# EFFICIENT AND ROBUST DEEP LEARNING

**by**

**ZHIQIANG TANG**

**A dissertation submitted to the**

**School of Graduate Studies**

**Rutgers, The State University of New Jersey**

**In partial fulfillment of the requirements**

**For the degree of**

**Doctor of Philosophy**

**Graduate Program in Computer Science**

**Written under the direction of**

**Dimitris N. Metaxas**

**And approved by**

_____

_____

_____

_____

**New Brunswick, New Jersey**

**May, 2021**

## ABSTRACT OF THE DISSERTATION

# Efficient and Robust Deep Learning

**By ZHIQIANG TANG**

**Dissertation Director:**
**Dimitris N. Metaxas**

Deep learning enables automatically discovering useful, multistage, task-specific features from high-dimensional raw data. Instead of relying on domain expertise to hand-engineer features, it uses a general-learning procedure that is readily applicable to many domains such as image analysis and natural language processing. Deep learning has made significant advances after decades of development, in which the dataset size, model size, and benchmark accuracy have dramatically increased. However, these three increasing trends pose corresponding challenges regarding data efficiency, model efficiency, and generalization robustness. To address these challenges, we research solutions from three perspectives: automatic data augmentation, efficient architecture design, and robust feature normalization. (i) Chapter 2 to Chapter 4 propose a series of automatic data augmentation methods to replace the hand-crafted rules that define the augmentation sampling distributions, magnitude ranges, and functions. Experiments show the automatic augmentation methods can apply to diverse tasks and effectively improve their performance without using extra training data. (ii) Chapter 5 introduces the quantized coupled U-Nets architecture to boost the efficiency of stacked U-Nets with broad applications to location-sensitive tasks. U-Net pairs are coupled

together through shortcut connections that can facilitate feature reuse across U-Nets and reduce redundant network weights. Quantizing weights, features, and gradients to low-bit representations can further make coupled U-Nets more lightweight, accelerating both training and testing. (iii) Chapter 6 presents two feature normalization techniques, SelfNorm and CrossNorm, to promote deep networks' robustness. SelfNorm utilizes attention to highlight vital feature statistics and suppress trivial ones, whereas CrossNorm augments feature statistics by randomly exchanging statistics between feature maps in training. SelfNorm and CrossNorm can reduce deep networks' sensitivity and bias to feature statistics and improve the robustness to out-of-distribution data, which usually results in unforeseen feature statistics. Overall, the proposed automatic data augmentation, efficient U-Net design, and robust feature normalization shed light on new perspectives for efficient and robust deep learning.

# Acknowledgements

Foremost, I wish to express my sincere appreciation to my advisor, Prof. Dimitris N. Metaxas, for his excellent support and guidance during my Ph.D. study. Prof. Dimitris N. Metaxas has been convincingly motivating and encouraging me to work on crucial problems in deep learning and computer vision. Nothing is more important than the considerable freedoms he gives to me in choosing research topics and building collaborations. He is always patient and supportive, even when the road got tough. I could not have imagined having a better advisor for my Ph.D. study.

I also want to thank other committee members: Prof. Sunjin Ahn, Prof. Hao Wang, and Prof. Xiaolei Huang, for their time and insightful comments regarding this dissertation. It is my great honor to have each of them serving on my committee.

My sincere thanks also go to Dr. Mu Li (Amazon AI) and Dr. Rogerio Feris (IBM Research AI), who offered me fantastic internship opportunities. During my internships, I felt much fortunate to work with Dr. Yi Zhu and Dr. Zhi Zhang from Amazon AI, and Dr. Leonid Karlinsky and Dr. Prasanna Sattigeri from IBM research AI. They provided me adequate computational resources and useful advice for my research work, which has become a part of my dissertation.

I am also genuinely grateful to my co-authors: Yunhe Guo, Shijie Geng, and Tingfeng Li, for their generous help and in-depth discussions on my research work. I wish them the best of luck in pursuing their Ph.D. degrees.

Last but not least, I would like to thank many current and former colleagues in the Computational Biomedicine Imaging and Modeling Center (CBIM): Prof. Shaoting Zhang, Dr. Jingjing Liu, Dr. Bo Liu, Dr. Han Zhang, Dr. Yan Zhu, Dr. Chaowei Tan, Dr. Ji Zhang, Yu Tian, Lezi Wang, Yizhe Zhu, Pengxiang Wu, Jingru Yi, Qi Chang, Hui Qu, Gang Qiao, Long Zhao, Qiaoying Huang, Fangda Han, Yuting Wang, Jiatong

Li. I have benefited tremendously from their kind help and valuable suggestions for my career, research, and life.

# Dedication

*To my family and friends*

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation

Deep learning [12] is a subset of machine learning algorithms that uses multiple learnable layers to progressively extract high-level features from raw data. The main advantage of deep learning is that it can learn feature representation hierarchically by building complex concepts on top of simpler concepts. Deep learning has a long and rich history dating back to the 1940s. Its development has gone through three phases: cybernetics in the 1940s–1960s [13, 14, 15], connectionism in the 1980s–1990s [16, 17, 18, 19], and the new name, deep learning, since 2006 [20, 21, 22]. The recent Deep Learning textbook [23] has identified three key increasing trends in dataset sizes, model sizes, and benchmark accuracy, which, we argue, also face three corresponding challenges.

**Trend 1: Increasing dataset sizes**. The availability of abundant data is an essential factor in modern deep learning's success. More training data generally can reduce skills required to train deep neural networks, better exploit deep neural networks' capacities, and alleviate the burden of generalization to new data after observing small data [23]. Indeed, deep learning progresses rapidly, along with increasing dataset sizes. Take object recognition for example, CIFAR [24] and Caltech [25] datasets, with tens of thousands of samples, appeared in the first decade of the 2000s. In the early 2010s, the ImageNet datasets [26], including millions of images, were considered large-scale in visual recognition. Recently in later 2010s, the standard "large" has further increased as technology companies have collected substantially larger datasets such as Google's JFT-300M [27] with hundreds of millions of images. Figure 1.1 shows how the size of datasets has increased over time.

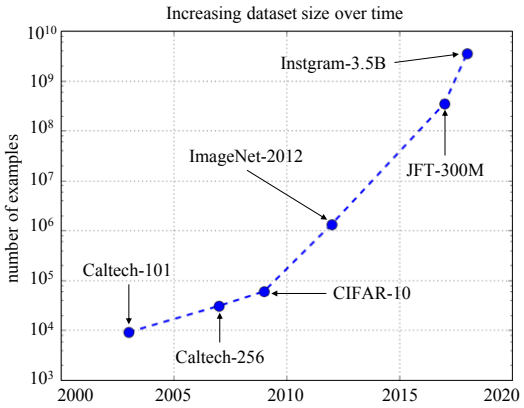



Figure 1.1: The size of object recognition datasets has increased substantially, especially in the 2010s.

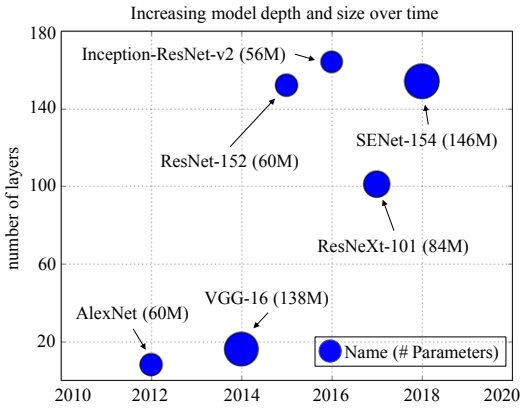



Figure 1.2: Since 2012, the network depth has increased more than 20x and the model parameters have doubled.

**Challenge 1: Data efficiency**. The reliance on large amounts of data, while bringing gains, also makes deep learning suffer from apparent drawbacks. First, obtaining a large number of annotations is expensive and time-consuming. For example, image segmentation [28] requires pixel-wise labeling, which can be even more difficult for medical images [29]. Moreover, data collection is also nontrivial in many fields, such as medical image analysis, due to the cost and privacy concerns [30]. Therefore, *how to reduce the reliance on large-scale data, i.e., improving data efficiency for deep learning, is an important research topic.*

**Trend 2: Growing model sizes**. The ability to train large models is another crucial reason to make deep learning successful. Although LeNet [18], the prototype of convolutional neural network (CNN), was successfully applied to handwritten digit recognition in 1989, CNN had not become popular until the emergence of AlexNet [31] in 2012. Compared to LeNet of 60K parameters, AlexNet of 60M parameters demands much more computational resources for training. Fortunately, the general-purpose GPUs made training AlexNet possible as a result of hardware advancement. Since then, GPUs' computational power has kept increasing, and CNN models have become significantly deeper and larger, as shown in Figure 1.2. Recently, giant networks like EfficientNet-L2 [32] and BiT-L [33] have had hundreds of millions of parameters, whose training requires more powerful TPU clouds.

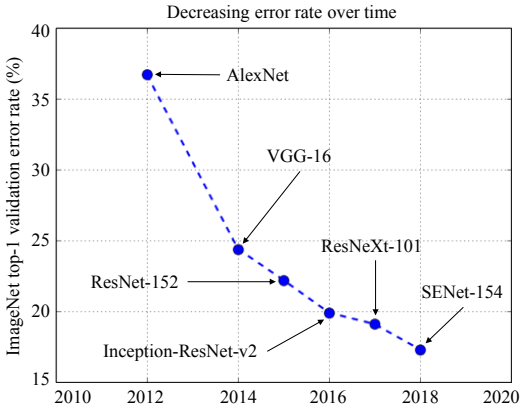



Figure 1.3: The ImageNet classification error has kept decreasing with more advanced (usually deeper and larger) networks, best viewed with Figure 1.2.

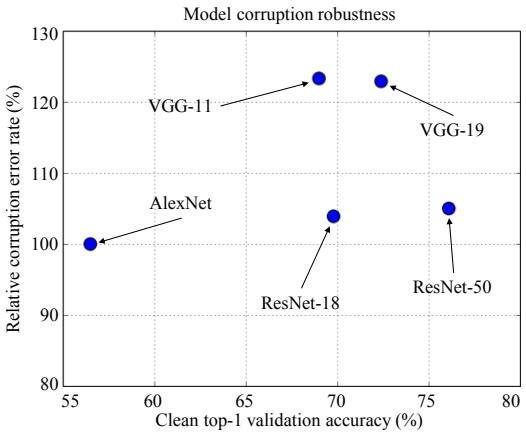



Figure 1.4: Although VGG and ResNet have much higher clean accuracy than AlexNet, their relative corruption errors [11] are also higher than AlexNet's.

**Challenge 2: Model efficiency**. Although high-end GPU and TPU clusters enable the training of large models, deploying the models to real-world applications is still challenging. Today, more and more applications run on mobile devices such as smartphones, robots, and drones. These devices usually have limited computational budgets (tens or hundreds of MFLOPs) [34] and therefore have difficulty in driving large models requiring billions of FLOPs [10]. Also, applications may require over-the-air updates that frequently deliver updated models from severs to clients. Large models can increase the communication burden and slow down the updates. Therefore, *developing compact and efficient models is imperative for mobile applications.*

**Trend 3: Rising benchmark accuracy**. Since the 1980s, researchers have continually broadened deep learning's application scope and improved deep models' accuracy. The earliest deep models [16] were designed to recognize 2-class objects in tightly cropped small images. After decades of development, modern deep neural networks can recognize objects from 1000 classes, and the images are high-resolution without severe cropping. In 2012, AlexNet won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) by a large margin, lowering previous state-of-the-art top-5 error from 26.2% to 15.3% [31]. This historic achievement has motivated intensive research on deep learning. Since then, more advanced deep models have continued to refresh the

ImageNet record, as shown in Figure 1.3. Simultaneously, deep learning has revolutionized other computer vision tasks, e.g., object detection [35, 36], and many other areas such as natural language processing [37, 38, 39] and medical image analysis [40, 41, 42].

**Challenge 3: Generalization robustness**. The impressive successes mainly build on the independent and identically distributed (IID) assumption, under which training and test data come from the same distribution. However, deep models still struggle to generalize to out-of-distribution (OOD) scenarios such as adversarial attacks [43, 5], common corruptions [11], and distribution gaps caused by data collection differences [44]. For instance, using AlexNet as the baseline, VGG [45] and ResNet [10] have higher relative corruption errors [11] meaning larger gaps between clean and corruption errors, albeit their higher accuracy on clean data, as shown in Figure 1.4. In contrast, humans are robust to the small changes in the OOD query data. As OOD data are inevitable in practice, *achieving OOD robustness is an indispensable goal for deep learning systems, especially for safety-critical applications such as self-driving cars.*

## 1.2 Contributions

To address the above three challenges regarding data efficiency, model efficiency, and generalization robustness, we propose automatic data augmentation methods, design efficient network architectures, and develop robust feature normalization techniques. Our research aims to improve the efficiency and robustness of deep learning, and we summarize more detailed contributions as follows.

**Contribution 1: Automatic data augmentation**. Data augmentation by adding modified copies of existing data is effective in improving data efficiency. However, traditional data augmentation requires expert knowledge to specify transformation operations, boundaries, and sampling distributions. We overcome the three limitations progressively through three coherent studies. First, we propose adversarial data augmentation to learn dynamic sampling distributions. The goal is to adaptively generate hard augmented examples and thus reduce the target network's overfitting during training. Second, we further remove the pre-specified boundary requirement by using

a learnable discriminator to regularize the augmented data distribution. Finally, we replace the domain-specific transformation operations with general differentiable networks: Spatial Transformation Network [46] and Variational Autoencoders [47]. The proposed three methods are increasingly automatic and thus continually save human efforts in performing data augmentation. Experiments on human pose estimation, image classification, and medical image segmentation tasks demonstrate our automatic data augmentation methods can effectively boost deep models' performance without using additional training data. Our work has been published as follows:

- Jointly Optimize Data Augmentation and Network Training: Adversarial Data Augmentation in Human Pose Estimation, in CVPR 2018 [3]
- AdaTransform: Adaptive Data Transformation, in ICCV 2019 [48]
- OnlineAugment: Online Data Augmentation with Less Domain Knowledge, in ECCV 2020 [49]

**Contribution 2: Efficient network architecture**. We focus on promoting the efficiency of U-Net, a fully convolutional network consisting of a pair of contracting and expansive paths. U-Net is an essential type of CNN with broad applications to location-sensitive tasks such as image segmentation and human pose estimation. State-of-the-art methods usually stack multiple U-Nets to iteratively refine the outputs. However, stacked U-Nets may have tens of millions of float parameters and need to process high-resolution images, causing high memory consumption in training and low inference speed in testing. To boost the efficiency of stacked U-Nets, we seek solutions comprehensively from three perspectives: reducing parameter number, shrinking representation bit-width, and saving training memory. Specifically, we couple U-Net pairs by directly connecting their corresponding blocks. The shortcut connections can facilitate feature reuse across U-Nets, cutting down $\sim$70% parameters and $\sim$30% inference time. We further ternarize or binarize float parameters to reduce the model size by 16x or 32x. The data flow, including features and gradients, is also quantized to low-bit representations, saving $\sim$4x training memory. Additionally, we share memories for features in shortcut-connected blocks and decrease $\sim$40% training memory. Our quantized

CU-Net (coupled U-Nets) is evaluated extensively on human pose estimation and facial landmark localization. Despite the significant efficiency advantages, quantized CU-Net still obtains comparable accuracy as stacked U-Nets. Our research has produced the following publications:

- CU-Net: Coupled U-Nets, in BMVC 2018 [50]
- Quantized Densely Connected U-Nets for Efficient Landmark Localization, in ECCV 2018 [51]
- Towards Efficient U-Nets: A Coupled and Quantized Approach, in TPAMI 2019 [52]

**Contribution 3: Robust feature normalization**. We investigate how feature normalization can make CNN more robust to OOD data, which is different from previous works employing normalization for training acceleration and stabilization. The key bridge connecting normalization to OOD robustness is that a feature map's mean and variance can encode some style cues, such as image color, which is less critical than content information like object shape in object recognition. Inspired by the relation, we propose SelfNorm and CrossNorm to advance CNN's OOD robustness. SelfNorm uses attention to recalibrate a feature map's mean and variance. It can help emphasize essential styles and suppress trivial ones, lessening CNN's sensitivity to appearance changes in OOD data. CrossNorm performs style augmentation by exchanging means and variances between feature maps in training, reducing the CNN's style bias. Although SelfNorm and CrossNorm seem to oppose each other (style reduction v.s. style augmentation), they can collaborate to achieve better OOD robustness. Extensive experiments demonstrate that SelfNorm and CrossNorm are readily applicable to different domains (vision and language), tasks (classification and segmentation), and settings (supervised and semi-supervised) and can significantly boost state-of-the-art OOD robustness. The supportive publication of this work is given as follows:

- SelfNorm and CrossNorm for Out-of-Distribution Robustness, Arxiv 2021 [53]

## 1.3 Dissertation Outline

We organize the remaining thesis as follows. First, three automatic data augmentation methods are introduced in Chapters 2, 3, and 4. Then we present the efficient coupled U-Nets in Chapter 5 and two robust normalization approaches in Chapter 6. Chapter 7 concludes the dissertation and looks into several future directions.

# Chapter 2

# Jointly Optimize Data Augmentation and Network Training: Adversarial Data Augmentation in Human Pose Estimation

## 2.1 Introduction

Deep Neural Networks (DNNs) have achieved significant improvements in many computer vision tasks [54, 55, 46, 56]. A key ingredient for the success of state-of-the-art deep learning models is the availability of large amounts of training data. However, data collection and annotation are costly, and for many tasks, only a few training examples may be available. In addition, natural images usually follow a long-tail distribution [57, 58]. Effective training examples that lead to more robust classifiers may still be rare even if a large amount of data have been collected.

A common solution for this problem is to perform random data augmentation [59, 60]. Prior to being fed into the network, training images are heuristically jittered by predefined transformations (*e.g.*, scaling, rotating, occluding) to increase variations. This strategy is simple, but data augmentation and network training are still treated as isolated processes, leading to the following issues.

First, the entire training set is usually applied the same random data augmentation strategy without considering individual differences. This may produce many ineffective variations that are either too "hard" or too "easy" to help the network training [61, 62]. Second, random data augmentations can hardly match the dynamic training status since they are usually sampled from static distributions. Third, Gaussian distribution are widely used, which cannot address the long-tail issue since there would be a small chance to sample rare but useful augmentations.

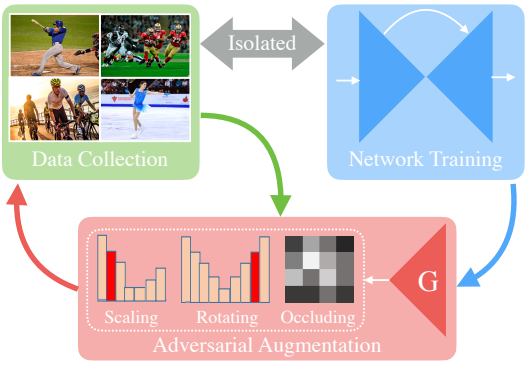



Figure 2.1: Data preparation and network training are usually isolated. We propose to bridge the two by generating adversarial augmentations online. The generations are conditioned to both training images and the status of the target network.

A natural question then arises: can data augmentation and network training be jointly optimized, so that effective augmentations can be generated online to improve the training?

In this work, we answer the above question by proposing a new approach that leverages adversarial learning for joint optimization of data augmentation and network training (see Figure 2.1). Specifically, we investigate the problem of human pose estimation, aiming to improve network training with bounded datasets. Note that our approach can be generalized to other vision tasks, such as face alignment [63] and instance segmentation [64, 65].

Given an off-the-shelf pose estimation network, our goal is to obtain improved training from a bounded dataset. Specifically, we propose an augmentation network that acts as a generator. It aims to create "hard" augmentations that intend to make the pose network fail. The pose network, on the other hand, is modeled as a discriminator. It evaluates the quality of the generations, and more importantly, tries to learn from the "hard" augmentations. The main idea is to generate adversarial data augmentations online, conditioned to both input images and the training status of the pose network. In other words, the augmentation network explores the weaknesses of the pose network which, at the same time, learns from adversarial augmentations for better performance.

Jointly optimizing the two networks is a non-trivial task. Our experiments indicate that a straightforward design, such as directly generating adversarial pixels [66, 67] or

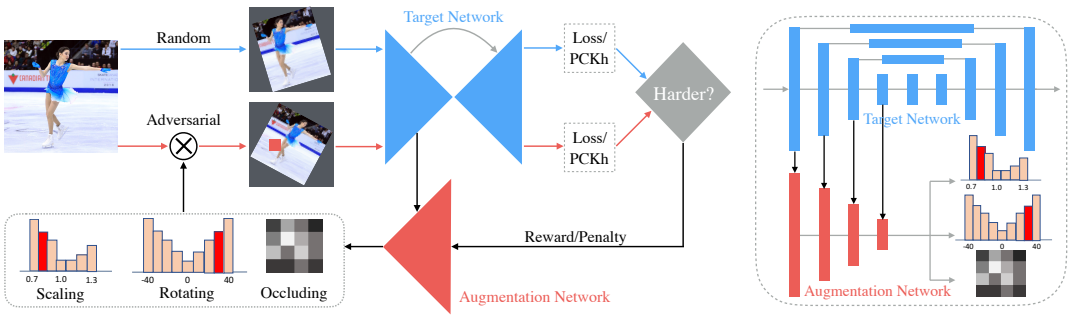



Figure 2.2: **Left**: Overview of our approach. We propose an augmentation network to help the training of the pose network. The former creates hard augmentations; the latter learn from generations and produces reward/penalty for model update. **Right**: Illustration of the augmentation network. Instead of raw images, it takes hierarchical features of an U-net as inputs.

deformations [62, 46], would yield problematic convergence behaviors (*e.g.* divergence or model collapse). Instead, the augmentation network is designed to generate adversarial distributions, from which augmentation operations (*i.e.* scaling, rotating, occluding) are sampled to create new data points. Besides, we propose a novel reward and penalty policy to address the issue of missing supervisions during the joint training. Moreover, instead of a raw image, the augmentation network is designed to take the byproduct, *i.e.* hierarchical features, of the pose network as the input. This can further improve the joint training efficiency using additional spatial constraints. To summarize, our key contributions are:

- To the best of our knowledge, we are the first to investigate the joint optimization of data augmentation and network training in human pose estimation.
- We propose an augmentation network to play a minimax game against the target network, by generating adversarial augmentations online.
- We take advantage of the widely used U-net design and propose a reward and penalty policy for the efficient joint training of the two networks.
- Strong performance on public benchmarks, e.g. MPII and LSP, as well as intensive ablation studies, validate our method substantially in various aspects.

## 2.2 Related Work

We provide a brief overview of previous methods that are most relevant to the proposed adversarial data augmentation in three categories.

**Adversarial learning.** Generative Adversarial Networks (GANs) [66, 68, 69] are designed as playing minimax games between generator and discriminator. Yu and Grauman [70] use GANs to synthesize image pairs to overcome the sparsity of supervision when learning to compare images. A-Fast-RCNN [62] uses GANs to generate deformations for object detection. Recent applications of GANs in human pose estimation include [71] and [72]. They both treat the pose estimation network as the generator and use a discriminator to provide additional supervision. However, in our design, the pose estimation network is treated as a discriminator, while the augmentation network is designed as a generator to create adversarial augmentations.

**Hard example mining.** It is widely used in training SVM models for object detection [73, 74, 61]. The idea is to perform an alternative optimization between model training and data selection. Hard example mining focuses on how to select hard examples from the training set for effective training. It cannot create new data that do not exist in the training set. In contrast, we propose an augmentation network (generator) to actively generate adversarial data augmentations. This will create new data points that may not exist in the training set to improve the pose network (discriminator) training.

**Human pose estimation.** DeepPose [60] proposed to use deep neural networks for human pose estimation. Since then, deep learning based methods started to dominate this area [75, 76, 77, 78, 79, 80, 81, 82, 83, 1]. For instance, Tompson *et al.* [84] used multiple branches of convolutional networks to fuse the features from an image pyramid and applied Markov Random Field for post-processing. Chen *et al.* [85] also tried to combine neural networks with graphical model inference to improve pose estimation accuracy.

Recently, cascade models have become popular for human pose estimation. They usually connect a series of deep neural networks in cascade to improve the estimation

in a stage-by-stage manner. For example, *Convolutional Pose Machines* [82] brings obvious improvements by cascading multiple networks and adding intermediate supervisions. Better performance is achieved with the stacked hourglass network architecture [1], which also relies on multi-stage pose estimation. More recently, Chu *et al.* [86] added some layers into the stacked hourglass network for attention modeling. Yang *et al.* [87] also enhanced its performance by using pyramid residual modules. In this work, instead of designing a new pose estimation network, we are more interested in how to jointly optimize data augmentation and network training so we can obtain improved training on any off-the-shelf deep neural network without looking for more data.

## 2.3 Adversarial Data Augmentation

Given a pre-designed pose network, *e.g.* the stacked hourglass pose estimator [1], our goal is to improve its training without looking for more data. Although random data augmentation is widely used, such augmentations that are sampled from static distributions can hardly follow the dynamic training status, which may produce many ineffective variations that are too "easy" to help the network training [61, 62].

Instead, we propose to leverage adversarial learning to optimize the data augmentation and the network training jointly. The main idea is to learn an augmentation network $G(\cdot|\theta_G)$ that generates "hard" augmentations that may increase the pose network loss. The pose network $D(\cdot|\theta_D)$, on the other hand, tries to learn from the adversarial augmentations and, at the same time, evaluates the quality of the generations. Please refer to Figure 2.2 for an overview of our approach.

**Generation path.** The augmentation network is designed as a generator. It outputs a set of distributions of augmentation operations. Mathematically, the augmentation network $G$ outputs adversarial augmentation $\tau_a(\cdot)$ that may increase $D$'s loss, compared with random augmentation $\tau_r(\cdot)$, by maximizing the expectation:

$$\max_{\theta_G} \mathop{\mathbb{E}}_{\mathbf{x}\sim\Omega} \mathop{\mathbb{E}}_{\substack{\tau_r\sim\Gamma \\ \tau_a\sim G(\mathbf{x},\theta_D)}} \mathcal{L}[D(\tau_a(\mathbf{x}),\mathbf{y})] - \mathcal{L}[D(\tau_r(\mathbf{x}),\mathbf{y})], \tag{2.1}$$

where $\Omega$ is the training image set and $\Gamma$ is the random augmentation space. $\mathcal{L}(\cdot,\cdot)$ is a predefined loss function and $\mathbf{y}$ is the image annotation. We highlight $G(\mathbf{x},\theta_D)$

to specify that the generation of $G$ is conditioned to both the input image $\mathbf{x}$ and the current status of the target network $D$.

**Discrimination path.** The pose network is designed as discriminator $D$, which plays two roles: 1) evaluating the generation quality as indicated in Equation equation 2.1; 2) trying to learn from adversarial generations for better performance by minimizing the expectation:

$$\min_{\theta_D} \mathop{\mathbb{E}}_{\mathbf{x}\sim\Omega} \mathop{\mathbb{E}}_{\tau_a\sim G(\mathbf{x},\theta_D)} \mathcal{L}[D(\tau_a(\mathbf{x}), \mathbf{y})], \tag{2.2}$$

where adversarial augmentation $\tau_a$ can better reflect the weakness of $D$ than random augmentation $\tau_r$, resulting in more effective network training.

**Joint training.** The joint training of $G$ and $D$ is a non-trivial task. Augmentation operations are usually not differentiable [62], which stops gradients from flowing from $D$ to $G$ in backpropagation. To solve this issue, we propose a reward and penalty policy to create online ground truth of $G$. So $G$ can always be updated to follow $D$'s training status. The details will be explained soon in Section 2.4.3.

It is crucial that $G$ generates distributions instead of direct operations [62] or adversarial pixels [67]. Our experiments indicate that, by sampling from distributions, the generation is more robust to outliers which may produce upside-down augmentations. Thus, there is less chance that $D$ would get trapped in a local optimum.

**Comparison with prior methods.** We want to stress that there is a sharp difference between our method and the recent adversarial human pose estimation techniques [71, 72]. The latter usually follow a common design that connects a pose network (generator) with an additional network (discriminator) to obtain adversarial loss. In contrast, we propose to learn an adversarial network (generator) to improve the pose network (discriminator), by jointly optimizing data augmentation and network training.

Our method is also different from others that perform online hard example mining [73, 61]. Our method can create new data points that may not exist in the dataset, whereas the latter is usually bounded by the dataset. An exception is [62] that uses GANs to generate deformations for object detection. However, how to jointly optimize data augmentation and network training, especially for human pose estimation, is still

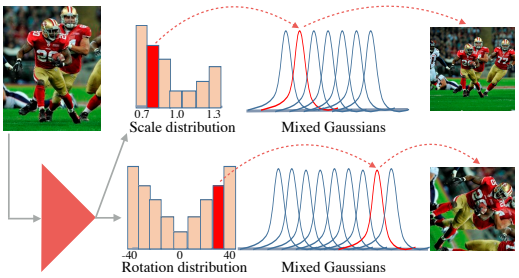



Figure 2.3: Adversarial scaling and rotating. Our generator predicts distributions of mixed Gaussian, from which scaling and rotating are then sampled to augment the training image.

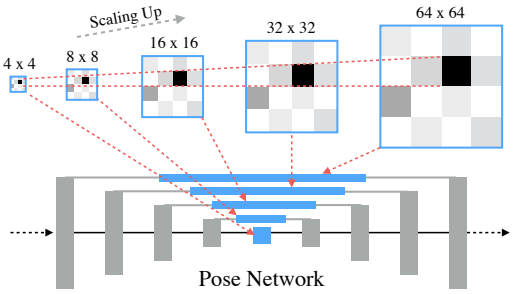



Figure 2.4: Adversarial Hierarchical Occluding. The occlusion mask is generated at the lowest resolution and then scaled up to apply on hierarchical bridge features of the pose network.

an open question without investigation.

## 2.4 Adversarial Human Pose Estimation

Our task is to improve the training of a pre-designed pose network. We take the widely used U-net design [1, 88] as an example. As illustrated in Figure 2.2 (right), the augmentation network follows an encoder architecture. It takes the bridged features of the U-net as inputs instead of raw images for efficient training. A set of distributions are then generated to sample three typical augmentations: scaling, rotating, and hierarchical occluding. Furthermore, we propose a reward and penalty strategy for efficient joint training.

### 2.4.1 Adversarial Scaling and Rotating (ASR)

The augmentation network generates adversarial augmentations by scaling and rotating the training images. The pose network then learns from the adversarial augmentations for more effective training. In our experiments, we find that a direct generation would collapse the training because it would easily generate upside-down augmentations that are the hardest in most cases. Instead, we divide the augmentation ranges into $m$ and $n$ bins (*e.g.* $m = 7$ for scaling and $n = 9$ for rotating). Each bin corresponds to a small bounded Gaussian. The augmentation network will first predict distributions over

scaling and rotating bins. Then, the corresponding Gaussian is activated by sampling from distributions. Please refer to Figure 2.3 for an illustration of the sampling process.

**ASR pre-training.** It is crucial to pre-train the augmentation network so it can obtain the sense of augmentation distributions before the joint training. For every training image, we can sample totally $m \times n$ augmentations, each of which is drawn from a pair of Gaussians. The augmentations are then fed forward into the target network to calculate the loss which represents how "difficult" the augmentation is. We accumulate $m \times n$ losses into the corresponding scaling and rotation bins. By normalizing the sum of bins to 1, we generate two vectors of probabilities, $P^s \in \mathbb{R}^m$ and $P^r \in \mathbb{R}^n$, which approximate the ground truth of scaling and rotation distributions, respectively.

Given the ground-truth distributions $P^s$ and $P^r$, we propose a KL-divergence loss to pre-train the augmentation network for scaling and rotating:

$$\mathcal{L}_{SR} = \sum_{i=1}^{m} P_i^s \log \frac{P_i^s}{\tilde{P}_i^s} + \sum_{i=1}^{n} P_i^r \log \frac{P_i^r}{\tilde{P}_i^r}, \tag{2.3}$$

where $\tilde{P}^s \in \mathbb{R}^m$ and $\tilde{P}^r \in \mathbb{R}^n$ are the predicted distributions following the above generation procedure, and $m$ and $n$ are the numbers of scale and rotation bins.

**Discussion.** Predicting distributions instead of direct augmentations has two advantages. First, it introduces uncertainties to avoid upside-down augmentations during the pre-training. Second, it helps to address the issue of missing ground truth during the joint training, which will be explained in Section 2.4.3. In our design, the scaling and rotating are directly applied on training images instead of deep features [62]. The reason is we want to preserve the location correspondence between image pixels and landmark coordinates. Otherwise, we might hurt the localization accuracy once the intermediate feature maps are disturbed.

### 2.4.2 Adversarial Hierarchical Occluding (AHO)

In addition to scaling and rotating, the augmentation network also generates occluding operations to make the task even "harder". The human body has a linked structure where joint locations are highly correlated to each other. By occluding parts of the image, the pose network is encouraged to learn strong references among visible and

invisible joints [89].

Different from scaling and rotating, we find that it is more effective to occlude deep features instead of image pixels. Specifically, the augmentation network generates a mask indicating which part of features to be occluded so that the pose network has more estimation errors. We only generate the mask at the lowest resolution of $4 \times 4$. The mask is then scaled up to $64 \times 64$ to apply on bridge features of the U-net. Figure 2.4 explains the proposed hierarchical occluding.

**AHO pre-training.** Similar to scaling and rotating, the augmentation network predicts an occluding distribution instead of an instance occluding mask. The first task is to create the ground truth of the occluding distribution. The idea is to assign values into a grid of $w \times h$ (*e.g.* $w = h = 4$). The value indicates the importance of the features at the corresponding cell. To achieve this, we vote a joint to one of the $w \times h$ cells according to its coordinates. By counting all joints from all images and normalizing the sum of cells to 1, we generate a heat map $P^o \in \mathbb{R}^{w \times h}$, which approximates the ground truth of the occluding distribution.

Given the ground-truth distribution $P^o$, we propose a KL-divergence loss to pre-train the AHO task:

$$\mathcal{L}_{AHO} = \sum_{i=1}^{h} \sum_{j=1}^{w} P^o_{i,j} \log \frac{P^o_{i,j}}{\tilde{P}^o_{i,j}}, \tag{2.4}$$

where $\tilde{P}^o \in \mathbb{R}^{w \times h}$ is the heat map predicted by the augmentation network. To generate the occluding mask, we sample one or two cells according to $\tilde{P}^o$, which are labeled as 0, while the rest are labeled as 1.

**Discussion.** Intuitively, there are three ways to apply hierarchical occluding: (1) a single mask scales up from the lowest to the highest resolutions, (2) a single mask scales down from the highest to the lowest resolutions, and (3) independent masks are generated at different resolutions. We exclusively use the first design in our approach since it would occlude more than needed due to the large receptive field in the second case, and the occluded information may be compensated at other resolutions in the third case.

**Algorithm 1:** Training scheme of a mini batch

**Input:** Mini-batch $\mathbf{X}$, augmentation net $G$, pose net $D$.
**Output:** G, D.

```
Randomly and equally divide X into X_1, X_2 and X_3;
Train D using X_1;
Train D, G using X_2 with ASR following Alg. 2;
Train D, G using X_3 with AHO following Alg. 2;
```

### 2.4.3 Joint Training of Two Networks

Once ASR and AHO are pre-trained, we can jointly optimize the augmentation network and the pose network. As we mentioned in Sec. 2.3, this is a non-trivial task since the augmentation ground truth is missing. A naive approach could be repeating the pre-training process as described in Section 2.4.1 and Section 2.4.2 online. However, it would be extremely time-consuming since there are a large number of augmentation combinations.

**Reward and penalty.** Instead, we propose a reward and penalty policy to address this issue. The key idea is that the augmentation network prediction should be updated according to the current target network status, while its quality should be evaluated by comparing with a reference.

To this end, we sample a pair of augmentations for each image: 1) an adversarial augmentation $\tau_a$ and 2) a random augmentation $\tau_r$, as indicated in Equation equation 2.1. If the adversarial augmentation is harder than the random one, we reward the augmentation network by increasing the probability of the sampled bin (ASR) or cell (AHO). Otherwise, we penalize it by decreasing the probability accordingly.

Mathematically, let $\tilde{P} \in \mathbb{R}^k$ denotes the predicted distribution of the augmentation network. $P \in \mathbb{R}^k$ denotes the ground truth we are looking for. $k$ is the number of bins (ASR) or cells (AHO) and $i$ is the sampled one.

If the adversarial augmentation $\tau_a$ leads to higher pose network loss (more "difficult") comparing with the reference (a random augmentation $\tau_r$), we update $P$ by rewarding:

$$P_i = \tilde{P}_i + \alpha\tilde{P}_i; \quad P_j = \tilde{P}_j - \frac{\alpha\tilde{P}_i}{k-1}, \forall j \neq i. \tag{2.5}$$

**Algorithm 2:** Training scheme of one image.

**Input:** Image $\mathbf{x}$, augmentation network $G$, pose network $D$.
**Output:** G, D.

Forward $D$ to get bridge features $\mathbf{f}$;
Forward $G$ with $\mathbf{f}$ to get a distribution $P$;
Sample an adversarial augmentation $\tilde{\mathbf{x}}$ from $P$;
Forward $D$ with $\tilde{\mathbf{x}}$ to compute loss $\tilde{\mathcal{L}}$;
Random augment $\mathbf{x}$ to get $\hat{\mathbf{x}}$;
Forward $D$ with $\hat{\mathbf{x}}$ to compute loss $\hat{\mathcal{L}}$;
Compare $\tilde{\mathcal{L}}$ with $\hat{\mathcal{L}}$ to update $G$ using equation 2.3 and equation 2.4;
Update $D$;

Similarly, if $\tau_a$ leads to lower pose network loss (less "difficult") comparing with $\tau_r$, we update $P$ by penalizing:

$$P_i = \tilde{P}_i - \beta\tilde{P}_i; \quad P_j = \tilde{P}_j - \frac{\beta\tilde{P}_i}{k-1}, \forall j \neq i, \tag{2.6}$$

where $0 < \alpha, \beta \leq 1$ are hyperparameters that control the amount of reward and penalty. The augmentation network keeps updating online, regardless of being rewarded or penalized, generating adversarial augmentations that intend to improve the pose network.

**Discussion.** The pose network can learn from the ordinary random augmentation to maintain its regular performance. More importantly, it can also learn from the adversarial augmentations to achieve better performance. The adversary augmentations may become too hard for the pose network if we apply ASR and AHO simultaneously. Thus, we alternately apply ASR and AHO on different images. Here we equally split every mini batch into three shares: one performs the random data augmentation, one performs ASR augmentation, and one performs AHO augmentation. Algorithm 1 provides a detailed account.

## 2.5 Experiments

In this section, we first show the visualization of network training states to verify the motivation of doing adversarial dynamic augmentation. Then we quantitatively evaluate the effectiveness of different components in the method and further compare with state-of-the-art approaches.

### 2.5.1 Experimental Settings

We use stacked hourglass [1] as the pose network. The augmentation network takes the top-down part of an hourglass and only uses one cell module in each resolution block. To evaluate the generalization capability of the proposed adversarial augmentation, we tested two types of modules: *Residual module* [10] and *Dense block* [90].

**Network design.** We test both residual and dense hourglass networks in our component evaluation experiments. For residual hourglass, each residual module has a bottleneck structure of BN-ReLU-Conv(1x1)-BN-ReLU-Conv(3x3)-BN-ReLU-Conv(1x1). The input/output dimension of each bottleneck is 256. The two $1 \times 1$ convolutions are used to halve and double the feature dimensions.

For dense hourglass, each module is a bottleneck structure of BN-ReLU-Conv(1x1)-BN-ReLU-Conv(3x3), with neck size 4, growth rate 32, and input dimension 128. The dimension increases by 32 after each dense layer. At the end of each dense block, we use BN-ReLU-Conv(1x1) to reduce the dimension to 128. We use the standard 8 stacked residual hourglasses [1] as our baseline when compared with state-of-the-art methods.

**Datasets.** We evaluate the proposed adversarial human pose estimation on two benchmark datasets: MPII Human Pose [91] and Leeds Sports Pose (LSP) [92]. MPII is collected from YouTube videos with a broad range of human activities. It has 25K images and 40K annotated persons (29K for training and 11K for testing). Following [84], we sample 3K samples from the training set for validation. Each person has 16 labeled joints.

The LSP dataset contains images from many sports scenes. Its extended version has 11K training samples and 1K testing samples. Each person in LSP has 14 labeled joints. Since there are usually multiple people in one image, we crop around each person and resize the image to $256 \times 256$. Typically, random scaling (0.75-1.25), rotating (-/+30°) and flipping are used to augment the data.

**Training.** We use PyTorch for the implementation. RMSProp [93] is used to optimize the networks. The adversarial training contains three stages. We first train hourglass for a few epochs with a learning rate of $2.5 \times 10^{-4}$. Then we freeze the

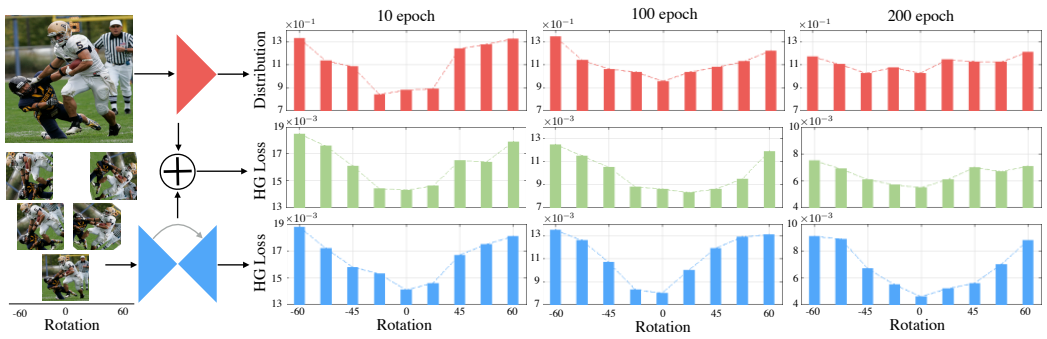



Figure 2.5: Network training status visualization: predicted rotating distributions of the agumentation network (**Top**), loss distributions of pose network trained by adversarial (**Middle**) and random (**Bottom**) rotating augmentations. The augmentation network predicts rotating distributions matching the loss distributions of pose network, according to the first two rows. The loss distribution in the last row maintains a similar shape all the time due to the fixed Gaussian sampling distribution.

hourglass model and use it train the AHO and ASR networks with learning rate $2.5 \times 10^{-4}$. Once they are pre-trained, we lower the learning rates of AHO and ASR networks to $5 \times 10^{-5}$ and jointly train the three networks. The learning rate of the target network is decayed to $5 \times 10^{-5}$ after the validation accuracy plateaus. In all experiments, the Percentage of Correct Keypoints (PCK) [94] is used to measure the pose estimation accuracy.

### 2.5.2 Visualization of the Training Status

In this experiment, we use a single residual hourglass. Each residual block contains 3 residual modules. We are interested in knowing how the pose network handles human images with different data augmentations: rotating, scaling and occluding. Since our method treats these three variations in a similar way, we take rotating as an example. More specifically, we visualize the loss distribution of hourglass on images with different rotations.

**Random data augmentation.** We train the pose network using random rotating sampled from a zero-centered Gaussian distribution as shown in the last row of Figure 2.5. We then test the trained pose network by applying the same rotating distribution on the testing data. We find that, at different training stages (training epochs), the target network loss always presents an inverted Gaussian-like distribution.

**Adversarial data augmentation.** In the beginning, the loss distribution of the pose network is similar to the case of random data augmentation. Since the pose network is pre-trained by the random data augmentation. However, the distribution becomes flatter as the training continues, which means the pose network could better handle the rotated images. The pose network learns from the adversarial data augmentation generated by the augmentation network.

**Augmentation network training status.** The status can be visualized by applying the generated rotating augmentation. Comparing the first two rows in Figure 2.5, we can find that the generated rotating distribution is similar to the loss distribution of the pose network. This means that the augmentation network could track the training status of the target network and generate effective data augmentations.

### 2.5.3 Component Evaluation

We first verify the effectiveness of ASR and AHO in both residual and dense hourglasses. We use 3 residual bottlenecks in each block of residual hourglass. In dense hourglass, we use 6 densely connected bottlenecks in one dense block. Note that the size of dense hourglass model is less than half of the residual hourglass. In Table 2.1, we compare variants of adversarial data augmentation on PCKh@0.5. Figure 2.6 shows the improvement of adversarial data augmentation compared with random data augmentation, on PCKh threshold from 0.1 to 0.5.

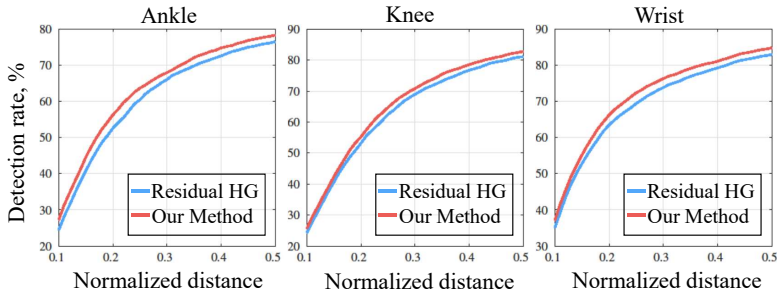
**ASR only.** Table 2.1 shows that ASR improves the accuracy of all the keypoints on both residual and dense hourglass, with average improvements of 0.5% and 0.5% respectively. This indicates that the generated adversarial scaling and rotating augmentations are effictive in training the pose network.

**AHO only.** Table 2.1 shows that AHO improves accuracy on both residual and dense hourglass, with average improvements of 0.4% and 0.4% respectively. Similarly, the pose network can also learn improved inference from the adversarial occluding generated by the augmentation network.

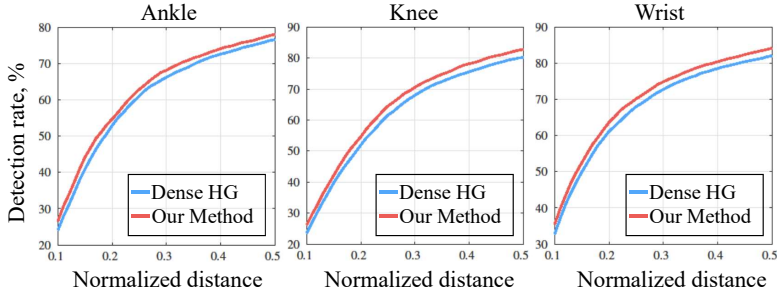
**ASR and AHO.** Applying both ASR and AHO can further improve the accuracy by 0.4%, compared with applying either of them. Figure 2.6 shows that ASR and AHO

Table 2.1: Comparison of random and adversarial data augmentation on the MPII validation set measured by PCKh@0.5.

| | Residual hourglass (size: **38M**) | | | | | | | | Dense hourglass (size: **18M**) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Head | Sho. | Elb. | Wri. | Hip | Knee | Ank. | Mean | Head | Sho. | Elb. | Wri. | Hip | Knee | Ank. | Mean |
| Random Aug. | 97.2 | 94.8 | 87.8 | 83.4 | 87.8 | 81.3 | 76.5 | 87.0 | 97.1 | 94.6 | 87.9 | 83.0 | 87.5 | 81.2 | 76.6 | 86.8 |
| +ASR | 97.3 | **95.2** | 88.2 | 84.2 | 88.2 | 81.8 | 77.3 | 87.5 | **97.2** | 95.0 | 88.3 | 83.5 | 87.7 | 81.8 | 77.4 | 87.3 |
| +AHO | 97.3 | 95.0 | 88.2 | 83.6 | 88.0 | 82.2 | 77.6 | 87.4 | 97.1 | 94.8 | 88.2 | 83.6 | 87.6 | 81.7 | 77.5 | 87.2 |
| +ASR+AHO | **97.3** | 95.1 | **88.7** | **84.7** | **88.4** | **82.5** | **78.1** | **87.8** | **97.2** | **95.2** | **88.8** | **84.1** | **88.1** | **82.0** | **77.9** | **87.6** |




(a)




(b)

Figure 2.6: Comparison of random and adversarial data augmentations on MPII validation set using PCKh@0.1-0.5. Consistent improvements on a range of normalized distances could be observed on both residual modules (**left**) and dense blocks (**right**).

can significantly improve the localization accuracy especially for joints that are usually more difficult to localize, such as ankle, knee and wrist.

**Dense hourglass vs Residual hourglass.** Table 2.1 also shows that the dense hourglass has comparable performance in terms of pose estimation accuracy, but much more parameter efficiency than the residual design (18M *vs.* 38M). The dense design facilitates the gradient flow through the direct connections among different feature blocks, which uses fewer parameters without sacrificing the estimation accuracy.

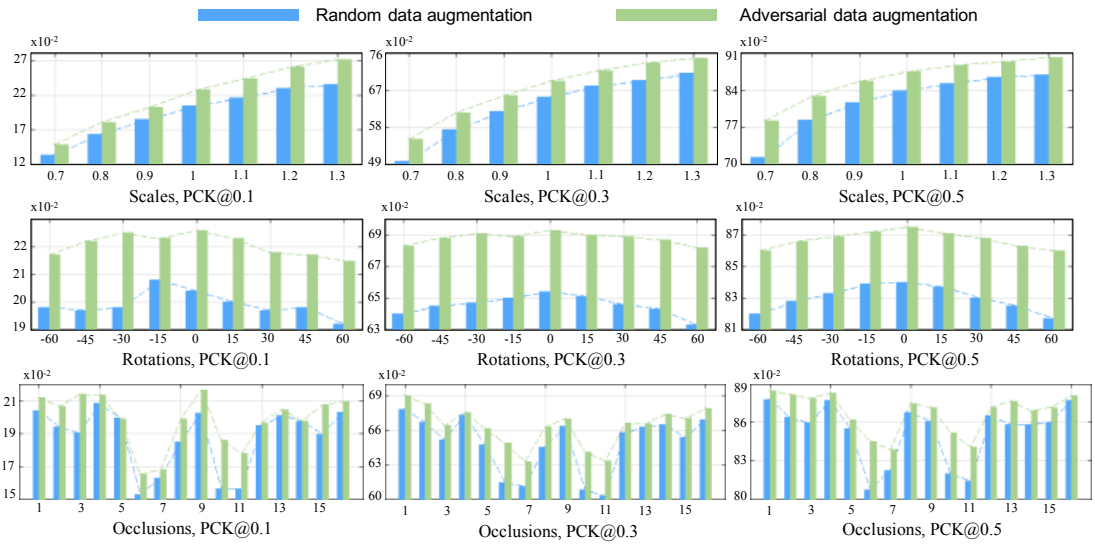



Figure 2.7: PCKh@0.1 (**Left column**), PCKh@0.3 (**Middle column**) and PCKh@0.5 (**Right column**) comparisons using different image scales (**Top row**), rotations(**Middle row**), and occlusions (**Bottom row**) on the MPII validation set. The adversarially trained hourglass network consistently outperforms the ordinary one when the test images have varied scales, rotations, and occlusions.

### 2.5.4 Robustness Evaluation

We evaluate the robustness to different image scales, rotations, and occlusions when 8 stacked hourglasses are trained with the adversarial data augmentation. Random data augmentation serves as the baseline and we show the PCKh comparisons under three different thresholds (PCKh@0.1, PCKh@0.3, PCKh@0.5) on the MPII validation set.

**Robustness to scale changes.** According to the first row in Figure 2.7, our adversarial data augmentation consistently outperforms the random augmentation when scale scale changes from 0.7 to 1.3. Under PCKh@0.3 and PCKh@0.5, the maximum gaps are 5.4% and 7%, both happening at the smallest scale 0.7. This indicates that the adversarially trained hourglass has more advantages on dealing with humans with small scales. When measuring with PCKh@0.1, the largest gap appears at the highest scale 1.3, while scale 0.7 has the smallest gap. This is due to the fact that the hourlgass discriminability is inherently limited. The largest resolution in hourlgass is 64x64. Therefore, it is hard to estimate joint positions of small scale humans with high precision.

**Robustness to rotation changes.** The second row of Figure 2.7 shows that the adversarially trained hourglass has reliable gains when rotation changes from $-60°$ to $60°$. More specifically, the maximum gaps, measured by PCKh@0.1, PCKh@0.3 and PCKh@0.5, are 2.7%, 4.9% and 4.3%, respectively. The last two gaps correspond to both the largest rotation 60 degrees, which means that the adversarially trained hourglass can better handle humans with large rotations. However, the first gap 2.7% comes up when rotation is $-30°$. Compared with the last two, the first is not only smaller but also does not happen at the largest rotations. Similar to the analysis in comparisons with respect to scale changes, this irregular phenomenon is caused by the fact that the hourglass of resolution $64 \times 64$ has troubles in predicting high precision locations. Thus, its predictions measured by PCKh@0.1 have much more uncertainties.

**Robustness to occlusions.** We generate the occlusion mask by occluding 1 of 16 cells at the lowest resolution of $4 \times 4$. The mask is then scaled up to occlude the bridge feature maps from $4 \times 4$ to $64 \times 64$. The adversarially trained hourglass outperforms the ordinary one in most settings. The maximum gaps, measured by PCKh@0.1, PCKh@0.3 and PCKh@0.5, are 2.4% 3.4% and 3.8%, as shown in the third row of Figure 2.7. We could find that the large gaps focus on the bins around the center of feature maps, which contain the most useful features. The adversarial data augmentation makes the hourglass learn how to infer the locations under occlusions. The gap measured by PCKh@0.1 is relatively smaller due to that it is more difficult for the hourglass to predict high precision locations.

### 2.5.5 Comparing with State-of-the-art Methods

**Quantitative comparison.** To compare with state-of-the-art methods, we apply the proposed adversarial data augmentation to train the hourglasses network (totally 8 stacked) [1]. The bridge features generated by the first hourglass network in the stack are input into the adversarial network. The same hierarchical occluding masks are applied to every hourglass network in the stack. Table 2.2 compares PCKh@0.5 accuracy of different methods on the MPII dataset. The proposed adversarial data augmentation can improve the baseline [1] by 0.6%, which achieves state-of-the-art

Table 2.2: PCKh@0.5 on the MPII test set. Our adversarial data augmentation improves baseline stacked HGs(8) [1].

| Method | Head | Sho. | Elb. | Wri. | Hip | Knee | Ank. | Mean |
|---|---|---|---|---|---|---|---|---|
| Pishchulin *et al.*[95] | 74.3 | 49.0 | 40.8 | 34.1 | 36.5 | 34.4 | 35.2 | 44.1 |
| Tompson *et al.*[84] | 95.8 | 90.3 | 80.5 | 74.3 | 77.6 | 69.7 | 62.8 | 79.6 |
| Carreira *et al.*[75] | 95.7 | 91.7 | 81.7 | 72.4 | 82.8 | 73.2 | 66.4 | 81.3 |
| Tompson *et al.*[76] | 96.1 | 91.9 | 83.9 | 77.8 | 80.9 | 72.3 | 64.8 | 82.0 |
| Hu *et al.*[77] | 95.0 | 91.6 | 83.0 | 76.6 | 81.9 | 74.5 | 69.5 | 82.4 |
| Pishchulin *et al.*[78] | 94.1 | 90.2 | 83.4 | 77.3 | 82.6 | 75.7 | 68.6 | 82.4 |
| Lifshitz *et al.*[79] | 97.8 | 93.3 | 85.7 | 80.4 | 85.3 | 76.6 | 70.2 | 85.0 |
| Gkioxary *et al.*[80] | 96.2 | 93.1 | 86.7 | 82.1 | 85.2 | 81.4 | 74.1 | 86.1 |
| Rafi *et al.*[96] | 97.2 | 93.9 | 86.4 | 81.3 | 86.8 | 80.6 | 73.4 | 86.3 |
| Belagiannis *et al.*[97] | 97.7 | 95.0 | 88.2 | 83.0 | 87.9 | 82.6 | 78.4 | 88.1 |
| Insafutdinov *et al.*[81] | 96.8 | 95.2 | 89.3 | 84.4 | 88.4 | 83.4 | 78.0 | 88.5 |
| Wei *et al.*[82] | 97.8 | 95.0 | 88.7 | 84.0 | 88.4 | 82.8 | 79.4 | 88.5 |
| Bulat *et al.*[83] | 97.9 | 95.1 | 89.9 | 85.3 | 89.4 | 85.7 | 81.7 | 89.7 |
| Chu *et al.*[86] | **98.5** | 96.3 | 91.9 | 88.1 | 90.6 | **88.0** | **85.0** | **91.5** |
| Stacked HGs(8) [1] | 98.2 | 96.3 | 91.2 | 87.1 | 90.1 | 87.4 | 83.6 | 90.9 |
| Ours: +ASR+AHO | 98.1 | **96.6** | **92.5** | **88.4** | **90.7** | 87.7 | 83.5 | **91.5** |

performance. Table 2.3 compares PCK@0.2 accuracy of different methods on the LSP dataset. Again, our method can improve the baseline [1] by 1.5%, which significantly outperforms state-of-the-art methods.

**Qualitative Comparison.** Figure 2.8 shows qualitative comparisons. We compare the random and adversarial data augmentation. We can observer the improvement resulted from the adversarial data augmentation. Interestingly, the pose network could handle the left-right confusions after the adversarial training.

## 2.6 Summary

In this work, we have proposed the adversarial data augmentation modeled as a generative adversarial learning problem. An effective training scheme with the reward and penalty policy is given to jointly optimize both target and data augmentation networks. We apply adversarial data augmentation to human pose estimation and design adversarial scaling and rotating as well as adversarial hierarchical occluding to boost existing pose estimators. Experiments on benchmark datasets show that adversarial

Table 2.3: PCK@0.2 on the LSP dataset. Clear improvements are observed over the baseline stacked HGs(8) [1].

| Method | Head | Sho. | Elb. | Wri. | Hip | Knee | Ank. | Mean |
|---|---|---|---|---|---|---|---|---|
| Belagiannis *et al.*[97] | 95.2 | 89.0 | 81.5 | 77.0 | 83.7 | 87.0 | 82.8 | 85.2 |
| Lifshitz *et al.*[79] | 96.8 | 89.0 | 82.7 | 79.1 | 90.9 | 86.0 | 82.5 | 86.7 |
| Pishchulin *et al.*[78] | 97.0 | 91.0 | 83.8 | 78.1 | 91.0 | 86.7 | 82.0 | 87.1 |
| Insafutdinov *et al.*[81] | 97.4 | 92.7 | 87.5 | 84.4 | 91.5 | 89.9 | 87.2 | 90.1 |
| Wei *et al.*[82] | 97.8 | 92.5 | 87.0 | 83.9 | 91.5 | 90.8 | 89.9 | 90.5 |
| Bulat *et al.*[83] | 97.2 | 92.1 | 88.1 | 85.2 | 92.2 | 91.4 | 88.7 | 90.7 |
| Chu *et al.*[86] | 98.1 | 93.7 | 89.3 | 86.9 | 93.4 | 94.0 | 92.5 | 92.6 |
| Stacked HGs(8) [1] | 98.2 | 94.0 | 91.2 | 87.2 | 93.5 | 94.5 | 92.6 | 93.0 |
| **Ours: ASR+AHO** | **98.6** | **95.3** | **92.8** | **90.0** | **94.8** | **95.3** | **94.5** | **94.5** |

Figure 2.8: Comparisons of the same Stacked HG network trained using random data augmentation (**top**) and adversarial data augmentation (**bottom**). Note the improvement on challenging joints (*e.g.* ankle, elbow, wrist), and left-right confusion.

data augmentation could improve state-of-the-art pose networks.

# Chapter 3

# AdaTransform: Adaptive Data Transformation

## 3.1 Introduction

The remarkable success of deep learning, from a data perspective, benefits from the ability to optimize millions of free parameters [10, 90] to capture extensive data variance. Yet, sufficient data varieties are not always available in practice due to data scarcity and annotation cost [98].

The technique of perturbing data without changing class labels, also known as *data augmentation*, is widely used to address this issue. Generally speaking, data augmentation can be either sampled from predefined distributions or generated by learnable agents. The former, known as *random augmentation* [26, 55], usually relies on handcraft rules without optimization, yielding insufficient training. The latter, known as *auto or adversarial augmentation* [99, 62, 3], also suffers from various limitations.

*Auto augmentation* [99] explores a huge solution space to achieve an optimal solution on the validation set, which is extremely time-consuming. The network training has to be repeated $15,000$ times to get the final policy. *Adversarial augmentation*, on the other hand, follows a greedy design to speed up learning. However, the current designs [62, 3] rely on comprehensive domain knowledge to specify the transformation types and boundaries. This inevitably results in restricted transformation space. Moreover, previous methods mainly focus on network training, neglecting the potential to apply data transformation in testing.

This raises research questions: 1) Can we learn data transformation more efficiently? 2) Can we explore the transformation space (types and boundaries) without comprehensive domain knowledge? 3) Can data transformation also help improve network

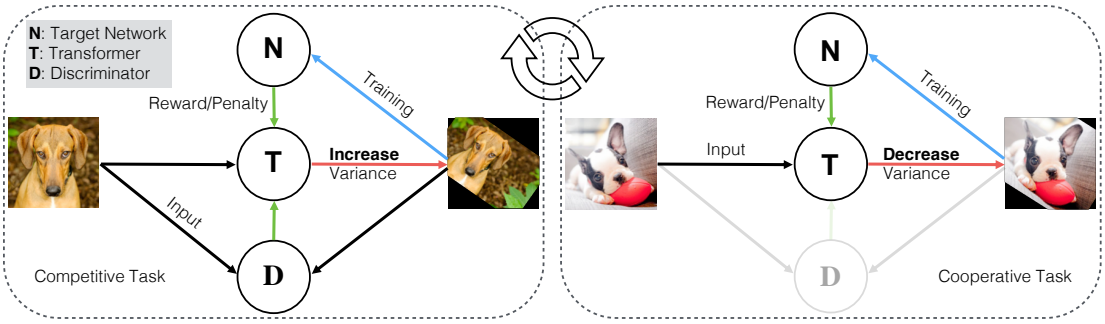



Figure 3.1: Overview of the adaptive data transformation. It consists of two tasks: competitive training and cooperative testing, and three components: a transformer $T$, a discriminator $D$, and a target network $N$. $T$ increases the training data variance by competing with both $D$ and $N$. It also cooperates with $N$ in testing to reduce the data variance.

deploying?

In this work, we answer the questions by proposing AdaTransform: adaptive data transformation. We leverage reinforcement learning in conjunction with adversarial training to compose meta-transformations (discrete transformation operations). This enables us to efficiently explore a large transformation space with limited domain knowledge.

Specifically, we learn data transformation in bi-direction: At the training stage, AdaTransform performs *a competitive task to increase data variance*, reducing overfitting; at the testing stage, AdaTransform performs *a cooperative task to decrease data variance*, yielding improved deploying. The two tasks are learned through optimizing a triplet: a transformer, a discriminator, and a target network, as illustrated in Figure 3.1. To summarize, our key contributions are:

- To the best of our knowledge, we are the first to investigate adaptive data transformation in order to improve both network training and testing.
- We propose to learn a competitive task (for training) and a cooperative task (for testing) simultaneously by jointly optimizing a triplet online.
- AdaTransform can automatically and efficiently explore the data transformation space, yielding a highly flexible and versatile solution for broad applications.
- Extensive experiments on *image classification*, *human pose estimation*, and *face*

*alignment* prove the favorable performance of AdaTransform especially when testing perturbations exist.

## 3.2 Related Work

We provide a brief overview of related works in the categories of data transformation, adversarial learning, reinforcement learning, hard example mining, human pose estimation, and face alignment.

**Data transformation.** Data transformations are commonly used to augment the training data [10, 55]. Recently, adversarial data augmentation [62, 3] has been proposed to train deep models. But they heavily rely on human knowledge and can only handle limited transformations. Some works [2, 99] try to learn data augmentation policy automatically. However, either they suffer from severe efficiency issues [2] or the policy learning is isolated from the target network training [99]. The high computational cost is due to the optimization of validation accuracy. The lack of joint optimization with the target network prevents it from dynamically increasing the data variance based on the individual images and target network state. Others [100, 101] learn to transfer the data transformations from large datasets to augment few-shot examples. The above methods are only to augment training data but cannot reduce the testing data variance. The Spatial Transformer Network (STN) [46] is designed to reduce the spatial variance of data. However, it can only handle differentiable spatial transformations, largely restricting its applications. Besides, it is only for the variance reduction but cannot increase the training data variance.

**Adversarial learning.** Generative Adversarial Networks (GANs) [66] includes two networks: generator and discriminator which compete against each other to improve generation performance. GANs are widely used in the image generations [66, 69] and translations [102]. Here we use the transformer to transform input images. It competes with the discriminator to make the transformed images still realistic but different from the original ones.

**Reinforcement learning.** In reinforcement learning (RL), an agent takes actions and then receives feedback from the environment, which may reward or penalize it. The agent learns to maximize its reward by taking appropriate actions. Reinforcement learning has been used with deep learning to play the game Go [103], search neural network architecture [104], etc. In this work, we use it to teach the transformer to handle data transformations.

**Hard example mining.** Hard example mining usually alternates between optimizing models and updating training data. Once a model is optimized on the current training set, it is used to collect more hard data for further training. This method was used in training SVM models for object detection [73]. Recently, Shrivastava *et al.* [61] adapted it into the neural network based object detector. The hard example mining focuses on selecting hard examples from existing data, the adaptive data transformation actively transforms the data to either increase or reduce their variance.

**Human pose estimation.** With recent advances in Deep Neural Networks (DNNs), image-based human pose estimation has achieved significant progress in the past few years [60, 84, 75]. DeepPose [60] is one of the first attempts of using DNNs for human pose estimation. Recently, multi-stage human pose prediction methods such as Convolutional Pose Machine [82] and stacked hourglasses [1] have become popular. The prediction results could be refined state-by-stage. Instead of designing a new pose estimator, we improve pose estimation performance by increasing the training data variance and reducing the testing data variance.

**Face alignment.** Similarly, DNNs have largely reshaped the field of face alignment. Traditional methods like [105] could be easily outperformed by the DNNs based [106, 107]. In the recent Menpo Facial Landmark Localization Challenge [108], stacked hourglasses [1] achieves state-of-the-art performance. Given an off-the-shelf face alignment DNN, the adaptive data transformation can be used to improve its performance.

## 3.3 Problem Definition and Task Modeling

Given a target network, *e.g.* an image classifier [10] or a human pose estimator [84], the adaptive data transformation, named as **AdaTransform**, aims to improve both training and testing of the target network. More specifically, the agent performs two different tasks: (1) At the training stage, it performs a **competitive task** to increase data variance, improving the training of the target network. (2) At the testing stage, it performs a **cooperative task** to reduce data variance, boosting its testing performance. The two tasks are learned simultaneously by jointly optimizing a triplet: a transformer $T$, a discriminator $D$, and a target network $N$. An illustration is given in Figure 3.1.

### 3.3.1 Transformer T

The transformer $T$ is designed to increase the data variance in the competitive task, while it learns to decrease the data variance in the cooperative task.

**Transformation Definition.** Transformation is domain-specific. It relies on both the data type and the target problem. Data of different modalities have dissimilar transformations. For example, images can utilize scale and rotation, while word replacement and switch may happen in text data.

Further, a transformation must preserve the data property of interest in the target problem. For instance, the shear operation can be applied in image classification since it does not change the image class labels. However, it is not a good choice for face recognition as it may alter the identity. The AdaTransform only needs limited domain knowledge to specify some meta-transformations. Then $T$ learns to compose them for both competitive and cooperative tasks.

**Competitive task.** $T$ learns to enlarge the data variance in training through increasing the loss of target network $N$. At the same time, it tries to fool the discriminator $D$ by making the transformed data realistic. Thus, $T$ must learn to satisfy the constraints from both $N$ and $D$:

$$\max_{\theta_T} \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\sim\Omega} \mathop{\mathbb{E}}_{\tau\sim T(\mathbf{x},0)} [\mathcal{L}(N(\tau(\mathbf{x})),\mathbf{y}) + \lambda \log(D(\tau(\mathbf{x})))], \tag{3.1}$$

where $\Omega$ is the training data, and $\tau$ is the transformation operation sampled from $T(x, 0)$ in the competitive mode. $\mathcal{L}(\cdot, \cdot)$ is a predefined target loss function. $\lambda$ balances the weight of two losses. $T$ competes with both $N$ and $D$ in the competitive task. The competitive $T$ is trained and applied to the training data.

**Cooperative task.** $T$ also learns to reduce the data variance by lowering the loss of target network $N$:

$$\min_{\theta_T} \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\sim\Omega} \mathop{\mathbb{E}}_{\tau\sim T(\mathbf{x},1)} [\mathcal{L}(N(\tau(\mathbf{x})), \mathbf{y})]. \tag{3.2}$$

where 1 indicates the cooperative mode of $T$. The discriminator $D$ is not used in the cooperative task because the transformed data of reduced variance can hardly fall out of the real data distribution. $T$ cooperates with $N$ in the cooperative task. The cooperative $T$ is trained on the training data and generalized to the testing data.

### 3.3.2 Discriminator D

The discriminator $D$ aims to control the variance of transformed data. It learns to assign low scores to out-of-distribution transformed data and high scores to in-distribution data. To this end, $D$ learns from both the original and transformed data as follows:

$$\max_{\theta_D} \mathop{\mathbb{E}}_{\mathbf{x}\sim\Omega} \mathop{\mathbb{E}}_{\tau\sim T(\mathbf{x})} [\log(1 - D(\tau(\mathbf{x})))] + \mathop{\mathbb{E}}_{\mathbf{x}'\sim\Omega} [\log(D(x'))]. \tag{3.3}$$

$D$ competes with the transformer $T$ in the competitive task. It is a critical design to automate competitive training. Human users can be saved from the heavy burden of specifying the transformation boundaries, especially when multi-types of transformations are available. Without $D$, $T$ would probably produce out-of-distribution transformations.

### 3.3.3 Target Network N

The goal of target network $N$ is to generalize well on the testing data. The training data usually have some distribution shift from the testing data. The current neural networks are so powerful that they can easily overfit the training data. The transformer $T$ can

reduce the overfitting by adaptively increasing the training data variance. $N$ learns from both the original and the transformed training data as follows:

$$\min_{\theta_N} \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\sim\Omega} \mathop{\mathbb{E}}_{\tau\sim T(\mathbf{x})} [\mathcal{L}(N(\tau(\mathbf{x})), \mathbf{y}) + \mathcal{L}(N(\mathbf{x}), \mathbf{y})], \tag{3.4}$$

The target network $N$ competes with the transformer $T$ through learning from its transformed data.

## 3.4 Learning Strategy

The triplet $T$, $D$, and $N$ are jointly learned in the adaptive data transformation. The main challenge comes from learning $T$ since many transformation operations are not differentiable. The gradients cannot flow to $T$ directly from $D$ and $N$. To deal with this issue, we use reinforcement learning with meta-transformations to train $T$.

### 3.4.1 Meta-transformation

The meta-transformations define the small transformation operations [99]. Table 3.1 lists examples of meta-transformations in natural images. A large transformation can be decomposed as a combination of multiple meta-transformations. Despite some precision loss, it barely affects the target network training. Specifying the meta-transformations requires much less domain knowledge than tuning the boundaries of multi-type transformations and choosing their combinations [62, 3].

The meta-transformation offers flexibility and scalability to achieve complex transformations. We can efficiently explore an ample transformation space by traversing the combinations of meta-transforms. More importantly, the meta-transformations make it possible to train $T$ in a tractable manner via reinforcement learning.

### 3.4.2 Reinforcement Learning Formulation

The transformer $T$ incrementally transforms the data using meta-transformations. An illustration is shown in Figure 3.2. Let $x$ and $\hat{x}$ denote the original and transformed data points. At step $t$, $T$ conditioning on $\hat{x}_{t-1}$ outputs the distribution $T(\hat{x}_{t-1})$ over all

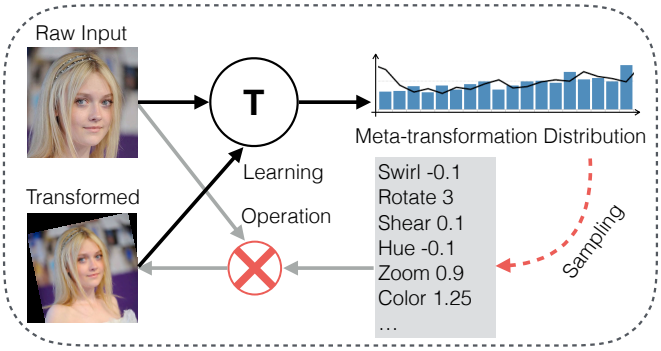



Figure 3.2: Incremental transformation. The transformer, conditioning on the input, outputs the distribution over the meta-transformations. A meta-transformation is sampled and transforms the input. Then the transformed data become the input and continue to be transformed.

the meta-transformations. Then the meta-transformation $\tau_t$ is sampled from it. The loss of transformed data $\hat{x}_t = \tau_t(\hat{x}_{t-1})$ is computed as:

$$\ell(\hat{x}_t) = \begin{cases} \mathcal{L}(N(\hat{x}_t), y) + \lambda \log(D(\hat{x}_t)), & \textit{competitive case.} \\ -\mathcal{L}(N(\hat{x}_t), y), & \textit{cooperative case.} \end{cases} \tag{3.5}$$

where $\mathcal{L}$ denotes the loss function for the target task and $\lambda$ is the weight of the discriminator loss. In the *competitive* mode, the transformer learns to expand the data variance by increasing the target network loss. On the other hand, it also tries to keep high probabilities of transformed data being realistic. In the *cooperative* mode, the transformer learns to reduce the data variance by increasing the negative target loss, i.e., decreasing the target loss.

The reward $r_t$ for meta-transformation $\tau_t$ is the incremental loss:

$$r_t = \ell(\hat{x}_t) - \ell(\hat{x}_{t-1}). \tag{3.6}$$

Suppose the transformer $T$ is applied $K$ steps, a reward sequence $\{r_1, r_2, \cdots, r_K\}$ is produced, where reward $r_1$ is $r_1 = \ell(\hat{x}_1) - \ell(x)$. Summing up these rewards results in

$$\sum_{t=1}^{K} r_t = \ell(\hat{x}_K) - \ell(x). \tag{3.7}$$

The discriminator and target network are fixed when training the transformer. Given an original data point $x$, $\ell(x)$ is a constant, which can be ignored. $\ell(\hat{x}_K)$ is the objective

**Algorithm 3:** Mini batch training of transformer $T$

**Input:** Mini-batch $B$, triplet $T$, $D$, and $N$.
**Output:** Transformer $T$

Replicate $B$ $s$ times to get $\mathbf{X}$ of size $M$;
Apply $T$ on $\mathbf{X}$ to get $\hat{X}$ and polices $\{\pi_t^i\} \in \mathbb{R}^{M \times K}$;
Compute rewards $\{r_t^i\} \in \mathbb{R}^{M \times K}$ of $\hat{X}$ by Eq. 3.5 and 3.6;
Get accumulated rewards $\{R_t^i\} \in \mathbb{R}^{M \times K}$ by Eq. 3.8;
Normalize $\{R_t^i\}$ to $\{\bar{R}_t^i\}$ by Eq. 3.9 and Eq. 3.10;
Call the gradient ascent on the sum of $\{\bar{R}_t^i \log \pi_t^i\}$;

in either Equations 3.1 or 3.2 since $\hat{x}_K$ is the final transformed data point. Therefore, optimizing the objectives in Equations 3.1 or 3.2 can be converted into maximizing the sum of rewards.

We apply the policy gradients to maximize the sum of rewards. Two common techniques are used to reduce the variance in estimating the rewards. First, we compute the reward for transformation $\tau_{t'}$ as the accumulated future reward $\sum_{t>=t'}^{K} r_t$ rather than $r_{t'}$ only. A discounting factor $\gamma$ is used to model the delaying effects of future rewards. Therefore, the accumulated discounted reward $R_{t'}$ is:

$$R_{t'} = \sum_{t>=t'}^{K} \gamma^{t-t'} r_t, \tag{3.8}$$

where we set $\gamma = 0.5$ in the experiments.

Also, the raw value of reward $R_{t'}$ may not be meaningful. Positive values do not necessarily mean rewards. We only push up the probability of a meta-transformation, if its reward is higher than the expectation. Here we use the mean of reward $R_{t'}$ within each mini-batch of $h$ training samples as the reference. For each original data point, we sample $s$ different rewards $R_{t'}$. Thus, the mean of $R_{t'}$ is:

$$b_{t'} = \frac{1}{h \times s} \sum_{i=1}^{h \times s} R_{t'}^i \tag{3.9}$$

Note that it is important to compute the reward mean online within the mini-batches instead of using the moving averages of all history rewards. Because the discriminator and target network become more and more powerful in training. The history rewards cannot reflect their current states well.

At step $t'$, each reward $R_{t'}$ is normalized by subtracting its mean $b_{t'}$. A positive

**Algorithm 4:** Joint training scheme of $T$, $D$, and $N$

**Input:** Training data $X$, triplet $T$, $D$, and $N$.
**Output:** Triplet $T$, $D$, and $N$.

```
while not end do
    for mini batch B in X do
        Apply T on B with probability p to get B̂;
        Train N with the mixed data B̂;
    end
    for mini batch B in X do
        Train competitive T with D, N by Alg. 3;
        Train cooperative T with N by Alg. 3;
    end
end
```

normalized value means reward, whereas a negative normalized one means penalty. According to the policy gradients formula, we compute the gradient of transformer $T$ at step $t'$:

$$\nabla_{\theta_T} T(\hat{x}_{t'}) = \sum_{i=1}^{h \times s} (R_{t'}^i - b_{t'}) \nabla_{\theta_T} \log \pi_{\theta_T}(\tau_{t'} | \hat{x}_{t'}), \tag{3.10}$$

where $\pi_{\theta_T}(\tau_{t'}|\hat{x}_{t'})$ is the policy, i.e., the probability of taking meta-transformation $\tau_{t'}$ given the input $\hat{x}_{t'}$. Updating transformer $T$ with the gradient ascent can push up or pull down the probabilities if the corresponding meta-transformations yield rewards or penalties at step $t'$.

Finally, we sum up the gradients of $T$ from all $K$ steps:

$$\nabla_{\theta_T} T(\cdot) = \sum_{t'=1}^{K} \nabla_{\theta_T} T(\hat{x}_{t'}). \tag{3.11}$$

Basically, the transformer $T$ is updated each time using the accumulated gradients from $K$ steps and $h \times s$ samples. Algorithm 3 summarizes the training scheme of $T$.

### 3.4.3 Joint learning of T, D, and N

The transformer $T$ is jointly optimized with the discriminator $D$ and target network $N$ during training. The training procedure is described in Algorithm 4. More specifically, we train $N$ for several epochs and then update $T$ and $D$ once. $N$ needs to learn from both the transformed and original data. To this end, we apply $T$ with some probability $p(0 < p < 1)$ on the training data of $N$.

Table 3.1: Examples of meta-transformations in natural images. A meta-transformation defines a small operation. A combination of multiple meta-transformations can approximate a large transformation space.

| Type | Meta-values |
|---|---|
| Rotation | 2.5°, −2.5°, 5°, −5° |
| Zoom | 0.9x, 1.1x, , 0.75x, 1.25x |
| Shear/Swirl | 0.1°, −0.1°, 0.25°, −0.25° |
| Hue Shift | 0.1, -0.1, 0.25, -0.25 |
| Brightness/Color | 0.75, 1.25, 0.5, 1.5 |
| Sharpness/Contrast | 0.75, 1.25, 0.5, 1.5 |
| Horizontal Flip | - |

$T$ and $D$ are updated alternately inside each iteration. Given a mini-batch of data, $D$ is updated on both the original (real) and transformed (fake) data of $T$. Then we update $T$ separately in the *competitive* and *cooperative* modes. $T$ receives the feedback from $N$ in the *cooperative* case while it requires the additional feedback from $D$ in the *competitive* case. We add a zero or one map to the input of $T$ as the condition of *competitive* or *cooperative* modes.

## 3.5 Applications of AdaTransform

AdaTransform provides a versatile solution for general data analytic tasks with proper domain knowledge. In this work, we focus on its application to visual tasks.

**AdaImgTransform.** For natural images, there are many available transformation types such as scale, rotation, translation, flipping, swirl, shear, contrast enhancement, color enhancement, brightness enhancement, sharpness enhancement, and hue shift. Table 3.1 lists the corresponding meta-transformations. We can adjust the meta-transformation pool according to domain knowledge of a specific task. We apply adaptive data transformation to learning to combine the proper meta-transformations conditioned on the input image, target network state, and the transformer mode. They can be used to either increase or reduce data variance.

**AdaCutout/AdaErasing.** Occlusions are quite common in natural images where the object of interest is partially occluded. The cutout [109] and random erasing [110]

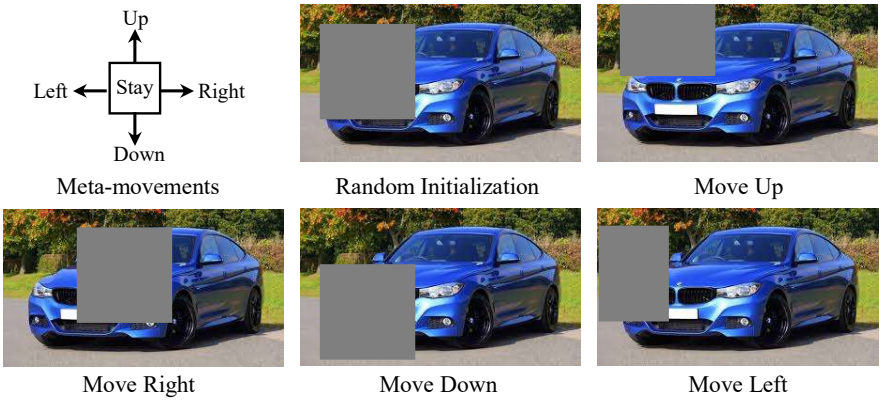



Figure 3.3: Meta-movements. AdaCutout/AdaErasing first samples a random mask, then moves it up, right, down, left.

are recently proposed to simulate the occlusions on the images. To be specific, a fixed-size square mask (cutout) or a flexible one (erasing) are used to occlude the image region centered at a randomly chosen position. We apply the adaptive transformation to control cutout or erasing. More specifically, we use random cutout or erasing for the initialization. Then the transformer learns to move the cutout mask progressively. Each step it can be moved up, right, down, left, or stay still. Figure 3.3 illustrates the five meta-movements.

## 3.6 Experiments

The experiments include three parts: ablation study, robustness test, and comparison with state-of-the-art methods. We evaluate AdaTransform on three different tasks: image classification, human pose estimation, and face alignment. We apply meta-transformations given in Table 3.1 for image classification. For the other two tasks, we remove shear and swirl due to the shifting of ground truth.

### 3.6.1 Experimental Settings

**Transformer $T$ and discriminator $D$.** The transformer and discriminator use the common networks. More specifically, the transformer has the architecture of ResNet-18 [10]. Additionally, we add the dropout layers after each $3 \times 3$ convolution layer and before the fully connected layer. The discriminator is the same as the one in DCGAN

[111].

**Target network $N$.** Different tasks have different target networks. In **image classificatin**, we use the 32-layer ResNet (ResNet32) [10] in the ablation study. The comparisons with state-of-the-art data augmentation method AutoAugment [2] are based on more complex models: Wide-ResNet-28-10 [112], Shake-Shake [113] and ShakeDrop [114]. For **human pose estimation** and **face alignment**, we use the two stacked hourglasses [1] in all the experiments.

**Hyperparameters.** We use two transformers for adaptive cutout (AdaCuout) and adaptive image transformation (AdaImgTransform). The AdaCutout transformer is trained with learning rate 3e-5 and weight decay 1e-5 whereas the AdaImgTransform transformer has learning rate 1e-4 and weight decay 1e-4. AdaCutout moves the occlusion mask 2 pixels each step. We set step number $K = 3$ for AdaCutout and $K = 8$ for the AdaImgTransform. Besides, AdaCutout is applied with probability 0.3 on each mini-batch when training the target network. On the other hand, we use AdaImgTransform on all the training data but stop it for the last ten epochs.

**Datasets.** We use the benchmark datasets: CIFAR-10 and CIFAR-100 for image classification; MPII Human Pose [91] and Leeds Sports Pose (LSP) [92] for human pose estimation; 300-W challenge [115] for face alignment. The 300-W test set consists of easy and challenging subsets. We use the classification accuracy/error, Percentage of Correct Key points (PCK), and normalized mean error (NME) as the measurements of image classification, human pose estimation, and face alignment. In particular, MPII and LSP use PCKh@0.5 and PCK@0.2.

### 3.6.2 Ablation Study

**Effect of transformation steps.** The transformer incrementally transforms an image for several steps. It is interesting to observe how test accuracy changes with the step number. We train 6 models for each step number using 10% CIFAR-10 training data. Figure 3.4 shows the *mean* and *std* of the testing accuracy. A modest increase of step number can produce more complex transformations, increasing testing accuracy. However, more transformation steps are difficult to learn and result in high model

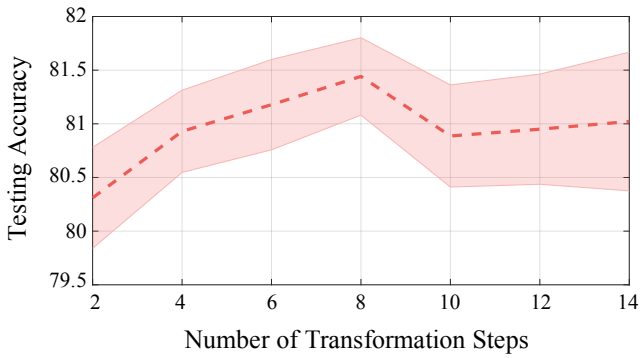



Figure 3.4: Effect of transformation steps. The dotted line and shaded areas represent *mean* and *std*, respectively. More steps would increase *std* (high model variance). The best trade-off between testing accuracy and model variance is obtained at 8-step.

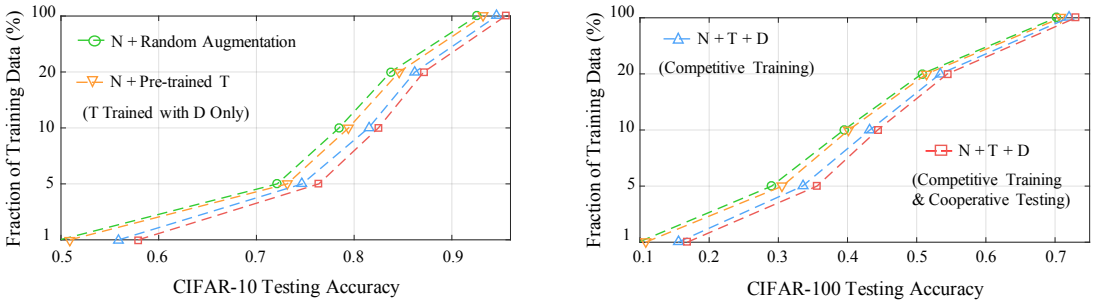



Figure 3.5: Validation of competitive and cooperative tasks. We show the testing accuracy with respect to the fraction of training data. The joint learning of competitive training and cooperative testing achieves the best performance (**lowest**). Its superior performance is more significant when less data is used in training (**right to left**).

variance.

**Validation of competitive and cooperative tasks.** We incrementally add each component and observe the changes in test accuracy. Figure 3.5 gives the comparison of four variants. They all use eight transformation steps and the same meta-transformation pool in Table 3.1. The competitive training and cooperative testing can both increase test accuracy with different percentages of training data. In the case of only 1% training data, the competitive training can improve ∼5% accuracy on both CIFAR-10 and CIFAR-100 over the pre-trained transformer, indicating the importance of joint training with the target network. The cooperative testing, on the other hand, further brings ∼2% gain for the two datasets. Even with 100% training data, competitive training and cooperative testing can separately get ∼1% improvements on both

Table 3.2: Evaluation of AdaCutout and AdaErasing using 10% training data of CIFAR-10 and CIFAR-100.

| Method | CIFAR-10 | CIFAR-100 |
|---|---|---|
| Cutout [109] | 77.21 | 40.41 |
| AdaCutout | **78.02** | **41.02** |
| Erasing [110] | 77.25 | 40.53 |
| AdaErasing | **78.12** | **41.21** |

Table 3.3: Effect of different types of adaptive transformation in human pose estimation. We report per-joint PCKh (%). A single kind of adaptive transformation would improve the performance compared with randomly performed. Jointly applying all transformations has the best performance.

| Method | Head | Sho. | Elb. | Wri. | Hip | Knee | Ank. | Mean |
|---|---|---|---|---|---|---|---|---|
| RandomAugment. | 95.7 | 95.0 | 89.1 | 83.4 | 88.2 | 84.0 | 80.2 | 88.1 |
| AdaImgTexture | 95.3 | 95.3 | 89.7 | 84.8 | 89.0 | 84.9 | 80.9 | 88.7 |
| AdaCutout | 95.5 | 95.2 | 89.7 | 84.6 | 88.5 | 84.7 | 80.9 | 88.6 |
| AdaScaleRotation | 95.5 | 95.6 | 89.8 | 85.0 | 89.4 | 84.7 | 80.8 | 88.9 |
| AdaAll | **95.8** | **96.0** | **90.1** | **85.4** | **89.8** | **85.7** | **81.3** | **89.3** |

datasets.

**Evaluation of AdaCutout and AdaErasing.** Apart from the above AdaImgTransform, we also evaluate AdaCutout and AdaErasing. The results are given in Table 3.2. Cutout and random erasing obtain similar accuracy. AdaCutout and AdaErasing can both improve the baselines.

**Effect of different types of adaptive transformation.** We categorize the transformations into three groups: spatial variations (scale and rotation), occlusion (Cutout [109]), and texture changes (image color, brightness, contrast, sharpness, and hue). It may be interesting to study their separate contributions. AdaTransform can utilize them both independently and jointly. Table 3.3 gives the results on human pose estimation. The spatial transformations bring more improvement (0.8%) than the other two (0.5% and 0.6%), indicating its importance in human pose estimation.

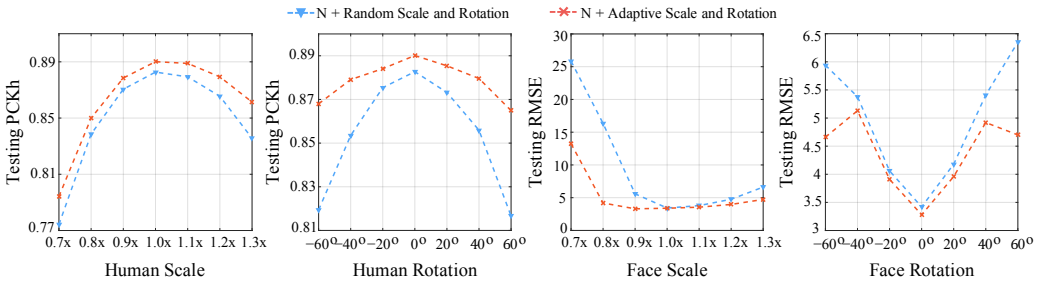



Figure 3.6: Robustness against rotations and scale perturbations. We investigate human pose estimation (**left two, the higher the better**) and face alignment (**right two, the lower the better**). Network ($N$) trained using adaptive data transformation outperform random ones with substantial margins. The performance improvements are more significant when increasing perturbations, indicating the effectiveness in learning more robust models.

Table 3.4: Robustness against texture (color, brightness, contrast, sharpness, and hue) perturbations. We investigate standard (**top two rows**) and perturbed (**bottom two rows**) testing. In particular, AdaImgTexture is more robust against texture perturbations.

| Method | Head | Sho. | Elb. | Wri. | Hip | Knee | Ank. | Mean |
|---|---|---|---|---|---|---|---|---|
| RandomAugment. | **95.7** | 95.0 | 89.1 | 83.4 | 88.2 | 84.0 | 80.2 | 88.1 |
| AdaImgTexture | 95.3 | **95.3** | **89.7** | **84.8** | **89.0** | **84.9** | **80.9** | **88.7** |
| RandomAugment. | 94.4 | 93.9 | 86.9 | 81.5 | 86.7 | 82.0 | 77.0 | 86.3 |
| AdaImgTexture | **94.9** | **94.9** | **88.5** | **83.2** | **88.2** | **83.6** | **79.7** | **87.8** |

### 3.6.3 Robustness Test

In the traditional test, testing images are usually static with no perturbations. However, in practice, an image may be affected by many factors, such as scale and rotation. A robust model should handle well not only the original image but also its variants under reasonable perturbations. In this experiment, we test models under the condition of different scales, rotations, and texture variations of testing data. To evaluate the robustness of AdaTransform, we compare the models trained with it and random augmentation.

**Robustness against scale and rotation perturbations.** Figure 3.6 shows the robustness comparisons in two tasks. AdaTransform can consistently improve testing performance over a range of scales and rotations, especially at the ends. In human pose estimation, we observe ∼3% accuracy increase for scales 0.7/1.3 and ∼5% increase for

Table 3.5: Comparison with AutoAugment [2] in terms of image classification errors. AdaTransform has comparable performance with all the three classifiers. However, it is much more efficient than AutoAugment.

| Model | CIFAR-10 | | CIFAR-100 | |
|---|---|---|---|---|
| | AutoAug. | Ours | AutoAug. | Ours |
| Wide-ResNet [112] | **2.68** | 2.95 | **17.09** | 17.42 |
| Shake-Shake[113] | **1.99** | 2.11 | **14.28** | 15.01 |
| ShakeDrop[114] | **1.48** | 1.72 | **10.67** | 11.21 |

rotations $-60°/60°$. For face alignment, the large error drops $\sim$12% and $\sim$2% happen at scale 0.7 and rotations $-60°/60°$.

**Robustness against texture perturbations.** To get reasonable texture perturbations, we train a transformer with only one discriminator using CIFAR-10. During testing, we use 15 trained transformer models to perturb the testing images. Table 3.4 gives the robustness comparisons with random augmentation. AdaTransform can get higher PCKh on both the standard test and test with texture perturbations. Moreover, the PCKh gap 1.5% in the perturbed test is much larger than the 0.6% in the standard test.

### 3.6.4 Comparison with State-of-the-art Methods

**Image classification.** We first compare AdaTransform (AdaImgTransform + AdaCutout) with state-of-the-art AutoAugment [2]. Table 3.5 shows the comparisons on both CIFAR-10 and CIFAR-100. AdaTransform obtains comparable performance as the AutoAugment. However, it needs to train only three models. In contrast, the AutoAugment requires to train fifteen thousand models to search the final augmentation policy. Although each model in AdaTransform may take longer to train, it is still much more efficient.

Note that the AutoAugment cannot work if only training several models. It is a purely reinforcement-based method, optimizing the validation error. The trained model number represents its search space. AdaTransform, on the other hand, integrates the adversarial training with reinforcement learning, optimizing the training loss.

Table 3.6: Comparison with adversarial data augmentation [3] in human pose estimation. We use two stacked hourglasses and report PCKh@0.5 on MPII validation set (**top**) and PCK@0.2 on LSP test set (**bottom**).

| Method | Head | Sho. | Elb. | Wri. | Hip | Knee | Ank. | Mean |
|---|---|---|---|---|---|---|---|---|
| AdvAug. [3] | **96.5** | 95.5 | 89.8 | 84.5 | 89.4 | 85.0 | 80.7 | 88.9 |
| AdaTransform | 95.8 | **96.0** | **90.1** | **85.4** | **89.8** | **85.7** | **81.3** | **89.3** |
| AdvAug. [3] | 96.8 | 93.7 | 90.9 | **88.0** | 92.0 | 93.7 | 92.4 | 92.5 |
| AdaTransform | **96.9** | **94.1** | **91.0** | 87.8 | **93.0** | **94.5** | **93.3** | **92.9** |

Table 3.7: Comparison with adversarial data augmentation [3] in face alignment (NME) on 300-W dataset.

| Method | Easy Subset | Hard Subset | Full Set |
|---|---|---|---|
| AdvAug. [3] | 2.87 | 4.98 | 3.28 |
| AdaTransform | **2.82** | **4.96** | **3.24** |

**Human pose estimation.** We also compare AdaTransform with state-of-the-art adversarial data augmentation [3] on human pose estimation. Table 3.6 gives the comparisons based on two stacked hourglasses [1]. AdaTransform obtains %0.4 mean improvements on both datasets. AdaTrasnform can search a larger transformation space by composing multi-type meta-transformations. In addition, Figures 3.7 and 3.8 show some examples of cooperative zoom-out and zoom-in.

**Face alignment.** AdaTransform and state-of-the-art adversarial data augmentation [3] can both apply to face alignment. We use two stacked hourglasses as the target network. The results are shown in Table 3.7. AdaTransform obtains %0.05 and %0.02 lower errors on the easy and challenging subsets, respectively.

## 3.7 Summary

We have proposed AdaTransform to manipulate data variance in bi-direction at the training and testing stages. AdaTransform can be learned efficiently by jointly optimizing a triplet online. Experimental results on image classification, human pose estimation, and face alignment demonstrate its superior performance in network training and testing especially when perturbations exist.

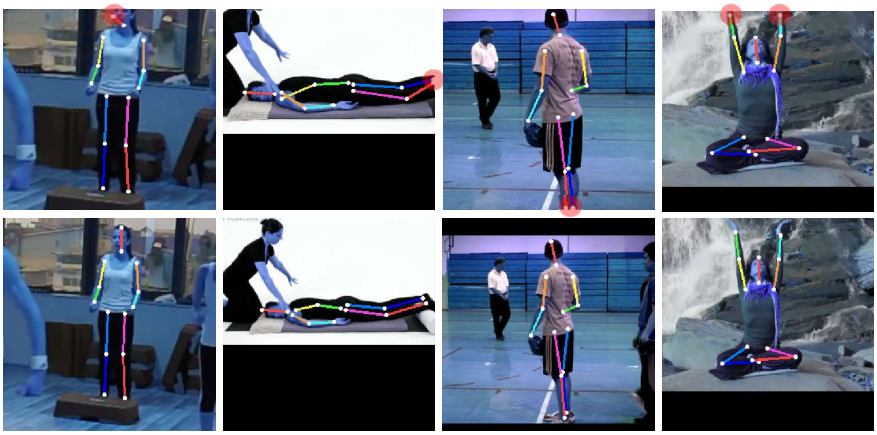


Figure 3.7: Cooperative zoom-out in human pose estimation. False positives, marked by red circles, are detected on the original scales (**Top**). Some human joints such as head, wrist, and ankle, may be too close to the image boundary or even fall out of scope in the original scale. Zoom-out (**Bottom**) can bring them into scope in this case.

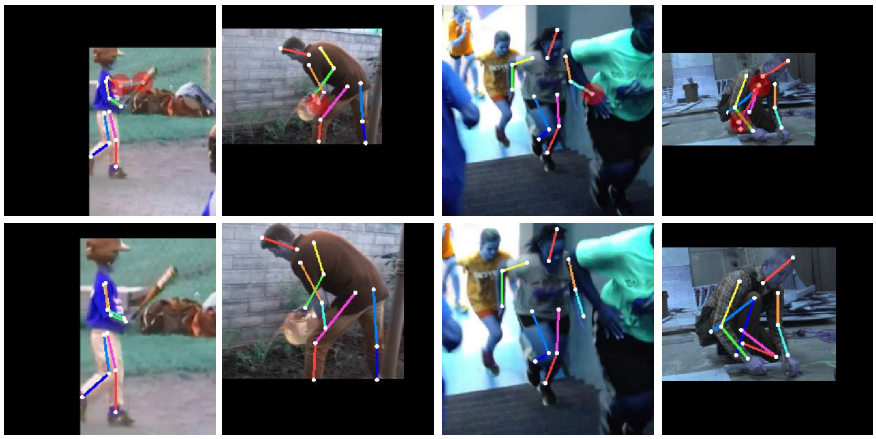


Figure 3.8: Cooperative zoom-in in human pose estimation. The false positives on the original scales (**Top**) are marked by red circles. Human pose estimation on a small scale is usually sensitive to background noises such as overlapped objects or people. Enlarging small scale can help alleviate the ambiguity caused by them (**Bottom**).

# Chapter 4

# OnlineAugment: Online Data Augmentation with Less Domain Knowledge

## 4.1 Introduction

Data augmentation is widely used in training deep neural networks. It is an essential ingredient of many state-of-the-art deep learning systems on image classification [116, 117, 118, 119], object detection [120, 121], segmentation [122, 40, 123], as well as text classification [124]. Current deep neural networks may have billions of parameters, tending to overfit the limited training data. Data augmentation aims to increase both the quantity and diversity of training data, thus alleviates overfitting and improves generalization.

Traditionally, data augmentation relies on hand-crafted policies. Designing the polices is usually inspired by domain knowledge and further verified by testing performance [45, 10]. For example, the typical routine in training CIFAR classifiers uses random cropping and horizontal flip to conduct data augmentation. Intuitively, these operations do not change the image labels, and they can also improve testing performance in practice.

Recently, AutoML techniques [125, 126] are used to automate the process of discovering augmentation polices. The resulted approaches, such as AutoAugment [9] and its variants [117, 127, 8, 7] are quite successful and achieve state-of-the-art results. We name them *offline data augmentation* since the policy learning and usage are isolated. Moreover, these approaches use pre-specified image processing functions as augmentation operations. Defining the basic operations requires domain knowledge, which may impede their applications to more tasks.

In this work, we propose *OnlineAugment*, which jointly optimizes data augmentation and target network training in an online manner. The merits of OnlineAugment are three-fold. First, it is orthogonal to the offline methods. Their complementary nature makes it possible to apply them together. Second, through the online learning, the augmentation network can adapt to the target network through training from the start to the end, saving it from the inconveniences of pre-training [128] or early stopping [99]. Third, it is easy to implement and train OnlineAugment. In contrast, learning offline policies usually rely on distributed training, as there are many parallel optimization processes.

Furthermore, we propose more general data augmentation operations with less domain knowledge. Instead of using pre-defined image processing functions, such as rotation and color, we design neural networks to perform data augmentation. Specifically, we devise three learnable models: augmentation STN (A-STN), deformation VAE (D-VAE), and Perturbation VAE (P-VAE). It is nontrivial to craft STN [46] and VAE [47] to conduct data augmentation. We also propose new losses to regularize them in training. Besides, OnlineAugment integrates both adversarial training and meta-learning in updating the augmentation networks. Adversarial training is to prevent overfitting, whereas meta-learning encourages generalization.

In summary, our key contributions are:

- We propose a new online data augmentation scheme based on meta-learned augmentation networks co-trained with the target task. Our framework is complementary to the state-of-the-art offline methods such as AutoAugment. Experiments on CIFAR, SVHN, and ImageNet show that on its own, OnlineAugment achieves comparable performances to AutoAugment. More excitingly, OnlineAugment can further boost state-of-the-art performances if used jointly with AutoAgument policies.

- We propose three complementary augmentation models responsible for different types of augmentations. They replace the image processing functions commonly used in contemporary approaches and make our method both more adaptive and

less dependent on domain knowledge.

- We show that the proposed OnlineAugment can generalize to tasks different from object classification by applying OnlineAugment to a liver&tumor segmentation task, demonstrating improved performance compared with the state-of-the-art RandAugment on this task.

## 4.2 Related Work

Data augmentation has been shown to improve the generalization of machine learning models and is especially effective in training deep neural networks. It is essential in the situation where only limited data is available for the target task, but is also crucial for generalization performance in case the data is abundant.

Known class-preserving transformation has been routinely used to expand labeled datasets for training image classifiers. These include operations such as cropping, horizontal and vertical flips, and rotation [129, 116, 117]. Recently, reinforcement learning has been used to learn the optimal sequence of such transformations for a given dataset that leads to improved generalization [99]. AutoAugment [9] falls under this category of methods and actively explores policies and validates their effectiveness by repeatedly training models from scratch. Due to the large search space, the searching process, based on reinforcement learning, severely suffers from high computational demand. Subsequent works [130, 7] in this direction have been aimed at reducing the computational complexity of the search strategies. However, they all follow the formulation of AutoAugment that first searches policies using a sampled small dataset, and then applies them to the final large dataset. Thus the policy learning and usage are isolated. Adversarial AutoAugment [131] jontly optimizes the polices and target learner. However, the learned policies are still based on domain-specific image processing functions.

More general transformations, such as Gaussian noise and dropout, are also effective in expanding the training set [132, 133, 109]. Spatial Transformer Network (STN) can perform more adaptive transformations than image processing functions such as rotation. However, it was designed for localization, not for data augmentation. In this work,

we craft STN to conduct data augmentation. Generative models are also helpful for data augmentation. DAGAN [101] employs a Generative Adversarial Network (GAN) [134] to learn data augmentation for few-shot classification. In this work, we devise two augmentation models based on Variational Auto-encoder (VAE) [47], as another popular generative model.

Adversarial training [43, 135, 5] can serve as a general data augmentation framework. It aims to generate adversarial perturbations on input data. The adversarial examples are further used in training models to improve their robustness. It has been shown that adversarial training can hurt model generalization although it can boost robustness [5, 136]. Concurrent work AdvProp [4] successfully adapts adversarial training to advance model generalization. It uses the common adversarial attack methods, such as PGD and I-FGSM, to generate additive noises. In contrast, our models can learn to generate more diverse augmentations: spatial transformation, deformation, and additive noises. We also use adversarial training together with meta-learning.

Learning data augmentation is to train the target learner better, i.e., learning to learn better. Validation data have been used in meta-learning literatures for few-shot learning [137, 138, 139], where very limited training data are available. Here we follow the MAML [140] algorithm to set a meta-objective for the augmentation network. That is, augmentations conducted on a training mini-batch is evaluated on another validation one.

## 4.3 The Online Data Augmentation Formulation

In this section, we introduce our online data augmentation paradigm: updating target model $\theta$ and augmentation model $\phi$ alternately. In this way, data augmentation and target model are learned jointly. Benefiting from the joint learning, the augmentation model $\phi$ can adapt to the target model $\theta$ in training.

For simplicity, let $\mathbf{x}$ be the annotated data, including both input and target. Note that $\mathbf{x}$ can come from any supervised task such as image classification or semantic

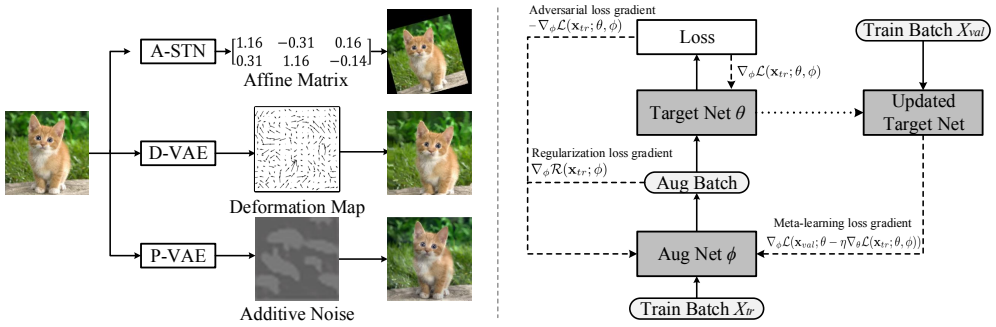



Figure 4.1: Augmentation illustrations (**Left**) and OnlineAugment scheme (**Right**). We propose three models to conduct spatial transformation, deformation, and noise perturbation. OnlineAugment can jointly optimize each plug-in augmentation network with the target network. Updating the augmentation network incorporates adversarial training, meta-learning, and some novel regularizations.

segmentation. Let $\theta$ and $\phi$ denote the target and augmentation models. During training, the target model $\theta$ learns from the augmented data $\phi(\mathbf{x})$ instead of the original $x$. Note that $\phi$ will also transform the ground truth annotation if necessary. For example, in semantic segmentation, $\phi$ applies the same spatial transformations to both an image and its segmentation mask. Without loss of generality, we assume $\theta$ and $\phi$ are parameterized by deep neural networks, which are mostly optimized by SGD and its variants. Given a training mini-batch $\mathbf{x}_{tr}$ sampled from training set $\mathcal{D}_{tr}$, $\theta$ is updated by stochastic gradient:

$$\nabla_\theta \mathcal{L}(\mathbf{x}_{tr};\theta,\phi), \tag{4.1}$$

where the choice of $\mathcal{L}$ depends on the task. In the case of object classification, $\mathcal{L}$ is a cross entropy function.

The goal of data augmentation is to improve the generalization of the target model. To this end, we draw on inspirations from adversarial training and meta-learning. Adversarial training aims to increase the training loss of the target model by generating hard augmentations. It can effectively address the overfitting issue of the target model. Meta-learning, on the other hand, can measure the impact of augmented data on the performance of validation data. If a validation set $\mathcal{D}_{val}$ is possible, we can sample from it a validation mini-batch $\mathbf{x}_{val}$. Otherwise, we can simulate the meta-tasks by sampling two separate mini-batches from train set $\mathcal{D}_{tr}$ as $\mathbf{x}_{tr}$ and $\mathbf{x}_{val}$. Mathematically, the

**Algorithm 5:** OnlineAugment: Online Data Augmentation

**Input:** Initial target model $\theta$, initial augmentation model $\phi$, training set $\mathcal{D}_{tr}$, and validation set $\mathcal{D}_{val}$

**while** *not converged* **do**
  Sample mini-batches $\mathbf{x}_{tr}$ and $\mathbf{x}_{val}$ from $\mathcal{D}_{tr}$ and $\mathcal{D}_{val}$ respectively
  Update augmentation model $\phi$ by stochastic gradient: $\nabla_\phi \mathcal{L}(\mathbf{x}_{val}; \theta - \eta \nabla_\theta \mathcal{L}(\mathbf{x}_{tr}; \theta, \phi)) + \lambda \nabla_\phi \mathcal{R}(\mathbf{x}_{tr}; \phi) - \beta \nabla_\phi \mathcal{L}(\mathbf{x}_{tr}; \theta, \phi)$
  Update target model $\theta$ by stochastic gradient: $\nabla_\theta \mathcal{L}(\mathbf{x}_{tr}; \theta, \phi)$
**end**

**Output:** Optimized target model $\theta^*$

stochastic gradient of augmentation model $\phi$ is computed as:

$$\nabla_\phi \mathcal{L}(\mathbf{x}_{val}; \theta - \eta \nabla_\theta \mathcal{L}(\mathbf{x}_{tr}; \theta, \phi)) + \lambda \nabla_\phi \mathcal{R}(\mathbf{x}_{tr}; \phi) - \beta \nabla_\phi \mathcal{L}(\mathbf{x}_{tr}; \theta, \phi), \tag{4.2}$$

where $\mathcal{L}(\mathbf{x}_{val}; \theta - \eta \nabla_\theta \mathcal{L}(\mathbf{x}_{tr}; \theta, \phi))$, $\mathcal{R}(\mathbf{x}_{tr}; \phi)$, and $-\mathcal{L}(\mathbf{x}_{tr}; \theta, \phi)$ are the generalization, regularization, and adversarial losses. $\lambda$ and $\beta$ are the balancing weights. $\theta - \eta \nabla_\theta \mathcal{L}(\mathbf{x}_{tr}; \theta, \phi)$ represents the the updated target network by augmented data $\phi(\mathbf{x}_{tr})$. For simplicity, here we use a vanilla gradient descent with learning rate $\eta$. Other more complex optimizers are also applicable. For efficient training, we use the second-order approximation [141] to compute the meta-gradient.

$\mathcal{R}(\mathbf{x}; \phi)$ measures the distance between the original and augmented data. Adding this regularization term aims to constrain the augmented data within reasonable distributions. Otherwise, adversarial training may cause meaningless augmentations that hurt training. Theoretically, the generalization term can also help regularize the augmentations implicitly. In practice, we find that the explicit regularization term is critical for practical adversarial training. Besides, adversarial training is performed by minimizing the negative training loss $-\mathcal{L}(\mathbf{x}_{tr}; \theta, \phi)$. In this way, the augmentation model $\phi$ learns to generate hard augmentations. The training scheme is presented in Algorithm 5 and the right figure in Fig. 4.1.

**Relation to offline augmentation methods.** The formulation of our online augmentation differs from the previous offline ones [9, 130, 7] mainly in three respects. First, OnlineAugment alternates updating the target model $\theta$ and augmentation model $\phi$. The offline methods usually perform a full optimization for $\theta$ in each step of updating $\phi$. Second, to get the optimized target model, the offline methods usually require a

two-stage training: learning policies and applying them. However, OnlineAugment can optimize the target model in one training process. Third, we use adversarial training in learning the augmentation model. The offline methods only have one generalization objective, maximizing performance on validation data.

**Relation to adversarial training methods.** Adversarial training [43, 135, 5] is mainly used to improve the robustness of the target model. Some works [5, 136] have shown that robust models usually come at the cost of degraded generalization to clean data. The goal of OnlineAugment is generalization rather than robustness. To pilot adversarial training for generalization, we design new regularization terms and add meta-learning in OnlineAugment.

## 4.4 Data Augmentation Models

After introducing the OnlineAugment scheme, we present three different data augmentation models in this section. The three models are motivated by our analysis of possible data transformations. Specifically, we summarize them into three types: global spatial transformation, local deformation, and intensity perturbation. Each transformation corresponds to a model below. They either change pixel locations or values. Note that the pixel in this work refers to an element in a generic feature map, not necessarily an image. Technically, we design the augmentation models based on the spatial transformer network (STN) [46] and variational auto-encoder (VAE) [47].

### 4.4.1 Global Spatial Transformation Model

There are several commonly used spatial transformation functions such as rotation, scale, and translation, which can be unified into more general ones, such as affine transformation. It is well-known that STN [46] can perform general spatial transformations. Briefly, STN contains three parts: a localization network, a grid generator, and a sampler. The last two modules are deterministic and differentiable functions, denoted as $\tau$. Therefore, our focus is to design a new localization network $\phi$ that outputs the transformation matrix.

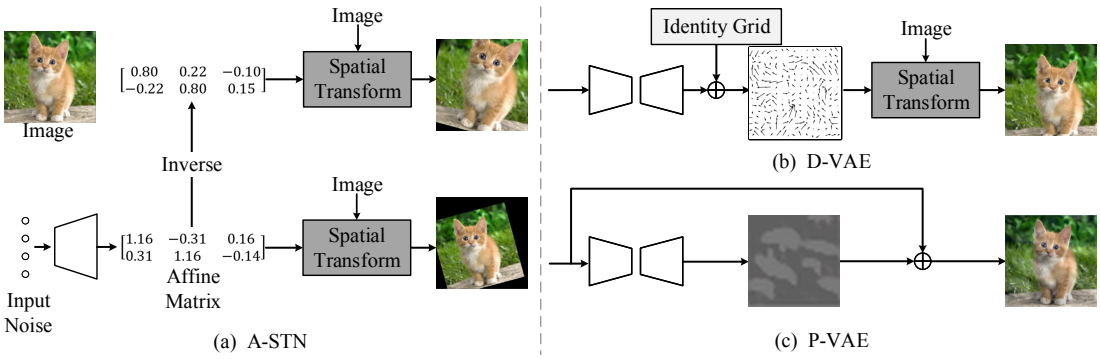



Figure 4.2: Augmentation models: A-STN **(a)**, D-VAE **(b)**, and P-VAE **(c)**. A-STN, conditioned on Gaussian noise, generate a transformation matrix. Both the matrix and its inverse are applied to an image for diversity. D-VAE or P-VAE takes an image as input, generating deformation grid maps or additive noise maps. The three models are trainable if plugged in the training scheme in Figure 4.1.

**Augmentation STN (A-STN).** Suppose we use affine transformation, the output should be a 6-dimension vector, further re-shaped into a 2×3 transformation matrix. Traditionally, the STN localization network uses images or feature maps as its input. Here we also provide an alternative input of Gaussian noises. Our design is motivated by the observation that the global spatial transformations are transferable in data augmentation. That is, the same spatial transformation is applicable to different images or feature maps in augmenting training data. Therefore, conditioning on the images or feature maps may not be necessary for generating the augmentation transformation.

The architecture of the localization network $\phi$ depends on the choices of its input. It can be a convolutional neural network (CNN) or a multi-layer perceptron (MLP) if conditioned on the generic feature map or 1-D Gaussian noise. We will give its detailed architectures in the experiment. Moreover, the MLP localization network itself is also transferable as it is unrelated to the target task. However, we may need to craft new CNN localization networks for different tasks. Therefore, it is preferable to implement the localization network $\phi$ as an MLP in practice.

**Double Cycle-consistency Regularization.** To apply the spatial transformation model to Algorithm 5, we need to design a proper regularization term $\mathcal{R}$. Empirically, increasing the spatial variance of training data can enhance the generalization power of the model. However, excessive spatial transformations probably bring negative impacts.

To constrain the spatial transformations within a reasonable scope, we propose a novel double cycle-consistency regularization, seen in Fig. 4.2 (a). The key idea is to measure the lost information during the spatial transformation process. Mathematically, we compute the double cycle-consistency loss:

$$\mathcal{R}_c^s(\mathbf{x};\phi) = \|\tau_\phi^{-1}(\tau_\phi(\mathbf{x})) - \mathbf{x}\|_2^2 + \|\tau_\phi(\tau_\phi^{-1}(\mathbf{x})) - \mathbf{x}\|_2^2, \tag{4.3}$$

where $\tau_\phi(\mathbf{x}) = \tau(\mathbf{x}, \phi(\mathbf{z}))$. The deterministic function $\tau$ transforms the image or feature map $\mathbf{x}$ using the generated affine matrix $\phi(\mathbf{z})$, where $\mathbf{z}$ is the Gaussian noise. $\tau_\phi^{-1}$ denotes the inverse transformation of $\tau_\phi$. Ideally, applying a transformation followed by its inverse will recover the original input, and vice versa. In reality, whichever is applied first may cause some irreversible information loss. For example, the zoom-in transformation discards the region falling out of scope. Applying the zoom-out transformation afterwards will produce zero or boundary padding, which is different from the original image region. We find a single cycle-consistency loss will lead to biased transformations. The localization network $\phi$ tends to output zoom-out transformations whose inverse can easily recover the original input. Fortunately, imposing the double cycle-consistency constraint can avoid the biases effectively, thereby producing more diverse transformations.

### 4.4.2 Local Deformation Model

Apart from the global spatial transformation model, we propose another complementary deformation model. The global transformation applies the same transformation matrix to all the pixels of an image or feature map. In the local deformation, each pixel, however, has an independent transformation.

**Input and Output.** It is cumbersome and also unnecessary to produce all the transformation matrices. Recall that STN performs transformations by the grid sampling. A better choice is to predict a grid map directly. For 2D transformations, the grid map has the shape $h \times w \times 2$, where $h$ and $w$ are the height and width of the input feature map. Each location in the grid map indicates the 2D coordinates to sample a pixel. A grid map is personalized to an image or feature map as each pixel has its own

transformation. Different from a low-dimension affine matrix, a deformation grid map may be unlikely to transfer. Therefore, our deformation model is conditioned on the image or feature map, generating the grid map.

**Deformation VAE (D-VAE).** The deformation model $\phi$, see in Fig. 4.2 (b), builds on the Variational Autoencoders (VAE) [47], a popular generative model. A VAE model consists of an encoder and a decoder. Similar to the original VAE, our deformation VAE also uses images or feature maps as the encoder input. However, in our setting, the decoder outputs the deformation grid maps instead of the reconstructed input. We refer to the deformation grid maps as deformation deltas $\Delta_\phi^d$. They are added on the grid maps of identity mapping $id$ to perform grid sampling $\tau$. The transformed input $\tau(\mathbf{x}, \Delta_\phi^d + id)$ serves as the reconstructed input. Following the original VAE, our deformation VAE is also trained to minimize both the reconstruction loss and the KL-divergence between the encoded distribution and the standard normal distribution:

$$\mathcal{R}_v^d(\mathbf{x};\phi) = \|\mathbf{x} - \tau(\mathbf{x}, \Delta_\phi^d + id)\|_2^2 + \mathcal{KL}(\mathcal{N}(\mu_\phi(\mathbf{x}), \Sigma_\phi(\mathbf{x})), \mathcal{N}(0, I)), \tag{4.4}$$

where $\mu_\phi(\mathbf{x})$ and $\Sigma_\phi(\mathbf{x})$ are the encoded mean and variance, parameterizing the Gaussian distribution.

**Smoothness Regularization.** Smooth deformations are essential to preserving the quality of deformed data. Otherwise, the deformed data may become noisy as each pixel is sampled from an independent location in the original image or feature map. This is especially important for location-sensitive tasks such as semantic segmentation. Given an arbitrary pixel $i$ and its neighbours $j \in \mathcal{N}(i)$, we enforce the local smoothness constraint on the deformation deltas:

$$\mathcal{R}_s^d(\mathbf{x};\phi) = \sum_i \sum_{j \in \mathcal{N}(i)} \|\Delta_\phi^d(i) - \Delta_\phi^d(j)\|_2^2. \tag{4.5}$$

The smoothness regularization can make the deformations consistent for nearby pixels. In the experiment, we use the combination $\lambda_v^d \mathcal{R}_v^d(\mathbf{x};\phi) + \lambda_s^d \mathcal{R}_s^d(\mathbf{x};\phi)$ to regularize our deformation augmentation model.

### 4.4.3 Intensity Perturbation Model

The above two models perform data augmentation by changing the pixel locations. Here we propose another model to manipulate the pixel values instead. As an analogy, the offline data augmentation methods [9, 130, 7] use some built-in image texture processing functions such as colour and brightness. These functions are designed based on the domain knowledge for natural images. In contrast, our intensity perturbation is more general without domain knowledge.

**Perturbation VAE (P-VAE).** Specifically, the intensity perturbation model $\phi$, shown in Fig. 4.2 (c), conditioned on the image or feature map, generates additive noises $\Delta_\phi^p$. As the deformation, we use the VAE model to learn the intensity perturbation. The reconstructed input is the sum of the input and generated noises $\mathbf{x}+\Delta_\phi^p$. Therefore, we can compute the VAE loss as:

$$\mathcal{R}_v^p(\mathbf{x};\phi) = \|\Delta_\phi^p\|_2^2 + \mathcal{KL}(\mathcal{N}(\mu_\phi(\mathbf{x}),\Sigma_\phi(\mathbf{x})),\mathcal{N}(0,I)), \tag{4.6}$$

where $\mu_\phi(\mathbf{x})$ and $\Sigma_\phi(\mathbf{x})$ are the mean and variance of the encoded Gaussian distribution. Note that P-VAE produces deltas $\Delta_\phi^p$ in the image or feature map domain while the deltas $\Delta_\phi^d$, predicted by D-VAE, lie in the grid map domain, which results in the different reconstruction losses in Equations 4.6 and 4.4.

**Relation to Adversarial Attacks.** Additive noise is a common tool in generating adversarial examples [4, 5]. Here we explore its potential in data augmentation. Although the adversarial examples serve as augmented data in adversarial training, they mainly serve to improve the model's robustness. Some evidence [142] has shown that adversarial training usually sacrifices the model generalization to clean data. However, our intensity perturbation model can improve generalization through OnlineAugment. Recently, concurrent work AdvProp [4] has successfully adapted the PGD attack [5] to data augmentation. In contrast, we design the perturbation VAE model to generate additive noises.

## 4.5 Experiments

In this section, we empirically evaluate the OnlineAugment scheme. More specifically, we plug in it the three augmentation models: Augmentation STN (A-STN), Deformation VAE (D-VAE), and Perturbation VAE (P-VAE). In the ablation study, we evaluation the three models both separately and jointly. We also provide detailed comparisons of OnlineAugment with offline AutoAugment [9] and their combined results. The comparisons to other state-of-the-art offline methods are reported on image classification and medical segmentation.

### 4.5.1 Experimental Settings

Applying OnlineAugment is simple in practice. The augmentation models A-STN, D-VAE, and P-VAE requires neither pre-training [128] nor early stopping [99] because they can adapt to the target network during online training. Inspired by AdvProp [4], we use multiple batch normalization layers to merge different types of augmentations. A-STN, D-VAE, and P-VAE are trained by Adam optimizer with the same learning rate of $1e-3$, weight decay $1e-4$, and $\beta_1 = 0.5$. Other Adam hyper-parameters are set by default in Pytorch.

**A-STN.** We design the noise conditioned A-STN as a 6-layer MLP. Specifically, A-STN takes only 1-dimensional Gaussian noises as input and outputs 6-dimensional affine parameters. Each hidden layer generates 8-dimension features. Batch normalization, ReLU, and dropout (0.5) are applied after each linear layer. The loss weights are set as $\lambda_c^s = 0.1$ and $\beta^s = 0.1$.

**D-VAE.** D-VAE consists of an encoder and a decoder. The encoder maps an image to the 32-dimensional latent space. It includes 5 $3 \times 3$ convolutional layers and three linear layers. The first convolutional layer increases the channel number from 3 to 32. After that, the feature channels double if the convolution stride is 2. There are two convolutional layers on each resolution. The first linear layer takes the reshaped convolutional features and outputs 512-dimensional latent features. Another two linear layers generate the mean and variance vectors of encoded Gaussian distributions. The

decoder is simply the reverse of the encoder with transposed convolutions. The last layer is a $1 \times 1$ convolution producing 2-channel grid maps. We use the weights $\lambda_v^d = 1$, $\lambda_s^d = 10$, and $\beta^d = 1e-2$.

**P-VAE.** P-VAE shares almost the same architecture as D-VAE. The only difference is the last layer in the decoder, because P-VAE needs to generate additive noises on the images. The latent space dimension is set to 8 for P-VAE. We also set the hyper-parameters $\lambda_v^p = 1e-3$ and $\beta^p = 10$.

**Image Classification.** We use datasets CIFAR-10, CIFAR-100, SVHN, and ImageNet with their standard training/test splits. To tune hyper-parameters efficiently, we also sample reduced datasets: R-CIFAR-10, R-CIFAR-100, and R-CIFAR-SVHN. Each of them consists of 4000 examples. The sampling is reproducible using the public sklearn StratifiedShuffleSplit function with random state 0. We report top-1 classification accuracy in most tables except for Table 4.6 with top-1 errors. The target networks are Wide-ResNet-28-10 [112] (W-ResNet), Shake-Shake network [113], and ResNet-50 [10]. We use Cutout [109] as the baselines in Tables 4.1, 4.2, 4.3, and 4.4.

**Medical Image Segmentation.** To test the generalization ability of our approach, we further conduct experiments on the medical image segmentation dataset LiTS [143]. LiTS published 131 liver CT images as a training set with liver and tumor annotations. Since these experiments are to prove the effectiveness of our augmentation algorithm, pursuing the highest performance is not our goal, we use the 2D UNet as the segmentation network to segment CT images on the axial plane slice by slice. We randomly split the 131 CT images into 81 training cases, 25 validation cases, and 25 test cases. We first resample all images to $2 \times 2 \times 2$ mm, then center crop each slice to 256 in size, and finally normalize the image window [-200, 250] to [0, 1] as the input of the network. Only A-STN and D-VAE are presented in this task, since P-VAE has no obvious performance improvement. Compared with classification tasks, segmentation tasks are sensitive to location. Therefore, for A-STN and D-VAE, we not only perform a bilinear grid sample on the images, but also perform a nearest neighbor grid sample for the label masks using the same grid.

We also compared our proposed OnlineAugment with RandAugment [130]. We

Table 4.1: Evaluation of the Gaussian noise input and double cycle-consistency regularization in A-STN. We compare them to the image condition and single cycle-consistency. Double cycle-consistency outperforms the single one. With the double one, the noise and image inputs get comparable accuracy.

| Dataset | Model | Cutout | Image Input +1 cycle | Image Input +2 cycles | Noise Input +1 cycle | Noise Input +2 cycles |
|---|---|---|---|---|---|---|
| R-CIFAR-10 | W-ResNet | 80.95 | 83.24 | 84.76 | 82.62 | **84.94** |

slightly modify RandAugment to fit our setting. As the number of transformations involved is relatively small, all transformations are used during training. Therefore, for global spatial transformation, the search space is the magnitude of 4 transforms, including rotate, translate-x, translate-y, and scale. For the deformation model, the search space is the magnitude of local deformations. At each iteration, the magnitude of each transformation is uniformly and randomly sampled between 0 and the upper bound for both global spatial transformation and local deformation.

### 4.5.2 Ablation study of A-STN, D-VAE, and P-VAE

**A-STN.** The A-STN may be conditioned on image or Gaussian noise. We compare these two choices in this ablation study. Also, we compare its regularization with single or double cycle-consistency losses. Table 4.1 gives the comparisons. The double cycle-consistency obtains higher accuracy than the single cycle one. Because it can make A-STN produce more diverse transformations. With the double-cycle consistency regularization, the noise and image conditions achieve comparable accuracy. We use the noise condition A-STN in other experiments since its architecture is transferable between tasks.

**D-VAE.** The smoothness regularization comes as an additional component in the D-VAE. We evaluate its effectiveness in both CIFAR-10 classification and liver segmentation tasks. Table 4.2 presents the results. Interestingly, the smoothness regularization has little effect on classification accuracy. However, it makes a difference (%2) for the liver segmentation because the liver segmentation is a location-sensitive task. The smoothness regularization can remove the noises along the boundaries of segmentation

Table 4.2: Evaluation of the smoothness regularization (SR) in D-VAE. We report the results on both image classification and segmentation. The smoothness regularization is more useful for the location-sensitive image segmentation task.

| Dataset | Model | Baseline | D-VAE Only | D-VAE + SR |
|---|---|---|---|---|
| R-CIFAR-10 | W-ResNet | 80.95 (Cutout) | 82.72 | **82.86** |
| Liver Segmentation | U-Net | 66.0 (No Aug.) | 68.51 | **70.49** |

Table 4.3: Comparisons of P-VAE to AdvProp [4] with iterative gradient methods PGD [5], GD, and I-FGSM [6]. Adversarial training plus only the noise regularization can make P-VAE comparable to AdvProp with GD or I-FGSM.

| Dataset | Model | Cutout | AP+PGD | AP+GD | AP+I-FGSM | P-VAE |
|---|---|---|---|---|---|---|
| R-CIFAR-10 | W-ResNet | 80.95 | 83.00 | 83.90 | 83.92 | **84.08** |

masks.

**P-VAE.** The P-VAE generates additive noises to perform data augmentation. AdvProp [142] has a similar goal, but utilizes the iterative gradient methods for the generation. For a fair comparison with AdvProp, we use only adversarial training and the noise regularization in training P-VAE. Table 4.3 shows the comparisons. P-VAE compares favorably to AdvProp with GD and I-FGSM. On the one hand, P-VAE learns some noise distributions while the iterative methods rely on the gradient ascent rules. On the other hand, P-VAE generates structured noises, while the iterative approaches produce more complex ones.

### 4.5.3 OnlineAugment *v.s.* AutoAugment

As AutoAugment is a representative offline augmentation method, we compare it to our OnlineAugment. Table 4.4 provides the separate and combined results of three data augmentation models. Each model, independently, can boost the generalization of two target networks on three datasets. Combining them can bring further improvements, achieving comparable performance as AutoAugment. We can also observe that the improvements for the Shake-Shake network [113] are lower than those of the Wide ResNet [112]. One possible explanation is that the stochastic shake-shake operations

Table 4.4: Ours *v.s.* AutoAugment (AA). The three models helps separately, and they together may perform on par with AA. The stochastic shake-shake operations may interfere with the online learning, reducing the improvements.

| Dataset | Model | Cutout | AA | A-STN | D-VAE | P-VAE | Comb. |
|---|---|---|---|---|---|---|---|
| R-CIFAR-10 | Wide-ResNet-28-10 | 80.95 | 85.43 | 84.94 | 82.72 | 84.18 | **85.65** |
| | Shake-Shake (26 2x32d) | 85.42 | **87.71** | 86.62 | 86.51 | 86.34 | 87.12 |
| R-CIFAR-100 | Wide-ResNet-28-10 | 41.64 | 47.87 | 46.55 | 45.42 | 47.45 | **48.31** |
| | Shake-Shake (26 2x32d) | 44.41 | **48.18** | 46.81 | 46.53 | 46.30 | 47.27 |
| R-SVHN | Wide-ResNet-28-10 | 90.16 | 93.27 | 92.73 | 91.32 | 91.61 | **93.29** |
| | Shake-Shake (26 2x32d) | 94.03 | **94.63** | 94.15 | 94.06 | 94.12 | 94.21 |

Table 4.5: Ours+AutoAugment (AA). We use A-STN, D-VAE, and P-VAE on top of the AutoAugment polices. Surprisingly, each model can further improve AutoAugment performance. It demonstrates that OnlineAugment is orthogonal to AutoAugment. The three models use more general augmentation operations.

| Dataset | Model | AA | +A-STN | +D-VAE | +P-VAE | +Comb. |
|---|---|---|---|---|---|---|
| R-CIFAR-10 | Wide-ResNet-28-10 | 85.43 | 89.39 | 87.40 | 87.63 | **89.40** |
| | Shake-Shake (26 2x32d) | 87.71 | 89.25 | 88.43 | 88.52 | **89.50** |
| R-CIFAR-100 | Wide-ResNet-28-10 | 47.87 | 52.94 | 50.01 | 51.02 | **53.72** |
| | Shake-Shake (26 2x32d) | 48.18 | **50.58** | 50.42 | 50.87 | 50.11 |
| R-SVHN | Wide-ResNet-28-10 | 93.27 | 94.32 | 93.72 | 94.17 | **94.69** |
| | Shake-Shake (26 2x32d) | 94.63 | 95.21 | 94.87 | **95.28** | 95.06 |

may affect the online learning of data augmentation.

### 4.5.4 OnlineAugment + AutoAugment.

Apart from comparing OnlineAugment with AutoAugment, it is more interesting to investigate their orthogonality. Table 4.5 summarizes the results. We can find that OnlineAugment can bring consistent improvements on top of AutoAugment for different target networks and datasets. Their orthogonality comes from the differences in training schemes and augmentation models. Different from OnlineAugment, AutoAugment learns data augmentation policies in an offline manner. Moreover, the three models (A-STN, D-VAE, and P-VAE) generate different augmentations from the image processing functions in AutoAugment. Note that OnlineAugment is also orthogonal to

Table 4.6: Comparisons with state-of-the-art methods. We compare our OnlineAugment with AutoAugment (AA), PBA [7], and Fast AutoAugment (FAA) [8] on three datasets. OnlineAugment alone obtains comparable test errors. Combining it with AutoAugment produces the lowest errors on three datasets.

| Dataset | Model | Baseline | Cutout | AA | PBA | FAA | Ours | Ours+AA |
|---|---|---|---|---|---|---|---|---|
| CIFAR-10 | Wide-ResNet-28-10 | 3.9 | 3.1 | 2.6 | 2.6 | 2.7 | 2.4 | **2.0** |
| CIFAR-100 | Wide-ResNet-28-10 | 18.8 | 18.4 | 17.1 | 16.7 | 17.3 | 16.6 | **16.3** |
| ImageNet | ResNet-50 | 23.7 | - | 22.4 | - | 22.4 | 22.5 | **22.0** |

Table 4.7: OnlineAugment *v.s.* RandAugment on LiTS measured by Dice score coefficient. Although A-STN is comparable to RandAug STN, D-VAE alone and its combination with A-STN obtain higher scores than the RandAug variants.

| Method | Liver | Tumor | Average |
|---|---|---|---|
| no Augmentaion | 89.04 | 44.73 | 66.88 |
| RandAug STN | **93.86** | 50.54 | 72.20 |
| RandAug Deformation | 90.49 | 46.93 | 68.71 |
| RandAug Combine | 91.91 | 51.11 | 71.51 |
| Ours A-STN | 92.01 | 52.26 | 72.13 |
| Ours D-VAE | 90.18 | 50.81 | 70.49 |
| Ours Combine | 93.12 | **53.58** | **73.35** |

other offline methods [9, 130, 7] since they have similar policies as AutoAugment.

### 4.5.5 Comparisons with State-of-the-art Methods.

**Image classification**. Besides AutoAugment, we also compare OnlineAugment with other state-of-the-art methods such as Fast AutoAugment (FAA) and Population-based Augmentation (PBA). They all belong to offline data augmentation. OnlineAugment alone gets the lowest test errors on CIFAR-10 and CIFAR-100 and comparable errors on ImageNet. Further, OnlineAugment, together with AutoAugment, achieves new state-of-the-art results on all the three image classification datasets.

**Medical image segmentation.** OnlineAugment can also easily apply to medical image segmentation. Table 4.7 gives the comparisons with state-of-the-art RandAugment. A-STN has comparable scores as RandAugment STN. However, both D-VAE and its joint application with A-STN outperform the corresponding RandAugment parts,

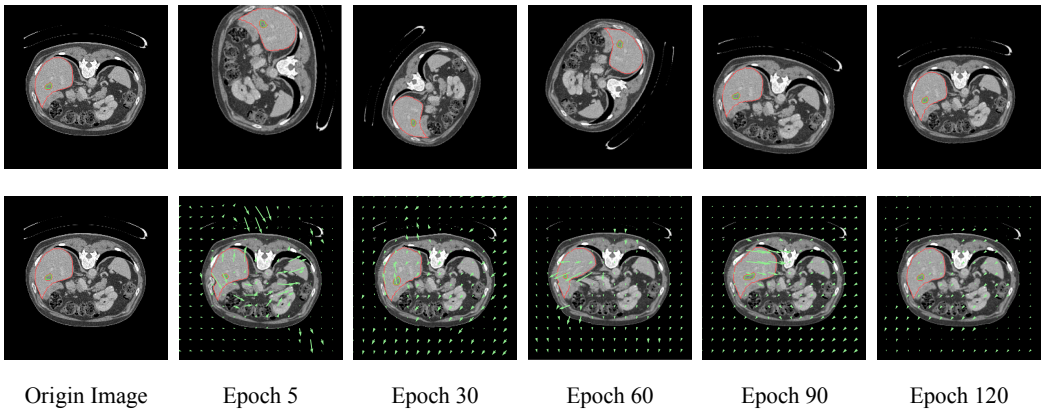



Figure 4.3: Visualization of two augmentation modules: A-STN (**top**) and D-VAE (**bottom**). Red line is the contour of liver while green line is the contour of tumor. Our OnlineAugment can generate diverse augmented images. Moreover, it can also adapt to the target network. As the target network converges during training, the magnitude of the augmentation will also decrease.

especially on the tumor segmentation. Fig. 4.3 illustrates the augmented images of A-STN, and D-VAE along with the training process. The augmentation is relatively large at the beginning and gradually becomes small as the training converges.

### 4.5.6 Running Time Comparison

For fair comparisons, we divide the running time into two parts: offline searching time and online training time. The offline data augmentation methods must learn augmentation policies on separate proxy tasks. Then they apply the policies for online training. Our OnlineAugment, by contrast, performs the online training directly. Tables 4.8 and 4.9 give offline and online time comparisons, respectively. Ours is superior in terms of zero offline time cost. For online training, ours is slower as it needs to update both the target learner and augmentation networks in each iteration. According to Table 4.9, the gap is smaller for small image training. In this case, A-STN is even faster.

There are several possible ways to improve the training efficiency of OnlineAugment. One is to reduce the frequency of updating the augmentation networks A-STN, D-VAE, and P-VAE. Currently, we update them in each iteration of updating the target learner. The online training time will significantly decrease if updating them less frequently.

Table 4.8: GPU hours of offline searching time. AutoAugment (AA) [9], Population-based Augmentation (PBA) [7], and Fast AutoAugment (Fast AA) [8] need to search augmentation polices on separate proxy tasks. In contrast, our OnlineAugment has no offline searching cost.

| Dataset | AA | PBA | Fast AA | OnlineAugment |
|---|---|---|---|---|
| CIFAR-10 | 5000 | 5.0 | 3.5 | 0.0 |
| SVHN | 1000 | 1.0 | 1.5 | 0.0 |
| ImageNet | 15000 | - | 450 | 0.0 |

Table 4.9: Per iteration seconds in online training. We measure the time using different input image resolutions and workers in Pytorch data loaders. In the experiments, we train ResNet50 [10] with batch size 128 using 1 RTX 8000 GPU and Intel(R) Xeon(R) Silver 4116 CPUs. Since all the offline methods share the same augmentation policy format, they should have equivalent online training time costs. Thus, we use AutoAugment (AA) [9] to represent all offline methods here. Ours have higher online time costs due to updating augmentation networks.

| | 32×32 Image (ResNet50, BS: 128) | | | | | 224×224 Image (ResNet50, BS: 128) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Workers | AA | A-STN | D-VAE | P-VAE | Comb. | AA | A-STN | D-VAE | P-VAE | Comb. |
| 0 | 0.17 | 0.14 | 0.17 | 0.17 | 0.27 | 0.84 | 1.21 | 1.37 | 1.32 | 3.29 |
| 1 | 0.17 | 0.12 | 0.14 | 0.15 | 0.27 | 0.50 | 0.94 | 1.06 | 1.03 | 2.97 |

Another direction is to optimize the architectures of A-STN, D-VAE, and P-VAE. We can also reduce the training costs by using more compact models.

### 4.5.7 Visualization of OnlineAugment Adaptivity

OnlineAugment can adapt to the target learner in training. To demonstrate the adaptivity, we draw curves in Figure 4.4 to measure the augmentation strength. We can find the augmentations are relatively small at the first few epochs. As the training continues, the augmentations become more substantial as the target learner already learns enough knowledge from the clean data. Finally, the training converges with reduced augmentations. The target learner with a small learning rate may not require significant data augmentations. An exception is that P-VAE has a considerable image distance at the beginning. The large noises are due to our initialization of the P-VAE model, which we leave for future studies.

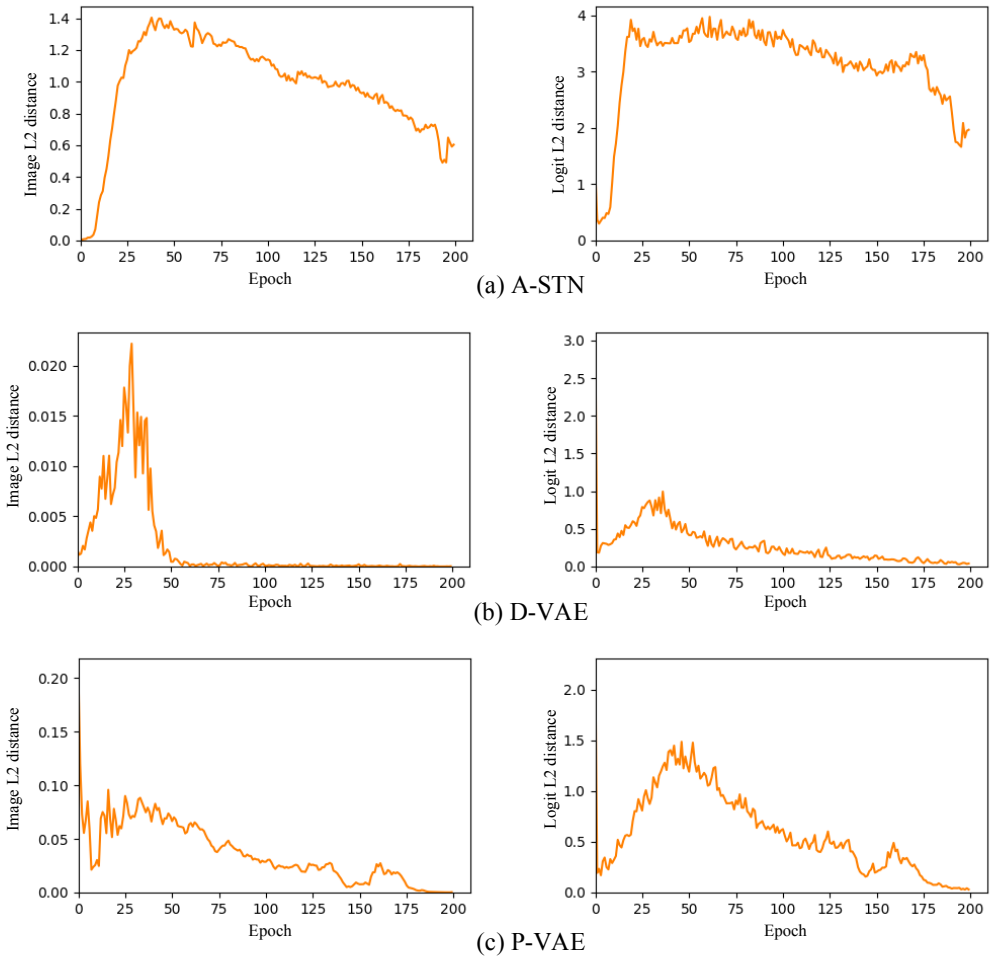



Figure 4.4: Illustration of augmentation strengths for A-STN, D-VAE, and P-VAE along epochs. We measure the strengths using the L2 distances between the clean and augmented data. The **left** and **right** columns show the L2 distances in the image and logit spaces. The trend is that the augmentation strengths increase in the early stages of training, while during the target network converges, the augmentation magnitude gradually decreases.

## 4.6 Summary

In this work, we have presented OnlineAugment - a new and powerful data augmentation technique. Our method adapts online to the learner state throughout the entire training. We have designed three new augmentation networks that are capable of learning a wide variety of local and global geometric and photometric transformations, requiring less domain knowledge of the target task. Our OnlineAugment integrates both adversarial training and meta-learning for efficient training. We also design essential regularization techniques to guide adaptive online augmentation learning. Extensive

experiments demonstrate the utility of the approach to a wide variety of tasks, matching (without requiring expensive offline augmentation policy search) the performance of the powerful AutoAugment policy, as well as improving upon the state-of-the-art in augmentation techniques when used jointly with AutoAugment.

# Chapter 5

# Towards Efficient U-Nets: A Coupled and Quantized Approach

## 5.1 Introduction

The U-Net architecture [40] is a basic category of Convolution Neural Network (CNN). It has been widely used in location-sensitive tasks, such as semantic segmentation [64], biomedical image segmentation [40], human pose estimation [60], and facial landmark localization [105]. A U-Net contains several corresponding top-down and bottom-up blocks with shortcut connections between them. The essence of U-Net is integrating both the local visual cues and global context information to make inferences.

Recently, stacked U-Nets, *e.g.* hourglasses (HGs) [1] have become a standard baseline in landmark localization tasks. Stacked U-Nets have multiple top-down and bottom-up processing which can refine inferences stage-by-stage. Many techniques, such as adversarial training [3], attention modeling [86], are used to further improve its inference accuracy. However, very few works try to improve the efficiency of stacked U-Nets.

Stacked U-Nets usually contain dozens of millions of float parameters. The massive high-precision computations require the high-end GPU devices with abundant memory. It is very challenging for the applications in resource-limited mobile devices. In this work, we aim to improve the efficiency of staked U-Nets in three aspects: parameter, memory, and bit-width.

**Parameter efficiency.** The shortcut connections can promote feature reuse, thereby reducing many redundant parameters. For a single U-Net, it is straightforward to change each block into a dense block where several convolutional layers are densely

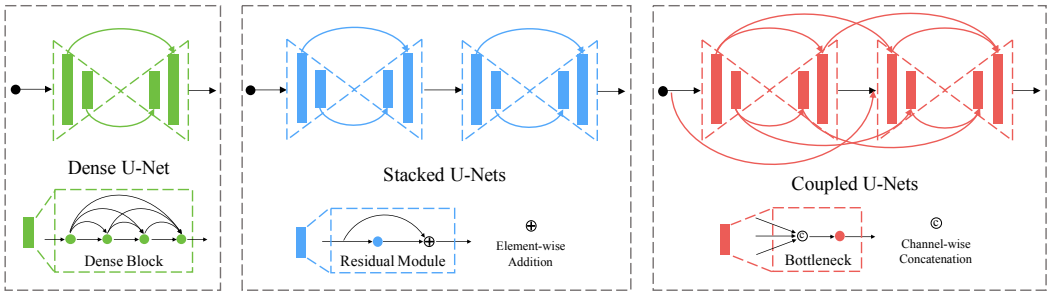



Figure 5.1: Illustration of a dense U-Net, stacked U-Nets, and coupled U-Nets. Coupled U-Nets is a hybrid of dense U-Net and stacked U-Nets, integrating the merits of both dense connectivity and multi-stage top-down and bottom-up refinement. Coupled U-Nets can save ∼70% parameters and ∼30% inference time of stacked U-Nets. Each block in coupled U-Nets is a bottleneck module which is different from the dense block.

connected.

While it is easy to apply dense connectivity to a single U-Net, adding shortcut connections properly in the stacked U-Nets is nontrivial. Our solution is to couple the stacked U-Nets, generating the coupled U-Nets (CU-Net). The key idea is to directly connect blocks of the same semantic meanings, *i.e.* having the same resolution in either top-down or bottom-up context, from any U-Net to all subsequent U-Nets. Refer to Fig. 5.1 for an illustration. The coupling connections encourages feature reuse across stacks, resulting in a light-weighted CU-Net.

Yet there is an issue in designing the CU-Net. The number of shortcut connections would have a quadratic growth if we couple every U-Net pair, *e.g.* $n$ stacked U-Nets would generate $O(n^2)$ connections. To balance parameter efficiency and inference accuracy, we propose the *order-K* coupling that couples a U-Net to its $K$ instance successors.

Additionally, we employ intermediate supervisions to provide additional gradients, compensating the trimmed off shortcut connections. The *order-K* coupling cuts down ∼70% parameter number and ∼30% forward time without sacrificing inference accuracy compared with stacked U-Nets [1]. Furthermore, we propose an iterative design that can further reduce the parameter number to ∼50%. More specifically, the CU-Net output of the first pass is used as the input of the second pass, which is equivalent to a double-depth CU-Net.

**Memory efficiency.** The shortcut connections may have a severe memory issue. For instance, a naive implementation intends to make feature copies repeatedly for all shortcut connections. We adapt the memory efficient implementation [144] to share memories for features in connected blocks. This technique can reduce the training memory by ∼40%.

**Bit-width efficiency.** In addition to the parameter and memory efficiency, we also investigate model quantization to improve the bit-width efficiency. Different from the common setup, we quantize both parameters and data flow ( intermediate features and gradients). On the one hand, we ternarize or binarize the float parameters, which shrinks 16× or 32× model size in testing. On the other hand, we quantize the data flow with different bit-width setups, which saves ∼4× training memory without compromising the performance. To the best of our knowledge, this is the first study to simultaneously quantize the parameters and the data flow in U-Nets.

In summary, we present a comprehensive study of efficient U-Nets [51] in three aspects: parameter, memory, and bit-width. Coupled U-Nets (CU-Nets), *order-K* coupling and iterative refinement are proposed to balance the parameter efficiency and inference accuracy. Besides, a memory sharing technique is employed to significantly cut down the training memory. Moreover, we investigate the bit-width efficiency by quantizing the parameters as well as the data flow. Two popular tasks, human pose estimation and facial landmark localization, are studied to validate our approach in various aspects. The experimental results prove that our model cuts down ∼70% parameter number and ∼30% inference time. Together with the quantization, we can shrink the model size by ∼98% and reduces ∼75% training memory with comparable performance as state-of-the-art U-Nets designs.

## 5.2 Related Work

In this section, we review the recent developments on designing convolutional network architectures, quantizing the neural networks and two landmark localization tasks: human pose estimation and facial landmark localization.

**Network Architecture.** The research on network architectures has been active since AlexNet [54] appeared. First, by using smaller filters, VGG [45] network became several times deeper than the AlexNet and obtained much better performance. Then the Highway Networks [145] extended its depths to more than 100 layers. The identity mappings make it possible to train very deep ResNet [10]. The popular stacked U-Nets [1] are designed based on residual modules. More recently, DenseNet [90] has outperformed the ResNet in image classification tasks. Some works [146, 147] have tried to use the dense connectivity locally within each U-Net block, following the DenseNet [90] design. However, the proposed coupling connectivity is global at the U-Net level. Moreover, we aim to improve the U-Net efficiency whereas they focus on accuracy. Our method is also related to DLA [148] in the sense of feature aggregation. However, the proposed coupling connectivity is designed for multiple stacked U-Nets whereas DLA [148] is for single U-Net.

**Network Quantization.** Training deep neural networks usually consumes a large amount of computational power, which makes it hard to deploy on mobile devices. Recently, network quantization approaches [149, 150, 151, 152, 153] offer an efficient solution to reduce the network size by cutting down high precision operations and operands. TWN [150] utilize two symmetric thresholds to ternarize the parameters to +1, 0, or -1. XNOR-Net [153] quantize the parameters and intermediate features. It also uses a scaling factor to approximate the real-value parameters and features. DoReFa-Net [151] quantizes gradients to low bit-width numbers. WAGE [152] proposes an integer-based implementation for training and inference simultaneously. These quantization methods are mainly designed for the image classification networks. In the recent binarized convolutional landmark localizer (BCLL) [154] architecture, XNOR-Net [153] is utilized for network binarization. However, BCLL only quantizes parameters for inference. Due to its high precision demand for training, it cannot save training memory and improve training efficiency. Therefore, we explore to quantize the proposed CU-Net in training and inference simultaneously. That is, we quantize the parameters as well as the intermediate features and gradients.

**Human Pose Estimation.** Starting from the DeepPose [60], CNN-based approaches [82, 75, 83, 78, 81, 79, 97, 155] have become the mainstream in human pose estimation and prediction. Recently, the architecture of stacked U-Nets [1] has obviously beaten all the previous ones in terms of usability and accuracy. Therefore, all recent state-of-the-art methods [86, 87, 71, 3] build on its architecture. Chu *et. al.* add the Conditional Random Field to refine its prediction. Yang *et. al.* replace the residual modules in stacked U-Nets with more sophisticated ones. Chen *et. al.* [71] use an additional network to provide adversarial supervisions. Peng *et. al.* [3] use adversarial data augmentation to train more robust stacked U-Nets. All these approaches focus on boosting the inference accuracy. In contrast, we study to improve the efficiency of stacked U-Nets in various aspects.

**Facial Landmark Localization.** Similarly, CNNs have largely reshaped the field of facial landmark localization. Traditional methods could be easily outperformed by the CNNs based [106, 107, 63, 156]. Especially, the network cascade has shown its effectiveness in detecting the facial keypoints. Sun *et. al.* [157] propose to use three levels of neural networks to predict landmark locations. Zhang *et. al.* [158] study the problem via cascades of stacked auto-encoders which gradually refine the landmark positions. In the recent Menpo Facial Landmark Localization Challenge [108], stacked U-Nets [1] achieve state-of-the-art performance. The proposed *order-K* coupled U-Nets could produce comparable localization errors but with much fewer parameters, smaller model size, and less training memory.

## 5.3 Method

We first describe the dense connectivity to improve the efficiency of a single U-Net. Then we introduce the coupling connectivity to boost the efficiency of stacked U-Nets. The *order-K* coupling is presented to balance the accuracy and parameter efficiency. We also give an iterative refinement to cut the CU-Net in one half. At last, we quantize its the parameters, intermediate features and gradients.

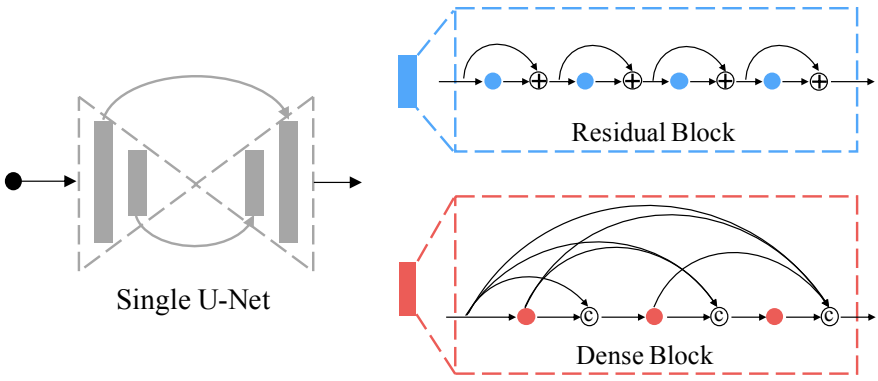



Figure 5.2: Illustration of single residual U-Net and dense U-Net. Each top-down or bottom-up block in the residual U-Net is a residual block of several residual modules. In contrast, each block in the dense U-Net is a dense block with several densely connected layers. The dense connections promote the feature reuse inside each block. Therefore, we could largely reduce the parameters of a single U-Net by replacing each residual block with a dense block.

### 5.3.1 Dense Connectivity for Single U-Net

A U-Net [40] is usually symmetric with the same number of top-down and bottom-up blocks. And a pair of top-down and bottom-up blocks with the same resolutions are connected. In this work, we refer to each top-down or bottom-up block as a semantic block. We intend to add shortcut connections to compress a single U-Net. One straightforward way is to densely connect the convolutional layers of a semantic block, forming a dense block. An illustration is shown in Figure 5.2.

The dense connections could increase the information flow in the U-Net. However, the single dense U-Net follows the DenseNet design [90]. That is, the dense connections are only within the local blocks. If we have several stacked U-Nets, how could we add the shortcut connections? It is more meaningful to improve the efficiency of stacked U-Nets since they are commonly used in practice.

### 5.3.2 Coupling Connectivity for Stacked U-Nets

Suppose multiple U-Nets are stacked together, for the $\ell^{th}$ top-down and bottom-up blocks in the $n^{th}$ U-Net, we use $f_\ell^n(\cdot)$ and $g_\ell^n(\cdot)$ to denote their non-linear transformations. Their outputs are represented by $\mathbf{x}_\ell^n$ and $\mathbf{y}_\ell^n$. $f_\ell^n(\cdot)$ and $g_\ell^n(\cdot)$ comprise operations of Convolution (Conv), Batch Normalization (BN) [159], rectified linear units (ReLU)

[160], and pooling. Note that the top-down and bottom-up blocks have independent index $\ell$. To make the resolutions of $x_\ell^n$ and $y_\ell^n$ match, their indexes $\ell$ both increase from the high to low resolutions.

**Stacked U-Nets.** We recap the popular stacked U-Nets [1] based on the residual modules. Basically, each block is a residual module. The feature transitions at the $\ell^{th}$ top-down and bottom-up blocks of the $n^{th}$ U-Net can be written as:

$$\mathbf{x}_\ell^n = f_\ell^n(\mathbf{x}_{\ell-1}^n), \mathbf{y}_{\ell-1}^n = g_{\ell-1}^n(\mathbf{y}_\ell^n + \mathbf{x}_\ell^n). \tag{5.1}$$

The shortcut connections only exist locally within each U-Net. It restricts the feature reuse across U-Nets. Thus, it contains many redundant parameters.

**Coupled U-Nets.** To facilitate the feature reuse across stacked U-Nets, we propose a global connectivity pattern. The same semantic blocks, i.e., blocks at the same locations of different U-Nets, have direct connections. Hence, we refer to this coupled U-Nets architecture as *CU-Net*. Figure 5.1 gives an illustration. It essentially merges features from multiple sources and then generates new features. Mathematically, the feature transitions at the $\ell^{th}$ top-down and bottom-up blocks of the $n^{th}$ U-Net can be formulated as:

$$\mathbf{x}_\ell^n = f_\ell^n([\mathbf{x}_{\ell-1}^n, \mathbf{X}_\ell^{n-1}]), \mathbf{y}_{\ell-1}^n = g_{\ell-1}^n([\mathbf{y}_\ell^n, \mathbf{x}_\ell^n, \mathbf{Y}_\ell^{n-1}]), \tag{5.2}$$

where $\mathbf{X}_\ell^{n-1} = \mathbf{x}_\ell^0, \mathbf{x}_\ell^1, \cdots, \mathbf{x}_\ell^{n-1}$ are the outputs of the $\ell^{th}$ top-down blocks in all preceding U-Nets. Similarly, $\mathbf{Y}_\ell^{n-1} = \mathbf{y}_\ell^0, \mathbf{y}_\ell^1, \cdots, \mathbf{y}_\ell^{n-1}$ represent the outputs from the $\ell^{th}$ bottom-up blocks. $[\cdots]$ denotes the feature concatenation, which could make information flow more efficiently than the summation operation in Equation 5.1.

According to Equation 5.2, a block receives features not only from connected blocks in the current U-Net but also the output features of the same semantic blocks from all its preceding U-Nets. Each U-Net becomes light-weighted, benefiting from the feature reuse across stacked U-Nets. Thus, the parameter efficiency is largely improved. Please note that the coupling connectivity is global whereas the dense connectivity in DenseNet [90] is local within blocks.

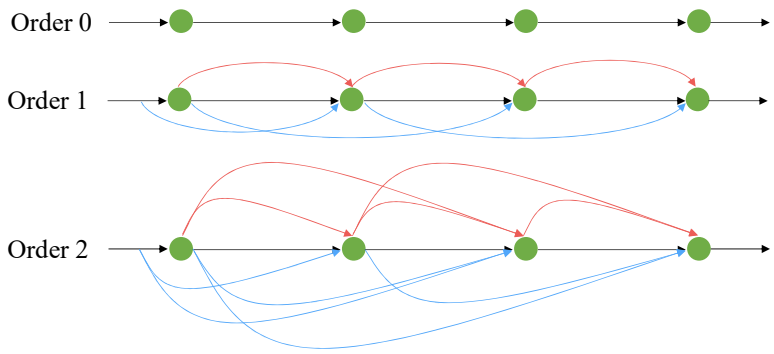



Figure 5.3: Illustration of *order-K* coupling. For simplicity, each dot represents one U-Net. The red lines are shortcut connections for the same semantic blocks in different U-Nets. The initial input and the U-Net outputs pass through the blue lines. *Order*-0 coupling (**Top**) strings U-Nets together only by their inputs and outputs, i.e., stacked U-Nets. *Order*-1 coupling (**Middle**) has shortcut connections only for adjacent U-Nets. Similarly, *order*-2 coupling (**Bottom**) has shortcut connections for 3 nearby U-Nets.

### 5.3.3 Order-K Coupling

In the above formulation of CU-Net, we connect blocks with the same semantic meanings across all U-Nets. The shortcut connections would have quadratic growth depth-wise. To make CU-Net more parameter efficient, we propose to cut off some trivial connections. For compensation, we add a supervision at the end of each U-Net. Both the intermediate supervisions and the shortcut connections could alleviate the vanishing gradient problem, helping train better CU-Net models. Mathematically, the features $\mathbf{X}_\ell^{n-1}$ and $\mathbf{Y}_\ell^{n-1}$ in Equation 5.2 become

$$\mathbf{X}_\ell^{n-1} = \mathbf{x}_\ell^{n-k}, \cdots, \mathbf{x}_\ell^{n-1}, \tag{5.3}$$

$$\mathbf{Y}_\ell^{n-1} = \mathbf{y}_\ell^{n-k}, \cdots, \mathbf{y}_\ell^{n-1}, \tag{5.4}$$

where $0 \leq k \leq n$ represents how many preceding nearby U-Nets connect with the current one. $k = n$ or $k = 0$ would result in the stacked U-Nets or fully densely connected U-Nets. A medium order could reduce the growth of CU-Net parameters from quadratic to linear. Therefore, the *order-K* coupling greatly improves the parameter efficiency of CU-Net and could make CU-Net grow several times deeper.

The proposed *order-K* coupling has a similar philosophy as the Variable Order Markov (VOM) models [161]. Each U-Net can be viewed as a state in the Markov model. The current U-Net depends on a fixed number of preceding nearby U-Nets, instead of

preceding either only one or all U-Nets. In this way, the long-range connections are cut off. Figure 5.3 illustrates the couplings of three different orders. The shortcut connections exist for the U-Net semantic blocks and U-Net inputs. We differentiate them with the red and blue colors in Figure 5.3. We define the coupling order based on either the red or blue lines. In Figure 5.3, both the red and blue shortcut connections follow the VOM patterns of *order*-0, *order*-1 and *order*-2.

We could extend the *order-K* coupling to more general *order-K* connectivity if each U-Net is simplified as a unit. Dense connectivity [90] is a special case of *order-K* connectivity on the limit of $K$. For small $K$, *order-K* connectivity is much more parameter efficient. But fewer connections may affect the inference accuracy of very deep CU-Net. To make CU-Net have both high parameter efficiency and inference accuracy, we propose to use *order-K* connectivity in conjunction with intermediate supervisions. In contrast, DenseNet [90] has only one supervision at the end. Thus, it cannot effectively take advantage of *order-K* connectivity.

### 5.3.4 Iterative Refinement

In order to further improve the parameter efficiency of CU-Net, we consider an iterative refinement. It uses only half of a CU-Net but may achieve comparable accuracy. In the iterative refinement, a CU-Net has two forward passes. In the first pass, we concatenate the inputs of the first and last U-Nets and merge them in a small dense block. More specifically, if the input features of a U-Net have $n$ channels, concatenating the inputs of the first and last U-Nets results in $2n$ channel features. Then we forward them through 4 densely connected Conv3×3 layers. Each layer produces $m$ channel new features. Then we concatenate all $2n + 4m$ features and use one Conv1×1 layer to compress them to $n$ channel features, the modified input. Then the modified input is fed forward in the CU-Net again. An illustration of the iterative refinement is shown in Figure 5.4.

The iterative refinement may cause the overfitting. We provide two techniques to avoid this. First, independent batch normalization parameters are learned in the two iterations. Second, we call the backpropagation separately for each forward pass. Two different mini-batch images are used to update the two iterations. We forward the first

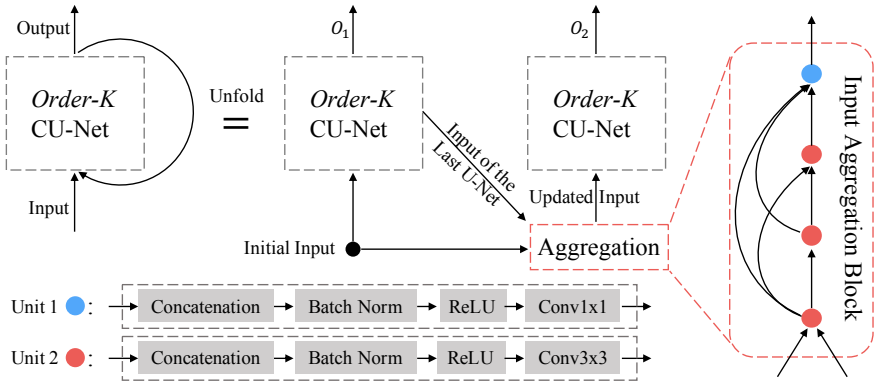



Figure 5.4: Illustration of iterative refinement. The same *order-K* CU-Net is used twice in the iterative refinement. In the first iteration, the input of the last U-Net is generated on the basis of the initial input. Then they are concatenated and further aggregated in a dense block. The updated input is forwarded through the *order-K* CU-Net to get the final output. Given a long CU-Net cascade, the iterative refinement has the potential to reduce its depth by half and still maintain comparable accuracy.

batch only for the first pass and update the network using the gradient descent. The second batch is forwarded in two passes and the gradient descent is only applied on the second pass.

In this iterative pipeline, the CU-Net has two groups of supervisions in the first and second iterations. Both detection supervision (binary cross entropy loss) and regression (mean squared error) supervision [83] are already used in landmark detection tasks. However, there is no investigation of how they compare with each other. To this end, we try different combinations of detection and regression supervisions for two iterations. Our comparison could give some guidance for future research.

### 5.3.5 Quantization of Parameter, Feature and Gradient

Apart from shrinking the parameter number, we also investigate quantizing each parameter, intermediate feature and gradient. Quantizing the parameters could improve the efficiency in both training and inference. For a parameter $w_i$ in a convolutional filter $W$, we could binarize a parameter through the sign function:

$$q(w_i) = sign(clip(w_i, -1, 1)), \tag{5.5}$$

where *clip* is a saturation function that clips parameter $w_i$ to [-1, 1]. Or we ternarize $w_i$ with a threshold-based function:

$$q(w_i) = \begin{cases} +1, & w_i > t \\ 0, & |w_i| \leq t \\ -1, & w_i < -t \end{cases}, \tag{5.6}$$

where $t \approx \frac{0.7}{n} \sum_{i=1}^{n} |w_i|$ is a positive threshold. As XNOR-Net [153], we also try to use a scaling factor $\alpha$ to approximate the real-value weight.

Quantizing the intermediate features and gradients, i.e., the dataflow, could boost training efficiency by reducing training memory. We follow the WAGE [152]. The dataflow is quantized to $k$-bit values by the following linear mapping function:

$$q(x, k) = clip(\sigma(k) \cdot round(x\sigma(k)) - 1 + \sigma(k), 1 - \sigma(k)) \tag{5.7}$$

where $k$ is the pre-defined bit-width and $\sigma(k) = \frac{1}{2^{k-1}}$ is the unit distance function. In the following experiments, we explore different combinations of bit-widths to balance the accuracy and training memory consumption.

## 5.4 Implementation

In this section, we give the detailed architecture of CU-Net and the implementation of *order-K* coupling through the queue data structure. Besides, a memory efficient implementation for the shortcut connections is also described.

### 5.4.1 CU-Net Architecture

In the original stacked U-Nets, there is mainly one feature flow going through each U-Net. The coupled U-Nets still have a main feature flow along the U-Nets cascade. Let $m$ denote the feature number in the main flow and $n$ represent the generated feature number at each semantic block of U-Net, i.e. red block in Figure 5.5. The generated features are forwarded through the shortcut connections.

Before entering a red semantic block, the main feature flow and the shortcut feature flow are merged by channel-wise concatenation. For each top-down block of the $i^{th}(i \geq$

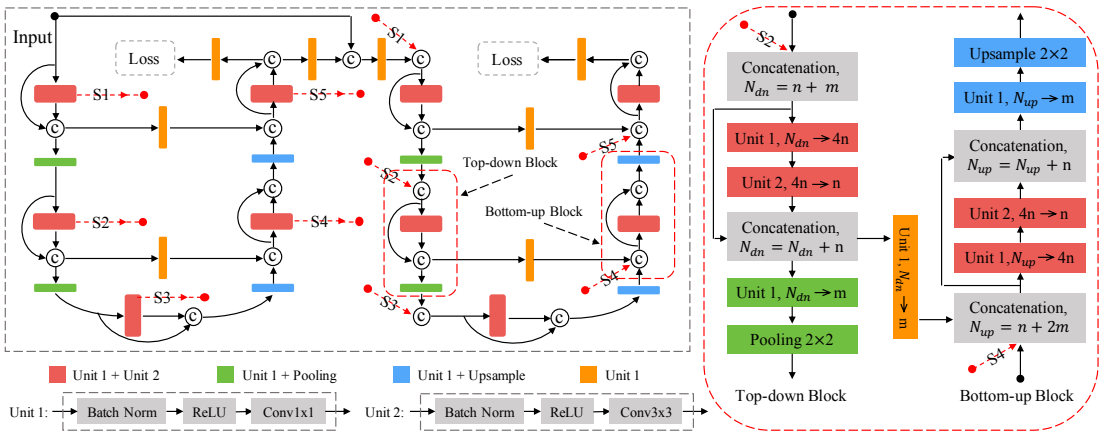



Figure 5.5: Implementation of CU-Net. The left figure shows 2 U-Nets coupled together through the red dot lines. Each U-Net has its own supervision. For simplicity, we only show a pair of top-down and bottom-up semantic blocks. The right figure gives the detailed implementation of a pair of semantic blocks in the second U-Net. Basically, $m$ features from the former block of current U-Net and $n$ features from the same semantic block of the preceding U-Net are concatenated and transformed to $4n$ features by a conv1x1. A conv3x3 then generates $n$ new features and another conv1x1 compresses the $m + n$ input and $n$ generated features to $m$ features. The bottom-up block needs to concatenate the additional $m$ skipped features.

0) U-Net, its inputs contain the $m$ features in the main flow and another $n \times min(i, K)$ features from the shortcut connections of previous $K$ U-Nets, where $min(i, K)$ indicates the lesser one of the order $K$ and $i$. They are concatenated channel-wise to $m + n \times min(i, K)$ features. For each bottom-up block of the $i^{th}(i \geq 0)$ U-Net, its inputs include additional $m$ shortcut features from the corresponding top-down blocks. Thus, its inputs have $2m + n \times min(i, K)$ features.

Then a $1 \times 1$ convolution compresses the input features to $4 \times n$ features. A following $3 \times 3$ convolution produces $n$ new features. Last, the $m + n \times min(i, K)$ (top-down block) or $2m + n \times min(i, K)$ (bottom-up block) input features and the $n$ generated features are concatenated. Another $1 \times 1$ convolution compresses them to $m$ output features, flowing into the next block.

### 5.4.2 *Order-K* Coupling Implementation

*Order-K* coupling describes the connectivity for the whole U-Net. Inside a U-Net, each semantic block has *order-K* connectivity. That is, the *order-K* coupling consists of the

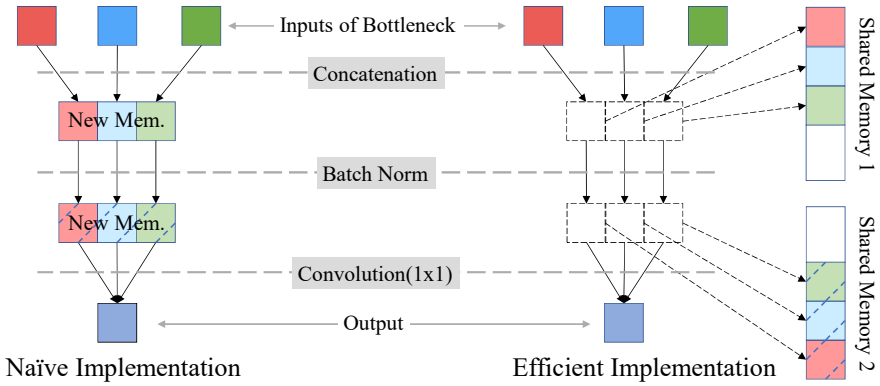



Figure 5.6: Illustration of memory efficient implementation. It is for the Concat-BN-ReLU-Conv($1 \times 1$) in each bottleneck structure. ReLU is not shown since it is an in-place operation with no additional memory request. The efficient implementation pre-allocates two fixed memory space to store the concatenated and normalized features of connected blocks. In contrast, the naive implementation always allocates new memories for them, causing high memory consumption.

*order-K* connectivity. Therefore, we only need to implement the *order-K* connectivity.

A shortcut connection means one feature reuse. In the *order-K* connectivity, a semantic block would use the features generated by the same semantic blocks in the preceding $K$ U-Nets. Thus, for the current U-Net, we need to store the history features only from the preceding $K$ U-Nets. As the U-Nets are used sequentially one-by-one, a length-$K$ history sliding window also moves forward.

We use the dynamic queue structure to simulate this process. More specifically, we assign one queue for the same semantic blocks in all U-Nets. To save the memory, a queue only stores the references to the features instead of the feature copies. If the current U-Net index is less than $K$, we keep appending the feature reference to the queue rear. Otherwise, we first remove the old reference from the queue front and then append a new reference to the rear. The feature reference at the front is the least recent. Therefore, it is removed with the highest priority.

### 5.4.3 Memory Efficient Implementation

Benefitting from *order-K* coupling, CU-Net is quite parameter efficient. However, a naive implementation would prevent one from training very deep CU-Nets because the concatenation operation produces features detached from its inputs. In other words,

the concatenated shortcut features require new space. Thus, the shortcut connections could increase the memory demand.

To reduce the training memory, we follow the efficient implementation [144]. More specifically, concatenation operations of the same semantic blocks in all U-Nets share a memory allocation, and their subsequent batch norm operations share another memory allocation. Suppose a CU-Net includes $N$ U-Nets each of which has $L$ top-down blocks and $L$ bottom-up blocks. We need to pre-allocate two memory spaces for each of $2L$ semantic blocks. For the $\ell^{th}$ top-down blocks, the concatenated features $[\mathbf{x}_{\ell-1}^{1}, \mathbf{X}_{\ell}^{0}], \cdots, [\mathbf{x}_{\ell-1}^{N-1}, \mathbf{X}_{\ell}^{N-2}]$ share the same memory space. Similarly, the concatenated features $[\mathbf{y}_{\ell-1}^{0}, \mathbf{x}_{\ell}^{0}], [\mathbf{y}_{\ell-1}^{1}, \mathbf{x}_{\ell}^{1}, \mathbf{Y}_{\ell}^{0}], \cdots, [\mathbf{y}_{\ell-1}^{N-1}, \mathbf{x}_{\ell}^{N-1}, \mathbf{Y}_{\ell}^{N-2}]$ in the $\ell^{th}$ bottom-up blocks share the same memory space.

In one shared memory allocation, later produced features would overlay the former features. Thus, the concatenations and their subsequent batch norm operations require to be re-computed in the backward phase. Figure 5.6 illustrates naive and efficient implementations.

## 5.5 Experiments

In this section, we evaluate each component in the proposed method. First, we select the hyper-parameters and evaluate the intermediate supervisions in the CU-Net. Then we compare the CU-Net with the single dense U-Net to show its accuracy superiority. After that, we evaluate the *order*-$K$ coupling with the intermediate supervisions. Then we show the *order*-1 CU-Net is much more parameter efficient than the stacked U-Nets [1]. We also evaluate the iterative refinement to halve CU-Net parameters and test the quantization of parameters, intermediate features, and gradients. Finally, we compare the quantized *order*-1 CU-Net with state-of-the-art methods in both human pose estimation and facial landmark localization. Some qualitative results are also shown in Figure 5.11.

**Network.** The input resolution is normalized to 256×256. Before the CU-Net, a Conv7 × 7 filter with stride 2 and a max pooling would produce 128 features with

resolution 64×64. Hence, the maximum resolution of CU-Net is 64×64. Each block in CU-Net has a bottleneck structure as shown on the right side of Figure 5.1. At the beginning of each bottleneck, features from different connections are concatenated and stored in shared memory. Then the concatenated features are compressed by the Conv1 $\times$ 1 to $4m$ features. At last, the Conv3 $\times$ 3 further produces $m$ new features. The batch norm and ReLU are used before the convolutions.

**Training.** We implement the CU-Net using the PyTorch. The CU-Net is trained by the optimizer RMSprop. When training human pose estimators, the initial learning rate is $2.5 \times 10^{-4}$ which is decayed to $5 \times 10^{-5}$ after 100 epochs. The whole training takes 200 epochs. The facial landmark localizers are easier to train. Also starting from $2.5 \times 10^{-4}$, its learning rate is divided by 5, 2 and 2 at epoch 30, 60 and 90 respectively. The above settings remain the same for quantized CU-Net. To match the pace of dataflow, we set the same bit-width for gradients and features. We quantize dataflows and parameters all over the CU-Net except for the first and last convolutional layers, since localization is a fine-grained task requiring high precision heatmaps.

**Human Pose Datasets.** We use two benchmark human pose estimation datasets: MPII Human Pose [91] and Leeds Sports Pose (LSP) [92]. The **MPII** is collected from YouTube videos with a broad range of human activities. It has 25K images and 40K annotated persons, which are split into a training set of 29K and a test set of 11K. Following [76], 3K samples are chosen from the training set as the validation set. Each person has 16 labeled joints. The **LSP** dataset contains images from many sports scenes. Its extended version has 11K training samples and 1K testing samples. Each person in LSP has 14 labeled joints. Since there are usually multiple people in one image, we crop around each person and resize the image to 256x256. We also use scaling (0.75-1.25), rotation (-/+30) and random flip to augment the data.

**Facial Landmark Datasets.** The experiments of the facial landmark localization are conducted on the composite of HELEN, AFW, LFPW, and IBUG which are re-annotated in the 300-W challenge [115]. Each face has 68 landmarks. Following [162] and [107], we use the training images of HELEN, LFPW and all images of AFW, 3148 images total, as the training set. The testing is done on the common subset

(testing images of HELEN and LFPW), challenge subset (all images from IBUG) and their union. We use the provided bounding boxes from the 300-W challenge to crop faces. The same augmentations of scaling and rotation as in human pose estimation are applied.

**Metric.** We use the standard metrics in both human pose estimation and face alignment. Specifically, Percentage of Correct Keypoints (PCK) is used to evaluate approaches for human pose estimation. A human joint is correctly detected if the L2 distance between the detected and groundtruth points is within a certain threshold. The MPII and LSP datasets use 50% of the head segment length and 20% of the L2 distance between the left shoulder and right hip as their thresholds. And the normalized mean error (NME) is employed to measure localizing facial landmarks. It measures the normalized L2 distance between the predicted and groundtruth landmarks. Following the convention of 300-W challenge, we use the inter-ocular distance to normalize mean error. For network quantization, we propose the balance index to balance accuracy and efficiency.

### 5.5.1 Hyper-Parameter Selection

There are two important hyper-parameters in designing the CU-Net. One is the feature number $m$ in the main feature stream. In the experiments, $m$ remains the same when the feature map resolution changes. The other hyper-parameter is the generated feature number $n$ in a block of U-Net. We have tried 6 combinations of $m$ and $n$ in 2 coupled U-Nets. Table 5.1 gives the PCKhs on the MPII validation set. Besides, we choose 4 from the 6 settings and show how their validation PCKhs change during the training process in Figure 5.7.

In Table 5.1, the smallest $m$ and $n$ are 64 and 16. We set the increments 64 and 8 for $m$ and $n$. We can observe how accuracy (PCKh) and parameter number change along with the two hyper-parameters. First, when $m$ and $n$ grow, the accuracy increase slows down from left to right (2.6%, 1.4%, 0.4%, 0.3% and 0.3%). Similar phenomena could be observed in Figure 5.7. Training is more stable when $m$ and $n$ become larger according to the curves in Figure 5.7. In contrast to accuracy, parameter number

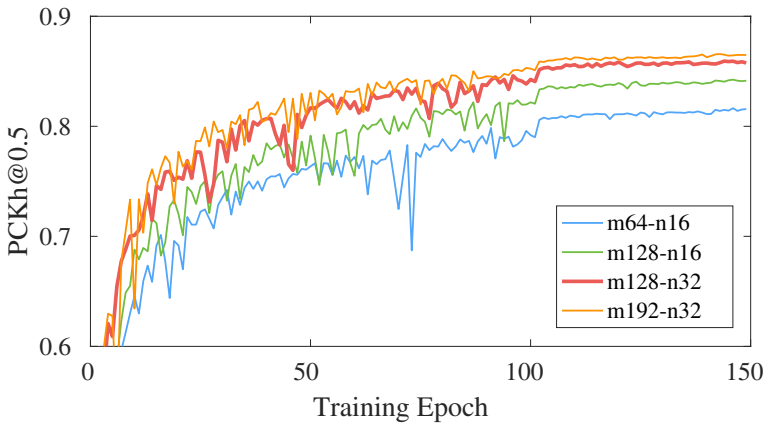



Figure 5.7: Validation PCKh curves of 2 coupled U-Nets(CU-Net-2) under different hyper-parameters $m$ and $n$. The converged curve reaches higher for larger $m$ and $n$. But the gap between adjacent curves becomes smaller. $m = 128$ and $n = 32$ is a good trade-off of accuracy and efficiency. Besides, Larger $m$ and $n$ also make the curve smoother.

Table 5.1: Comparison of different hyper-parameters $m$ and $n$ measured by the parameter number and the PCKh on the MPII validation set. We use 2 coupled U-Nets(CU-Net-2). The PCKh increase becomes less from the left to the right while the parameter number growly consistently. A good trade-off between the PCKh and parameter number is $m$=128 and $n$=32.

| $m$ | 64 | 128 | 128 | 128 | 192 | 192 |
|---|---|---|---|---|---|---|
| $n$ | 16 | 16 | 24 | 32 | 24 | 32 |
| # Parameters | 0.5M | 1.0M | 1.4M | 1.9M | 2.4M | 2.9M |
| PCKh@0.5 (%) | 81.6 | 84.2 | 85.6 | 86.0 | 86.3 | 86.6 |

grows consistently (0.5M, 0.4M, 0.5M, 0.5M and 0.5M), along with increasing $m$ and $n$. Through balancing the accuracy and parameter number, we choose $m$=128 and $n$=32. We fix this setting in the following experiments.

### 5.5.2 Evaluation of Intermediate Supervisions

Generally, the supervision of a CU-Net is the supervision of its last U-Net. Since a CU-Net contains several U-Nets, we consider adding supervisions for preceding U-Nets. More specifically, we only add the supervision at the end of a U-Net. Fortunately, the coupling connections do not prevent us from doing this. Note that if the supervision number is smaller than the U-Net number, we distribute the supervisions as uniformly

Table 5.2: PCKhs of the CU-Net with varied intermediate supervisions on the MPII validation set. CU-Net-2 denotes a CU-Net with 2 U-Nets. The intermediate supervisions lower the PCKh of CU-Net-2. However, it improves the PCKh of deeper networks CU-Net-4 and CU-Net-8. Deeper CU-Net requires more intermediate supervisions to get the highest PCKh. But full intermediate supervisions are not optimal.

| | CU-Net-2 | | CU-Net-4 | | | | CU-Net-8 | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| # Supervisions | 1 | 2 | 1 | 2 | 3 | 4 | 1 | 2 | 4 | 8 |
| PCKh@0.5 (%) | 86.0 | 85.8 | 87.6 | 88.1 | 88.0 | 87.8 | 88.6 | 89.3 | 89.5 | 89.0 |

as possible. For example, if 2 supervisions exist in 4 coupled U-Nets, they are at the end of the second and fourth U-Nets.

Table 5.2 gives the PCKh comparison of CU-Net with different numbers of supervisions. For CU-Net-2, adding supervision for the first U-Net makes the validation PCKh drop by 0.2%. The coupling connections already strengthen the gradient propagation. The additional supervision makes the gradients too strong, leading to a little overfitting.

However, observations are different for more coupled U-Nets. According to Table 5.2, additional supervisions could improve the PCKh of 4 coupled U-Nets (CU-Net-4). The CU-Net-4 obtains the highest PCKh with 1 additional supervision. Similar results appear for the CU-Net-8. But 3 additional supervisions help obtain the highest PCKh. CU-Net-4 and CU-Net-8 are much deeper than the CU-Net-2. Some intermediate supervisions could alleviate the vanishing gradient problem, which help CU-Net get better convergences. The CU-Net-8 is twice deeper than the CU-Net-4, thereby benefiting from more additional intermediate supervisions.

Both the intermediate supervisions and the coupling connections help train better CU-Net models. However, to obtain the highest accuracy, their amounts should have a balance. If one increases, the other needs to decrease. Since the intermediate supervisions require very few extra parameters, we would like to use full intermediate supervisions. That is, each U-Net in the CU-Net has one supervision. In this way, we could cut off some coupling connections to further reduce some parameters.

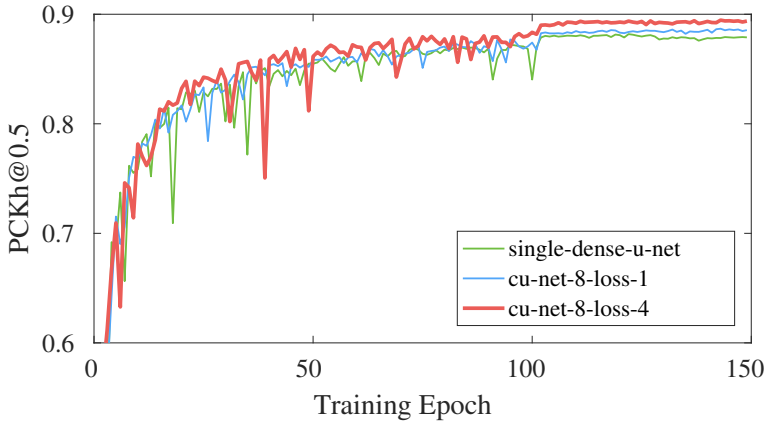



Figure 5.8: Validation PCKh curves of single dense U-Net and 8 coupled U-Nets (CU-Net-8) with 1 and 4 supervisions. They have equivalent amounts of parameters. The CU-Net-8 get higher converged PCKh than the single dense U-Net. The additional intermediate supervisions bring more PCKh gains. But its curve fluctuates more before the convergence.

Table 5.3: CU-Net *v.s.* single dense U-Net on MPII validation set measured by PCKh(%) and parameter #. The ratio is parameter # divided by the corresponding baselines(stacked 4 and 8 U-Nets). The CU-Net can get higher PCKh than the dense U-Net given a few more parameters.

| Method | PCKh | # Para. | Baseline | Ratio |
|---|---|---|---|---|
| Dense U-Net (8) | 88.1 | 6.1M | 25.5M | 23.9% |
| CU-Net-8 | 89.5 | 7.9M | 25.5M | 31.0% |
| Dense U-Net (4) | 87.1 | 2.9M | 12.9M | 22.5% |
| CU-Net-4 | 88.1 | 3.9M | 12.9M | 30.2% |

### 5.5.3 CU-Net *v.s.* Single Dense U-Net

There are two ways of adding the shortcut connections in the U-Net. One is adding dense connections in each block of a single U-Net, resulting in the single dense U-Net. The other is using the coupling connections in stacked U-Nets. Table 5.3 compares the PCKh and parameter number of CU-Net and single dense U-Net. For a fair comparison, the single dense U-Net and the CU-Net have the same number of conv3×3 layers. We add one layer in each dense block of the dense U-Net every time we increase one U-Net in the CU-Net.

According to Table 5.3, the CU-Net obviously outperforms the dense U-Net by 1.4% and 1.0% for the 4 and 8 U-Nets. We use the parameter numbers of stacked 4 and 8 U-Nets as the baselines since the main goal is to reduce the parameters of stacked U-Nets.

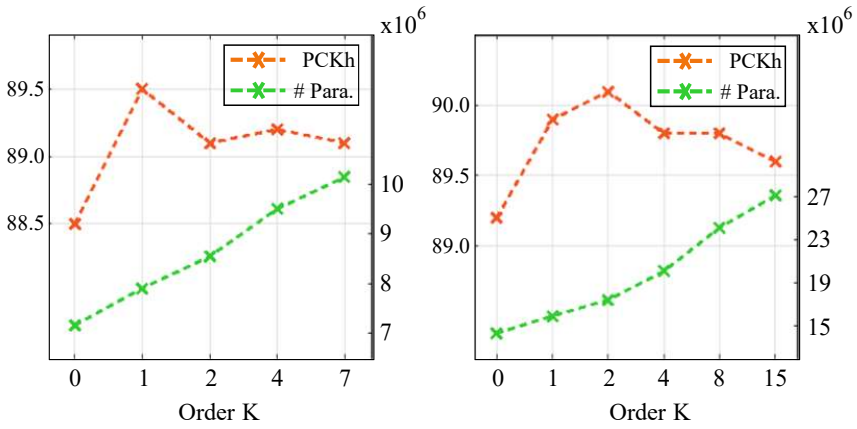



Figure 5.9: Relation of PCKh(%), # parameters and *order-K* coupling on MPII validation set. The parameter number of CU-Net grows approximately linearly with the order of coupling. However, the PCKh first increases and then decreases. A small order 1 or 2 would be a good balance for prediction accuracy and parameter efficiency.

Although the CU-Net has a few more parameters than the dense U-Net, the CU-Net is cost-effective. For example, the dense U-Net (8) and CU-Net-4 both increase 1.0% over the dense U-Net (4). However, they have 2.1× and 1.3× parameters of the dense U-Net (4).

We also show the validation PCKh curves of the 8 U-Nets setting in Figure 5.8. The converged PCKh curves of CU-Net and single dense U-Net have gaps. Besides, the CU-Net PCKh curve fluctuates more when adding the intermediate supervisions. Because additional supervisions can make the CU-Net parameters updated with larger steps in the training.

### 5.5.4 Evaluation of *Order-K* Coupling

The *order-K* coupling couples a U-Net only to its $K$ successors. Each U-Net has its own independent supervision. In this experiment, we investigate how the PCKh and convolution parameter number change along with the order value. Figure 5.9 gives the results from MPII validation set. The left and right figures show results of CU-Net with 8 and 16 U-Nets. It is clear that the convolution parameter number increases as the order becomes larger. However, the left and right PCKh curves have a similar shape of first increasing and then decreasing. The *order*-1 coupling is always better than the

Table 5.4: *Order*-1 CU-Net-8 *v.s. order*-7 CU-Net-8, measured by training and validation PCKhs(%) on MPII. *Order*-7 has higher training PCKh in all epochs. However, its validation PCKh is lower at last. Thus, *order*-7 with full intermediate supervisions overfits the training set a little bit.

| PCKh on training set | | | | |
|---|---|---|---|---|
| Epoch | 1 | 50 | 100 | 150 |
| *Order*-1 CU-Net-8 | 20.3 | 83.2 | 87.7 | 91.7 |
| *Order*-7 CU-Net-8 | **25.2** | **84.7** | **89.3** | **93.1** |
| PCKh on validation set | | | | |
| Epoch | 1 | 50 | 100 | 150 |
| *Order*-1 CU-Net-8 | 29.4 | 82.8 | **85.7** | **87.1** |
| *Order*-7 CU-Net-8 | **36.6** | **84.0** | 85.1 | 86.7 |

Table 5.5: *Order*-1 CU-Net *v.s.* stacked residual U-Nets on MPII validation set measured by PCKh(%), parameter number, and inference time. With the same number of U-Nets, *Order*-1 CU-Net achieves comparable performance as stacked U-Nets. But it has only ∼30% parameters. The inference time is reduced by ∼30%, benefiting from fewer parameters.

| Method | PCKh | # Para. | Ratio | Time(ms) | Ratio |
|---|---|---|---|---|---|
| Stacked U-Nets (16) | - | 50.5M | 100% | 104.8 | 100% |
| CU-Net-16 | 89.9 | 15.9M | 31.5% | 70.8 | 67.6% |
| Stacked U-Nets (8) | 89.3 | 25.5M | 100% | 48.9 | 100% |
| CU-Net-8 | 89.5 | 7.9M | 31.0% | 36.1 | 73.8% |
| Stacked U-Nets (4) | 88.3 | 12.9M | 100% | 28.2 | 100% |
| CU-Net-4 | 88.2 | 3.9M | 30.2% | 18.9 | 67.0% |

*order*-0 one.

However, high order may not be a good choice because we already use full intermediate supervisions. The balance of coupling and intermediate supervisions is broken for the high order ones. Too dense coupling make gradients accumulate too much, causing overfitting. Further evidence of overfitting is shown in Table 5.4. The *order*-7 coupling has the higher training PCKh than *order*-1 in all training epochs, but its validation PCKh is a little lower in the last training epochs. Thus, we use *order*-1 coupling with full intermediate supervisions in the following experiments.

### 5.5.5 CU-Net *v.s.* Stacked Residual U-Nets

The CU-Net is proposed to improve the parameter efficiency of stacked U-Nets with the residual modules. To validate this, we compare the *order*-1 CU-Net with the stacked

residual U-Nets [1]. This experiment is done on the MPII validation set. Table 5.5 shows three pairs of comparisons with 4, 8 and 16 U-Nets. The PCKh, convolution parameter number and inference time are reported. The inference time refers to the time of forwarding one image under the testing mode. We compute the average inference time on the MPII validation set. For a fair comparison, we test all the models using the Torch toolbox and K40 GPU.

In Table 5.5, with the same number of U-Nets, *order*-1 CU-Net could obtain comparable or even better accuracy. More importantly, feature reuse across U-Nets make each U-Net in CU-Net light-weighted: parameter number and inference time of stacked residual U-Nets decrease by $\sim$70% and $\sim$30%, respectively. In addition, the high parameter efficiency makes it possible to train *order*-1 CU-Net-16 in a 12G GPU with batch size 16.

### 5.5.6 Evaluation of Iterative Refinement

Iterative refinement is designed to reduce the parameters of CU-Net by one half. We validate the design in two steps. First, we verify that adding an iteration on a CU-Net could improve the accuracy. The experiment is done on the 300-W dataset using *order*-1 CU-Net-4. Results are shown in Table 5.6. For both detection and regression supervisions, adding an iteration could lower the localization errors, demonstrating the effectiveness of iterative refinement. Meanwhile, the model parameters only increase 0.2M. Besides, the regression supervision outperforms the detection one no matter in the iterative or non-iterative setting, making it a better choice for the landmark localization.

The regression and detection supervisions have different groundtruths. The regression supervision draws a gaussian circle centered at a keypoint coordinate. The detection supervision needs to strictly divide a groundtruth heatmap into two areas: keypoint area and non-keypoint area. In practice, it is hard to specify an accurate area for a keypoint. The Gaussian groundtruth can alleviate this concern by decreasing the confidence scores gradually. We think the advantage of regression supervision comes from its more reasonable groundtruth heatmaps.

Table 5.6: NME(%) on 300-W using *order*-1 CU-Net-4 with iterative refinement, detection, and regression supervisions. Iterative refinement can lower errors and regression supervision outperforms detection supervision.

| Method | Easy Subset | Hard Subset | Full Set | # Para. |
|---|---|---|---|---|
| Detection only | 3.63 | 5.60 | 4.01 | 3.9M |
| Regression only | 2.91 | 5.12 | 3.34 | 3.9M |
| Detection Detection | 3.52 | 5.59 | 3.93 | 4.1M |
| Detection Regression | 2.95 | 5.12 | 3.37 | 4.1M |
| Regression Regression | 2.87 | 4.97 | 3.28 | 4.1M |

Table 5.7: Iterative *order*-1 CU-Net-4 *v.s.* non-iterative *order*-1 CU-Net-8 on 300-W measured by NME(%). Iterative CU-Net-4, with few additional parameters on CU-Net-4, achieves comparable performance as CU-Net-8. Thus, the iterative refinement has the potential to halve parameters of CU-Net but still maintain comparable performance.

| Method | Easy Subset | Hard Subset | Full Set | # Parameters |
|---|---|---|---|---|
| CU-Net-4 | 2.91 | 5.12 | 3.34 | 3.9M |
| Iter. CU-Net-4 | 2.87 | 4.97 | 3.28 | 4.1M |
| CU-Net-8 | 2.82 | 5.07 | 3.26 | 7.9M |

Second, we prove an iterative CU-Net could get comparable accuracy as a double-length CU-Net. More specifically, we compare iterative *order*-1 CU-Net-4 with *order*-1 CU-Net-8. Table 5.7 gives the comparison. We can find that the iterative CU-Net-4 could obtain comparable NME as CU-Net-8. However, the CU-Net-8 has double parameters of CU-Net-4, whereas the iterative CU-Net-4 has only 0.2M additional parameters.

### 5.5.7 Evaluation of Parameter and Dataflow Quantization

In this experiment, we study quantizing the parameters and dataflow, i.e., intermediate features and gradients. Through network quantization, high precision parameters and operations can be efficiently represented by a few discrete values. We try a series of bit-widths to find appropriate choices. We use the *order*-1 CU-Net-4 on the 300-W dataset and the *order*-1 CU-Net-2 on the MPII validation set. The results are shown in Tables 5.8, 5.9 and 5.10. BW and TW represent binarized weight and ternarized weight respectively. The suffixes $\alpha$ and QIG denote the float scaling factor $\alpha$ and quantized

Table 5.8: Performance of different combinations of bit-width values on the 300-W dataset measured by NME(%). All quantized networks are based on *order*-1 CU-Net-4. BW and TW are short for binarized and ternarized parameters, $\alpha$ represents float scaling factor, QFG is short for quantized intermediate features and gradients. $Bit_F$, $Bit_P$, $Bit_G$ represents the bit-width of features, parameters, gradients respectively. Training memory and model size are represented by their compression ratios to those of the original CU-Net-4. The balance index is calculated by Equation 5.8. The CU-Net-4-BW-$\alpha$ gets the lowest error. Considering together accuracy, training memory and model size, the CU-Net-4-BW-$\alpha$(818) has the smallest balance index.

| Method | $Bit_I$ | $Bit_W$ | $Bit_G$ | NME(%) Full set | NME(%) Easy set | NME(%) Hard set | Training Memory | Model Size | Balance Index |
|---|---|---|---|---|---|---|---|---|---|
| CU-Net-4 | 32 | 32 | 32 | 3.38 | 2.95 | 5.13 | 1.00 | 1.00 | 11.4 |
| BW-QIG | 6 | 1 | 6 | 5.93 | 5.10 | 9.34 | 0.17 | 0.03 | 0.18 |
| BW-QIG | 8 | 1 | 8 | 4.30 | 3.67 | 6.86 | 0.25 | 0.03 | **0.14** |
| BW-$\alpha$-QIG | 8 | 1 | 8 | 4.47 | 3.75 | 7.40 | 0.25 | 0.03 | 0.15 |
| BW | 32 | 1 | 32 | 3.75 | 3.20 | 5.99 | 1.00 | 0.03 | 0.42 |
| BW-$\alpha$ | 32 | 1 | 32 | **3.58** | 3.12 | 5.45 | 1.00 | 0.03 | 0.38 |
| TW | 32 | 2 | 32 | 3.73 | 3.21 | 5.85 | 1.00 | 0.06 | 0.83 |
| TW-QIG | 6 | 2 | 6 | 4.27 | 3.70 | 6.59 | 0.17 | 0.06 | 0.19 |
| TW-QIG | 8 | 2 | 8 | 4.13 | 3.55 | 6.50 | 0.25 | 0.06 | 0.26 |

intermediate features and gradients.

**Binary Parameters.** According to Tables 5.8 and 5.9, the binarized parameters with the scaling factor $\alpha$ achieves PCKh 85.5% in human pose estimation and NME 3.58% in facial landmark localization. They are very close to the original 86.1% and 3.38%, without any quantization. Even without the scaling factor $\alpha$, the decrease of PCKh and increase of NME are small. This indicates binarizing the CU-Net parameters does not affect much its localization accuracy. However, the model size is substantially decreased by 32×.

**Ternary Parameters.** Ternary representation has one more bit than the binary one. Its stronger representation power could improve the accuracy of CU-Net. Based on Table 5.8, ternary parameters reduces the NME of binary parameters by 0.02%. With the quantization of features and gradients, we can observe more obvious NME decreases 1.66% and 0.17%. Consistent changes on the PCKh can be found in Table 5.9.

Table 5.9: Performance of different quantization configurations for *order*-1 CU-Net-2 on the MPII validation dataset measured by PCKh(%), training memory, model size, and balance index. The CU-Net-2-BW-$\alpha$ gets the highest accuracy. Considering together accuracy, training memory and model size, the CU-Net-2-BW-$\alpha$(818) has the smallest balance index.

| Method | $Bit_F$ | $Bit_P$ | $Bit_G$ | PCKh (%) | Training Memory | Model Size | Balance Index |
|---|---|---|---|---|---|---|---|
| CU-Net-2 | 32 | 32 | 32 | 86.1 | 1.00 | 1.00 | 193 |
| BW-QFG | 6 | 1 | 6 | 62.7 | 0.17 | 0.03 | 7.10 |
| BW-QFG | 8 | 1 | 8 | 81.5 | 0.25 | 0.03 | **2.57** |
| BW | 32 | 1 | 32 | 84.7 | 1.00 | 0.03 | 6.84 |
| BW-$\alpha$ | 32 | 1 | 32 | **85.5** | 1.00 | 0.03 | 7.97 |
| TW | 32 | 2 | 32 | 84.9 | 1.00 | 0.06 | 14.0 |
| TW-QFG | 6 | 2 | 6 | 74.0 | 0.17 | 0.06 | 6.90 |
| TW-QFG | 8 | 2 | 8 | 81.7 | 0.25 | 0.06 | 5.02 |

**Features and Gradients Quantization.** Quantizing the intermediate features and gradients of CU-Net could substantially reduce training memory, improving training efficiency. We try 6-bit and 8-bit with either the binary or ternary parameters. The 8-bit quantization is obviously better than the 6-bits one, especially for the binary parameters. The 8-bit quantization with binary parameters decreases the PCKh by 4.6% and increases the NME by 0.92%. However, the training memory is significantly reduced (75%). For mobile devices with limited computational resources, slightly performance drop is tolerable provided the large efficiency enhancement.

**Balancing Accuracy and Quantization.** The quantization of parameters and dataflow could significantly increase the testing and training efficiency but at the cost of some accuracy decrease. Thus, we need a trade-off between them. To this end, we propose the balance index (BI) in Equation 5.8 to balance the quantization and accuracy.

$$BI = E^2 \cdot TM \cdot MS \tag{5.8}$$

where $TM$ and $MS$ are short for training memory and model size. Instead of using their raw values, we use their ratios to those without any quantization. $E$ denotes the error in the landmark localization. We set $E$ as the NME for facial landmark localization

Table 5.10: Detailed PCKh comparison of different quantization configurations for *order*-1 CU-Net-2 on MPII validation sets. The parameter binarization or ternarization have small influence on the accuracy of individual human joints. But the quantization of intermediate features and gradients lowers the accuracy of challenging human joints: elbow, wrist, ankle and knee.

| Method | Head | Sho. | Elb. | Wri. | Hip | Knee | Ank. | Mean |
|---|---|---|---|---|---|---|---|---|
| CU-Net-2 | 96.1 | 94.5 | 86.7 | 81.1 | 86.9 | 81.0 | 76.3 | 86.1 |
| CU-Net-2+BW-$\alpha$ | 95.9 | 94.4 | 86.6 | 80.3 | 86.5 | 79.5 | 75.2 | 85.5 |
| CU-Net-2+BW | 96.3 | 94.2 | 85.2 | 79.1 | 85.8 | 78.2 | 74.3 | 84.7 |
| CU-Net-2+BW-QIG(818) | 95.6 | 92.4 | 82.0 | 74.7 | 82.6 | 74.8 | 68.6 | 81.5 |
| CU-Net-2+BW-QIG(616) | 87.7 | 77.4 | 61.8 | 49.8 | 64.2 | 50.8 | 47.2 | 62.7 |
| CU-Net-2+TW | 96.2 | 93.9 | 85.7 | 79.6 | 85.8 | 79.1 | 74.1 | 84.9 |
| CU-Net-2+TW-QIG(828) | 95.4 | 92.6 | 82.1 | 75.2 | 83.0 | 74.9 | 68.7 | 81.7 |
| CU-Net-2+TW-QIG(626) | 94.2 | 87.2 | 73.4 | 65.1 | 76.0 | 64.7 | 57.6 | 74.0 |

and 1-PCKh for human pose estimation. The $E^2$ is calculated in the above formula to emphasize the prior importance of accuracy. Smaller BI indicates better balance.

In Tables 5.8 and 5.9, BW-$\alpha$-QIG(818) gets the smallest BI. Its NME increases by only 0.92% and its PCKh decreases by only 4.6%. However, it reduces training memory 4× and model size 32×, which has the best balance for accuracy and efficiency.

**Quantization Impact on Individual Human Joints.** In addition to the average PCKhs In Table 5.9, we also give the PCKhs of individual human joints under various quantization settings in Table 5.10. The parameter binarization does not affect much the joint accuracy. However, the quantization of intermediate features and gradients causes obvious decreases of challenging joints like wrist, knee, and ankle. This means that, although the parameter quantization does not lose much useful information, the parameter update still requires high precision representations. A possible solution is to explore better input and gradient quantization strategies.

### 5.5.8 Evaluation of Memory Efficient Implementation

Memory-efficient implementation makes it possible to train very deep CU-Net. Figure 5.10 shows the training memory consumption of both naive and memory-efficient implementations of *order*-1 CU-Net. The linear growths of training memory along with number of U-Nets is because of the fixed order coupling. But the memory growth of

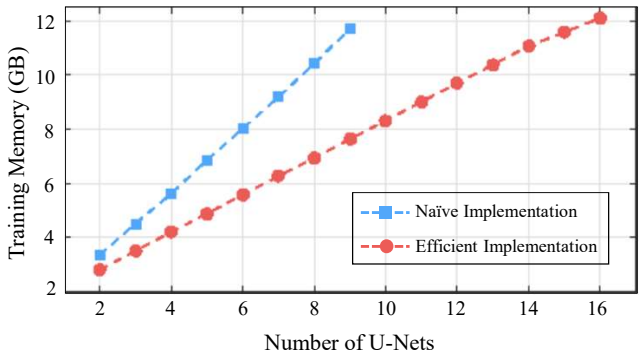



Figure 5.10: Naive implementation *vs.* memory-efficient implementation. The *order*-1 coupling, batch size 16 and a 12GB GPU are used. The naive implementation can only support 9 U-Nets at most. In contrast, the memory-efficient implementation allows training 16 U-Nets, which nearly doubles the depth of CU-Net.

Table 5.11: Comparison of convolution parameter number (Million), model size (Megabyte) and inference time (millisecond) with state-of-the-art methods. CU-Net-8 can significantly reduce parameter number ∼69%-∼86% and inference time ∼26%-∼86%.

| Method | Yang *et al.*[87] | Wei *et al.*[82] | Chu *et al.*[86] | Newell *et al.*[1] | *Order*-1 CU-Net-8 | *Order*-1 CU-Net-16 |
|---|---|---|---|---|---|---|
| # Parameters (M) | 28.0 | 29.7 | 58.1 | 25.5 | **7.9** | 15.9 |
| Inference Time (ms) | 137.7 | 112.4 | 251.0 | 48.9 | **36.1** | 70.8 |

efficient implementation is much slower than that of the naive one. With batch size 16, we could train the CU-Net-16 in one 12GB GPU. Under the same setting, the naive implementation could support only CU-Net-9.

### 5.5.9 Comparison with State-of-the-art Methods

We compare the CU-Net with state-of-the-art approaches for both human pose estimation and facial landmark localization. Both the efficiency and accuracy are compared.

**Efficiency.** Table 5.11 compares the CU-Net with state-of-the-art methods in terms of efficiency. The CU-Net-8 has only ∼14%-∼31% parameter number of them. Fewer parameters can usually accelerate the inference speed. According to Table 5.11, the CU-Net-8 uses ∼74% time of Newell *et. al* [1] (stacked 8 U-Nets). Yang *et. al* [87] and Chu *et. al* [86] use more sophisticated modules and graphical models based on Newell *et. al* [1], resulting in much higher time cost. Wei *et. al* [82] also has high time expense

Table 5.12: NME(%) comparison with state-of-the-art facial landmark localization methods on 300-W dataset. The CU-Net-BW-$\alpha$ refers to the CU-Net with binarized parameters and scaling factor $\alpha$. It obtains comparable error with state-of-the-art method [1]. But it has $\sim$**50**$\times$ smaller model size.

| Method | CFAN [158] | Deep Reg [163] | CFSS [162] | TCDCN [106] | DDN [164] | MDM [165] | TSR [107] | HGs(4) [1] | *Order*-1 CU-Net-8 | *Order*-1 CU-Net-8-BW-$\alpha$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Easy subset | 5.50 | 4.51 | 4.73 | 4.80 | - | 4.83 | 4.36 | 2.90 | **2.82** | 3.00 |
| Hard subset | 16.78 | 13.80 | 9.98 | 8.60 | - | 10.14 | 7.56 | 5.15 | **5.07** | 5.36 |
| Full set | 7.69 | 6.31 | 5.76 | 5.54 | 5.59 | 5.88 | 4.99 | 3.35 | **3.26** | 3.46 |

mainly due to its larger input resolution and convolution kernel size. The CU-Net-16 consumes more time than Newell *et. al* [1], albeit its fewer parameters. Because the CU-Net-16 has double depth of Newell *et. al* [1]. However, the CU-Net-16 uses only $\sim$28%-$\sim$63% time of other more complex methods.

Moreover, network quantization can also improve model efficiency. For example, the parameter binarization can reduce the model size by $\sim$32$\times$. The parameter binarization and feature quantization can bring $\sim$2$\times$-$\sim$58$\times$ speedup according to the theoretical analysis of [153]. With the binarized parameters, the convolutions only have the addition and subtraction without the multiplication operations, resulting in $\sim$2$\times$ speedup. If the features are also binarized, it can achieve $\sim$58$\times$ speedup with the bit-counting operations [149]. Generally, measuring the inference time of the quantized networks requires special software and hardware supports. We only provide a theoretical analysis here due to our resource limitations.

**Accuracy.** Tables 5.12, 5.13 and 5.14 show accuracy comparisons on both facial landmark localization and human pose estimation. Despite its high efficiency, the CU-Net can still achieve obtain comparable state-of-the-art accuracy on the benchmark 300-W, MPII and LSP test sets.

## 5.6 Summary

We presented the efficient CU-Net design that connects blocks with the same semantic meanings in stacked U-Nets. *Order-K* coupling and iterative refinement are introduced

Table 5.13: PCKh(%) comparison on MPII test sets. The CU-Net-BW-$\alpha$ refers to the CU-Net with binary parameters and scaling factor $\alpha$. The *order*-1 CU-Net-16-BW-$\alpha$ could achieve comparable accuracy. More importantly, it has $\sim$**2%** model size compared with other state-of-the-art approaches.

| Method | Head | Sho. | Elb. | Wri. | Hip | Knee | Ank. | Mean |
|---|---|---|---|---|---|---|---|---|
| Pishchulin *et al.* [95] | 74.3 | 49.0 | 40.8 | 34.1 | 36.5 | 34.4 | 35.2 | 44.1 |
| Tompson *et al.* [84] | 95.8 | 90.3 | 80.5 | 74.3 | 77.6 | 69.7 | 62.8 | 79.6 |
| Carreira *et al.* [75] | 95.7 | 91.7 | 81.7 | 72.4 | 82.8 | 73.2 | 66.4 | 81.3 |
| Tompson *et al.* [76] | 96.1 | 91.9 | 83.9 | 77.8 | 80.9 | 72.3 | 64.8 | 82.0 |
| Hu *et al.* [77] | 95.0 | 91.6 | 83.0 | 76.6 | 81.9 | 74.5 | 69.5 | 82.4 |
| Pishchulin *et al.* [78] | 94.1 | 90.2 | 83.4 | 77.3 | 82.6 | 75.7 | 68.6 | 82.4 |
| Lifshitz *et al.* [79] | 97.8 | 93.3 | 85.7 | 80.4 | 85.3 | 76.6 | 70.2 | 85.0 |
| Gkioxary *et al.* [80] | 96.2 | 93.1 | 86.7 | 82.1 | 85.2 | 81.4 | 74.1 | 86.1 |
| Rafi *et al.* [96] | 97.2 | 93.9 | 86.4 | 81.3 | 86.8 | 80.6 | 73.4 | 86.3 |
| Belagiannis *et al.* [97] | 97.7 | 95.0 | 88.2 | 83.0 | 87.9 | 82.6 | 78.4 | 88.1 |
| Insafutdinov *et al.* [81] | 96.8 | 95.2 | 89.3 | 84.4 | 88.4 | 83.4 | 78.0 | 88.5 |
| Wei *et al.* [82] | 97.8 | 95.0 | 88.7 | 84.0 | 88.4 | 82.8 | 79.4 | 88.5 |
| Bulat *et al.* [83] | 97.9 | 95.1 | 89.9 | 85.3 | 89.4 | 85.7 | 81.7 | 89.7 |
| Newell *et al.* [1] | 98.2 | 96.3 | 91.2 | 87.1 | 90.1 | 87.4 | 83.6 | 90.9 |
| Chu *et al.* [86] | **98.5** | 96.3 | 91.9 | **88.1** | **90.6** | **88.0** | **85.0** | **91.5** |
| *Order*-1 CU-Net-8 | 97.4 | 96.2 | 91.8 | 87.3 | 90.0 | 87.0 | 83.3 | 90.8 |
| *Order*-1 CU-Net-16 | 97.4 | **96.4** | **92.1** | 87.7 | 90.2 | 87.7 | 84.3 | 91.2 |
| +BW+$\alpha$ | 97.6 | 96.4 | 91.7 | 87.3 | 90.4 | 87.3 | 83.8 | 91.0 |

to further improve the parameter efficiency. We also study the quantization of parameters, intermediate features and gradients. Experiments on both human pose estimation and facical landmark localization show that the CU-Net could achieve comparable state-of-the-art accuracy but with only $\sim$30% parameters, $\sim$70% inference time, $\sim$2% model size and $\sim$25% training memory.

Table 5.14: PCK(%) comparison on LSP test set. The CU-Net-BW-$\alpha$ refers to the CU-Net with binary parameters and scaling factor $\alpha$. The *order*-1 CU-Net-16-BW-$\alpha$ could obtain comparable state-of-the-art accuracy. But it has $\sim$**50**$\times$ smaller model size than other state-of-the-art methods.

| Method | Head | Sho. | Elb. | Wri. | Hip | Knee | Ank. | Mean |
|---|---|---|---|---|---|---|---|---|
| Belagiannis *et al.* [97] | 95.2 | 89.0 | 81.5 | 77.0 | 83.7 | 87.0 | 82.8 | 85.2 |
| Lifshitz *et al.* [79] | 96.8 | 89.0 | 82.7 | 79.1 | 90.9 | 86.0 | 82.5 | 86.7 |
| Pishchulin *et al.*[78] | 97.0 | 91.0 | 83.8 | 78.1 | 91.0 | 86.7 | 82.0 | 87.1 |
| Insafutdinov *et al.* [81] | 97.4 | 92.7 | 87.5 | 84.4 | 91.5 | 89.9 | 87.2 | 90.1 |
| Wei *et al.* [82] | 97.8 | 92.5 | 87.0 | 83.9 | 91.5 | 90.8 | 89.9 | 90.5 |
| Bulat *et al.* [83] | 97.2 | 92.1 | 88.1 | 85.2 | 92.2 | 91.4 | 88.7 | 90.7 |
| Chu *et al.* [86] | 98.1 | 93.7 | 89.3 | 86.9 | 93.4 | 94.0 | 92.5 | 92.6 |
| Newell *et al.* [1] | 98.2 | 94.0 | 91.2 | 87.2 | 93.5 | **94.5** | 92.6 | 93.0 |
| Yang *et al.* [87] | **98.3** | 94.5 | 92.2 | 88.9 | **94.4** | 95.0 | 93.7 | 93.9 |
| *Order*-1 CU-Net-8 | 97.1 | 94.7 | 91.6 | 89.0 | 93.7 | 94.2 | 93.7 | 93.4 |
| *Order*-1 CU-Net-16 | 97.5 | **95.0** | **92.5** | **90.1** | 93.7 | **95.2** | 94.2 | **94.0** |
| +BW+$\alpha$ | 97.8 | 94.3 | 91.8 | 89.3 | 93.1 | 94.9 | **94.4** | 93.6 |

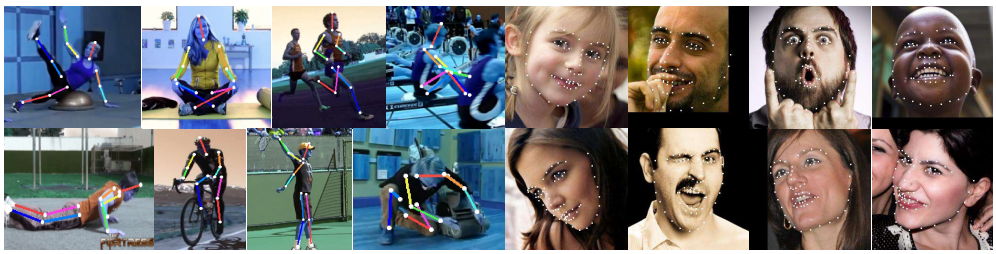


Figure 5.11: Qualitative results of human pose estimation and facial landmark localization. The quantized CU-Net could handle a wide range of human poses, even with occlusions. It could also detect accurate facial landmarks with various head poses and expressions.

# Chapter 6

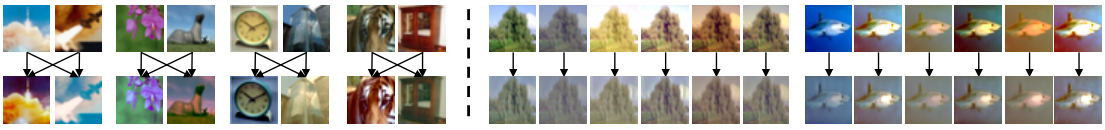
# SelfNorm and CrossNorm for Out-of-Distribution Robustness

## 6.1 Introduction

Normalization methods, e.g., Batch Normalization [159], Layer Normalization [166], and Instance Normalization [167], play a pivotal role in training deep neural networks. These methods generally try to make training more stable and convergence faster, assuming that training and test data come from the same distribution. However, few studies investigate normalization in improving OOD generalization in real-world scenarios. For example, image corruptions [11], e.g., snow and blur, can cause test data out of the clean training distribution. Moreover, training on synthetic data [44] to generalize to realistic data can significantly reduce the annotation burden. This work aims to encourage the interaction between normalization and OOD generalization. Specifically, we manipulate feature mean and variance to make models generalize better to out-of-distribution data.

Our inspiration comes from the observation that channel-wise mean and variance of feature maps carry some style information. For instance, exchanging the RGB means and variances between two instances can transfer style between them, as shown in Figure 6.1 (a). For many tasks such as CIFAR classification [24], the style encoded by channel-wise mean and variance is usually less critical in recognizing the object than other information such as object shape. Therefore, we propose CrossNorm which swaps channel-wise mean and variance of feature maps. CrossNorm can augment styles in training, making the model more robust to appearance changes.

Furthermore, given one image in different styles, we can reduce the style discrepancy when adjusting the RGB means and variances properly, as illustrated in Figure 6.1 (b). Intuitively, the style recalibration can reduce appearance variance, which may be useful



(a) Switch RGB mean and variance (b) Recalibrate RGB mean and variance

Figure 6.1: CIFAR examples of exchanging (**Left**) and adjusting (**Right**) RGB mean and variance.

in bridging distribution gaps between training and unforeseen testing data. To this end, we propose SelfNorm to use attention [168] to adjust channel-wise mean and variance automatically.

It is interesting to analyze the distinction and connection between CrossNorm and SelfNorm. At first glance, they take opposite actions (style augmentation v.s. style reduction). Even so, they use the same tool: channel-wise statistics and pursue the same goal: OOD robustness. Additionally, CrossNorm can increase the capacity of SelfNorm by style augmentation. SelfNorm, with the help from CrossNorm, can generalize better to OOD data.

**Concept and Intuition**. The style concept here refers to a family of weak cues associated with the semantic content of interest. For instance, the image style in object recognition can include many appearance-related factors such as color, contrast, and brightness. Style sometimes may help in decision-making, but the model should rely more on more vital content cues to become robust. To reduce its bias rather than discard it, we use CrossNorm with probability in training. The insight beneath CrossNorm is that each instance, or feature map, has its unique style. Further, style cues are not equally important. For example, the yellow color seems more useful than other style cues in recognizing orange. In light of this, the intuition behind SelfNorm is that attention may help emphasize essential styles and suppress trivial ones.

**Assumption**. Although we use the channel-wise mean and variance to modify styles, we do not assume that they are sufficient to represent all style cues. Better style representations are available with more complex statistics [169] or even style transfer models [170, 171]. We choose the first and second-order statistics mainly because they are simple, efficient to compute, and can connect normalization to out-of-distribution

generalization. In summary, the key contributions are:

- We propose SelfNorm and CrossNorm, two simple yet effective normalization techniques to enhance out-of-distribution generalization.
- SelfNorm and CrossNorm form a unity of opposites in using feature mean and variance for model robustness.
- They are domain agnostic and can advance state-of-the-art robustness performance for different domains (vision or language), settings (fully or semi-supervised), and tasks (classification and segmentation).

## 6.2 Related Work

**Out-of-distribution generalization**. Although the current deep models continue to break records on benchmark IID datasets, they still struggle to generalize to OOD data caused by common corruptions [11] and dataset gaps [44]. To improve the robustness against corruption, Stylized-ImageNet [172] conducts style augmentation to reduce the texture bias of CNNs. Compared to it, CrossNorm has two main advantages. First, CrossNorm is efficient as it transfer styles directly in the feature space of the target CNNs. However, Stylized-ImageNet relies on external style datasets and pre-trained style transfer models. Second, CrossNorm can advance the performance on both clean and corrupted data, while Stylized-ImageNet hurts clean generalization. In contrast to the consistent styles within one dataset, the external ones can result in massive distribution shifts. Recently, AugMix [173] trains robust models by mixing multiple augmented images based on random image primitives or image-to-image networks [174]. Adversarial noises training (ANT) [175] can also improve the robustness against corruption. CrossNorm is domain agnostic and orthogonal to AugMix and ANT, making it possible for their joint application. Moreover, unsupervised domain adaptation is also useful for corruption robustness in some situations [176].

Besides common corruptions, generalization with distribution gaps [44] across different datasets also suffers from problems. IBN [177] mixes instance and batch normalization to shrink the domain distances. SelfNorm can bridge the domain gaps by

style recalibration. Domain randomization [178] uses style augmentation for domain generalization on segmentation datasets. It suffers from the same issues of Stylized-ImageNet as it also uses pre-trained style transfer models and additional style datasets. By contrast, CrossNorm is more efficient and balances better between the source and target domains' performance. Beyond the vision field, many natural language processing (NLP) applications also face the out-of-distribution generalization challenges [179]. Benefiting from the domain-agnostic property, SelfNorm and CrossNorm can also improve model robustness in the NLP area.

**Normalization and attention**. Batch Normalization [159] is a milestone technique that inspires many following normalization methods such as Instance Normalization [167], Layer Normalization [166], and Group Normalization [180]. Recently, some works integrate attention [168] into feature normalization. Mode normalization [181] and attentive normalization [182] use attention to weigh a mixture of batch normalizations. Examplar normalization [183] learns to combine multi-type normalizations by attention. By contrast, SelfNorm uses attention with only instance normalization. More importantly, unlike previous normalization approaches, SelfNorm and CrossNorm aim to improve out-of-distribution generalization. In addition, SelfNorm is different from SE [168], though they use similar attention. First, SelfNorm learns to recalibrate channel-wise mean and variance instead of channel features in SE. Second, SE models the interdependency between channels, while SelfNorm deals with each channel independently. Also, a SelfNorm unit, with $O(n)$, is more lightweight than a SE one, of $O(n^2)$, where $n$ denotes the channel number. The difference analysis here also applies to the Channel Attention [184], similar to SE.

**Data augmentation**. Data augmentation is an important tool in training deep models. Current popular data augmentation techniques are either label-preserving [9, 8, 7] or label-perturbing [119, 185]. The label-preserving methods usually rely on domain-specific image primitives, e.g., rotation and color, making them inflexible for tasks beyond the vision domain. The label-perturbing techniques mainly work for classification and may have trouble in broader applications, e.g., segmentation. CrossNorm, as a data augmentation method, is readily applicable to diverse domains (vision and

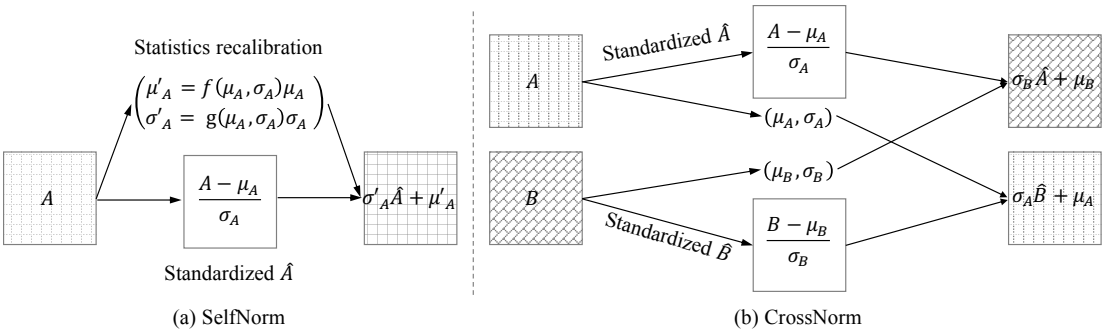



Figure 6.2: SelfNorm (**left**) and CrossNorm (**right**). SelfNorm uses attention to recalibrate the mean and variance of a feature map, while CrossNorm swaps the statistics between a pair of feature maps.

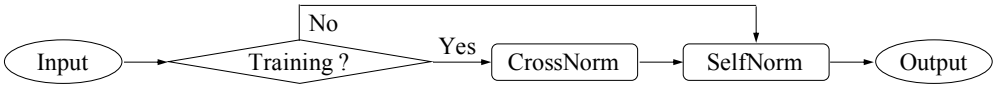



Figure 6.3: Flowchart for SelfNorm and CrossNorm. SelfNorm learns in training but functions in testing, while CrossNorm works in training.

language) and tasks (classification and segmentation). The goal of CrossNorm is to boost out-of-distribution generalization, which is also different from many former data augmentation methods.

## 6.3 SelfNorm and CrossNorm

**Background**. Technically, SelfNorm and CrossNorm share the same origin: instance normalization [167]. In 2D CNNs, each instance has $C$ feature maps of size $H \times W$. Given $\mathbf{A} \in R^{H \times W}$, instance normalization first normalizes the feature map and then conducts affine transformation:

$$\gamma \frac{\mathbf{A} - \mu_{\boldsymbol{A}}}{\sigma_{\mathbf{A}}} + \beta, \tag{6.1}$$

where $\mu_{\mathbf{A}}$ and $\sigma_{\mathbf{A}}$ are the mean and standard deviation; $\gamma$ and $\beta$ denotes learnable affine parameters. As shown in Figure 6.1 and also pointed out by the style transfer practices [186, 170, 171], $\mu_{\mathbf{A}}$ and $\sigma_{\mathbf{A}}$ can encode some style information.

**SelfNorm**. SelfNorm replaces $\beta$ and $\gamma$ with recalibrated mean $\mu'_{\mathbf{A}} = f(\mu_{\mathbf{A}}, \sigma_{\mathbf{A}})\mu_{\mathbf{A}}$ and standard deviation $\sigma'_{\mathbf{A}} = g(\mu_{\mathbf{A}}, \sigma_{\mathbf{A}})\sigma_{\mathbf{A}}$, as illustrated in Figure 6.2, where $f$ and $g$

are the attention functions. The adjusted channel becomes:

$$\sigma'_{\mathbf{A}} \frac{\mathbf{A} - \mu_{\mathbf{A}}}{\sigma_{\mathbf{A}}} + \mu'_{\mathbf{A}}. \tag{6.2}$$

As $f$ and $g$ learn to scale $\mu_{\mathbf{A}}$ and $\sigma_{\mathbf{A}}$ based on themselves, $\mathbf{A}$ *kind of normalizes itself by self-gating, hence SelfNorm.* SelfNorm is inspired by the fact that attention can help the model emphasize informative features and suppress less useful ones. In terms of recalibrating $\mu_{\mathbf{A}}$ and $\sigma_{\mathbf{A}}$, SelfNorm expects to highlight the discriminative styles and understate trivial ones. In practice, we use a fully connected (FC) network to wrap attention functions $f$ and $g$. The architecture is efficient as its input and output are both two scalars. Since each channel has its independent statistics, SelfNorm recalibrates each channel separately using $C$ lightweight FC networks, hence complexity $O(C)$.

**CrossNorm**. CrossNorm exchanges $\mu_{\mathbf{A}}$ and $\sigma_{\mathbf{A}}$ of channel $\mathbf{A}$ with $\mu_{\mathbf{B}}$ and $\sigma_{\mathbf{B}}$ of channel $\mathbf{B}$, i.e., changing $\beta$ and $\gamma$ to each other's $\mu$ and $\sigma$, shown in Figure 6.2:

$$\sigma_{\mathbf{B}} \frac{\mathbf{A} - \mu_{\mathbf{A}}}{\sigma_{\mathbf{A}}} + \mu_{\mathbf{B}} \qquad \sigma_{\mathbf{A}} \frac{\mathbf{B} - \mu_{\mathbf{B}}}{\sigma_{\mathbf{B}}} + \mu_{\mathbf{A}}, \tag{6.3}$$

where $\mathbf{A}$ *and* $\mathbf{B}$ *seem to normalize each other, hence CrossNorm.* CrossNorm is motivated by the key observation that a target dataset, such as a classification dataset, has rich, though subtle, styles. *Specifically, each instance, or even every channel, has its unique style.* Exchanging the statistics can perform efficient style augmentation, reducing the style bias in decision-making. In mini-batch training, we turn on CrossNorm with some probability.

**Unity of Opposites**. SelfNorm and CrossNorm both start from instance normalization but head in opposite directions. SelfNorm recalibrates statistics to focus on only necessary styles, reducing standardized features (zero-mean and unit-variance) and statistics mixtures' diversity. In contrast, CrossNorm transfers statistics between channels, enriching the combinations of standardized features and statistics. They perform opposite operations mainly because they target different stages. SelfNorm dedicates to style recalibration during testing, while CrossNorm functions only in training. Note that SelfNorm is a learnable module, requiring training to work. Figure 6.3 shows the flowchart of SelfNorm and CrossNorm. Additionally, SelfNorm helps make the model

less sensitive to appearance changes, while CrossNorm aims to lessen the model's style bias. Despite these differences, they both can facilitate out-of-distribution generalization. Further, CrossNorm can boost the performance of SelfNorm because its style augmentation can prevent SelfNorm from overfitting to specific styles. Overall, the two seemingly opposed methods form a unity of using normalization statistics to advance out-of-distribution robustness.

### 6.3.1 CrossNorm Variants

The core idea of CrossNorm is to swap mean and variance between channel features. For 2D CNNs, given one instance $\mathbf{X} \in R^{C\times H\times W}$, CrossNorm can exchange statistics between its $C$ channels:

$$\{(\mathbf{A},\mathbf{B}) \in (\mathbf{X}_{i,:,:},\mathbf{X}_{j,:,:}) \mid i \neq j, 0 < i,j < C\}, \tag{6.4}$$

where $\mathbf{A}$ and $\mathbf{B}$ refer to the channel pair in Equation 6.3. If two instances $\mathbf{X},\mathbf{Y} \in R^{C\times H\times W}$ given, CrossNorm can swap statistics between their corresponding channels, i.e., $\mathbf{A}$ and $\mathbf{B}$ become:

$$\{(\mathbf{A},\mathbf{B}) \in (\mathbf{X}_{i,:,:},\mathbf{Y}_{i,:,:}) \mid 0 < i < C\}. \tag{6.5}$$

Compared with one-instance CrossNorm, the two-instance one tends to consider instance-level style instead of channel-level.

Moreover, distinct spatial regions probably have different mean and variance statistics. To promote the style diversity, we propose to crop regions for CrossNorm:

$$\{(\mathbf{A},\mathbf{B}) \in (\mathrm{crop}(\mathbf{A}),\mathrm{crop}(\mathbf{B})) \mid r_{\mathrm{crop}} \geq t\} \tag{6.6}$$

where the crop function returns a square with area ratio $r$ no less than a threshold $t(0 < t \leq 1)$. The whole channel is a special case in cropping. There are three cropping choices: content only, style only, and both. For content cropping, we crop A only when we use its standardized feature map. In other words, no cropping applies to A when it provides its statistics to B. Cropping both means cropping A and B no matter we employ their standardized feature map or statistics. The cropping strategy can produce diverse styles for both the two-instance and one-instance CrossNorms.

### 6.3.2 Modular Design

SelfNorm and CrossNorm can naturally work in the feature space, making it flexible to plug them into many network locations. Two questions come: how many units are necessary and where to place them? To simplify the questions, we turn to the modular design by embedding them into a network cell. For example, in ResNet [10], we put them into a residual module. The search space significantly shrinks for the limited positions in a residual module. We will investigate the position choices in experiments. The modular design allows using multiple SelfNorms and CrossNorms in a network. We will show in the ablation study that accumulated style recalibrations are helpful for model robustness. Since excessive style augmentations are harmful [172], we randomly turn on only some CrossNorms in a forward process. Random sampling encourages diverse augmentations even though the same data pass multiple times.

## 6.4 Experiments

We evaluate SelfNorm (SN) and CrossNorm (CN) on out-of-distribution data that arise from image corruptions and dataset differences. The evaluation uses not only supervised and semi-supervised settings but also image classification and segmentation tasks. In addition to the vision tasks, we also apply them to a NLP task. Due to limited space, we leave all ablation studies in the appendix.

**Image classification datasets.** We use benchmark datasets: CIFAR-10 [24], CIFAR-100, and ImageNet[26]. To evaluate the model robustness against corruption, we use the datasets: CIFAR-10-C, CIFAR-100-C, and ImageNet-C [11]. These datasets are the original test data poisoned by 15 everyday image corruptions from 4 general types: noise, blur, weather, and digital. Each noise has 5 intensity levels when injected into images.

**Image segmentation datasets.** We further validate our method using a domain generalization setting, where the models are trained without any target domain data and tested on the unseen domain. We use the synthetic dataset Grand Theft Auto V (GTA5) [44] as the source domain and generalize to the real-world dataset Cityscapes

[28]. GTA5 has the training, validation, and test divisions of 12,403, 6,382, and 6,181, more than those of 2,975, 500, and 1,525 from Cityscapes. Despite the differences, their pixel categories are compatible with each other, allowing to evaluate models' generalization capability from one to another.

**Sentiment classification datasets.** Besides vision tasks, we demonstrate that our method can also work well on NLP tasks. We use the out-of-distribution (OOD) generalization setting in binary sentiment classification. The model is trained on IMDb dataset [187] and is tested on SST-2 testing dataset [188]. The IMDb dataset collects highly polarized full-length lay movie reviews with 25,000 positive and 25,000 negative reviews. The SST-2 contains 9613 and 1821 reviews for training and testing, which is also a binary sentiment classification dataset but instead contains pithy expert movie reviews.

**Metric.** For image classification, we use test errors to measure robustness. Given corruption type $c$ and severity $s$, let $E_s^c$ denote the test error. For CIFAR datasets, we use the average over 15 corruptions and 5 severities: $1/75 \sum_{c=1}^{15} \sum_{s=1}^{5} E_{c,s}$. In contrast, for ImageNet, we normalize the corruption errors by those of AlexNet [117]: $1/15 \sum_{c=1}^{15} (\sum_{s=1}^{5} E_s^c / \sum_{s=1}^{5} E_{c,s}^{AlexNet})$. The above two metrics follow the convention [173] and are denoted as mean corruption errors (mCE) whether they are normalized or not. Different from classification, segmentation use the mean Intersection over Union (mIoU) over all categories as metric. For sentiment classification, we report accuracy as the metric.

**Hyper-parameters.** In the experiments, a SN unit uses one fully connected layer, followed by Batch Norm and a sigmoid layer. We put CN ahead of SN, and plug them into every cell in a network, e.g., each residual module in a ResNet. During training, we turn on only some CNs with probability to avoid excessive data augmentation. Usually, we conduct a grid search on four combinations of active numbers (1, 2) and probability (0.25, 0.5). For CN with cropping, we sample the bounding box ratio uniformly and set the threshold $t = 0.1$.

Table 6.1: mCE (%) on CIFAR-10-C and CIFAR-100-C. SNCN obtains lower errors than most previous methods with different backbones. Albeit some higher errors than AugMix, it is totally domain agnostic without relying on the image primitives, e.g., rotation, in AugMix. As SNCN and AugMix are orthogonal, their joint usage brings new state-of-the-art results.

| CIFAR-10-C | Basic | Cutout | Mixup | CutMix | AutoAug | AdvTr. | AugMix | SN | CN | SNCN | SNCN+AugMix |
|---|---|---|---|---|---|---|---|---|---|---|---|
| AllConvNet | 30.8 | 32.9 | 24.6 | 31.3 | 29.2 | 28.1 | 15.0 | 24.0 | 26.0 | 17.2 | **11.8** |
| DenseNet | 30.7 | 32.1 | 24.6 | 33.5 | 26.6 | 27.6 | 12.7 | 22.0 | 24.7 | 18.5 | **10.4** |
| WideResNet | 26.9 | 26.8 | 22.3 | 27.1 | 23.9 | 26.2 | 11.2 | 20.8 | 21.6 | 16.9 | **9.9** |
| ResNeXt | 27.5 | 28.9 | 22.6 | 29.5 | 24.2 | 27.0 | 10.9 | 21.5 | 22.4 | 15.7 | **9.1** |
| Mean | 29.0 | 30.2 | 23.5 | 30.3 | 26.0 | 27.2 | 12.5 | 22.1 | 23.7 | 17.0 | **10.3** |
| CIFAR-100-C | Basic | Cutout | Mixup | CutMix | AutoAug | AdvTr. | AugMix | SN | CN | SNCN | SNCN+AugMix |
| AllConvNet | 56.4 | 56.8 | 53.4 | 56.0 | 55.1 | 56.0 | 42.7 | 50.3 | 52.2 | 42.8 | **36.8** |
| DenseNet | 59.3 | 59.6 | 55.4 | 59.2 | 53.9 | 55.2 | 39.6 | 53.9 | 55.4 | 48.5 | **37.0** |
| WideResNet | 53.3 | 53.5 | 50.4 | 52.9 | 49.6 | 55.1 | 35.9 | 47.4 | 48.8 | 43.7 | **33.4** |
| ResNeXt | 53.4 | 54.6 | 51.4 | 54.1 | 51.3 | 54.4 | 34.9 | 47.6 | 47.0 | 40.8 | **30.8** |
| Mean | 55.6 | 56.1 | 52.6 | 55.5 | 52.5 | 55.2 | 38.3 | 49.8 | 50.9 | 43.5 | **34.7** |

Table 6.2: Clean error and mCE (%) of ResNet50 trained 90 epochs on ImageNet. SNCN, using simple domain-agnostic statistics, achieves comparable performance as AugMix. Jointly applying SNCN with AugMix and IBN can produce the lowest clean and corruption errors.

| | | Noise | | | Blur | | | | Weather | | | | Digital | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Aug. | Clean | Gauss. | Shot | Impulse | Defocus | Glass | Motion | Zoom | Snow | Frost | Fog | Bright | Contrast | Elastic | Pixel | JPEG | mCE |
| Standard | 23.9 | 79 | 80 | 82 | 82 | 90 | 84 | 80 | 86 | 81 | 75 | 65 | 79 | 91 | 77 | 80 | 80.6 |
| Patch Uniform | 24.5 | 67 | 68 | 70 | 74 | 83 | 81 | 77 | 80 | 74 | 75 | 62 | 77 | 84 | 71 | 71 | 74.3 |
| Random AA* | 23.6 | 70 | 71 | 72 | 80 | 86 | 82 | 81 | 81 | 77 | 72 | 61 | 75 | 88 | 73 | 72 | 76.1 |
| MaxBlur pool | 23.0 | 73 | 74 | 76 | 74 | 86 | 78 | 77 | 77 | 72 | 63 | 56 | 68 | 86 | 71 | 71 | 73.4 |
| SIN | 27.2 | 69 | 70 | 70 | 77 | 84 | 76 | 82 | 74 | 75 | 69 | 65 | 69 | 80 | 64 | 77 | 73.3 |
| AugMix* | 23.4 | 66 | 66 | 66 | 69 | 80 | 65 | 68 | 72 | 72 | 66 | 60 | 63 | 78 | 66 | 71 | 68.4 |
| CN | 23.4 | 73 | 75 | 75 | 78 | 89 | 79 | 82 | 79 | 75 | 66 | 61 | 69 | 97 | 69 | 74 | 75.3 |
| SN | 23.7 | 69 | 71 | 69 | 77 | 87 | 77 | 80 | 75 | 77 | 70 | 61 | 73 | 83 | 61 | 71 | 73.4 |
| SNCN | 23.3 | 66 | 67 | 65 | 77 | 89 | 76 | 80 | 72 | 72 | 67 | 59 | 47 | 83 | 62 | 72 | 70.4 |
| SNCN+AugMix | **22.3** | 61 | 62 | 60 | 70 | 77 | 62 | 68 | 62 | 65 | 63 | 55 | 43 | 73 | 55 | 66 | **62.8** |

### 6.4.1 Robustness against Unseen Corruptions for Image Classification

**Supervised training on CIFAR.** Following AugMix [173], we evaluate SN and CN with four different backbones: an All Convolutional Network [189], a DenseNet-BC (k = 12, d = 100) [190], a 40-2 Wide ResNet [112], and a ResNeXt-29 (32×4) [191]. We also use the same hyper-parameters in the AugMix Github repository[1].

According to Table 6.1, individual SN and CN can outperform most previous approaches on robustness against unseen corruptions and combining them can decrease

[1]https://github.com/google-research/augmix

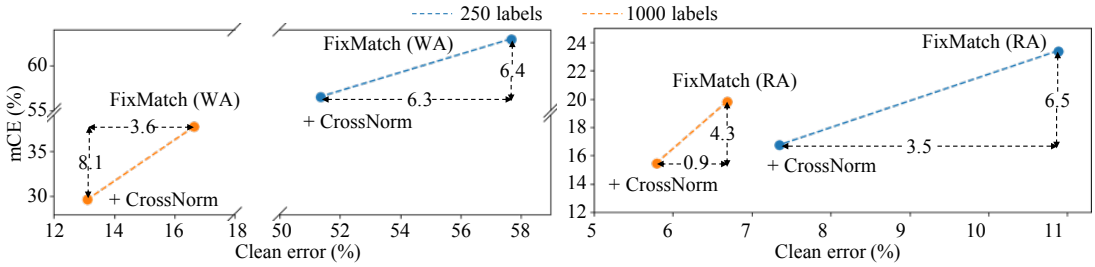



Figure 6.4: CN for semi-supervised CIFAR-10 classification. We apply CN on top of FixMatch with weak augmentation (WA) (**Left**), or strong RandAugment (RA) (**Right**). For either case, CN can substantially reduce both clean and corruption errors. Compared with RA, CN performs domain agnostic data augmentation, easily applicable to new domains.

the mean error by ∼12% on both CIFAR-10-C and CIFAR-100-C. One possible explanation is that the corruptions mainly change image textures. SN and CN, through style recalibration and augmentation, may help reduce the texture sensitivity and bias, making the classifiers more robust to unseen corruptions. Also, the domain-agnostic SN and CN are orthogonal to AugMix, which relies on domain-specific operations. Their joint application can continue to lower the mCEs by 2.2% and 3.6% on top of AugMix.

**Supervised training on ImageNet.** Following the AugMix Github repository, we train a ResNet-50 for 90 epochs with weight decay 1e-4. The learning rate starts from 0.1, divided by 10 at epochs 30 and 60. Note that AugMix reports the results of 180 epochs in their work. For a fair comparison, we also train it 90 epochs in our experiments. Besides, we also add Instance-batch normalization (IBN) [177] in the final combination with AugMix. It was initially designed for domain generalization but can also boost model robustness against corruption.

Table 6.2 gives the results on ImageNet. We can observe that both clean and corrupted errors decrease when applying SN and CN separately. Their joint usage can make the clean and corruption errors drop by 10.2% and 0.6% simultaneously, closing the gap with AugMix. Moreover, applying SN and CN on top of AugMix can significantly lower its clean and corruption errors by 1.1% and 5.6%, respectively, achieving state-of-the-art performance. IBN also makes some contributions here since it is complementary to other components.

**Semi-supervised training on CIFAR.** Apart from supervised training, we also evaluate CN in semi-supervised learning. Following state-of-the-art FixMatch [192] setting, we train a 28-2 Wide ResNet for 1024 epochs on CIFAR-10. The SGD optimizer applies with Nesterov momentum 0.9, learning rate 0.03, and weight decay 5e-4. The probability threshold to generate pseudo-labels is 0.95, and the weight for unlabeled data loss is 1. We sample 250 and 4,000 labeled data with random seed 1, leaving the rest as unlabeled data. In each experiment, we apply CrossNorm to either all data or only unlabeled data and choose the better one. Our experiments use the Pytorch FixMatch implementation [2], which has higher errors than the FixMatch reported. We choose it because it has the most stars among all the Pytorch implementations on Github.

Figure 6.4 shows the semi-supervised results. We run FixMatch with the strong RandAugment [130] or only weak random flip and crop augmentations. With either FixMatch version, CN can always decrease both the clean and corruption errors, demonstrating its effectiveness in semi-supervised training. Especially with the help of CN, training with 250 labels even has 3% lower corruption error than with 1000 labels, according to the right sub-figure. Additionally, two points are noteworthy here. First, we try FixMatch with only weak augmentations to simulate more general situations. For new domains other than natural images, humans may have the limited domain knowledge to design advanced augmentation operations. Fortunately, CN is domain-agnostic and easily applicable to such situations. Moreover, previous semi-supervised methods mainly focus on in-distribution generalization. Here we introduce out-of-distribution robustness as another metric for more comprehensive evaluation.

### 6.4.2 Generalization from Synthetic to realistic data for Image Segmentation

**Training setup.** We perform domain generalization from GTA5 (synthetic) to Cityscapes (realistic), following the setting of IBN [177]. It uses 1/4 training data in GTA5 to

[2]https://github.com/kekmodel/FixMatch-pytorch

Table 6.3: Segmentation results (mIoU) on GTAV-Cityscapes domain generalization using a FCN with ResNet50. SN and CN are comparable with IBN and domain randomization (DR) on the target domain. Combining SN and CN can achieve state-of-the-art performance.

| Methods | Baseline | IBN | DR | SN | CN | SNCN |
|---|---|---|---|---|---|---|
| Source | 63.7 | 64.2 | 49.0 | **64.6** | 61.2 | 63.5 |
| Target | 21.4 | 29.6 | 32.7 | 29.9 | 32.0 | **36.5** |

match the data scale of Cityscapes. We train the FCN [64] with ResNet50 backbone in source domain GTA5 for 80 epochs with batch size 16. The network is initialized with ImageNet pre-trained weights. Then we test the model on both the source domain and target domain. The training uses random scaling, flip, rotation, and cropping ($713 \times 713$) for data augmentation. We use the 2-instance CN with style cropping in this setting. Besides, we re-implement the domain randomization [178] and make the training iterations the same as ours. It transfers the synthetic images to 15 auxiliary domains with ImageNet image styles.

**Results.** Based on Table 6.3, SN and CN both can substantially increase the segmentation accuracy on the target domain by 8.5% and 10.6%. SN learns to highlight the discriminative styles that are likely to share across domains. CN performs style augmentation to make the model focus more on domain-invariant features. SN and CN get comparable generalization performance as state-of-the-art IBN [177] and domain randomization [178]. However, CN significantly outperforms the domain randomization method by 12.2% on the source accuracy. Because the domain randomization transfers external styles to the source training data, causing dramatic distribution shifts. Moreover, combining SN and CN gives the best generalization performance while still maintaining high source accuracy.

### 6.4.3 Out-of-Distribution Generalization for Sentiment Classification

**Setup.** We also conduct out-of-distribution generalization on the binary sentiment classification task in the NLP field to validate the versatility of SelfNorm and CrossNorm. The model is trained on the IMDb dataset and then tested on SST-2 dataset. Follow

Table 6.4: Accuracy (Acc) on OOD generalization for sentiment classification using GloVe embedding and ConvNets model. We train the model on IMDb source dataset and test on SST-2 target dataset.

| Methods | Baseline | SN | CN | SNCN |
|---|---|---|---|---|
| Source | 85.67 | **86.30 (↑ 0.63)** | 85.14 (↓ 0.53) | 85.92 (↑ 0.25) |
| Target | 71.86 | 73.93 (↑ 2.07) | 73.03 (↑ 1.17) | **74.91 (↑ 3.05)** |

the setting of [179], we use GloVe [193] word embedding and the Convolutional Neural Networks (ConvNets) [194] as the classification model. We use the implementation of ConvNets in this repository[3]. The convolutional layers with three kernel sizes (3,4,5) are used to extract $n-gram$ features within the review texts. SN and CN units are placed between the embedding layer and the convolutional layers. We use the Adam optimizer and train the model for 20 epochs.

**Results.** From Table 6.4, we can find that SN improves the performance in both the source and target domains by 2.07% and 0.63%. CN can also increase target accuracy without much degradation in the source domain. Combining them gives a 3.05% accuracy boost. This experiment indicates that SN and CN can also work in the NLP area, not limited to the vision tasks. Despite the lack of intuitive explanations as for the image data, the mean and variance statistics in NLP data are also useful in facilitating out-of-distribution generalization.

### 6.4.4 Visualization

Apart from the quantitative comparisons, we also provide some visualization results of SN and CN to better understand their effects.

**Visualization setup**. Our visualization builds on the technique: understanding deep image representations by inverting them [195]. The goal is to find an image whose feature representation best matches the given one. The search is done automatically by a SGD optimizer with learning rate 1e4, momentum 0.9, and 200 iterations. The

[3]https://github.com/bentrevett/pytorch-sentiment-analysis

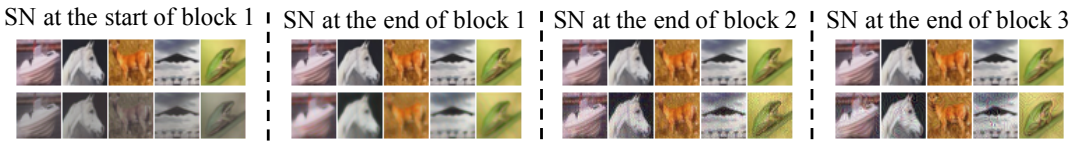



Figure 6.5: Visualizing 4 single SNs by comparing images before (**Top**) and after (**Bottom**) them. The left two, lying in shallow locations, can adjust styles by suppressing color and adding blur. As SN goes deeper, the recalibration effect is subtle, due to the high-level feature abstraction.

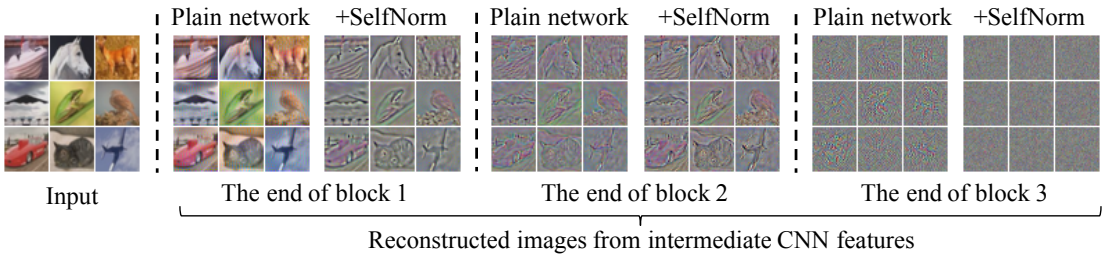



Figure 6.6: Visualizing accumulated SNs by comparing inverted images. SNs in block 1 can wash away much style information preserved in the vanilla network. Similarly, the plain network's final representation retains some high-frequency signals which are suppressed by SNs.

learning rate is divided by 10 every 40 iterations. During the optimization, the network is in its evaluation mode with its parameters fixed. In the experiment, we use WideResNet-40-2 and images from CIFAR-10. In visualizing CN, we use the training images and a model trained for 1 epoch. The SN visualization uses test images and a well-trained model. We use different settings for them because CN is for training, while SN works in testing.

**Visualization of individual SNs**. To visualize SN at a network location, we first forward an image to obtain the target representation immediately after the SN. Then we turn off the chosen SN and optimize the original image to make its representation fit the target one. In this way, we can examine SN's effect by observing changes in image space. As shown in Figure 6.5, SN can primarily reduce the contrast and color at the first network block. The effect becomes more subtle as SN goes deeper into the network. One possible explanation is that the high-level representations lose many low-level details, making it difficult to visualize their changes.

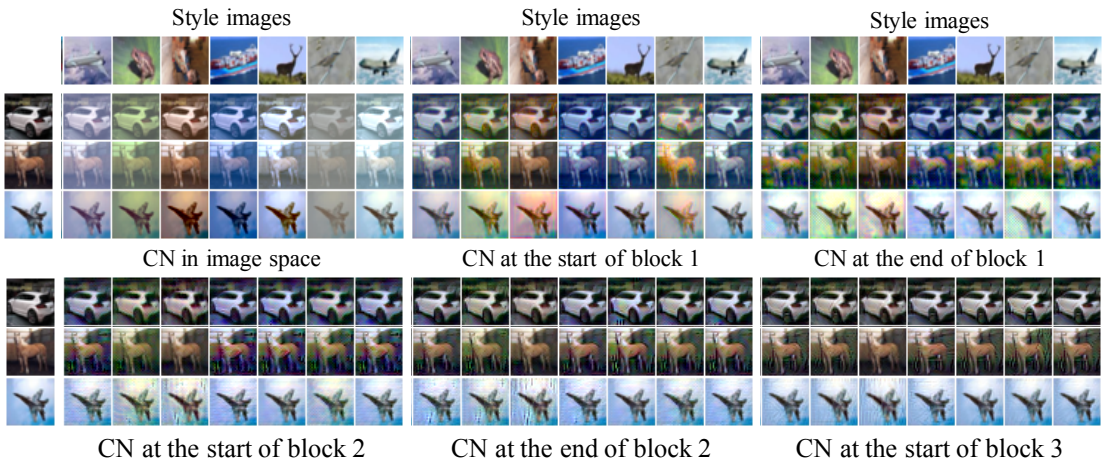



Figure 6.7: CN visualization at image level and different locations inside a WideResNet-40-2. Both the content (**Row**) and style (**Column**) images are from CIFAR-10. The style rendering changes from global to local as CN gets deeper in the network.

**Visualizing multiple SNs**. In addition to visualizing individual SNs, it is also interesting to see their compound effect. To this end, we reconstruct an image from random noises by matching its representation with a given one. The reconstructed image can show what information is preserved by the feature representation. By comparing two reconstructed images from a network with or without SN, we can observe the joined recalibration effects of SNs before a selected location. From Figure 6.6, we can find SNs in the first two network blocks can suppress much style information and preserve object shapes. The reconstructions from block 3 do not look visually informative due to the high-level abstraction. Even so, SNs can restrain the high-frequency signals kept in the vanilla network.

**CN visualization**. In the CN visualization, we pair one content image with multiple style images for better illustration. We first forward them to get their representations at a chosen position. Then, we compute the standardized features from the content image representation and the means and variances of the style image representation. The optimization starts from the content image and tries to fit its representation to the target one mixing the standardized features with different means and variances. Figure 6.7 shows diverse style changes made by CN. The style changes become more local and subtle as CN moves deeper in the network.

Table 6.5: 1-instance CN *v.s.* 2-instance CN. We report the mCEs of WideResNet-40-2 on CIFAR-100-C. The 2-instance mode consistently obtains lower errors than the 1-instance one. Moreover, proper cropping can further help decrease the errors.

| | 1-instance | | | | 2-instance | | | |
|---|---|---|---|---|---|---|---|---|
| Crop | Neither | Content | Style | Both | Neither | Content | Style | Both |
| mCE | 50.5 | 50.6 | 50.6 | 49.6 | 48.8 | 49.0 | **47.9** | 48.5 |

Table 6.6: Exploration of Block choices for SN and CN. We compare the mCEs (%) when applying SN and CN to image space or different blocks in WideResNet-40-2. Using them in all blocks gives the lowest errors on CIFAR-100-C.

| Stages | N/A | Image | Block 1 | Block 2 | Block 3 | All |
|---|---|---|---|---|---|---|
| SN | 53.3 | 52.9 | 48.9 | 52.2 | 51.3 | **47.4** |
| CN | 53.3 | 54.3 | 52.2 | 51.2 | 51.5 | **48.8** |

Table 6.7: Order study of SN and CN. They both locate at the post-addition position in each residual cell of WideResNet-40-2, and we report the mCE on CIFAR-100-C.

| Order | SN→CN | CN→SN |
|---|---|---|
| mCE(%) | 46.9 | 46.6 |

Table 6.8: Comparison with SE. SN obtains much lower mCE than SE when they are applied to WideResNet-40-2 and CIFAR-100-C. We place SN in the post-addition location.

| | Basic | SE | SE(post-addition) | SN |
|---|---|---|---|---|
| mCE(%) | 53.3 | 52.3 | 51.0 | **47.4** |

### 6.4.5 Ablation Study on CIFAR

**CN variants**. CN can be in 1-instance or 2-instance mode with four cropping options. According to Table 6.5, the 2-instance mode constantly gets lower errors than the 1-instance. Furthermore, cropping can help decrease the error since it can encourage style augmentation diversity. Note that style cropping may not always be superior. We will give a more detailed study on cropping choices later.

**Blocks choices for SN and CN**. Given both SN and CN placed at the post-addition location of a residual cell in WideResNet-40-2, we study which blocks in a network should build on the modified cells. No residual cell is required if applying SN

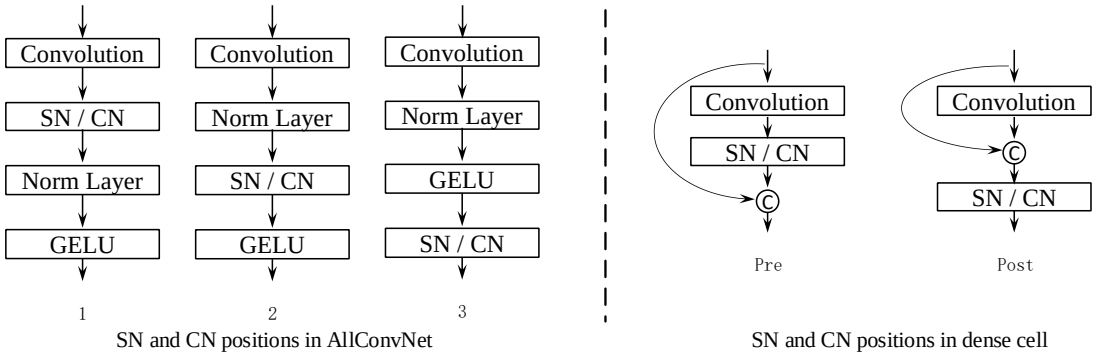



Figure 6.8: Illustration of SN and CN positions in AllConvNet block, and dense cells in DenseNet. For blocks in AllConvNet, we name the position after convolution layer as *1*, after normalization layer as *2*, and after GELU layer as *3*. For dense cells in DenseNet, we label the position before feature concatenation as *Pre*, and after concatenation as *Post*.

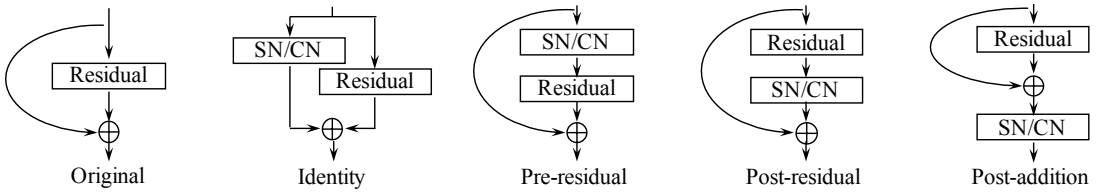



Figure 6.9: Illustration of SN and CN positions in a residual module. We explore four positions: identity, pre-residual, post-residual, and post-addition.

and CN to the image space. According to Table 6.6, They both perform the best on CIFAR-100-C when plugged into all blocks.

**Order of SN and CN**. In this experiment, we study two orders: SN→CN and CN→SN when plugging them into the post-addition place in all residual cells of WideResNet-40-2. Table 6.7 shows very close mCEs, indicating their order has little influence on the robustness performance.

**SN** *v.s.* **SE**. Although SN shares a similar attention mechanism with SE, SN obtains much lower corruption error than SE, according to Table 6.8. SN recalibrates the feature map statistics, suppressing unseen styles in the OOD data, whereas SE, modeling the interdependence between feature channels, may not help OOD robustness.

**Modular positions**. Here we investigate the positions of SN and CN inside network cells. We give a comprehensive study on different cells and measure the performance on both CIFAR-10-C and CIFAR-100-C. Specifically, we conduct experiments using

Table 6.9: Evaluation of **SN** modular positions for AllConvNet, DenseNet, WIdeResNet and ResNeXt. The impacts of different positions are measured by mCE on both CIFAR-10-C (**Top**) and CIFAR-100-C (**Bottom**). Note that the four backbones have three types of cells whose positions are illustrated in Figures 6.8 and 6.9.

| SN on CIFAR-10-C | | | | |
|---|---|---|---|---|
| Position | 1 | 2 | 3 | - |
| AllConvNet | **24.0** | 26.4 | 25.6 | - |
| Position | Pre | Post | - | - |
| DenseNet | 23.4 | **22.0** | - | - |
| Position | Residual | Post | Pre | Identity |
| WideResNet | 22.7 | 21.3 | **20.8** | 22.3 |
| Position | Residual | Post | Pre | Identity |
| ResNeXt | 21.9 | 24.8 | **21.5** | 22.0 |
| SN on CIFAR-100-C | | | | |
| Position | 1 | 2 | 3 | - |
| AllConvNet | **50.3** | 51.6 | 51.0 | - |
| Position | Pre | Post | - | - |
| DenseNet | **53.9** | 54.7 | - | - |
| Position | Residual | Post | Pre | Identity |
| WideResNet | 49.3 | **47.4** | 49.8 | 48.4 |
| Position | Residual | Post | Pre | Identity |
| ResNeXt | **47.6** | 49.0 | 50.9 | 50.4 |

four backbones: AllConvNet, DenseNet, WIdeResNet and ResNeXt, consisting of three types of cells: naive convolutional cell, dense cell, and residual module, illustrated in Figures 6.8 and 6.9. According to Tables 6.9 and 6.10, SN's optimal positions are different for CIFAR-10-C and CIFAR-100-C, while CN has stable best positions across the two datasets.

**Cropping Choices for CN**. Cropping enables diverse statistics transfer between feature maps. Here we study four cropping choices: neither (no cropping), style, content, and both. In Table 6.11, we can find the best cropping choice may change over backbones and datasets.

**Incremental ablation study**. SN and CN are general and straightforward normalization techniques to improve the OOD robustness. They are orthogonal to each other, and other methods such as the consistency regularization [173] and AugMix [173]. According to Table 6.12, they can lower the corruption error both separately

Table 6.10: Evaluation of **CN** positions in the cells of four backbones. We measure the position influence by mCE on CIFAR-10-C (**Top**) and CIFAR-100-C (**Bottom**). The position choices in the naive convolution cell, dense cell, and residual module are shown in Figures 6.8 and 6.9.

| CN on CIFAR-10-C | | | | |
|---|---|---|---|---|
| Position | 1 | 2 | 3 | - |
| AllConvNet | **26.0** | 26.3 | 26.8 | - |
| Position | Pre | Post | - | - |
| DenseNet | **24.7** | 29.2 | - | - |
| Position | Residual | Post | Pre | Identity |
| WideResNet | 25.2 | **21.6** | 24.9 | 23.3 |
| Position | Residual | Post | Pre | Identity |
| ResNeXt | 26.7 | **22.4** | 23.8 | 26.9 |
| CN on CIFAR-100-C | | | | |
| Position | 1 | 2 | 3 | - |
| AllConvNet | **52.2** | 52.5 | 52.7 | - |
| Position | Pre | Post | - | - |
| DenseNet | **55.4** | 57.6 | - | - |
| Position | Residual | Post | Pre | Identity |
| WideResNet | 52.1 | **48.8** | 51.7 | 50.3 |
| Position | Residual | Post | Pre | Identity |
| ResNeXt | 51.5 | **47.0** | 47.9 | 50.2 |

and jointly. On top of them, proper cropping, consistency regularization, and domain-specific AugMix can further advance the OOD robustness.

### 6.4.6 Ablation Study on ImageNet

**CN** *v.s.* **Stylized-ImageNet**. We also compare CN to Stylized-ImageNet, which transfers styles from external datasets to perform style augmentation. Stylized-ImageNet finetunes a pre-trained ResNet-50 for 45 epochs with double data (stylized and original ImageNets) in each epoch. To compare CrossNorm with Stylized-ImageNet, we perform the finetuning for 90 epochs using only the original ImageNet. In Table 6.13, although Stylized-ImageNet has 2% lower corruption error than CN, its clean error is 3.8% higher. Because the external styles in Stylized-ImageNet cause large distribution shifts, impairing its clean generalization. In contrast, The more consistent yet diverse internal styles help CN decreases both corruption and clean errors.

Table 6.11: Study of CN cropping choices. We evaluate four cropping choices: neither, content, style, and both when jointly using SN and CN in four backbones. The performance is measured by mCE on both CIFAR-10-C (**Top**) and CIFAR-100-C (**Bottom**). We put the modular positions next to the backbone names.

| SNCN with cropping on CIFAR-10-C | | | | |
|---|---|---|---|---|
| Backbone | Neither | Content | Style | Both |
| AllConvNet, 1 | 19.0 | 20.3 | **18.8** | 20.3 |
| DenseNet, Conv1 Pre | 18.8 | **18.2** | 18.7 | 18.8 |
| WideResNet, Post | 17.9 | 18.0 | **16.8** | 17.5 |
| ResNeXt, Post | **17.7** | 18.5 | 18.4 | 18.6 |
| SNCN with cropping on CIFAR-100-C | | | | |
| Backbone | Neither | Content | Style | Both |
| AllConvNet, 1 | 44.2 | 46.9 | **43.9** | 46.1 |
| DenseNet, Conv1 Pre | 51.4 | 49.4 | 49.1 | **49.0** |
| WideResNet, Post | 46.6 | 45.1 | 45.8 | **44.5** |
| ResNeXt, Post | **41.0** | 44.9 | 43.0 | 46.5 |

**SN and CN locations**. Moreover, in Table 6.14, we also investigate the SN and CN locations in a residual module using ImageNet and ResNet50. Similar to the CIFAR results, the post-addition position performs the best for corruption robustness.

**Ablation study with IBN**. Table 6.15 reports the results of applying SN or CN with IBN. We can observe that they can cooperate to improve the corruption robustness of ResNet50. Moreover, integrating SN, CN, IBN, and AugMix can bring the lowest corruption error. This shows SN and CN's advantage that they are general and simple to boost other state-of-the-art methods.

## 6.5 Summary

In this work, we have presented SelfNorm and CrossNorm, two simple yet effective normalization techniques to improve OOD robustness. They form a unity of opposites as they confront and conform to each other in terms of approach (statistics usage) and goal (OOD robustness). Beyond their extensive applications, they may also shed light on developing domain agnostic methods applicable to multiple fields such as vision and language, and broad OOD generalization circumstances such as unseen corruptions and

Table 6.12: Incremental ablation study for SN, CN, cropping, consistency regularization and AugMix. We report the mCEs of four backbones on both CIFAR-10-C (**Top**) and CIFAR-100-C (**Bottom**). The modular position and cropping choice are also given in each row.

| CIFAR-10-C | | | | | | | |
|---|---|---|---|---|---|---|---|
| Backbone | Basic | SN | CN | SN+CN +Crop | SN+CN+Crop +Consistency | AugMix | SNCN+Crop +AugMix |
| AllConvNet, 1, style | 30.8 | 24.0 | 26.0 | 18.8 | 17.2 | 15.0 | **11.8** |
| DenseNet, Conv1 Pre, both | 30.7 | 22.0 | 24.7 | 18.8 | 18.5 | 12.7 | **10.4** |
| WideResNet, Post, both | 26.9 | 20.8 | 21.6 | 17.5 | 16.9 | 11.2 | **9.9** |
| ResNeXt, Post, neither | 27.5 | 21.5 | 22.4 | 17.7 | 15.7 | 10.9 | **9.1** |
| CIFAR-100-C | | | | | | | |
| Backbone | Basic | SN | CN | SN+CN +Crop | SN+CN+Crop +Consistency | AugMix | SNCN+Crop +AugMix |
| AllConvNet, 1, style | 56.4 | 50.3 | 52.2 | 43.9 | 42.8 | 42.7 | **36.8** |
| DenseNet, Conv1 Pre, both | 59.3 | 53.9 | 55.4 | 49.0 | 48.5 | 39.6 | **37.0** |
| WideResNet, Post, both | 53.30 | 47.4 | 48.8 | 44.5 | 43.7 | 35.9 | **33.4** |
| ResNeXt, Post, neither | 53.4 | 47.6 | 47.0 | 41.0 | 40.8 | 34.9 | **30.8** |

Table 6.13: Comparison of Stylized-ImageNet and CN. Following the Stylized-ImageNet setup, we finetune a pre-trained ResNet50 model 90 epochs on ImageNet. Compared with SIN, CN holds a better balance between clean and corruption errors.

| | Basic | Stylized-ImageNet | CN |
|---|---|---|---|
| Clean error (%) | 23.9 | 27.2 | **23.4** |
| mCE(%) | 80.6 | **73.3** | 75.3 |

distribution gaps across datasets. Given the simplicity of SelfNorm and CrossNorm, we believe there is substantial room for improvement. The current channel-wise mean and variance are not optimal to encode diverse styles. One possible direction is to explore better style representations.

Table 6.14: Investigation of SN (**Top**) and CN (**Bottom**) positions in a residual module of ResNet50 trained 90 epochs on ImageNet.

| SN modular positions | | | | |
|---|---|---|---|---|
| Position | Identity | Pre-Residual | Post-Residual | Post-Addition |
| Clean error (%) | 24.0 | **23.0** | 23.2 | 23.7 |
| mCE(%) | 75.5 | 75.8 | 74.8 | **73.4** |
| CN modular positions | | | | |
| Position | Identity | Pre-Residual | Post-Residual | Post-Addition |
| Clean error (%) | 25.2 | **23.4** | 23.5 | **23.4** |
| mCE(%) | 78.2 | 75.8 | 77.5 | **75.3** |

Table 6.15: Ablation study of IBN, SN, CN, consistency regularization(CR), and AugMix(AM) on ImageNet-C with ResNet50. IBN, initially designed for domain generalization, can also decrease mCE. Both SN and CN can further lower the error based on IBN. Combining them with AM gives the best robustness performance.

| | ResNet50 | ResNet50+IBN(a) | | | | ResNet50+IBN(b) | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Basic | Basic | +CN | +CN+CR | +CN+AM | Basic | +SN | +SN+AM | +SNCN+AM |
| Clean err(%) | 23.9 | 23.2 | 23.1 | 22.6 | 22.5 | 24.0 | 23.5 | **22.3** | **22.3** |
| mCE(%) | 80.6 | 75.1 | 73.2 | 73.6 | 66.4 | 74.1 | 72.6 | 64.1 | **62.8** |

# Chapter 7

# Conclusions and Future Work

## 7.1 Conclusions

This dissertation has analyzed three trends in recent deep learning development, revealed three current challenges, and tackled them from three prospects: automatic data augmentation, efficient architecture design, and robust feature normalization.

In chapters 2-4, we present three methods to increasingly automate the data augmentation process by learning sampling distributions, regularizing augmented data distributions without domain expertise, and removing the reliance on human-specified augmentation operations. We experimentally demonstrate that they are easy to use in diverse tasks (human pose estimation, image classification, and image segmentation) and effective in boosting performance without using additional training data.

Chapter 5 studies the problem of designing efficient U-Net architectures. We provide solutions to improve parameter efficiency, bit-width efficiency, and memory efficiency through shortcut connections, representation (parameters, features, gradients) quantization, and memory sharing. The proposed quantized coupled U-Nets have demonstrated significant efficiency advantages over previous popular stacked U-Nets while maintaining comparable accuracy in the tasks of human pose estimation and facial landmark localization.

Finally, we provide two normalization techniques, SelfNorm and CrossNorm, for OOD robustness in Chapter 6. SelfNorm employs attention to suppress trivial normalization statistics that are difficult to generalize to OOD data, while CrossNorm enriches the combinations of statistics and standardized features by swapping statistics between feature maps. Experiments across diverse domains and tasks show that SelfNorm and

CrossNorm can work well individually and together in advancing the OOD robustness and further boosting previous state-of-the-art methods.

## 7.2 Future Work

Deep learning still confronts the open problems of lacking data efficiency, model efficiency, and OOD robustness. Overcoming the problems will have profound importance for both theory and practice. We discuss some possible directions to explore in future.

One limitation of the proposed automatic data augmentation approaches is that they are developed mainly for supervised training, but it is also worthwhile to study data augmentation strategies in unsupervised and self-supervised settings. For example, unlabeled video data are prevalent and contain diverse naturally existing data variations. Since humans generally learn a lot from observing dynamic objects and scenes, teaching deep models to do so is exciting and meaningful. Beyond data augmentation, incorporating active learning with deep learning is also worth investigation. The motivation is that not all labeled data are equally important. Given a set of labeled data, can we identify a subset that contributes more than the rest? When labeling additional data, can the training algorithm tell what kind of data to label to maximize gains? Answering these questions would make training deep models significantly more data-efficient.

Moreover, the proposed efficient U-Net architecture relies on hand-crafted heuristics to determine the shortcut connections and quantization bit-widths. A potential improvement is to use the neural architecture search to learn the design/compression policies. Changing from rule-based to learning-based policies fits the goals of both autonomation and efficiency in deep learning. Whether manually designed or automatically searched, a produced efficient architecture is generally specialized to one edge device or platform with certain resource constraints. However, in practice, efficient deployment is expected to meet various hardware constraints such as different smartphone generations and battery levels. Therefore, avoiding repeatedly developing efficient models for each case is an important future direction.

While the research community has the consensus to build deep models that can generalize well to OOD scenarios, it is still unclear how to comprehensively evaluate the OOD robustness. Existing robustness datasets, consisting of limited synthetic or realistic distortions, are far from capturing the rich data variations that naturally occur, leading to inadequate evaluations. One possible solution for computer vision tasks is to test deep models on real-world videos instead of the benchmark image data. Going one step further, we hope to decouple the model robustness from specific datasets, e.g., developing robust models with theoretical guarantees. In addition to pure generalization, adaptation for robustness is an exciting direction. Humans, though known to be good at generalization, still keep adapting their behaviors and decisions to new environments. Similarly, we believe the deep models would harvest more robustness if they make online adaptation to the deployment environments.

# References


[1] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *ECCV*, 2016.

[2] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv*, 2018.

[3] Xi Peng, Zhiqiang Tang, Fei Yang, Rogerio S Feris, and Dimitris Metaxas. Jointly optimize data augmentation and network training: Adversarial data augmentation in human pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2226–2234, 2018.

[4] Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, Alan Yuille, and Quoc V Le. Adversarial examples improve image recognition. *arXiv preprint arXiv:1911.09665*, 2019.

[5] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[6] Ting-Jui Chang, Yukun He, and Peng Li. Efficient two-step adversarial defense for deep neural networks. *arXiv preprint arXiv:1810.03739*, 2018.

[7] Daniel Ho, Eric Liang, Ion Stoica, Pieter Abbeel, and Xi Chen. Population based augmentation: Efficient learning of augmentation policy schedules. *arXiv preprint arXiv:1905.05393*, 2019.

[8] Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. Fast autoaugment. In *NeurIPS*, pages 6662–6672, 2019.

[9] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *CVPR*, pages 113–123, 2019.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[11] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*, 2019.

[12] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[13] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[14] Donald Olding Hebb. *The organization of behavior: a neuropsychological theory*. J. Wiley; Chapman & Hall, 1949.

[15] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[16] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[17] Geoffrey E Hinton. Distributed representations. 1984.

[18] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[19] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[20] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[21] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[22] Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann L Cun. Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems*, pages 1137–1144, 2007.

[23] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[24] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[25] Gregory Griffin, Alex Holub, and Pietro Perona. Caltech-256 object category dataset. 2007.

[26] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[27] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852, 2017.

[28] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[29] Ignacio Arganda-Carreras, Srinivas C Turaga, Daniel R Berger, Dan Cireşan, Alessandro Giusti, Luca M Gambardella, Jürgen Schmidhuber, Dmitry Laptev, Sarvesh Dwivedi, Joachim M Buhmann, et al. Crowdsourcing the creation of image segmentation algorithms for connectomics. *Frontiers in neuroanatomy*, 9:142, 2015.

[30] Georgios A Kaissis, Marcus R Makowski, Daniel Rückert, and Rickmer F Braren. Secure, privacy-preserving and federated machine learning in medical imaging. *Nature Machine Intelligence*, pages 1–7, 2020.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[32] Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V Le. Self-training with noisy student improves imagenet classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10687–10698, 2020.

[33] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. *arXiv preprint arXiv:1912.11370*, 6:1, 2019.

[34] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.

[35] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[36] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, 2016.

[37] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28:649–657, 2015.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[39] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[40] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[41] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 fourth international conference on 3D vision (3DV)*, pages 565–571. IEEE, 2016.

[42] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.

[43] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[44] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *European conference on computer vision*, pages 102–118. Springer, 2016.

[45] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[46] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. *Advances in neural information processing systems*, 28:2017–2025, 2015.

[47] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[48] Zhiqiang Tang, Xi Peng, Tingfeng Li, Yizhe Zhu, and Dimitris N Metaxas. Adatransform: Adaptive data transformation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2998–3006, 2019.

[49] Zhiqiang Tang, Yunhe Gao, Leonid Karlinsky, Prasanna Sattigeri, Rogerio Feris, and Dimitris Metaxas. Onlineaugment: Online data augmentation with less domain knowledge. *arXiv preprint arXiv:2007.09271*, 2020.

[50] Zhiqiang Tang, Xi Peng, Shijie Geng, Yizhe Zhu, and Dimitris N Metaxas. Cu-net: coupled u-nets. *arXiv preprint arXiv:1808.06521*, 2018.

[51] Zhiqiang Tang, Xi Peng, Shijie Geng, Lingfei Wu, Shaoting Zhang, and Dimitris Metaxas. Quantized densely connected u-nets for efficient landmark localization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 339–354, 2018.

[52] Zhiqiang Tang, Xi Peng, Kang Li, and Dimitris N Metaxas. Towards efficient u-nets: A coupled and quantized approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[53] Zhiqiang Tang, Yunhe Gao, Yi Zhu, Zhi Zhang, Mu Li, and Dimitris Metaxas. Selfnorm and crossnorm for out-of-distribution robustness. *arXiv preprint arXiv:2102.02811*, 2021.

[54] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[55] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[56] Mohamed Elhoseiny, Yizhe Zhu, Han Zhang, and Ahmed Elgammal. Link the head to the "beak": Zero shot learning from noisy text description at part precision. In *CVPR*, 2017.

[57] Xiangxin Zhu, Dragomir Anguelov, and Deva Ramanan. Capturing long-tail distributions of object subcategories. In *CVPR*, 2014.

[58] Zhiqiang Tang, Yifan Zhang, Zechao Li, and Hanqing Lu. Face clustering in videos with proportion prior. In *IJCAI*, 2015.

[59] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *IEEE*, 1998.

[60] Alexander Toshev and Christian Szegedy. Deeppose: Human pose estimation via deep neural networks. In *CVPR*, 2014.

[61] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *CVPR*, 2016.

[62] Xiaolong Wang, Abhinav Shrivastava, and Abhinav Gupta. A-fast-rcnn: Hard positive generation via adversary for object detection. *arXiv*, 2017.

[63] Xi Peng, Rogerio S Feris, Xiaoyu Wang, and Dimitris N Metaxas. A recurrent encoder-decoder network for sequential face alignment. In *ECCV*, 2016.

[64] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[65] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *ICCV*, 2017.

[66] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

[67] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *arXiv*, 2016.

[68] He Zhang, Vishwanath Sindagi, and Vishal M Patel. Image de-raining using a conditional generative adversarial network. *arXiv*, 2017.

[69] Yizhe Zhu, Mohamed Elhoseiny, Bingchen Liu, Xi Peng, and Ahmed Elgammal. A generative adversarial approach for zero-shot learning from noisy texts. In *CVPR*, 2018.

[70] Aron Yu and Kristen Grauman. Semantic jitter: Dense supervision for visual comparisons via synthetic images. In *ICCV*, 2017.

[71] Yu Chen, Chunhua Shen, Xiu-Shen Wei, Lingqiao Liu, and Jian Yang. Adversarial posenet: A structure-aware convolutional network for human pose estimation. In *ICCV*, 2017.

[72] Chia-Jung Chou, Jui-Ting Chien, and Hwann-Tzong Chen. Self adversarial training for human pose estimation. *arXiv*, 2017.

[73] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *IJCV*, 2013.

[74] Lingfei Wu, Kesheng John Wu, Alex Sim, Michael Churchill, Jong Y Choi, Andreas Stathopoulos, Choong-Seock Chang, and Scott Klasky. Towards real-time detection and tracking of spatio-temporal features: Blob-filaments in fusion plasma. *TBD*, 2016.

[75] Joao Carreira, Pulkit Agrawal, Katerina Fragkiadaki, and Jitendra Malik. Human pose estimation with iterative error feedback. In *CVPR*, 2016.

[76] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In *CVPR*, 2015.

[77] Peiyun Hu and Deva Ramanan. Bottom-up and top-down reasoning with hierarchical rectified gaussians. In *CVPR*, 2016.

[78] Leonid Pishchulin, Eldar Insafutdinov, Siyu Tang, Bjoern Andres, Mykhaylo Andriluka, Peter V Gehler, and Bernt Schiele. Deepcut: Joint subset partition and labeling for multi person pose estimation. In *CVPR*, 2016.

[79] Ita Lifshitz, Ethan Fetaya, and Shimon Ullman. Human pose estimation using deep consensus voting. In *ECCV*, 2016.

[80] Georgia Gkioxari, Alexander Toshev, and Navdeep Jaitly. Chained predictions using convolutional neural networks. In *ECCV*, 2016.

[81] Eldar Insafutdinov, Leonid Pishchulin, Bjoern Andres, Mykhaylo Andriluka, and Bernt Schiele. Deepercut: A deeper, stronger, and faster multi-person pose estimation model. In *ECCV*, 2016.

[82] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines. In *CVPR*, 2016.

[83] Adrian Bulat and Georgios Tzimiropoulos. Human pose estimation via convolutional part heatmap regression. In *ECCV*, 2016.

[84] Jonathan J Tompson, Arjun Jain, Yann LeCun, and Christoph Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. In *NIPS*, 2014.

[85] Xianjie Chen and Alan L Yuille. Articulated pose estimation by a graphical model with image dependent pairwise relations. In *NIPS*, 2014.

[86] Xiao Chu, Wei Yang, Wanli Ouyang, Cheng Ma, Alan L Yuille, and Xiaogang Wang. Multi-context attention for human pose estimation. *arXiv*, 2017.

[87] Wei Yang, Shuang Li, Wanli Ouyang, Hongsheng Li, and Xiaogang Wang. Learning feature pyramids for human pose estimation. *arXiv*, 2017.

[88] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, Lecture Notes in Computer Science, 2015.

[89] Xi Peng, Junzhou Huang, Qiong Hu, Shaoting Zhang, Ahmed Elgammal, and Dimitris Metaxas. From circle to 3-sphere: Head pose estimation by instance parameterization. *CVIU*, 2015.

[90] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. *arXiv*, 2016.

[91] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2D human pose estimation: New benchmark and state of the art analysis. In *CVPR*, 2014.

[92] Sam Johnson and Mark Everingham. Clustered pose and nonlinear appearance models for human pose estimation. In *BMVC*, 2010.

[93] T Tieleman and G Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. In *NNML*, 2012.

[94] Yi Yang and Deva Ramanan. Articulated pose estimation with flexible mixtures-of-parts. In *CVPR*, 2011.

[95] Leonid Pishchulin, Mykhaylo Andriluka, Peter Gehler, and Bernt Schiele. Strong appearance and expressive spatial models for human pose estimation. In *ICCV*, 2013.

[96] Umer Rafi, Bastian Leibe, Juergen Gall, and Ilya Kostrikov. An efficient convolutional network for human pose estimation. In *BMVC*, 2016.

[97] Vasileios Belagiannis and Andrew Zisserman. Recurrent human pose estimation. In *FG*, 2017.

[98] Yong Zhang, Baoyuan Wu, Weiming Dong, Zhifeng Li, Wei Liu, Bao-Gang Hu, and Qiang Ji. Joint representation and estimator learning for facial action unit intensity estimation. In *CVPR*, 2019.

[99] Alexander J Ratner, Henry Ehrenberg, Zeshan Hussain, Jared Dunnmon, and Christopher Ré. Learning to compose domain-specific transformations for data augmentation. In *NIPS*, 2017.

[100] Eli Schwartz, Leonid Karlinsky, Joseph Shtok, Sivan Harary, Mattias Marder, Rogerio Feris, Abhishek Kumar, Raja Giryes, and Alex M Bronstein. Delta-encoder: an effective sample synthesis method for few-shot object recognition. In *NIPS*, 2018.

[101] Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *arXiv*, 2017.

[102] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *ICCV*, 2017.

[103] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 2017.

[104] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv.*, 2018.

[105] Xuehan Xiong and Fernando De la Torre. Supervised descent method and its applications to face alignment. In *CVPR*, pages 532–539. IEEE, 2013.

[106] Zhanpeng Zhang, Ping Luo, Chen C. Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *ECCV*, 2014.

[107] Jiangjing Lv, Xiaohu Shao, Junliang Xing, Cheng Cheng, and Xi Zhou. A deep regression architecture with two-stage re-initialization for high performance facial landmark detection. In *CVPR*, 2017.

[108] Stefanos Zafeiriou, George Trigeorgis, Grigorios Chrysos, Jiankang Deng, and Jie Shen. The menpo facial landmark localisation challenge: A step towards the solution. In *CVPRW*, 2017.

[109] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv*, 2017.

[110] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. *arXiv*, 2017.

[111] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv*, 2015.

[112] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv*, 2016.

[113] Xavier Gastaldi. Shake-shake regularization. *arXiv*, 2017.

[114] Yoshihiro Yamada, Masakazu Iwamura, Takuya Akiba, and Koichi Kise. Shakedrop regularization for deep residual learning. *arXiv*, 2018.

[115] Christos Sagonas, Georgios Tzimiropoulos, Stefanos Zafeiriou, and Maja Pantic. 300 faces in-the-wild challenge: The first facial landmark localization challenge. In *ICCVW*, pages 397–403. IEEE, 2013.

[116] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *Icdar*, volume 3, 2003.

[117] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, pages 1097–1105, 2012.

[118] Terrance DeVries and Graham W Taylor. Dataset augmentation in feature space. *arXiv preprint arXiv:1702.05538*, 2017.

[119] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[120] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron, 2018.

[121] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *NeurIPS*, 2016.

[122] Jun Fu, Jing Liu, Haijie Tian, Yong Li, Yongjun Bao, Zhiwei Fang, and Hanqing Lu. Dual attention network for scene segmentation. In *CVPR*, pages 3146–3154, 2019.

[123] Maoke Yang, Kun Yu, Chi Zhang, Zhiwei Li, and Kuiyuan Yang. Denseaspp for semantic segmentation in street scenes. In *CVPR*, 2018.

[124] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*, pages 5754–5764, 2019.

[125] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[126] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[127] Barret Zoph, Ekin D Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, and Quoc V Le. Learning data augmentation strategies for object detection. *arXiv preprint arXiv:1906.11172*, 2019.

[128] Donghoon Lee, Hyunsin Park, Trung Pham, and Chang D. Yoo. Learning augmentation network via influence functions. In *CVPR*, 2020.

[129] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *CVPR*, pages 3642–3649. IEEE, 2012.

[130] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. *arXiv preprint arXiv:1909.13719*, 2019.

[131] Xinyu Zhang, Qiang Wang, Jian Zhang, and Zhao Zhong. Adversarial autoaugment. In *ICLR*, 2019.

[132] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014.

[133] Luke Taylor and Geoff Nitschke. Improving deep learning using generic data augmentation. *arXiv preprint arXiv:1708.06020*, 2017.

[134] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, pages 2672–2680, 2014.

[135] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[136] Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan L Yuille, and Kaiming He. Feature denoising for improving adversarial robustness. In *CVPR*, pages 501–509, 2019.

[137] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[138] Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel. Meta-learning for semi-supervised few-shot classification. *arXiv preprint arXiv:1803.00676*, 2018.

[139] Jonathan Lorraine and David Duvenaud. Stochastic hyperparameter optimization through hypernetworks. *arXiv preprint arXiv:1802.09419*, 2018.

[140] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, pages 1126–1135. JMLR. org, 2017.

[141] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[142] Aditi Raghunathan, Sang Michael Xie, Fanny Yang, John C Duchi, and Percy Liang. Adversarial training can hurt generalization. *arXiv preprint arXiv:1906.06032*, 2019.

[143] Patrick Bilic, Patrick Ferdinand Christ, Eugene Vorontsov, Grzegorz Chlebus, Hao Chen, Qi Dou, Chi-Wing Fu, Xiao Han, Pheng-Ann Heng, Jürgen Hesser, et al. The liver tumor segmentation benchmark (lits). *arXiv preprint arXiv:1901.04056*, 2019.

[144] Geoff Pleiss, Danlu Chen, Gao Huang, Tongcheng Li, Laurens M., and K. Q Weinberger. Memory-efficient implementation of densenets. *arXiv*, 2017.

[145] Rupesh K Srivastava, Klaus Greff, and Jürgen S. Training very deep networks. In *NIPS*, pages 2377–2385, 2015.

[146] Xiaomeng Li, Hao Chen, Xiaojuan Qi, Qi Dou, Chi-Wing Fu, and Pheng Ann Heng. H-denseunet: Hybrid densely connected unet for liver and liver tumor segmentation from ct volumes. *arXiv*, 2017.

[147] Simon Jégou, Michal Drozdzal, David Vazquez, Adriana Romero, and Yoshua Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *CVPRW*, pages 1175–1183. IEEE, 2017.

[148] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. In *CVPR*, 2018.

[149] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv*, 2016.

[150] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *arXiv*, 2016.

[151] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv*, 2016.

[152] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. Training and inference with integers in deep neural networks. In *ICLR*, 2018.

[153] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

[154] Adrian Bulat and Georgios Tzimiropoulos. Binarized convolutional landmark localizers for human pose estimation and face alignment with limited resources. In *ICCV*, 2017.

[155] Long Zhao, Xi Peng, Yu Tian, Mubbasir Kapadia, and Dimitris Metaxas. Learning to forecast and refine residual motion for image-to-video generation. In *ECCV*, 2018.

[156] Feng Liu, Qijun Zhao, Xiaoming Liu, and Dan Zeng. Joint face alignment and 3d face reconstruction with application to face recognition. *ArXiv*, 2017.

[157] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep convolutional network cascade for facial point detection. In *CVPR*, 2013.

[158] Jie Zhang, Shiguang Shan, Meina Kan, and Xilin Chen. Coarse-to-fine auto-encoder networks (cfan) for real-time face alignment. In *ECCV*, 2014.

[159] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv*, 2015.

[160] Xavier Glorot, Antoine B., and Yoshua B. Deep sparse rectifier neural networks. In *AISTAT*, pages 315–323, 2011.

[161] Ron Begleiter, Ran El-Yaniv, and Golan Yona. On prediction using variable order markov models. *Journal of Artificial Intelligence Research*, 2004.

[162] Shizhan Zhu, Cheng Li, Chen Change Loy, and Xiaoou Tang. Face alignment by coarse-to-fine shape searching. In *CVPR*, 2015.

[163] Baoguang Shi, Xiang Bai, W. Liu, and J. Wang. Deep regression face alignment. *arXiv*, 2014.

[164] Xiang Yu, Feng Zhou, and Manmohan Chandraker. Deep deformation network for object landmark localization. In *ECCV*, pages 52–70. Springer, 2016.

[165] George Trigeorgis, Patrick Snape, Mihalis A Nicolaou, E. A., and S. Z. Mnemonic descent method: A recurrent process applied for end-to-end face alignment. In *CVPR*, 2016.

[166] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[167] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[168] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[169] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal style transfer via feature transforms. In *Advances in neural information processing systems*, pages 386–396, 2017.

[170] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6924–6932, 2017.

[171] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1501–1510, 2017.

[172] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. In *Proceedings of the International Conference on Learning Representations*, 2019.

[173] Dan Hendrycks, Norman Mu, Ekin D Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan. Augmix: A simple data processing method to improve robustness and uncertainty. In *Proceedings of the International Conference on Learning Representations*, 2020.

[174] Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, et al. The many faces of robustness: A critical analysis of out-of-distribution generalization. *arXiv preprint arXiv:2006.16241*, 2020.

[175] Evgenia Rusak, Lukas Schott, Roland S Zimmermann, Julian Bitterwolf, Oliver Bringmann, Matthias Bethge, and Wieland Brendel. A simple way to make neural networks robust against diverse image corruptions. *arXiv preprint arXiv:2001.06057*, 2020.

[176] Steffen Schneider, Evgenia Rusak, Luisa Eck, Oliver Bringmann, Wieland Brendel, and Matthias Bethge. Improving robustness against common corruptions by covariate shift adaptation. *Advances in Neural Information Processing Systems*, 33, 2020.

[177] Xingang Pan, Ping Luo, Jianping Shi, and Xiaoou Tang. Two at once: Enhancing learning and generalization capacities via ibn-net. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 464–479, 2018.

[178] Xiangyu Yue, Yang Zhang, Sicheng Zhao, Alberto Sangiovanni-Vincentelli, Kurt Keutzer, and Boqing Gong. Domain randomization and pyramid consistency:

Simulation-to-real generalization without accessing target domain data. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2100–2110, 2019.

[179] Dan Hendrycks, Xiaoyuan Liu, Eric Wallace, Adam Dziedzic, Rishabh Krishnan, and Dawn Song. Pretrained transformers improve out-of-distribution robustness. *arXiv preprint arXiv:2004.06100*, 2020.

[180] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[181] Lucas Deecke, Iain Murray, and Hakan Bilen. Mode normalization. *arXiv preprint arXiv:1810.05466*, 2018.

[182] Xilai Li, Wei Sun, and Tianfu Wu. Attentive normalization. *arXiv preprint arXiv:1908.01259*, 2019.

[183] Ruimao Zhang, Zhanglin Peng, Lingyun Wu, Zhen Li, and Ping Luo. Exemplar normalization for learning deep representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12726–12735, 2020.

[184] Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, and Yun Fu. Image super-resolution using very deep residual channel attention networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 286–301, 2018.

[185] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *International Conference on Computer Vision (ICCV)*, 2019.

[186] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. *arXiv preprint arXiv:1610.07629*, 2016.

[187] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011.

[188] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[189] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[190] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[191] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[192] Kihyuk Sohn, David Berthelot, Chun-Liang Li, Zizhao Zhang, Nicholas Carlini, Ekin D Cubuk, Alex Kurakin, Han Zhang, and Colin Raffel. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *arXiv preprint arXiv:2001.07685*, 2020.

[193] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[194] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[195] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5188–5196, 2015.