A STUDY ON DEEP LEARNING: TRAINING, MODELS AND APPLICATIONS

HENGYUE PAN

A DISSERTATION SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO
April 2017

Abstract

In the past few years, deep learning has become a very important research field that has attracted a lot of research interests, attributing to the development of the computational hardware like high performance GPUs, training deep models, such as fully-connected deep neural networks (DNNs) and convolutional neural networks (CNNs), from scratch becomes practical, and using well-trained deep models to deal with real-world large scale problems also becomes possible. This dissertation mainly focuses on three important problems in deep learning, i.e., training algorithm, computational models and applications, and provides several methods to improve the performances of different deep learning methods.

The first method of this dissertation is a DNN training algorithm called Annealed Gradient Descent (AGD). During the DNN optimization procedure, AGD optimizes a sequence of gradually improved smoother mosaic functions that approximate the original non-convex objective function according to a pre-defined annealing schedule. This dissertation presents a theoretical analysis on the convergence properties and learning speed of AGD to show its benefits. The proposed AGD algorithm is applied to learn DNNs for hand-written digital recognition in MNIST database and speech recognition in Switchboard database. Experimental results have shown that AGD yields comparable performance as SGD but it can significantly expedite training of DNNs in big data sets (by about 40% faster).

Secondly, this dissertation proposes to apply a novel model, namely Hybrid Orthogonal Projection and Estimation (HOPE), to CNNs to introduce orthogonality into the network structure. HOPE can be viewed as a hybrid model to combine feature extraction with mixture models. It is an effective model to extract useful information from the original high-dimension feature vectors and meanwhile filter out irrelevant noises. In this work, three different ways to apply the HOPE models to CNNs, i.e., *HOPE-Input*, *single-HOPE-Block* and *multi-HOPE-Blocks*, are presented. For *HOPE-Input* CNNs, a

HOPE layer is directly used right after the network input to de-correlate high-dimension input feature vectors. Alternatively, in *single-HOPE-Block* and *multi-HOPE-Blocks* C-NNs, the HOPE layers are used to replace one or more blocks in the CNNs, where one block may include several convolutional layers and one pooling layer. The experimental results on CIFAR-10, CIFAR-100 and ImageNet databases have shown that the orthogonal constraints imposed by the HOPE layers can significantly improve the performance of CNNs in the image classification tasks. In the CIFAR experiments, HOPE CNNs achieve one of the best performance when image augmentation has not been applied, and top 5 performance with image augmentation.

The third proposed method is to apply CNNs to a task called image saliency detection, and achieves good performance. In this approach, a gradient descent method is used to iteratively modify the input images based on pixel-wise gradients to reduce a pre-defined cost function, which is defined to measure the class-specific objectness and clamp the class-irrelevant outputs. This designation can help to maintain image background when modify the images. The pixel-wise gradients can be efficiently computed using the back-propagation algorithm. Moreover, SLIC superpixels and LAB color based low level saliency features are applied to smooth and refine the saliency maps. The CNNs based saliency method is quite computationally efficient, much faster than other state-of-the-art deep learning based saliency methods. Experimental results on two benchmark tasks, namely Pascal VOC 2012 and MSRA10k, have shown that the proposed methods can generate high-quality salience maps, at least comparable with many slow and complicated deep learning methods.

The last method is also for image saliency detection tasks. However, this method is based on a class of deep learning model called Generative Adversarial Network (GAN). The proposed method follows the basic idea of GAN, i.e., training two models simultaneously: the G-Network takes images as inputs and can generate corresponding synthetic saliency maps, and the D-Network can determine if the sample is a synthetic saliency map rather than ground-truth saliency map. However, different from GAN, the proposed method uses fully supervised learning to learn both G-Network and D-Network. Therefore, the proposed method is called Supervised Adversarial Network (SAN). Moreover, SAN introduces a different G-Network and conv-comparison layers to further improve the saliency performance. Experimental results show that the SAN model can also generate state-of-the-art saliency maps for complicate images.

Acknowledgments

First and foremost, I would like to express my highest appreciation to my supervisor Professor Hui Jiang for his kind and continuous support to my study and research. He tells me how to find good ideas in the related research field and implement them, and also gives me enough freedom to work by myself. By discussing with him, I can gradually improve my idea and achieve better results.

I am also thankful to Professor Shiyao Jin in National University of Defense Technology (NUDT) in China. He is the supervisor of my graduate study from 2010 to 2012. He helped me to make change from a undergraduate student to a graduate student. I learned that courses study is not all for graduate students. Instead, we need to do research, read papers and try to publish our own research results. By working in his research group, I began to learn the basic ideas of computer vision and read a lot of related papers, which are important for my further research in York University.

I hope to express gratitude to my co-supervisors Professor Aijun An and Professor James Elder. They also provide me with a lot of help through the daily discussion and annual meetings. With their help, I passed the qualifying examination and thesis proposal, and work well on my research.

I would like to express gratitude to China Scholarship Council (CSC), which provides me with scholarship to support my Ph.D. study in York University. The scholarship helps me a lot and make me do not need to worry about money matters.

I would like to express appreciation to York University and the department of electrical engineering and computer science (EECS) for the supports from the related faculties. Without their help, I cannot solve many problems such as tuition waive, medical insurance and server problems.

I hope to express sincere gratitude to my friends and teammates in York University, iFlytek and NUDT. Dr. Ossama Abdel-Hamid helped me a lot when I entered the research

group. He told me how to implement neural networks and how to use the computing resources of our group. Dr. Cong Liu, Quan Liu, Shiliang Zhang and Ziyang Wu helped me in both work and daily life when I did my internship in iFlytek. Professor Zhiying Wang, Mrs. Hongjin Lin, Mrs. Ling Wen and Mr. Han Li gave me important suggestions about how to do research effectively and efficiently and how to find job in universities.

I would like to express warm and kindly appreciation to my parents. They not only gave me life 29 years also, but also guide me to learn how to do a good person and how to find my target and work for it. Their love and encouragements are irreplaceable in my life.

At last, I hope to express my sincere gratitude to one of my best friend, Dr. Jianbiao Mao. He shared the same student apartment with me in NUDT for 5 years and participated a lot of important moment in my life. When I prepared for the Graduate Entrance Exam in NUDT, he worked with me and we encouraged each other. Before I went to Canada, he gave me a *Lonely Planet (Canada)* to help my daily life. Dr. Jianbiao Mao died on Oct. 12, 2016 at the age 28 because of myelofibrosis and leucocythemia, and I always deeply miss him.

Contents

List of Tables

List of Figures

List of Acronyms

AGD	Annealed Gradient Descent
ANN	Artificial Neural Network
ASGD	Average Stochastic Gradient Descent
ASR	Automatically Speech Recognition
BP	Back-Propagation
CE	Cross-Entropy
CNN	Convolutional neural network
CPU	Central processing unit
DNN	Deep Neural Network
GAN	Generative Adversarial Network
GD	Gradient Descent
GMM	Gaussian mixture model
GPU	Graphics Processing Unit
HMM	Hidden Markov model
MCCNN	Multi-Colunm CNNs
MLP	Multi-Layer Perceptron
PCA	Principal Components Analysis
PER	Phone error rate
RBM	Restricted Boltzmann Machine
RCNN	Recurrent Convolutional Neural Networks
ReLU	Rectified Linear Unit
SAN	Supervised Adversarial Network
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
SVRG	Stochastic Variance Reduced Gradient
WER	Word error rate
ZCA	Zero-phase Component Analysis

Chapter 1

Introduction

The development of deep learning is a very long story, which can be traced back to 1950s. The presenting of two layer perceptron [13] made the researchers believe that they have revealed the truth of artificial intelligence. However, in 1969, Minsky et al. published a book called *Perceptrons* [14], in which they prove that perceptrons have some weakness such as cannot express some complicate functions. Moreover, they argued that multi-layer perceptrons (MLP) have no research value because of the limitation of computational ability and the lack of learning algorithms. This book can be viewed as an important label of the period called 'AI winter'. In 1980s, Rumelhart et al. presented the famous back-propagation (BP) algorithm [15], which solved some problems of gradient calculation through the introduction of chain rules. Thus the training of MLP became practical. The introduction of hidden layers equip perceptrons with ability of simulate all possible functions [16]. Starting from the BP algorithm, researchers tend to consider neural networks as a mathematical problem instead of biological problem. Even though the BP algorithm brought benefits on the learning of MLPs, it still suffered from low efficiency and many local optima. Moreover, too many hyper-parameters such as learning rate, momentum and number of hidden nodes also obstructed the widely usage of MLPs. In 1990s, Cortes and Vapnik defined support vector machine (SVM) [17], which showed many advantages compare with MLPs, such as global optimum point, faster learning speed and less hyper-parameters. SVM caused another winter of MLPs, and this situation lasted more than 10 years until the presenting of the fast learning method of deep belief networks [18], where the layer-wise pre-training technique and network fine-tuning are combined to speed-up the training process. After that, deep learning began to show excellent performance on different tasks such as speech processing and computer vision due to the development of computer hardware, especially the

General-Purpose computing on Graphics Processing Units (GPGPU). GPGPU results in hundreds or even thousands times improvement on the computational performance. Also, increasing number of training data are collected for different tasks. And as a result, training deep learning models like fully connected deep neural networks (DNNs) and deep convolutional neural networks (CNNs) from scratch becomes possible. Therefore, currently deep learning models such as DNNs, CNNs, RNNs (Recurrent Neural Networks) and LSTMs (Long Short Term Memory networks) become an important branch in artificial neural networks (ANNs).

Generally speaking, ANNs are computational models that inspired by an human's brain systems. It can be described as highly interconnected "neurons", which can receive input vectors and output corresponding features. A typical ANN contains three layers: the input layer, hidden layer and output layer. DNNs is the ANNs that contain at least two hidden layers, which may result in more powerful representation ability because the deep structures can extract increasingly abstract features of input data. DNNs serve as an important branch of deep learning, which can be applied to variety application fields.

CNNs [19] is a kind of feed-forward ANNs. A lot of experiments prove that Fully-connected DNNs are powerful for pattern recognition tasks. However, there still remains some problems of DNNs especially on computer vision tasks [20], such as the lack of invariance and the deficiency to catch the local features. CNNs are proposed to solve those difficulties. For instance, CNNs provide shift invariance by using weights sharing method and extract local features by using local receptive fields. Therefore, CNNs become the major model for image related tasks.

More recently, DNNs and CNNs not only work well on the classification-related tasks, but also other more complicated tasks. For instance, many computer vision tasks such as image generation [21] and image saliency detection [12] can be addressed by using deep learning models. A lot of researches reflect that DNNs and CNNs have ability to handle more complex features from large amount of training data. However, there are still some challenging research topics in deep learning, such as how to further speed up the training process, how to find a good model to boost the performance, and how to apply deep learning methods more effectively in different tasks.

1.1 Contributions

This dissertation proposes several novel methods, which include training algorithm, new model and applications, to improve the performance of DNNs and CNNs.

First of all, the dissertation provides a novel DNN training algorithm called Annealed Gradient Descent (AGD) to increase the training speed and prevent the training process from finding some bad local minima. Comparing with the regular stochastic gradient descent (SGD), AGD tries to optimize a low resolution approximation function that derives from a group of pre-trained codebooks instead of original training data. Furthermore, the approximation resolution is gradually improved according to an annealing schedule over the optimization course. The original non-convex objective function will be optimized at the end of training. This dissertation theoretically proves that AGD can find good optimization results with fast learning speed. The details of AGD algorithm are shown in Chapter 4.

Secondly, the dissertation proposes to use a novel model called Hybrid Orthogonal Projection and Estimation (HOPE) in CNNs for better classification performance. This model introduces a linear orthogonal projection to reduce the dimensionality of the raw high-dimension data and then uses a finite mixture distribution to model the extracted features. By splitting the feature extraction and data modeling into two separate stages, it may derive a good feature extraction model that can generate better low-dimension features for the further learning process. Based on the analysis in [22], this dissertation finds a suitable way to apply HOPE to CNNs and gets state-of-the-art performance on image classification tasks on CIFAR-10, CIFAR-100 [23] and ImageNet [24] databases. Chapter 5 discusses the details of HOPE CNNs.

Besides training algorithm and new model, this dissertation also considers applications of deep learning on image saliency detection, which is a kind of important task in computer vision. In Chapter 6, a CNN-based saliency method is presented, which is a much simpler and more computationally efficient method to generate class-specific object saliency maps directly from a well-trained classification CNN model. This approach use the gradient descent (GD) method to iteratively modify input images based on pixel-wise gradients to reduce a pre-defined cost function. The experimental results show that the CNN-based saliency method can generate high-quality saliency maps in relatively short time (more than 3 times faster than the state-of-the-art deep learning based method in [12]). Chapter 7 precents another novel saliency detection method based on Generative Adversarial Network (GAN) [21], which is called Supervised Adversarial Network

(SAN). Specifically, SAN also introduces two CNNs: the G-Network is designed to generate synthetic saliency maps from the input images, and the D-Network is used to determine the given saliency map is a synthetic saliency map or a ground truth saliency map. However, different from the regular GAN in [21], SAN applies fully supervised learning method to update both G-Network and D-Network, uses different G-Network structure, and introduces a new type of hidden layer called 'conv-comparison layer' to make the model more suitable for saliency detection instead of image generation. Experiments show that the SAN model can also provide state-of-the-art saliency detection performance on many complex images.

1.2 Outline

In this dissertation, the next two chapters (Chapter 2 and Chapter 3) provide some important background knowledge related to ANNs and image processing. Chapter 4 describes the AGD training algorithm, which includes theoretical analysis and experiments. Chapter 5 describes the proposed HOPE CNN model, which firstly describes the basic idea of HOPE and then shows how to apply HOPE on CNNs. Chapter 6 and Chapter 7 presents two different way to apply deep learning models for image saliency detection: Chapter 6 shows the CNN-based fast saliency method, and Chapter 7 provides the SAN model for saliency detection. Finally, Chapter 8 concludes this dissertation and discusses some potential future works.

Chapter 2

Artificial Neural Networks

Artificial Neural Networks (ANNs) are proposed to simulate the natural human brains for good pattern recognition ability. They can be described as highly interconnected "neurons" which can receive input vectors and output the corresponding features. In 1943, Warren McCulloch and Walter Pitts proposed a computational model called threshold logic, which can be viewed as an early prototype of ANNs [25]. Their research showed that ANNs are a kind of potential tools to implement artificial intelligence. In 1958, Frank Rosenblatt proposed the basic idea of perceptron, which is a two-layer neural network (one input layer and one output layer) with linear connection [13]. In later 1980s, the development of the famous back-propagation algorithm [15] make it possible to train multi-layer ANNs such as multi-layer perceptron(MLP). Moreover, by using non-linear activation functions such as sigmoid and relu, multi-layer perceptron has been proved that it has ability to approximate all possible functions in real-world [26]. However, because of the insufficiency of the computational performance and lack of training data, training large scale ANNs (e.g. DNNs and CNNs) was impractical. Fortunately, in 2007, NVIDIA released the first edition of CUDA. CUDA is a GPU based parallel computing toolkit, which enables researchers to use GPU for general purpose computing. This brings about hundreds times speed-up in the training of ANNs. Moreover, some large-scale databases such as ImageNet [24] and COCO [27] are released, which can help the learning of deep models. Therefore, ANNs begin to going deeper and deeper and yield excellent performance in variety of applications.

Recently, deep learning has achieved significant successes in many real-world applications, such as speech recognition and computer vision. It becomes a very interesting problem to learn large-scale deeply-structured ANNs, such as DNNs and CNNs, from

big databases. Deep learning algorithms can detect both low-level and high-level features from the input data automatically, and as a result, they may have excellent performance in practice.

When learning deep models, the back-propagation algorithm, sometimes being combined with layer-wise greedy pre-training, is the most important training method. The back-propagation algorithm collects error signals from the output layer by comparing network outputs and training targets and back-propagates the error signals layer-by-layer to efficiently derive gradients with respect to all network weights. The weights are then updated by using the gradient descent algorithm. However, since deep models are highly non-convex in nature, the learning process may finally result in a bad local optima. Moreover, since the deep models contain a large number of free parameters, a finite size training set may lead to over-fitting. A lot of efforts aim to deal with those problems, such as data augmentation, dropout, and the proposed AGD algorithm in this dissertation.

This chapter reviews two famous deep learning models: DNNs and CNNs, as well as their training methods and applications.

2.1 Deep Neural Networks

Deep Neural Networks (DNNs) is the fully-connected ANNs that contain at least two hidden layers (see an example in Figure 2.1), which may result in more powerful representation ability because the deep structures can extract increasingly abstract features of input data. DNNs serve as an important branch of deep learning, which can be applied in variety application fields. This section reviews the basic model of DNNs and their training algorithm, and also discuss some applications.

2.1.1 Basic Model

[28] provides the basic computational model of DNNs. Assuming that a DNN has L hidden layers, and given an input vector $\mathbf{x}$, then some important notations can be defined:

1. $\mathbf{W}^l$ and $\mathbf{b}^l$: the weight matrix and bias of the lth hidden layer

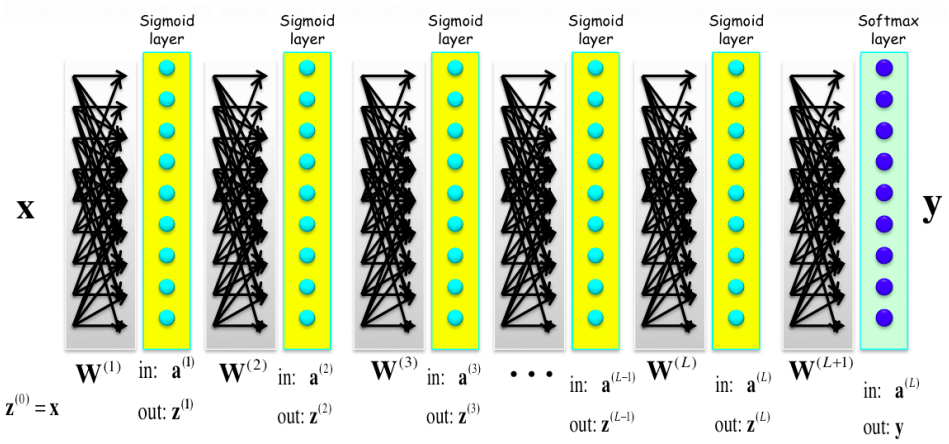

FIGURE 2.1: Structure of a Deep Neural Network

2. $\mathbf{z}^l$: the output of the lth hidden layer, and $\mathbf{z}^l = f(\mathbf{a}^l)$, where $f(\cdot)$ is a non-linear activation function

3. $\mathbf{a}^l$: the activation of the lth hidden layer, and $\mathbf{a}^l = \mathbf{W}^l\mathbf{z}^{l-1} + \mathbf{b}^{l-1}$

4. $\mathbf{o} = \mathbf{z}^{L+1}$: the final output of the DNN, and $\mathbf{o} = softmax(\mathbf{a}^{L+1})$

where the definition of the softmax function is:

$$softmax(x) = \frac{\exp(x)}{\sum_j \exp(x)} \tag{2.1}$$

For the hidden layers, an activation function $f(\cdot)$ should be pre-defined. One well-known and successful activation function is sigmoid function (see Figure 2.2):

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

Using sigmoid function needs to carefully fine-tune some hyper-parameters including learning rate, momentum and mini-batch size to get the best performance. More importantly, for deep models, applying sigmoid function may result in the problem of gradient vanishing.

Currently, a novel activation function called rectified linear unit (ReLU) [1] is becoming increasingly prevalent. The form of ReLU is $f(x) = \max(0, x)$, which is much simpler than the standard sigmoid function. Using ReLU may increase the training speed of DNNs because its gradient can be calculated much easier (Figure 2.3). Moreover, large number of experiments suggest that ReLU DNNs allow to use much larger mini-batch

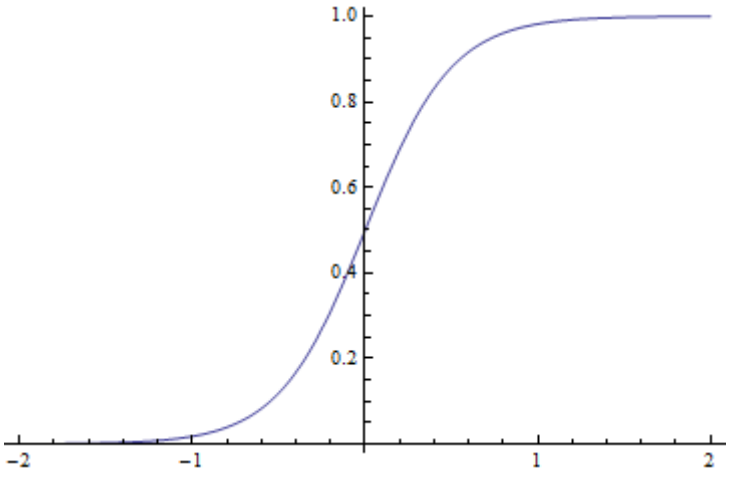

FIGURE 2.2: Sigmoid Function

size to get excellent performance, while sigmoid function requires smaller mini-batch size [29]. This means that the ReLU DNNs can be parallelized much easier by introducing GPU computing or mulit-CPUs.

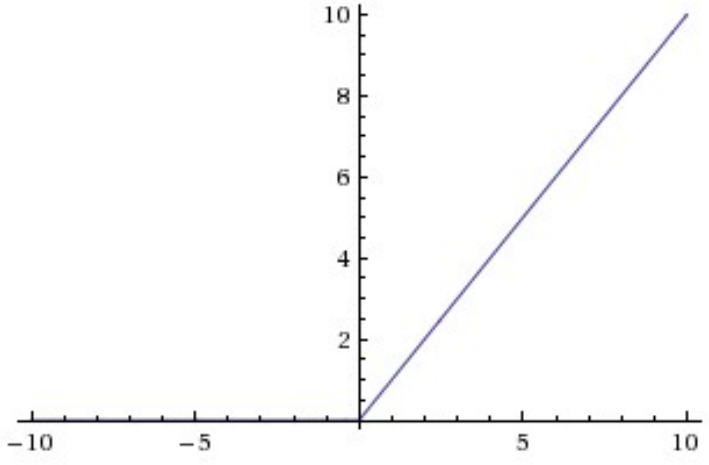

FIGURE 2.3: ReLU Function

Furthermore, He et al. proposed a generalization of ReLU, i.e. Parametric Rectified Linear Unit (PReLU), to further increase the performance of ReLU [30]. Compare with regular ReLU, PReLU has a different definition when the input x less than zero:

$$f(x) = \begin{cases} x & \text{x} \geq 0, \\ ax & \text{x} < 0. \end{cases} \tag{2.3}$$

where a is a learnable hyper-parameter. Comparing with the constant a in Leaky ReLU (LReLU) in [31], PReLU is more flexible and has slightly better performance on many databases.

2.1.2 Network Training

DNNs can be trained to optimize the cross-entropy (CE) criterion, which measures the difference between the desired labels and the actual outputs calculated by DNNs, and the standard error back-propagation algorithm can be applied through Stochastic Gradient Descent (SGD). By merging the bias $\mathbf{b}^l$ into the weight matrix $\mathbf{W}^l$, only the iterative updating rule of $\mathbf{W}^l$ need to be considered by using gradient information as below:

$$\mathbf{W}^l = \mathbf{W}^l - \lambda \frac{\partial \mathcal{F}_{CE}}{\partial \mathbf{W}^l}, \ (1 < l < L+1) \tag{2.4}$$

where λ is the learning rate.

For any input vector $\mathbf{x}$, the output of a given DNN is calculated using the softmax function as follows:

$$\mathbf{o}_{nt} = \Pr(t_n|\mathbf{x}_n) = \frac{e^{\mathbf{a}_{nt}^{L+1}}}{\sum_{t'} e^{\mathbf{a}_{nt'}^{L+1}}} \tag{2.5}$$

where a_{nt}^{L+1} denotes the activation signal at the output layer corresponding to the nth training sample pair and the output label t_n. Assuming that the whole training set consists of all training samples and their corresponding target labels, the CE objective function can be expressed as the following form:

$$\mathcal{F}_{CE}(\mathbf{W}) = -\sum_{n=1}^{N} \log \Pr(t_n|\mathbf{x}_n) = -\sum_{n=1}^{N} \log \mathbf{o}_{nt} \tag{2.6}$$

where $\mathbf{W}$ denotes all connection weights in the DNN, and $\mathbf{o}_n$ is the output at the top softmax layer given the input vector $\mathbf{x}_n$.

In the standard back-propagation (BP) algorithm, the most important quantities need to be calculated are the gradients of the CE objective function with respect to the input

activations, a_j^l, at each layer, which are normally called as *error signals*:

$$e_{nj}^l \equiv -\frac{\partial}{\partial \mathbf{a}_j^l} \log \Pr(t_n|\mathbf{x}_n). \tag{2.7}$$

For the CE objective function in Eq. (2.6), the error signals at the output layer can be computed as:

$$e_{nt}^{L+1} = -\frac{\partial \log \mathbf{o}_{nt}}{\partial \mathbf{a}_{nt}} = \mathbf{o}_{nt} - \delta_{nt} \tag{2.8}$$

where $\delta_{nt} = 1$ if t equals to the target label t_n and otherwise $\delta_{nt} = 0$. And the error signals for all other layers ($1 \leq l \leq L$) can be calculated from e_{nt}^{l+1} following a standard error back-propagation procedure as follows:

$$\mathbf{e}^l = (\mathbf{W}^{l+1})' \mathbf{e}^{l+1} \cdot \nabla \mathbf{z}^l \quad (l = L, ..., 1) \tag{2.9}$$

where the vector $\mathbf{e}^l$ is constructed by concatenating all error signals in layer l, $\cdot$ stands for the element-wise multiplication between two vectors with the same size, and ∇ means to calculate the gradients.

Once all error signals for all nodes j in all layers l given each input vector are provided, the gradients for all DNN weights can be easily derived from these error signals. In this case, the gradients with respect to all DNN weights can be easily obtained as:

$$\frac{\partial \mathcal{F}_{CE}(\mathbf{W})}{\partial \mathbf{W}^l} = \mathbf{e}^l \nabla \mathbf{z}^{l-1} \quad (\forall\, l) \tag{2.10}$$

Substituting Eq. (2.10) into Eq. (2.4), all DNN weights can be updated iteratively based on SGD algorithm.

However, training DNNs by using back-propagation is a difficult task. In [32], Erhan et al. argue that one important challenge of training deep models is that the objective functions are always highly non-convex. Therefore, it may exist numerous local minimum points, and the gradient based learning algorithm may easy to get stuck into bad local optima. Applying SGD may help to relieve this situation, since the random noise introduced by each training sample may help the algorithm to escape from some bad local minima, but the random noise may also slow down the learning speed.

One possible way to deal with this difficulty is to initialize the DNNs by using greedy layer-wise pre-training under some unsupervised criterions [28], where Restricted Boltzmann Machines (RBMs) or auto-encoders can be employed. A hypothesis that can explain the effects of unsupervised pre-training is proposed in [32]. In this paper, Erhan et al. present that the pre-training procedure can restrict the model parameters inside a relatively small region of parameter space, which corresponds to capturing structure of the input distribution $P(\mathbf{x})$ in each layer, and those information catched by pre-training process can serve as a good initialization for the further supervised fine-tuning. The experiments show that unsupervised pre-training is robust to the random initialization and can provide the DNNs with better generalization performance.

Another potential weakness of DNNs training is over-fitting. The increasing number of hidden layers cause more and more adjustable model parameters, and over-fitting then becomes easier to happen. Even though some strategies like early stopping and regularization schemes can be used to prevent over-fitting, it still remains a major challenge when training some extremely large DNNs with relatively small size training set. Model combination is a good method to relieve over-fitting. Unfortunately, training large-scale DNNs is time-consuming, and as a result, training several DNNs for model combination often becomes impractical. To deal with those problems, previous researchers proposed dropout method [33].

The basic idea of dropout is to randomly drop each unit in a DNN with probability p ($p = 0.5$ for example), and "drop" means that remove the unit and its related edges from the network. During the training process, dropping out should be done independently for each hidden unit and for each training sample. As a result, dropout can be regarded as a sub-sampling of the DNN, and a DNN with n hidden units can be viewed as a collection of 2^n possible weight shared sub-DNNs.

When dropout is combined with DNN training, the feed forward operation can be defined as below:

$$\begin{aligned} r_i^l &\sim \text{Bernoulli}(p) \\ \tilde{\mathbf{z}}^l &= \mathbf{r}^l \cdot \mathbf{z}^l \\ \mathbf{a}^{l+1} &= \mathbf{W}^{l+1}\tilde{\mathbf{z}}^l + \mathbf{b}^{l+1} \\ \mathbf{z}^{l+1} &= f(\mathbf{a}^{l+1}) \end{aligned} \tag{2.11}$$

where $\mathbf{r}^l$ is a vector that contains Bernoulli random variables to determine that whether one node in layer l should be dropped, and each element in $\mathbf{r}^l$ has probability p of

equaling to 1. During the learning, the gradient of the loss function should be back propagated through the dropped network.

In the test phase, the DNN weights need to be re-scaled: $\mathbf{W}_{test} = p\mathbf{W}$. The purpose of this operation is to approximate the average over all possible dropped network. By doing this, the dropout DNNs can share some advantages of model combination method.

In [1], Dahl et al. combine rectified linear activation function and dropout during the DNN training, and provide experiment results on DARPA EARS database, which contains 50 hours English broadcast news data. The test results are listed in Table 2.1. This table implies that dropout combine with rectified linear can provide better performance even though without pre-training.

TABLE 2.1: Performance on DARPA EARS Database [1]

ACTIVATION FUNC	PRE-TRAINING	DROPOUT	TRAINING ERR	TEST ERR
RECTIFIED LINEAR	YES	YES	10.7%	18.5%
RECTIFIED LINEAR	NO	YES	11.0%	18.9%
SIGMOID	YES	NO	11.3%	19.4%

Based on dropout, several generalization methods have been presented. Wan et al. designed dropconnect algorithm in [34]. Instead of dropping a subset of units, dropconnect randomly selects some elements in the weight matrices and set them to 0. Experiments on some databases show that dropconnect can provide better performance compare with regular DNNs and even dropout. In [35], an adaptive edition of dropout called standout is proposed, and the basic idea is to overlay a binary belief network on the DNN and use it to regularize the hidden units by selectively setting activities to 0. This overlaid belief network can be trained during the DNN training. The experiments show that standout can also provide lower error rate than standard dropout.

2.2 Deep Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [19] also belong to feed-forward ANNs. As this dissertation mentioned before, the fully-connected DNNs are powerful when doing pattern recognition tasks. However, there are still some problems of DNNs especially when deal with image processing tasks [20].

Firstly, typical images (grey-scale or color images) may corresponding to large input features, which will obviously increase the size of weight matrices, and then may lead to over-fitting problem and require more memory resources. These difficulties sometimes make the DNNs become impractical in real-world image-related applications.

Secondly, the fully-connected DNNs are short of invariance of translations and local distortions of the input features. Even though pre-processing of input images can be performed to provide some invariance, they are far from perfect and may easy to cause variations of the inputs. It is true that increasing the size of DNNs may handle those variations, but it requires large number of training samples and memory size, and also makes the training process much slower.

Thirdly, images have a lot of 2D local structures, but this kind of local features cannot be reflected in fully-connected DNNs because the input order of DNNs cannot affect the network output, and this may also bring about negative impacts on the recognition performance.

CNNs are proposed to solve those difficulties. For instance, CNNs can provide shift invariance by using weights sharing technique and can extract local features by using local receptive fields.

2.2.1 Basic Model

A typical CNN has three kinds of layers: convolutional layers, pooling layers and fully-connected layers, and in most cases one or more convolutional layers and one pooling layer can be grouped as one block (see Figure 2.4 as an instance).

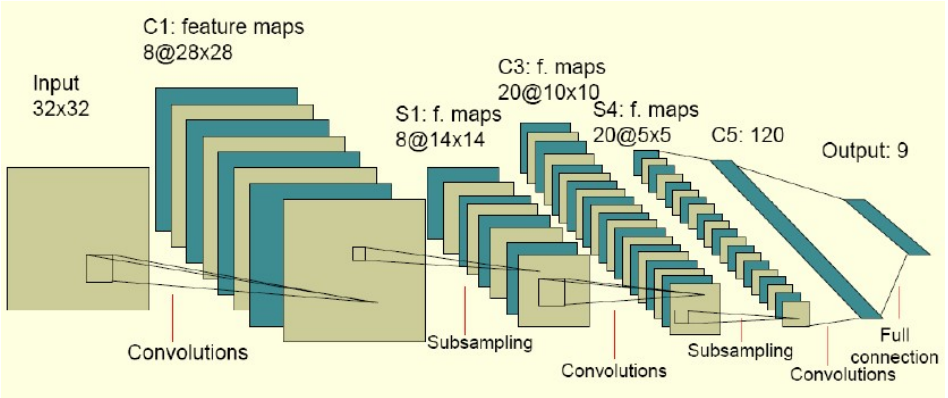

FIGURE 2.4: An Example of Deep Convolutional Neural Networks

1. **Convolutional Layers**

As Figure 2.4 show, one convolutional layer of CNNs may contain several feature maps, and each feature map is related to a local receptive field, which scans the previous layer and receive signals from local areas. Obviously, this operation is equivalent to a convolution with a small size window [19]. In addition, images have a property that one specific pattern may appear in the entire image. To utilize this nature, the weight sharing technique is introduced in each feature map, which means that the same weights will be used to process all local regions in the input feature maps. Another benefit of weight sharing is that it can significantly reduce the number of free parameters.

To compute the output of one feature map, the convolution kernel w with the size k should be defined at first, and then the output of the lth layer can be calculated as below:

$$\begin{aligned} a^l_{x,y} &= w^l_{x,y} * h^l_{x,y} + b^l_{x,y} \\ &= \sum_{x'=0}^{k-1} \sum_{y'=0}^{k-1} w^l_{x',y'} f(a^{l-1}_{x-x',y-y'}) + b^l_{x,y} \end{aligned} \tag{2.12}$$

where f denotes a pre-defined activation function (which is similar as DNNs), and x, y have the range $0 < x < k-1$ and $0 < y < k-1$ are the range that pixels are defined. For the undefined pixels, the zero-padding can be used to make the output feature map has the same size as input.

2. **Pooling Layers**

To increase the robustness of CNNs, one possible way is to reduce the resolution of higher layers because it can make CNNs tolerate to a slight distortion or translation in the input features [19]. Therefore, pooling layers can be used after one or more convolutional layers to do down-sampling and then reduce the resolution of the feature maps. The pooling layers perform down-sampling by using a small pooling window and calculate one single value to replace all values inside the window, thus it generates smaller feature maps than the previous convolution layer.

In practice, two kinds of pooling functions are usually used: maximum pooling and average pooling [7]. The maximum pooling function function is defined in Eq. (2.13):

$$p_{i,j} = \max_{m=1}^{G} \max_{n=1}^{G} q_{(i-1)\times s+m,(j-1)\times s+n} \tag{2.13}$$

where i and m are the index in one feature map, G is the pooling size, and s is the shift size that control the overlap of adjacent pooling windows. Similarly, the average

pooling function can be defined as below:

$$p_{i,j} = r \sum_{m=1}^{G} \sum_{n=1}^{G} q_{(i-1)\times s+m,(j-1)\times s+n} \tag{2.14}$$

where r denotes a scaling factor.

The pooling layers in CNNs may be viewed as a feature extraction procedure, which extract the most prominent signal component within a small neighbouring region. The pooling layers, using either average pooling or max pooling, are critical in CNNs [36] since they can make the models more tolerable to the slight distortion or translation in the original images [19]. In [37], a theoretical analysis of average pooling and max pooling is made to reflect how pooling can affect the network performance. However, in most cases, pooling layers are still used based on empirical information.

3. **Fully Connected Layers**

To further improve the performance of CNNs, several fully connected layers can be applied upon the convolution layers and pooling layers. The structure of the fully connected layers are identical with regular DNNs.

2.2.2 Network Training

Similar with DNNs, error back-propagation algorithm can also be used to train CNNs. Because of the existence of pooling layers and the usage of weight sharing, the back-propagation needs some small modification. When propagating error signals through a pooling layer, the kind of pooling function should be considered. For example, for the maximum pooling function, only the signal that passed the most active unit in each group of the pooled units should be kept [7].

To update the shared weights in one convolution kernel, the weights updates over all units in the corresponding feature maps should be collected. Specifically, the gradient of the error signal E with respect to the convolution kernel of the lth layer w^l can be calculated as below:

$$\frac{\partial E}{\partial w^l_{x,y}} = \sum_{x'=0}^{k-1} \sum_{y'=0}^{k-1} \frac{\partial E}{\partial a^l_{x',y'}} \frac{\partial a^l_{x',y'}}{\partial w^l_{x,y}} = \sum_{x'=0}^{k-1} \sum_{y'=0}^{k-1} \delta^l_{x',y'} \frac{\partial a^l_{x',y'}}{\partial w^l_{x,y}} \tag{2.15}$$

By doing some simple transformation, $\frac{\partial a^l_{x',y'}}{\partial w^l_{x,y}}$ can be re-written as:

$$\begin{aligned}\frac{\partial a^l_{x',y'}}{\partial w^l_{x,y}} &= \frac{\partial}{\partial w^l_{x,y}}(\sum_{x''=0}^{k-1}\sum_{y''=0}^{k-1} w^l_{x'',y''} f(a^{l-1}_{x'-x'',y'-y''}) + b^l) \\ &= \frac{\partial}{\partial w^l_{x,y}}(w^l_{x,y} f(a^{l-1}_{x'-x,y'-y})) = f(a^{l-1}_{x'-x,y'-y})\end{aligned} \tag{2.16}$$

Therefore, Eq. (2.15) can be re-written as:

$$\begin{aligned}\frac{\partial E}{\partial w^l_{x,y}} &= \sum_{x'=0}^{k-1}\sum_{y'=0}^{k-1} \delta^l_{x',y'} f(a^{l-1}_{x'-x,y'-y}) \\ &= \delta^l_{x,y} * f(a^{l-1}_{-x,-y}) = \delta^l_{x,y} * f(rot_{180^\circ}(a^{l-1}_{x,y}))\end{aligned} \tag{2.17}$$

where rot_{180° means to rotate the matrix for 180 degrees. In Eq. (2.17), $\delta^l_{x,y}$ is defined as $\frac{\partial E}{\partial a^l_{x,y}}$. Again, by using chain rules, it can be expressed as:

$$\frac{\partial E}{\partial a^l_{x,y}} = \sum_{x'=0}^{k-1}\sum_{y'=0}^{k-1} \frac{\partial E}{\partial a^{l+1}_{x',y'}} \frac{\partial a^{l+1}_{x',y'}}{\partial a^l_{x,y}} = \sum_{x'=0}^{k-1}\sum_{y'=0}^{k-1} \delta^{l+1}_{x',y'} \frac{\partial a^{l+1}_{x',y'}}{\partial a^l_{x,y}} \tag{2.18}$$

Similar with Eq. (2.16), the term $\frac{\partial a^{l+1}_{x',y'}}{\partial a^l_{x,y}}$ can be re-written as:

$$\frac{\partial a^{l+1}_{x',y'}}{\partial a^l_{x,y}} = w^{l+1}_{x'-x,y'-y} \frac{\partial}{\partial a^l_{x,y}}(f(a^l_{x,y})) \tag{2.19}$$

Then subsituting Eq. (2.19) into Eq. (2.18):

$$\begin{aligned}\delta^l_{x,y} &= \sum_{x'=0}^{k-1}\sum_{y'=0}^{k-1} \delta^{l+1}_{x',y'} w^{l+1}_{x'-x,y'-y} \frac{\partial}{\partial a^l_{x,y}}(f(a^l_{x,y})) \\ &= \delta^{l+1}_{x,y} * rot_{180^\circ}(w^{l+1}_{x,y}) \frac{\partial}{\partial a^l_{x,y}}(f(o^l_{x,y}))\end{aligned} \tag{2.20}$$

Eq. (2.17) and Eq. (2.20) implies that the back-propagation of CNNs can also be viewed as a convolution procedure, which helps the implementation of CNNs.

Some techniques like unsupervised pre-training and dropout can also be used to further increase the performance of CNNs. Furthermore, Ossama et al. [7] proposed a limited weight sharing (LWS) method for CNNs that make them can work well with speech recognition tasks. Different with the weight sharing over the whole feature map, LWS only forces the weights of the same group of pooled units to be identical.

2.3 Applications

This section briefly reviews some applications of ANNs, and the selected applications can be divided into two parts: image processing and speech processing.

2.3.1 Image Classification Tasks

In [2], Ciresan et al. implemented DNNs on CUDA platform, and the utilization of GPU parallel computing significantly accelerate the training speed of DNNs. In experiments, DNNs are used to solve the hand-written digits classification. MNIST database is a well-known handwritten digital images database, which contains 60000 training samples and 10000 test samples (The gray-scale hand-written digits are much simpler compare with RGB natural images thus can be handled by fully-connected DNNs). However, when using the deep architecture, the training sample size of MNIST is too small to prevent over-fitting. Instead of applying unsupervised pre-training, this paper considers to expand the training sample size by using image augmentation methods. Specifically, 4 kinds of randomly augmentations (rotation, scaling, horizontal shearing and elastic deformation) are introduced, and the image augmentation methods should be executed at the beginning of each training epoch. And as a result, the DNNs can almost never see the same training sample twice during the training procedure. The test results listed in Table 2.2 reflect the excellent performance of the proposed method. In this table Architecture indicates the number of hidden layers and the size of each hidden layer.

In [38], Ciresan et al. proposed a noval multi-column CNNs to further improve the image classification performance of deep architectures. In this paper, the proposed CNN contains hundreds of feature maps in each layer, and using maximum pooling as

TABLE 2.2: Average Test Error and Best Test Error of DNNs with Different Network Configurations on MNIST Database [2]

ARCHITECTURE	AVERAGE	BEST	TRAINING TIME
1000, 500	0.49%	0.44%	23.4 *(hr)*
1500, 1000, 500	0.46%	0.40%	44.2 *(hr)*
2000, 1500, 1000, 500	0.41%	0.39%	66.7 *(hr)*
2500, 2000, 1500, 1000, 500	0.35%	0.32%	114.5 *(hr)*

the down-sampling method. The main difference with the regular CNNs is that several columns of CNNs are combined and construct the so called multi-colunm CNNs (MC-CNNs). Before training, all columns of CNNs are initialized randomly, and then each column can be trained by using the same inputs, or inputs that derived from various pre-processing methods. Finally the outputs of each columns should be averaged to get the final classification results. Because of the acceleration provided by GPUs, the training of several colunms of CNNs can be done in relatively short time. On MNIST database, MCCNNs achieve 0.23% error rate, which is even better than DNNs that mentioned before. Furthermore, this method has been used to deal with some real-world problem-s like traffic sign classification [39] and also achieves excellent performance (99.46% average recognition rate).

In ILSVRC-2012 competition, Krizhevsky et al. [40] proposed to use a deep CNN to classify the ImageNet database, which contains 1.2 million high-resolution images. The proposed CNN includes 5 convolutional layers and 2 fully-connected layers. To reduce overfitting, data augmentation and dropout are applied during the training process. On the test set of ImageNet, the proposed CNN achieves 15.3% top-5 error rate. Comparing with traditional image classification algorithms such as Fisher Vectors [41] (26.2% top-5 error), the proposed CNN has much better classification performance.

In [42], a novel CNN model called All-CNN is defined. This model proposed to use regular convolutional layers with larger stride to replace the original pooling layers for better performance. This modification introduces more learnable parameters into the network, and reduces the classification error on CIFAR-10 test set (9.08% vs 9.47%, without data augmentation).

Instead of removing pooling layers from CNNs, [43] proposed to generalize the pooling functions. Based on the traditional average pooling, denote by $f_{avg}(x)$, and maximum

pooling, denote by $f_{max}(x)$, two novel pooling methods, i.e. mixed max-average pooling and gated max-average pooling, are presented. The definition of mixed max-average pooling is:

$$f_{mix}(x) = af_{max}(x) + (1-a)f_{avg}(x) \tag{2.21}$$

where a is the weight of two pooling methods and can be learned via BP algorithm. Gated max-average pooling has more complicated form, which introduces learnable gating mask w and sigmoid activation function instead of scalar weight a in Eq. (2.21):

$$f_{gate}(x) = sigmoid(w^T x)f_{max}(x) + (1 - sigmoid(w^T x))f_{avg}(x) \tag{2.22}$$

The gating mask w can also be learned by using BP algorithm. Finally, the tree-pooling method can be defined based on gated max-average pooling. Specifically, several pooling filters are used to generate the leaf nodes of tree-pooling. Then every two leaf nodes can be combined by using Eq. (2.21). This process continues layer-by-layer until the root of the tree is generated. The experimental results show that the tree-pooling method achieves state-of-the-art performance on CIFAR-10 test set (7.62% error rate without data augmentation and 6.05% with data augmentation).

CNNs can also be used in the field of object detection. In [3], a CNN based algorithm called DetectorNet is presented to locate the bounding boxes of objects in given images. The network structure applied for DetectorNet includes five convolutional layers and two fully connected layers. Each layer uses ReLU as the activation function, and maximum pooling is used as down-sampling method. The main difference between DetectorNet and standard CNNs is that in DetectorNet a regression layer is utilized as the top layer instead of a softmax layer to generate the object binary masks, which can represent the location of objects. By resizing the image, the output binary mask may represent one or more objects. Moreover, this work proposed DNN-generated Masks to further improve the detection performance. In experiments, the DetectorNet is compared with several other object detectors on Pascal VOC 2007 database, and it shows state-of-the-art performance of various classes. The test results are shown in Table 2.3. However, one drawback of this method is its relatively high computational cost at training time because the network should be trained for each object type and mask type. Therefore, the future work should aim to train a single network to detect different classes of objects.

TABLE 2.3: Average Precision of DetectorNet and Other Methods on Pascal VOC 2007 Database [3]

OBJECT CLASSES	BIRD	BUS	CAT	COW	TV
DETECTORNET	0.194	0.532	0.272	0.348	0.470
SLIDING WINDOWS	0.068	0.294	0.101	0.131	0.119
3-LAYER DNN	0.094	0.440	0.213	0.193	0.393
DPM APPROACH	0.107	0.513	0.179	0.240	0.413

Scene labeling is a task that aims to label a given image pixel-by-pixel based on the classes of objects. This task is also an important branch in image processing. CNNs can be used to solve this problem. In [4], a modified CNN model named recurrent convolutional neural networks (RCNN) is proposed to do scene labeling and achieved outstanding performance with good efficiency. Recurrent means that its architecture consists of several instances of regular CNNs, and all instances share the same model parameters. The experiments results are provided in Table 2.4, which proves that RCNNs have both good accuracy and fast execution speed. This research illustrates that the scene labeling tasks can be solved effectively and efficiently without use of expensive graphical model or segmentation methods, and can achieve state-of-the-art performance with fast inference time.

TABLE 2.4: Performance of RCNN and Other Methods on Stanford Background Database [4]

METHODS	PIXEL ACCURACY (%)	COMPUTING TIME (S)
ALGORITHM IN [44]	76.4	10 TO 600
ALGORITHM IN [45]	81.9	> 60
REGULAR CNN	79.4	15
RCNN (2 INSTANCES)	76.2	1.1
RCNN (3 INSTANCES)	80.2	10.7

2.3.2 Speech Recognition Tasks

In the past few years context-dependent DNN-HMM (Hidden Markov Model) hybrid model achieves significant success in the field of automatical speech recognition (ASR). Different from the traditional GMM-HMM ASR system, DNNs serve as the acoustic models to generate the posterior probabilities of HMM states, and then HMM is used to do decoding and get the recognition results. In [5], a noise robust DNN-HMM system

is proposed by combining DNNs with several noise robustness methods include multi-condition training data, enhanced feature, model adaptation and dropout. Experiments on Wall Street Journal (WSJ0) database can reflect the performance of the noise robust system. Specifically, the multi-condition training set of WSJ0 contains 7137 utterances from 83 speakers, and the test set can be divided into 4 groups: clean, noisy, clean with channel distortion and noisy with channel distortion (denote by A, B, C, and D respectively). These four kinds of test sets are used to evaluate the robustness of the algorithm. Two kinds of input features for DNN-HMM and the baseline GMM-HMM are applied: MFCC and FBANK-24. Table 2.5 reflects that DNN-HMM shows better performance and robustness compare with GMM-HMM. Specifically, DNN-HMM model can work well with noisy data, and the utilize of FBANK-24 features further increase the recognition accuracy.

TABLE 2.5: Word Error Rate (%) of GMM and DNN Acoustic Model on WSJ0 Database [5]

METHOD/FEATURES	A	B	C	D	AVERAGE
GMM-HMM(MFCC)	12.5	18.3	20.5	31.9	23.0
DNN-HMM(MFCC)	5.7	10.4	10.9	22.6	15.3
DNN-HMM(FBANK-24)	5.0	9.2	9.0	20.6	13.8

In [6] and [46], a multiple DNNs (mDNNs) acoustic model is presented to replace the standard single large scale DNN to compute the states posterior probabilities. During the training phase, all tied states of context-dependent HMMs are firstly divided into some clusters, and then several DNNs are trained by using those clusters respectively in parallel. The outputs of the multiple DNNs can be combined to calculate the posterior probabilities for decoding. Table 2.6 shows that the mDNN-HMM model brings about significant acceleration (over 7 times acceleration with 4 GPUs) with negligible performance hit. Thus this model is practical for large-scale ASR tasks.

TABLE 2.6: Word Error Rate (%) of DNN and mDNN Acoustic Model on Switchboard Database [6]

# OF HIDDEN LAYERS	3	4	5	6
DNN-HMM WORD ERR	17.7	16.9	16.4	16.2
EXECUTION TIME (*hr*)	11.0	13.0	13.6	15.0
MDNN-HMM WORD ERR	17.8	17.0	16.9	16.7
EXECUTION TIME (*hr*, 3 GPUS)	3.0	3.7	4.3	5.0
SPEED-UP	3.7X	3.5X	3.2X	3.0X

TABLE 2.7: Performance of CNNs and DNNs with Different Configuration on TIMIT Database [7]

NETWORK STRUCTURE	AVERAGE PER(%)
DNN $(2000 + 2 * 1000)$	22.02
DNN $(2000 + 4 * 1000)$	21.87
CNN (LWS(M:150 P:6 S:2 F:8)$+2 * 1000$)	20.17
CNN (FWS(M:300 P:6 S:2 F:8)$+2 * 1000$)	20.31
CNN (FWS(M:150 P:4 S:2 F:8)+FWS(M:300 P:2 S:2 F:6)$+2 * 1000$)	20.23
CNN (FWS(M:150 P:4 S:2 F:8)+LWS(M:150 P:2 S:2 F:6)$+2 * 1000$)	20.36

Ossama et al. proposed a modified CNNs that can also work with ASR problems [7][47]. In this research, the input data should firstly be re-organized to make them compatible with the CNN structure. For instance, the input features can be arranged as three 2D feature maps, each of which represents the original data, first-order derivative and second-order derivative respectively. Each feature map contains several adjacent frames and the re-organized features maps can be feed into the CNN. Moreover, to make the CNN more suitable for ASR tasks, the limited weight sharing (LWS) method is proposed instead of the traditional fully weight sharing (FWS). TIMIT database is used to evaluate (using phone error rate (PER) to measure the performance) the proposed method, and the results of different configurations are presented in Table 2.7, where 'm' is the number of feature maps, 'p' is the pooling size, 's' is the shift size, and 'f' is the filter size. The experiments implies that CNNs can provide about $6 - 10\%$ relative error reduction compare with fully-connected DNNs. Moreover, the applying of LWS scheme makes the CNNs more suitable to deal with ASR tasks.

Chapter 3

Image Processing

Image Processing is an important research field in computer vision. Recently, increasing number of works begin to consider to apply deep learning algorithms to solve the image processing related problems, which may include image classification, object localization and detection, image saliency detection and segmentation, and image generation. CNNs become one of the most important tool for those variety of tasks. This chapter will review several important tasks in image processing.

3.1 Image Pre-Processing

When using CNNs to deal with image processing tasks, it may not be a good choice to directly use the original image (gray-scale, RGB or other color channels) as the input feature, since the raw features may be too complicated and noisy for the deep learning systems. Moreover, most of the commonly-used image datasets suffer from the lack of training samples. For instance, Pascal VOC 2012 [48] has only 5717 training images, and MNIST has only 60000 training samples. The insufficient training samples may lead to over-fitting when train large scale CNNs. All of the facts above imply that some pre-processing methods are required for image processing tasks. This section will review some important pre-processing methods, which include image whitening, image data augmentation and superpixel segmentation.

3.1.1 Image Whitening

Whitening transformation is a kind of linear transformation that transform the input signals to white noises. The main idea of the whitening transformation is to convert a sequence of random variables to another new sequence with an identity covariance matrix [49]. The identity covariance matrix implies that the new random variables are de-correlated and have variance of 1.

In the natural images, the adjacent pixels are always have high correlation, which may bring about a lot of redundancy in the image information. Doing image whitening can remove those correlation and help the learning algorithm to extract the independent information from the training data.

Assume that there is a training set $X \in R^{n \times m}$ (all features have zero mean value), where n is the feature dimension and m is the sample size. Then the covariance matrix of the training data is:

$$\Sigma = \frac{1}{m} X X^T \tag{3.1}$$

Then the singular value decomposition on Σ can be done as Eq. (3.2) shown:

$$[U, S, V] = svd(\Sigma) \tag{3.2}$$

where U is the eigenvector matrix of Σ, S is the eigenvalue diagonal matrix, and $V = U^T$ (because Σ is a symmetric matrix). Then it is easy to know that $\Sigma = USV$.

Based on Eq. (3.1) and Eq. (3.2), the definition of PCA whitening is:

$$X_{PCA} = S^{\frac{1}{2}} U^T X = \begin{bmatrix} \frac{1}{\sqrt{\lambda_1}} & \dots & 0 \\ 0 & \dots & 0 \\ 0 & \dots & \frac{1}{\sqrt{\lambda_n}} \end{bmatrix} X_{rot} \tag{3.3}$$

where $X_{rot} = U^T X$ denotes to rotate X to the basis of all eigenvectors. Then it is very easy to derive that $\Sigma_{PCA} = I$, where I denotes the identity matrix.

Based on the definition of PCA whitening, it is easy to define ZCA whitening:

$$X_{ZCA} = U X_{PCA} \tag{3.4}$$

which can be viewed as transform the PCA whitened data back to the original feature space. Also, it is easy to verify that $\Sigma_{ZCA} = I$.

Experimental results show that using ZCA image whitening on CIFAR-10 database can improve classification accuracy on relatively small-scale CNNs. However, for deep CNNs, ZCA whitening cannot bring obvious benefits. One potential reason is that deep networks can already gradually de-correlate features.

3.1.2 Image Data Augmentation

Even though the number of free parameters in CNNs has been significantly decreased by using local receptive fields and weight sharing, training deep CNNs still requires huge number of training data. However, when using supervised learning, the well-labeled training data are always far from enough. Therefore, data augmentation becomes an important pre-processing for image processing tasks. In practice, even ImageNet dataset (around 1.4 million training images) [24] also needs to be augmented to train very deep neural networks such as vgg nets [50] and deep residual nets [51]. The simplest way of image data augmentation is image fliping. For instance, [52] proposed a deep neural network for CIFAR-10 [53] image classification. The author argues that when randomly do horizontal flip for 50% images on the training set, the classification accuracy on test set improves from 91.30% to 92.45%. In [54], more complicate image data augmentation methods are considered to improve the classification performance of CNNs on the ImageNet database . The paper mainly focus on the so-called key augmentations, includes color casting, vignetting, stretch and cropping and lens distortion. In Chapter 5 of this dissertation, four kinds of image augmentations are also used for better classification performance on both CIFAR-10 and CIFAR-100 databases, i.e., translation, rotation, scaling and color space translation. Figure 3.1 displays some examples of the selected data augmentation.

3.1.3 Superpixel

Superpixel segmentation is another important image pre-processing method for image processing tasks. Superpixel is a image region that has perceptually uniform feature, which can be viewed as an over-segmentation of images. For the digital images, pixel-grids are the basic units, and most computer vision algorithms may use them as the input data. However, in some cases, simply using raw pixels has some drawbacks. Pixel-grid itself is not a natural way to represent images. It simply divides natural images into large number of pixels and dispatches RGB values (or other color channels

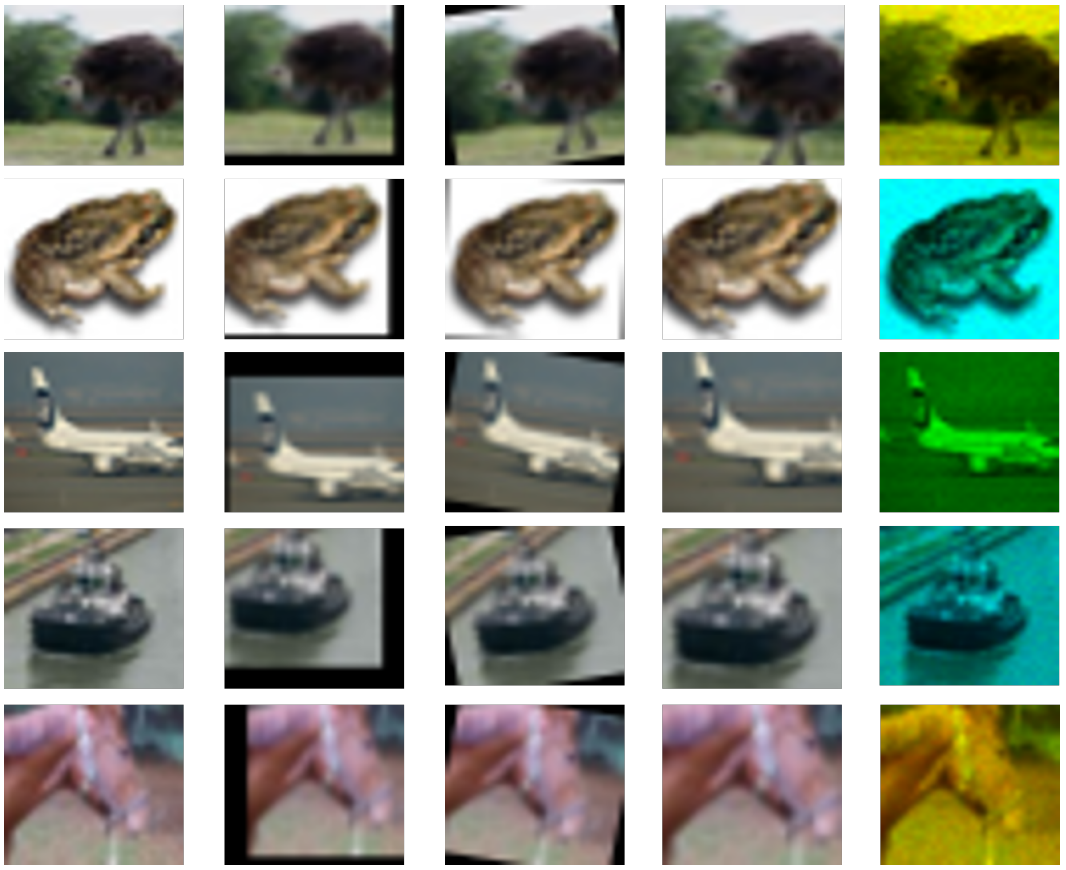

FIGURE 3.1: From Left to Right: original images, translated images, rotated images, scaled images and color space translated images.

like YUV) to each pixel based on the corresponding color information. This may lead to some disadvantages in practice. Firstly, pixel-grids cannot reflect the some high-level information of images, such as different objects or some regions with similar color/texture features. Secondly, currently most images have high resolution, which means that they may include huge number of pixel-grids. For example, one 300-by-500 image has 150,000 pixels, which may significantly increase the computational complexity. Therefore, some superpixel algorithms are proposed to group several adjacent pixels that share the similar color or texture into one superpixel to reflect some high-level feature and reduce the complexity of the images (from hundreds of thousands to around one hundred) without losing useful feature [55]. Superpixels can be applied to different kinds of applications such as video objects tracking [56], image saliency detection [57] and segmentation [58].

There are a lot of superpixel extraction algorithms proposed in previous researches. In [59], superpixel segmentation is used as an important pre-processing for the final image segmentation. In this paper, Normalized Cuts [60] algorithm combined with contour

and texture features are used to generate superpixel maps. The experiments show that most superpixels have similar size and shape, and can keep most important features and structures in the input images. Those facts can help the further segmentation.

In [61], the Entropy Rate Superpixel Segmentation method is proposed. This method introduces a novel objective function that contains two components to guide the superpixel generation. Specifically, the algorithm uses undirected graphs to represent images, and introduces a random walk process on the graphs. The first term of the objective function is defined as the entropy rate of the random walk. Based on the definition of the entropy rate term, it encourages compact and homogeneous superpixels. The second term is the balancing term, which makes all superpixels tend to have similar size. By using greedy optimization, the algorithm can generate superpixels with similar size and keep the object boundaries.

Seeds algorithm [62] is another superpixel extraction method that uses a simple hill-climbing optimization method to reduce the computational costs. Different with most of the superpixel algorithms, Seeds starts from an initial superpixel partition (for example dividing the original images by using regular grids), i.e., some seeds, and gradually modify the superpixels by changing the boundaries. To guide the boundaries changing process, a robust energy function is proposed based on the color distribution inside the superpixels (the color distribution term) and the shape of the superpixels (the boundary term). The hill-climbing optimization means that in each iteration some small local changes of the superpixels will be proposed, and if the energy function increases, the superpixel partition will be updated. Experimental results show that after several iterations good superpixel maps can be generated.

Simple Linear Iterative Clustering (SLIC) [63] is also a well-know superpixel generation method, which based on K-means clustering algorithm and can provide state-of-the-art performance on different image databases. Specifically, SLIC can be divided into three steps:

(1) Initialization: assuming that the image size is $N \times N$, and the expectation number of superpixels is k. The algorithm tends to generate homogeneous sized superpixels with the approximate size $S \times S$. Obviously, S should roughly equals to $N/\sqrt{k}$. Therefore, similar with Seeds algorithm, SLIC firstly splits the image into k $S \times S$ grids, and the k initial cluster centroids should be sampled from them. The algorithm firstly selects the center of each grid as the seed, and then move it to the point that has lowest gradient in its 3×3 neighborhood to avoid the edges in the image. For the images in the CIELAB

color space, all data pixels can be represented as a 5-dimension vector $[l, a, b, x, y]$, where x and y denote the location of the pixel. To measure the distance of two feature vectors P_i and P_j, a new distance measurement is defined:

$$D = \sqrt{{d_c}^2 + (\frac{d_s}{S})^2 m^2} \tag{3.5}$$

where

$$d_c = \sqrt{(l_j - l_i)^2 + (a_j - a_i)^2 + (b_j - b_i)^2}$$

is the color distance,

$$d_s = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$

is the spatial distance, and m is used to balance the importance of the two distances.

(2) Assignment: in this step each pixel in the image should be assigned to its nearest cluster centroid. This procedure is based on K-means algorithm, the only difference is that in SLIC, the search space of each centroid is limited in a region with the size $2S \times 2S$, which can significantly reduce the computational complexity.

(3) Update: in the update step, the cluster centroids should be updated by computing the mean vector $[l_m, a_m, b_m, x_m, y_m]$ of all pixels that belong to the cluster. Just like K-means, the assignment and update can be run for several epochs, and finally, the disjoint pixels should be reassigned to their adjacent superpixels.

The experimental results reflect that SLIC algorithm can provide high quality superpixel maps with high efficiency on variety of image databases. Therefore, in this dissertation, it is used for the saliency detection algorithms in Chapter 6 and 7 to smooth the raw saliency maps.

3.2 Image Processing with Deep Learning

There are diversity tasks of image processing, including image classification, objects localization, objects detection, saliency detection and segmentation. Currently, most of researches imply that deep learning techniques can deal with those variety of tasks effectively.

Image classification is the basic task for deep learning, since neural networks are originally designed to work with classification jobs. Based on the classification results,

CNNs can be used to do more complex tasks. In Large Scale Visual Recognition Challenge (ILSVRC), image classification is combined with object localization as one task (CLS-LOC). The requirement of CLS-LOC is that firstly generate class labels for the input images, and then using bounding boxes to localize the corresponding objects. Object detection is much harder compare with localization, which needs the algorithm firstly find one or more objects inside the input image by using bounding boxes, and then provide their class labels. Comparing with localization or detection, image saliency detection and segmentation hope to generate pixel-wised saliency or segmentation maps. Saliency detection and segmentation have tight relationship with this dissertation.

3.2.1 Image Saliency Detection

'Saliency' is originally a definition in the fields of neuroanatomy and psychology. This term denotes a stimulus (from a variety of sources, such as images or sounds) that can grab more attention from the observer. For human beings and other organisms, the ability of saliency detection is important since it can help them to find the important resources or potential threats from the surrounding environment. In visual perception, contrast is one important reason of saliency. For example, an object in an image that has different color or texture with its surrounding backgrounds may tend to be recognized as a salient object [64].

In the domain of computer vision, visual saliency detection is also a well-known research topic, since making computers learn how to find salient objects automatically is important for the development of artificial intelligence. Previous researchers have proposed variety of methods to model the procedure of human attention for image saliency detection [65]. Bottom-up saliency methods are based on the assumption that saliency regions in one image may differ from the background. These methods tend to use low-level features such as color distribution, local/global contrast and texture to generate saliency maps. In [10], a bottom-up saliency method is proposed, which considers region contrast and spatial coherence at the same time to achieve good performance. To measure the saliency value of one pixel I_k in a given image I , one simple way is to calculate its color contrast to all other pixels in the image:

$$S(I_k) = \sum_{\forall I_i \in I} D(I_k, I_i) \tag{3.6}$$

where $D(I_k, I_i)$ is the color distance measurement. Eq. (3.6) has some subtle assumptions: (1) The same color in the image has the same saliency value, which means that all pixels with the same color should either be foreground (high saliency value) or background (low saliency value). (2) The salient objects should only occupy small part of the image. In practice, however, it is easy to know that this simple method cannot deal with the images that have complicate backgrounds or very large salient objects [66]. Moreover, the efficiency of this method is not good enough, especially for the large scale images. To solve those problems, [10] proposed the region contrast method (RC). In RC, the input images should firstly be segmented into superpixels by using a graph-based segmentation method [67]. Then, the superpixels can be viewed as the basic unit to calculate the saliency values:

$$S(r_k) = \sum_{r_i \neq r_k} exp(-D_s(r_k, r_i)/\sigma_s^2) w(r_i) D_r(r_k, r_i) \tag{3.7}$$

where $D_s(r_k, r_i)$ is the spatial distance between r_k and r_i, and σ_s is used to balance the weights of the spatial distance. $w(r_i)$ is the number of pixels in the superpixel region r_i, which implies that large superpixels have high priority to be selected as foreground. $D_r(r_k, r_i)$ is the distance measurement of two superpixel regions, which is defined in Eq. (3.8):

$$D_r(r_k, r_i) = \sum_{a=1}^{n_k} \sum_{b=1}^{n_i} f(c_a^k) f(c_b^i) D(c_a^k, c_b^i) \tag{3.8}$$

where n_k is the number of colors in the superpixel r_k, c_a^k is the ath color in r_k, and $f(\cdot)$ denotes the probability.

Experiment results show that RC can deal with more natural images, but its performance is still not good for complicated tasks (see Figure 3.2).

To address the problems of bottom-up methods, some deep learning based saliency methods are proposed, because DNNs and CNNs may have ability to extract multi-level features from the images and those different level of features can be used for object saliency detection. [12] proposed a CNN-based multi-contexts saliency method called multi-context deep saliency. The method considers global context and local context simultaneously to generate high quality saliency maps. The multi-context deep saliency firstly performs superpixel segmentation on all input images by using SLIC algorithm

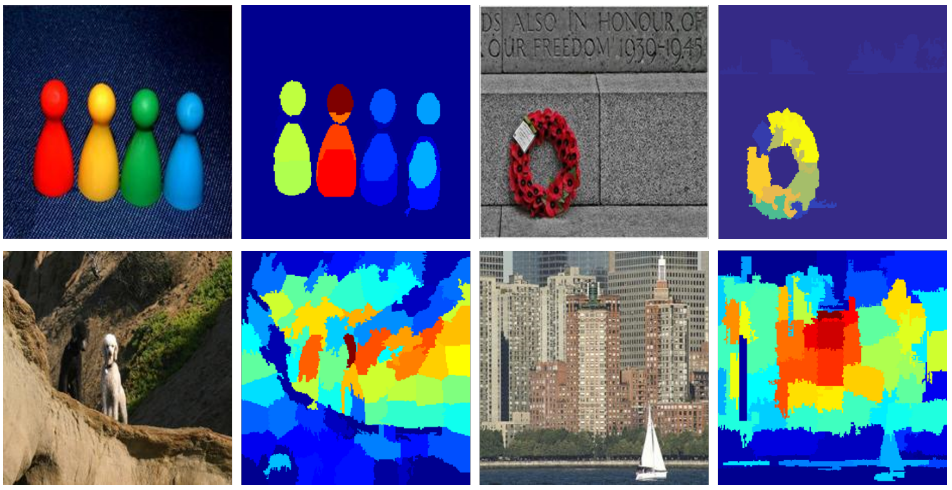

FIGURE 3.2: RC works well for simple images (row 1), but can hardly deal with complex images (row 2). In the saliency maps, red means the region has high saliency value, and blue means low saliency value

[63], then pre-trains two CNNs to model global context and local context respectively. All superpixels in one image should be used as the center to generate the inputs of both global-context CNN and local-context CNN. For the global-context inputs, the square context window should include the whole image, and the regions exceeding the boundaries should be padded by using the average pixel value the the training set. The padded images should be resized to 227-by-227 as the network inputs. It is easy to learn that the global-context CNN can catch the feature of the whole image. By contrast, the local-context CNN is designed to get the local feature of the images, and its context window size is only one ninth of the global context CNN. The output layers of the two CNNs have two neurons, which are used to indicate the centered superpixel in the input belongs to saliency region or background. Based on the outputs of the two CNNs, the saliency value of one superpixel can be defined as:

$$score(x_{gc}, x_{lc}) = P(y = 1|x_{gc}, x_{lc}; \theta_1) \tag{3.9}$$

where x_{gc} and x_{lc} denote the output of the global-context CNN and local-context CNN respectively, $y = 1$ means that the centered superpixel belongs to saliency region, and θ_1 is the set of hyperparameters of the models. Experiments on several databases show that the usage of CNNs results in better performance compare with bottom-up methods (see Figure 3.3).

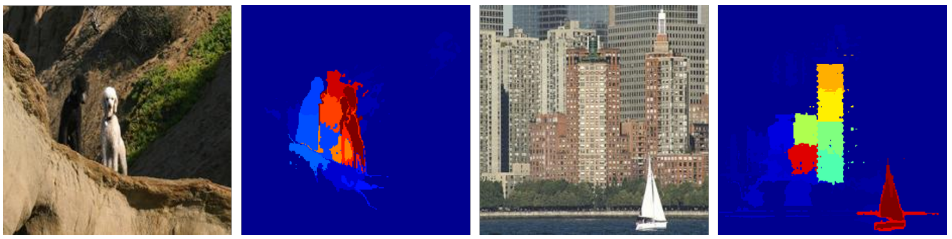

FIGURE 3.3: Multi-context deep saliency can deal with complicated images.

3.2.2 Image Semantic Segmentation

Comparing with superpixel segmentation and object saliency detection, image semantic segmentation is a more challenging task. Superpixel can be viewed as a kind of over-segmentation, which can divide one image into hundreds of small superpixel regions. On the contrary, semantic segmentation aims to split an input image into few parts based on the semantic meanings, i.e. one part for one object in the image, and the number of the parts is always much less than superpixel maps. Different with saliency detection, semantic segmentation not only focus on the salient objects, but tries to find all potential objects or regions in the image and gives class labels for them. However, image saliency detection can be viewed as an important pre-step for semantic segmentation, and some algorithms can generate segmentation based on saliency maps. Semantic segmentation algorithms are significant for computer systems to understand the natural images.

[68] proposed a semantic segmentation algorithm that utilizes fully convolutional networks (FCNs), which can make pixel-wised prediction on the images for segmentation tasks. The meaning of fully convolutional networks is that the fully connected layers in CNNs have been transformed into regular convolutional layers. This transformation enables the CNN to generate heatmaps for segmentation instead of the classification vectors, and at the same time predicts class labels of the potential objects. However, in practice the scales of the output feature maps of FCNs are still too small, thus when up-sampling them to the original image size, the resulting segmentation maps are always very coarse. To fix this problem, skips are added into the FCNs, which can use lower layers along with the final output layer to refine the segmentation results. The combination of higher and lower layers enables the FCNs to be able to view both local feature and global feature simultaneously. The experimental results show that the FCNs based method can generate high quality semantic segmentation results.

Regions with CNN features (R-CNN)[69] is another well-known semantic segmentation algorithm, which combines region proposals with CNNs for better accuracy. The

R-CNN system firstly do object detection on the input images, and after that, it can generate semantic segmentation based on the detection results. R-CNN includes three modules. The first one is used to generate category-independent region proposals to serve as object candidates, and the selective search [70] algorithm is used in this module. The second module is a CNN to extract a 4096-dimensional feature vector for each region proposal. In this step each region should be reshaped to 227-by-227 and then input into the CNN. Finally, a group of linear SVMs are trained for every class, and they will be used to score the extracted features from the second module. After that, the regions that overlap with some higher scored regions will be rejected. The object detection procedure generates bounding boxes of objects with class labels and scores. Then based on the framework of second-ordering pooling segmentation [71], the features extracted in the second module can be used to generate segmentation maps. Based on the basic model of R-CNN, Fast R-CNN [72] and Faster R-CNN [73] further improve the efficiency of the R-CNN and make it more practical for real-world tasks.

Chapter 4

Annealed Gradient Descent for Deep Learning

This chapter mainly focuses on training algorithms of DNNs. The highly non-convex nature of the objective functions of DNNs makes them very hard to be optimized. Experiments reflect that the efficiency of traditional gradients based training methods is far from enough. In this chapter, we consider to find a novel training algorithm to accelerate the learning speed of DNNs.

4.1 Introduction

The past few decades have witnessed the success of gradient descent algorithms in machine learning. By only calculating the local gradient information of the loss function, gradient descent (GD) algorithms may provide reasonably good optimization results for different types of problems. Among many, stochastic gradient descent (SGD) is a very popular method for modern learning systems, which only use one or a few randomly-selected training samples to update the model parameters in each iteration [74]. Comparing with the batch GD algorithms, SGD needs far less computing resources especially when it deals with huge tasks involving big data. In [75], it has been proved that SGD can process asymptotically more training samples than batch GD algorithms given the same amount of computing resources. [76] proposed a new SGD framework that can achieve a linear convergence rate for strongly-convex optimization problems. In [77], the performance of SGD was analyzed on non-smooth convex objective functions, and a bound of the expected optimization errors is provided.

On the other hand, SGD also has its own drawbacks. The random noise introduced by data sampling leads to noisy gradient estimations, which may slow down convergence speed and hurt the performance [78]. Moreover, because of its sequential nature, SGD is hard to be parallelized.

More recently, several methods have been proposed to parallelize SGD to accelerate its training speed for big-data applications. Initially, some researchers have proposed to implement SGD method across multiple computing nodes that are synchronized for updating model parameters. Unfortunately, it has been found that the delay of the required server synchronization is always much longer than the time to calculate the gradient [79]. Therefore, several other methods have been proposed to parallelize SGD without frequent synchronization among computing node. For example, [80] has presented a parallelized SGD algorithm, which dispatches all training samples into several computing nodes to update the parameters independently, and the final models will be combined by averaging all separate models at the end of each training epoch. Moreover, [81] has proved that the performance of this parallelized SGD algorithm depends on the number of SGD runs, and it has been successfully applied to train large-scale support vector machines. However, the convergence of this simple parallelized SGD method requires that the training process is a convex optimization procedure. Moreover, [82] has shown that the delay introduced by unsynchronized model updates can be asymptotically neglected when optimizing smooth and convex problems. Similarly, [83] has proposed a parallelized SGD framework called 'HOGWILD!', which can remove most of the memory locking and synchronization. However, it has found that this method works well only for sparse models. In summary, the above mentioned parallelized SGD methods heavily rely on the assumptions that the learning problems are convex and sparse, and these methods may suffer loss of performance when dealing with more general non-convex problems, such as deep learning problems.

Even though today's development of computing hardware makes it possible to train large DNNs directly, it is still very slow to train state-of-the-art DNNs for any real-world applications since the major training of DNNs still largely depends on mini-batch SGD algorithm. Therefore, it is much needed in deep learning to develop any new optimization method that is faster to solve large-scale deep learning problems. One idea is 'start small' [84], in which the network training begins from simple training data with small working memory, and the data complexity and memory usage should be gradually increased. This process simulates the learning procedure of human beings, especially some complex domains like language. Krueger et al. implemented 'shaping'

learning in neural network training [85]. During the training, the task should be splited into several sub-components to build a suitable training sequence. Experiments show that shaping method can significantly boosting the learning speed. In [86], Bengio et al. proposed a training strategy for deep learning called curriculum learning. The basic idea of which is to make the learning process start from small tasks that are easily to be solved, and gradually increase the complexity of the tasks. Experiments results imply that when using a suitable curriculum, this training strategy may provide similar performance as unsupervised pre-training and help the algorithm to find a good local minimum. Curriculum learning method can serve as an important basis for the proposed AGD algorithm.

In this chapter, the AGD algorithm is proposed for the training of DNNs. Instead of directly optimizing the original non-convex objective function, the basic idea of AGD is to optimize a low resolution approximation function that may be smoother and easier to optimize. Furthermore, the approximation resolution is gradually improved according to a pre-defined annealing schedule over the optimization course. In this work, the AGD algorithm approximates a non-convex objective function based on some pre-trained codebooks, where the approximation precision can be easily controlled by choosing different number of codewords in codebooks. Comparing with [86], the main contribution of AGD is that it provides a suitable way for approximation (by introducing pre-trained codebooks), and more importantly, the proposed research provides a bound for the difference between the parameters derived by AGD and the regular GD algorithm. In this chapter, AGD has been applied to train DNNs for various tasks to verify its efficiency and effectiveness. Experiments have shown that the AGD algorithm yields about 40% speed-up in total training time of DNNs, and also leads to similar recognition performance as the regular mini-batch SGD.

4.2 Gradient Descent Algorithm

4.2.1 Empirical Risk Function

In machine learning, we normally use a *loss function*, $Q(x, y, \theta)$, to measure the 'cost' of a given input x (y is the corresponding ground truth label of x) and the underlying model parameters are denoted as θ, and the expected value of the loss function is the

so-called *expected risk function*, $R(\theta)$:

$$R(\theta) = \mathbf{E}[Q(x, y, \theta)] \triangleq \int Q(x, y, \theta) dP(x, y) \tag{4.1}$$

where $P(x, y)$ denotes the ground truth distribution over all possible events.

The fundamental goal of many machine learning problems is to minimize the above expected risk function. In practice, however, it is extremely hard to do so because $P(x, y)$ is always unknown. Therefore, we normally use a finite training set that includes N independent pairs of sample $O_N = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, which are presumably randomly sampled from the above unknown distribution. Based on the training set, we may derive the so-called *empirical risk function*, $R_N(\theta)$, to approximate the expected risk function in Eq. (4.1):

$$R(\theta) \approx R_N(\theta) = \frac{1}{N} \sum_{n=1}^{N} Q(x_n, y_n, \theta) \tag{4.2}$$

If the training set is sufficiently large, under some minor conditions, minimizing the empirical risk function in Eq. (4.2) may also lead to minimizing the expected risk function in Eq. (4.1) [87]. For notational clarity, without confusion, we drop label y_n from the loss function for the rest of this dissertation.

4.2.2 Gradient Descent Algorithm

To minimize the empirical risk function, we can use gradient descent algorithms, which update θ along the direction of the negative gradient based on a pre-defined learning rate λ. Generally speaking, there are two different types of gradient descent algorithms: batch gradient descent (batch GD) and stochastic gradient descent (SGD).

In each iteration, batch GD should consider all of the training samples to calculate the average gradient and then update the parameters:

$$\hat{\theta}_{t+1} = \hat{\theta}_t - \lambda_t \cdot \nabla_\theta R_N(\hat{\theta}_t) = \hat{\theta}_t - \lambda_t \cdot \frac{1}{N} \sum_{n=1}^{N} \frac{\partial Q(x_n, \hat{\theta}_t)}{\partial \theta} \tag{4.3}$$

where λ_t is the learning rate at iteration t. By contrast, SGD only takes one training sample (which is randomly sampled from the training set) into account in each iteration:

$$\bar{\theta}_{t+1} = \bar{\theta}_t - \lambda_t \cdot \frac{\partial Q(x_n, \bar{\theta}_t)}{\partial \theta}. \tag{4.4}$$

If we set a suitable learning rate, both batch GD and SGD can finally converge to a local minimum θ^* of the empirical risk function [88]. In practice, to reduce variance of the estimated gradients in SGD, a variant SGD, called mini-batch SGD, is normally used, where a randomly selected mini-batch of data samples is used to estimate gradient for each model update, as opposed to only one sample in SGD.

Generally speaking, batch GD works well for convex optimization while SGD may be used to solve non-convex optimization problems due to the random noises in its gradient estimation. Meanwhile, SGD needs far less computing resources comparing with batch algorithm, but its convergence speed is very slow because of the sampling noise, and it is very hard to be parallelized. Therefore, when dealing with some large scale tasks, SGD may run very slowly.

4.2.3 Reduce the Gradient Variance

Besides mini-batch SGD, there are also many other methods that can decrease the gradient variance, which will be discussed in this part.

4.2.3.1 Average Stochastic Gradient Descent

According to [89], SGD can reach an almost optimal convergence rate without the requirement of large amount of priori knowledge by using the proposed average method. And in the average stochastic gradient descent (ASGD) algorithm, the standard SGD should be performed first, and then update the average parameters $\bar{\theta}_t$ recursively:

$$\begin{aligned} \theta_{t+1} &= \theta_t - \lambda_t \cdot \frac{\partial Q(\mathbf{x}_n, \theta_t)}{\partial \theta} \\ \bar{\theta}_{t+1} &= \frac{t}{t+1}\bar{\theta}_t + \frac{1}{t+1}\theta_{t+1} \end{aligned} \tag{4.5}$$

Bottou et al. claimed that the $\bar{\theta}_t$ will have an optimal asymptotic convergence speed when we select a suitable sequence of λ_t [90].

4.2.3.2 Stochastic Variance Reduced Gradient

In [91], Johnson et al. presented a method called stochastic variance reduced gradient (SVRG), which can reduce the variance during the stochastic learning.

In the standard SGD learning procedure, owing to the variance introduced by each single training sample, the learning rate λ should not too large, and it should finally decay to zero to guarantee the algorithm can converge to a local minimum. This will slow down the convergence speed. To fix this problem, SVRG introduces $\tilde{\theta}$: $\tilde{\theta}_t = \theta_t$ or $\tilde{\theta}_t = \theta_i$ for randomly chosen $i \in \{0, 1, ..., t-1\}$. Then a average gradient can be maintained as below:

$$\tilde{\mu} = \nabla_\theta R_N(\tilde{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \frac{\partial Q(\mathbf{x}_n, \tilde{\theta}_t)}{\partial \theta} \tag{4.6}$$

Now the updating rule of SVRG can be defined as below:

$$\theta_{t+1} = \theta_t - \lambda_t \left(\frac{\partial Q(\mathbf{x}_n, \theta_t)}{\partial \theta} - \frac{\partial Q(\mathbf{x}_i, \tilde{\theta}_t)}{\partial \theta} + \tilde{\mu} \right). \tag{4.7}$$

where $\mathbf{x}_i$ is randomly selected from the training set O_N.

The SVRG algorithm obviously reduces the variance of SGD and thus it can use a relatively larger learning rate to increase the learning speed. The experiments on both convex (logistic regression) and non-convex (neural network) tasks show that SVRG has faster learning speed. However, SVRG needs to store several copies of model parameters in the memory and it becomes a trouble when large-scale models like DNNs are used. Therefore, for these kinds of large models, using mini-batch SGD maybe a better choice.

4.2.4 Parallelized Stochastic Gradient Descent

Even though the sequential nature of SGD makes it seems very hard to be parallelized, previous researchers still hope to design parallel SGD framework to speed up SGD. At beginning, SGD has been implemented across multiple computing nodes that are synchronized for updating model parameters. Unfortunately, this implementation requires synchronization between the computing nodes and the delay is always much larger than the time to calculate the gradient [79]. Therefore, several other parallel SGD methods have been proposed to avoid frequent synchronization among computing node.

4.2.4.1 A Simple Parallel Model

In [92], a very simple parallelized SGD framework has been proposed to solve convex optimization problems, and the basic process of this method is shown in **Algorithm 1**.

Algorithm 1 Simple Parallel SGD

Input: training set $O_N = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$, Learning rate λ, Number of Machines k
Define $T = \lfloor N/k \rfloor$
Randomly partition O_N, and dispatch T examples to each machine
for all $i = 1$ **to** k **parallel do**
  Randomly shuffle the data on machine i.
  Initialize $\theta_{i,0} = 0$.
  for all $t \in \{1, ..., T\}$ **do**
    Get the tth example on the ith machine (the current machine) $x_{i,t}$
    $\theta_{i,t} = \theta_{i,t-1} - \lambda_{i,t} \cdot \frac{\partial Q(\mathbf{x}_{i,t}\theta_{i,t})}{\partial \theta}$
  end for
end for
Average from all computing nodes $\theta = \frac{1}{k}\sum_{i=1}^{k} \theta_{i,t}$ and **return** θ.

This parallelized SGD algorithm removes all communication between computing nodes until the end thus extraordinarily reduce the synchronization consuming. Moreover, [81] has proved that the performance of this algorithm depends on the number of SGD runs, and it has been successfully used to train large-scale support vector machine. However, it is obvious that this method cannot be applied to solve the non-convex problems such as neural networks because the algorithm cannot ensure that the model in every computing node can finally converge to the same local minimum.

4.2.4.2 Asynchronous Stochastic Gradient Descent

To deal with the low efficiency of synchronous SGD, the idea of asynchronous SGD was proposed, which is another way to implement the parallelized SGD framework. "Asynchronous" means that each computing node needs not to synchronize the model with each other. Instead, the model replicas will be trained independently and the models communicate updates through a centralized parameter server [93]. In [93], an asynchronous SGD method called Downpour SGD is proposed. The basic idea of Downpour SGD is to divide the whole training set into several subsets and process training on each subset. All models are trained independently by using SGD, and each model replica should communicate with the central parameter server to upload the gradient and download the new model. To reduce the communication costs, two hyper-parameters n_{fetch}

and n_{push} can be defined, and each model replica can only get new model parameters from the server every n_{fetch} steps and submit the gradient every n_{push} steps.

One obvious advantage of Downpour SGD is that it is more robust to machine failures compare with synchronous SGD. If one machine fails, the whole training process of synchronous SGD is delayed, while for asynchronous SGD delays will only happen on the failed machine. The experiments show that Downpour SGD may accelerate the training speed of non-convex tasks like deep neural networks, but its speed up still not perfect: 1000 machines can only provide about 4 times speed up. However, this method lays an important basis for related works, such as two GPU asynchronous SGD methods by Paine et al. [94] and Zhang et al. [95] respectively.

HOGWILD![83] algorithm is also follow the basic rule of asynchronous SGD. This algorithm allows all computing nodes equal access to shared memory and can update the model parameters at the same time. This scheme may lead to severe problem because all computing nodes are possible to overwrite others' progress. However, Niu et al. argue that when the data access is sparse, which means that each SGD process only update a small part of the parameters, the memory overwrite will become rare and the performance loss can be ignored.

Assume the model parameter θ is n dimensional. To implement the HOGWILD! algorithm, the empirical risk function in Eq. 4.2 can be re-written as a summation of several sub-functions:

$$R_N(\theta) = \sum_{e \in E} R_{Ne}(\theta_e) \tag{4.8}$$

Where e is a small subset of $E = \{1, ..., n\}$ and θ_e is a vector that only the coordinates indexed by e are not zero (equals to their corresponding elements in θ). Obviously, each individual R_{Ne} is sparse and it may lead to a sparse data access during training. The updating procedure of each computing node is shown in **Algorithm 2**:

Algorithm 2 HOGWILD! Algorithm for Individual Computing Node

Loop
Sample e uniformly at random from E
Read current model parameters θ_e and evaluate the gradient $G_e(\theta_e)$
for $v \in e$ **do do**
  $\theta_v = \theta_v - \lambda G_{ev}(\theta_e)$
end for
End Loop

This algorithm allows more computing nodes to write to θ at the same time. Fortunately, ascribe to the sparsity of $R_{Ne}(\theta_e)$ and its gradient, most of the overwrite problem can be avoided, and the experiments on some problems such as sparse SVM show that HOGWILD! can work well with convex optimization tasks and may provide good acceleration rate. However, this method cannot provide good performance when deals with non-convex problems.

The purposes of AGD algorithm include both reduce gradient variance and speed up the learning process. The rest of this chapter will present the AGD algorithm and the experiments.

4.3 The Mosaic Risk Function

Some previous researches have considered about the problems of critical points (include local optima and saddle points) in non-convex optimization. According to [96], bad quality local minima may hinder the optimization process especially in small-scale neural networks. [97] argues that for the practical high dimensional problems, saddle points become the most difficult problem instead of local optima. In practice, any local search algorithms may be easily trapped into a nearby shallow local optimum point or saddle point, which makes it hard for optimization to proceed further. Computing the second order derivative can deal with saddle points. However, this method may cost too large memory space, thus can hardly work in practice. SGD relies on the sampling noise to alleviate this problem. Another way to tackle this problem is to optimize a smoother approximation of the original rugged non-convex function. In this work, we propose to approximate the original objective function based on a group of relatively small codebooks, which are generated by clustering the whole training set. In this way, a low resolution approximation of the objective function can be derived, which can obviously reduce the number of critical points, and is much simpler and easier to be optimized by using simple local search algorithms.

Assuming that we use a discrete codebook, denoted as $C = \{c_1, c_2, ..., c_K\}$, where $K \ll N$, to approximate the original training set. For one training sample x_n, the nearest codeword in C is selected as its approximation:

$$c^{(n)} = \arg\min_{c_k \in C} \parallel x_n - c_k \parallel \tag{4.9}$$

and the quantization error ϵ_n is $\parallel x_n - c^{(n)} \parallel$.

Next, we may derive a low resolution risk function $\tilde{R}_\epsilon$, called *mosaic risk function*, to approximate the empirical risk function $R_N(\theta)$ (where $\epsilon \equiv \max_n \epsilon_n$):

$$\tilde{R}_\epsilon(\theta) = \frac{1}{N}\sum_{n=1}^{N} Q(c^{(n)}, \theta) = \sum_{k=1}^{K} \frac{\omega_k}{N} \cdot Q(c_k, \theta) \tag{4.10}$$

where ω_k is the number of training samples in the whole training set that are approximated by the codeword c_k, i.e., $\omega_k = \sum_{n=1}^{N} \delta(c^{(n)} - c_k)$.

Assuming that the loss function $Q(x, \theta)$ is twice Lipschitz-continuous with respect to the input sample x and model parameters θ, we have:

$$\begin{aligned}
&\parallel Q(x_i, \theta) - Q(x_j, \theta) \parallel < L_0 \parallel x_i - x_j \parallel \text{ and} \\
&\parallel Q'(x_i, \theta) - Q'(x_j, \theta) \parallel < L_1 \parallel x_i - x_j \parallel \text{ and} \\
&\parallel Q(x, \theta_i) - Q(x, \theta_j) \parallel < L_0' \cdot \parallel \theta_i - \theta_j \parallel \text{ and} \\
&\parallel Q'(x, \theta_i) - Q'(x, \theta_j) \parallel < L_1' \cdot \parallel \theta_i - \theta_j \parallel
\end{aligned} \tag{4.11}$$

Where L_0, L_1, L_0' and L_1' are constants. Then it is easy to prove that for any θ, the mosaic risk function can provide a bounded approximation for the empirical risk function:

$$\parallel R_N(\theta) - \tilde{R}_\epsilon(\theta) \parallel < \epsilon \cdot L_0 \quad (\forall \theta) \tag{4.12}$$

When we deal with a non-convex loss function, the mosaic risk function will show a very important benefit due to its low resolution: because a smaller size codebook is used to approximate the training set, and one codeword may represent a large number of different training samples, the mosaic risk function is much simpler and may get rid of a lot of bad local minima comparing with the original empirical risk function ([86] can also support this argument). Therefore, if we use gradient descent method to optimize the mosaic risk function, called *mosaic gradient descent (MGD)*, we can find one of its local minimum much easier and much faster, and this local minimum on mosaic risk function is a good initialization for further learning. If batch GD is used to optimize the mosaic risk function, the training phase may also be speed up due to smaller number of codewords.

When using MGD to minimize the mosaic risk function, we can get the following parameter updating sequence:

$$\tilde{\theta}_{t+1} = \tilde{\theta}_t - \lambda_t \cdot \nabla_\theta \tilde{R}_\epsilon = \tilde{\theta}_t - \lambda_t \sum_{k=1}^{K} \frac{\omega_k}{N} \cdot \frac{\partial Q(c_k, \tilde{\theta}_t)}{\partial \theta} \tag{4.13}$$

Obviously, MGD generates a different sequence of the model parameters $\tilde{\theta}_t$ from SGD.

Moreover, we may extend the above MGD to a stochastic version using only a random mini-batch of data for the model updating in Eq. (4.13) rather than the whole training set. All data in the selected mini-batch are approximated by codewords as in Eq. (4.9). This is called mini-batch MGD. Of course, it may be better to use much larger batch size in mini-batch MGD than that of mini-batch SGD to explore the overall structure of the mosaic function and reduce the gradient variance.

The following part will show that under some minor conditions, minimizing the mosaic risk function leads the learning process convergent into a bounded neighbourhood of a local optimum of the empirical risk function. Moreover, we also show that MGD may be able to provide faster convergence rate than GD and SGD under certain conditions.

4.3.1 Convergence Analysis

If the learning rates satisfy some minor conditions, the batch GD algorithm in Eq. (4.3) is guaranteed to converge to a local optimum of the empirical risk function. In the following, we firstly compare the MGD update sequence in Eq. (4.13) with the GD update in Eq. (4.3). Obviously, we have the following lemma:

Lemma 4.1 (MGD vs. GD)**.** *Assume that the two update sequences in Eq.* (4.3) *and Eq.* (4.13) *start from the same initial parameters θ_0, and use the same sequence of learning rates λ_t, then we have:*

$$\| \tilde{\theta}_t - \hat{\theta}_t \| < \epsilon \cdot L_1 \cdot \sum_{\tau=1}^{t-1} \lambda_\tau \tag{4.14}$$

Proof: *(1) At $t = 1$, assume that GD and MGD starts from the same initialization θ_0 and share the same sequence of learning rate. Then it is easy to show:*

$$\begin{aligned}
&\| \tilde{\theta}_1 - \hat{\theta}_1 \| = \frac{\lambda_0}{N} \| \sum_{n=1}^{N} \frac{\partial Q(x_n, \theta_0)}{\partial \theta_0} - \sum_{n=1}^{N} \frac{\partial Q(c^{(n)}, \theta_0)}{\partial \theta_0} \| \\
&\leq \frac{\lambda_0}{N} \cdot \sum_{n=1}^{N} \| \frac{\partial Q(x_n, \theta_0)}{\partial \theta_0} - \frac{\partial Q(c^{(n)}, \theta_0)}{\partial \theta_0} \| \\
&\leq \frac{\lambda_0}{N} \cdot L_1' \cdot \sum_{n=1}^{N} \| x_n - c^{(n)} \| \\
&\leq \epsilon \cdot L_1' \cdot \lambda_0
\end{aligned} \tag{4.15}$$

(2) Assume that the Lemma 4.1 holds when $t = k$, i.e.:

$$\| \tilde{\theta}_k - \hat{\theta}_k \| < \epsilon \cdot L_1' \cdot \sum_{\tau=1}^{k-1} \lambda_\tau \tag{4.16}$$

then when $t = k + 1$, consider the conditions in Eq.(4.11), we have:

$$\begin{aligned}
&\| \tilde{\theta}_{k+1} - \hat{\theta}_{k+1} \| \\
&= \| (\tilde{\theta}_k - \hat{\theta}_k) + \frac{\lambda_k}{N} \sum_{n=1}^{N} (\frac{\partial Q(x_n, \hat{\theta}_k)}{\partial \theta} - \frac{\partial Q(c^{(n)}, \tilde{\theta}_k)}{\partial \theta}) \| \\
&\leq \| \tilde{\theta}_k - \hat{\theta}_k \| + \frac{\lambda_k}{N} \sum_{n=1}^{N} \| \frac{\partial Q(x_n, \hat{\theta}_k)}{\partial \theta} - \frac{\partial Q(c^{(n)}, \tilde{\theta}_k)}{\partial \theta} \| \\
&\leq \epsilon \cdot L_1' \cdot \sum_{\tau=1}^{k-1} \lambda_\tau + \lambda_k \cdot L_1' \cdot \| x_n - c^{(n)} \| \\
&= \epsilon \cdot L_1' \cdot \sum_{\tau=1}^{k} \lambda_\tau
\end{aligned} \tag{4.17}$$

thus the Lemma 4.1 will also hold when $t = k + 1$. ■

Lemma 4.1 means that if we run both MGD and batch GD algorithm for t iterations, the difference between two resultant model parameters is bounded and it is proportional to the maximum quantization error, ϵ, in the mosaic function.

Based on Lemma 4.1, the following theorem can be derived:

Theorem 4.2 (MGD vs. GD)**.** *When we use the empirical risk function Eq.* (4.2) *to measure the two parameters $\tilde{\theta}_t$ in Eq.* (4.13) *and $\hat{\theta}_t$ in Eq.*(4.3)*, the difference is also*

bounded as:

$$\| R_N(\tilde{\theta}_t) - R_N(\hat{\theta}_t) \| \leq \epsilon \cdot L_0' \cdot L_1 \cdot \sum_{\tau=1}^{t} \lambda_\tau \tag{4.18}$$

Proof: *Based on the assumptions in Eq.*(4.11) *we have:*

$$\begin{aligned}
&\| R_N(\tilde{\theta}_t) - R_N(\hat{\theta}_t) \| = \\
&\frac{1}{N} \cdot \| \sum_{n=1}^{N} (Q(x_n, \tilde{\theta}_t) - Q(x_n, \hat{\theta}_t)) \| \\
&\leq \frac{1}{N} \cdot \sum_{n=1}^{N} \| Q(x_n, \tilde{\theta}_t) - Q(x_n, \hat{\theta}_t) \| \\
&\leq L_0' \cdot \| \tilde{\theta}_t - \hat{\theta}_t \| \\
&\leq L_0' \cdot \epsilon \cdot L_1 \cdot \sum_{\tau=1}^{t} \lambda_\tau. \quad \blacksquare
\end{aligned} \tag{4.19}$$

In Lemma 4.1 and Theorem 4.2, the bounds are related the summation of the learning rates. However, in many deep learning practices such as DNNs/CNNs training, a sequence of quickly-decayed learning rates should be applied to guarantee the convergence, and in these situations the summation of the learning rates also have a bound. Therefore, Theorem 4.2 shows that model parameters $\tilde{\theta}_t$ derived by MGD provide a good estimation of $\hat{\theta}_t$ provided by batch GD algorithm when measured by the empirical risk function. The difference is bounded by a quantity proportional to the quantization error in the mosaic function. As a result, when the quantization error is sufficiently small, MGD may converge into a bounded neighborhood of a local optimal point of the original empirical risk function.

4.3.2 Faster Learning

In this part the learning speed of MGD will be studied. If we want to optimize the empirical risk function $R_N(\theta)$ up to a given precision ρ, i.e., $\| \theta - \theta^* \| < \rho$, by using batch GD in Eq. (4.3), it will take $O(\log \frac{1}{\rho})$ iterations, and the complexity of each iteration is $O(N)$. Thus the overall complexity of the batch algorithm is $O(N \log(\frac{1}{\rho}))$ [98].

Alternatively, we can run the MGD algorithm on Eq. (4.13) for t iterations, and based on Lemma 4.1 we have:

Lemma 4.3. *If we run the MGD algorithm in Eq.* (4.13) *for* t *iterations, the model parameters can reach the precision as:*

$$\| \tilde{\theta}_t - \theta^* \| < \rho + \epsilon \cdot L_1 \cdot \sum_{\tau=1}^{t} \lambda_\tau \tag{4.20}$$

and the overall computational complexity of MGD is $O(K \cdot t)$.

Based on Lemma 4.3, Theorem 4.4 can be derived as below:

Theorem 4.4 (MGD vs. GD)**.** *Assuming that there exists a codebook* C *containing* K *codewords, which can approximate the whole training set well enough, and* K *is sufficiently small, i.e.*

$$\epsilon \ll \frac{\rho}{L_1 \cdot \sum_{\tau=1}^{t} \lambda_\tau} \quad \textit{and} \quad K < \frac{N \cdot O(\log(\frac{1}{\rho}))}{t} \tag{4.21}$$

then to achieve the same optimization precision, optimizing the mosaic risk function using MGD requires less computing resources and yield faster convergence speed than batch GD in Eq. (4.3)*.*

Similar to Theorem 4.4, we also have Theorem 4.5 that compares the resources requirement between MGD and SGD:

Theorem 4.5 (MGD vs. SGD)**.** *In SGD, we need to run* $O(\frac{1}{\rho})$ *iterations to achieve the optimization precision* ρ *[74]. Similar to Theorem 4.4, if we find a codebook which satisfies the quantization error requirement and remains sufficiently small as follows:*

$$\epsilon \ll \frac{\rho}{L_1 \cdot \sum_{\tau=1}^{t} \lambda_\tau} \quad \textit{and} \quad K < \frac{1}{t} \cdot O(\frac{1}{\rho}) \tag{4.22}$$

then MGD will require less computing resources than SGD to achieve the optimization precision ρ*.*

In the case of big data, i.e., N is extremely large ($N \gg K$), or in an early stage of optimization, when we only require a rough optimization precision, i.e., ρ is allowed to be relatively large, such codebook may exist. In these cases, it may be beneficial to run MGD instead of GD or SGD since MGD has faster convergence speed than batch GD and SGD. Moreover, as opposed to pure serial computation in SGD, the gradient computation in each MGD iteration can be easily parallelized due to MGD allows much

larger mini-batch size. Therefore, MGD may provide an even faster training speed if multiple computing units are available.

4.4 Annealed Gradient Descent

Theorems 4.4 and 4.5 imply that MGD may possibly converge faster than either GD or SGD but it remains unclear how to find a codebook that simultaneously satisfy both conditions in these theorems. Moreover, different levels of optimization precision may be required in various stages of the training process. For example, at the beginning, when all model parameters stay far away from any local optimal point, we may not need to calculate a very accurate gradient, i.e., ρ is allowed to be relatively large at this time. On the other hand, as the parameters move towards a close neighbourhood of an optimal point, we may require a very small ρ to perform an accurate local search to converge more effectively. As suggested by Eq.(4.21), the required quantization error, ϵ, is proportionally related to ρ. For a fixed set of training data, the quantization error, ϵ, in turn depends on the size of the codebook, K. This suggests to use an annealing schedule of $\{\epsilon_1, \epsilon_2, \cdots\}$ (with $\epsilon_{i+1} < \epsilon_i$) for the whole training process, where ϵ gradually decreases as training continues. At the beginning, we can use a small low resolution codebook (relatively big ϵ) to run MGD to learn model parameters. As the training process proceeds, ϵ should gradually decrease by using larger and larger codebooks. At the final learning stage, the original training samples will be used to fine-tune model parameters.

Therefore, the basic idea of AGD is to construct deeply-structured hierarchical codebooks, in which quantization error ϵ slowly decreases from the top layer down to the bottom layer, and the last layer finally connects to the original training samples. The training procedure starts from the top to use each layer of codebooks to do MGD updates in Eq. (4.13) and gradually move down the hierarchy based on a pre-specified annealing schedule until we finally use the training samples to fine-tune the model parameters. If a proper annealing schedule is used, this annealed learning process may accelerate the training speed as implied by Theorems 4.4 and 4.5. More importantly, it may help to converge to a better local optimum at the end because AGD optimizes a much simpler mosaic objective functions from the early stage of training.

4.4.1 Hierarchical Codebooks

Here the regular k-means (with Euclidean metric) based top-down hierarchical clustering algorithm is used to construct the required hierarchical codebooks, where the centroid of each cluster is used as a codeword for each layer. The structure of the hierarchical codebooks is shown in Figure 4.1.

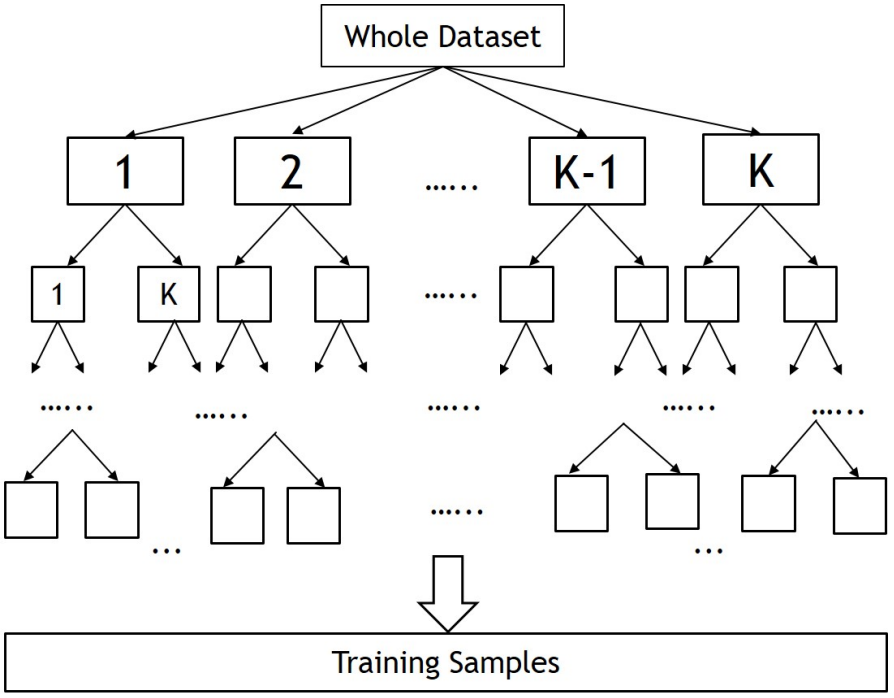

FIGURE 4.1: Illustration of a hierarchical codebook for AGD

To build the codebook, we firstly divide the training databases into several subsets based on their class labels (each subset only contains training samples belong to one class), then do hierarchical k-means on each subset. Notice that this is very easy to be parallelized because all subsets are independent with each other and can run hierarchical k-means respectively. We firstly define a split factor K, then using k-means to split each subset into K clusters, and using the centroid of each cluster to build the first layer of codebook. Then we again apply k-means on all clusters to divide them into K sub-clusters and get the next codebook layer. We do this procedure several times until the number of codewords in the last codebook layer becomes large enough. Finally we connect the original training samples at the bottom and get the hierarchical codebook, in which the sample size is gradually increase from the top layer to the bottom layer as Figure 4.1 shows. By using the k-means results of each layer, we can also build maps between all training data and their corresponding codewords. Figure 4.2 provides some

examples from a MNIST codebook, and from which we can see that the precision of the codewords are gradually increase.

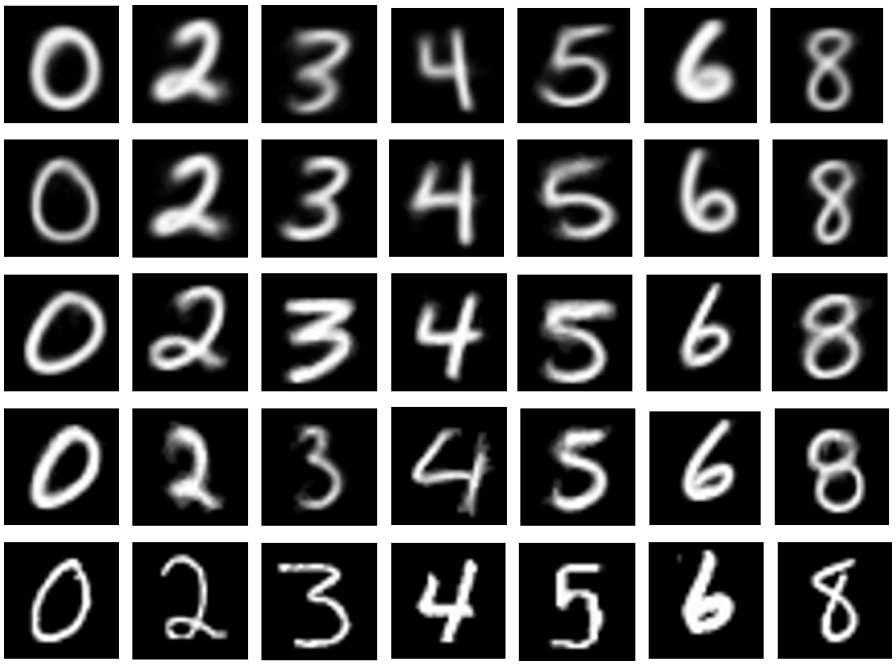

FIGURE 4.2: Examples of codewords in hierarchical codebooks of MNIST: Row 1: the second layer of the codebook; Row 2: the third layer of the codebook; Row 3: the fourth layer of the codebook; Row 4: the fifth layer of the codebook; Row 5: original training samples

4.4.2 Annealed Gradient Descent Algorithm

In AGD, we first specify an annealing schedule, i.e., $\{\epsilon_1, \epsilon_2, \cdots\}$ ($\epsilon_{i+1} < \epsilon_i$). In each epoch of AGD, for each training sample in each selected data batch, we select a codeword from the uppermost layer of the hierarchical codebooks that barely satisfies the required quantization error ϵ_i, to construct the mosaic function for MGD at this stage. Since ϵ_i gradually decreases in the annealing schedule, we can slowly move to use more and more precise codewords (eventually the original data samples) in AGD. The annealed gradient descent (AGD) algorithm is shown as in **Algorithm 3**. During the AGD training, varying batch sizes may even be applied. For example, we can start from a very large batch size (even the whole training set) at the beginning and slowly decrease the batch size from epoch to epoch. In general, we normally use much larger batch size than those used in mini-batch SGD. If a suitable annealing schedule is specified, many initial

Algorithm 3 Annealed Gradient Descent(AGD)

Input: training set O, hierarchical codebook C, annealing schedule $\{\epsilon_1, \epsilon_2, ..., \epsilon_T \mid \epsilon_{i+1} < \epsilon_i\}$
for each epoch $i = 1$ **to** T **do**
  for each batch X **do**
    For each training sample in X, select a codeword $c^{(n)}$ at the uppermost layer of the C satisfying ϵ_i;
    Use MGD to optimize the mosaic risk function;
  end for
end for

AGD epochs may be designated to run faster MGD with smaller codebooks, yielding faster training speed than mini-batch SGD or GD in overall.

4.5 Experiments

In this section, we apply the proposed AGD algorithm to learning fully connected DNNs for image classification in the MNIST database and speech recognition in the Switchboard database. AGD and the regular mini-batch SGD are both used to train DNNs based on the minimum cross-entropy error criterion. AGD is compared with mini-batch SGD in terms of total training time and the final recognition performance.

For each data set, we firstly use a standard k-means algorithm to build a deeply-structured hierarchical codebook. We use $K = 5$ as the split factor in MNIST and $K = 4$ in Switchboard which can result in enough depth of the codebooks. In our experiments MNIST database contains 10 classes and Switchboard contains 8991 classes, thus the hierarchical k-means has good potential to be parallelled. Therefore, the clustering procedure can be parallelized very easy among multiple of CPUs to make it much faster than the actual DNNs training. In our experiments, the running time of k-means clustering is about 4.3 hours in MNIST (using 5 CPUs) and about 9.2 hours in Switchboard (using 8 CPUs). If we use more CPUs, the k-means running time can be further reduced. As shown later, the k-means execution time is not significant when comparing with the necessary DNN training times. Moreover, for each database we only need to run k-means once and after that the same codebook can be applied to train DNNs based on different annealing schedules. Note that no pre-training is used in all experiments.

4.5.1 MNIST: Image Recognition

The MNIST database [99] contains 60,000 training images and 10,000 test images, and the images are 28-by-28 in size. Here, we use data augmentation through image deformation at the beginning of each epoch to enlarge the training set as in [2]. We use the configuration of the 3-hidden-layer DNN (1500, 1000, 500 nodes in each hidden layer) in [2] as the network structure and use SGD and AGD to do network training. We use ReLU as the activation function. Following [2], all hyper-parameters in the DNN are fine-tuned towards the best possible performance. In SGD, we use mini-batch of 10 samples and initial learning rate of 0.4. Notice that in MNIST experiments momentum is not applied during the training phase. In MGD, we use a much larger mini-batch size of 4500 and initial learning rate of 0.8. By observing the training cross-entropy and training error rate, we find that 550 epochs are enough to guarantee that the training has converged.

As we know, we should shrink the learning rates during the training process for better convergence. Specifically, when the training mean square error (MSE) rate becomes smaller than a pre-defined threshold r, we use the formula $\lambda_{t+1} = \lambda_t \cdot \frac{m}{m+t}$ to gradually decrease the learning rates, where m is a pre-defined constant that can control the decreasing speed of the learning rate. In our experiments, we set $r = 0.17$ and $m = 10000$.As for the AGD annealing schedule in MNIST, we start from $\epsilon_1 = 7.5$ (this value is based on the average quantization error in the first layer of the codebook) and $\epsilon_{i+1} = 0.999 \cdot \epsilon_i$.

In Figure 4.3, we have shown the learning curves of both SGD and AGD in terms of cross-entropy and classification error rate on the MNIST training set. Since each MGD epoch runs much faster than a SGD epoch, all learning curves are plotted as a function of total training time instead of epochs.

From the two pictures in Figure 4.3 we can see that AGD (in blue) finishes the same number of epochs much earlier than SGD (in red). Meanwhile, AGD also achieves a slightly better cross-entropy and classification error on the training set, and this fact implies that AGD converges to a better local minimum than SGD.

In addition, in Table 4.1, we also give the total training time and the best classification error on the test set for both AGD and SGD. From results in Figure 4.3 and Table 4.1 we can find that the proposed AGD training algorithm can yield slightly better classification

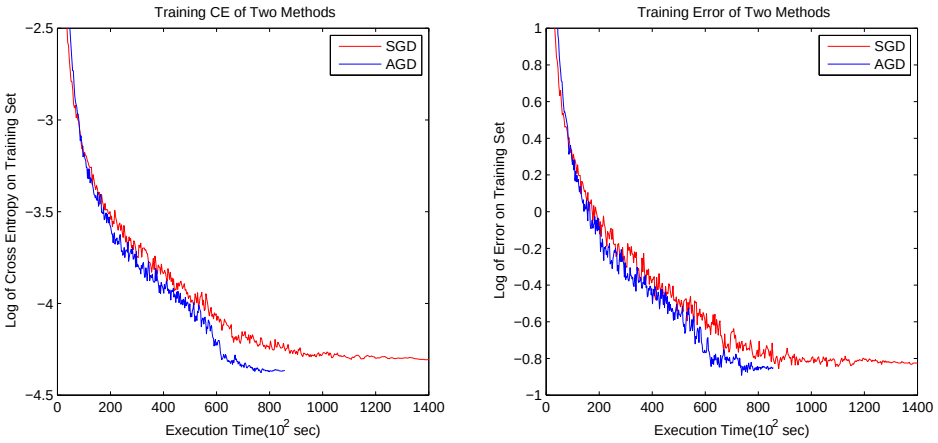

FIGURE 4.3: Learning curves on the MNIST training set (Left: cross entropy error; Right: classification error) of SGD and AGD as a function of elapsed training time.

TABLE 4.1: Comparison between SGD and AGD in terms of total training time (using one GPU) and the best classification error rate on MNIST.

METHODS	TRAINING TIME	TEST ERROR
SGD	30.3 *(hr)*	0.44%
AGD	18.4 *(hr)*	0.42%

performance in the test set, and more importantly reduce the total DNN training time by about 40%.

4.5.2 Switchboard: Speech Recognition

Switchboard is a 320-hour English transcription task, which contains 332,576 utterances in its training set (amounting to about 126 millions of training samples in total). The standard NIST Hub5e2000 is selected as the test set in this work, which has 1831 utterances. Following [100–102], we train a 6-hidden-layer DNN with 2048 nodes per layer based on minimum cross-entropy criterion. In Switchboard experiments ReLU are also used as activation function. We compare the cross entropy and frame classification errors on the training and test sets to evaluate the performance of SGD and AGD, meanwhile we also evaluate word error rates in speech recognition for the test set.

Here similar hyper-parameters of the DNN are used as in [100]. For example, we use mini-batch of 1024 samples and initial learning rate of 0.2 in SGD, and mini-batch of 6144 and initial learning rate of 1.0 in MGD. We use 0.9 as the momentum in both

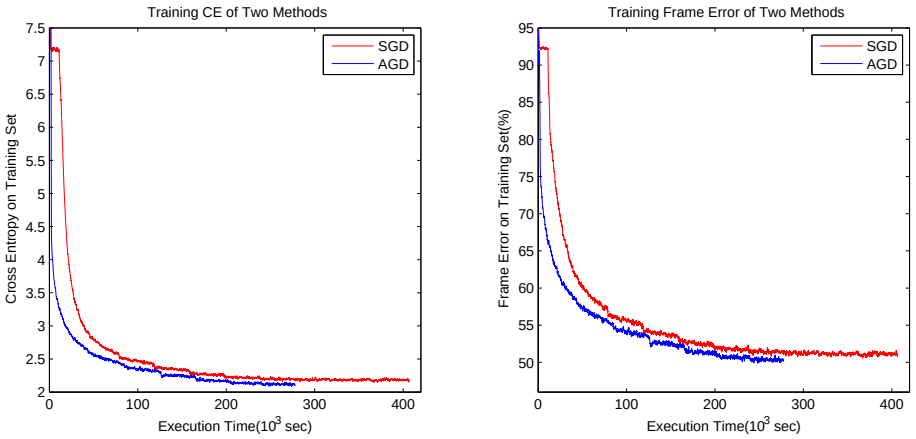

FIGURE 4.4: Learning curves on the Switchboard training set (Left: cross entropy error; Right: frame classification error) of SGD and AGD as a function of elapsed training time.

SGD and MGD. We run 10 epochs in SGD [100] and 17 epochs in AGD. During the training process, we need to shrink the learning rates slowly. Specifically, we multiply the learning rate by 0.8 every epoch after the 5-th epoch in SGD and the 12-th epoch in AGD. Note that our SGD baseline is solid and comparable with the best results reported in [100], [103] and [104] in terms of both training speed and recognition performance.

As for the AGD annealing schedule in Switchboard, unlike MNIST, it starts from the initial value ϵ_1 (17.5 in this case), and use the formula $\epsilon_{i+1} = \epsilon_i - \Delta_\epsilon$ to reduce ϵ_i by subtracting a constant value every epoch until it reaches the pre-defined value (8.5 in this case). The reason for using this formula is that the number of epochs is far less than MNIST experiments. In Switchboard, we have evaluated three different annealing schedules to show how they affect the training speed and the final recognition performance as in Table 4.2. In these three schedules, we use different values for Δ_ϵ, e.g. 1.0, 0.9 and 0.8 respectively. Obviously, the annealing schedule 1 ($\Delta_\epsilon = 1.0$) decreases fastest and it provides the best performance but relatively slower training speed. In contrast, schedule 3 ($\Delta_\epsilon = 0.8$) gives the fastest speed but slightly worse performance because more epochs will be dispatched to run MGD.

Figure 4.4 shows the learning curves of both SGD and AGD (using the annealing schedule 1 with $\Delta_\epsilon = 1$) in terms of cross-entropy and frame error on the Switchboard training set as a function of elapsed training time. Clearly, AGD runs much faster than SGD in Switchboard as well. Meanwhile, AGD can also achieve a slightly better local minimum than SGD as measured in both figures.

TABLE 4.2: Comparison between SGD and AGD in terms of total training time (using one GPU) and word error rate in speech recognition on Switchboard.

METHODS	TRAINING TIME	WORD ERROR
SGD	114.05 *(hr)*	15.5%
AGD *($\Delta_\epsilon = 1.0$)*	78.79 *(hr)*	15.3%
AGD *($\Delta_\epsilon = 0.9$)*	66.63 *(hr)*	15.6%
AGD *($\Delta_\epsilon = 0.8$)*	55.35 *(hr)*	16.2%

In Table 4.2, we give the total training times and the word error rates in speech recognition. We report the experimental results for all 3 different annealing schedules. The results have shown that the proposed AGD method can yield similar recognition performance with much faster training speed than SGD (about 30% to 40% reduction in total training time) when a suitable annealing schedule is used.

4.6 Conclusion

This chapter has proposed the annealed gradient descent (AGD) algorithm for non-convex optimization in deep learning, which can converge to a better local minimum with faster speed when compared with the regular mini-batch SGD algorithm. In this work, AGD has been applied to train large scale DNNs for image classification and speech recognition tasks. Experimental results have shown that AGD significantly outperforms SGD in terms of convergence speed. Therefore, the AGD algorithm is especially suitable for the large scale non-convex optimization problems in deep learning. One potential future work is to combine AGD with convolutional neural networks (CNNs) and more complex image recognition tasks. In this case we may run k-means on image patches instead of the whole input images.

Chapter 5

Hybrid Orthogonal Projection and Estimation for CNNs

Chapter 4 presented the AGD algorithm, which can be used to train deep learning models more efficiently and effectively. This chapter will focus on a novel deep learning model, i.e., Hybrid Orthogonal Projection and Estimation (HOPE), which can improve the performance of both DNNs and CNNs.

5.1 Introduction

CNNs [19] currently play an important role in the deep learning and computer vision fields. Researchers have revealed that CNNs can give the state-of-the-art performance in many computer vision tasks, especially for image classification and recognition tasks [105–107]. Comparing with the fully connected DNNs, CNNs are superior in exploiting spatial constraints and in turn extracting better local features from input images using the convolution layers and weight sharing, and furthermore may provide better invariance through the pooling mechanism. All of these make CNNs very suitable for image-related tasks [20]. Moreover, large-scale deep CNNs can be effectively learned end-to-end in a supervised way from a large amount of labelled images.

In the past several years, a tremendous amount of research efforts have been devoted to further improve the performance of deep CNNs. In [108, 109], the dropout method has been proposed to prevent CNNs from over-fitting by randomly dropping a small portion of hidden nodes in the network during the training procedure. Many experiments

have confirmed that the dropout technique can significantly improve the network performance, especially when only a small training set is available. Besides, a similar idea, called dropconnect [110], has been proposed to drop connections between layers instead of hidden nodes during the training stage. Another interesting research field is to design good nonlinear activation functions for neural networks beyond the popular ReLU, such as maxout [111] and PReLU [30], which are also demonstrated to yield improvement in terms of classification performance. On the other hand, another important path to improve model performance is to search for some new CNN structures. For example, in [112], Network in Network (NIN) has been proposed, in which one micro neural network is used to replace the regular linear convolutional filter. Recurrent Convolutional Neural Network (RCNN) [113] is another new CNN structure, which introduce recurrent connections into the convolution layers. In [114], the spectral pooling method is proposed, which applies discrete Fourier transform into the pooling layers to preserve more useful information after the dimensionality reduction. In [42], it has proposed a new CNN structure using larger stride convolution layers to replace the pooling layers, and the authors argue that the larger stride convolution layers can perform equally well as the pooling layers and also achieve similar performance in the experiments.

More recently, a novel model, called Hybrid Orthogonal Projection and Estimation (HOPE) [22], has been proposed to learn neural networks in either supervised or unsupervised ways. This model introduces a linear orthogonal projection to reduce the dimensionality of the raw high-dimension data and then uses a finite mixture distribution to model the extracted features. By splitting the feature extraction and data modeling into two separate stages, it may derive a good feature extraction model that can generate better low-dimension features for the further learning process. More importantly, based on the analysis in [22], the HOPE model has a tight relationship with neural networks since each hidden layer of DNNs can also be viewed as a HOPE model being composed of the feature extraction stage and data modeling stage. Therefore, the maximum likelihood based unsupervised learning as well as the minimum cross-entropy error based supervised learning algorithms can be used to learn neural networks under the HOPE framework for deep learning. In this case, the standard back-propagation method may be used to optimize the objective function to learn the models except that the orthogonal constraints are imposed for all projection layers during the training procedure.

However, [22] has not taken CNNs into account but merely investigated the HOPE models for the fully connected DNNs and demonstrated good performance in the small MNIST data set. To make the HOPE model work with more image-related tasks, it is

important to consider how to combine the basic idea of the HOPE model with non-fully connected CNNs. In this chapter, we extend the HOPE model to the popular CNNs by considering the special model structures of both convolution and pooling layers, and further consider how to introduce the orthogonal constraints into the CNN model structure and learn CNNs under the HOPE framework. The most straightforward idea is to use a HOPE layer as the first hidden layer in CNNs to de-correlate the high-dimension input CNN features and remove the irrelevant noises as a result, which is called HOPE-Input layer. This idea is similar as the original formulation in [22] except the HOPE model is applied to each convolutional filter. Moreover, we may introduce even more HOPE layers into the CNNs for better performance. Generally speaking, we can split one CNN into several building blocks, and each block may include several convolutional layers and end with one pooling layer. For each convolutional layer in one block, a HOPE layer (which includes one orthogonal projection layer and one model layer) can be used to replace it. For the pooling layer, we just use one HOPE projection layer to replace it, since the orthogonal projection procedure shares a similar purpose with pooling, i.e., reduce the resolution of the lower-level feature maps and then make the models more tolerable to the slight distortion or translation in the original images [19]. This projection layer, along with the first convolutional layer in the next block, can be viewed as another HOPE layer. In practice, we can either replace one block (single-HOPE-Block CNNs) or multiple blocks (multi-HOPE-Blocks CNNs) by using HOPE layers for better performance. Our experimental results on both CIFAR-10, CIFAR-100 and ImageNet databases have shown that the application of HOPE layers results in significant performance improvement over the regular CNN baseline models [1].

5.2 Hybrid Orthogonal Projection and Estimation (HOPE) Framework

In the original Hybrid Orthogonal Projection and Estimation (HOPE) formulation [22], it is assumed that any high-dimension feature vector can be modelling by a hybrid model consisting of feature extraction using a linear orthogonal projection and statistic modeling using a finite mixture model. Assume each high-dimension feature vector $\mathbf{x}$ is of dimension D, the linear orthogonal projection will map $\mathbf{x}$ to an M-dimension feature space ($M < D$), and the projected vector may retain the most useful information of $\mathbf{x}$.

[1] The code of our HOPE CNN can be downloaded via: https://github.com/mowangphy/HOPE-CNN

Specifically, we can define a $D \times D$ orthogonal matrix $[\mathbf{U};\ \ V]$ which satisfies:

$$[\mathbf{z};\ \ \mathbf{n}] = \begin{bmatrix} \mathbf{U} \\ V \end{bmatrix} \mathbf{x} \tag{5.1}$$

where $\mathbf{z}$ is an M-dimension vector, called the signal component, and $\mathbf{n}$ is the residual noise vector with the dimensionality of $D - M$.

In practice, $\mathbf{z}$ is heavily de-correlated but it may still locate in a rather high dimension feature space. In the HOPE formulation, it is proposed to model $\mathbf{z}$ with a finite mixture model:

$$p(\mathbf{z}) = \sum_{k=1}^{K} \pi_k \cdot f_k(\mathbf{z}|\theta_k) \tag{5.2}$$

where K is the number of mixture components, π_k is the mixture weight of the kth component ($\sum_{k=1}^{K} \pi_k = 1$), $f_k()$ denotes a selected distribution from the exponential family, and θ_k denotes all model parameters of $f_k()$. As discussed in [22], if the von Mises-Fisher (vMF) distribution is chosen for $f_k()$, the resultant HOPE model is equivalent in mathematical formulation to a hidden layer in neural networks using the popular rectified linear units (ReLU).

The HOPE model combines a linear orthogonal projection and a finite mixture model under a unified generative modeling framework. It can be learned unsupervisingly based on maximum likelihood estimation from unlabelled data as well as discriminatively from labelled data. In [22], the HOPE model has been applied to the fully connected DNNs and learn the models accordingly in either supervised or unsupervised ways. For one hidden layer with input vector $\mathbf{x}$ ($\mathbf{x} \in R^D$) and output vector $\mathbf{y}$ ($\mathbf{y} \in R^G$), it is first split into two layers: i) The first layer is a linear orthogonal projection layer, which is used to project $\mathbf{x}$ to a feature vector $\mathbf{z}$ ($\mathbf{z} \in R^M, M < D$) and remove the noise signals by using an orthogonal projection matrix $\mathbf{U}$:

$$\mathbf{z} = \mathbf{U}\mathbf{x}. \tag{5.3}$$

ii) The second layer is a non-linear model layer, which convert $\mathbf{z}$ to the output vector $\mathbf{y}$ following the selected model $f_k()$ and a nonlinear log-likelihood pruning operation. An example of a HOPE layer in DNNs is shown in Figure 5.1.

As in [22], all HOPE model parameters, including the projection matrix U and the model matrix W, can be learned, using the error back-propagation algorithm with stochastic

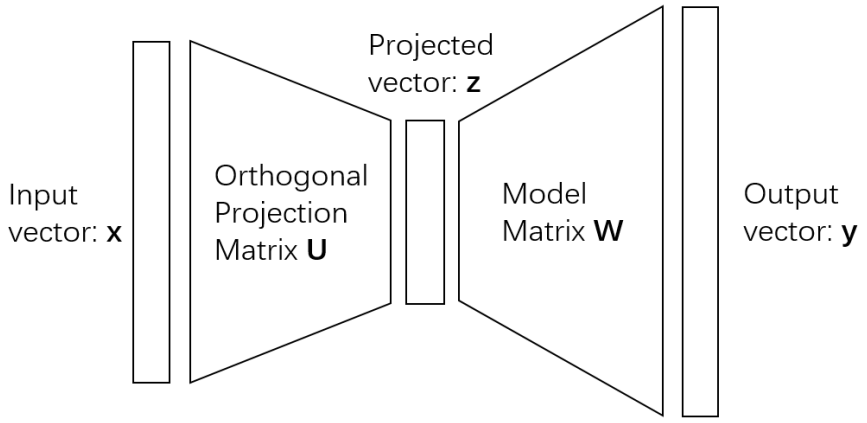

FIGURE 5.1: The HOPE model is viewed as a hidden layer in DNNs.

gradient descent, to optimize an objective function subject to an orthogonal constraint, $\mathbf{U}\mathbf{U}^T = \mathbf{I}$, for each projection layer. As in [22], for computational simplicity, the constraint is cast as the following penalty term to gradually de-correlate the matrix $\mathbf{U}$ during the learning process:

$$P(\mathbf{U}) = \sum_{i=1}^{M} \sum_{j=i+1}^{M} \frac{|\mathbf{u}_i \cdot \mathbf{u}_j|}{|\mathbf{u}_i| \cdot |\mathbf{u}_j|}. \tag{5.4}$$

In [22], both unsupervised learning and supervised learning are studied for DNNs under the HOPE framework. The above orthogonal constraint is found to be equally important in both scenarios. In this chapter, we will study how to supervisingly learn CNNs under the HOPE formulation and more specifically investigate how to introduce the orthogonality into the CNN model structure.

5.3 HOPE model for CNNs

In [22], the authors have applied the HOPE model to the fully connected DNNs and have achieved good performance in experiments on small data sets like MNIST. However, more widely used neural models in computer vision, i.e. CNNs, have not been considered. Unlike DNNs, CNNs adopt some unique model structures and have achieved huge successes in many large-scale image classification tasks. Therefore, it is interesting to consider how to combine the HOPE model with CNNs to further improve image classification performance.

5.3.1 Applying the HOPE model to CNNs

To apply the HOPE model to CNNs, the most straightforward solution is to split each convolution layer into a concatenation of a projection layer and a model layer and impose the orthogonal constraints onto the projection layer as in [22]. Assume that we have a regular convolution layer in CNNs, which uses some $S \times S$ linear filters to map from C_i input feature maps to C_m output feature maps. As shown in Figure 5.2, under the HOPE framework, we propose to split this convolution layer into the two separate layers:

i) One linear orthogonal projection layer with the projection matrix $\mathbf{U}$: it linearly maps a 3-dimension tensor with the size of $S \times S \times C_i$ into a vector $1 \times 1 \times C_p$, C_p denotes the feature maps to be used in this projection layer. As the projection filters convolve with the input layer, it generates a total of C_p feature maps in the projection layer. The projection filter itself is a 4-dimension tensor with the size of $S \times S \times C_i \times C_p$. Based on the definition of the convolution procedure and follow the formulation in [22], we can reshape this 4-dimension tensor as a matrix $\mathbf{U}$ with the size of $(S \cdot S \cdot C_i) \times C_p$, as shown in Figure 5.2.

ii) One non-linear model layer with the weight matrix W: it has exactly same structure as a regular convolutional layer, which maps the C_p projected feature maps into C_m output feature maps. Differing from [22], instead of only mapping the projected vector, the proposed model layer here takes all projected vectors within each $S \times S$ region into account and map all projected features within this region into the final output feature maps. We have found that this modification is critical in CNNs for better performance in image classification. In our implementation, we use ReLU as the non-linear activation function. Since we apply supervised learning method to learn HOPE CNNs, we do not need to explicitly define the mixture model, and the mixture model can be learned automatically from training data.

Figure 5.2 shows the whole structure of one HOPE layer in CNNs. Since the projection layer is linear, we may collapse these two layers to derive a normal convolution layer in CNNs. However, as argued in [22], there are many advantages to separate them so as to learn CNNs under the HOPE framework.

Note that C_p is always far less than $S \times S \times C_i$ in the above HOPE formulation, it implies that the orthogonal projection may help to remove irrelevant noises in this step.

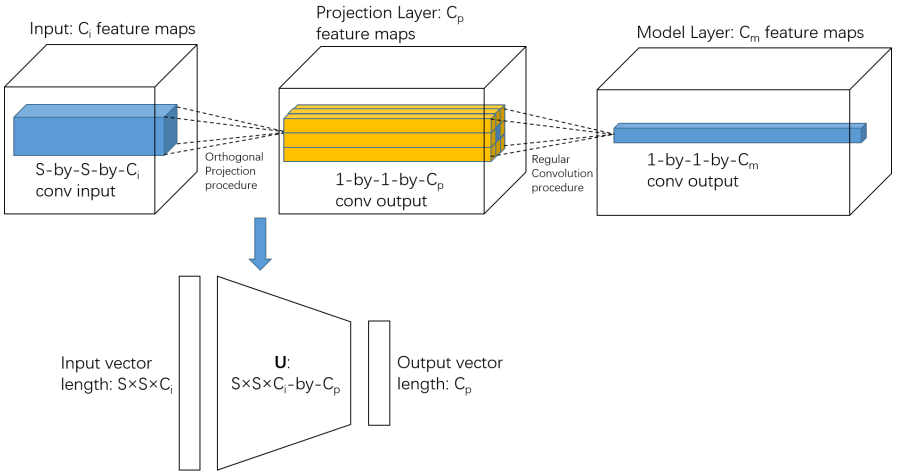

FIGURE 5.2: A convolution layer in CNNs may be viewed as a HOPE model.

In this paper, we only consider the supervised learning of CNNs under the HOPE framework. In this case, the model parameters in the model layer can be learned in the same way as in the convolutional CNNs. However, for the projection layers, we need to impose the orthogonal constraint, $\mathbf{U}\mathbf{U}^T = \mathbf{I}$ during the learning process. Following [22], we cast this constraint as a penalty term in eq. (5.4).

First of all, we need to derive the gradient of the penalty term $P(\mathbf{U})$ with respect to $\mathbf{U}$ as follows:

$$\frac{\partial P(\mathbf{U})}{\partial \mathbf{u}_i} = \sum_{j=1}^{M} \left(\frac{|\mathbf{u}_i \cdot \mathbf{u}_j|}{|\mathbf{u}_i| \cdot |\mathbf{u}_j|}\right) \cdot \left(\left(\frac{\mathbf{u}_j}{\mathbf{u}_i \cdot \mathbf{u}_j}\right) - \left(\frac{\mathbf{u}_i}{\mathbf{u}_i \cdot \mathbf{u}_i}\right) \right) \tag{5.5}$$

To facilitate the above computation in GPUs, we may equivalently represent the above gradient computation as a matrix form, i.e., essentially a multiplication of the two matrices $\mathbf{D}$ and $\mathbf{B}$ as follows:

$$\frac{\partial P(\mathbf{U})}{\partial \mathbf{U}} = (\mathbf{D} - \mathbf{B})\mathbf{U} \tag{5.6}$$

where $\mathbf{D}$ is an M-by-M matrix of $d_{ij} = \frac{\text{sign}(\mathbf{u}_i \cdot \mathbf{u}_j)}{|\mathbf{u}_i| \cdot |\mathbf{u}_j|}$ $(1 < i, j < M)$ and $\mathbf{B}$ is another M-by-M diagonal matrix of $b_{ii} = \frac{\sum_j g_{ij}}{\mathbf{u}_i \cdot \mathbf{u}_i}$ with $g_{ij} = \frac{|\mathbf{u}_i \cdot \mathbf{u}_j|}{|\mathbf{u}_i| \cdot |\mathbf{u}_j|}$ $(1 < i, j < M)$.

Secondly, we can combine the above $\frac{\partial P(\mathbf{U})}{\partial \mathbf{U}}$ with the gradient $\Delta\mathbf{U}$, which is calculated from the objective function:

$$\widetilde{\Delta \mathbf{U}} = \Delta \mathbf{U} + \beta \cdot \frac{\partial P(\mathbf{U})}{\partial \mathbf{U}} \tag{5.7}$$

where β is a pre-defined parameter to balance the orthogonal penalty term. Finally, the projection matrix $\mathbf{U}$ can be updated as follows:

$$\mathbf{U}^{(n)} = \mathbf{U}^{(n-1)} - \gamma \cdot \widetilde{\Delta \mathbf{U}} \tag{5.8}$$

where γ is the learning rate for the weight update. During the learning process, $\mathbf{U}$ is gradually de-correlated and eventually becomes an orthogonal matrix.

5.3.2 The HOPE-Input Layer

The first way to apply the HOPE model to CNNs is to use the above HOPE layer to replace the first convolution layer right after the image pixel input. The HOPE formulation may help to de-correlate the raw image pixel inputs and filter out irrelevant noises in the first place. This is called as one HOPE-Input layer.

5.3.3 HOPE-Blocks

In many cases, simply applying one HOPE-Input layer is not enough to remove noise signals from features and achieve good performance. Therefore, we need to introduce more HOPE layers into the baseline CNN. In practice, one CNN can be divided into some building blocks, and each block may include several convolutional layers and end with one pooling layer. We can use these blocks as the basic units to introduce HOPE layers. Figure 5.3 shows an example of one HOPE-Block, and here we apply three HOPE layers to replace the corresponding convolutional layers (for the first convolutional layer, the projection layer is from the last block).

For the pooling layer, we may need to consider a slightly different way to apply HOPE model. In CNNs, the pooling layers [40] are traditionally considered as important for good performance. [42] has shown that the pooling layers result in the reduction of feature dimensionality, which help the CNNs to *view* much larger regions of the input feature maps, and generate more stable and invariant high level features.

Since the projection layer in HOPE models shares a similar objective with pooling layers, i.e., reducing the feature dimensionality, remove noise and increase the stability of the feature, we can just use one HOPE projection layer to replace one pooling layer, and

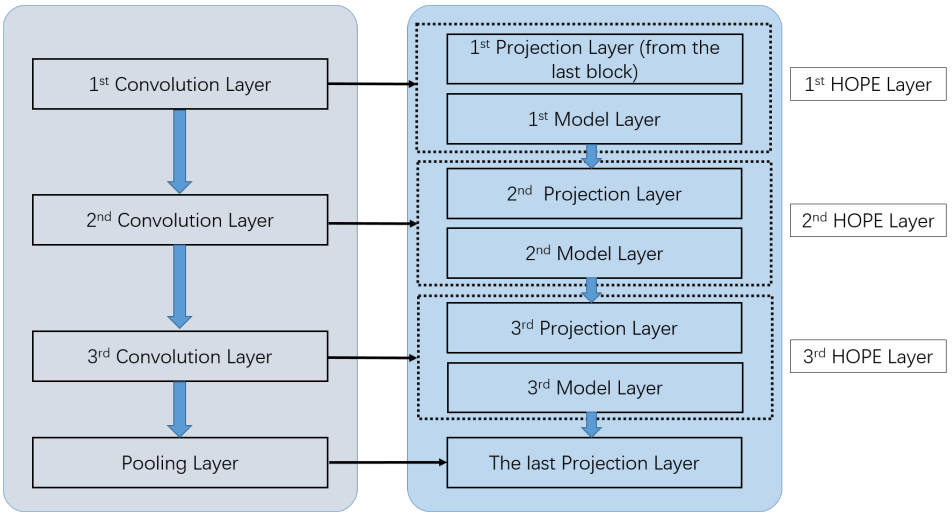

FIGURE 5.3: One HOPE-Block

view the next convolutional layer as the model layer. Comparing with the regular pooling layers, we believe that the HOPE projection layer may be advantageous in feature extraction since the learnable linear orthogonal projection may help to de-correlate the input feature maps more precisely and generate better features for the upper layers.

In practice, we can just introduce one HOPE-Block to replace the first building block in the baseline CNN (single-HOPE-Block), or apply multiple HOPE-Blocks (multi-HOPE-Blocks).

5.4 Experiments

In this section, we use three widely used image classification databases, namely CIFAR-10, CIFAR-100 [23] and ImageNet [24], to evaluate the performance of our proposed HOPE methods.

5.4.1 Databases

CIFAR-10 and CIFAR-100 are two popular databases that are widely used in computer vision. Both databases contain 50,000 32-by-32 RGB images for training and 10,000 images for validation. The main difference between these two databases is that CIFAR-10 only divides all images into 10 coarse classes, but CIFAR-100 divides them into 100 fine classes.

To expand the training sample size and reduce over-fitting, we also consider to use data augmentation techniques on the two databases. Specifically, In each mini-batch, we will randomly select half of the images to apply four kinds of augmentation methods respectively:

- **Translation:** The selected images will be randomly translate horizontally and vertically for at most 5 pixels
- **Rotation:** The selected images will be randomly rotated by the -5 to 5 degree. The rotated images should be cropped to keep the original size.
- **Scaling:** We firstly randomly extract a patch from the input image (the patch size is pre-defined), and resize the patch to the original image size. This procedure equals to zoom-in.
- **Color space translation:** For each channel of one image, we define a translation matrix. Each element in the translation matrix is corresponding to one pixel in the corresponding color channel, and the value lays between 0.95 and 1.05. The image will element-wise multiply with the translation matrix for the color space translation.

Comparing with CIFAR-10 and CIFAR-100, ImageNet is an image database with much larger sample size (nearly 1.3 million training images and 50,000 validation images in the classification tasks), and all images are divided into 1000 classes. During the experiments, all images should be re-scaled to 224-by-224 to feed into CNNs.

5.4.2 CIFAR Experiments

In our CIFAR-10 and CIFAR-100 experiments, we consider several different CNN structures as specified in Figure 5.4 in detail. Firstly, we follow the CNN structure that is defined by Sergey Zagoruyko as our baseline CNNs.[2] Then we evaluate the HOPE-Input CNN, single-HOPE-Block and multi-HOPE-Block (using 2, 3, 4 and 5 HOPE blocks respectively) CNNs as discussed in Section 5.3, and compare them with the baseline models. In Figure 5.4, we have provided the detailed description of the structure of four CNNs (baseline, HOPE-Input, single-HOPE-Block and multi-HOPE-Blocks (using 5-HOPE-Blocks CNN as the example)) used in our experiments.

[2]See https://github.com/szagoruyko/cifar.torch for more information. According to the website, without using data augmentation, the best performance on the CIFAR-10 test set is 8.7% in error rate. By using RGB color channel instead of YUV, our reproduced baseline performance is 8.30% in this chapter.

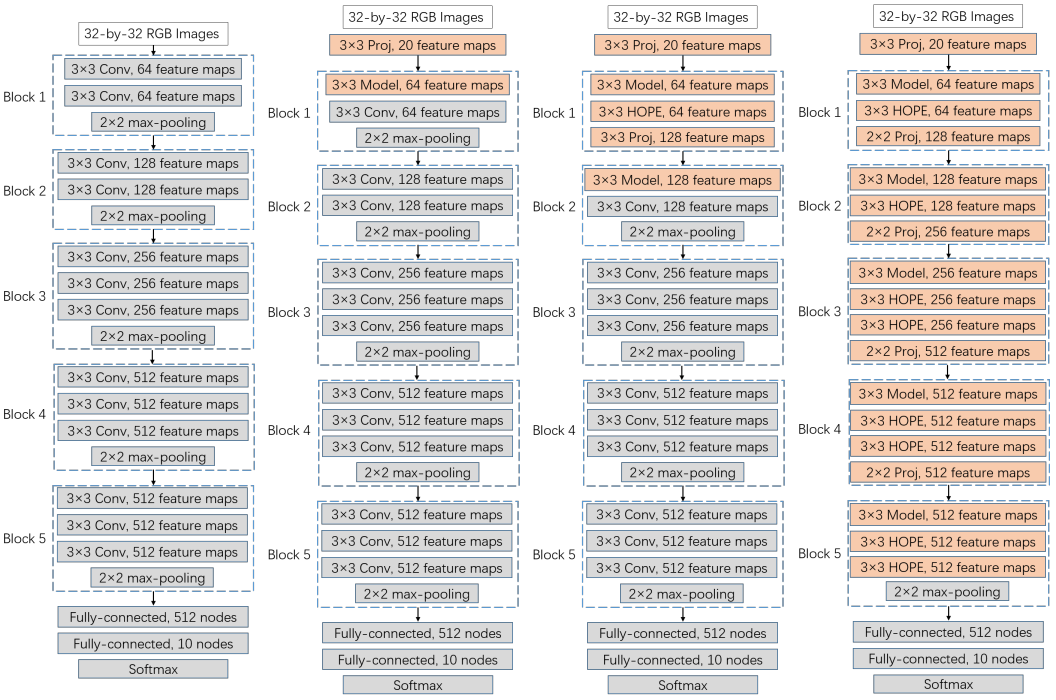

FIGURE 5.4: From Left to Right: Baseline CNN, HOPE-Input CNN, single-HOPE-Block CNN, and multi-HOPE-Blocks CNN. Where *Proj* denotes to one HOPE projection layer, *Model* denotes to one HOPE model layer, and *HOPE* denotes to one whole HOPE layer (includes one projection layer and one model layer).

In Figure 5.4, for the HOPE-Input layers, we use 20 feature maps in the projection layer. For the HOPE layer in the Block 1 (single-HOPE-Block and multi-HOPE-Blocks CNNs), the projection layer contains 48 feature maps. For the rest HOPE layers in the multi-HOPE-Blocks CNN, the projection layer should have the same number of feature maps as the model layer. In practice, the feature map number of the input layer and the first block should be tuned very carefully. The reason of this fact is that the shallow layers tend to have highly correlated data, which may more sensitive to the size of projection layers. It is easy to learn that too many feature maps or too few feature maps can both lead to worse performance. We did several tests on HOPE-Input and single-HOPE-Block CNNs to determine the best scale of the projection layers, and the results are shown in Table 5.1.

To further investigate the performance of the HOPE CNNs (HOPE-Input, single-HOPE-Block and multi-HOPE-Blocks), we also consider the model configurations called as LIN CNNs (Lin-Input, single-Lin-Block and multi-Lin-Blocks), which uses the same model structure as the HOPE CNNs except that the orthogonal constraint in Eq. (5.4) is NOT applied in training.

TABLE 5.1: The relationship of the size of projection layers and classification performance (using CIFAR-10 tests as examples).

	PROJ LAYER 1	PROJ LAYER 2	TEST ERR
HOPE-INPUT	10	-	8.01%
	15	-	7.86%
	20	-	7.81%
	25	-	7.92%
SINGLE-HOPE-BLOCK	20	24	8.12%
	20	32	7.71%
	20	48	7.29%
	20	64	7.57%

In all CIFAR experiments, we use the mini-batch SGD with a batch size of 100 images to perform 400 epochs of network training. The initial learning rate is 0.06, and the learning rate should be halved after every 25 epochs. We also use momentum of 0.9 and weight decay rate of 0.0005. In batch normalization [115], we set $\epsilon = 0.001$. For the HOPE layers, we use a initial β that equals to 0.15, and the β should be divided by 1.75 after every 25 epochs. All weights in CNNs are initialized by using the method proposed by He et al[30]. For the multi-HOPE-Blocks CNN experiments, we introduce 2 - 5 HOPE-Blocks (overall 6 blocks), since the last block is the fully connected layers.

TABLE 5.2: The learning speed of different CNNs.

METHODS	LEARNING SPEED
BASELINE	220 IMAGES/S
LIN-INPUT	206 IMAGES/S
HOPE-INPUT	203 IMAGES/S
SINGLE-LIN-BLOCK	200 IMAGES/S
SINGLE-HOPE-BLOCK	195 IMAGES/S
2-LIN-BLOCKS	195 IMAGES/S
2-HOPE-BLOCKS	186 IMAGES/S
3-LIN-BLOCKS	178 IMAGES/S
3-HOPE-BLOCKS	170 IMAGES/S
4-LIN-BLOCKS	157 IMAGES/S
4-HOPE-BLOCKS	152 IMAGES/S
5-LIN-BLOCKS	149 IMAGES/S
5-HOPE-BLOCKS	141 IMAGES/S

5.4.2.1 Learning Speed

We firstly consider the computational efficiency of the proposed HOPE methods in learning CNNs. Our computing platform includes Intel Xeon E5-1650 CPU (6 cores), 64 GB memory and a Nvidia Geforce TITAN X GPU (12 GB memory). Our method is implemented with MatConvNet [116], which is a CUDA based CNN toolbox in Matlab. The learning speed of all CNNs are listed in Table 5.2.

From Table 5.2, we can see that using the more complicated HOPE layers in CNNs will slow down the computation of CNNs in GPUs, but the speeds are still affordable. Moreover, the learning speed of the HOPE methods is similar with the corresponding LIN methods, which implies that the computational overhead for the orthogonal projection constraint is negligible in training.

5.4.2.2 Performance on CIFAR-10 and CIFAR-100

We use the classification error rate on the test sets of the selected databases to evaluate the performance of all CNN models. Besides the CNN configurations we mentioned above, we also include several well-known CNN models from the previous work to compare with our methods, including Tree-Pooling [43], DNGO [117] , Fitnet4-LSUV [118], Spectral Pooling [114], R-CNN [113], ALL-CNN [42], Maxout Networks [111] and Network in Network [112]. All selected models have the comparable model size with our proposed CNNs.

From all results summarized in Table 5.3, we can see that the proposed HOPE-based CNNs models work well in both CIFAR-10 and CIFAR-100 databases. In the cases that the data augmentation methods are not applied, the single-HOPE-Block CNNs can achieve the best performances on both CIFAR-10 and CIFAR-100, which is the state-of-the-art performance without data augmentation. Moreover, we can see that HOPE-Input, single-HOPE-Block and multi-HOPE-Blocks CNNs consistently outperform the counterpart LIN models that do not use the orthogonal constraints. This implies that the orthogonality introduced by the HOPE methods is quite useful to improve the performance of CNNs in the image classification tasks.

Table 5.3 also shows that after data augmentation the proposed HOPE method can also achieve state-of-the-art performance on both CIFAR-10 and CIFAR-100 databases.

TABLE 5.3: The classification error rates of all examined CNNs on the test set of CIFAR-10 and CIFAR-100. CIFAR-10+ and CIFAR-100+ denote to the databases with data augmentation.

	CIFAR-10	CIFAR-10+	CIFAR-100	CIFAR-100+
BASELINE	8.30%	6.96%	30.71%	29.38%
LIN-INPUT	7.97%	7.01%	30.13%	28.91%
HOPE-INPUT	7.81%	6.80%	29.96%	28.79%
SINGLE-LIN-BLOCK	8.30%	7.27%	31.85%	30.27%
SINGLE-HOPE-BLOCK	**7.29%**	**6.05%**	**29.47%**	**26.99%**
2-LIN-BLOCKS	8.63%	7.41%	33.01%	31.46%
2-HOPE-BLOCKS	7.59%	6.24%	31.04%	27.56%
3-LIN-BLOCKS	8.97%	7.86%	34.62%	32.16%
3-HOPE-BLOCKS	7.75%	6.48%	31.19%	28.27%
4-LIN-BLOCKS	9.31%	8.10%	38.70%	35.15%
4-HOPE-BLOCKS	7.98%	6.53%	32.06%	28.73%
5-LIN-BLOCKS	9.33%	8.16%	39.28%	35.52%
5-HOPE-BLOCKS	7.96%	6.61%	32.66%	28.70%
TREE-POOLING [43]	7.62%	6.05%	32.37%	-
DNGO [117]	-	6.37%	-	27.40%
FITNET4-LSUV [118]	-	6.06%	-	27.66%
SPECTRAL POOLING [114]	8.60%	-	31.60%	-
R-CNN [113]	8.69%	7.09%	31.75%	-
ALL-CNN [42]	9.08%	7.25%	33.71%	-
MAXOUT [111]	11.68%	9.38%	34.54%	-
NETWORK IN NETWORK [112]	10.41%	8.81%	35.68%	-

One possible reason of the fact that single-HOPE-Block CNNs show better performance compare with multi-HOPE-Blocks CNNs is that multi-HOPE-Blocks introduce much more HOPE layers than single-HOPE-Block CNNs, which will significantly increase the number of model parameters under our network configuration (nearly 100% more compare with the baseline network in the 5-HOPE-Blocks CNN). This may lead to overfitting and decrease the performance on the test sets.

5.4.3 ImageNet Experiments

In our ImageNet experiments, we directly apply VGG-16 [119] as the baseline network [3]. Then we evaluate the HOPE-Input CNN and single-HOPE-Block CNN as discussed in section 5.3, and compare them with the original VGG-16 model. Similar to CIFAR

[3]See http://http://www.vlfeat.org/matconvnet/pretrained/ for more information. According to the website, the top-5 error of VGG-16 on ImageNet validation set is 9.5% on MatConvNet platform

experiments, we also take the corresponding LIN CNNs into account to further demonstrate the effectiveness of HOPE.

In all ImageNet experiments, we use 128 mini-batch size to train the CNNs for 50 epochs. The initial learning rate and β are 0.025 and 0.035 respectively, and both of them should be halved after every 3 epochs. The momentum and weight decay rate are 0.9 and 0.0005 respectively. In batch normalization [115], we set $\epsilon = 0.001$. All weights in CNNs will be initialized by using the method proposed by He et al [30].

5.4.3.1 Learning Speed

Since ImageNet database has very large sample size, and the CNNs are much deeper comparing with their counterparts in CIFAR experiments, we use 4 Nvidia Geforce TITAN X GPUs to do network training, and the learning speeds are listed in Table 5.4.

TABLE 5.4: The learning speed and top-5 classification error (validation set) of different CNNs on ImageNet experiments.

METHODS	LEARNING SPEED	TOP-5 ERROR
VGG-16	58 IMAGES/S	9.5%
LIN-INPUT	56 IMAGES/S	9.5%
HOPE-INPUT	55 IMAGES/S	9.3%
SINGLE-LIN-BLOCK	53 IMAGES/S	9.6%
SINGLE-HOPE-BLOCK	52 IMAGES/S	**9.0%**

5.4.3.2 Performance on ImageNet

From Table 5.4, we can see that HOPE models also work well on the ImageNet database. And Single-HOPE-Block CNN shows the best performance among all selected models. HOPE-Input CNN can also yield improvement over the VGG-16 baseline model.

5.5 Conclusions

This chapter proposes several methods to apply the recent HOPE model to CNNs for image classification. We have analyzed the relationship between the CNNs and HOPE model, and found a suitable way to use the HOPE method to replace the convolution

and pooling layers in CNNs. Experimental results on the CIFAR-10, CIFAR-100 and ImageNet databases have shown that our proposed HOPE methods work well with C-NNs, and can yield the state-of-the-art classification performance in these two data sets. This study has confirmed that the orthogonal constraints imposed by the HOPE models can significantly improve the performance of CNNs in these image classification tasks.

Chapter 6

A CNN based Fast Image Saliency Detection Method

This chapter will discuss how to use CNNs to deal with the problem of image saliency detection. 'Visual saliency' is originally a biological definition, which represent the objects in the visual world that can attract human beings' attention. In computer vision, the goal of image saliency detection is the find one or more image regions that may attract most attention of viewers [120]. Generally speaking, many image saliency detection system can generate pixel-wised saliency maps for the given input images, and the value of each pixel in one saliency map denotes how likely it belongs to a salient region.

6.1 Introduction

This chapter discusses one important application of deep learning: how to apply the popular deep learning techniques to one important computer vision problem, namely image saliency detection. The saliency detection attempts to locate the objects that have the most interests in an image, where human may also pay more attention [120]. The main goal of the saliency detection is to compute a saliency map that topographically represents the level of saliency for visual attention [121]. For each pixel in an image, the saliency map can provide how likely this pixel belongs to the salient objects [122]. Computing such saliency maps has recently raised a great amount of research interest [123]. The computed saliency maps have been shown to be beneficial to various vision

tasks, such as image segmentation, object recognition and visual tracking [10]. Saliency detection has been extensively studied in computer vision, and a variety of methods have been proposed to generate the saliency maps for images. Under the assumption that the salient objects probably are the parts that significantly differ from their surroundings, most of the existing methods use low-level image features to detect saliency regions based on the criteria related to color contrast, rarity and symmetry of image patches [9, 10, 120, 122, 124, 125].

In some cases, the global topological cues may be leveraged to refine the perceptual saliency maps [121, 126]. In these methods, the saliency is normally measured based on different mathematical models, including decision theoretic models, Bayesian models, information theoretic models, graphical models, and spectral analysis models [123]. For example, in [121], a boolean map is created to represent global topological cues in an image, which in turn is used to guide the generation of the saliency maps. In [127], the visual saliency algorithm considers the prior information and the local features simultaneously in a probabilistic model. The algorithm defines task-related components as the prior information to help the feature selection procedure when generating the saliency maps.

The traditional saliency detection methods mentioned above normally work well for the images contain simple dominant foreground objects in a homogenous background. However, they are usually not robust enough to handle more challenging images from a complex scene, such as relatively small objects in heterogenous backgrounds [128].

Different from the previous low-level methods, this chapter proposes a novel deep learning method for the object saliency detection based on the powerful CNNs. As shown in [105–107], relying on a pre-trained classification CNN, we can achieve a fairly high accuracy in object category recognition for many real-world images. Even though CNNs can recognize what kind of objects are contained in an image, it is not straightforward for them to precisely locate the recognized objects in the image. In [129–131], some rather complicated and time-consuming post-processing stages are needed to detect and locate the objects for semantic image segmentation. In [12], two CNNs are applied to generate superpixel based global saliency features and local saliency features, which should be combined for the final saliency maps.

In this work, we propose a much simpler and more computationally efficient method to generate a class-specific image saliency map directly from the classification CNN model. Specifically, we use a gradient descent method to iteratively modify each input

image based on the pixel-wise gradients to reduce a pre-defined cost function, which is defined to measure the class-specific objectness and clamp the class-irrelevant outputs to maintain image background. The gradients with respect to all image pixels can be efficiently computed using the back-propagation algorithm for CNNs. After the back-propagation procedure, the discrepancy between the modified image and the original one is calculated as the raw saliency map for this image. The raw saliency maps are smoothed by using SLIC [63] superpixel maps and refined by using low level saliency features. Since we only need to run a small number of gradient descent iterations in the saliency detection, our methods are extremely computationally efficient (average processing time for one image in one GPU is around 1.22 seconds).

Experimental results on two databases, namely Pascal VOC 2012 [48] and MSRA10k [132], have shown that the proposed methods can generate high-quality salience maps, at least comparable with many slow and complicated deep learning methods. On the other hand, comparing with the traditional low-level methods, our approach excels on many difficult images, containing complex background, highly-variable salient objects, multiple objects, and/or very small objects.

6.2 The Proposed Approach for Image Saliency Detection

This section will consider the main idea of the CNN based image saliency detection method, and also discuss how to smooth and refine the raw saliency maps for better performance.

6.2.1 Backpropagating and partially clamping CNNs to generate raw saliency maps

As we have known, CNNs can automatically learn all sorts of features from a large amount of labelled images, and a well-trained CNN can achieve a very good classification accuracy in recognizing objects in images. In this work, based on the idea of explanation vectors in [133], we argue that the classification CNNs themselves may have learned enough features and information to generate good object saliency for the images. Extending a preliminary study in [11], we explore a novel method to generate

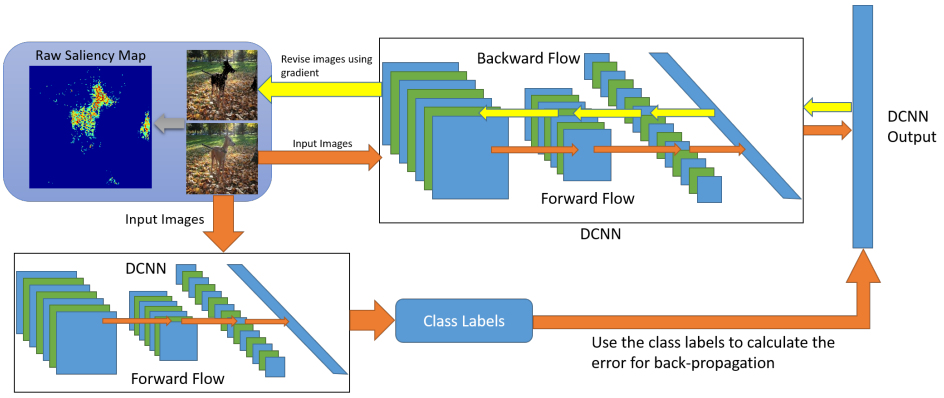

FIGURE 6.1: The proposed method to generate the object-specific saliency maps directly from CNNs.

the saliency maps directly from CNNs. The key idea of the proposed approach is shown in Figure 6.1. After an input image is recognized by a CNN as containing one particular object, if we can modify the input image in such a way that the CNN no longer recognizes the object from it and meanwhile attempts to maintain image background as much as possible, the discrepancy between the modified image and the original one may serve as a good saliency map for the recognized object. In this paper, we propose to use a gradient descent method to iteratively modify the input image based on the pixel-wise gradients to reduce a cost function formulated in the output layer of the CNN. The proposed cost function is defined to measure the class-specific objectness. The cost function is reduced under the constraint that all class-irrelevant CNN outputs are clamped to the original values, which is fundamentally different with [11], which has not try to keep the class-irrelevant output values. The image is modified by the gradients computed by applying the back-propagation procedure all the way to the input layer. In this way, the underlying object may be erased from the image while the irrelevant background may be largely retained.

First of all, we simply train a regular CNN for the image classification. After the CNN is learned, we may apply our saliency detection method to generate the class-specific object saliency map. For each input image X, we firstly use the pre-trained classification CNN to generate its class label, denoted as l, as in a normal classification step. Meanwhile, we obtain the CNN outputs prior to the final softmax layer, denoted as $\{o_k \mid k = 1, \cdots, N\}$ (N is the number of different classes). Apparently, o_l achieves the maximum value (due to the image is recognized as the class l). Here, we assume that the CNN output o_l is mainly relevant to the underlying salient objects in the image while

the remaining CNN outputs $\{o_k \mid k \neq l\}$ are more relevant to the image background excluding the underlying object.

Under this assumption, we propose a procedure to modify the image to reduce the l-th output of the CNN as much as possible and meanwhile clamp the other outputs to their original values o_k. We further denote the output nodes (prior to softmax) of the CNN in the saliency generation procedure as $\{a_i \mid i = 1, \cdots, N\}$. Therefore, for the image X, we attempt modify X to reduce the corresponding largest CNN output, i.e. a_l, subject to the constraint that all remaining CNN outputs are clamped to their initial values: $a_k = o_k (k = 1, \cdots, N \text{ and } k \neq l)$.

Next, we propose to cast the above constraints as penalty terms to construct the following cost function:

$$\mathcal{F}(X|l) = a_l + \frac{\gamma}{2} \sum_{k \not= l} (a_k - o_k)^2 \tag{6.1}$$

where γ is a hyperparameter to balance the contribution from the constraints. In this way, the original constrained optimization problem has been converted into an unconstrained problem, and the value of the objective function can be easily reduced by using SGD methods. This simple unbounded objective function works very well in practice and it results in equally good saliency maps as other more complicated bounded objective functions we have investigated.

Obviously, this cost function is constructed based on the assumption that the recognized l-th output of the CNN, i.e. a_l, corresponds to the foreground area in the input image while the remaining outputs of CNN are more relevant to the image background. Therefore, if we modify the image X to reduce the above cost function and hopefully the underlying object (belonging to class l) will be removed as the consequence due to that fact that a_l is reduced significantly, but the background remains largely unchanged due to the rest CNN outputs are clamped in this procedure. The proposed method uses an iterative gradient descent procedure to modify X as follows:

$$X^{(t+1)} \leftarrow X^{(t)} - \epsilon \cdot \max \left(\frac{\partial \mathcal{F}(X|l)}{\partial X} \Big|_{X=X^{(t)}}, 0 \right) \tag{6.2}$$

where ϵ is the learning rate, and all negative gradients are floored in the gradient descent updates. We have good rationale for doing that. As we know, the values of all image pixels are non-negative in nature. If we want to reduce the CNN output of a target class, conceptually speaking, there are two different ways: i) removing the underlying objects by cutting them out (technically subtracting positive values from image pixels); ii)

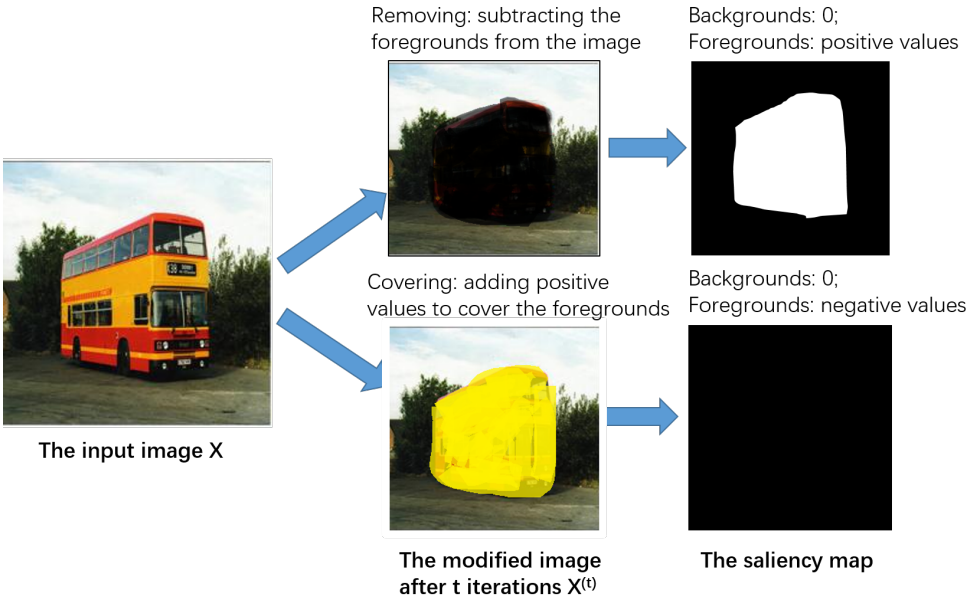

FIGURE 6.2: Up branch: removing the foregrounds from the input images results in the saliency map with positive salient objects; Down branch: covering the foregrounds of the input images results in the saliency maps with negative salient objects, which may bring about some inefficiency in the post-processing.

covering the underlying objects by smearing them (technically adding positive values to image pixels). In order to correctly localize the objects by differentiating the original image and the modified one, it is clear that using i) instead ii) to reduce the CNN output is a better choice. This is the basic reason to floor negative gradients. Moreover, after flooring all negative gradients, we ensure the final difference images do not have negative values, which significantly facilitate the postprocessing procedures (see Figure 6.2).

We have observed in experiments that the cost function $\mathcal{F}(X|l)$ can be significantly reduced by running only a small number of updates (typically 30-35 iterations) for each image, which guarantees the efficiency of the proposed method. This iterative updating procedure shows better performance than [11], which adopts linear approximation to the CNN objective function and uses the gradients to modify input images only once to generate saliency maps. As we know, CNNs are highly nonlinear and it is beneficial to use multiple gradients to iteratively modify images by taking the nonlinearity into account.

We can easily compute the above gradients using the standard back-propagation algorithm. Based on the cost function $\mathcal{F}(X|l)$ in Eq. (6.1), we can derive the error signals

in the output layer, $e_i = \frac{\partial \mathcal{F}(X|l)}{\partial a_i}$ $(i = 1, \cdots, N)$, as follows:

$$e_i = \begin{cases} \gamma(a_i - o_i) & \text{if } i \neq l, \\ 1 & \text{if } i = l. \end{cases} \tag{6.3}$$

These error signals are back-propagated all the way to the input layer to derive the above gradient, $\frac{\partial \mathcal{F}(X|l)}{\partial X}$, for saliency detection. Notice that during the above process all weights of the CNN should keep unchange.

At the end of the gradient descent updates, the raw object saliency map $\mathbf{S}$ is computed as the difference between the modified image and the original one, i.e. $X^{(0)} - X^{(T)}$. For colour images, we average the differences over the RGB channels to obtain a pixel-wise raw saliency map, which is then normalized to be of unit norm. After that, we can apply a simple threshold to filter out some weak signals (in most situations they are corresponding to background) of the raw saliency maps (see the second column in Figure 6.3).

6.2.2 SLIC based saliency map smoothing

In practice, we have found that the continuity of the above raw saliency map $\mathbf{S}$ is still not good enough in many cases. In fact, nature images include many context information, i.e., the foregrounds and backgrounds are not totally independent. However, our objective function has not take those context information into account. Therefore, some post-processing methods are needed to improve the quality of saliency maps. Roughly speaking, we have observed that most of the strong signals in the raw saliency maps are located in the saliency region. However, from Figure 6.3 we can see that some problems may still exist, such as background noises, blurred edges or small holes in the foreground. In order to further smooth the saliency maps, we use SLIC superpixels [63] to impose a continuity constraint that all image pixels located in a superpixel region always have the same saliency value. More specifically, we firstly generate the superpixel maps of all test images (In our experiments we will spilt each test image into 75 superpixels, and the compact factor is set to 10). If i-th pixel in an image belongs to the jth superpixel P_j, then the smoothed saliency value can be calculated as Eq. (6.4) shows:

$$\bar{\mathbf{S}}_i = \frac{1}{N_j} \sum_{k \in P_j} \mathbf{S}_k \quad (\forall i \in P_j) \tag{6.4}$$

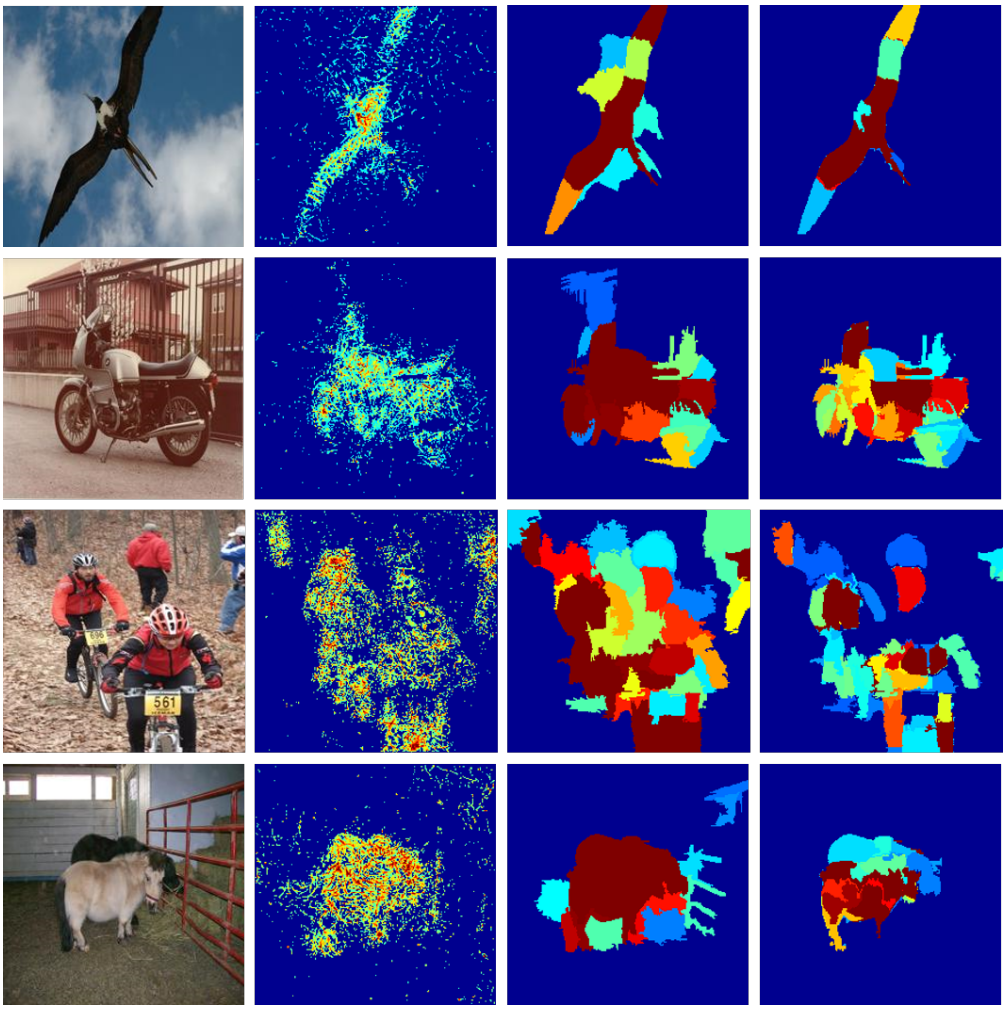

FIGURE 6.3: From left to right: original images, raw saliency maps, smoothed saliency maps and refined saliency maps

Where N_j is the number of pixels in P_j, and we use $\bar{\mathbf{S}}$ to denote the smoothed saliency maps. We can again remove some weak signals in $\bar{\mathbf{S}}$. Obviously, comparing with $\mathbf{S}$, we can see that $\bar{\mathbf{S}}$ may fill holes in the saliency regions, sharpen the object edges, and also significantly reduce the isolated background noises (see the third column in Figure 6.3).

6.2.3 Refine saliency maps using low level features

In the last part, we have generated the smoothed saliency maps, which can provide much better performance than the original raw saliency maps. On top of that, we propose to introduce some constraints based on low-level features to further improve the quality of the saliency maps.

Based on the main idea of [125], we can generate low level saliency features for each test image in very short time. Firstly, we again apply the SLIC superpixel generation method in [63] to generate superpixel maps for the test images. Next, for one superpixel P_i in an image, we calculate its color feature C_i by averaging the LAB color value over its all pixels, and use the color feature to calculate its global color contrast GC_i as:

$$GC_i = \sum_j \parallel C_i - C_j \parallel_2^2 . \tag{6.5}$$

where $\parallel \cdot \parallel_2$ denotes the Euclidean distance.

Following [125], we can further calculate the color distribution maps and smooth the global color contrast maps as the raw low-level saliency maps, which is denoted as S_L. Moreover, S_L is applied to refine the smoothed saliency map $\bar{\mathbf{S}}$ that generated from the last step. Here, we normalize S_L between α and $1 + \alpha$, where $0 < \alpha < 1$. The reason to use α is that the low level features contain a lot of errors, which may over-smooth some saliency values in the foreground of some images. By using α, we can prevent this refining procedure from removing some correct saliency regions in $\bar{\mathbf{S}}$. The refined saliency map $\hat{\mathbf{S}}$ can be generated as:

$$\hat{\mathbf{S}} = S_L \odot \bar{\mathbf{S}}. \tag{6.6}$$

where $\odot$ denotes the element-wise multiplication.

At the end, we further set some weak signals to zero in $\hat{\mathbf{S}}$ and re-normalize it (see the fourth column in Figure 6.3). The entire algorithm to generate the final saliency maps is shown in **Algorithm 4**.

6.3 Experiments

We select two benchmark databases to evaluate the performance of the proposed object saliency detection methods, namely Pascal VOC 2012 [48] and MSRA10k [132]. For Pascal VOC 2012, we use the 1449 validation images in its segmentation task as the test set, while for MSRA10k we directly use all $10,000$ images to do the test. Both databases provide the pixel-wise segmentation map (ground truth), thus we can easily measure the performances of different saliency algorithms. Notice that all images in the two databases should be re-scaled to 224-by-224 for our experiments.

Algorithm 4 CNN based Object Saliency Detection

Input: an input image X, CNN, SLIC superpixel map P, low level saliency feature S_L;
Use CNN to recognize the object label for X as l;
$X^{(0)} = X$;
for each epoch $t = 1$ **to** T **do**
  forward pass: compute the cost function $\mathcal{F}(X|l)$;
  backward pass: back-propagate to input layer to compute gradient: $\frac{\partial \mathcal{F}(X|l)}{\partial X}$;
  $X^{(t)} \leftarrow X^{(t-1)} - \epsilon \cdot \max\left(\frac{\partial \mathcal{F}(X|l)}{\partial X}, 0\right)$;
end for
Average over RGB: $\mathbf{S} = \frac{1}{3}\sum_{i=1}^{3}(X_i^{(0)} - X_i^{(T)})$;
Prune noises with a threshold θ: $\mathbf{S} = \max(\mathbf{S} - \theta, 0)$;
Normalize: $\mathbf{S} = \frac{\mathbf{S}}{\|\mathbf{S}\|}$;
Smoothing: using P to smooth $\mathbf{S}$ as $\bar{\mathbf{S}}$;
Prune noises again
Refine: $\hat{\mathbf{S}} = S_L \odot \bar{\mathbf{S}}$;
Prune noises and normalize again;
Output: the refined saliency map $\hat{\mathbf{S}}$;

Our approach will mainly compare with three exisiting methods: i) Region Contrast saliency method and the SaliencyCut segmentation method in [10]. This method is one of the most popular bottom-up image saliency detection methods in the literature and it has achieved the state-of-the-art saliency detection and segmentation performance on many tasks; ii) CNN based image saliency detection method proposed in [11]. Similar to the proposed approach, this method also uses CNNs and the back-propagation algorithm to generate saliency maps, but its objective function has not considered to clamp the outputs that corresponding to image background, and it will only run one iteration to generate saliency maps; iii) the multi-context deep learning based saliency proposed by Zhao et al. [12]. This method uses two CNNs to calculate superpixel based global context and local context respectively, and the two level contexts are further combined to generate the final multi-context saliency maps. This method is one of the state-of-the-art deep learning based image saliency algorithm.

To evaluate the performance of the saliency detection methods, two evaluation metrics are selected. The first one is precision-recall curves (PR-curves) against the ground truth. As [10], for each saliency map, we vary the cutoff threshold from 0 to 255 to generate 256 precision and recall pairs, which are used to plot a PR-curve. Besides, we also use F_β to measure the performance for saliency detection, which is calculated based on precision $Prec$ and recall Rec values with a non-negative weight parameter β

as follows [122]:

$$F_\beta = \frac{(1+\beta^2)Prec \times Rec}{\beta^2 Prec + Rec} \tag{6.7}$$

In this paper, we follow [10] to set $\beta^2 = 0.3$ to emphasize the importance of $Prec$. We may derive a sequence of F_β values along the PR-curve for each saliency map and the largest one is selected as the performance measure (see [122]).

6.3.1 Databases

Pascal VOC 2012 database [48] is a classical but still challenging image database that can be used for several vision tasks including image classification and saliency. This database currently contains 5717 training images and 5823 validation images with 20 labeled categories. However, among them, only 1449 validation images that include ground truth information are used to evaluate the performance in our image saliency tasks. Therefore, to expand the training set and improve the classification performance of the CNN, we merge the original training set with the remaining 4374 validation images without pixel-level saliency ground truth to form a new training set, which has 10197 training samples. For images that are labelled to have more than one class of objects, we use the area of the labelled objects to measure their importance, and use the class of the largest object to label the images for our CNN training process.

Unfortunately, the Pascal training set is still relatively small for CNN training. Therefore, a pre-trained CNN for the ImageNet database has been used, which contains 13 convolutional layers and 3 fully connected layers[1], as the initial network, and only use the above-mentioned training data to fine-tune this CNN with MatConvNet in [116].

In the fine-tuning process, the last layer of the pre-trained network should be replaced by a 20-nodes layer, which is corresponding to the number of classes in Pascal VOC 2012. This new output layer will be initialized randomly. Then 3 fine-tune strategies have been considered: 1) update the parameters of all hidden layers with same learning rates; 2) update all hidden layers, but only apply large learning rate for the last layer, which corresponding to the output of the CNN; 3) only update the last layer, and keep other parameters unchange. We have listed top-1 and top-5 classification error rates to measure the performance of the 3 fine-tune methods in Table 6.1. Based on the performance of the 3 methods, the method 1 will be used to do the CNN fine-tuning.

[1]We use the net *imagenet-vgg-verydeep-16* [50].

TABLE 6.1: The classification error rates of three fine-tune methods on the Pascal VOC 2012 test sets.

	METHOD1	METHOD2	METHOD3
TOP-1 ERR	16.7%	20.4%	19.1%
TOP-5 ERR	1.53%	2.08%	1.79%

Table 6.1 shows that the top-1 error on the validation set of Pascal VOC 2012 is 16.7% when using the first fine-tuning method. This classification error implies that the training sample size of Pascal VOC 2012 is still not enough for training deep convolutional networks well. However, as we will see, the proposed algorithms can still yield good performance for saliency detection. If we have more training data, we may expect even better saliency results.

MSRA10k [132] is another widely-used image saliency database, which is constructed based on Microsoft MSRA saliency database [120]. MSRA10k selects $10,000$ images from MSRA and includes pixel-wised salient objects information instead of bounding boxes, which make it suitable for our task. However, MSRA10k dose not include the corresponding training set and class labels of all images. Therefore, for MSRA10k, we directly use the CNN *imagenet-vgg-verydeep-16* [50] (without any fine-tuning) to proceed our algorithm.

6.3.2 Saliency Results

In this part we will provide saliency detection results on the selected two databases. In the following, the PR-curves, F_β values and some sample images will be used to evaluate the performance of different saliency detection methods.

6.3.2.1 Efficiency

We firstly consider the speed of the proposed saliency detection method. Here the CNN training time will not be taken into account because for all of the experiments based on one database, the CNN only needs to be trained once. We can even directly use the well-trained DCNN for ImageNet classification without any fine-tune, and the saliency results are also good. Our computing platform includes Intel Xeon E5-1650 CPU (6

TABLE 6.2: The time for processing one image of different saliency methods.

METHODS	REGION CONTRAST[10]	CNN BASED METHOD [11]	DEEP SALIENCY[12]	PROPOSED METHOD
TIME	1.92S	0.03S	4.38S	1.22S

cores), 64 GB memory and Nvidia Geforce TITAN X GPU (12 GB memory). The time consumption of processing one image of different algorithms are listed in Table 6.2.

From Table 6.2 we can learn that the proposed method yields faster processing speed than [10] and [12]. Due to the iterative updating procedure (35 epochs vs. 1 epoch) and the introducing of SLIC superpixel and low level feature, our method is slower than [11]. However, in the next part we can find that the proposed method has much better performance than [11].

6.3.2.2 The Selection of Hyperparameters

In the cost function Eq. (6.1), the variable γ is important because it can balance the contribution from the constraints. If we set $\gamma = 0$, the objective function will degenerate to a similar form as [11], and this case can be viewed as the method in [11] (run 35 epochs instead of 1 epoch) plus the proposed post-processing process. We selected different configuration of γ to show the importance of this parameter. The results are shown in Table 6.3.

TABLE 6.3: F_β values for different γ (Pascal VOC 2012, run 35 epochs, β = 0.3).

γ	0.0	1.0	5.0	10.0	20.0	30.0
F_β	0.487	0.601	0.647	0.661	0.670	0.671
γ	40.0	60.0	100.0	**200.0**	300.0	500.0
F_β	0.672	0.676	0.680	**0.685**	0.677	0.676

From the experiment results we can learn the importance of γ in the proposed objective function. Specifically, the existing of γ can significantly improve the saliency performance, and when γ less than 200, a larger γ will lead to better saliency performance. Based on the results in Table 6.3, we set $\gamma = 200$ in our rest experiments.

Another hyper-parameter we need to consider is the number of epochs for the saliency procedure. Generally speaking, a larger number of epoch may increase the performance

TABLE 6.4: The F_β value ($\beta = 0.3$) of different saliency methods on Pascal VOC 2012 and MSRA10k databases: Aim [8], MISS CB-1 [9], Region Contrast [10], CNN based Method [11], Deep Saliency [12] and our three kinds of saliency maps

	PASCAL VOC 2012	MSRA10K
AIM [8]	0.587	-
MISS CB-1 [9]	0.583	-
REGION CONTRAST [10]	0.561	0.843
CNN BASED METHOD [11]	0.347	0.357
DEEP SALIENCY [12]	0.678	**0.927**
RAW SALIENCY MAP	0.574	0.591
SMOOTHED SALIENCY MAP	0.659	0.687
REFINED SALIENCY MAP	**0.685**	0.902

but slow down the speed. Thus we hope to set a suitable number of epoch to balance the F_β and execution time. Based on the experiments, we use $T = 35$ as a good tradeoff between performance and efficiency. During the back-propagation, we use 20.0 as the initial learning rate, and the learning rate is multiplied with 0.999 after each epoch. We use 100 mini-batch size in the saliency process.

6.3.2.3 Pascal VOC 2012

To measure the performance of object saliency detection on Pascal VOC 2012, we first plot the PR-curves for different methods, which are all shown in Figure 6.4 (left). From the PR-curves, we can see that the performance of our proposed saliency detection methods significantly outperform the region contrast in [10] and the CNN based saliency method in [11]. The proposed method also yield better performance than the method in [12] (which the currently the state-of-the-art method in saliency detection).

Table 6.4 shows the F_β values of the different saliency and segmentation methods (for Pascal VOC 2012 we include two extra well-known methods AIM [8] and MISS CB-1 [9] as baselines), from which we can see that the proposed saliency detection method gives the better F_β value than [8], [9], [10], [11], and [12]. Moreover, comparing with [12], our method also yields much faster speed. Also, by comparing the F_β of our raw saliency map, smoothed saliency map and refined saliency map, we can learn the advantages introduced by our post-processing.

There are many other segmentation methods on Pascal VOC 2012 database, and many of them need to use very complicate algorithms, such as MCG [134] and GrabCut [135].

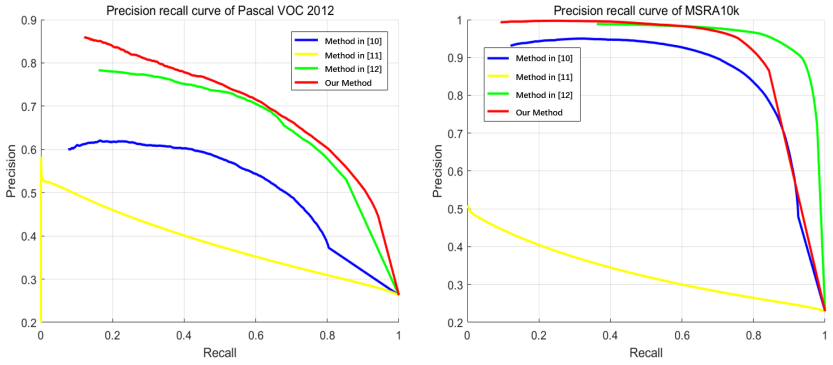

FIGURE 6.4: The PR-curves of different saliency methods on the Pascal VOC 2012 test set (Left) and MSRA10k (Right).

This paper focuses on saliency detection, not segmentation, thus we only use very simple post-processing methods and avoid to apply those relatively complicate segmentation methods. According to [9], when applying MCG to generate segmentation map, the F_β score will increase from 0.583 to 0.679 (still slightly worse than our final performance).

Finally, in Figure 6.5 (Row 1 to 4), we provide some examples of the saliency detection results from the Pascal VOC 2012 test set. From these examples we can see that the region contrast algorithm does not work well when the input images have complex background or contain highly variable salient objects, and this problem is fairly common among most bottom-up saliency and segmentation algorithms. On the other hand, we can also see that with the help of SLIC superpixels and low level features, our method can provide better performance than [12].

6.3.2.4 MSRA10k

Similarly, we also use PR-curves and F_β to evaluate the saliency and segmentation performance on MSRA10k database. From Figure 6.4(right), we can see that the proposed method is significantly better than [11], and also has slightly better performance than [10]. As shown in Table 6.4, our methods also give better F_β value than [10] and [11].

Compare with Pascal VOC 2012, we can find that our post-process methods play more important role on MSRA10k. The main reason is that the images in MSRA10k are much simpler, and the low level features work well on it.

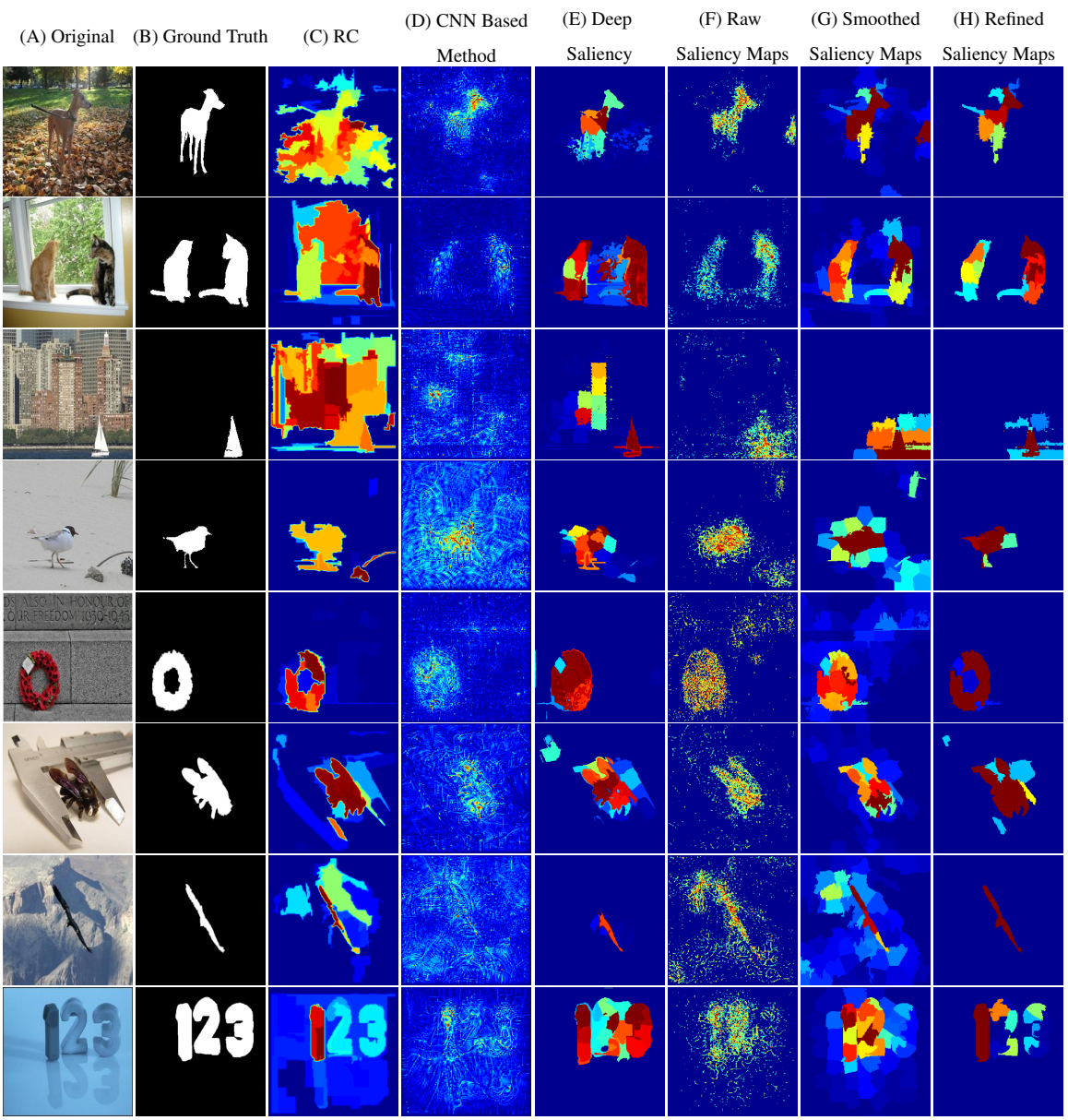

FIGURE 6.5: Saliency Results of Pascal VOC 2012 (Row 1 to 4) and MSRA10k (Row 5 to 8). (A) original images, (B) ground truth, (C) Region Contrast saliency maps [10], (D) CNN based saliency maps by using [11], (E) multi-context deep saliency method [12], (F) our raw saliency maps, (G) our smoothed saliency maps, (H) our refined saliency maps.

From Figure 6.4 and Table 7.1, we can see that our method performs slightly worse than [12] in the MSRA10k database. The main reason is attributed to that we directly use a mismatched CNN trained from the ImageNet dataset. We cannot fine-tune the model for this database due to the lack of the training set and class labels in MSRA10k. As shown in the figures, the gap between two methods is very small even though we use a mismatched CNN for our method.

In Figure 6.5, we also select some MSRA10k images to show the saliency results (Row 5 to 8).

6.4 Conclusion

This chapter has proposed a novel CNN-based method for object saliency detection. The method firstly train a regular CNN for saliency detection. After that, for each test image, we firstly recognize the image class label, and then we can use the pre-trained CNN to generate a saliency map. Specifically, we attempt to reduce a cost function defined to measure the class-specific objectness of each image, and we back-propagate the corresponding error signal all way to the input layer and use the gradient of inputs to revise the input images. After several iterations, the difference between the original input images and the revised images is calculated as a raw saliency map. The raw saliency maps are then smoothed and refined by using SLIC superpixels and low level saliency features. We have evaluated our methods on two benchmark tasks, namely Pascal VOC 2012 [48] and MSRA10k [132]. Experimental results have shown that the proposed methods can generate high-quality saliency maps in relatively short time (more than 3 times faster than the state-of-the-art CNN based method in [12]), which clearly outperforming many other existing methods. Comparing with many low-level feature methods, the proposed CNN-based approach excels on many difficult images, containing complex background, highly-variable salient objects, multiple objects, and very small objects.

Chapter 7

Supervised Adversarial Network for Image Saliency Detection

Chapter 7 discusses another deep learning based saliency detection framework called 'Supervised Adversarial Network (SAN)'. The basic idea of SAN comes from Generative Adversarial Network (GAN). By introducing some important modifications, SAN has ability to address the saliency detection problems, and can generate high quality saliency maps for some very complicated images.

7.1 Introduction

In the recent few years, Generative Adversarial Network (GAN)[21] becomes a prevalent research topic. Originally, the GAN model was designed to do image generation [21, 136, 137]. To improve the performance of traditional generative model and their unsupervised learning algorithms, GAN introduces two networks, i.e., Generator-Network (G-Network) and Discriminator-Network (D-Network), and make them 'combat' with each other to improve the performance. The G-Network tries to generate 'fake' images to cheat the D-Network, while the D-Network is trained to distinguish 'fake' images from real natural images. Experiments show that the adversarial learning process improves the network performance, and many generated images of GAN may look like natural images. Moreover, in [138], a GAN based face modification algorithm is proposed, which can modify the input faces, such as changing age, gender and expressions. To improve the face quality, [138] proposes to use extra CNNs to introduce identity-aware loss and enhance the visual quality of the generated faces. [139] argues that the

main problem of GAN is the instability during the training process, and lack of performance measurement. Therefore, it proposes a stable learning method as well as a suitable way to evaluate the quality of generated images for GAN models. Recently, some researches present that GAN also have potential to be applied in some other research fields. For instance, in [140], GAN is used in semantic segmentation, and the class specific label maps as well as a corresponding loss function are applied to adversarial training.

In this chapter, we propose a novel model, called 'Supervised Adversarial Network (SAN)', to deal with image saliency detection. This model uses the adversarial feature of GAN, but introduces some new modifications to make it work on saliency tasks. Firstly, we modify the network structure of the G-Network to make it compatible with saliency detection. The G-Network should take natural images as inputs and output the corresponding saliency maps (we call them synthetic saliency maps). Secondly, we apply a novel layer called 'conv-comparison' layer in the D-Network to force the synthetic saliency maps have some identical high-level feature as ground-truth saliency maps. Thirdly, we apply fully-supervised training to provide more precise gradient and relieve the problem of gradient vanishing. After the training, we further do post-processing such as superpixel smoothing and low-level feature refining on synthetic saliency maps to further improve the performance. The experimental results on Pascal VOC 2012 [48] show that SAN model yields good performance on saliency detection tasks, especially for those relatively complicate images.

7.2 Supervised Adversarial Networks

In most of previous methods, such as [21], [136] and [137], GAN models are designed to generate fake natural images that can 'cheat' the classification CNNs. The unsupervised learning method allows GAN model to use large amount of training data to improve the performance. Comparing with image generation, image saliency detection is a very different task, which has rigid ground-truth that can be used to measure the performance. In this section, we discuss the proposed Supervised Adversarial Networks (SAN) for image saliency detection.

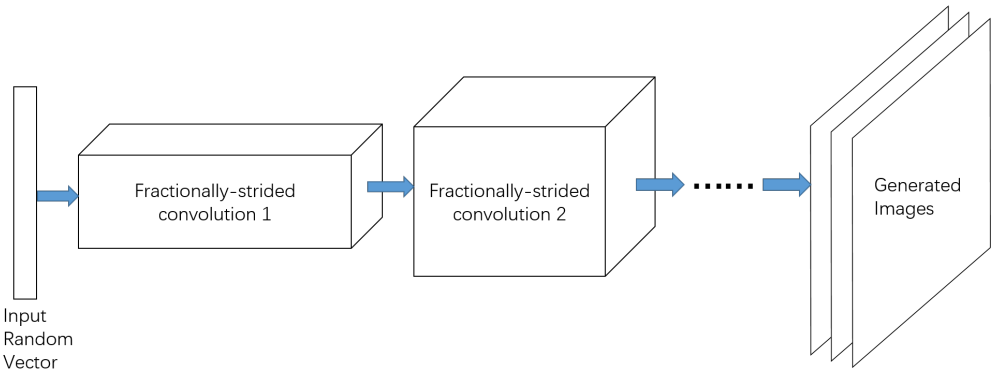

FIGURE 7.1: The basic structure of the G-Network in GAN. By using fractionally-strided convolutions, the input vector will be converted into several feature maps. The size of feature maps will gradually increase while the number will decrease. Finally the output is the fake images.

7.2.1 G-Network

Comparing with GAN model, SAN has a different structure of G-Network. In GAN, the G-Network receives random vectors as inputs, and applied several fractionally-strided convolution layers to expand the input vectors to several square feature maps, and the final outputs are fake images [141] (See Figure 7.1 for more details).

By contrast, based on the definition of image saliency detection, the G-Network in SAN requires to use natural images as input, and the output should be the corresponding saliency maps. Therefore, the G-Network in SAN should use the regular convolution layers instead of the fractionally-strided convolution layers in GAN, and remove the pooling layers to guarantee that the saliency maps have the same size as the input images (See Figure 7.2 for more details). Specifically, in the G-Networks of SAN, every hidden convolutional layer should be followed by one batch normalization layer and one regular ReLU layer. However, following the idea in [141], we do not apply batch normalization to the output layer, and the activation function of the output layer is sigmoid instead of ReLU.

According to Figure 7.2, the synthetic saliency maps of SAN can have multiple feature maps, since the experiments reflect that this configuration can increase the stability and performance of the network. When we generate multiple dimension saliency maps in practice, the corresponding ground-truth for D-Network learning should be expanded to the same number of dimensions. And finally we can take average over all dimensions to get the final saliency maps.

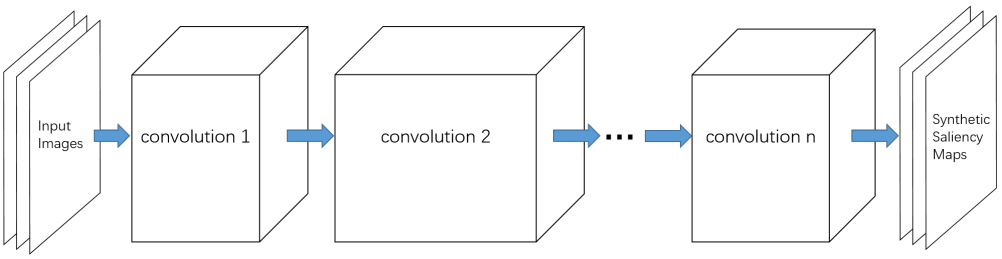

FIGURE 7.2: The basic structure of the G-Network in SAN. The feature maps' size of all layers are same, while the number of feature maps should firstly increase and then decrease.

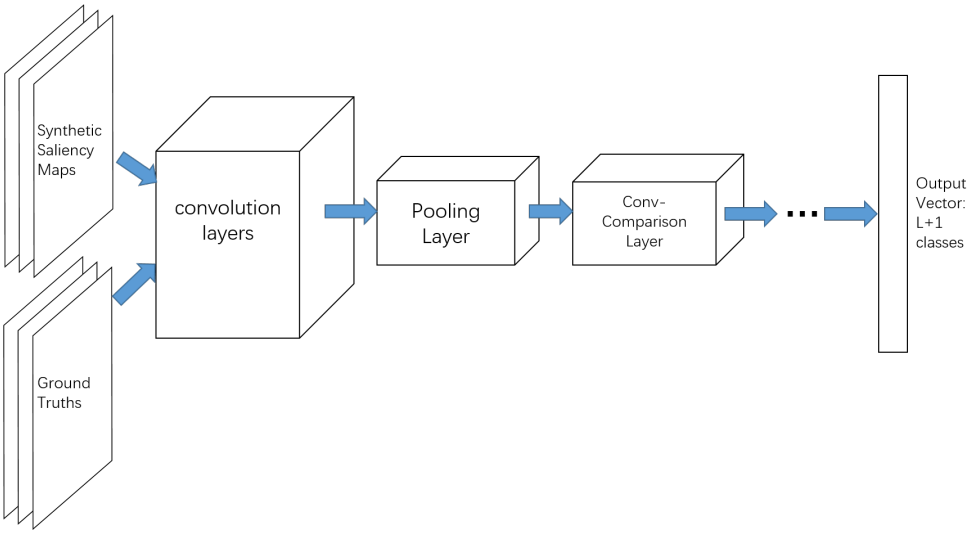

FIGURE 7.3: The basic structure of the D-Network in SAN.

7.2.2 D-Network

Generally speaking, the D-Network in SAN has similar structure as its counterpart in GAN, since both of them are designed for classification. Following the configuration in [141], the D-Network also applies batch normalization to most convolution layers, but it will use leaky ReLU [31] instead of the regular ReLU. Moreover, all pooling layers in the D-Network are replaced by convolution layers with stride 2 (without non-linear activation function).

In GAN, there are only two classes, i.e., real images or fake images. However, this may not be suitable for image saliency detection tasks. Comparing with image generation, image saliency detection has clear and definite ground-truth to measure the performance of algorithms, and as a result, to generate higher quality saliency maps, we may need more precise and class specified gradients to update both D-Network and G-Network.

Moreover, some preliminary experiments show that using two classes on saliency task will make D-Network to achieve nearly 100% classification accuracy very fast, and gradient vanishing will happen much easier when updating the G-Network. Therefore, instead of two classes, we introduce $L+1$ classes into the SAN model, where L is the class number of the training database, and the extra 1 class denotes the synthetic saliency maps. This makes SAN model can be trained under the fully supervised learning criteria.

Moreover, comparing with GAN model, SAN introduces a new kind of layer to further improve the saliency performance, i.e., conv-comparison layer. During the forward procedure, assuming that we input one ground-truth saliency map S_g and its corresponding synthetic saliency map S_s into the D-Network, then the output of the conv-comparison layer can be denoted as C_g and C_s, respectively. One very obvious consideration is: if we force C_s similar as C_g, then S_s may also tend to be similar with S_g. Here we do not directly compare S_s and S_g since the higher layer in CNNs can extract more abstract features with higher dimensions from the input data, which can provide much more rigid constraint are thus very suitable for the comparison. In the back-propagation process, the conv-comparison layer not only back-propagates the error signals from the upper layers (denote by E_u), but also calculates mean square error (MSE) between the C_g and C_s to generate another error signal:

$$E_c = \frac{1}{2}||C_s - C_g||^2 \tag{7.1}$$

Then the back-propagated gradient with respect to the output of the conv-comparison (i.e. C_s) layer can be calculated as:

$$\frac{\partial E}{\partial C_s} = (1-\alpha)\frac{\partial E_u}{\partial C_s} + \alpha\frac{\partial E_c}{\partial C_s} \tag{7.2}$$

where α is used to balance the importance of the two errors. Notice that we may need to normalize the gradient of E_c to make it have the same scale as the gradient of E_u. Then the new error signal can be used to update the weights of the conv-comparison layer. In practice, we may apply more than one conv-comparison layers in the D-Network to provide stronger constraint and then further improve performance (See Figure 7.3 for more details).

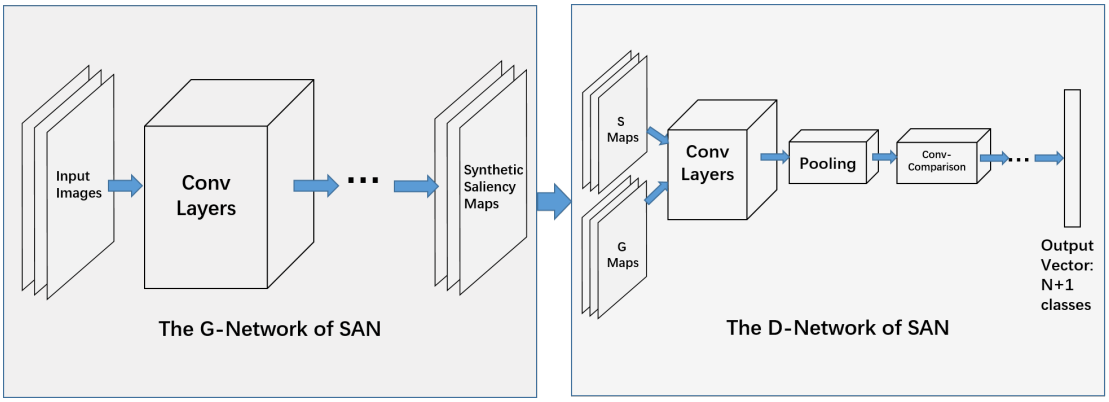

FIGURE 7.4: The whole SAN model that includes the G-Network and D-Network.

7.2.3 Model Training

At beginning, all model parameters in SAN should be initialized randomly using the initialization method in [30]. Then the training process of SAN can be divided into three parts:

(1) saliency maps generation: In this step, the G-Network works like a pure feed-forward network, which receives training images as inputs and outputs the corresponding synthetic saliency maps. The set of all synthetic saliency maps is denoted by $\mathbf{S}$.

(2) updating D-Network: In this step, we will use $\mathbf{S}$ as well as the ground-truth saliency maps of the training set (denote by $\mathbf{G}$) as the training data of the D-Network. The data in $\mathbf{G}$ are labelled by using the labels of their corresponding training images (from 1 to L), while all elements in $\mathbf{S}$ will be labelled as $L+1$, which denotes synthetic saliency maps. By using the pairs of training data and labels, the D-Network can be trained supervisingly via regular error back-propagation algorithm. The well-trained D-Network will achieve a good balance between classification accuracy and providing large enough gradients when updating the G-Network.

(3) updating G-Network: After get the well-trained D-Network, we update the G-Network to make it capable to generate better synthetic saliency maps that can 'deceive' the D-Network. This step also can be done by using supervised learning. Specifically, we firstly concatenate the G-Network and D-Network (denote by GD-Network, see Figure 7.4 for more details) and use the training images set I as input. During this step all images will be labelled using their original class labels (i.e., from 1 to L). In the forward process, the input signals will firstly pass the G-Network to generate synthetic saliency maps, then the generated saliency maps will pass the D-Network to the output layer.

In the backward phase, we fix the weights of D-Network and only update G-Network. During the learning process, the labelling method forces all synthetic saliency maps belong to the corresponding ground-truthes classes. By only updating the G-Network, the distribution of the generated synthetic saliency maps may approach the ground-truthes. In this way, the G-Network may tend to generate higher quality synthetic saliency maps that can make the D-Network recognize them as ground-truth saliency maps.

7.2.4 Post-Processing

After the network training, we can use the G-Network to generate raw saliency maps for validation images. After that, we may use some simple post-processing methods to further improve the quality of raw saliency maps. Similar with Chapter 6, we firstly filter out some weak signals since it is very possible that those signals are corresponding to background regions. After that, we introduce SLIC superpixels [63] to smooth the raw saliency maps. This operation can weaken some background noises and sharpen the edges of foreground objects. We can further use low-level saliency features mentioned in [125] to refine the smoothed saliency maps to remove some incorrect saliency regions. Finally the refined saliency maps will be normalized, and we will again filter out weak signals. Figure 7.5 shows some examples of post-processing. Even though the raw saliency maps of SAN have better quality compare with the saliency method in Chapter 6, the post-processing method still can improve the performance by removing background noises and filling some holes in foreground.

7.3 Experiments

We test the presented SAN model on a well-known computer vision database, i.e. Pascal VOC 2012 [48], and compare it with several other saliency algorithms. To measure the performance of the selected methods, we use F_β value as the measurements as [10] mentioned. Notice that in our experiments β is set to 0.3 to emphasize the importance of precision.

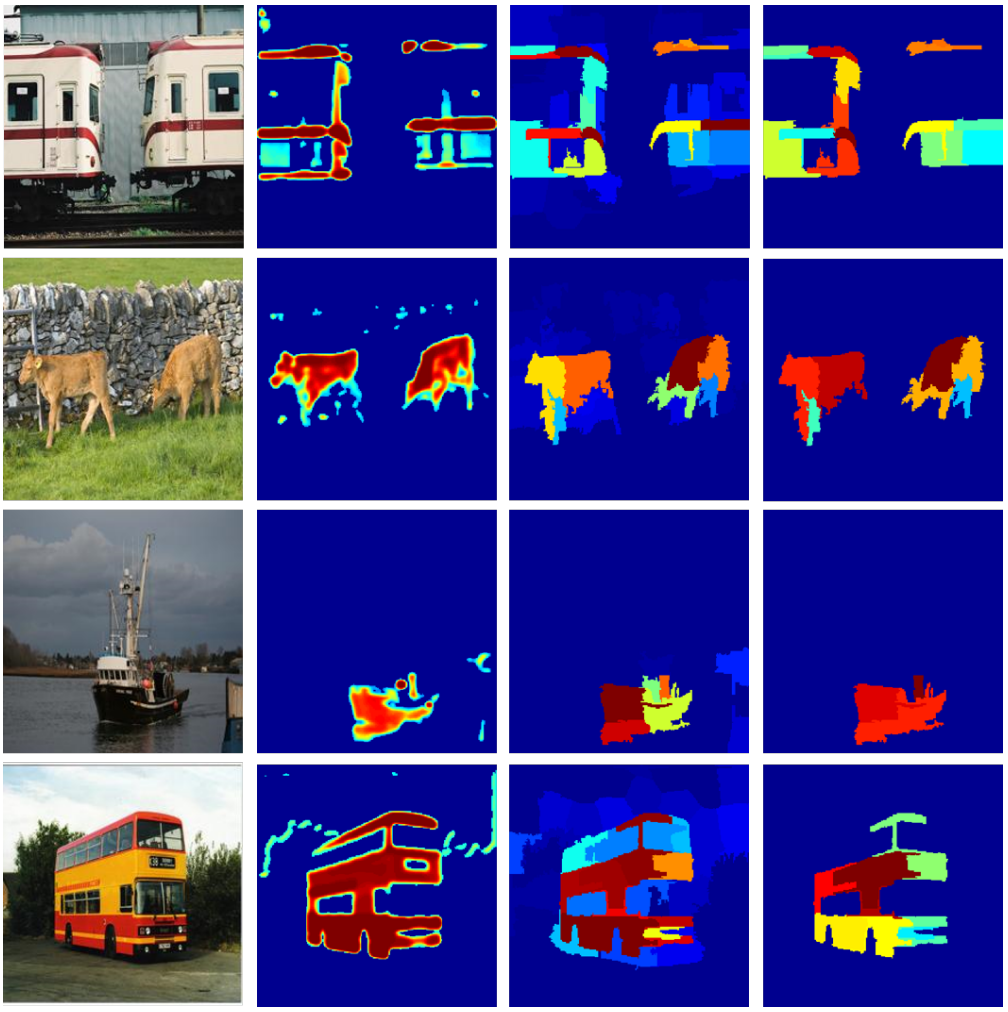

FIGURE 7.5: From left to right: original images, raw saliency maps, smoothed saliency maps and refined saliency maps

7.3.1 Database

In image saliency detection and semantic segmentation, Pascal VOC 2012 database [48] is a classical and also challenging image database. For the saliency and segmentation tasks, this database provides 1464 training data and 1449 validation data with their pixel-wised segmentation ground-truthes as well as class labels of all foreground objects. Therefore, Pascal VOC 2012 is suitable for training SAN model and evaluate the algorithm performance.

7.3.2 Baseline Methods

Firstly, we design three baselines to compare with SAN model and demonstrate the advantages of our configuration. The first one (denote by Baseline 1) is used to reflect the benefits of adversarial learning. Specifically, we remove the D-Network from our model and simply train a G-Network to generate saliency maps. During the training procedure, we firstly initialize the G-Network randomly, and for each training image, we calculate the mean square error (MSE) between the synthetic saliency map and ground-truth to get gradient and update the network. This baseline test will show the saliency performance without adversarial training. The second test (denote by Baseline 2) shares the unsupervised learning criteria with GAN model. Specifically, in this baseline we do not take the class labels of training dataset into account. Instead, similar with GAN, we simply consider two classes, i.e., synthetic saliency maps and ground-truth saliency maps, and calculate error signal to update both the D-Network and G-Network based on them. This method will show the importance of the supervised training method introduced in SAN. The Baseline 3 shares most of configurations with SAN, and the only difference is that we use regular convolution layers to instead all conv-comparison layers in its D-Network. This baseline can reflect the positive influences brought by conv-comparison layers.

Besides the three baselines, we also select two state-of-the-art third-party saliency detection algorithms to compare with the proposed SAN model. The first one is a bottom-up saliency method called Region Contrast (RC) [10]. This method considers the global contrast of each superpixel region and introduces spatial constraint to generate saliency maps. The second one is a deep learning based saliency method called multi-context deep saliency, which is proposed by Zhao et al. [12]. This method introduces two CNNs to model both global and local contexts for each superpixel and generates high quality saliency maps based on those contexts information.

7.3.3 Saliency Results

In this part we will provide saliency detection results on Pascal VOC 2012. In the following, we use F_β values and some sample images to evaluate the performance of the proposed SAN model and the other selected baselines. Our computing platform includes Intel Xeon E5-1650 CPU (6 cores), 64 GB memory and Nvidia Geforce TITAN

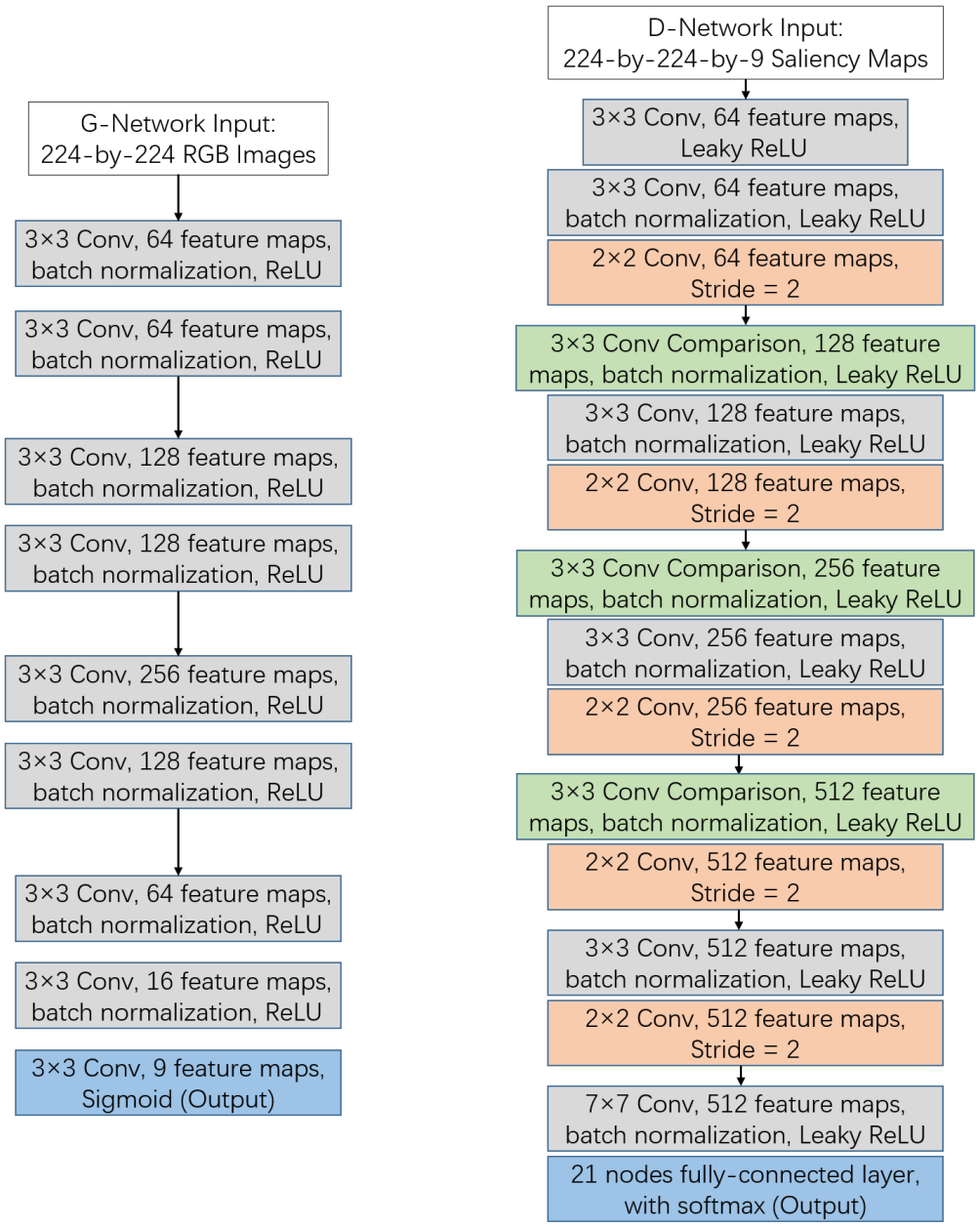

FIGURE 7.6: The configurations of the G-Network and D-Network in our experiments.

X GPU (12 GB memory). Our algorithms are implemented on MatConvNet platform [116], which is a matlab and CUDA based deep learning toolkit.

7.3.3.1 The Selection of Hyperparameters

In our experiments, the synthetic saliency maps and corresponding ground-truthes have 9 dimensions, and we take average over the 9 dimensions of generated saliency maps to get the final results for evaluation. Since Pascal VOC 2012 has 20 pre-defined classes,

thus the output layer of the D-Network needs 21 node to denote all pre-defined classes and one extra class of synthetic saliency maps. In our implementation, the G-Network of SAN has nine convolution layers, while the D-Network has 15 convolution layers (3 of them are defined as conv-comparison layers) and 1 fully-connected layer (See Figure 7.6 for more details).

During the learning, we run the training algorithm for 20 iterations. In each iteration, we firstly update the D-Network for 6 epochs, and then update the G-Network for 2 epochs. For the D-Network training, we use 16 mini-batch size. The initial learning rate is 0.0006 and it needs to multiply with 0.98 after every epoch. For the G-Network, we use SGD to do training. The initial learning rate is 0.0001 and the decay rate is also 0.98. We do not use momentum and weight decay in the training process. For the conv-comparison layers in the D-Network, we set $\alpha = 0.8$ to emphasize the gradient of E_c.

7.3.3.2 Performance

Table 7.1 shows the F_β values of all selected saliency detection algorithms. Comparing with Baseline 1, Baseline 2 and Baseline 3, we can learn that the adversarial learning, fully-supervised training and conv-comparison layers bring about a lot of advantages to saliency results. Moreover, SAN provides better performance than the bottom-up method in [10]. Comparing with the CNN based saliency methods in [12], the proposed method can also provide slightly better performance.

TABLE 7.1: The F_β value of different saliency methods on Pascal VOC 2012 ($\beta = 0.3$).

METHODS	F_β
RC [10]	0.561
DEEP SALIENCY [12]	0.678
BASELINE 1	0.403
BASELINE 2	0.382
BASELINE 3	0.522
SAN WITHOUT POST-PROCESSING	0.613
SAN	**0.681**

Finally, in Figure 7.7, we provide some examples of the saliency detection results from the Pascal VOC 2012 validation set. From these examples we can see that SAN has ability to deal with the image saliency detection tasks in variety of complicate images.

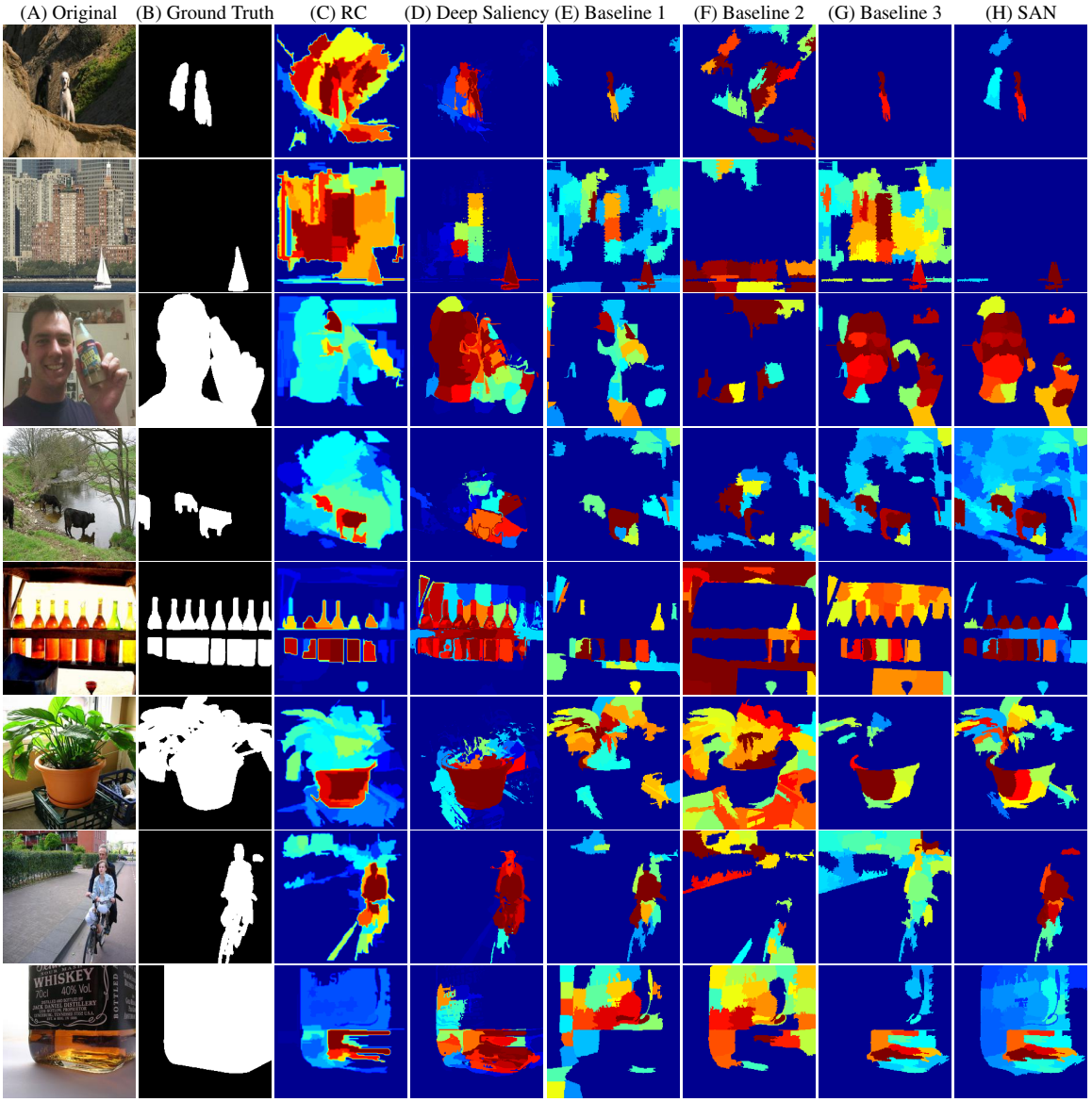

FIGURE 7.7: Saliency Results of Pascal VOC 2012. (A) original images, (B) ground truth, (C) Region Contrast saliency maps [10], (D) multi-context deep saliency method [12], (E) Baseline 1, (F) Baseline 2, (G) Baseline 3. (H) the proposed SAN model.

7.4 Conclusion

In this chapter, we have proposed a novel Supervised Adversarial Network (SAN) for image saliency detection. The method relies on the well-known Generative Adversarial Network (GAN) [21], and introduces many modifications to make the model suitable for saliency detection tasks. Specifically, we define a fully-convolution G-Network, which takes images as inputs and outputs corresponding synthetic saliency maps. After that, the synthetic saliency maps and ground-truth saliency maps are used to train the

D-Network. We divide the synthetic saliency maps and ground-truthes into $L+1$ classes based on the class labels of the training set, which allows us to train both G-Network and D-Network supervisingly. Moreover, we introduce a novel kind of layer called conv-comparison layer into the D-Network to introduce more constraints to improve the quality of saliency maps. The synthetic saliency maps will be smoothed and refined using SLIC superpixels [63] and low level saliency features [125]. We have evaluated the performance of the proposed method on Pascal VOC 2012 [48] database. Experimental results have shown that the introducing of adversarial learning, fully-supervised training and conv-comparison layers can significantly improve the saliency results. Moreover, comparing with the state-of-the-art saliency detection algorithms in [10] and [12], our proposed method can generate better saliency maps, and also has ability to deal with difficult images.

Chapter 8

Conclusion and Future Works

8.1 Conclusion

This dissertation has proposed several novel works about deep learning, which cover training algorithm, new model and applications.

Chapter 2 and 3 have reviewed some basic knowledge about artificial neural networks and one of their important applications, i.e., image processing. After that, Chapter 4 has presented a novel training method for DNNs called annealed gradient descent (AGD). Since the optimization of DNNs is a kind of highly non-convex problem, the bad quality local optima and saddle points always hinder the optimization algorithms from finding good results. SGD algorithm may avoid to trap into some bad local optima or saddle points by introducing noisy gradients, but it can also slow down the learning speed. Moreover, the sequential nature makes SGD very hard to be parallelized. Previous researchers have proposed some parallelized SGD methods, however, those methods can only handle convex or sparse problems. Thus applying them on DNNs training may not result in good performance. By investigating the nature of non-convex objective functions of DNNs, a group of mosaic risk functions are defined in the proposed AGD training algorithm to approximate the original objective functions to accelerate the learning speed. To get the mosaic risk functions, the K-means clustering algorithm is firstly used to build hierarchical codebooks for the training set, then the mosaic risk functions with different quantization error can be derived. The mosaic risk functions are much smoother than the original non-convex objective functions, and most low quality local optima can be hidden. Therefore, it is possible to apply full-batch gradient descent or mini-batch SGD with large batch size to optimize them (the optimization process is

called MGD), and it has been proved that using MGD results in good initializations for the SGD optimization on the original objective functions. The experimental results on MNIST (hand-written digits database) and Swithchboard (320-hour English speech transcription database) have shown that the AGD algorithm results in better classification performance on test sets, and has much faster learning speed compare with the traditional SGD algorithm.

Chapter 5 has proposed to apply a novel model called Hybrid Orthogonal Projection and Estimation (HOPE) on CNNs. HOPE model was firstly proposed to work with fully connected DNNs. This model is based on an assumption that DNNs can gradually remove the correlation of the input data, and it defines one orthogonal projection layer and one model layer to replace one DNN layer to explicitly reduce the data correlation. Experiments on DNNs have already shown that HOPE can improve the classification performance through supervised or unsupervised learning methods. To extend the HOPE model to CNNs, Chapter 5 proposed a suitable way to transform the convolution procedure to matrix multiplication and defined forward and backward method for the HOPE CNN layers. HOPE layers can be used right after the input layer, or to replace convolution layers and pooling layers. Moreover, this dissertation proposed several way to apply HOPE into CNNs, i.e., HOPE-Input, single-HOPE-Block and multi-HOPE-Blocks. Experimental results reflect that comparing with different baselines, single-HOPE-Block can result in the best performance, which is state-of-the-art on CIFAR-10, CIFAR-100 (with or without data augmentation) and ImageNet databases.

Chapter 6 has discussed one important application of deep learning, i.e., image saliency detection. Currently most deep learning based saliency methods have much better performance compared with traditional bottom-up saliency methods, and one possible reason is that deep structures have the ability to extract high level features from the input images. Previous deep learning based saliency methods always have complicated network structure, training methods or post-processing procedure, which result in low efficiency. To solve this problem, this dissertation proposed a CNN based fast saliency method. Specifically, the proposed method modified the objective function of the CNNs to make them can measure the class-specific objectness and clamp the class-irrelevant outputs. By executing back-propagation to modify the input images for several epochs, the algorithm can remove the foreground objects from the images but keep the background regions unchanged. And as a result, the difference between the input images and the modified images can be viewed as raw saliency maps. To further improve performance, SLIC superpixels were used to smooth the raw saliency maps, and the

smoothed saliency maps were further refined by using low-level saliency features. The experiments show that the proposed method works well on both Pascal VOC 2012 and MSRA10k databases, and the speed is much faster than the previous state-of-the-art methods.

Chapter 7 proposed supervised adversarial network (SAN) to perform image saliency detection, which uses the basic idea of GAN to calculate saliency maps of the input images. GAN was originally designed for image generation. Therefore, we introduced some modification to make it suitable for saliency detection tasks. Firstly, we applied a different network structure of the G-Network to make it receive natural images as inputs and output the corresponding saliency maps. Secondly, we introduced a fully supervised training process to improve the saliency performance. Thirdly, a novel layer called 'conv-comparison layer' was applied in the D-Network to force the synthetic saliency maps have similar features as ground-truth saliency maps. By using the above modifications, SAN can generate high-quality raw saliency maps for complicated natural images. After that, we applied the same post-processing method as Chapter 6 to further improve the performance. Experimental results on Pascal VOC 2012 show that SAN can also provide state-of-the-art saliency performance.

8.2 Future Work

This dissertation has proposed several works related to deep learning. Besides the methods mentioned above, there are still some related ideas that have potential to further improve the performance.

(1) Applying AGD algorithm on CNN training: Comparing with hand-written digits and speech signals, natural images are more complicated. The objects that belong to the same class may have diversity of shape, scale, color and texture. Therefore, simply using K-means to build hierarchical codebooks may not suitable for many image databases like CIFAR-10, CIFAR-100 and ImageNet. One possible way to solve this problem is to extract small patches from the original training images and use the patches to build codebooks. Then the codewords will be used to re-construct the full-sized codebook images. After that, the re-constructed codebook images can be used to do MGD optimization. This method may have better performance than directly cluster on the original training images and has potential to speed up the CNN learning.

(2) Unsupervised HOPE model for CNNs: the proposed HOPE CNN in this dissertation has implemented the HOPE model on supervised training CNNs and achieves state-of-the-art classification performance. However, the original edition of HOPE that works on DNNs also provided unsupervised and semi-supervised learning frameworks. Those frameworks are especially important when the labeled training data are not enough. Thus in the next step, one significant research topic is to build unsupervised learning framework for HOPE CNNs. Some preliminaries researches reflect that it is much harder to use GPU to speed up unsupervised CNN learning than supervised learning. Thus the efficiency also needs some consideration.

(3) Applying HOPE to more deep learning frameworks: besides fully-connected DNNs and CNNs, there are still variety of deep learning frameworks that work well on different applications. For instance, deep residual networks [51] can provide state-of-the-art classification accuracy on ImageNet database; recurrent neural networks (RNNs) are significantly indispensable in natural language processing. Combining HOPE layers with those models may also result in even better performance.

(4) Updating the CNN based saliency method for object detection: the CNN based saliency method in Chapter 6 provides excellent performance. However, even though it can find foreground objects from input images effectively and efficiently, it cannot output the class labels of the objects in the situation of multi-class objects. One possible way to update the method is to endow it with ability to generate class label for every foreground object. If so, this method will become an object detection algorithm.

(5) More applications of GAN model: in addition to image related tasks like image generation or image saliency detection, there are still some other potential applications for the GAN model. For example, GAN may also be used to generate 'fake' speech signals and make them sounds like human's voice. Moreover, GAN may even be used to generate videos. Those ideas may result in various interesting applications.

Publications

1. Hengyue Pan, and Hui Jiang. ”Annealed gradient descent for deep learning.” Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence. AUAI Press, 2015.

2. Hengyue Pan, Bo Wang, and Hui Jiang. ”Deep learning for object saliency detection and image segmentation.” arXiv preprint arXiv:1505.01173 (2015).

3. Hengyue Pan, and Hui Jiang. ”A Deep Learning Based Fast Image Saliency Detection Algorithm.” arXiv preprint arXiv:1602.00577 (2016).

4. Hengyue Pan, and Hui Jiang. ”A Fast Method for Saliency Detection by Back-Propagating A Convolutional Neural Network and Clamping Its Partial Outputs.” 2017 International Joint Conference on Neural Networks (IJCNN). IEEE, 2017.

5. Hengyue Pan, and Hui Jiang. ”Learning Convolutional Neural Networks using Hybrid Orthogonal Projection and Estimation.” arXiv preprint arXiv:1606.05929 (2016). Submitted to 2017 Asian Conference on Machine Learning (ACML).

6. Hengyue Pan, and Hui Jiang. ”Supervised Adversarial Networks for Image Saliency Detection.” arXiv preprint arXiv:1704.07242 (2017). Submitted to 2017 Asian Conference on Machine Learning (ACML).

Bibliography

[1] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8609–8613. IEEE, 2013.

[2] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep big simple neural nets excel on handwritten digit recognition. *Neural Computation*, 22(12):3207–3220, 2010.

[3] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In *Advances in Neural Information Processing Systems*, pages 2553–2561, 2013.

[4] Pedro Pinheiro and Ronan Collobert. Recurrent convolutional neural networks for scene labeling. In *Proceedings of The 31st International Conference on Machine Learning*, pages 82–90, 2014.

[5] Michael L Seltzer, Dong Yu, and Yongqiang Wang. An investigation of deep neural networks for noise robust speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7398–7402. IEEE, 2013.

[6] Pan Zhou, Lirong Dai, Hui Jiang, Yu Hu, and Qingfeng Liu. A state-clustering based multiple deep neural networks modelling approach for speech recognition. In *IEEE International Conference of Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

[7] Ossama Abdel-Hamid, A-r Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Trans. on Audio, Speech and Language Processing*, 22(10):1533–1545, 2014.

[8] Neil Bruce and John Tsotsos. Saliency based on information maximization. In *Advances in neural information processing systems*, pages 155–162, 2005.

[9] Shafin Rahman and Neil Bruce. Saliency, scale and information: Towards a unifying theory. In *Advances in Neural Information Processing Systems*, pages 2179–2187, 2015.

[10] Ming-Ming Cheng, Niloy J. Mitra, Xiaolei Huang, Philip H. S. Torr, and Shi-Min Hu. Global contrast based salient region detection. In *Computer Vision and Pattern Recognition (CVPR)*, pages 409–416. IEEE, 2011.

[11] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *arXiv preprint arXiv:1312.6034*. 2014.

[12] Rui Zhao, Wanli Ouyang, Hongsheng Li, and Xiaogang Wang. Saliency detection by multi-context deep learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1265–1274, 2015.

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

[14] Marvin Minsky and Seymour Papert. Perceptrons. 1969.

[15] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[16] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[17] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

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

[19] Yann LeCun, Boser B., JS Denker, D Henderson, Richard E. Howard, W. Hubbard, and Lawrence D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*. Citeseer, 1990.

[20] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361, 1995.

[21] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[22] Shiliang Zhang, Hui Jiang, and Lirong Dai. Hybrid orthogonal projection and estimation (HOPE): A new framework to learn neural networks. *Journal of Machine Learning Research*, 17(37):1–33, 2016. URL `http://jmlr.org/papers/v17/15-335.html`.

[23] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

[24] 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.

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

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

[27] 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.

[28] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *The Journal of Machine Learning Research*, 10:1–40, 2009.

[29] Shiliang Zhang, Hui Jiang, Si Wei, and Li-Rong Dai. Rectified linear neural networks with tied-scalar regularization for lvcsr. In *INTERSPEECH*, pages 2635–2639, 2015.

[30] 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.

[31] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

[32] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, 11:625–660, 2010.

[33] Nitish Srivastava. *Improving neural networks with dropout*. PhD thesis, University of Toronto, 2013.

[34] Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013.

[35] Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *Advances in Neural Information Processing Systems*, pages 3084–3092, 2013.

[36] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE, 2009.

[37] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118, 2010.

[38] Dan Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

[39] Dan Cireşan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32: 333–338, 2012.

[40] 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.

[41] Jorge Sánchez and Florent Perronnin. High-dimensional signature compression for large-scale image classification. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1665–1672. IEEE, 2011.

[42] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[43] Chen-Yu Lee, Patrick W Gallagher, and Zhuowen Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *International conference on artificial intelligence and statistics*, 2016.

[44] Stephen Gould, Richard Fulton, and Daphne Koller. Decomposing a scene into geometric and semantically consistent regions. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1–8. IEEE, 2009.

[45] Victor Lempitsky, Andrea Vedaldi, and Andrew Zisserman. Pylon model for semantic segmentation. In *Advances in neural information processing systems*, pages 1485–1493, 2011.

[46] Pan Zhou, Lirong Dai, and Hui Jiang. Sequence training of multiple deep neural networks for better performance and faster training speed. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 5627–5631. IEEE, 2014.

[47] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, and Gerald Penn. Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4277–4280. IEEE, 2012.

[48] 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.

[49] Agnan Kessy, Alex Lewin, and Korbinian Strimmer. Optimal whitening and decorrelation. *arXiv preprint arXiv:1512.00809*, 2015.

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

[51] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[52] Sergey Zagoruyko. 92.45% on cifar-10 in torch. *Torch Blog*, 2015.

[53] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset, 2014.

[54] Ren Wu, Shengen Yan, Yi Shan, Qingqing Dang, and Gang Sun. Deep image: Scaling up image recognition. *arXiv preprint arXiv:1501.02876*, 7(8), 2015.

[55] Xiaofeng Ren. Superpixel: Empirical studies and applications.

[56] Shu Wang, Huchuan Lu, Fan Yang, and Ming-Hsuan Yang. Superpixel tracking. In *2011 International Conference on Computer Vision*, pages 1323–1330. IEEE, 2011.

[57] Hengyue Pan and Hui Jiang. A deep learning based fast image saliency detection algorithm. *arXiv preprint arXiv:1602.00577*, 2016.

[58] Brian Fulkerson, Andrea Vedaldi, Stefano Soatto, et al. Class segmentation and object localization with superpixel neighborhoods. In *ICCV*, volume 9, pages 670–677. Citeseer, 2009.

[59] Xiaofeng Ren and Jitendra Malik. Learning a classification model for segmentation. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 10–17. IEEE, 2003.

[60] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[61] Ming-Yu Liu, Oncel Tuzel, Srikumar Ramalingam, and Rama Chellappa. Entropy rate superpixel segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2097–2104. IEEE, 2011.

[62] Michael Van den Bergh, Xavier Boix, Gemma Roig, Benjamin de Capitani, and Luc Van Gool. Seeds: Superpixels extracted via energy-driven sampling. In *European conference on computer vision*, pages 13–26. Springer, 2012.

[63] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Susstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34 (11):2274–2282, 2012.

[64] Walter Schneider and Richard M Shiffrin. Controlled and automatic human information processing: I. detection, search, and attention. *Psychological review*, 84(1):1, 1977.

[65] Simone Frintrop, Erich Rome, and Henrik I Christensen. Computational visual attention systems and their cognitive foundations: A survey. *ACM Transactions on Applied Perception (TAP)*, 7(1):6, 2010.

[66] Laurent Itti, Christof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 20(11):1254–1259, 1998.

[67] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[68] 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.

[69] 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.

[70] 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.

[71] Joao Carreira, Rui Caseiro, Jorge Batista, and Cristian Sminchisescu. Semantic segmentation with second-order pooling. In *European Conference on Computer Vision*, pages 430–443. Springer, 2012.

[72] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015.

[73] 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.

[74] Léon Bottou. Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9):142, 1998.

[75] Léon Bottou and Yann LeCun. Large scale online learning. In *Advances in neural information processing systems 17 (NIPS'04)*, page 217, 2004.

[76] Nicolas L Roux, Mark Schmidt, and Francis R Bach. A stochastic gradient method with an exponential convergence _rate for finite training sets. In *Advances in Neural Information Processing Systems*, pages 2663–2671, 2012.

[77] Ohad Shamir and Tong Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 71–79, 2013.

[78] Noboru Murata and Shun-ichi Amari. Statistical analysis of learning dynamics. *Signal Processing*, 74(1):3–28, 1999.

[79] Huasha Zhao and John F Canny. Communication-efficient distributed stochastic gradient descent with butterfly mixing. *University of California, Berkeley*, 2012.

[80] John Langford, Alex J Smola, and Martin Zinkevich. Slow learners are fast. *Advances in Neural Information Processing Systems 22 (NIPS'09)*, pages 2331–2339, 2009.

[81] Christian Bockermann and Sangkyun Lee. Scalable stochastic gradient descent with improved confidence. In *NIPS Workshop on Big Learning–Algorithms, Systems, and Tools for Learning at Scale*, 2012.

[82] Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems*, pages 873–881, 2011.

[83] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing System 24 (NIPS'11)*, pages 693–701, 2011.

[84] Jeffrey L Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993.

[85] Kai A Krueger and Peter Dayan. Flexible shaping: How learning in small steps helps. *Cognition*, 110(3):380–394, 2009.

[86] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[87] Vladimir Naumovich Vapnik. *Statistical Learning Theory*. John Wiley and Sons, 1st edition, 1998.

[88] Léon Bottou. Stochastic learning. In *Advanced lectures on machine learning*, pages 146–168. Springer, 2004.

[89] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

[90] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[91] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, pages 315–323, 2013.

[92] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems 23 (NIPS'10)*, pages 2595–2603, 2010.

[93] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

[94] Thomas Paine, Hailin Jin, Jianchao Yang, Zhe Lin, and Thomas Huang. Gpu asynchronous stochastic gradient descent to speed up neural network training. *arXiv preprint arXiv:1312.6186*, 2013.

[95] Shanshan Zhang, Ce Zhang, Zhao You, Rong Zheng, and Bo Xu. Asynchronous stochastic gradient descent for dnn training. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6660–6663. IEEE, 2013.

[96] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *AISTATS*, 2015.

[97] Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems*, pages 2933–2941, 2014.

[98] Léon Bottou. Stochastic gradient tricks. *Neural networks: tricks of the trade. Springer, Berlin*, pages 430–445, 2012.

[99] 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.

[100] Frank Seide, Gang Li, and Dong Yu. Conversational speech transcription using context-dependent deep neural networks. In *Interspeech*, pages 437–440, 2011.

[101] Jia Pan, Cong Liu, Zhiguo Wang, Yu Hu, and Hui Jiang. Investigation of deep neural networks (dnn) for large vocabulary continuous speech recognition: Why dnn surpasses gmms in acoustic modeling. In *Chinese Spoken Language Processing (ISCSLP), 2012 8th International Symposium on*, pages 301–305. IEEE, 2012.

[102] Yebo Bao, Hui Jiang, Cong Liu, Yu Hu, and Lirong Dai. Investigation on dimensionality reduction of concatenated features with deep neural network for lvcsr systems. In *Signal Processing (ICSP), 2012 IEEE 11th International Conference on*, volume 1, pages 562–566. IEEE, 2012.

[103] Frank Seide, Gang Li, Xie Chen, and Dong Yu. Feature engineering in context-dependent deep neural networks for conversational speech transcription. In *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, pages 24–29. IEEE, 2011.

[104] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29 (6):82–97, 2012.

[105] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

[106] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *arXiv preprint arXiv:1409.4842*. 2014.

[107] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*. 2015.

[108] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[109] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[110] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013.

[111] Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1319–1327, 2013.

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

[113] Ming Liang and Xiaolin Hu. Recurrent convolutional neural network for object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3367–3375, 2015.

[114] Oren Rippel, Jasper Snoek, and Ryan P Adams. Spectral representations for convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 2440–2448, 2015.

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

[116] A. Vedaldi and K. Lenc. Matconvnet – convolutional neural networks for matlab. *CoRR*, abs/1412.4564, 2014.

[117] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mostofa Ali, Ryan P Adams, et al. Scalable bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, 2015.

[118] Dmytro Mishkin and Jiri Matas. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015.

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

[120] Tie Liu, Zejian Yuan, Jian Sun, Jingdong Wang, Nanning Zheng, Xiaoou Tang, and Heung-Yeung Shum. Learning to detect a salient object. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 33(2):353–367, 2011.

[121] Jianming Zhang and Stan Sclaroff. Saliency detection: A boolean map approach. In *ICCV*. 2013.

[122] Ali Borji, Dicky N Sihite, and Laurent Itti. Salient object detection: A benchmark. In *ECCV*, pages 414–429. Springer, 2012.

[123] Ali Borji and Laurent Itti. State-of-the-art in visual attention modeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):185–207, 2013.

[124] Nicolas Riche, Matei Mancas, Bernard Gosselin, and Thierry Dutoit. Rare: A new bottom-up saliency model. In *IEEE International Conference on Image Processing (ICIP)*, pages 641–644, 2012.

[125] Keren Fu, Chen Gong, Jie Yang, Yue Zhou, and Irene Yu-Hua Gu. Superpixel based color contrast and color distribution driven salient object detection. *Signal Processing: Image Communication*, 28(10):1448–1463, 2013.

[126] Jonathan Harel, Christof Koch, and Pietro Perona. Graph-based visual saliency. In *Advances in neural information processing systems (NIPS)*, pages 545–552, 2006.

[127] Jia Li, Yonghong Tian, Tiejun Huang, and Wen Gao. Probabilistic multi-task learning for visual saliency estimation in video. *International journal of computer vision*, 90(2):150–165, 2010.

[128] Jia Li, Yonghong Tian, and Tiejun Huang. Visual saliency with statistical priors. *International journal of computer vision*, 107(3):239–253, 2014.

[129] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *International Conference on Learning Representations (ICLR)*. 2014.

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

[131] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Simultaneous detection and segmentation. In *ECCV*, pages 297–312. 2014.

[132] Ali Borji, Ming-Ming Cheng, Huaizu Jiang, and Jia Li. Salient object detection: A survey. *ArXiv e-prints*, 2014.

[133] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Mueller. How to explain individual classification decisions. *Journal of Machine Learning Research*, 11:1803–1831, 2010.

[134] Pablo Arbeláez, Jordi Pont-Tuset, Jonathan T Barron, Ferran Marques, and Jitendra Malik. Multiscale combinatorial grouping. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 328–335, 2014.

[135] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics (TOG)*, 23(3):309–314, 2004.

[136] Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li. Mode regularized generative adversarial networks. *arXiv preprint arXiv:1612.02136*, 2016.

[137] Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*, 2016.

[138] Mu Li, Wangmeng Zuo, and David Zhang. Deep identity-aware transfer of facial attributes. *arXiv preprint arXiv:1610.05586*, 2016.

[139] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[140] Pauline Luc, Camille Couprie, Soumith Chintala, and Jakob Verbeek. Semantic segmentation using adversarial networks. *arXiv preprint arXiv:1611.08408*, 2016.

[141] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.