©Copyright 2019
Junyuan Xie

Transfer Learning with Deep Neural Networks for Computer Vision

Junyuan Xie

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2019

Reading Committee:

Ali Farhadi, Chair

Hannaneh Hajishirzi

Linda Shapiro

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Transfer Learning with Deep Neural Networks for Computer Vision

Junyuan Xie

Chair of the Supervisory Committee:
Associate Professor Ali Farhadi
Computer Science and Engineering

Deep Neural Networks (DNNs) have played a major role in advancing computer vision research in the past years. DNNs are especially effective for tasks where large amount of labeled data is available. However, for many tasks such as object detection and semantic segmentation, labeled data is expansive to acquire. In such cases, it is beneficial to apply transfer learning and leverage data from another domain where labels are cheaper to collect.

Transfer learning often involves two stages: pre-training on a source task with a large amount of data and fine-tuning on the target task with relatively less data. In this thesis, we first study the process of training Convolutional Neural Netowrks (CNNs) for image classification, which is the most widely used source task in computer vision. We examine each step in this process in detail and propose various modifications that improve model accuracy on the source task. Next, we fine-tune the improved source model on target tasks to show that these improvements on the source task can be transferred to improvements on target tasks.

In the rest of this thesis, we present our works on transfer learning in various application domains including clustering, automatic 2D-to-3D conversion, and object detection. We demonstrate how transfer learning, in different forms, helps improving performance on the target task by leveraging other datasets and source tasks.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

CNN: a class of neural networks composed of convolution operators. Commonly used for image data.

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to Professor Ali Farhadi and Linda Shapiro for their advise and mentoring. He is also thankful to many fellow students at University of Washington for their kind help and inspiring discussions.

Chapter 1

INTRODUCTION

In recent years, deep neural networks (DNNs) have played a key role in advancing machine learning research. DNNs based methods have been established as the state-of-the-art in many domains. For example, in 2010 Deng *et al*. proposed to replace traditional hand-engineered features with learned features from DNNs and significantly improved speech recognition accuracy [128]. DNNs are now the dominate approach to speech recognition [127]. In 2012, Krizhevsky *et al*. won the ImageNet competition on image classification. Since then, numerous follow up works have been published and the state-of-the-art performance on image classification has been rising rapidly.

However, successful application of DNN based methods often requires a large amount of labeled data. This has not been a impediment for image classification task because of the availability of large scale public datasets like ImageNet [20], which has millions of images. Similarly, in speech recognition companies have accumulated tens of thousands of hours of speech data for training [4]. In many other applications, however, labeled data can be much more expansive and scarce. For example, semantic segmentation requires a category label for each pixel in the image, which is obviously much more expansive than the per image label needed for image classification. As a result, datasets for these tasks are typically much smaller, with only hundreds or thousands of examples.

To adapt DNNs to small datasets, many researchers have turned to transfer learning. The goal of transfer learning is to improve performance on the target task by leveraging a large amount of training data from another task, which we called the source task. In computer vision, the most popular source task is image classification. Indeed, state-of-the-art semantic segmantation [136, 131, 75] models and object detection models are all derived

from backbone networks pre-trained on ImageNet.

1.1 Background

In this thesis, we study the intersection of three fields: transfer learning, deep neural networks, and computer vision. We study how to best apply DNNs for various computer vision tasks as well as how transfer learning can be used to improve performance.

1.1.1 Deep Learning

Deep learning is a branch of machine learning that promotes stacking many processing layers to learn representations at multiple level of abstraction. The most widely used instantiation of deep learning is deep neural networks, which has been shown to be effective in many domains including speech recognition, image classification, object detection, and natural language processing [64].

1.1.2 Computer Vision

Computer vision research aims to build algorithms that can generate high-level understanding from images and videos. In recent years, deep learning has become the most successful approach to many computer vision tasks. For example, AlexNet [59], now succeeded by many follow up works, proposed to use a network of multiple stacked convolution and non-linearity layers to classify images and dramatically out-performed previous methods. Faster-RCNN [96] detect object locations in images by combining convolution layers with non-maximum suppression and other operations. Deep learning allows the combination of many different operations to fit the specific needs of different computer vision tasks. The combined system can be trained end-to-end with back-propagation, which often yields better results than traditional systems with independently trained stages.

1.1.3 Transfer Learning

Transfer learning aims at improving learning on a new task, *i.e.* the target task, by leveraging knowledge learned from a related but different task, *i.e.* the source task. Many deep learning methods use transfer learning to take advantage of additional data outside the target domain, especially when data on the target task is scarce. For example, datasets for object detection and semantic segmentation are usually orders of magnitudes smaller than image classification datasets. Therefore, object detection and semantic segmentation algorithms are often built on top of models pretrained on image classification tasks [96, 12].

Transfer learning can be used in many forms. In [47], Hinton*et al.* proposed deep belief networks, which first pretrains on unlabeled data and then fine-tunes on labeled data. With semi-supervised learning, models are simultaneously train on labeled data and unlabeled data. With multi-task learning, a single model is trained to perform multiple tasks at the same time. There are many other approaches to transfer learning designed for various types of data. Please refer to [110] for a survey of this field.

1.2 Thesis Outline

The reset of this thesis is organized as follows:

We start by focusing on the first stage of transfer learning: pre-training on the source task. In Chapter 2, we study how can performance on the source task be improved and what is the relationship between improvements on the source task and improvements on the target task. To this end, we experiment with a combination of techniques for improving image classification accuracy on ImageNet, which is the most widely used source task in computer vision. We also empirically validate if these improvements can be transferred to target tasks including object detection and semantic segmentation.

In the following chapters, we moved to the second stage of transfer learning: fine-tuning on the target task given a pre-trained model. In Chapter 3, we describe deep embedded clustering (DEC), a clustering algorithm that combines unsupervised pre-training and self-

supervised fine-tuning. DEC first uses unsupervised autoendocers to learn a mapping from the data space to a lower-dimensional feature space and then iteratively refines the mapping with a clustering objective. Our experiments on image and text corpora show that DEC brings significant improvement over previous state-of-the-art clustering methods.

In Chapter 4, we present Deep3D, a 2D-to-3D video conversion algorithm. Deep3D is trained to minimize the pixel-wise reconstruction error of the right view when given the left view. We take a backbone network pre-trained for classifying 2D images and fine-tune it on stereo image pairs for 3D reconstruction. Deep3D outperforms baselines in both quantitative and human subject evaluations.

In Chapter 5, we focus on object detection and study various factors that affect detection accuracy. In particular, we adapt mixup [132] to object detection and show that it helps to improve generalization and avoid confusion caused by context. We also demonstrate that in order to attain the best effect of mixup training in detection, the source model also needs to be pre-trained with mixup on image classification.

Finally, Chapter 6 concludes this thesis.

Chapter 2

IMAGE CLASSIFICATION AND TRANSFER LEARNING

The cost of labeling data varies greatly among common computer vision tasks. For example, image classification only requires one categorical label per image, while object detection requires the precise location of all objects in the scene. As a result, there are much less data for object detection than image classification. Transfer learning is often used to leverage large datasets from other tasks, *i.e.* the source task, for improving performance on target tasks with scarce data. Image classification is the most widely used source task in computer vision because there is a large amount of labeled data for it.

Transfer learning consists of two stages: pre-training on the source task and fine-tuning on the target task. In this chapter, we focus on the first stage while holding the second stage constant. We study how performance on the source task can be improved and whether improvements on the source task leads to improvements on the target task. Specifically, We experiment with a combination of techniques for improving image classification accuracy on ImageNet. We also demonstrate that these improvements transfers to improvements on target tasks including object detection and semantic segmentation.

2.1 Introduction

Since the introduction of AlexNet [59] in 2012, deep convolutional neural networks have become the dominating approach for image classification. Various new network architectures have been proposed since then, including VGG [102], NiN [69], Inception [13], ResNet [42], DenseNet [53], and NASNet [141]. At the same time, we have seen a steady trend of model accuracy improvement. For example, the top-1 validation accuracy on ImageNet [98] has been raised from 62.5% (AlexNet) to 82.7% (NASNet-A).

Model	FLOPs	top-1	top-5
ResNet-50 [42]	3.9 G	75.3	92.2
ResNeXt-50 [122]	4.2 G	77.8	-
SE-ResNet-50 [52]	3.9 G	76.71	93.38
SE-ResNeXt-50 [52]	4.3 G	78.90	94.51
DenseNet-201 [53]	4.3 G	77.42	93.66
ResNet-50 + tricks (ours)	4.3 G	**79.29**	**94.63**

Table 2.1: **Computational costs and validation accuracy of various models.** ResNet, trained with our "tricks", is able to outperform newer and improved architectures trained with standard pipeline.

However, these advancements did not solely come from improved model architecture. Training procedure refinements, including changes in loss functions, data preprocessing, and optimization methods also played a major role [43]. A large number of such refinements has been proposed in the past years but received relatively less attention. In the literature, some of them were only briefly mentioned as implementation details while others can only be found in source code.

In this chapter, we examine a collection of training procedure and model architecture refinements that improve model accuracy but barely change computational complexity. Many of them are minor "tricks" like modifying the stride size of a particular convolution layer or adjusting learning rate schedule. Collectively, however, they make a big difference. We will evaluate them on multiple network architectures and datasets and report their impact to the final model accuracy.

Our empirical evaluation shows that several tricks lead to significant accuracy improvement and combining them together can further boost the model accuracy. We compare ResNet-50, after applying all tricks, to other related networks in Table 2.1. Note that

these tricks raises ResNet-50's top-1 validation accuracy from 75.3% to 79.29% on ImageNet. It also outperforms other more recent and improved network architectures, such as SE-ResNeXt-50. In addition, we show that our approach can generalize to other networks (Inception V3 [13] and MobileNet [51]) and datasets (Place365 [138]). We further show that models trained with our tricks bring better transfer learning performance in other application domains such as object detection and semantic segmentation.

Chapter Outline. We first set up a baseline training procedure in Section 2.2, and then discuss several tricks that are useful for efficient training on new hardware in Section 2.3. In Section 2.4 we review three minor model architecture tweaks for ResNet and propose a new one. Four additional training procedure refinements are then discussed in Section 2.5. At last, we study if these more accurate models can help transfer learning in Section 2.6.

Our model implementations and training scripts are publicly available in GluonCV [1].

2.2 Training Procedures

The template of training a neural network with mini-batch stochastic gradient descent is shown in Algorithm 1. In each iteration, we randomly sample b images to compute the gradients and then update the network parameters. It stops after K passes through the dataset. All functions and hyper-parameters in Algorithm 1 can be implemented in many different ways. In this section, we first specify a baseline implementation of Algorithm 1.

2.2.1 Baseline Training Procedure

We follow a widely used implementation [39] of ResNet as our baseline. The preprocessing pipelines between training and validation are different. During training, we perform the following steps one-by-one:

[1] `https://github.com/dmlc/gluon-cv`

Algorithm 1 Train a neural network with mini-batch stochastic gradient descent.

```
initialize(net)
for epoch = 1, ..., K do
    for batch = 1, ..., #images/b do
        images ← uniformly random sample b images
        X, y ← preprocess(images)
        z ← forward(net, X)
        ℓ ← loss(z, y)
        grad ← backward(ℓ)
        update(net, grad)
    end for
end for
```

1. Randomly sample an image and decode it into 32-bit floating point raw pixel values in $[0, 255]$.

2. Randomly crop a rectangular region whose aspect ratio is randomly sampled in $[3/4, 4/3]$ and area randomly sampled in $[8\%, 100\%]$, then resize the cropped region into a 224-by-224 square image.

3. Flip horizontally with 0.5 probability.

4. Scale hue, saturation, and brightness with coefficients uniformly drawn from $[0.6, 1.4]$.

5. Add PCA noise with a coefficient sampled from a normal distribution $\mathcal{N}(0, 0.1)$.

6. Normalize RGB channels by subtracting 123.68, 116.779, 103.939 and dividing by 58.393, 57.12, 57.375, respectively.

During validation, we resize each image's shorter edge to 256 pixels while keeping its aspect ratio. Next, we crop out the 224-by-224 region in the center and normalize RGB

channels similar to training. We do not perform any random augmentations during validation.

The weights of both convolutional and fully-connected layers are initialized with the Xavier algorithm [35]. In particular, we set the parameter to random values uniformly drawn from $[-a, a]$, where $a = \sqrt{6/(d_{in} + d_{out})}$. Here d_{in} and d_{out} are the input and output channel sizes, respectively. All biases are initialized to 0. For batch normalization layers, γ vectors are initialized to 1 and β vectors to 0.

Nesterov Accelerated Gradient (NAG) descent [87] is used for training. Each model is trained for 120 epochs on 8 Nvidia V100 GPUs with a total batch size of 256. The learning rate is initialized to 0.1 and divided by 10 at the 30th, 60th, and 90th epochs.

2.2.2 Experiment Results

We evaluate three CNNs: ResNet-50 [42], Inception-V3 [13], and MobileNet [51]. For Inception-V3 we resize the input images into 299x299. We use the ISLVRC2012 [98] dataset, which has 1.3 million images for training and 1000 classes. The validation accuracies are shown in Table 2.2. As can be seen, our ResNet-50 results are slightly better than the reference results, while our baseline Inception-V3 and MobileNet are slightly lower in accuracy due to different training procedure.

2.3 Efficient Training

Hardware, especially GPUs, has been rapidly evolving in recent years. As a result, the optimal choices for many performance related trade-offs have changed. For example, it is now more efficient to use lower numerical precision and larger batch sizes during training. In this section, we review various techniques that enable low precision and large batch training without sacrificing model accuracy. Some techniques can even improve both accuracy and training speed.

Model	Baseline		Reference	
	Top-1	Top-5	Top-1	Top-5
ResNet-50 [42]	75.87	92.70	75.3	92.2
Inception-V3 [108]	77.32	93.43	78.8	94.4
MobileNet [51]	69.03	88.71	70.6	-

Table 2.2: **Validation accuracy of reference implementations and our baseline.** Note that the numbers for Inception V3 are obtained with 299-by-299 input images.

2.3.1 Large-batch training

Mini-batch SGD groups multiple samples to a mini-batch to increase parallelism and decrease communication costs. Using large batch size, however, may slow down the training progress. For convex problems, convergence rate decreases as batch size increases. Similar empirical results have been reported for neural networks [104]. In other words, for the same number of epochs, training with a large batch size results in a model with degraded validation accuracy compared to the ones trained with smaller batch sizes.

Multiple works [37, 55] have proposed heuristics to solve this issue. In the following paragraphs, we will examine four heuristics that help scale the batch size up for single machine training.

Linear scaling learning rate. In mini-batch SGD, gradient descending is a random process because the examples are randomly selected in each batch. Increasing the batch size does not change the expectation of the stochastic gradient but reduces its variance. In other words, a large batch size reduces the noise in the gradient, so we may increase the learning rate to make a larger progress along the opposite of the gradient direction. Goyal *et al.* [37] reports that linearly increasing the learning rate with the batch size works empirically for ResNet-50 training. In particular, if we follow He *et al.* [42] to choose 0.1 as the initial

learning rate for batch size 256, then when changing to a larger batch size b, we will increase the initial learning rate to $0.1 \times b/256$.

Learning rate warmup. At the beginning of the training, all parameters are typically random values and therefore far away from the final solution. Using a too large learning rate may result in numerical instability. In the warmup heuristic, we use a small learning rate at the beginning and then switch back to the initial learning rate when the training process is stable [42]. Goyal *et al*. [37] proposes a gradual warmup strategy that increases the learning rate from 0 to the initial learning rate linearly. In other words, assume we will use the first m batches (e.g. 5 data epochs) to warm up, and the initial learning rate is η, then at batch i, $1 \leq i \leq m$, we will set the learning rate to be $i\eta/m$.

Zero γ. A ResNet network consists of multiple residual blocks, each block consists of several convolutional layers. Given input x, assume block(x) is the output for the last layer in the block, this residual block then outputs $x + \text{block}(x)$. Note that the last layer of a block could be a batch normalization (BN) layer. The BN layer first standardizes its input, denoted by $\hat{x}$, and then performs a scale transformation $\gamma\hat{x} + \beta$. Both γ and β are learnable parameters whose elements are initialized to 1s and 0s, respectively. In the zero γ initialization heuristic, we initialize $\gamma = 0$ for all BN layers that sit at the end of a residual block. Therefore, all residual blocks just return their inputs, mimics network that has less number of layers and is easier to train at the initial stage.

No bias decay. The weight decay is often applied to all learnable parameters including both weights and bias. It's equivalent to applying an L2 regularization to all parameters to drive their values towards 0. As pointed out by Jia *et al*. [55], however, it's recommended to only apply the regularization to weights to avoid overfitting. The no bias decay heuristic follows this recommendation, it only applies the weight decay to the weights in convolution and fully-connected layers. Other parameters, including the biases and γ and β in BN layers,

are left unregularized.

Note that LARS [31] offers layer-wise adaptive learning rate and is reported to be effective for extremely large batch sizes (beyond 16K). While in this paper we limit ourselves to methods that are sufficient for single machine training, in which case a batch size no more than 2K often leads to good system efficiency.

2.3.2 Low-precision training

Neural networks are commonly trained with 32-bit floating point (FP32) precision. That is, all numbers are stored in FP32 format and both inputs and outputs of arithmetic operations are FP32 numbers as well. New hardware, however, may have enhanced arithmetic logic unit for lower precision data types. For example, the previously mentioned Nvidia V100 offers 14 TFLOPS in FP32 but over 100 TFLOPS in FP16. As in Table 2.3, the overall training speed is accelerated by 2 to 3 times after switching from FP32 to FP16 on V100.

Despite the performance benefit, a reduced precision has a narrower range that makes results more likely to be out-of-range and then disturb the training progress. Micikevicius *et al.* [81] proposes to store all parameters and activations in FP16 and use FP16 to compute gradients. At the same time, all parameters have an copy in FP32 for parameter updating. In addition, multiplying a scalar to the loss to better align the range of the gradient into FP16 is also a practical solution.

2.3.3 Experiment Results

The evaluation results for ResNet-50 are shown in Table 2.3. Compared to the baseline with batch size 256 and FP32, using a larger 1024 batch size and FP16 reduces the training time for ResNet-50 from 13.3-min per epoch to 4.4-min per epoch. In addition, by stacking all heuristics for large-batch training, the model trained with 1024 batch size and FP16 even slightly increased 0.5% top-1 accuracy compared to the baseline model.

The ablation study of all heuristics is shown in Table 2.4. Increasing batch size from 256 to 1024 by linear scaling learning rate alone leads to a 0.9% decrease of the top-1 accuracy

Model	Efficient			Baseline		
	Time/epoch	Top-1	Top-5	Time/epoch	Top-1	Top-5
ResNet-50	**4.4 min**	**76.21**	**92.97**	13.3 min	75.87	92.70
Inception-V3	**8 min**	**77.50**	**93.60**	19.8 min	77.32	93.43
MobileNet	**3.7 min**	**71.90**	**90.47**	6.2 min	69.03	88.71

Table 2.3: Comparison of the training time and validation accuracy for ResNet-50 between the baseline (BS=256 with FP32) and a more hardware efficient setting (BS=1024 with FP16).

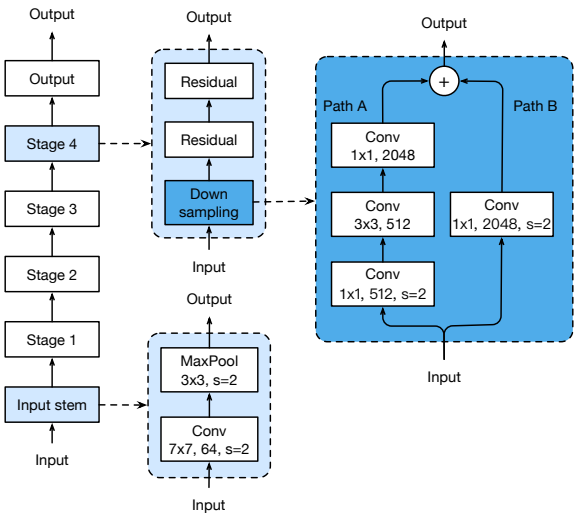


Figure 2.1: The architecture of ResNet-50. The convolution kernel size, output channel size and stride size (default is 1) are illustrated, similar for pooling layers.

while stacking the rest three heuristics bridges the gap. Switching from FP32 to FP16 at the end of training does not affect the accuracy.

Heuristic	BS=256		BS=1024	
	Top-1	Top-5	Top-1	Top-5
Linear scaling	75.87	92.70	75.17	92.54
+ LR warmup	76.03	92.81	75.93	92.84
+ Zero γ	76.19	93.03	76.37	92.96
+ No bias decay	76.16	92.97	76.03	92.86
+ FP16	76.15	93.09	76.21	92.97

Table 2.4: The breakdown effect for each effective training heuristic on ResNet-50.

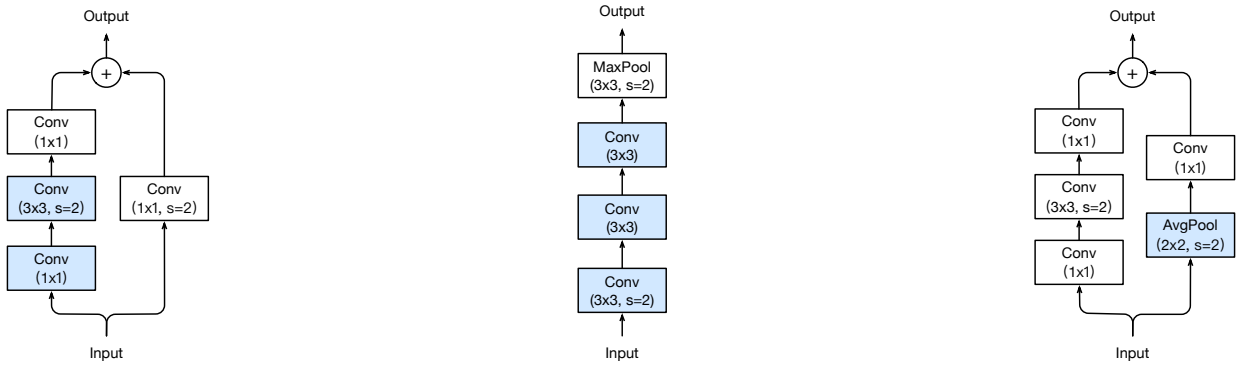
2.4 Model Tweaks

A model tweak is a minor adjustment to the network architecture, such as changing the stride of a particular convolution layer. Such a tweak often barely changes the computational complexity but might have a non-negligible effect on the model accuracy. In this section, we will use ResNet as an example to investigate the effects of model tweaks.

2.4.1 ResNet Architecture

We will briefly present the ResNet architecture, especially its modules related to the model tweaks. For detailed information please refer to He *et al*. [42]. A ResNet network consists of an input stem, four subsequent stages and a final output layer, which is illustrated in Figure 2.1. The input stem has a 7×7 convolution with an output channel of 64 and a stride of 2, followed by a 3×3 max pooling layer also with a stride of 2. The input stem reduces the input width and height by 4 times and increases its channel size to 64.

Starting from stage 2, each stage begins with a downsampling block, which is then followed by several residual blocks. In the downsampling block, there are path A and path B. Path A has three convolutions, whose kernel sizes are 1×1, 3×3 and 1×1, respectively.



(a) ResNet-B (b) ResNet-C (c) ResNet-D

Figure 2.2: Three ResNet tweaks. ResNet-B modifies the downsampling block of Resnet. ResNet-C further modifies the input stem. On top of that, ResNet-D again modifies the downsampling block.

The first convolution has a stride of 2 to halve the input width and height, and the last convolution's output channel is 4 times larger than the previous two, which is called the bottleneck structure. Path B uses a 1×1 convolution with a stride of 2 to transform the input shape to be the output shape of path A, so we can sum outputs of both paths to obtain the output of the downsampling block. A residual block is similar to a downsampling block except for only using convolutions with a stride of 1.

One can vary the number of residual blocks in each stage to obtain different ResNet models, such as ResNet-50 and ResNet-152, where the number presents the number of convolutional layers in the network.

2.4.2 ResNet Tweaks

Next, we revisit two popular ResNet tweaks, we call them ResNet-B and ResNet-C, respectively. We propose a new model tweak ResNet-D afterwards.

ResNet-B. This tweak first appeared in a Torch implementation of ResNet [39] and then adopted by multiple works [37, 52, 122]. It changes the downsampling block of ResNet. The observation is that the convolution in path A ignores three-quarters of the input feature map because it uses a kernel size 1×1 with a stride of 2. ResNet-B switches the strides size of the first two convolutions in path A, as shown in Figure 2.2a, so no information is ignored. Because the second convolution has a kernel size 3×3, the output shape of path A remains unchanged.

ResNet-C. This tweak was proposed in Inception-v2 [108] originally, and it can be found on the implementations of other models, such as SENet [52], PSPNet [136], DeepLabV3 [13], and ShuffleNetV2 [91]. The observation is that the computational cost of a convolution is quadratic to the kernel width or height. A 7×7 convolution is 5.4 times more expensive than a 3×3 convolution. So this tweak replacing the 7×7 convolution in the input stem with three conservative 3×3 convolutions, which is shown in Figure 2.2b, with the first and second convolutions have their output channel of 32 and a stride of 2, while the last convolution uses a 64 output channel.

ResNet-D. Inspired by ResNet-B, we note that the 1×1 convolution in the path B of the downsampling block also ignores 3/4 of input feature maps, we would like to modify it so no information will be ignored. Empirically, we found adding a 2×2 average pooling layer with a stride of 2 before the convolution, whose stride is changed to 1, works well in practice and impacts the computational cost little. This tweak is illustrated in Figure 2.2c.

2.4.3 Experiment Results

We evaluate ResNet-50 with the three tweaks and settings described in Section 2.3, namely the batch size is 1024 and precision is FP16. The results are shown in Table 2.5.

Suggested by the results, ResNet-B receives more information in path A of the downsampling blocks and improves validation accuracy by around 0.5% compared to ResNet-50.

Model	#params	FLOPs	Top-1	Top-5
ResNet-50	25 M	**3.8 G**	76.21	92.97
ResNet-50-B	25 M	4.1 G	76.66	93.28
ResNet-50-C	25 M	4.3 G	76.87	93.48
ResNet-50-D	25 M	4.3 G	**77.16**	**93.52**

Table 2.5: Compare ResNet-50 with three model tweaks on model size, FLOPs and ImageNet validation accuracy.

Replacing the 7×7 convolution with three 3×3 ones gives another 0.2% improvement. Taking more information in path B of the downsampling blocks improves the validation accuracy by another 0.3%. In total, ResNet-50-D improves ResNet-50 by 1%.

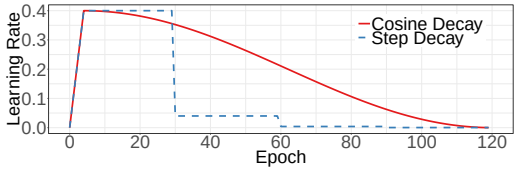
On the other hand, these four models have the same model size. ResNet-D has the largest computational cost, but its difference compared to ResNet-50 is within 15% in terms of floating point operations. In practice, we observed ResNet-50-D is only 3% slower in training throughput compared to ResNet-50.

2.5 Training Refinements

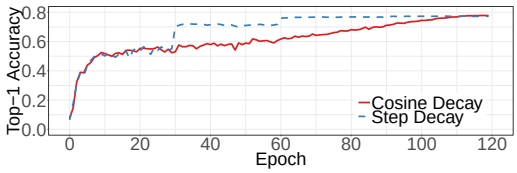
In this section, we will describe four training refinements that aim to further improve the model accuracy.

2.5.1 Cosine Learning Rate Decay

Learning rate adjustment is crucial to the training. After the learning rate warmup described in Section 2.3.1, we typically steadily decrease the value from the initial learning rate. The widely used strategy is exponentially decaying the learning rate. He *et al.* [42] decreases rate at 0.1 for every 30 epochs, we call it "step decay". Szegedy *et al.* [108] decreases rate at 0.94



(a) Learning Rate Schedule



(b) Validation Accuracy

Figure 2.3: Visualization of learning rate schedules with warm-up. Top: cosine and step schedules for batch size 1024. Bottom: Top-1 validation accuracy curve with regard to the two schedules.

for every two epochs.

In contrast to it, Loshchilov *et al.* [76] propose a cosine annealing strategy. An simplified version is decreasing the learning rate from the initial value to 0 by following the cosine function. Assume the total number of batches is T (the warmup stage is ignored), then at batch t, the learning rate η_t is computed as:

$$\eta_t = \frac{1}{2}\left(1 + \cos\left(\frac{t\pi}{T}\right)\right)\eta, \tag{2.1}$$

where η is the initial learning rate. We call this scheduling as "cosine" decay.

The comparison between step decay and cosine decay are illustrated in Figure 2.3a. As can be seen, the cosine decay decreases the learning rate slowly at the beginning, and then becomes almost linear decreasing in the middle, and slows down again at the end. Compared to the step decay, the cosine decay starts to decay the learning since the beginning but remains large until step decay reduces the learning rate by 10x, which potentially improves the training progress.

2.5.2 Label Smoothing

The last layer of a image classification network is often a fully-connected layer with a hidden size being equal to the number of labels, denote by K, to output the predicted confidence scores. Given an image, denote by z_i the predicted score for class i. These scores can be normalized by the softmax operator to obtain predicted probabilities. Denote by q the output of the softmax operator $q = \text{softmax}(z)$, the probability for class i, q_i, can be computed by:

$$q_i = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}. \tag{2.2}$$

It's easy to see $q_i > 0$ and $\sum_{i=1}^{K} q_i = 1$, so q is a valid probability distribution.

On the other hand, assume the true label of this image is y, we can construct a truth probability distribution to be $p_i = 1$ if $i = y$ and 0 otherwise. During training, we minimize the negative cross entropy loss

$$\ell(p, q) = -\sum_{i=1}^{K} q_i \log p_i \tag{2.3}$$

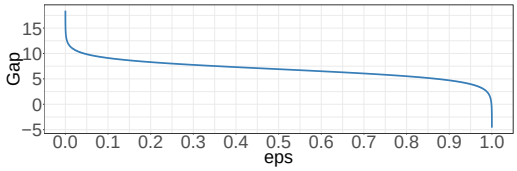
to update model parameters to make these two probability distributions similar to each other. In particular, by the way how p is constructed, we know $\ell(p, q) = -\log p_y = -z_y + \log\left(\sum_{i=1}^{K} \exp(z_i)\right)$. The optimal solution is $z_y^* = \inf$ while keeping others small enough. In other words, it encourages the output scores dramatically distinctive which potentially leads to overfitting.

The idea of label smoothing was first proposed to train Inception-v2 [108]. It changes the construction of the true probability to

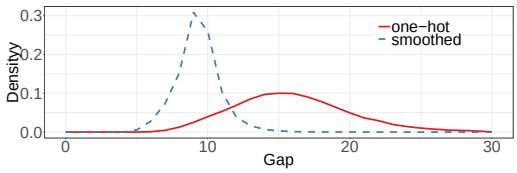
$$q_i = \begin{cases} 1 - \varepsilon & \text{if } i = y, \\ \varepsilon/(K-1) & \text{otherwise,} \end{cases} \tag{2.4}$$

where ε is a small constant. Now the optimal solution becomes

$$z_i^* = \begin{cases} \log((K-1)(1-\varepsilon)/\varepsilon) + \alpha & \text{if } i = y, \\ \alpha & \text{otherwise,} \end{cases} \tag{2.5}$$



(a) Theoretical gap



(b) Empirical gap from ImageNet validation set

Figure 2.4: Visualization of the effectiveness of label smoothing on ImageNet. Top: theoretical gap between z_p^* and others decreases when increasing ε. Bottom: The empirical distributions of the gap between the maximum prediction and the average of the rest.

where α can be an arbitrary real number. This encourages a finite output from the fully-connected layer and can generalize better.

When $\varepsilon = 0$, the gap $\log((K-1)(1-\varepsilon)/\varepsilon)$ will be ∞ and as ε increases, the gap decreases. Specifically when $\varepsilon = (K-1)/K$, all optimal z_i^* will be identical. Figure 2.4a shows how the gap changes as we move ε, given $K = 1000$ for ImageNet dataset.

We empirically compare the output value from two ResNet-50-D models that are trained with and without label smoothing respectively and calculate the gap between the maximum prediction value and the average of the rest. Under $\varepsilon = 0.1$ and $K = 1000$, the theoretical gap is around 9.1. Figure 2.4b demonstrate the gap distributions from the two models predicting over the validation set of ImageNet. It is clear that with label smoothing the distribution centers at the theoretical value and has fewer extreme values.

2.5.3 Knowledge Distillation

In knowledge distillation [46], we use a teacher model to help train the current model, which is called the student model. The teacher model is often a pre-trained model with higher accuracy, so by imitation, the student model is able to improve its own accuracy while keeping the model complexity the same. One example is using a ResNet-152 as the teacher model to help training ResNet-50.

During training, we add a distillation loss to penalize the difference between the softmax outputs from the teacher model and the learner model. Given an input, assume p is the true probability distribution, and z and r are outputs of the last fully-connected layer of the student model and the teacher model, respectively. Remember previously we use a negative cross entropy loss $\ell(p, \text{softmax}(z))$ to measure the difference between p and z, here we use the same loss again for the distillation. Therefore, the loss is changed to

$$\ell(p, \text{softmax}(z)) + T^2 \ell(\text{softmax}(r/T), \text{softmax}(z/T)), \tag{2.6}$$

where T is the temperature hyper-parameter to make the softmax outputs smoother thus distill the knowledge of label distribution from teacher's prediction.

2.5.4 Mixup Training

In Section 2.2.1 we described how images are augmented before training. Here we consider another augmentation method called mixup [133]. In mixup, each time we randomly sample two examples (x_i, y_i) and (x_j, y_j). Then we form a new example by a weighted linear interpolation of these two examples:

$$\hat{x} = \lambda x_i + (1 - \lambda) x_j, \tag{2.7}$$

$$\hat{y} = \lambda y_i + (1 - \lambda) y_j, \tag{2.8}$$

where $\lambda \in [0, 1]$ is a random number drawn from the $\mathbf{Beta}(\alpha, \alpha)$ distribution. In mixup

Refinements	ResNet-50-D		Inception-V3		MobileNet	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Efficient	77.16	93.52	77.50	93.60	71.90	90.53
+ cosine decay	77.91	93.81	78.19	94.06	72.83	91.00
+ label smoothing	78.31	94.09	78.40	94.13	72.93	91.14
+ distill w/o mixup	78.67	94.36	78.26	94.01	71.97	90.89
+ mixup w/o distill	79.15	94.58	**78.77**	**94.39**	**73.28**	**91.30**
+ distill w/ mixup	**79.29**	**94.63**	78.34	94.16	72.51	91.02

Table 2.6: The validation accuracies on ImageNet for stacking training refinements one by one. The baseline models are obtained from Section 2.3.

training, we only use the new example $(\hat{x}, \hat{y})$.

2.5.5 Experiment Results

Now we evaluate the four training refinements. We set $\varepsilon = 0.1$ for label smoothing by following Szegedy *et al.* [108]. For the model distillation we use $T = 20$, specifically a pretrained ResNet-152-D model with both cosine decay and label smoothing applied is used as the teacher. In the mixup training, we choose $\alpha = 0.2$ in the Beta distribution and increase the number of epochs from 120 to 200 because the mixed examples ask for a longer training progress to converge better. When combining the mixup training with distillation, we train the teacher model with mixup as well.

We demonstrate that the refinements are not only limited to ResNet architecture or the ImageNet dataset. First, we train ResNet-50-D, Inception-V3 and MobileNet on ImageNet dataset with refinements. The validation accuracies for applying these training refinements one-by-one are shown in Table 2.6. By stacking cosine decay, label smoothing and mixup, we have steadily improving ResNet, InceptionV3 and MobileNet models. Distillation works

Model	Val Top-1 Acc	Val Top-5 Acc	Test Top-1 Acc	Test Top-5 Acc
ResNet-50-D Efficient	56.34	86.87	57.18	87.28
ResNet-50-D Best	**56.70**	**87.33**	**57.63**	**87.82**

Table 2.7: Results on both the validation set and the test set of MIT Places 365 dataset. Prediction are generated as stated in Section 2.2.1. ResNet-50-D Efficient refers to ResNet-50-D trained with settings from Section 2.3, and ResNet-50-D Best further incorporate cosine scheduling, label smoothing and mixup.

well on ResNet, however, it does not work well on Inception-V3 and MobileNet. Our interpretation is that the teacher model is not from the same family of the student, therefore has different distribution in the prediction, and brings negative impact to the model.

To support our tricks is transferable to other dataset, we train a ResNet-50-D model on MIT Places365 dataset with and without the refinements. Results are reported in Table 2.7. We see the refinements improve the top-5 accuracy consistently on both the validation and test set.

2.6 Transfer Learning

Transfer learning is one major down-streaming use case of trained image classification models. In this section, we will investigate if these improvements discussed so far can benefit transfer learning. In particular, we pick two important computer vision tasks, object detection and semantic segmentation, and evaluate their performance by varying base models.

2.6.1 Object Detection

The goal of object detection is to locate bounding boxes of objects in an image. We evaluate performance using PASCAL VOC [25]. Similar to Ren *et al*. [96], we use union set of VOC

Refinement	Top-1	mAP
B-standard	76.14	77.54
D-efficient	77.16	78.30
+ cosine	77.91	79.23
+ smooth	78.34	80.71
+ distill w/o mixup	78.67	80.96
+ mixup w/o distill	79.16	81.10
+ distill w/ mixup	79.29	**81.33**

Table 2.8: Faster-RCNN performance with various pre-trained base networks evaluated on Pascal VOC.

2007 *trainval* and VOC 2012 *trainval* for training, and VOC 2007 test for evaluation, respectively. We train Faster-RCNN [96] on this dataset, with refinements from Detectron [34] such as linear warmup and long training schedule. The VGG-19 base model in Faster-RCNN is replaced with various pretrained models in the previous discussion. We keep other settings the same so the gain is solely from the base models.

Mean average precision (mAP) results are reported in Table 2.8. We can observe that a base model with a higher validation accuracy leads to a higher mAP for Faster-RNN in a consistent manner. In particular, the best base model with accuracy 79.29% on ImageNet leads to the best mAP at 81.33% on VOC, which outperforms the standard model by 4%.

2.6.2 Semantic Segmentation

Semantic segmentation predicts the category for every pixel from the input images. We use Fully Convolutional Network (FCN) [75] for this task and train models on the ADE20K [139] dataset. Following PSPNet [136] and Zhang *et al.* [131], we replace the base network with various pre-trained models discussed in previous sections and apply dilation network strat-

Refinement	Top-1	PixAcc	mIoU
B-standard	76.14	78.08	37.05
D-efficient	77.16	78.88	38.88
+ cosine	77.91	**79.25**	**39.33**
+ smooth	78.34	78.64	38.75
+ distill w/o mixup	78.67	78.97	38.90
+ mixup w/o distill	79.16	78.47	37.99
+ mixup w/ distill	79.29	78.72	38.40

Table 2.9: FCN performance with various base networks evaluated on ADE20K.

egy [12, 129] on stage-3 and stage-4. A fully convolutional decoder is built on top of the base network to make the final prediction.

Both pixel accuracy (pixAcc) and mean intersection over union (mIoU) are reported in Table 2.9. In contradiction to our results on object detection, the cosine learning rate schedule effectively improves the accuracy of the FCN performance, while other refinements provide suboptimal results. A potential explanation to the phenomenon is that semantic segmentation predicts in the pixel level. While models trained with label smoothing, distillation and mixup favor soften labels, blurred pixel-level information may be blurred and degrade overall pixel-level accuracy.

2.7 Conclusion

In this chapter, we survey a dozen tricks to train deep convolutional neural networks to improve model accuracy. These tricks introduce minor modifications to the model architecture, data preprocessing, loss function, and learning rate schedule. Our empirical results on ResNet-50, Inception-V3 and MobileNet indicate that these tricks improve model accuracy consistently. More excitingly, stacking all of them together leads to a significantly higher

accuracy. In addition, these improved pre-trained models show strong advantages in transfer learning, which improve both object detection and semantic segmentation. We believe the benefits can extend to broader domains where classification base models are favored.

Chapter 3

DEEP EMBEDED CLUSTERING

In this chapter, we describe Deep Embedded Clustering (DEC) [121], a clustering algorithm that combines unsupervised pre-training and self-supervised fine-tuning. DEC first uses unsupervised autoendocers to learn a mapping from the data space to a lower-dimensional feature space and then iteratively refines the mapping with a clustering objective. Our experiments on image and text corpora show that DEC brings significant improvement over previous state-of-the-art clustering methods.

3.1 Introduction

Clustering, an essential data analysis and visualization tool, has been studied extensively in unsupervised machine learning from different perspectives: What defines a cluster? What is the right distance metric? How to efficiently group instances into clusters? How to validate clusters? And so on. Numerous different distance functions and embedding methods have been explored in the literature. Relatively little work has focused on the unsupervised learning of the feature space in which to perform clustering.

A notion of *distance* or *dissimilarity* is central to data clustering algorithms. Distance, in turn, relies on representing the data in a feature space. The k-means clustering algorithm [78], for example, uses the Euclidean distance between points in a given feature space, which for images might be raw pixels or gradient-orientation histograms. The choice of feature space is customarily left as an application-specific detail for the end-user to determine. Yet it is clear that the choice of feature space is crucial; for all but the simplest image datasets, clustering with Euclidean distance on raw pixels is completely ineffective. In this paper, we revisit cluster analysis and ask: *Can we use a data driven approach to solve for the feature space*

and cluster memberships jointly?

We take inspiration from recent work on deep learning for computer vision [59, 33, 130, 74], where clear gains on benchmark tasks have resulted from learning better features. These improvements, however, were obtained with *supervised* learning, whereas our goal is *unsupervised* data clustering. To this end, we define a parameterized non-linear mapping from the data space X to a lower-dimensional feature space Z, where we optimize a clustering objective. Unlike previous work, which operates on the data space or a shallow linear embedded space, we use stochastic gradient descent (SGD) via backpropagation on a clustering objective to learn the mapping, which is parameterized by a deep neural network. We refer to this clustering algorithm as *Deep Embedded Clustering*, or DEC.

Optimizing DEC is challenging. We want to simultaneously solve for cluster assignment and the underlying feature representation. However, unlike in supervised learning, we cannot train our deep network with labeled data. Instead we propose to iteratively refine clusters with an auxiliary target distribution derived from the current soft cluster assignment. This process gradually improves the clustering as well as the feature representation.

Our experiments show significant improvements over state-of-the-art clustering methods in terms of both accuracy and running time on image and textual datasets. We evaluate DEC on MNIST [65], STL [16], and REUTERS [67], comparing it with standard and state-of-the-art clustering methods [89, 125]. In addition, our experiments show that DEC is significantly less sensitive to the choice of hyperparameters compared to state-of-the-art methods. This robustness is an important property of our clustering algorithm since, when applied to real data, supervision is not available for hyperparameter cross-validation.

Our contributions are: (a) joint optimization of deep embedding and clustering; (b) a novel iterative refinement via soft assignment; and (c) state-of-the-art clustering results in terms of clustering accuracy and speed. Our Caffe-based [56] implementation of DEC is available at `https://github.com/piiswrong/dec`.

3.2 Related work

Clustering has been extensively studied in machine learning in terms of feature selection [11, 72, 2], distance functions [123, 119], grouping methods [78, 116, 68], and cluster validation [40]. Space does not allow for a comprehensive literature study and we refer readers to [1] for a survey.

One branch of popular methods for clustering is k-means [78] and Gaussian Mixture Models (GMM) [10]. These methods are fast and applicable to a wide range of problems. However, their distance metrics are limited to the original data space and they tend to be ineffective when input dimensionality is high [106].

Several variants of k-means have been proposed to address issues with higher-dimensional input spaces. [19, 126] perform joint dimensionality reduction and clustering by first clustering the data with k-means and then projecting the data into a lower dimensions where the inter-cluster variance is maximized. This process is repeated in EM-style iterations until convergence. However, this framework is limited to linear embedding; our method employs deep neural networks to perform non-linear embedding that is necessary for more complex data.

Spectral clustering and its variants have gained popularity recently [116]. They allow more flexible distance metrics and generally perform better than k-means. Combining spectral clustering and embedding has been explored in [125, 89]. [109] proposes an algorithm based on spectral clustering, but replaces eigenvalue decomposition with deep autoencoder, which improves performance but further increases memory consumption.

Most spectral clustering algorithms need to compute the full graph Laplacian matrix and therefore have quadratic or super quadratic complexities in the number of data points. This means they need specialized machines with large memory for any dataset larger than a few tens of thousands of points. In order to scale spectral clustering to large datasets, approximate algorithms were invented to trade off performance for speed [124]. Our method, however, is linear in the number of data points and scales gracefully to large datasets.

Minimizing the Kullback-Leibler (KL) divergence between a data distribution and an embedded distribution has been used for data visualization and dimensionality reduction [114]. T-SNE, for instance, is a non-parametric algorithm in this school and a parametric variant of t-SNE [112] uses deep neural network to parametrize the embedding. The complexity of t-SNE is $O(n^2)$, where n is the number of data points, but it can be approximated in $O(n \log n)$ [113].

We take inspiration from parametric t-SNE. Instead of minimizing KL divergence to produce an embedding that is faithful to distances in the original data space, we define a centroid-based probability distribution and minimize its KL divergence to an auxiliary target distribution to simultaneously improve clustering assignment and feature representation. A centroid-based method also has the benefit of reducing complexity to $O(nk)$, where k is the number of centroids.

3.3 Deep embedded clustering

Consider the problem of clustering a set of n points $\{x_i \in X\}_{i=1}^n$ into k clusters, each represented by a centroid $\mu_j, j = 1, \ldots, k$. Instead of clustering directly in the *data space* X, we propose to first transform the data with a non-linear mapping $f_\theta : X \to Z$, where θ are learnable parameters and Z is the latent *feature space*. The dimensionality of Z is typically much smaller than X in order to avoid the "curse of dimensionality" [8]. To parametrize f_θ, deep neural networks (DNNs) are a natural choice due to their theoretical function approximation properties [50] and their demonstrated feature learning capabilities [9].

The proposed algorithm (DEC) clusters data by *simultaneously* learning a set of k cluster centers $\{\mu_j \in Z\}_{j=1}^k$ in the feature space Z and the parameters θ of the DNN that maps data points into Z. DEC has two phases: (1) parameter initialization with a deep autoencoder [115] and (2) parameter optimization (i.e., clustering), where we iterate between computing an auxiliary target distribution and minimizing the Kullback–Leibler (KL) divergence to it. We start by describing phase (2) parameter optimization/clustering, given an initial estimate of θ and $\{\mu_j\}_{j=1}^k$.

3.3.1 Clustering with KL divergence

Given an initial estimate of the non-linear mapping f_θ and the initial cluster centroids $\{\mu_j\}_{j=1}^k$, we propose to improve the clustering using an unsupervised algorithm that alternates between two steps. In the first step, we compute a soft assignment between the embedded points and the cluster centroids. In the second step, we update the deep mapping f_θ and refine the cluster centroids by learning from current high confidence assignments using an auxiliary target distribution. This process is repeated until a convergence criterion is met.

Soft assignment. Following [114] we use the Student's t-distribution as a kernel to measure the similarity between embedded point z_i and centroid μ_j:

$$q_{ij} = \frac{(1 + \|z_i - \mu_j\|^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{j'}(1 + \|z_i - \mu_{j'}\|^2/\alpha)^{-\frac{\alpha+1}{2}}}, \tag{3.1}$$

where $z_i = f_\theta(x_i) \in Z$ corresponds to $x_i \in X$ after embedding, α are the degrees of freedom of the Student's t-distribution and q_{ij} can be interpreted as the probability of assigning sample i to cluster j (i.e., a soft assignment). Since we cannot cross-validate α on a validation set in the unsupervised setting, and learning it is superfluous [112], we let $\alpha = 1$ for all experiments.

KL-divergence minimization. We propose to iteratively refine the clusters by learning from their high confidence assignments with the help of an auxiliary target distribution. Specifically, our model is trained by matching the soft assignment to the target distribution. To this end, we define our objective as a KL divergence loss between the soft assignments q_i and the auxiliary distribution p_i as follows:

$$L = \mathrm{KL}(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{3.2}$$

The choice of target distributions P is crucial for DEC's performance. A naive approach would be setting each p_i to a delta distribution (to the nearest centroid) for data points above a confidence threshold and ignore the rest. However, because q_i are soft assignments, it is

more natural and flexible to use softer probabilistic targets. Specifically, we would like our target distribution to have the following properties: (1) strengthen predictions (i.e., improve cluster purity), (2) put more emphasis on data points assigned with high confidence, and (3) normalize loss contribution of each centroid to prevent large clusters from distorting the hidden feature space.

In our experiments, we compute p_i by first raising q_i to the second power and then normalizing by frequency per cluster:

$$p_{ij} = \frac{q_{ij}^2 / f_j}{\sum_{j'} q_{ij'}^2 / f_{j'}}, \tag{3.3}$$

where $f_j = \sum_i q_{ij}$ are soft cluster frequencies. Please refer to section 3.5.1 for discussions on empirical properties of L and P.

Our training strategy can be seen as a form of self-training [90]. As in self-training, we take an initial classifier and an unlabeled dataset, then label the dataset with the classifier in order to train on its own high confidence predictions. Indeed, in experiments we observe that DEC improves upon the initial estimate in each iteration by learning from high confidence predictions, which in turn helps to improve low confidence ones.

Optimization. We jointly optimize the cluster centers $\{\mu_j\}$ and DNN parameters θ using Stochastic Gradient Descent (SGD) with momentum. The gradients of L with respect to feature-space embedding of each data point z_i and each cluster centroid μ_j are computed as:

$$\begin{aligned} \frac{\partial L}{\partial z_i} &= \frac{\alpha+1}{\alpha} \sum_j (1 + \frac{\|z_i - \mu_j\|^2}{\alpha})^{-1} \\ &\quad \times (p_{ij} - q_{ij})(z_i - \mu_j), \end{aligned} \tag{3.4}$$

$$\begin{aligned} \frac{\partial L}{\partial \mu_j} &= -\frac{\alpha+1}{\alpha} \sum_i (1 + \frac{\|z_i - \mu_j\|^2}{\alpha})^{-1} \\ &\quad \times (p_{ij} - q_{ij})(z_i - \mu_j). \end{aligned} \tag{3.5}$$

The gradients $\partial L / \partial z_i$ are then passed down to the DNN and used in standard backpropagation to compute the DNN's parameter gradient $\partial L / \partial \theta$. For the purpose of discovering

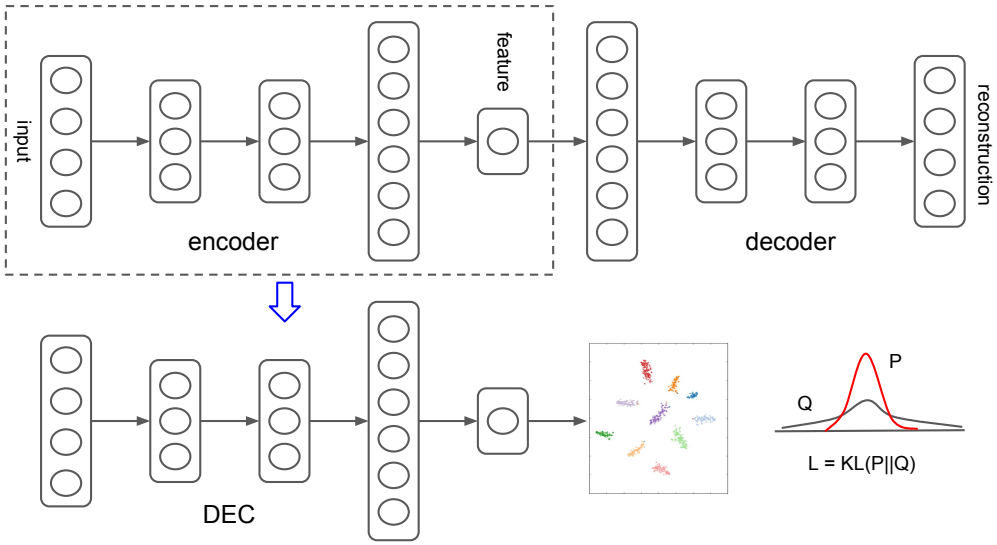


Figure 3.1: Network structure

cluster assignments, we stop our procedure when less than $tol\%$ of points change cluster assignment between two consecutive iterations.

3.3.2 Parameter initialization

Thus far we have discussed how DEC proceeds given initial estimates of the DNN parameters θ and the cluster centroids $\{\mu_j\}$. Now we discuss how the parameters and centroids are initialized.

We initialize DEC with a stacked autoencoder (SAE) because recent research has shown that they consistently produce semantically meaningful and well-separated representations on real-world datasets [115, 48, 63]. Thus the unsupervised representation learned by SAE naturally facilitates the learning of clustering representations with DEC.

We initialize the SAE network layer by layer with each layer being a denoising autoencoder trained to reconstruct the previous layer's output after random corruption [115]. A denoising

autoencoder is a two layer neural network defined as:

$$\tilde{x} \sim \mathit{Dropout}(x) \tag{3.6}$$

$$h = g_1(W_1\tilde{x} + b_1) \tag{3.7}$$

$$\tilde{h} \sim \mathit{Dropout}(h) \tag{3.8}$$

$$y = g_2(W_2\tilde{h} + b_2) \tag{3.9}$$

where $\mathit{Dropout}(\cdot)$ [105] is a stochastic mapping that randomly sets a portion of its input dimensions to 0, g_1 and g_2 are activation functions for encoding and decoding layer respectively, and $\theta = \{W_1, b_1, W_2, b_2\}$ are model parameters. Training is performed by minimizing the least-squares loss $\|x - y\|_2^2$. After training of one layer, we use its output h as the input to train the next layer. We use rectified linear units (ReLUs) [84] in all encoder/decoder pairs, except for g_2 of the *first* pair (it needs to reconstruct input data that may have positive and negative values, such as zero-mean images) and g_1 of the *last* pair (so the final data embedding retains full information [115]).

After greedy layer-wise training, we concatenate all encoder layers followed by all decoder layers, in reverse layer-wise training order, to form a deep autoencoder and then finetune it to minimize reconstruction loss. The final result is a multilayer deep autoencoder with a bottleneck coding layer in the middle. We then discard the decoder layers and use the encoder layers as our initial mapping between the data space and the feature space, as shown in Fig. 3.1.

To initialize the cluster centers, we pass the data through the initialized DNN to get embedded data points and then perform standard k-means clustering in the feature space Z to obtain k initial centroids $\{\mu_j\}_{j=1}^k$.

Table 3.1: Dataset statistics.

Dataset	# Points	# classes	Dimension	% of largest class
MNIST [65]	70000	10	784	11%
STL-10 [16]	13000	10	1428	10%
REUTERS-10K	10000	4	2000	43%
REUTERS [67]	685071	4	2000	43%

3.4 Experiments

3.4.1 Datasets

We evaluate the proposed method (DEC) on one text dataset and two image datasets and compare it against other algorithms including k-means, LDGMI [125], and SEC [89]. LDGMI and SEC are spectral clustering based algorithms that use a Laplacian matrix and various transformations to improve clustering performance. Empirical evidence reported in [125, 89] shows that LDMGI and SEC outperform traditional spectral clustering methods on a wide range of datasets. We show qualitative and quantitative results that demonstrate the benefit of DEC compared to LDGMI and SEC.

In order to study the performance and generality of different algorithms, we perform experiment on two image datasets and one text data set:

- **MNIST**: The MNIST dataset consists of 70000 handwritten digits of 28-by-28 pixel size. The digits are centered and size-normalized [65].

- **STL-10**: A dataset of 96-by-96 color images. There are 10 classes with 1300 examples each. It also contains 100000 unlabeled images of the same resolution [16]. We also used the unlabeled set when training our autoencoders. Similar to [22], we concatenated HOG feature and a 8-by-8 color map to use as input to all algorithms.

- **REUTERS**: Reuters contains about 810000 English news stories labeled with a category tree [67]. We used the four root categories: corporate/industrial, government/social, markets, and economics as labels and further pruned all documents that are labeled by multiple root categories to get 685071 articles. We then computed tf-idf features on the 2000 most frequently occurring word stems. Since some algorithms do not scale to the full Reuters dataset, we also sampled a random subset of 10000 examples, which we call REUTERS-10k, for comparison purposes.

A summary of dataset statistics is shown in Table 3.1. For all algorithms, we normalize all datasets so that $\frac{1}{d}\|x_i\|_2^2$ is approximately 1, where d is the dimensionality of the data space point $x_i \in X$.

3.4.2 Evaluation Metric

We use the standard unsupervised evaluation metric and protocols for evaluations and comparisons to other algorithms [125]. For all algorithms we set the number of clusters to the number of ground-truth categories and evaluate performance with *unsupervised clustering accuracy (ACC)*:

$$ACC = \max_m \frac{\sum_{i=1}^n \mathbf{1}\{l_i = m(c_i)\}}{n}, \tag{3.10}$$

where l_i is the ground-truth label, c_i is the cluster assignment produced by the algorithm, and m ranges over all possible one-to-one mappings between clusters and labels.

Intuitively this metric takes a cluster assignment from an *unsupervised* algorithm and a ground truth assignment and then finds the best matching between them. The best mapping can be efficiently computed by the Hungarian algorithm [60].

3.4.3 Implementation

Determining hyperparameters by cross-validation on a validation set is not an option in unsupervised clustering. Thus we use commonly used parameters for DNNs and avoid dataset specific tuning as much as possible. Specifically, inspired by [112], we set network dimensions

(a) MNIST (b) STL-10

Figure 3.2: Each row contains the top 10 scoring elements from one cluster.

to d–500–500–2000–10 for all datasets, where d is the data-space dimension, which varies between datasets. All layers are densely (fully) connected.

During greedy layer-wise pretraining we initialize the weights to random numbers drawn from a zero-mean Gaussian distribution with a standard deviation of 0.01. Each layer is pretrained for 50000 iterations with a dropout rate of 20%. The entire deep autoencoder is further finetuned for 100000 iterations without dropout. For both layer-wise pretraining and end-to-end finetuning of the autoencoder the minibatch size is set to 256, starting learning rate is set to 0.1, which is divided by 10 every 20000 iterations, and weight decay is set to 0. All of the above parameters are set to achieve a reasonably good reconstruction loss and are held constant across all datasets. Dataset-specific settings of these parameters might improve performance on each dataset, but we refrain from this type of unrealistic parameter tuning. To initialize centroids, we run k-means with 20 restarts and select the best solution. In the KL divergence minimization phase, we train with a constant learning rate of 0.01. The convergence threshold is set to $tol = 0.1\%$. Our implementation is based on Python

Table 3.2: Comparison of clustering accuracy (Eq. 3.10) on four datasets.

Method	MNIST	STL-HOG	REUTERS-10k	REUTERS
k-means	53.49%	28.39%	52.42%	53.29%
LDMGI	84.09%	33.08%	43.84%	N/A
SEC	80.37%	30.75%	60.08%	N/A
DEC w/o backprop	79.82%	34.06%	70.05%	69.62%
DEC (ours)	**84.30%**	**35.90%**	**72.17%**	**75.63%**

and Caffe [56] and is available at `https://github.com/piiswrong/dec`.

For all baseline algorithms, we perform 20 random restarts when initializing centroids and pick the result with the best objective value. For a fair comparison with previous work [125], we vary one hyperparameter for each algorithm over 9 possible choices and report the best accuracy in Table 3.2 and the range of accuracies in Fig. 3.3. For LDGMI and SEC, we use the same parameter and range as in their corresponding papers. For our proposed algorithm, we vary λ, the parameter that controls annealing speed, over $2^i \times 10, i = 0, 1, ..., 8$. Since k-means does not have tunable hyperparameters (aside from k), we simply run them 9 times. GMMs perform similarly to k-means so we only report k-means results. Traditional spectral clustering performs worse than LDGMI and SEC so we only report the latter [125, 89].

3.4.4 Experiment results

We evaluate the performance of our algorithm both quantitatively and qualitatively. In Table 3.2, we report the best performance, over 9 hyperparameter settings, of each algorithm. Note that DEC outperforms all other methods, sometimes with a significant margin. To demonstrate the effectiveness of end-to-end training, we also show the results from freezing the non-linear mapping f_θ during clustering. We find that this ablation ("DEC w/o backprop") generally performs worse than DEC.

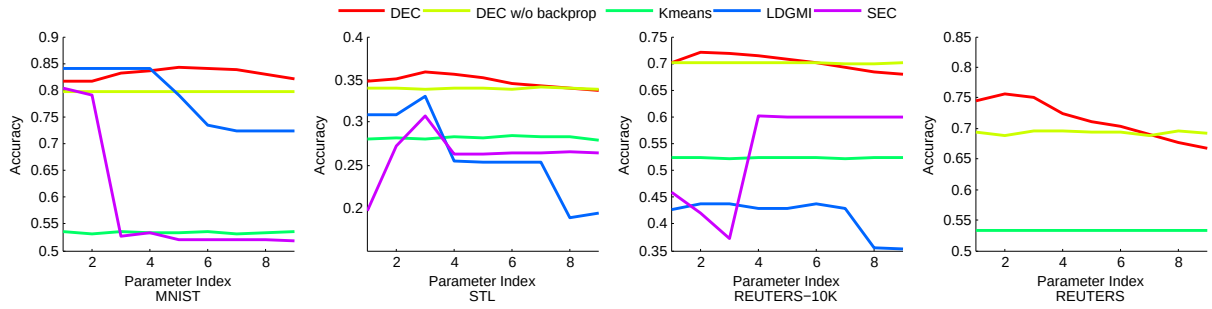


Figure 3.3: Clustering accuracy for different hyperparameter choices for each algorithm. DEC outperforms other methods and is more robust to hyperparameter changes compared to either LDGMI or SEC. Robustness is important because cross-validation is not possible in real-world applications of cluster analysis. This figure is best viewed in color.

In order to investigate the effect of hyperparameters, we plot the accuracy of each method under all 9 settings (Fig. 3.3). We observe that DEC is more consistent across hyperparameter ranges compared to LDGMI and SEC. For DEC, hyperparameter $\lambda = 40$ gives near optimal performance on all dataset, whereas for other algorithms the optimal hyperparameter varies widely. Moreover, DEC can process the entire REUTERS dataset in half an hour with GPU acceleration while the second best algorithms, LDGMI and SEC, would need months of computation time and terabytes of memory. We, indeed, could not run these methods on the full REUTERS dataset and report N/A in Table 3.2 (GPU adaptation of these methods is non-trivial).

In Fig. 3.2 we show 10 top scoring images from each cluster in MNIST and STL. Each row corresponds to a cluster and images are sorted from left to right based on their distance to the cluster center. We observe that for MNIST, DEC's cluster assignment corresponds to natural clusters very well, with the exception of confusing 4 and 9, while for STL, DEC is mostly correct with airplanes, trucks and cars, but spends part of its attention on poses instead of categories when it comes to animal classes.

3.5 Discussion

3.5.1 Assumptions and Objective

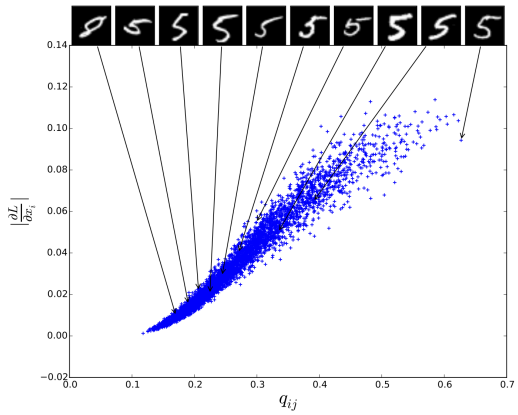
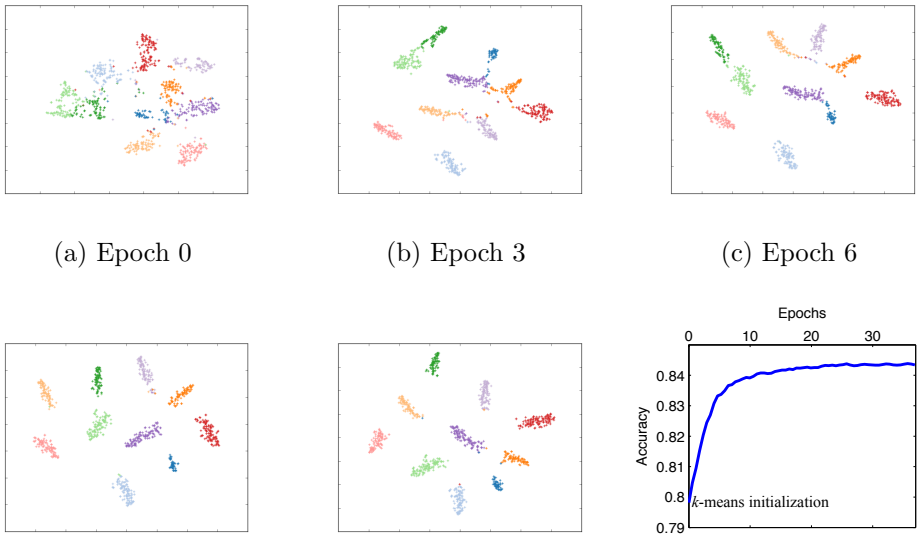


Figure 3.4: Gradient visualization at the start of KL divergence minimization. This plot shows the magnitude of the gradient of the loss L vs. the cluster soft assignment probability q_{ij}. See text for discussion.

The underlying assumption of DEC is that the initial classifier's high confidence predictions are mostly correct. To verify that this assumption holds for our task and that our choice of P has the desired properties, we plot the magnitude of the gradient of L with respect to each embedded point, $|\partial L/\partial z_i|$, against its soft assignment, q_{ij}, to a randomly chosen MNIST cluster j (Fig. 3.4).

We observe points that are closer to the cluster center (large q_{ij}) contribute more to the gradient. We also show the raw images of 10 data points at each 10 percentile sorted by q_{ij}. Instances with higher similarity are more canonical examples of "5". As confidence decreases, instances become more ambiguous and eventually turn into a mislabeled "8" suggesting the soundness of our assumptions.



(a) Epoch 0 (b) Epoch 3 (c) Epoch 6

(d) Epoch 9 (e) Epoch 12 (f) Accuracy vs. epochs

Figure 3.5: We visualize the latent representation as the KL divergence minimization phase proceeds on MNIST. Note the separation of clusters from epoch 0 to epoch 12. We also plot the accuracy of DEC at different epochs, showing that KL divergence minimization improves clustering accuracy. This figure is best viewed in color.

3.5.2 Contribution of Iterative Optimization

In Fig. 3.5 we visualize the progression of the embedded representation of a random subset of MNIST during training. For visualization we use t-SNE [114] applied to the embedded points z_i. It is clear that the clusters are becoming increasingly well separated. Fig. 3.5 (f) shows how accuracy correspondingly improves over SGD epochs.

3.5.3 Contribution of Autoencoder Initialization

To better understand the contribution of each component, we show the performance of all algorithms with autoencoder features in Table 3.3. We observe that SEC and LDMGI's performance do not change significantly with autoencoder feature, while k-means improved

Table 3.3: Comparison of clustering accuracy (Eq. 3.10) on autoencoder (AE) feature.

Method	MNIST	STL-HOG	REUTERS-10k	REUTERS
AE+k-means	81.84%	33.92%	66.59%	71.97%
AE+LDMGI	83.98%	32.04%	42.92%	N/A
AE+SEC	81.56%	32.29%	61.86%	N/A
DEC (ours)	**84.30%**	**35.90%**	**72.17%**	**75.63%**

but is still below DEC. This demonstrates the power of deep embedding and the benefit of fine-tuning with the proposed KL divergence objective.

3.5.4 Performance on Imbalanced Data

Table 3.4: Clustering accuracy (Eq. 3.10) on imbalanced subsample of MNIST.

Method \ r_{min}	0.1	0.3	0.5	0.7	0.9
k-means	47.14%	49.93%	53.65%	54.16%	54.39%
AE+k-means	66.82%	74.91%	77.93%	80.04%	81.31%
DEC	70.10%	80.92%	82.68%	84.69%	85.41%

In order to study the effect of imbalanced data, we sample subsets of MNIST with various retention rates. For minimum retention rate r_{min}, data points of class 0 will be kept with probability r_{min} and class 9 with probability 1, with the other classes linearly in between. As a result the largest cluster will be $1/r_{min}$ times as large as the smallest one. From Table 3.4 we can see that DEC is fairly robust against cluster size variation. We also observe that KL divergence minimization (DEC) consistently improves clustering accuracy after autoencoder and k-means initialization (shown as AE+k-means).

3.5.5 Number of Clusters

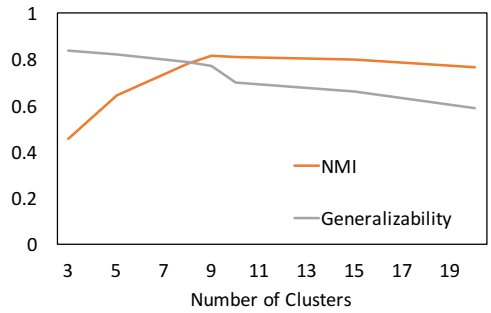


Figure 3.6: Selection of the centroid count, k. This is a plot of Normalized Mutual Information (NMI) and Generalizability vs. number of clusters. Note that there is a sharp drop of generalizability from 9 to 10 which means that 9 is the optimal number of clusters. Indeed, we observe that 9 gives the highest NMI.

So far we have assumed that the number of natural clusters is given to simplify comparison between algorithms. However, in practice this quantity is often unknown. Therefore a method for determining the optimal number of clusters is needed. To this end, we define two metrics: (1) the standard metric, Normalized Mutual Information (NMI), for evaluating clustering results with different cluster number:

$$NMI(l,c) = \frac{I(l,c)}{\frac{1}{2}[H(l) + H(c)]},$$

where I is the mutual information metric and H is entropy, and (2) generalizability (G) which is defined as the ratio between training and validation loss:

$$G = \frac{L_{train}}{L_{validation}}.$$

G is small when training loss is lower than validation loss, which indicate a high degree of overfitting.

Fig. 3.6 shows a sharp drop in generalizability when cluster number increases from 9 to 10, which suggests that 9 is the optimal number of clusters. We indeed observe the highest

NMI score at 9, which demonstrates that generalizability is a good metric for selecting cluster number. NMI is highest at 9 instead 10 because 9 and 4 are similar in writing and DEC thinks that they should form a single cluster. This corresponds well with our qualitative results in Fig. 3.2.

3.6 Conclusion

This paper presents Deep Embedded Clustering, or DEC—an algorithm that clusters a set of data points in a jointly optimized feature space. DEC works by iteratively optimizing a KL divergence based clustering objective with a self-training target distribution. Our method can be viewed as an unsupervised extension of semisupervised self-training. Our framework provide a way to learn a representation specialized for clustering without groundtruth cluster membership labels.

Empirical studies demonstrate the strength of our proposed algorithm. DEC offers improved performance as well as robustness with respect to hyperparameter settings, which is particularly important in unsupervised tasks since cross-validation is not possible. DEC also has the virtue of linear complexity in the number of data points which allows it to scale to large datasets.

Chapter 4

AUTOMATIC 2D-TO-3D CONVERSION

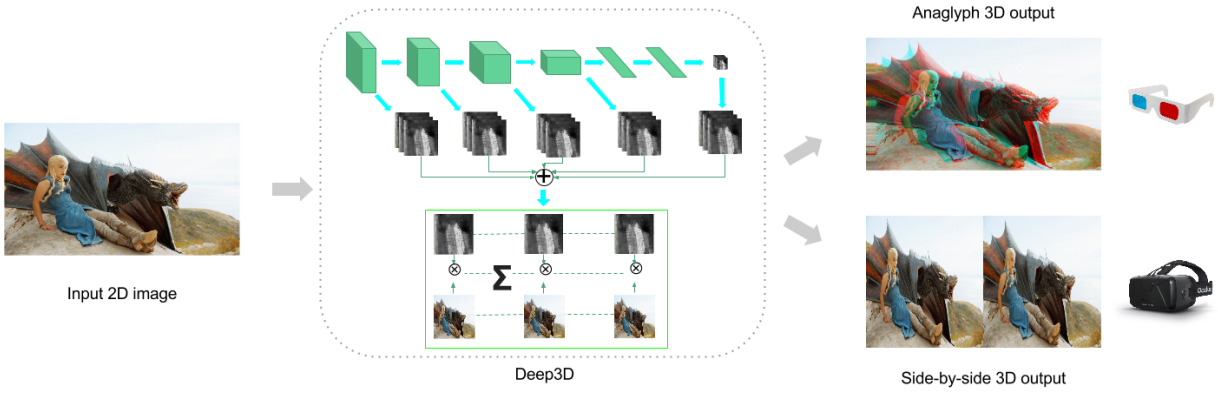


Figure 4.1: Overall architecture of Deep3D.

In this section, we describe Deep3D [120], an 2D-to-3D video conversion algorithm. Deep3D is trained to minimize the pixel-wise reconstruction error of the right view when given the left view. We take a backbone network pre-trained for classifying 2D images and fine-tune it on stereo image pairs for 3D reconstruction.

4.1 Introduction

3D movies are popular and comprise a large segment of the movie theater market, ranging between 14% and 21% of all box office sales between 2010 and 2014 in the U.S. and Canada [83]. Moreover, the emerging market of Virtual Reality (VR) head-mounted displays will likely drive an increased demand for 3D content.

3D videos and images are usually stored in stereoscopic format. For each frame, the format includes two projections of the same scene, one of which is exposed to the viewer's left eye and the other to the viewer's right eye, thus giving the viewer the experience of seeing the scene in three dimensions.

There are two approaches to making 3D movies: shooting natively in 3D or converting to 3D after shooting in 2D. Shooting in 3D requires costly special-purpose stereo camera rigs. Aside from equipment costs, there are cinemagraphic issues that may preclude the use of stereo camera rigs. For example, some inexpensive optical special effects, such as forced perspective[1], are not compatible with multi-view capture devices. 2D-to-3D conversion offers an alternative to filming in 3D. Professional conversion processes typically rely on "depth artists" who manually create a depth map for each 3D frame. Standard Depth Image-Based Rendering (DIBR) algorithms can then be used to combine the original frame with the depth map in order to arrive at a stereo image pair [27]. However, this process is still expensive as it requires intensive human effort.

Each year about 20 new 3D movies are produced. High production cost is the main hurdle in the way of scaling up the 3D movie industry. Automated 2D-to-3D conversion would eliminate this obstacle.

In this chapter, we propose a fully automated, data-driven approach to the problem of 2D-to-3D video conversion. Solving this problem entails reasoning about depth from a single image and synthesizing a novel view for the other eye. Inferring depth (or disparity) from a single image, however, is a highly under-constrained problem. In addition to depth ambiguities, some pixels in the novel view correspond to geometry that's not visible in the available view, which causes missing data that must be hallucinated with an in-painting algorithm.

In spite of these difficulties, our intuition is that given the vast number of stereo-frame pairs that exist in already-produced 3D movies it should be possible to train a machine

[1]Forced perspective is an optical illusion technique that makes objects appear larger or smaller than they really are. It breaks down when viewed from another angle, which prevents stereo filming.

learning model to predict the novel view from the given view. To that end, we design a deep neural network that takes as input the left eye's view, internally estimates a soft (probabilistic) disparity map, and then renders a novel image for the right eye. We train our model end-to-end on ground-truth stereo-frame pairs with the objective of directly predicting one view from the other. The internal disparity-like map produced by the network is computed only in service of creating a good right eye view. We show that this approach is easier to train for than the alternative of using heuristics to derive a disparity map, training the model to predict disparity directly, and then using the predicted disparity to render the new image. Our model also performs in-painting implicitly without the need for post-processing.

Evaluating the quality of the 3D scene generated from the left view is non-trivial. For quantitative evaluations, we use a dataset of 3D movies and report pixel-wise metrics comparing the reconstructed right view and the ground-truth right view. We also conduct human subject experiments to show the effectiveness of our solution. We compare our method with the ground-truth and baselines that use state-of-the-art single view depth estimation techniques. Our quantitative and qualitative analyses demonstrate the benefits of our solution.

4.2 Related Work

Most existing automatic 2D-to-3D conversion pipelines can be roughly divided into two stages. First, a depth map is estimated from an image of the input view, then a DIBR algorithm combines the depth map with the input view to generate the missing view of a stereo pair. Early attempts to estimate depth from a single image utilize various hand engineered features and cues including defocus, scattering, and texture gradients [140, 17]. These methods only rely on one cue. As a result, they perform best in restricted situations where the particular cue is present. In contrast, humans perceive depth by seamlessly combining information from multiple sources.

More recent research has moved to learning-based methods [134, 58, 5, 101, 7]. These approaches take single-view 2D images and their depth maps as supervision and try to learn a mapping from 2D image to depth map. Learning-based methods combine multiple cues and

have better generalization, such as recent works that use deep convolutional neural networks (DCNNs) to advance the state-of-the-art for this problem [24, 71]. However, collecting high quality image-depth pairs is difficult, expensive, and subject to sensor-dependent constraints. As a result, existing depth data set mainly consists of a small number of static indoor and, less commonly, outdoor scenes [85, 30]. The lack of volume and variations in these datasets limits the generality of learning-based methods. Moreover, the depth maps produced by these methods are only an intermediate representation and a separate DIBR step is still needed to generate the final result.

Monocular depth prediction is challenging and we conjecture that performing that task accurately is unnecessary. Motivated by the recent trend towards training end-to-end differentiable systems [137, 66], we propose a method that requires stereo pairs for training and learns to directly predict the right view from the left view. In our approach, DIBR is implemented using an internal probabilistic disparity representation, and while it learns something akin to a disparity map the system is allowed to use that internal representation as it likes in service of predicting the novel view. This flexibility allows the algorithm to naturally handle in-painting. Unlike 2D image / depth map pairs, there is a vast amount of training data available to our approach since roughly 10 to 20 3D movies have been produced each year since 2008 and each has hundreds of thousands frames.

Our model is inspired by Flynn et al.'s DeepStereo approach [29], in which they propose to use a probabilistic selection layer to model the rendering process in a differentiable way so that it can be trained together with a DCNN. Specifically we use the same probabilistic selection layer, but improve upon their approach in two significant ways. First, their formulation requires two or more calibrated views in order to synthesize a novel view—a restriction that makes it impossible to train from existing 3D movies. We remove this limitation by restructuring the network input and layout. Second, their method works on small patches (28×28 pixels) which limits the network's receptive field to local structures. Our approach processes the entire image, allowing large receptive fields that are necessary to take advantage of high-level abstractions and regularities, such as the fact that large people tend to

appear close to the camera while small people tend to be far away.

4.3 Method

Previous work on 2D-to-3D conversion usually consists of two steps: estimating an accurate depth map from the left view and rendering the right view with a Depth Image-Based Rendering (DIBR) algorithm. Instead, we propose to directly regress on the right view with a pixel-wise loss. Naively following this approach, however, leads to poor results because it does not capture the structure of the task (see Section 4.5.4). Inspired by previous work, we utilize a DIBR process to capture the fact that most output pixels are shifted copies of input pixels. However, unlike previous work we don't constrain the system to produce an accurate depth map, nor do we require depth maps as supervision for training. Instead, we propose a model that predicts a probabilistic disparity-like map as an intermediate output and combines it with the input view using a differentiable selection layer that models the DIBR process. During training, the disparity-like map produced by the model is never directly compared to a true disparity map and it ends up serving the dual purposes of representing horizontal disparity and performing in-painting. Our model can be trained end-to-end thanks to the differentiable selection layer.

4.3.1 Model Architecture

Recent research has shown that incorporating lower level features benefits pixel wise prediction tasks including semantic segmentation, depth estimation, and optical flow estimation [24, 28]. Given the similarity between our task and depth estimation, it is natrual to incorporate this idea. Our network, as shown in Fig. 4.2, has a branch after each pooling layer that upsamples the incoming feature maps using so-called "deconvolution" layers (i.e., a learned upsampling filter). The upsampled feature maps from each level are summed together to give a feature representation that has the same size as the input image. We perform one more convolution on the summed feature representation and apply a softmax transform across channels at each spatial location. The output of this softmax layer is interpreted as a prob-

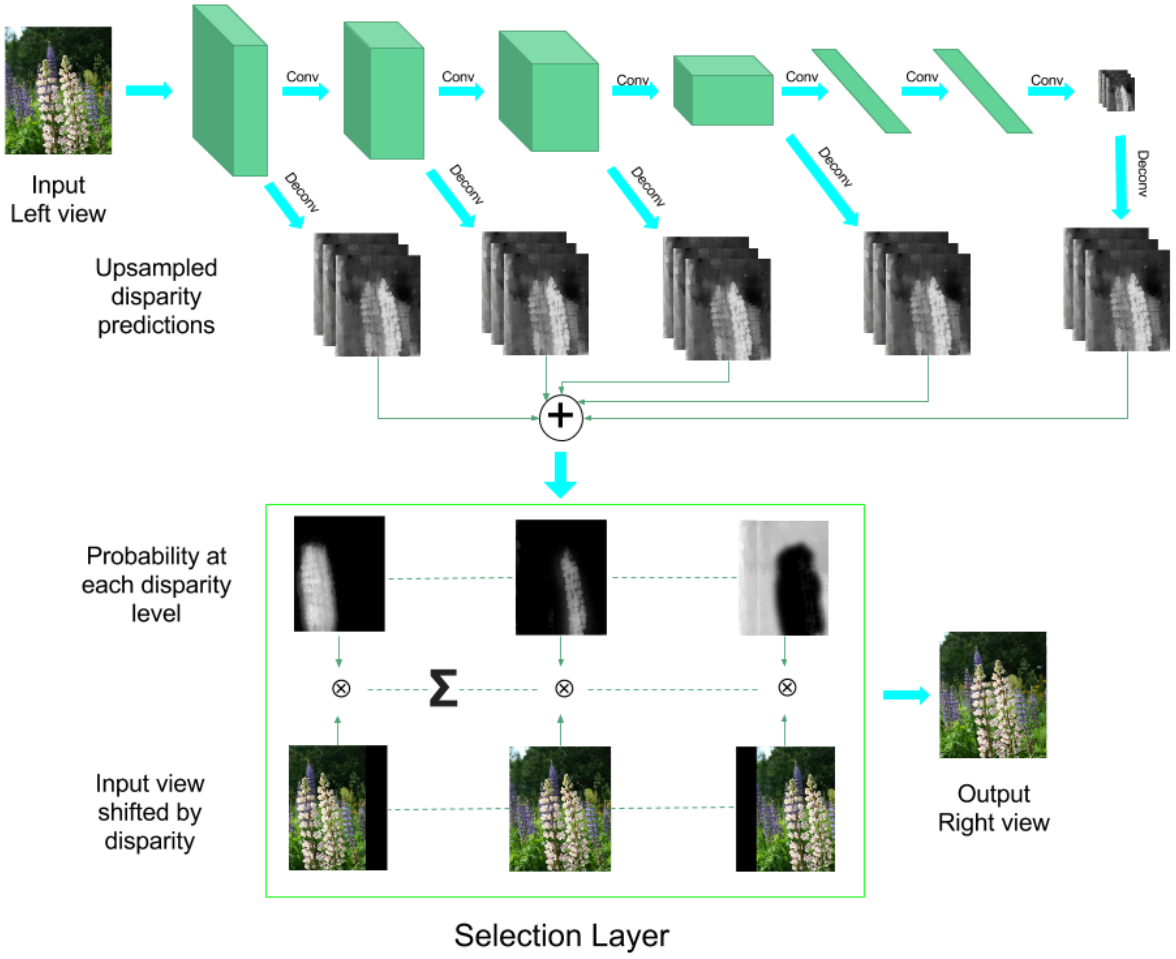


Figure 4.2: Deep3D model architecture. Our model combines information from multiple levels and is trained end-to-end to directly generate the right view from the left view. The base network predicts a probabilistic disparity assignment which is then used by the selection layer to model Depth Image-Based Rendering (DIBR) in a differentiable way. This also allows implicit in-painting.

abilistic disparity map. We then feed this disparity map and the left view to the selection layer, which outputs the right view.

Bilinear interpolation by deconvolution. Similar to [28] we use "deconvolutional" layers to upsample lower layer features maps before feeding them to the final representation. Deconvolutional layers are implemented by reversing the forward and backward computations of a convolution layer. Although technically it is still performing convolution, we call it deconvolutional layer following the convention.

We found that initializing the deconvolutional layers to be equivalent to bilinear interpolation can facilitate training. Specifically, for upsampling by factor S, we use a deconvolutional layer with $2S$ by $2S$ kernel, S by S stride, and $S/2$ by $S/2$ padding. The kernel weight w is then initialized with:

$$C = \frac{2S - 1 - (S \bmod 2)}{2S} \tag{4.1}$$

$$w_{ij} = (1 - |\frac{i}{S - C}|)(1 - |\frac{j}{S - C}|) \tag{4.2}$$

4.3.2 Reconstruction with Selection Layer

The selection layer models the DIBR step in traditional 2D-to-3D conversion. In traditional 2D-to-3D conversion, given the left view I and a depth map Z, a disparity map D is first computed with

$$D = \frac{B(Z - f)}{Z} \tag{4.3}$$

where the baseline B is the distance between the two cameras, Z is the input depth and f is the distance from cameras to the plane of focus. See Fig. 4.3 for illustration. The right view O is then generated with:

$$O_{i,j+D_{ij}} = I_{i,j}. \tag{4.4}$$

However this is not differentiable with respect to D so we cannot train it together with a deep neural network. Instead, our network predicts a probability distribution across possible

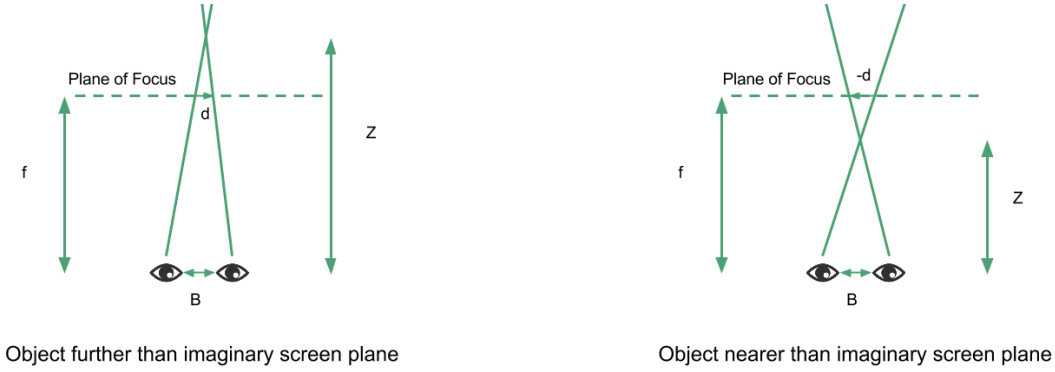


Figure 4.3: Depth to disparity conversion. Given the distance between the eyes B and the distance between the eyes and the plane of focus f, we can compute disparity from depth with Eqn. 4.3. Disparity is negative if object is closer than the plane of focus and positive if it is further away.

disparity values d at each pixel location $D^d_{i,j}$, where $\sum_d D^d_{i,j} = 1$ for all i, j. We define a shifted stack of the left view as $I^d_{i,j} = I_{i,j-d}$, then the selection layer reconstructs the right view with:

$$O_{i,j} = \sum_d I^d_{i,j} D^d_{i,j} \tag{4.5}$$

This is now differentiable with respect to $D^d_{i,j}$ so we can compute an L1 loss between the output and ground-truth right view Y as the training objective:

$$L = |O - Y| \tag{4.6}$$

We use L1 loss because recent research has shown that it outperforms L2 loss for pixel-wise prediction tasks [79].

4.3.3 Scaling Up to Full Resolution

Modern movies are usually distributed in at least 1080p resolution, which has 1920 pixel by 1080 pixel frames. In our experiments, We reduce input frames to 432 by 180 to preserve

aspect ratio and save computation time. As a result, the generated right view frames will only have a resolution of 432 by 180, which is unacceptably low for movie viewing.

To address this issue, we first observe that the disparity map usually has much less high-frequency content than the original color image. Therefore we can scale up the predicted disparity map and couple it with the original high resolution left view to render a full resolution right view. The right view rendered this way has better image quality compared to the naively 4x-upsampled prediction.

4.4 Dataset

Since Deep3D can be trained directly on stereo pairs without ground-truth depth maps as supervision, we can take advantage of the large volume of existing stereo videos instead of using traditional scene depth datasets like KITTI [30] and NYU Depth [85]. We collected 27 non-animation 3D movies produced in recent years and randomly partitioned them to 18 for training and 9 for testing. Our dataset contains around 5 million frames while KITTI and NYU Depth only provide several hundred frames. During training, each input left frame is resized to 432 by 180 pixels and a crop of size 384 by 160 pixels is randomly selected from the frame. The target right frame undergoes the same transformations. We do not use horizontal flipping.

4.5 Experiments

In our main experiments we use a single frame at a time as input without exploiting temporal information. This choice ensures fair comparison to single-frame baseline algorithms and also allows applying trained models to static photos in addition to videos. However, it is natural to hypothesize that motion provides important cues for depth, thus we also conducted additional experiments that use consecutive RGB frames and computed optical flow as input, following [117]. These results are discussed in Section 4.5.4.

4.5.1 Implementation Details

For quantitative evaluation we use the non-upsampled output size of 384 by 160 pixels. For qualitative and human subject evaluation we upsample the output by a factor of 4 using the method described in Section 4.3.3. Our network is based on VGG16, which is a large convolutional network trained on ImageNet [103]. We initialize the main branch convolutional layers (colored green in Fig.4.2) with VGG16 weight and initialize all other weights with normal distribution with a standard deviation of 0.01.

To integrate information from lower level features, we create a side branch after each pooling layer by applying batch normalization [54] followed by a 3×3 convolution layer. This is then followed by a deconvolution layer initialized to be equivalent to bilinear upsampling. The output dimensions of the deconvolution layers match the final prediction dimensions. We use batch normalization to connect pretrained VGG16 layers to randomly initialized layers because it solves the numerical instability problem caused by VGG16's large and non-uniform activation magnitude.

We also connect the top VGG16 convolution layer feature to two randomly initialized fully connected layers (colored blue in Fig.4.2) with 4096 hidden units followed by a linear layer. We then reshape the output of the linear layer to 33 channels of 12 by 5 feature maps which is then fed to a deconvolution layer. We then sum across all up sampled feature maps and do a convolution to get the final feature representation. The representation is then fed to the selection layer. The selection layer interprets this representation as the probability over empty or disparity -15 to 16 (a total of 33 channels).

In all experiments Deep3D is trained with a mini-batch size of 64 for $100,000$ iterations in total. The initial learning rate is set to 0.002 and reduce it by a factor of 10 after every $20,000$ iterations. No weight decay is used and dropout with rate 0.5 is only applied after the fully connected layers. Training takes two days on one NVidia GTX Titan X GPU. Once trained, Deep3D can reconstruct novel right views at more than 100 frames per second. Our implementation is based on MXNet [14] and available for download at

`https://github.com/piiswrong/deep3d`.

4.5.2 Comparison Algorithms

We used three baseline algorithms for comparison:

1. Global Disparity: the right view is computed by shifting the left view with a global disparity δ that is determined by minimizing Mean Absolution Error (MAE) on the validation set.

2. The DNN-based single image depth estimation algorithm of Eigen et al. [24] plus a standard DIBR method as described in Section 4.3.2.

3. Ground-truth stereo pairs. We only show the ground-truth in human subject studies since in quantitative evaluations it always gives zero error.

To the best of our knowledge, Deep3D is the first algorithm that can be trained directly on stereo pairs, while all previous methods requires ground-truth depth map for training. For this reason, we cannot retrain comparison algorithms on our 3D movie data set. Instead, we take the model released by Eigen et al. [24] and evaluate it on our test set. While it is a stretch to hope that a model trained on NYU Depth will generalize well to 3D movies, this fact underscores a principal strength of our approach: by directly training on stereo pairs, we can exploit vastly more training data.

Because [24] predicts depth rather then disparity, we need to convert depth to disparity with Eqn. 4.3 for rendering with DIBR. However, [24] does not predict the distance to the plane of focus f, a quantity that is unknown and varies across shots due to zooming. The interpupillary distance B is also unknown, but it is fixed across shots. The value of B and f can be determined in two ways:

1. Optimize for MAE on the validation set and use fixed values for B and f across the whole test set. This approach corresponds to the lower bound of [24]'s performance.

2. Fix B across the test set, but pick the f that gives the lowest MAE for each *test* frame. This corresponds to having access to oracle plane of focus distance and thus the upper bound on [24]'s performance.

We do both and report them as two separate baselines, [24] and [24] + Oracle. For fair comparisons, we also do this optimization for Deep3D's predictions and report the performance of Deep3D and Deep3D + Oracle.

4.5.3 Results

Table 4.1: Deep3D evaluation. We compare pixel-wise reconstruction error for each method using Mean Absolute Error (MAE) as metric.

Method	MAE
Global Disparity	7.75
[24]	7.75
Deep3D (ours)	**6.87**
[24] + Oracle	6.31
Deep3D + Oracle	**5.47**

Quantitative evaluation. For quantitative evaluation, we compute Mean Absolute Error (MAE) as:

$$MAE = \frac{1}{HW}|y - g(x)|, \tag{4.7}$$

$$\tag{4.8}$$

where x is the left view, y is the right view, $g(\cdot)$ is the model, and H and W are height and width of the image respectively. The results are shown in Table 4.1. We observe that Deep3D outperforms baselines with and without oracle distance of focus plane.

Figure 4.4: Qualitative results. Column one shows an anaglyph of the predicted 3D image (best viewed in color with red-blue 3D glasses). Each anaglyph is followed by 12 heat maps of disparity channels -3 to 8 (closer to far). In the first row, the man is closer and appears in the first 3 channels while the woman is further away and appears in 4th-5th channels; the background appears in the last 4 channels.

Figure 4.5: Comparison between [24] and Deep3D. The first column shows the input image. The second column shows the prediction of [24] and the third column shows Deep3D's prediction. This figure shows that Deep3D is better at delineating people and figuring out their distance from the camera.

Qualitative evaluation. To better understand the proposed method, we show qualitative results in Fig. 4.4. Each entry starts with a stereo pair predicted by Deep3D shown in anaglyph, followed by 12 channels of internal soft disparity assignment, ordered from near (-3) to far (+8). We observe that Deep3D is able to infer depth from multiple cues including size, occlusion, and geometric structure.

We also compare Deep3D's internal disparity maps (column 3) to [24]'s depth predictions (column 2) in 4.5. This figure demonstrates that Deep3D is better at delineating people and figuring out their distance from the camera.

Note that the disparity maps generated by Deep3D tend to be noisy at image regions with low horizontal gradient, however this does not affect the quality of the final reconstruction because if a row of pixels have the same value, any disparity assignment would give the same reconstruction. Disparity prediction only needs to be accurate around vertical edges and we

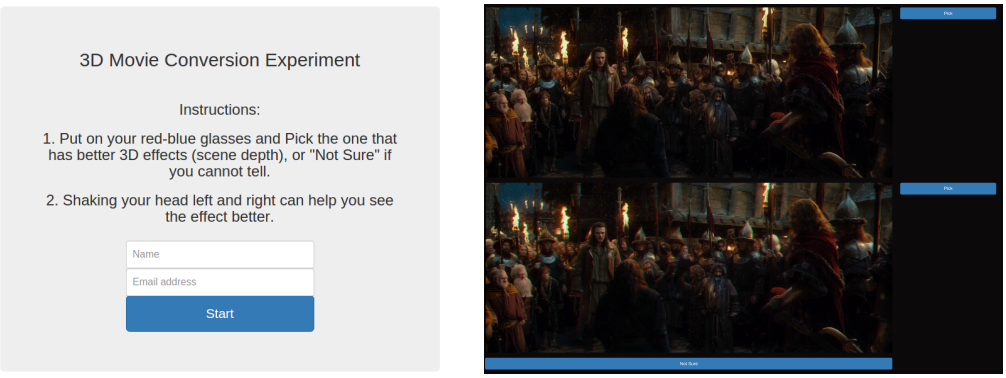


Figure 4.6: Human subject study setup. Each subject is shown 50 pairs of 3D anaglyph images. Each pair consists of the same scene generated by 2 randomly selected methods. The subjects are instructed to wear red-blue 3D glasses and pick the one with better 3D effects or "Not Sure" if they cannot tell. The study result is shown in Table 4.2

indeed observe that Deep3D tends to focus on such regions.

Human subject evaluation. We also conducted a human subject study to evaluate the visual quality of the predictions of different algorithms. We used four algorithms for this experiment: Global Disparity, [24] + Oracle, Deep3D without Oracle, and the ground-truth[2].

For the human subject study, we randomly selected 500 frames from the test set. Each annotator is shown a sequence of trials. In each trial, the annotator sees two anaglyph 3D images, which are reconstructed from the same 2D frame by two algorithms, and is instructed to wear red-blue 3D glasses and pick the one with better 3D effects or select "not sure" if they are similar. The interface for this study is shown in Fig. 4.6. Each annotator is given 50 such pairs and we collected decisions on all $C_4^2 500$ pairs from 60 annotators.

Table 4.2 shows that Deep3D outperforms the naive Global Disparity baseline by a 49% margin and outperforms [24] + Oracle by a 32% margin. When facing against the ground

[2] [24] without Oracle and Deep3D + Oracle are left out due to annotator budget. Note that a change in average scene depth only pushes a scene further away or pull it closer and usually doesn't affect the perception of depth variation in the scene.

Table 4.2: Human Subject Evaluation. Each entry represents the frequency of the row method being preferred to the column method by human subjects. Note that 66% of times subjects prefer Deep3D to [24] and 24% of the times Deep3D is preferred over the ground truth.

	Global Disparity	[24] + Oracle	Deep3D (ours)	Ground Truth
Global Disparity	N/A	26.94%	25.42%	7.88%
[24] + Oracle	73.06%	N/A	33.92%	10.27%
Deep3D (ours)	74.58%	66.08%	N/A	24.48%
Ground Truth	92.12%	89.73%	75.52%	N/A

truth, Deep3D's prediction is preferred 24.48% of the time while [24] + Oracle is only preferred 10.27% of the time and Global Disparity baseline is preferred 7.88% of the time.

4.5.4 Algorithm Analysis

Ablation study. To understand the contribution of each component of the proposed algorithm, we show the performance of Deep3D with parts removed in Tab. 4.3. In Deep3D w/o lower level feature we show the performance of Deep3D without branching off from lower convolution layers. The resulting network only has one feed-forward path that consists of 5 convolution and pooling module and 2 fully connected layers. We observe that the performance significantly decreases compared to the full method.

In Deep3D w/o direct training on stereo pairs we show the performance of training on disparity maps generated from stereo pairs by block matching algorithm [49] instead of directly training on stereo pairs. The predicted disparity maps are then fed to DIBR method to render the right view. This approach results in decreased performance and demonstrates the effectiveness of Deep3D's end-to-end training scheme.

Table 4.3: Ablation studies. We evaluate different components of Deep3D by removing them from the model to further understand the contribution of each component. Note that removing lower level features and selection layer both result in performance drop.

Method	MAE
Deep3D w/o lower level feature	8.24
Deep3D w/o direct training on stereo pairs	7.29
Deep3D w/o selection layer	7.01
Deep3D	6.87

We also show the result from directly regressing on the novel view without internal disparity representation and selection layer. Empirically this also leads to decreased performance, demonstrating the effectiveness of modeling the DIBR process.

Temporal information. In our main experiment and evaluation we only used one still frame of RGB image as input. We made this choice for fair comparisons and more general application domains. Incorporating temporal information into Deep3D can be handled in two ways: use multiple consecutive RGB frames as input to the network, or provide temporal information through optical flow frames similar to [117].

We briefly explored both directions and found moderate performance improvements in terms of pixel-wise metrics. We believe more effort along this direction, such as model structure adjustment, hyper-parameter tuning, and explicit modeling of time will lead to larger performance gains at the cost of restricting application domain to videos only.

4.6 Conclusions

In this paper we proposed a fully automatic 2D-to-3D conversion algorithm based on deep convolutional neural networks. Our method is trained end-to-end on stereo image pairs

Table 4.4: Temporal information. We incorporate temporal information by extending the input to include multiple consecutive RGB frames or optical flow frames. We observe that additional temporal information leads to performance gains.

Method	MAE
Deep3D with 5 RGB frames	6.81
Deep3D with 1 RGB frames and 5 optical flow frames	6.86
Deep3D	6.87

directly, thus able to exploit orders of magnitude more data than traditional learning based 2D-to-3D conversion methods. Quantitatively, our method outperforms baseline algorithms. In human subject study stereo images generated by our method are consistently preferred by subjects over results from baseline algorithms. When facing against the ground truth, our results have a higher chance of confusing subjects than baseline results.

In our experiment and evaluations we only used still images as input while ignoring temporal information from video. The benefit of this design is that the trained model can be applied to not only videos but also photos. However, in the context of video conversion, it is likely that taking advantage of temporal information can improve performance. We briefly experimented with this idea but found little quantitative performance gain. We conjecture this may be due to the complexity of effectively incorporating temporal information. We believe this is an interesting direction for future research.

Chapter 5

OBJECT DETECTION

Object detection aims at classifying and localizing all objects in an image. It is a heavily studied task in computer vision. The cost of labeling images for object detection training is relatively high because of the need for labeling multiple objects and their locations per image. As a result, most object detection algorithms leverage data from other domains by using pre-trained models.

In this chapter, we explore a number of tricks that help boosting the performance of object detection models. Our experiments show that these tricks can bring as much as 5% absolute precision increment without sacrificing inference speed [135]. In particular, we highlight one technique, mixup, which improves accuracy as while as alleviating confusion caused by context. We also investigate the interaction of mixup and transfer learning and found that mixup is most effective when it is applied to both the pre-training and fine-tuning stages.

5.1 Introduction

Object detection is one of the cutting edge research areas in computer vision. Most state-of-the-art detectors, including single stage (SSD [73] and YOLO [95]) and multi-stage (Faster-RCNN, etc. [96]) models, are based on pre-trained backbone networks like VGG [103], ResNet [42], Inception [107] and MobileNet [51, 100].

However, due to inherent complexities and fast progress in the field, researchers have not converged to a standard setting for initialization, data pre-processing, optimization, etc. As a result, many results are not comparable to each other. It is also difficult to delineate the contributions from improved pre-trained backbone networks and improved object detection

algorithms.

In this chapter, we explore effective techniques that can boost the accuracy of state-of-the-art object detection methods without introducing extra computational cost. We systematically examine many components of the training pipeline, including learning rate scheduling and weight decay, to provide best practices and guidelines. In particular, we highlight one technique, mixup, that constructs auxiliary training samples by blending two images at variable ratio. mixup was first proposed for image classification in [132]. In this chapter, we adapt it to object detection and found that it helps improving accuracy as well as decreasing confusion caused by context.

The rest of this chapter is organized as follows. First, we introduce related works in Section 5.2. Then, the proposed tricks are detailed in Section 5.3. Next, we show experiment results in Section 5.4. Finally, Section 5.5 concludes this chapter.

5.2 Related Work

5.2.1 Scattering tricks from Image Classification

Image classification serves as the foundation of all most all computer vision tasks. Classification models are cheap compared with popular object detection and semantic segmentation models, therefore attractive enormous researchers to prototyping ideas. In this section, we briefly introduce previous work that opened the shed for this work. Learning rate warm up heuristic [36] was introduced to overcome the negative effect of extremely large mini-batch size. Interestingly, even though mini-batch size used in typical object detection training is nowhere close to the scale in image classification(*e.g.* 10k or 30k [36]), a large amount of anchor size(up to 30k) is effectively contributing to batch size implicitly. a gradual warmup heuristic is crucial to YOLOv3 [95] as in our experiments. There is a line of approaches trying to address the vulnerability of deep neural network. Label smoothing was introduced in [107], which modifies the hard ground truth labeling in cross entropy loss. Zhang *et al.* [132] proposed *mixup* to alleviate adversarial perturbation. Cosine annealing strategy for

learning rate decay is proposed in [77] in response to traditional step policy. In this work, we dive deeper into the heuristic techniques introduced by image classification in the context of object detection.

5.2.2 Deep Object Detection Pipelines

Most state-of-the-art deep neural network based object detection models are derived from multiple stages and single stage pipelines, starting from R-CNN [33] and YOLO [94], respectively. In single stage pipelines, predictions are generated by a single convolutional network and therefore preserve the spatial alignments (except that YOLO used Fully Connected layers at the end). However, in multiple stage pipelines, e.g. Fast R-CNN [32] and Faster-RCNN [96], final predictions are generated from features which were sampled and pooled in a specific region of interests (RoIs). RoIs were either propagated by neural networks or deterministic algorithms (e.g. Selective Search [111]). This major difference caused significant divergence in data processing and network optimization. For example, due to the lack of spatial variation in single stage pipelines, spatial data augmentation is crucial to the performance as proven in Single-Shot MultiBox Object Detector (SSD) [73]. Due to lack of exploration, many training details are exclusive to one series. In this work, we systematically explore the mutually beneficial tweaks and tricks that may help to boost the performance for both pipelines.

5.3 Technique Details

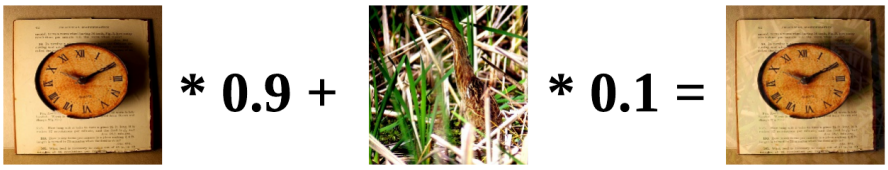


Figure 5.1: mixup visualization of image classification with typical mixup ratio at 0.1 : 0.9.

Figure 5.2: Geometry preserved alignment of mixed images for object detection. Image pixels are blended, object labels are merged with respect to generated new image.

In this section, we propose a visual coherent image mixup method for object detection and introduce data processing and training scheduler designed for systematically improving the model performance of object detectors.

5.3.1 Visually Coherent Image Mixup for Object Detection

Mixup, introduced by Zhang *et al.* [132] is proved to be successful in alleviating adversarial perturbations in classification networks. The distribution of blending ratio in mixup algorithm proposed by Zhang *et al.* [132] is drawn from beta distribution with $(a = 0.2, b = 0.2)$. The majority of mixups are barely noises with such beta distribution. Inspired by the heuristic experiments in Rosenfeld *et al.* [97], we focus on the natural co-occurrence object presentations which play significant roles in object detection. The semi-adversarial object patch transplantation method is not a traditional attack, but introduced natural occlusions, spatial signal perturbations that are common but good challengers to existing object detectors.

In our empirical experiments, continue increasing blending ratio used in the mixup, the objects in resulting frames are more vibrant and coherent to the natural presentations, similar to the transition frames commonly in low FPS movies. The visual comparisons of image

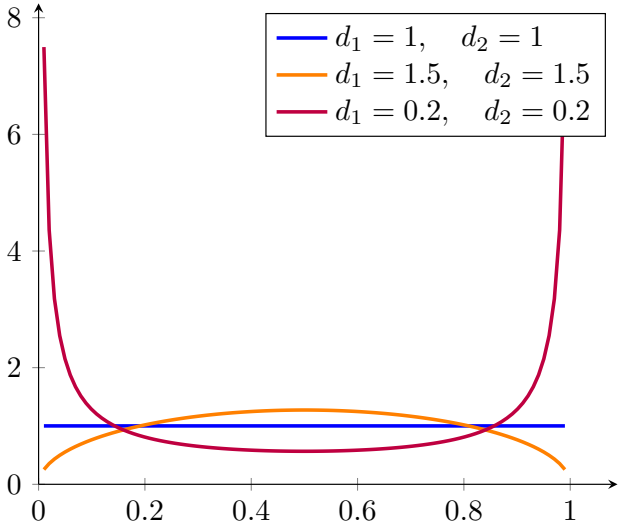


Figure 5.3: Comparison of different random weighted mixup sampling distributions. Red curve indicate the typical mixup ratio used in image classification.

classification and such high ratio mixup are illustrated in Fig. 5.1 and Fig. 5.2, respectively. In particular, we use geometry preserved alignment for image mixup to avoid distort images at the initial steps. We also choose a beta distribution with more visually coherent ratios $a >= 1$ and $b >= 1$ instead of following the same practice in image classification, as depicted in Figure 5.3.

We also experimentally tested empirical mixup ratio distributions using the YOLOv3 network on Pascal VOC dataset. Table. 5.1 shows the actual improvements by adopting detection mixups. Beta distribution with α and β both equal to 1.5 is marginally better than 1.0 (equivalent to uniform distribution) and better than fixed even mixup.

To validate the effectiveness of visually coherent mixup, we followed the same experiments of "Elephant in the room" [97] by sliding an elephant image patch through an indoor room image. We trained two YOLOv3 models on COCO 2017 dataset with identical settings except for that model **mix** is using our mixup approach. We depict some surprising discoveries in Fig. 5.4.

Model	mAP @ 0.5
baseline	81.5
0.5:0.5 evenly	83.05
beta(1.0, 1.0), weighted loss	83.48
beta(1.5, 1.5), weighted loss	83.54

Table 5.1: Effect of various mixup approaches, validated with YOLOv3 [95] on Pascal VOC 2007 test set.

5.3.2 Classification Head Label Smoothing

For each object, detection networks often compute a probability distribution over all classes with softmax function:

$$p_i = \frac{e^{z_i}}{\sum_j e^{z_j}}, \tag{5.1}$$

where z_is are the unnormalized logits directly from the last linear layer for classification prediction. For object detection during training, we only modify the classification loss by comparing the output distribution p against the ground truth distribution q with cross-entropy

$$L = \sum_i q_i \log p_i. \tag{5.2}$$

q is often a one-hot distribution, where the correct class has probability one while all other classes have zero. Softmax function, however, can only approach this distribution when $z_i \gg z_j, \forall j \neq i$ but never reach it. This encourages the model to be too confident in its predictions and is prone to over-fitting.

Label smoothing was proposed by Szegedy *et al.* [107] as a form of regularization. We

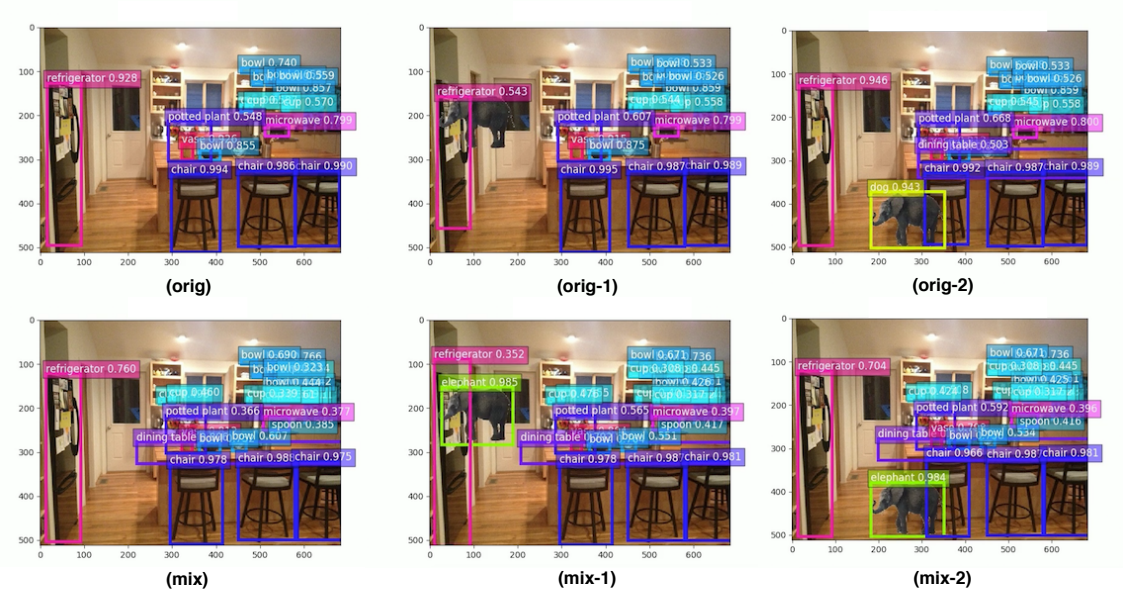


Figure 5.4: Elephant in the room example. Model trained with geometry preserved mixup (bottom) is more robust against alien objects compared to baseline (top).

smooth the ground truth distribution with

$$q_i' = (1 - \epsilon)q_i + \frac{\epsilon}{K}, \tag{5.3}$$

where K is the total number of classes and ϵ is a small constant. This technique reduces the model's confidence, measured by the difference between the largest and smallest logits.

In the case of sigmoid outputs in 0 to 1.0 as in YOLOv3 [95], label smoothing is even simpler by correcting the upper and lower limit of the range of targets as in Eq. 5.3.

5.3.3 Data Pre-processing

Unlike image classification domain, where the network is extremely tolerant to image geometrical transformation. It is actually encouraged to do so in order to improve generalization accuracy. However, for object detection image pre-processing, we need to carry additional cautious since detection networks are more sensitive to such transformations.

We experimentally reviewed the following data augmentation methods:

- Random geometry transformation. Including random cropping (with constraints), random expansion, random horizontal flip and random resize (with random interpolation).

- Random color jittering including brightness, hue, saturation, and contrast.

In terms of types of detection networks, there are two pipelines for generating final predictions. First is single stage detector network, where final outputs are generated from every single pixel on feature map, for example, SSD[73] and YOLO[95] networks which generate detection results proportional to spatial shape of an input image. The second is multi-stage proposal and sampling based approached, following Fast-RCNN[96], a large amount of ROI candidates are generated and sampled with a fixed number, the detection results are produced by repeatedly cropping the corresponding region on feature maps and the number of predictions is proportional to the fixed sampling number.

Since sampling-based approaches repeat enormous crop like operations on feature maps, it substitutes the operation of randomly cropping the input image, therefore these networks do not require extensive geometric augmentations applied during the training stage.

5.3.4 Training Scheduler Revamping

During training, the learning rate usually starts with a relatively big number and gradually becomes smaller throughout the training process. For example, the step schedule is the most widely used learning rate schedule. With step schedule, the learning rate is multiplied by a constant number below 1 after reaching pre-defined epochs or iterations. For instance, the default step learning rate schedule for Faster-RCNN [96] is to reduce learning rate by ratio 0.1 at $60k$ iterations. Similarly, YOLOv3 [95] uses same ratio 0.1 to reduce learning rate at $40k$ and $45k$ iterations. Step scheduler has sharp learning rate transition which may cause the optimizer to re-stabilize the learning momentum in the next few iterations. In contrast, a smoother cosine learning rate adjustment was proposed by Loshchilov *et al*. [77]. Cosine schedule scales the learning rate according to the value of cosine function on 0 to pi. It starts

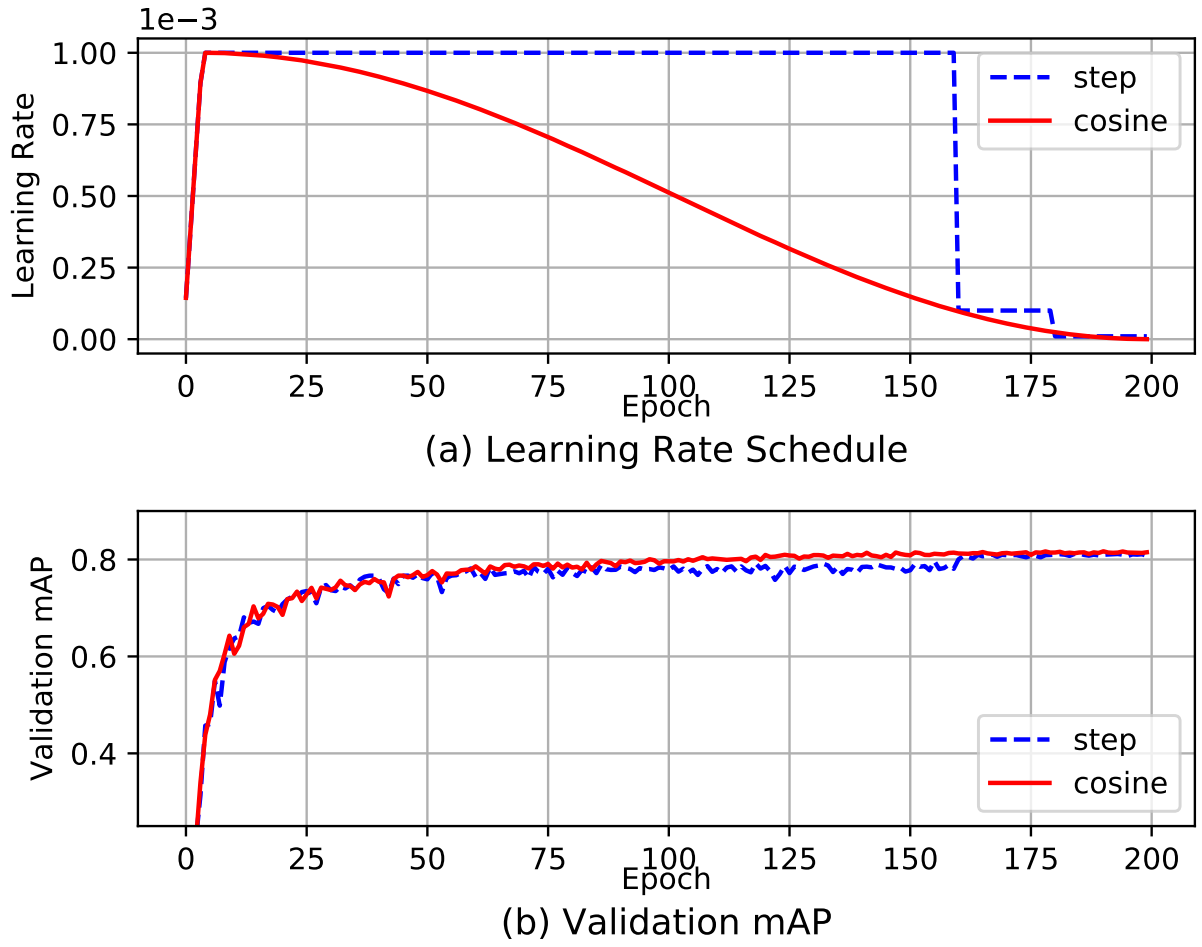


Figure 5.5: Visualization of learning rate scheduling with warm-up enabled for YOLOv3 training on Pascal VOC. (a): cosine and step schedules for batch size 64. (b): Validation mAP comparison curves using step and cosine learning scheduler.

with slowly reducing large learning rate, then reduces the learning rate quickly halfway, and finally ends up with tiny slope reducing small learning rate until it reaches 0.

Warm up learning rate is another common strategy to avoid gradient explosion during the initial training iterations. Warm-up learning rate schedule is critical to several object detection algorithms, e.g., YOLO v3, which has a dominant gradient from negative examples

in the very beginning iterations where sigmoid classification score is initialized around 0.5 and biased towards 0 for the majority predictions.

Training with cosine schedule and proper warmup lead to better validation accuracy, as depicted in Fig. 5.5, validation mAP achieved by applying cosine learning rate decay outperforms step learning rate schedule at all times in training. Due to the higher frequency of learning rate adjustment, it also suffers less from plateau phenomenon of step decay that validation performance will be stuck for a while until learning rate is reduced.

5.3.5 Synchronized Batch Normalization

In recent years, the massive computation requirements forced training environments to equip multiple devices (usually GPUs) to accelerate training. Despite handling different hyper-parameters in response to larger batch sizes during training, Batch Normalization [54] is drawing the attention of multi-device users due to the implementation details. Although the typical implementation of Batch Normalization working on multiple devices (GPUs) is fast (with no communication overhead), it inevitably reduces the size of batch size and causing slightly different statistics during computation, which potentially degraded performance. This is not a significant issue in some standard vision tasks such as ImageNet classification (as the batch size per device is usually large enough to obtain good statistics). However, it will hurt the performance in some tasks that the batch size is usually very small (*e.g.*, 1 per GPU). Recently, Peng *et al*. has proved the importance of synchronized batch normalization in object detection with the analysis made by Pen *et al*. [92]. In this work, we systematically review the importance of Synchronized Batch Normalization with YOLOv3 [95] and Faster-RCNN [96] to evaluate the impacts on larger (usually larger than 64) and smaller (usually less than 4) batch trainers, respectively.

5.3.6 No Decay to Bias Terms

The weight decay is often applied to all learnable parameters including both weights and bias. It equals to applying an L2 regularization to all parameters to drive their values towards 0.

However, it's only recommended to apply the regularization to weights to avoid over-fitting, as pointed by Jia *et al*. [55]. The no bias decay heuristic follows this recommendation, it only applies the weight decay to the weights in convolution and fully-connected layers. Other parameters, including the biases and γ and β in Batch Normalization layers, are left un-regularized.

5.4 Experiments

Figure 5.6: Example detection results by applying BoF on COCO 2017 validation set.

Model	mAP	Delta
- data augmentation	64.26	-15.99
baseline	80.25	0
+ synchronize BN	80.81	+0.56
+ random training shapes	81.23	+0.98
+ cosine lr schedule	81.69	+1.44
+ class label smoothing	82.14	+1.89
+ mixup	**83.68**	**+3.43**

Table 5.2: Training Refinements on YOLOv3, evaluated at 416×416 on Pascal VOC 2007 test set.

In order to compare incremental improvements of all tweaks for object detection, we used YOLOv3 [95] and Faster-RCNN [96] as representatives for single and multiple stages pipelines, respectively. To accommodate massive training tasks, we use Pascal VOC [26] for fine-graded trick evaluation, and COCO [70] dataset for overall performance gain and generalization ability verification.

Results on Pascal VOC.

Following the common settings used in Fast-RCNN [32] and SSD [73], we use Pascal VOC 2007 *trainval* and 2012 *trainval* for training and 2007 test set for validation. The results are reported in mean average precision defined in Pascal VOC development kit [26]. For YOLOv3 models, we consistently measure mean average precision (mAP) at 416×416 resolution. If random shape training is enabled, YOLOv3 models will be fed with random resolutions from 320×320 to 608×608 with 32×32 increments, otherwise they are always trained with fixed 416×416 input data. Faster RCNN models take arbitrary input resolutions. In order to regulate training memory consumption, the shorter sides of input images are resized to 600

Model	mAP	Delta
- data augmentation	77.61	-0.16
baseline	77.77	0
+ cosine lr schedule	79.59	+1.82
+ class label smoothing	80.23	+2.46
+ mixup	**81.32**	**+3.55**

Table 5.3: Training Refinements on Faster-RCNN, evaluated at 600×1000 on Pascal VOC 2007 test set.

Model	Orig $AP^{0.5:0.95}_{bbox}$	**Our BoF** $AP^{0.5:0.95}_{bbox}$	Absolute delta
Faster-RCNN resnet50 [34]	36.5	**37.6**	**+1.1**
Faster-RCNN resnet101 [34]	39.4	**41.1**	**+1.7**
YOLOv3 [95]	33.0	**37.0**	**+4.0**

Table 5.4: Overview of improvements achieved by applying bag of freebies(BoF), evaluated on MS COCO [70] 2017 val set.

pixels while ensuring the longer side in smaller than 1000 pixels. Training and validation of Faster-RCNN models follow the same pre-processing steps, except that training images have chances of 0.5 to flip horizontally as additional data augmentation. The incremental evaluations of YOLOv3 and Faster-RCNN with our bags of freebies (BoF) are detailed in Table. 5.2 and Table. 5.3, respectively. The results are motivating that by stacking the tricks, YOLOv3 models can achieve up to 3.5% performance gain and Faster-RCNN models have less effective tricks but obtained similar performance bump. One phenomena has been discovered that data augmentation has minimal effect on multi-stage pipelines, *i.e.*, Faster-RCNN,

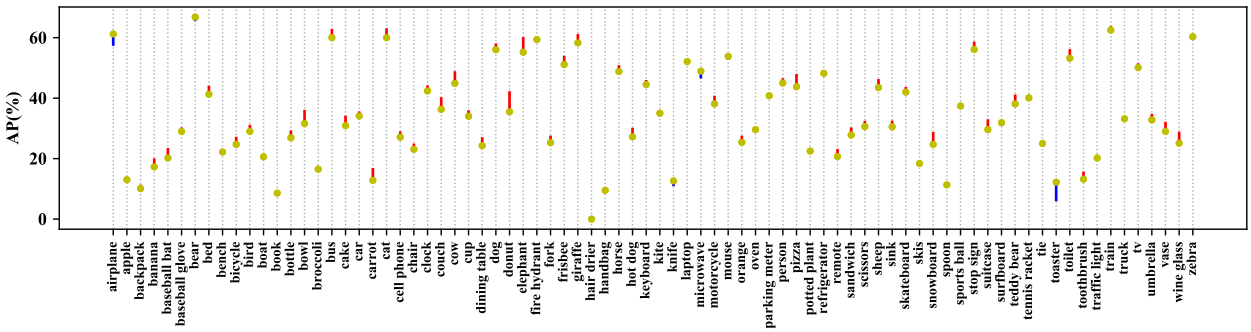


Figure 5.7: **COCO 80 category AP analysis with YOLOv3 [95].** Red lines indicate performance gain using BoF, while blue lines indicate performance drop.

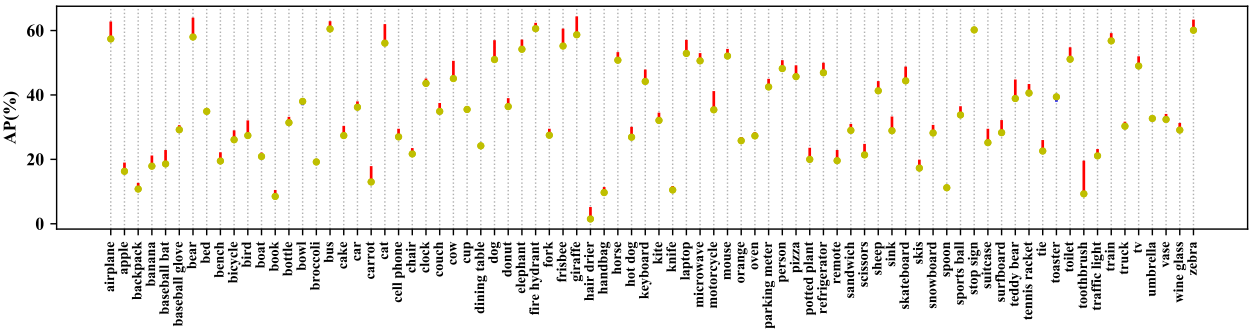


Figure 5.8: **COCO 80 category AP analysis with Faster-RCNN resnet50 [96].** Red lines indicate performance gain using BoF, while blue lines indicate performance drop.

but poses significant compact to YOLOv3 models, due to the lack of spatial mutational operations.

Results on MS COCO

COCO 2017 is 10 times larger than Pascal VOC and contains tiny objects compared with PASCAL VOC. We use MS COCO to validate the generalization of our bags of tricks in this work. We use similar training and validation settings as Pascal VOC, except that Faster-RCNN models are resized to 800 $\times$ 1300 pixels in response to smaller objects. The results

are shown in Table. 5.4. In summary, our proposed bags of freebies boosted Faster-RCNN models by 1.1% and 1.7% absolute mean AP over existing state-of-the-art implementations [34] with ResNet 50 and 101 base models, respectively. Our proposed bag of tricks also outperforms existing YOLOv3 models by as large as 4.0% absolute mAP. Note that all these results are obtained by generating better weights in a fully compatible inference model, *i.e.*, all these achievements are free lunch during inference.

Mixup and Transfer Learning

	-Mixup YOLO3	+Mixup YOLO3
-Mixup darknet53	35.0	35.3
+Mixup darknet53	36.4	37.0

Table 5.5: Combined analysis of impacts of mixup methodology for pre-trained image classification and detection network.

	-Mixup FRCNN	+Mixup FRCNN
-Mixup R101	39.9	40.1
+Mixup R101	40.1	41.1

Table 5.6: Combined analysis of impacts of mixup methodology for pre-trained image classification and detection network.

Mixup can be applied in two phases of object detection networks: 1) pre-training classification network backbone with mixup [44, 132]; 2) training detection networks using proposed visually coherent image mixup for object detection. We compare the results using Darknet

53-layer based YOLO3 [95] implementation and ResNet101 [42] based Faster-RCNN [96] in Table 5.5 and Table 5.6, respectively. The results show that applying mixup to both phases of training yields compounding improvements.

5.5 Conclusion

In this chapter, we proposed a bag of training enhancements that significantly improved model performances while introducing zero overhead during inference. Our empirical experiments with YOLOv3 [95] and Faster-RCNN [96] on Pascal VOC and COCO datasets show that the bag of tricks consistently improves object detection models. By stacking all these tweaks, we observe additive improvements and suggest a wider adoption in future object detection training pipelines.

Chapter 6

CONCLUSIONS

In this thesis, we studied the intersection of transfer learning, deep learning, and computer vision. We applied transfer learning to various computer vision applications including object detection, 2D-to-3D conversion, and clustering. We also studied the most widely used source task in computer vision, i.e. image classification, and empirically examined the "transferability" between improvements on the source task and improvements on the target task.

In Chapter 2, we first experimented with a combination of techniques for improving image classification accuracy on ImageNet, which is the most widely used source task in computer vision. Then, we empirically demonstrated that improvements can be transferred to target tasks including object detection and semantic segmentation.

In the following chapters, we studied the role of transfer learning in a number of computer vision tasks. In Chapter 3, we described deep embedded clustering (DEC), a clustering algorithm that combines unsupervised pre-training and self-supervised fine-tuning. DEC first used unsupervised autoendocers to learn a mapping from the data space to a lower-dimensional feature space and then iteratively refines the mapping with a clustering objective. Our experiments on image and text corpora showed that DEC brings significant improvement over previous state-of-the-art clustering methods.

In Chapter 4, we presented Deep3D, a 2D-to-3D video conversion algorithm. Deep3D was trained to minimize the pixel-wise reconstruction error of the right view when given the left view. We took a backbone network pre-trained for classifying 2D images and fine-tune it on stereo image pairs for 3D reconstruction. Deep3D outperformed baselines in both quantitative and human subject evaluations.

Finally, in Chapter 5, we focused on object detection and studied various factors that affect detection accuracy. In particular, we adapted mixup [132] to object detection and showed that it helps to improve generalization and avoid confusion caused by context. We also demonstrated that in order to attain the best effect of mixup training in detection, the source model also needs to be pre-trained with mixup on image classification.

The main contribution of this thesis is two fold: 1) we showed that transfer learning can be used in many forms to improve performance on various computer vision tasks, and 2) we empirically studied the link between performance on the source task and performance after transfer learning. For future work, it would be interesting to form a theoretical framework for explaining transferability.

BIBLIOGRAPHY

[1] Charu C Aggarwal and Chandan K Reddy. *Data clustering: algorithms and applications*. CRC Press, 2013.

[2] Salem Alelyani, Jiliang Tang, and Huan Liu. Feature selection for clustering: A review. *Data Clustering: Algorithms and Applications*, 2013.

[3] Massih-Reza Amini and Patrick Gallinari. Semi-supervised logistic regression. In *ECAI*, 2002.

[4] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Y. Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin. *CoRR*, abs/1512.02595, 2015.

[5] Vikram Appia and Umit Batur. Fully automatic 2d to 3d conversion with aid of high-level image features. In *IS&T/SPIE Electronic Imaging*, pages 90110W–90110W. International Society for Optics and Photonics, 2014.

[6] N. N. Author. Suppressed for anonymity, 2011.

[7] Mohammad Haris Baig, Vignesh Jagadeesh, Robinson Piramuthu, Arpit Bhardwaj, Wei Di, and Neel Sundaresan. Im2depth: Scalable exemplar based depth transfer. In *Applications of Computer Vision (WACV), 2014 IEEE Winter Conference on*, pages 145–152. IEEE, 2014.

[8] R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, Princeton, New Jersey, 1961.

[9] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. 2013.

[10] Christopher M Bishop. *Pattern recognition and machine learning*. springer New York, 2006.

[11] Christos Boutsidis, Petros Drineas, and Michael W Mahoney. Unsupervised feature selection for the k-means clustering problem. In *NIPS*, 2009.

[12] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2018.

[13] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, abs/1706.05587, 2017.

[14] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[15] Chun-Hung Cheng, Ada Waichee Fu, and Yi Zhang. Entropy-based subspace clustering for mining numerical data. In *ACM SIGKDD*, 1999.

[16] Adam Coates, Andrew Y Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics*, pages 215–223, 2011.

[17] Fabio Cozman and Eric Krotkov. Depth from scattering. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 801–806. IEEE, 1997.

[18] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[19] Fernando De la Torre and Takeo Kanade. Discriminative cluster analysis. In *ICML*, 2006.

[20] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[21] DMLC. Gluoncv toolkit. `https://github.com/dmlc/gluon-cv`, 2018.

[22] Carl Doersch, Saurabh Singh, Abhinav Gupta, Josef Sivic, and Alexei Efros. What makes paris look like paris? *ACM Transactions on Graphics*, 2012.

[23] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley and Sons, 2nd edition, 2000.

[24] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, pages 2366–2374, 2014.

[25] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[26] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[27] Christoph Fehn. Depth-image-based rendering (dibr), compression, and transmission for a new approach on 3d-tv. In *Electronic Imaging 2004*, pages 93–104. International Society for Optics and Photonics, 2004.

[28] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazırbaş, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *arXiv preprint arXiv:1504.06852*, 2015.

[29] John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. Deepstereo: Learning to predict new views from the world's imagery. *arXiv preprint arXiv:1506.06825*, 2015.

[30] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[31] Boris Ginsburg, Igor Gitman, and Yang You. Large batch training of convolutional networks with layer-wise adaptive rate scaling. 2018.

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

[33] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[34] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. `https://github.com/facebookresearch/detectron`, 2018.

[35] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[36] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[37] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017.

[38] Yves Grandvalet and Yoshua Bengio. Semi-supervised learning by entropy minimization. *NIPS*, 2005.

[39] Sam Gross and Michael Wilber. Training and investigating residual nets. http://torch.ch/blog/2016/02/04/resnets.html.

[40] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 2001.

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[42] 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.

[43] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. *CoRR*, abs/1812.01187, 2018.

[44] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks, 2018.

[45] Zengyou He, Xiaofei Xu, and Shengchun Deng. k-ANMI: A mutual information based clustering algorithm for categorical data. *Information Fusion*, 2008.

[46] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[47] Geoffrey E Hinton. Deep belief networks. *Scholarpedia*, 4(5):5947, 2009.

[48] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[49] Heiko Hirschmüller. Stereo processing by semiglobal matching and mutual information. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(2):328–341, 2008.

[50] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[51] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[52] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 7, 2017.

[53] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269. IEEE, 2017.

[54] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[55] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018.

[56] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[57] M. J. Kearns. *Computational Complexity of Machine Learning*. PhD thesis, Department of Computer Science, Harvard University, 1989.

[58] Janusz Konrad, Meng Wang, Prakash Ishwar, Chen Wu, and Dipankar Mukherjee. Learning-based, automatic 2d-to-3d image and video conversion. *Image Processing, IEEE Transactions on*, 22(9):3485–3496, 2013.

[59] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[60] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[61] M Pawan Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In *NIPS*, pages 1189–1197, 2010.

[62] P. Langley. Crafting papers on machine learning. In Pat Langley, editor, *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pages 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

[63] Quoc V Le. Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8595–8598. IEEE, 2013.

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

[65] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[66] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*, 2015.

[67] David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *JMLR*, 2004.

[68] Tao Li, Sheng Ma, and Mitsunori Ogihara. Entropy-based criterion in categorical clustering. In *ICML*, 2004.

[69] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[70] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[71] Fayao Liu, Chunhua Shen, and Guosheng Lin. Deep convolutional neural fields for depth estimation from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5162–5170, 2015.

[72] Huan Liu and Lei Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 2005.

[73] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[74] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *arXiv preprint arXiv:1411.4038*, 2014.

[75] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[76] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016.

[77] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[78] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, pages 281–297, 1967.

[79] Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.

[80] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors. *Machine Learning: An Artificial Intelligence Approach, Vol. I.* Tioga, Palo Alto, CA, 1983.

[81] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

[82] T. M. Mitchell. The need for biases in learning generalizations. Technical report, Computer Science Department, Rutgers University, New Brunswick, MA, 1980.

[83] Motion Picture Association of America. Theatrical market statistics. 2014.

[84] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[85] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[86] Patrick Ndjiki-Nya, Martin Köppel, Dimitar Doshkov, Haricharan Lakshman, Philipp Merkle, Karsten Müller, and Thomas Wiegand. Depth image-based rendering with advanced texture synthesis for 3-d video. *Multimedia, IEEE Transactions on*, 13(3):453–465, 2011.

[87] Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. Akad. Nauk SSSR*, volume 269, pages 543–547, 1983.

[88] A. Newell and P. S. Rosenbloom. Mechanisms of skill acquisition and the law of practice. In J. R. Anderson, editor, *Cognitive Skills and Their Acquisition*, chapter 1, pages 1–51. Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, 1981.

[89] Feiping Nie, Zinan Zeng, Ivor W Tsang, Dong Xu, and Changshui Zhang. Spectral embedded clustering: A framework for in-sample and out-of-sample spectral clustering. *IEEE Transactions on Neural Networks*, 2011.

[90] Kamal Nigam and Rayid Ghani. Analyzing the effectiveness and applicability of co-training. In *Proc. of the ninth international conference on Information and knowledge management*, 2000.

[91] Hai-Tao Zheng Ningning Ma, Xiangyu Zhang and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. *arXiv preprint arXiv:1807.11164*, 2018.

[92] Chao Peng, Tete Xiao, Zeming Li, Yuning Jiang, Xiangyu Zhang, Kai Jia, Gang Yu, and Jian Sun. Megdet: A large mini-batch object detector. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6181–6189, 2018.

[93] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *CoRR*, abs/1802.01548, 2018.

[94] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[95] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[96] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[97] Amir Rosenfeld, Richard Zemel, and John K Tsotsos. The elephant in the room. *arXiv preprint arXiv:1808.03305*, 2018.

[98] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[99] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):211–229, 1959.

[100] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[101] Ashutosh Saxena, Min Sun, and Andrew Y Ng. Make3d: Learning 3d scene structure from a single still image. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(5):824–840, 2009.

[102] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

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

[104] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[105] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 2014.

[106] Michael Steinbach, Levent Ertöz, and Vipin Kumar. The challenges of clustering high dimensional data. In *New Directions in Statistical Physics*, pages 273–309. Springer, 2004.

[107] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[108] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[109] Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. Learning deep representations for graph clustering. In *AAAI Conference on Artificial Intelligence*, 2014.

[110] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, pages 242–264. IGI Global, 2010.

[111] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[112] Laurens van der Maaten. Learning a parametric embedding by preserving local structure. In *International Conference on Artificial Intelligence and Statistics*, 2009.

[113] Laurens van Der Maaten. Accelerating t-SNE using tree-based algorithms. *JMLR*, 2014.

[114] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *JMLR*, 2008.

[115] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *JMLR*, 2010.

[116] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 2007.

[117] Limin Wang, Yuanjun Xiong, Zhe Wang, and Yu Qiao. Towards good practices for very deep two-stream convnets. *arXiv preprint arXiv:1507.02159*, 2015.

[118] K Wong and F Ernst. *Single Image Depth-from-Defocus*. PhD thesis, Netherlands: Delft university of Technology & Philips Natlab Research, Eindhoven, 2004.

[119] Shiming Xiang, Feiping Nie, and Changshui Zhang. Learning a mahalanobis distance metric for data clustering and classification. *Pattern Recognition*, 2008.

[120] Junyuan Xie, Ross Girshick, and Ali Farhadi. Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks. In *European Conference on Computer Vision*, pages 842–857. Springer, 2016.

[121] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pages 478–487, 2016.

[122] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017.

[123] Eric P Xing, Michael I Jordan, Stuart Russell, and Andrew Y Ng. Distance metric learning with application to clustering with side-information. In *NIPS*, 2002.

[124] Donghui Yan, Ling Huang, and Michael I Jordan. Fast approximate spectral clustering. In *ACM SIGKDD*, 2009.

[125] Yi Yang, Dong Xu, Feiping Nie, Shuicheng Yan, and Yueting Zhuang. Image clustering using local discriminant models and global integration. *IEEE Transactions on Image Processing*, 2010.

[126] Jieping Ye, Zheng Zhao, and Mingrui Wu. Discriminative k-means for clustering. In *NIPS*, 2008.

[127] Dong Yu and Li Deng. Automatic speech recognition: A deep learning approach, 2015.

[128] Dong Yu, Li Deng, and George Dahl. Roles of pre-training and fine-tuning in context-dependent dbn-hmms for real-world speech recognition. In *Proc. NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

[129] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[130] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*. 2014.

[131] Hang Zhang, Kristin Dana, Jianping Shi, Zhongyue Zhang, Xiaogang Wang, Ambrish Tyagi, and Amit Agrawal. Context encoding for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

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

[133] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *CoRR*, abs/1710.09412, 2017.

[134] Liang Zhang, Carlos Vázquez, and Sebastian Knorr. 3d-tv content creation: automatic 2d-to-3d video conversion. *Broadcasting, IEEE Transactions on*, 57(2):372–383, 2011.

[135] Zhi Zhang, Tong He, Hang Zhang, Zhongyuan Zhang, Junyuan Xie, and Mu Li. Bag of freebies for training object detection neural networks, 2019.

[136] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 6230–6239. IEEE, 2017.

[137] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1529–1537, 2015.

[138] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 2017.

[139] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[140] Shaojie Zhuo and Terence Sim. On the recovery of depth from a single defocused image. In *Computer Analysis of Images and Patterns*, pages 889–897. Springer, 2009.

[141] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017.

VITA

Junyuan Xie received his Bachelor of Science degree from University of Science and Technology of China. He is currently a PhD candidate in the Paul G. Allen School of Computer Science and Engineering at University of Washington.