vision*, 2021.

[96] X. Wang, L. Lian, Z. Miao, Z. Liu, and S. X. Yu, “Long-tailed recognition by routing diverse distribution-aware experts,” in *International Conference on Learning Representations*, 2021.

[97] Z. Weng, M. G. Ogut, S. Limonchik, and S. Yeung, “Unsupervised discovery of the long-tail in instance segmentation using hierarchical self-supervision,” in *Computer Vision and Pattern Recognition*, 2021.

[98] A. Desai, T.-Y. Wu, S. Tripathi, and N. Vasconcelos, “Learning of visual relations: The devil is in the tails,” in *International Conference on Computer Vision*, 2021.

[99] H. Guo and S. Wang, “Long-tailed multi-label visual recognition by collaborative training on uniform and re-balanced samplings,” in *Computer Vision and Pattern Recognition*, 2021, pp. 15 089–15 098.

[100] J. Cai, Y. Wang, and J.-N. Hwang, “Ace: Ally complementary experts for solving long-tailed recognition in one-shot,” in *International Conference on Computer Vision*, 2021.

[101] J. Cui, S. Liu, Z. Tian, Z. Zhong, and J. Jia, “Reslt: Residual learning for long-tailed recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[102] Y. Zhang, B. Hooi, L. Hong, and J. Feng, “Self-supervised aggregation of diverse experts for test-agnostic long-tailed recognition,” in *Advances in Neural Information Processing Systems*, 2022.

[103] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2016.

[104] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2014.

[105] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017.

[106] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, E. Brynjolfsson, S. Buch, D. Card, R. Castellon, N. S. Chatterji, A. S. Chen, K. A. Creel, J. Davis, D. Demszky, C. Donahue, M. Doumbouya, E. Durmus, S. Ermon, J. Etchemendy, K. Ethayarajh, L. Fei-Fei, C. Finn, T. Gale, L. E. Gillespie, K. Goel, N. D. Goodman, S. Grossman, N. Guha, T. Hashimoto, P. Henderson, J. Hewitt, D. E. Ho, J. Hong, K. Hsu, J. Huang, T. F. Icard, S. Jain, D. Jurafsky, P. Kalluri, S. Karamcheti, G. Keeling, F. Khani, O. Khattab, P. W. Koh, M. S. Krass, R. Krishna, R. Kuditipudi, A. Kumar, F. Ladhak, M. Lee, T. Lee, J. Leskovec, I. Levent, X. L. Li, X. Li, T. Ma, A. Malik, C. D. Manning, S. P. Mirchandani, E. Mitchell, Z. Munyikwa, S. Nair, A. Narayan, D. Narayanan, B. Newman, A. Nie, J. C. Niebles, H. Nilforoshan, J. F. Nyarko, G. Ogut, L. Orr, I. Papadimitriou, J. S. Park, C. Piech, E. Portelance, C. Potts, A. Raghunathan, R. Reich, H. Ren, F. Rong, Y. H. Roohani, C. Ruiz, J. Ryan, C. R'e, D. Sadigh, S. Sagawa, K. Santhanam, A. Shih, K. P. Srinivasan, A. Tamkin, R. Taori, A. W. Thomas, F. Tramèr, R. E. Wang, W. Wang, B. Wu, J. Wu, Y. Wu, S. M. Xie, M. Yasunaga, J. You, M. A. Zaharia, M. Zhang, T. Zhang, X. Zhang, Y. Zhang, L. Zheng, K. Zhou, and P. Liang, "On the opportunities and risks of foundation models," *ArXiv*, 2021. [Online]. Available: https://crfm.stanford.edu/assets/report.pdf

[107] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2014.

[108] Z. Yang, Z. Hu, Y. Deng, C. Dyer, and A. Smola, "Neural machine translation with recurrent attention modeling," in *Proceedings of the 15th Conference*

of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers, 2017.

[109] E. A. Nadaraya, "On estimating regression," *Theory of Probability & Its Applications*, 1964.

[110] G. S. Watson, "Smooth regression analysis," *Sankhyā: The Indian Journal of Statistics, Series A (1961-2002)*, 1964.

[111] E. Parzen, "On Consistent Estimates of the Spectrum of a Stationary Time Series," *The Annals of Mathematical Statistics*, 1957.

[112] A. Graves, "Generating sequences with recurrent neural networks," 2014.

[113] L. Rabiner and B. Juang, *Fundamentals of Speech Recognition*, ser. Prentice-Hall Signal Processing Series: Advanced monographs. PTR Prentice Hall, 1993. [Online]. Available: https://books.google.com.vn/books?id=XEVqQgAACAAJ

[114] Z. Lin, M. Feng, C. N. dos Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, "A STRUCTURED SELF-ATTENTIVE SENTENCE EMBEDDING," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=BJC_jUqxe

[115] J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine reading," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, J. Su, K. Duh, and X. Carreras, Eds. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 551–561. [Online]. Available: https://aclanthology.org/D16-1053

[116] A. Parikh, O. Täckström, D. Das, and J. Uszkoreit, "A decomposable attention model for natural language inference," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, J. Su, K. Duh, and X. Carreras, Eds. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 2249–2255. [Online]. Available: https://aclanthology.org/D16-1244

[117] R. Paulus, C. Xiong, and R. Socher, "A deep reinforced model for abstractive summarization," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=HkAClQgA-

[118] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2009, pp. 248–255.

[119] Y. Li, H. Zhao, X. Qi, L. Wang, Z. Li, J. Sun, and J. Jia, “Fully convolutional networks for panoptic segmentation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 214–223.

[120] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” 2015.

[121] A. Javaloy and I. Valera, “Rotograd: Gradient homogenization in multitask learning,” in *ICLR*, 2022.

[122] R. Murray, B. Swenson, and S. Kar, “Revisiting normalized gradient descent: Fast evasion of saddle points,” *IEEE Transactions on Automatic Control*, 2019.

[123] Y. Cui, M. Jia, T.-Y. Lin, Y. Song, and S. Belongie, “Class-balanced loss based on effective number of samples,” in *CVPR*, 2019, pp. 9260–9269.

[124] J. Ren, C. Yu, s. sheng, X. Ma, H. Zhao, S. Yi, and h. Li, “Balanced meta-softmax for long-tailed visual recognition,” in *NeurIPS*, 2020, pp. 4175–4186.

[125] K. Cao, C. Wei, A. Gaidon, N. Arechiga, and T. Ma, “Learning imbalanced datasets with label-distribution-aware margin loss,” in *NeurIPS*, 2019.

[126] T.-Y. Pan, C. Zhang, Y. Li, H. Hu, D. Xuan, S. Changpinyo, B. Gong, and W.-L. Chao, “On model calibration for long-tailed object detection and instance segmentation,” in *NeurIPS*, 2021.

[127] O. Oktay, J. Schlemper, L. L. Folgoc, M. Lee, M. Heinrich, K. Misawa, K. Mori, S. McDonagh, N. Y. Hammerla, B. Kainz, B. Glocker, and D. Rueckert, “Attention u-net: Learning where to look for the pancreas,” *arXiv*, 2018.

[128] Z. Zhou, M. M. Rahman Siddiquee, N. Tajbakhsh, and J. Liang, “Unet++: A nested u-net architecture for medical image segmentation,” in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2018*, 2018.

[129] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, “Rethinking atrous convolution for semantic image segmentation,” 2017.

[130] Z. Wang, M. Berman, A. Rannen-Triki, P. Torr, D. Tuia, T. Tuytelaars, L. V. Gool, J. Yu, and M. B. Blaschko, “Revisiting evaluation metrics for semantic segmentation: Optimization and evaluation of fine-grained intersection over union,” in *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

[131] Y. Zhang, B. Kang, B. Hooi, S. Yan, and J. Feng, “Deep long-tailed learning: A survey,” *IEEE TPAMI*, 2023.

[132] Y. Shi, J. Seely, P. Torr, S. N, A. Hannun, N. Usunier, and G. Synnaeve, “Gradient matching for domain generalization,” in *ICLR*, 2022.

[133] K. Zhou, Y. Yang, T. Hospedales, and T. Xiang, “Learning to generate novel domains for domain generalization,” in *Computer Vision – ECCV 2020*, 2020.

[134] R. Meng, X. Li, W. Chen, S. Yang, J. Song, X. Wang, L. Zhang, M. Song, D. Xie, and S. Pu, “Attention diversification for domain generalization,” in *Computer Vision – ECCV 2022*, 2022.