Development of Recurrent Neural Networks and Its Applications to Activity Recognition

Shuai Li

University of Wollongong

Follow this and additional works at: https://ro.uow.edu.au/theses1

Copyright Warning

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site.

You are reminded of the following: This work is copyright. Apart from any use permitted under the Copyright Act 1968, no part of this work may be reproduced by any process, nor may any other exclusive right be exercised, without the permission of the author. Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

Unless otherwise indicated, the views expressed in this thesis are those of the author and do not necessarily represent the views of the University of Wollongong.

Recommended Citation


Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au
Development of Recurrent Neural Networks and Its Applications to Activity Recognition

Shuai Li

This thesis is presented as part of the requirements for the conferral of the degree:

Doctor of Philosophy

Supervisor:
A/PR Wanqing Li

Co-supervisor:
Prof. Chris Cook

The University of Wollongong
School of Computing and Information Technology

July, 2018
Declaration

I, Shuai Li, declare that this thesis is submitted in partial fulfilment of the requirements for the conferral of the degree *Doctor of Philosophy*, from the University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualifications at any other academic institution.

________________________

Shuai Li

October 15, 2018
Publications

Publications to date resulted from the research presented in this thesis.


Abstract

Deep learning, including the Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN), has enjoyed great success in the last decade. They have been widely applied in the areas of image and video-related tasks, such as image recognition and action recognition. Despite their great success, there are still some fundamental problems that have not been resolved. This thesis addresses the generic challenges in the CNNs and RNNs as well as challenges specific to their use in human activity recognition. First, an RNN-based pooling function is developed to replace the handcrafted and predefined pooling functions. Together with the other layers of the CNNs, this allows all components of the network to be trained from data. Then an independently recurrent neural network (IndRNN) is proposed to solve the gradient vanishing and exploding problem in the conventional RNNs. Unlike the traditional RNNs, IndRNN can learn very long-term patterns (over 5000 time steps) and can be stacked to construct very deep networks (over 21 layers). Application of IndRNN to activity recognition is studied where a deep IndRNN based attention model is developed. Finally, applications in complex scenes are explored using camouflaged moving background modelling. Extensive experiments have been conducted to validate the methods proposed in this thesis.
Contents

Abstract v

1 Introduction 1
   1.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
   1.2 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
   1.3 Thesis Organization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2 Literature Review 5
   2.1 Convolutional Neural Network (CNN) . . . . . . . . . . . . . . . . . . . . . 5
      2.1.1 Plain CNNs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
      2.1.2 Residual CNNs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
      2.1.3 Pooling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
   2.2 Recurrent Neural Network (RNN) . . . . . . . . . . . . . . . . . . . . . . . 11
      2.2.1 Simple RNNs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
      2.2.2 Long short-term memory (LSTM) and variants . . . . . . . . . . . . 13
   2.3 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

3 Fully Trainable Network 16
   3.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
   3.2 The proposed fully trainable network . . . . . . . . . . . . . . . . . . . . . 16
      3.2.1 Extension of an LSTM unit for pooling . . . . . . . . . . . . . . . . 17
      3.2.2 FTN architectures . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
   3.3 Experimental Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>vii</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.1 LSTM for average and max pooling</td>
<td>21</td>
</tr>
<tr>
<td>3.3.2 Analysis of the LSTM based pooling</td>
<td>24</td>
</tr>
<tr>
<td>3.3.3 Classification performance on PASCAL VOC 2012</td>
<td>31</td>
</tr>
<tr>
<td>3.3.4 Classification performance on CIFAR-10 and CIFAR-100</td>
<td>32</td>
</tr>
<tr>
<td>3.4 Summary</td>
<td>36</td>
</tr>
<tr>
<td>4 Independently Recurrent Neural Network (IndRNN)</td>
<td>37</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>37</td>
</tr>
<tr>
<td>4.2 Proposed Independently Recurrent Neural Network</td>
<td>39</td>
</tr>
<tr>
<td>4.2.1 Backpropagation Through Time for An IndRNN</td>
<td>40</td>
</tr>
<tr>
<td>4.3 Multiple-layer IndRNN</td>
<td>43</td>
</tr>
<tr>
<td>4.3.1 Deeper and Longer IndRNN Architectures</td>
<td>44</td>
</tr>
<tr>
<td>4.4 Experiments</td>
<td>45</td>
</tr>
<tr>
<td>4.4.1 Adding Problem</td>
<td>45</td>
</tr>
<tr>
<td>4.4.2 Sequential MNIST Classification</td>
<td>49</td>
</tr>
<tr>
<td>4.4.3 Language Modeling</td>
<td>50</td>
</tr>
<tr>
<td>4.4.4 Complexity Evaluation</td>
<td>52</td>
</tr>
<tr>
<td>4.5 Summary</td>
<td>53</td>
</tr>
<tr>
<td>5 Application to Skeleton based Activity Recognition</td>
<td>55</td>
</tr>
<tr>
<td>5.1 Introduction</td>
<td>55</td>
</tr>
<tr>
<td>5.2 Related Work</td>
<td>57</td>
</tr>
<tr>
<td>5.3 Proposed Method</td>
<td>59</td>
</tr>
<tr>
<td>5.3.1 IndRNN-based Deep Attention Model</td>
<td>59</td>
</tr>
<tr>
<td>5.3.2 Triplet Attention Loss</td>
<td>62</td>
</tr>
<tr>
<td>5.4 Experiments and Analysis</td>
<td>65</td>
</tr>
<tr>
<td>5.4.1 Results on NTU RGB+D dataset</td>
<td>65</td>
</tr>
<tr>
<td>5.4.2 Results on SBU Kinect Interaction Dataset (SBU)</td>
<td>73</td>
</tr>
<tr>
<td>5.4.3 Results on Berkeley MHAD Dataset</td>
<td>75</td>
</tr>
</tbody>
</table>
5.5 Summary ................................................................. 76

6 A Fusion Framework for Camouflaged Moving Foreground Detection in the Wavelet Domain 77
6.1 Introduction .......................................................... 77
6.2 Related work ......................................................... 78
  6.2.1 Statistical Background Models .............................. 78
  6.2.2 Transform Domain Background Models ...................... 81
  6.2.3 Background Models for Camouflaged Scenes ................ 84
6.3 Motivation and overview ........................................ 85
  6.3.1 Motivation ...................................................... 85
  6.3.2 Overview of the proposed framework ....................... 88
6.4 The proposed method .............................................. 90
  6.4.1 Foreground and background formulation for each wavelet band .... 90
  6.4.2 Fusion of the likelihood from different wavelet bands .......... 92
6.5 Experiments ........................................................ 94
  6.5.1 Experimental setting ........................................ 94
  6.5.2 Performance evaluation ..................................... 96
6.6 Summary .......................................................... 102

7 Conclusion 103
7.1 Summary of Contributions ...................................... 103
7.2 Future Work ...................................................... 105

Bibliography 108
Chapter 1

Introduction

1.1 Motivation

Deep learning, including convolutional neural networks (CNN) and recurrent neural networks (RNN), has been a great success in the last decade. It has achieved promising performance on many computer vision tasks including image classification, object detection and recognition, and action recognition and prediction. Through the applications of deep learning, CNNs have been shown to be capable of extracting spatial features. By contrast, RNNs have been proven successful in modeling sequential or time-series data such as the temporal patterns in action recognition.

Despite the success of CNNs and RNNs, there are still unresolved issues related to their architectures. For instance, a CNN is generally composed of convolution layers, pooling layers, and fully connected layers. Among them, the convolution layer and the fully connected layer are fully learned while the pooling layer is still handcrafted. Since the power of CNNs comes from their ability to adapt to the data through learning, the natural question to ask is “could pooling be learned from data in a similar way as other components?” Although some attempts are reported in the literature, the pooling layers are still restricted to a predefined form. Therefore, a pooling function that can be completely learned is highly desirable.

As for RNNs, a long-standing problem is the gradient vanishing and exploding problem resulting from the repeated use of the recurrent weight matrix. This makes RNNs very difficult to train and to recognize long-term patterns. Some RNN variants such as long short-term memory
CHAPTER 1. INTRODUCTION

(LSTM) and gated recurrent unit (GRU) have been developed to address these problems, but
the use of specific hyperbolic tangent and the sigmoid activation functions results in gradient
decay over layers. Consequently, construction of an efficiently trainable deep RNN is challeng-
ing. Moreover, in the current RNN structures, the neurons in each layer are entangled with each
other, which makes it very difficult to interpret and understand each neuron’s behaviour.

In addition to the above challenges with respect to the architecture of CNNs and RNNs,
there are also challenges at the application to make the results interpretable. For instance, both
CNN and RNN have been widely applied to activity recognition, an active field in computer
vision intending to understand the human motion from a sequence of images, depth maps and/or
skeletons. They are often simply treated as a black box with little being known about why and
how it works even though the results are good.

1.2 Contributions

This thesis focuses on improving the CNNs and RNNs by dealing with the above challenges.
Specifically, the contributions can be summarized as follows:

- A RNN-based learnable pooling function is proposed. Combined with the other learn-
able components, a fully trainable network (FTN) is developed. Experimental results
demonstrate that the proposed RNN based pooling can well approximate the existing
pooling functions with just one neuron. Furthermore, experiments have also shown that
the proposed FTN can improve the performance over conventional CNNs and achieve
state-of-the-art performance in image classification.

- A new type of RNN, referred to as independently recurrent neural network (IndRNN),
is proposed, where neurons in one layer are independent of each other and they are con-
nected across layers. We have shown that an IndRNN can be easily regulated to prevent
the gradient exploding and vanishing problems while allowing the network to learn long-
term dependencies. Moreover, an IndRNN can work with non-saturated activation func-
tions such as ReLU (rectified linear unit) and be still trained robustly. Multiple IndRNNs
can be stacked to construct a network that is deeper than the existing RNNs. Experimental results have shown that the proposed IndRNN is able to process very long sequences (over 5000 time steps vs. less than 1000 steps in a conventional RNN), can be used to construct very deep networks (21 layers used in the experiment) and still be trained robustly. Better performances have been achieved on various tasks by using IndRNNs compared with the traditional RNN and LSTM.

- A deep IndRNN based attention model is constructed and applied to skeleton-based activity recognition. A new triplet loss function is developed to supervise the learning of attention in order to enforce the intra-class attention distance to be smaller than the inter-class attention distance and at the same time to allow multiple attention weight patterns exist for a same class. Significantly better and robust performance is achieved over the traditional RNN models.

- A fusion framework is developed in the wavelet domain to address the camouflaged moving foreground detection problem. Experimental results demonstrate that the proposed method performs significantly better than the existing methods in terms of the camouflaged foreground detection.

1.3 Thesis Organization

The rest of the thesis is organized as follows.

Chapter 2 provides a literature review of the existing CNN and RNN models. The widely used CNN and RNN architectures are described. The problems in the existing architectures such as the predefined pooling function and the exploding and vanishing gradient are explained in detail.

Chapter 3 presents a RNN-based pooling method and the corresponding fully trainable network. Different FTN architectures are explored. Experimental results are presented to demonstrate that the proposed RNN-based pooling can well approximate the existing max and average pooling functions. The effectiveness of the FTN on existing tasks is also demonstrated.
Chapter 4 presents the independently recurrent neural network (IndRNN). The advantage of IndRNN over traditional RNNs are demonstrated by showing how the existing gradient vanishing and exploding problem is solved. Different IndRNN network architectures including residual IndRNN are illustrated. Performances on various tasks are presented.

Chapter 5 applies the IndRNN to skeleton based activity recognition and attention based IndRNN models are explored. The attention behaviour for the skeleton based activity recognition is exploited to construct a new triplet loss function.

Chapter 6 presents a fusion framework to address the camouflaged moving foreground detection problem. Foreground and background models are formulated in multiple wavelet bands. The detection results from different bands are fused by considering the properties of different wavelet bands. Experimental results have shown that the proposed method performs significantly better than the existing methods in terms of the camouflaged foreground detection.

Chapter 7 concludes this thesis with a summary of the main results and suggests future research works.
Chapter 2

Literature Review

This chapter reviews the existing methods in the literature related to the topics in this thesis. CNNs and RNNs are both introduced and their limitations are discussed.

2.1 Convolutional Neural Network (CNN)

Convolutional neural networks (CNNs) are feedforward neural networks. In the following, we first give a brief review on the widely used CNN architectures in the categories of plain CNN and residual CNN, respectively. Then a critical review on pooling, one common component of both architectures, is provided to show the disadvantages of the existing pooling methods.

2.1.1 Plain CNNs

Conventional CNNs are generally constructed based on the basic components of convolutional layer and activation function such as ReLU (rectified linear unit). Every few convolutional layers (3 for example), a pooling layer is inserted to reduce the size of the feature maps, and a few (usually two) fully connected layers are added at the end of the network. Some typical examples are the “LeNet” [1], “AlexNet” [2] and “VGGNet”[3]. Taking the “VGGNet” for example, it is composed of 5 stacks of convolutional layers with a 2 × 2 pooling layer at the end of each stack, and 2 layers of fully connected layers at the bottom of the network. To accelerate the training of CNNs, batch normalization [4] is usually used.

These network architectures based on the simple components of convolutional layer and ac-
CHAPTER 2. LITERATURE REVIEW

Convolutional Neural Networks (CNNs) are usually referred to as plain CNNs to distinguish them from the residual network. An illustration of a plain CNN is shown at Fig. 2.1 where each convolution layer is composed of a convolution operation, an activation function (ReLU) and batch normalization if needed. There are many more network architectures available in the literature [5]–[7].

It is known that gradient decays over layers and with the current training techniques based on the gradient backpropagation chain rule, such gradient decay behaviour limits the depth of the networks, usually around 20 layers. Fig. 2.2 from [8] shows an example of the performance on CIFAR classification using plain networks with different numbers of layers. It can be seen that with the number of layers increasing from 20 to 50, the performance drops. However, as shown in various reports [9], the deep layers reveal high level information and it is desirable to develop deep networks, which may further improve the performance.

2.1.2 Residual CNNs

As mentioned above, gradient decays over layers within a plain CNN and thus its depth is constrained. To overcome this issue, highway networks [10] and residual networks [8] have been developed. In the highway networks, in addition to the connection between two adjacent layers, two nonlinear transforms termed as transform gate and carry gate are introduced. The gates determine how much of the output is produced by transforming the input and carrying it. In the residual networks, the two nonlinear transforms are simplified to a shortcut connection with fixed weights 1 between two layers. A typical component (also known as the residual block) is shown in Fig. 2.3(a). It can be seen that, with the residual connection, the output of deeper layers is the combination of the output of plain CNN block and the output of the earlier layers. Therefore, in the training process, the gradients of a deeper layer can be directly propagated to earlier layers, greatly reducing the gradient vanishing problem across layers. This allows very deep networks (over 100 layers) to be trained. Fig. 2.4 from [8] shows an example of the performance on CIFAR classification task using the residual network with different numbers of layers. It can be seen that the performance increases with increasing number of layers.

This architecture is further improved to a preactivation type [11] as shown in Fig. 2.3(b),
**Figure 2.1:** Illustration of a plain CNN. (a) Basic architecture, (b) Basic component of a convolutional layer.

**Figure 2.2:** Demonstration of the performance of plain networks with different number of layers. Courtesy of [8].
Figure 2.3: Basic components of Residual CNN. (a) the conventional type, (b) the preactivation type. Courtesy of [11]

which has a better performance than the simple residual network mentioned above. Several works based on the residual connection have also been proposed in the literature such as the wide ResNets [12] and DenseNet [13], which are not further reviewed here. In addition to the residual networks to increase the depth of the networks, there are also networks with certain structures in each layer such as the “InceptionNet” [14] to increase the width of the network. In the “InceptionNet” [14], in each stack of layers, different convolutional kernels and layer structures are used. Works on combining both the advantage of residual networks and inception structures have also been explored in the literature which show better performance [15].

Despite the great success of residual connections enhanced networks, they still use a handcrafted pooling component. In the following, the existing pooling techniques are reviewed and its shortcomings are analysed.
CHAPTER 2. LITERATURE REVIEW

Figure 2.4: Demonstration of the performance of the residual network with different number of layers. Courtesy of [8].

2.1.3 Pooling

Motivated from biology [16] where responses of simple cells are fed into a complex cell through some pooling operations, the spatial pooling approach has been found very useful in many computer vision tasks. In CNNs, pooling is an important component for aggregating local features and reducing computational burden. Up to now, the most commonly used pooling methods are still the max pooling and average pooling. The max pooling selects the most salient feature in a pooling region while the average pooling treats the features in a pooling region equally.

However, the features in a local pooling region may be heterogeneous, leading to a loss of information on weak features through max pooling and loss of discriminative information through average pooling [17]. It has been shown that such pooling methods cannot achieve the optimal performance due to this information loss [17]. A theoretical analysis of max pooling and average pooling for classification is provided in [18] based on the i.i.d. Bernoulli distribution assumption for binary features and the exponential distribution for continuous features. It shows that the pooling cardinality and sparsity of the features affect its classification performance and the performance highly depends on the distribution of the features which is hard to estimate.
CHAPTER 2. LITERATURE REVIEW

In addition to the max pooling and average pooling, there are some other pooling methods reported in the literature. In [19], a protected pooling method was proposed where a concave function is used to combine the features. The concave function is designed to protect weak codes in order to preserve details. In [20], a stochastic pooling method was proposed where a multinomial distribution formed from the activation values is used. In this way, the locations with large values are picked as output more frequently than others. Similarly in [21], a rank based pooling was proposed to emphasize information with high rank over others. In [22], spectral pooling was proposed which preforms dimensionality reduction by truncating the representation in the frequency domain. On the other hand, the convolution with sliding strides larger than one pixel [23] can be regarded as an extra convolutional layer with a pooling operation which selects the value of a fixed location. The learning pooling in [24] further simplifies the convolutional operation with an independent linear operation on each channel. These methods are all heavily engineered to certain functions or certain forms which cannot achieve optimal performance on different data, different tasks and different networks.

There are also methods proposed to combine different pooling functions. In [25], a mixed pooling method was proposed where max pooling and average pooling are randomly selected with a stochastic procedure. Similarly, in [26], the max pooling and average pooling are combined in a tree structure. In [27] and [28], a geometric $l_p$-norm pooling was proposed to generalize max pooling and average pooling, which can be represented as $(\sum_{i=1}^{N} |x_i|^p)^{1/p}$. When $p = 1$, $l_p$-norm pooling reduces to average pooling and when $p = \infty$, $l_p$-norm pooling reduces to max pooling.

Pooling is a process that maps the $N \times N$ to 1 where $N \times N$ represents the size of the local pooling region. The existing pooling methods certainly lose information in this mapping process. Compared to the other layers in CNNs such as the convolutional layers and the fully connected layers, the predefined pooling function may not be optimal for a given dataset. Moreover, different pooling methods are used in different CNN architectures. Even in one CNN architecture such as the “InceptionNet”[14], different pooling methods are used. While it is difficult to select an appropriate pooling strategy for a better performance, it is also hard to explain how and why
one pooling strategy works better than others. Therefore, a flexible pooling function that can be learned from data for each pooling layer of a network architecture is highly desirable.

2.2 Recurrent Neural Network (RNN)

Unlike standard feedforward neural networks such as CNNs, recurrent neural networks (RNNs) [29] retain a state that can represent information from the past. Therefore, a RNN is naturally applicable to sequential data and has been widely used in video analysis, speech recognition and neural machine translation. In the following, simple RNN and its variants such as long short-term memory (LSTM) [30] are reviewed.

2.2.1 Simple RNNs

Simple RNNs add a cycle to connect adjacent time steps, termed as recurrent connection. An illustration of a recurrent neuron and its input and output is shown in Fig. 2.5. By unfolding the neuron through the time domain as in Fig. 2.6, a more direct look at the neuron processing inputs of multiple time instances is provided. At time step \( t \), a recurrent neuron receives inputs from the current data point \( x(t) \) and also the previous hidden state \( h(t - 1) \). The output of the neuron is calculated based on the hidden state \( h(t) \) at time \( t \). In this way, input from the previous time \( x(t - 1) \) can influence the current output \( y(t) \) at time \( t \) and later by way of the recurrent connections.

It is known that a RNN suffers from the gradient vanishing and exploding problem due to the repeated multiplication of the recurrent weight matrix. This makes it very difficult to train and capture long dependencies. To address this problem, work on initialization and training techniques, such as initializing the recurrent weights to a proper range or regulating the norm of the gradients over time, were proposed in the literature. In [31], an initialization technique was proposed for an RNN with ReLU activation, termed as IRNN, which initializes the recurrent weight matrix to be the identity matrix and bias to be zero. In [32], the recurrent weight matrix was further suggested to be a positive definite matrix with the highest eigenvalue of unity and all the remaining eigenvalues less than 1. In [33], the geometry of RNNs was investigated.
Figure 2.5: Illustration of a recurrent neuron (a) and its input and output (b).

Figure 2.6: A recurrent neuron unfolded in time
and a path-normalized optimization method for training was proposed for RNNs with ReLU activation. In [34], a penalty term on the squared distance between successive hidden states’ norms was proposed to prevent the exponential growth of IRNN’s activation. Although these methods help ease the gradient exploding, they are not able to completely avoid the problem (the eigenvalues of the recurrent weight matrix may still be larger than 1 in the process of training). Moreover, the training of an RNN with ReLU is very sensitive to the learning rate. When the learning rate is large, the gradient is likely to explode.

There are some RNN variants trying to solve the gradient vanishing and exploding problem by altering the recurrent connections. In [35], a skip RNN was proposed where a binary state update gate is added to control the network to skip the processing of one time step. In this way, less time steps may be processed and the gradient vanishing and exploding problem can be alleviated to some extent. In [36], [37], a unitary evolution RNN was proposed where the recurrent weights are empirically defined in the form of a unitary matrix. In this case, the norm of the backpropagated gradient can be bounded without exploding. In [38], a Fourier Recurrent Unit (FRU) was proposed where the hidden states over time are summarized with Fourier basis functions and the gradients are bounded. However, such methods usually introduce other transforms on the recurrent weight which complicates the recurrent units, making it hard to use and interpret.

### 2.2.2 Long short-term memory (LSTM) and variants

In order to solve the gradient vanishing and exploding problem, long short-term memory (LSTM) [30] is introduced. The key component of LSTM is a constant error carousel (CEC). It is a self-connected linear unit which enforces a constant error flow through time. Multiplicative gates including input gate, output gate and forget gate are introduced, respectively, to protect the memory content stored in CEC, to protect other units from perturbation by current irrelevant memory contents in CEC, and to reset the memory units once the memory is out of date and useless. The gates are controlled by input data, recurrent input and the current cell state (termed as peephole connections).
A well known LSTM variant is the gated recurrent unit (GRU) [39]. It is composed of a reset gate and an update gate. The update gate selects whether the hidden state is to be updated with a new hidden state, while the reset gate decides whether the previous hidden state is ignored. It has been reported in various papers [40] that GRU achieves similar performance as LSTM and so this not further explained here. There are also some other LSTM variants [40]–[43] reported in the literature. However, these architectures [40]–[43] generally take a similar form as LSTM and show a similar performance as well, and so are not discussed further here.

There is also research on making the LSTM learn longer sequences and construct deeper networks. In [38], auxiliary loss is developed to force LSTM to construct the previous events or predict next events of a sequence, making LSTM process longer sequences. In [44], different deep RNN architectures were investigated including stacking multiple layers to process the hidden states, which results in a deep transition RNN. This is further improved in [45] where a global gating unit is added to allow signals to flow from upper recurrent layers to lower layers. In [46], adaptive computation steps are taken between two time steps, which also forms a deep transition. In [47], a recurrent highway network was proposed where at each time step, multiple layers with highway connections are used to process the input and recurrent input. There is also research on extending LSTM to process multidimensional inputs [48], [49] and bidirectional extension [50].

LSTM and its variants enforce a constant error flow over time steps and use gates on the input and the recurrent input to regulate the information flow through the network. However, the use of gates based on the recurrent input prevents parallel computation and thus increases the computational complexity of the whole network. To process the states of the network over time in parallel, the recurrent connections are fixed in [51], [52]. While this strategy greatly simplifies the computational complexity, it reduces the capability of their RNNs since the recurrent connections are no longer trainable.

On the other hand, in LSTM and its variants, the hyperbolic tangent and the sigmoid functions are usually used as the activation function resulting in gradient decay over layers. Consequently, construction and training of a deep LSTM based RNN network is practically difficult. By
contrast, existing CNNs using non-saturated activation function such as ReLU can be stacked into a very deep network (e.g. over 20 layers using the basic convolutional layers and over 100 layers with residual connections [8]) and be still trained efficiently. Although residual connections have been attempted for LSTM models in several works [53], [54], there have been no significant improvement (mostly due to the reason that gradient decays in LSTM with the use of the hyperbolic tangent and the sigmoid functions as mentioned above). Therefore, RNN architectures that can be stacked with multiple layers and efficiently trained are still highly desired.

2.3 Summary

This Chapter first reviews the existing popular deep learning architectures including CNNs and RNNs. Some limitations are discussed and the following issues, in particular, will be addressed in this thesis:

- the handcrafted pooling functions used in the existing CNN architectures, making pooling not adaptive to data.

- the gradient vanishing and exploding problem in the training of RNNs, making RNNs not robust and difficult for processing long sequences.

- the shallow depth of RNN networks because of the gradient decay over layers.

- the difficulty in understanding RNN features due to the entanglement of different neurons.

In addition to the related works described as above, works specifically related to particular topics are further reviewed in each chapter.
Chapter 3

Fully Trainable Network

3.1 Introduction

The basic component of a CNN is a stack of convolutional layers (usually more than 2) followed by a pooling layer as shown in Fig. 3.1(a). The convolutional layer can be of many forms such as the traditional convolution structure [3], inception structure [14] and the residual structure [8]. Normalization layers [4] may be used after or before each convolutional layer. The pooling layer is often a max pooling, average pooling or a pooling function as discussed above. Instead of using a pooling layer, a convolutional layer with stride larger than 1 can be used [23] as shown in Fig. 3.1(b) to reduce the dimension of the output features. This chapter proposes a learnable pooling function based on recurrent neural units (RNN). With the capability of a RNN in aggregating features of a sequence, RNNs can effectively aggregate the feature in a local pooling region. Together with the convolutional layers and fully connected layers in CNNs, such a learnable pooling leads to a fully trainable network (FTN).

3.2 The proposed fully trainable network

Unlike the conventional CNNs as shown in Fig. 3.1(a) and Fig. 3.1(b), in the proposed FTN, the basic component is a stack of convolutional layers followed by a RNN layer as shown in Fig. 3.1(c). Specifically, the features in each pooling region are scanned into a sequence as input to the RNN layer. There are many ways to perform the scan. The output of the RNN layer
at the last time stamp is the aggregated feature of the local pooling region, and so is treated as
the pooled value. It has been empirically shown that the performance of the FTN is insensitive
to the scanning order. Thus simple horizontal scanning is adopted in this chapter. In addition to
reducing the dimension of the features, the RNN layer also intends to capture the pattern of the
features in a local region. A FTN is constructed by stacking such components.

Notice that in general any type of RNN can be used in the FTN for pooling. This chapter
adopts the commonly used LSTM unit. In the following, FTNs are explained in detail with
respect to the extension of an LSTM unit for pooling, and the FTN architectures, respectively.

3.2.1 Extension of an LSTM unit for pooling

In the study of CNNs, a general consensus is that for deep networks non-saturated activation
functions such as rectified linear units (ReLU) are easier to train than the saturated activation
functions such as logistic and hyperbolic tangent functions. In this chapter, it is proposed to
extend a conventional LSTM unit with non-saturated activation functions to preform pooling.
Such an extension facilitates the training of the LSTM in a consistent way with other layers in
a FTN.

The key component in LSTM [30] is a constant error carousel (CEC) which enforces a con-
stant error flow over time steps. Fig. 3.2 illustrates an LSTM without considering peephole connections. In addition to the CEC, LSTM contains three gates (input gate, forget gate and output gate), and two modulations (input modulation and output modulation). The gates are controlled by the current input and the recurrent input. The activation function for the gates is usually the sigmoid function ($\sigma$). The activation functions ($\psi$) used in input and output modulations are usually the hyperbolic tangent function ($tanh$). For input $x$ at each time step $t$, the LSTM updates its states as follows:

$$
\begin{align*}
    \mathbf{i}^t &= \sigma(W_i \mathbf{x}^t + R_i \mathbf{h}^{t-1} + \mathbf{b}_i) \\
    \mathbf{f}^t &= \sigma(W_f \mathbf{x}^t + R_f \mathbf{h}^{t-1} + \mathbf{b}_f) \\
    \mathbf{o}^t &= \sigma(W_o \mathbf{x}^t + R_o \mathbf{h}^{t-1} + \mathbf{b}_o) \\
    \mathbf{g}^t &= \psi(W_g \mathbf{x}^t + R_g \mathbf{h}^{t-1} + \mathbf{b}_g) \\
    \mathbf{c}^t &= \mathbf{i}^t \odot \mathbf{g}^t + \mathbf{f}^t \odot \mathbf{c}^{t-1} \\
    \mathbf{h}^t &= \mathbf{o}^t \odot \psi(\mathbf{c}^t)
\end{align*}
$$

where $\mathbf{x}^t \in \mathbb{R}^M$, $\mathbf{h}^{t-1} \in \mathbb{R}^N$ and $M, N$ represent the dimension of the input feature at time step $t$ and the number of the neurons in LSTM, respectively. $\mathbf{i}^t$, $\mathbf{f}^t$ and $\mathbf{o}^t$ are the outputs of the input gate, forget gate and output gate, respectively. $\mathbf{g}^t$, $\mathbf{c}^t$ and $\mathbf{h}^t$ are the output of the input modulation, the cell state and the output of the LSTM, respectively. $W_v$, $R_v$ and $b_v$ are the weight of the current input, the weight of the recurrent input, and the bias, respectively, for the input gate ($v = i$), forget gate ($v = f$), output gate ($v = o$) and input modulation ($v = g$). $\odot$ represents the point-wise multiplication.

With the hyperbolic tangent function used as the activation function for input and output modulations, the output of the LSTM is constrained to the range of $(-1, 1)$. However, the convolutional layers and fully connected layers in most CNN architectures employ non-saturated activation functions such as ReLU where their output ranges in $[0, +\infty)$. Therefore, the activation functions of the input and output modulations in a LSTM unit are required to be the same.
CHAPTER 3. FULLY TRAINABLE NETWORK

Figure 3.2: Illustration of a LSTM unit.

as those used in the convolutional layers.

However, change of the activation functions of the input and output modulations in a LSTM unit from a saturated function to a non-saturated function would usually make the training of the LSTM hard to converge according to [31]. In this chapter, one LSTM unit is applied to pool features in a local patch in each channel. Since the processing of all samples in the batch is the same, we take the processing of one data sample as an example to illustrate the LSTM based pooling process. Accordingly, the dimension of input feature from one data sample and the number of neurons are both 1, i.e., \( M = 1 \) and \( N = 1 \). That is, the input, recurrent input, cell state and outputs of all the gates at each time instance and the corresponding weight parameters are all of dimension 1. Let them be noted as \( x^t, h^t, i^t, f^t, o^t, c^t, w_g, r_g, w_i, r_i, b_v \), respectively. The following proposition is used as a regulation in the training of LSTM.

**Proposition:** \( w_g > 0 \) is a necessary condition for a LSTM neuron with ReLU activation function to converge when processing non-negative input features \( (x^t) \).

**Proof:** (Proof by Contradiction) Let the bias of the input modulation be first ignored and considered later, that is, \( g^t = \psi(w_g x^t + r_g h^{t-1}) \), where \( \psi \) is the ReLU activation function and \( x^t \geq 0 \). The initial state of the recurrent input \( (h^0) \) and cell \( (c^0) \) are both set to be 0 which is used in most networks. Assume \( w_g \leq 0 \). Starting with time instance 1, with \( x^1 \geq 0 \), the output of the input modulation \( g^1 \) is zero. Since the outputs of all the gates including input gate, output gate and forget gate are non-negative, the output of LSTM \( (h^1) \) and the cell state \( (c^1) \) is 0. Hence, the recurrent input and the cell state for the next time instance 2 remains 0. Together
with \( x^2 \geq 0 \), the output stays 0. It can be deduced that under such circumstances, the output of LSTM stays 0, which cannot be trained to converge. Therefore, the assumption does not hold and the opposite proposition \((w_g > 0)\) is true. On the other hand, bias determines the threshold to activate a neuron. For a LSTM unit, a negative bias for the input modulation deactivates the neuron for small inputs, making the neurons incapable of processing small features. Therefore, the bias for the input modulation is suggested to be constrained to be non-negative as well, in order to preserve the unit’s ability to deal with small inputs.

Experimental results presented in the Section 3.3 of this chapter show that a LSTM unit with ReLU activation function can be trained robustly if this proposition is met.

### 3.2.2 FTN architectures

The proposed LSTM based pooling can be integrated with convolutional layers in different ways to create different FTN architectures.

- One FTN architecture is that each local pooling region has its own LSTM to be trained and these LSTM units can work as different pooling functions for different regions. In this FTN, pooling is adaptive to local regions.

- The second FTN architecture has one LSTM per layer that is shared by all local regions in the layer. In this case, one pooling operation is performed on all local regions. However, pooling at different layers can be different depending on the training. For instance, the LSTM in one layer may act like a max pooling and the LSTM in another layer may act like an average pooling or a different function that the LSTM would best approximate for the training data.

- The third FTN architecture is one LSTM unit shared by all pooling layers. This is equivalent to the conventional CNN where either max or average pooling is adopted.

Obviously, the first architecture is the most powerful one as it is adaptive to each region. However, it also has the maximum number of parameters to be trained for the pooling, and the
network may become overfitting if it is not well regularized. With the large number of parameters, the memory needed to train the network is also large and the training takes longer time. Thus in the experiments, the second and third architectures are evaluated and compared to illustrate the benefits of the proposed LSTM based pooling over the traditional pooling.

3.3 Experimental Results

Experiments were first conducted to validate that one LSTM unit can well approximate the max and average pooling functions, thus showing the proposed LSTM based pooling is appropriate to be used as pooling in the network. Next, experiments were devised to illustrate that an adaptive pooling such as the proposed LSTM based pooling can outperform the predefined max or average pooling and analysis on the learned pooling function is provided. Finally, the performance of FTN is verified on conventional classification tasks by comparing with existing state-of-the-art methods.

3.3.1 LSTM for average and max pooling

The average pooling is a simple linear function which can be easily approximated by LSTM. However, the max pooling function is a highly non-linear function. In the following, an experiment was devised to show that one LSTM unit is able to approximate the max pooling function to a high degree of accuracy.

Simulation setup: ReLU was assumed as the activation function for the convolutional layers that a LSTM unit would work with. Experiments for other non-saturated activation functions can be conducted in a similar way. For ReLU ($max(0,x)$), the range of the output in theory is $[0, +\infty)$. In experiments of image classification, it is observed that the outputs generally fall in the range $[0, 300]$. Therefore, random numbers in this range were generated as the input to simulate the output of a convolutional layer. Since the pooling sizes used most in CNN are $2 \times 2$, $3 \times 3$ and $4 \times 4$, three sets of experiments were conducted with lengths of the input being 4, 9 and 16, respectively. One LSTM unit with the modified activation function (ReLU here) was used. The algorithms are implemented using Theano [55] and Lasagne [56], and run on
CHAPTER 3. FULLY TRAINABLE NETWORK

Table 3.1: MAE(10^{-5}) of one LSTM unit with the modified activation function to approximate a max pooling function.

<table>
<thead>
<tr>
<th>Pool size</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 × 2</td>
<td>8.97</td>
<td>8.91</td>
<td>9.19</td>
</tr>
<tr>
<td>3 × 3</td>
<td>4.42</td>
<td>4.39</td>
<td>4.40</td>
</tr>
<tr>
<td>4 × 4</td>
<td>5.41</td>
<td>5.28</td>
<td>5.32</td>
</tr>
</tbody>
</table>

GPU (Titan XP).

Training: The LSTM was trained by minimizing the mean absolute error (MAE) between the output and the max value of the input to approximate the max pooling using mini-batch gradient descent with Nesterov momentum [57] and the batch size was set to 128. The initial learning rate was set to 0.1 and the momentum was set to 0.9. Regularizations such as weight decay and dropout were not used since infinite training examples can be generated. 10^4 batches were considered as an epoch and one epoch was used for validation. The learning rate was decreased by a factor of 10 when the validation accuracy stopped improving. The input weight and bias of the input modulation in the LSTM unit was initialized and regulated as described in subsection 3.2.1.

Testing: The batch size used for testing is the same as that for training. One epoch (10^4 batches) of data was generated for testing. The performance of the trained network is evaluated on three sets of input data: T1: random numbers in the range [0, 300]; T2: 50% of random numbers in the range [0, 300] and the other 50% being 0; T3: 20% of random numbers in the range [0, 300] and the other 80% being 0. The tests were designed to simulate the cases of general patches, relatively sparse patches and highly sparse patches considering that the responses of the convolutional neurons can be sparse. The performance for different pooling sizes are tabulated in Table 3.1.

From Table 3.1, it can be seen that one LSTM unit is able to well approximate the max pooling function as the errors are all smaller than 10^{-4} (which is negligible compared to the data range of [0, 300]). It can be also seen that the performance is insensitive to the pooling sizes. The experiment on using the trained model for data of a bigger range is also conducted
Table 3.2: MAE($10^{-3}$) of one LSTM unit trained with data range $[0, 300]$ to approximate a max pooling function on data range $[0, 3000]$.

<table>
<thead>
<tr>
<th>Pool size</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2 \times 2$</td>
<td>0.994</td>
<td>0.993</td>
<td>0.994</td>
</tr>
<tr>
<td>$3 \times 3$</td>
<td>2.03</td>
<td>2.03</td>
<td>2.03</td>
</tr>
<tr>
<td>$4 \times 4$</td>
<td>9.10</td>
<td>9.10</td>
<td>9.10</td>
</tr>
</tbody>
</table>

to demonstrate that the learned pooling function can well process the data that is larger than the trained data range or at least in a relatively larger range. Experiments were conducted assuming the range of the test data is $[0, 3000]$ while the LSTM was trained on data within $[0, 300]$. The results in terms of MAE are shown in the Table 3.2. It can be seen that although the MAE is larger than that for data in the range $[0, 300]$, it is still relatively small (all smaller than 0.01) considering the large input range (3000). This demonstrates that the learned pooling function can deal with the data larger than the trained data. To further demonstrate that our RNN-based pooling can approximate max pooling well, the pre-trained RNN-based pooling is used for classification on ImageNET as shown in the following.

**pre-trained LSTM-based pooling for ImageNET Classification**

Limited by our computing resources, we are not able to train a large network on ImageNET. We only show the result of replacing the pooling function in the existing pre-trained network with our proposed RNN based pooling function (pre-trained as in Subsection 3.3.1) in Table 3.3. It can be seen that the network combined with our proposed pooling function almost achieves the same performance as the original network (within marginal error), which validates that our proposed pooling function can well approximate the max pooling function. For VGG19 network, the network with the proposed RNN based pooling surprisingly achieves slightly better performance than the original network in terms of the top5 error rate. Since the network is not trained after replacing the pooling, we conjecture that the slight increase may be due to the small discriminability introduced by the proposed RNN based pooling. It can be seen that the slight error in the max pooling approximation task does not affect the performance of the whole
Table 3.3: Results on ImageNET in terms of validation error rate (%) using the pre-trained network with max pooling and the pre-trained LSTM based pooling (proposed), respectively.

<table>
<thead>
<tr>
<th></th>
<th>Top-1</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Max pooling</td>
<td>Proposed (No training)</td>
</tr>
<tr>
<td>VGG16</td>
<td>26.12</td>
<td>26.11</td>
</tr>
<tr>
<td>VGG19</td>
<td>26.03</td>
<td>26.03</td>
</tr>
</tbody>
</table>

Table 3.4: Classification result comparison on CIFAR-10 in terms of test error rate (%) using different sizes of networks.

<table>
<thead>
<tr>
<th>Network</th>
<th>Max pooling</th>
<th>Average pooling</th>
<th>Proposed pooling (shared)</th>
<th>Proposed pooling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv_8</td>
<td>57.44</td>
<td>56.18</td>
<td>50.96</td>
<td>50.52</td>
</tr>
<tr>
<td>Conv_16</td>
<td>32.75</td>
<td>32.86</td>
<td>28.58</td>
<td>25.72</td>
</tr>
<tr>
<td>Conv_32</td>
<td>18.77</td>
<td>20.82</td>
<td>16.11</td>
<td>15.48</td>
</tr>
<tr>
<td>Conv_64</td>
<td>13.27</td>
<td>14.75</td>
<td>12.04</td>
<td>11.83</td>
</tr>
</tbody>
</table>

network. Considering that RNNs are universal approximators and can be further trained with data and task, it is reasonable to assume that the proposed FTN with the RNN based pooling can outperform the traditional CNNs with max pooling. Note that this experiment is only used to demonstrate that a learned max pooling function works very well by just replacing the pooling function in the existing models without further training. In the following, the performance of FTN over CNNs will be illustrated and the learned pooling functions will be investigated.

3.3.2 Analysis of the LSTM based pooling

Performance on different sizes of networks

To illustrate the effectiveness of the proposed LSTM based pooling, experiments were conducted on the popular CIFAR-10 dataset. The dataset was preprocessed in the same way as in [58]. That is, the dataset is preprocessed with global contrast normalization and ZCA whitening, and the images were padded with four zero pixels at borders. While training, 32 × 32 random crops with random horizontal flipping were used as input.

A CNN, composing of two stacks of 3 × 3 convolutional layers (2 layers in each stack) with a pooling layer at the end of each stack, 2 fully connected layers, and an additional fully con-
nected layer of 10 units together with a softmax output layer for classification was used. The same number of units, denoted by \( N \), were used in the convolutional layers and the 2 fully connected layers. The corresponding network is denoted as Conv\_N. To better demonstrate the effectiveness of the LSTM based pooling, a large local pooling region, namely \( 4 \times 4 \) and \( 8 \times 8 \) for the first and second pooling layers, respectively, were used. After the pooling layers, the size of the input to the fully connected layers is \( 1 \times 1 \), thus the fully connected layers work in the same way as convolutional layers. A leaky ReLU unit with leakiness of 0.3 was used as the activation function for both convolutional layers and fully connected layers, which has been reported [59] to achieve a good performance on classification. Batch normalization [4] was used for convolutional layers and dropout (drooping rate 50%) was applied after each fully connected layer. Total norm constraint on the gradients as in [60] was used to stabilize the training. The initial learning rate was set to 0.01 and decreased by a factor of 10 after 50k and 40k iterations, respectively, and the training ended at the 122K-th iteration. SGD with Nesterov momentum [57] of 0.9 was used for training, and the batch size was 100.

The results are shown in Table 3.4. The column of “proposed pooling” and “proposed pooling (shared)” represent the second and third architectures described in subsection 3.2.2, respectively. That is to say, for “proposed pooling (shared)”, both pooling layers share one LSTM unit while for “proposed pooling”, each pooling layer has one LSTM unit. From the table, three observations can be made:

- The networks with the proposed LSTM pooling always improve the accuracy (lower the error rate) compared with the corresponding CNNs coupled with the traditional max pooling or average pooling function.

- When the network is very small such as Conv\_8, the performance improvement due to the LSTM based pooling is significant, up to 7 percentage points. As the network size increases, the improvement decreases. It is conjectured that although a fixed pooling function may not optimally aggregate the local features, extra convolution kernels may compensate this to some extent. Thus with the increase of the convolution units, the gain
of using a better pooling function over traditional pooling method drops. However, given the data and task, if the network is too large, it may become overfitting. Therefore, a better pooling function to aggregate the features while reducing the requirement of neurons is highly desired.

- The performance of the second architecture of FTN is better than the third, i.e., different LSTM units for different pooling layers improves the performance of FTN. This indicates that the optimal pooling functions for different pooling layers are likely to be different.

Figure 3.3: Illustration of the location selection in the max pooling over different sizes of the networks.
Figure 3.4: Outputs of the learned pooling function from networks of different sizes in comparison with the max pooling and average pooling. (a) Learned function of the first pooling layer, (b) learned function of the second pooling layer.
Analysis of the learned pooling function

As shown in Table 3.4 and described above, the performance gap between the proposed pooling and the existing pooling methods becomes smaller with the increase of the number of convolution kernels, especially for the max pooling. Since pooling is a $N \times N$ to 1 mapping process, information of certain locations in the pooling region may be lost in the existing pooling process, leading to a degraded performance of the network. In the following, we first show that networks are trained to preserve information of different locations in a pooling region.

For a CNN, max pooling is used independently for each channel. And for each channel, it selects the value of one location (which is the location with the maximal value) as the output and the information at that location is implicitly carried forward in this channel. That is to say, for the pooling process over multiple channels, information from a number of locations may be selected and preserved by one or more channels. Fig. 3.3 shows the histogram of the number of locations that have been selected by at least one channel. The output of max pooling for 5000 randomly selected local patches in the dataset are used for different networks. The pooling size is $4 \times 4$ in all networks, leading to a local region of 16 locations. For the Conv_8 network, the number of neurons is 8, and thus at most 8 locations can be selected by the max pooling (when locations selected by different channels are all different). Fig. 3.3(a) shows that generally more than 4 locations have been selected in one or more channels, and for some pooling regions, all 8 locations are selected. For the Conv_16 network shown in 3.3(b), generally more than 6 locations have been selected. For the Conv_64 network shown in 3.3(b), in over 50% pooling regions, all 16 locations have been selected. It is reasonable to assume that when the number of the neurons is large enough, information from all locations may be implicitly carried forward after the pooling operation. It indicates that networks (convolution kernels) are trained to sample information from all locations. Consequently, with the increase of the number of neurons, the effect of a good pooling operation may be reduced since information from more locations could be sampled through different channels. However, in this case, noise may also be kept by different channels and when the number of neurons is very large, the network becomes overfitting and degrades the performance. On the other hand, since LSTM
can aggregate information of a sequence, it can adaptively sample more information from all the pooling locations rather than just the existing pooling functions. Therefore, the performance of the proposed FTN is significantly better than the traditional CNN when the number of neurons used is small. Moreover, compared to CNN, FTN can lower the requirement of the number of neurons for a better performance in order to avoid overfitting.

To illustrate the learned LSTM based pooling functions for different pooling layers in the network of different sizes, the output of the pooling function in comparison with the max pooling and average pooling is shown in Fig. 3.4. For better illustration, random values with a fixed maximum value (1.5) are used as input and the output is rearranged according to the magnitude of the average pooling result. Fig. 3.4(a) shows the output of the learned pooling function from the first pooling layer. “Conv_N” represents the outputs obtained from the different pooling functions learned from their corresponding “Conv_N” networks, respectively. Note that the mean value of each pooling result can be compensated by the bias of the neurons in the following layer, thus the variation of each pooling result is more meaningful. It can be seen that the learned pooling functions of the first pooling layer work similarly as the average pooling. Especially for the small networks such as “Conv_8” and “Conv_16”, the output highly correlates with the output of average pooling and the variation is relatively small. This indicates that average pooling may perform better than max pooling for the first pooling layer of small networks, which agrees with our results shown in Table 3.4.

For the learned pooling function of the second pooling layer as shown in Fig. 3.4(b), it can be seen that the variation is very large, i.e., highly sensitive to the different patterns of the inputs. First, compared with the input to the first pooling layer, each input to the second pooling layer corresponds to a larger region of the original image and thus more useful information for the task. Second, it is known that outputs of the higher layers in the network capture high level information, and information at different locations may produce different contexts for the final classification task. For example, the same input with different orders may produce different results. Thus it is very important for the pooling layer to aggregate information while capturing useful patterns. This can be done using our proposed pooling but not possible for the traditional
Table 3.5: Complexity comparison between CNN and the proposed FTN in terms of time (sec per batch).

<table>
<thead>
<tr>
<th></th>
<th>Train with cudnn</th>
<th>Train without cudnn</th>
<th>Test with cudnn</th>
<th>Test without cudnn</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN</td>
<td>0.013</td>
<td>0.021</td>
<td>0.0027</td>
<td>0.0037</td>
</tr>
<tr>
<td>FTN</td>
<td>0.033</td>
<td>0.035</td>
<td>0.0068</td>
<td>0.0076</td>
</tr>
</tbody>
</table>

max pooling and average pooling, leading to the performance improvement shown in Table 3.4.

By comparing the learned pooling functions of two layers, it can be seen that the optimal pooling functions for different pooling layers are quite different. In the traditional CNNs, max pooling and average pooling are often selected empirically. In the “InceptionNet” [14], both max pooling and averaging pooling are adopted at different layers also empirically. In such a case, the whole network is unlikely to achieve the best performance. On the other hand, our proposed LSTM based pooling is able to be adaptive for each layer to the training data and thus achieve a better performance, as shown in Table 3.4 by comparing the performance the “Proposed pooling (shared)” and “Proposed pooling”.

Complexity

As mentioned in Subsection 3.2, the proposed LSTM based pooling first transforms the $N \times N$ local region to a sequential input of length $N \times N$. Then it processes this sequential input. For example, for the general $2 \times 2$ pooling, LSTM needs to process inputs of 4 time steps. It is known that the update of LSTM at each time step in Eq. (3.1) can be regarded as convolution with multiple channels. So the complexity of the proposed pooling is similar to performing 4 convolutional layers. To evaluate its complexity, a similar network as the Conv_64 network (except that the pooling size is set to be $2 \times 2$) was used. Batch size was set to be 1 to purely monitor the computation without considering memory issues. The program was implemented based on Theano [61] and Lasagne, and runs on a TITAN X GPU. The time used in the training and testing process is shown in Table 3.5. Since convolution is heavily optimized in cudnn (the deep neural network library developed in NVIDIA CUDA), we show both results obtained with
cudnn and without cudnn. It can be seen that the complexity of training the above FTN network is about two-three times that for training CNN. This is consistent with our above analysis that the complexity of the proposed LSTM based pooling is similar to performing 4 convolutional layers. Since pooling is only applied a few times depending on the size of the input (around 5 times for input of size 256), the complexity of training FTN is bounded. Compared to the current networks over 100 layers, the increase in training time is acceptable considering the benefit of having a learnable pooling function to improve the performance. Moreover, compared to the image modelling methods that use RNN to process the whole image in a sequential manner, the time increase due to the proposed pooling is relatively very small. It is worth noting that since the proposed pooling is learned from data for a network, it can be used as a tool to develop new pooling functions for different applications.

3.3.3 Classification performance on PASCAL VOC 2012

This subsection reports the results on PASCAL Visual Object Classes Challenge (VOC) 2012 dataset [62]. VOC contains 22531 images in 20 classes for training (5717) and validation (5823), and testing (10991). For this dataset, since there may be multiple labels for each image, instead of using softmax as in Subsection 3.3.2, sigmoid is used at the end of the network to indicate whether each class presents in an image. To demonstrate the performance of our RNN-based pooling over the existing pooling methods, networks similar to those in Subsection 3.3.2 are chosen where two stacks of convolutional layers are used and the number of parameters used for the convolutional layers is the same, referred to as Conv_N. Two pooling layers are used with pooling size $4 \times 4$ and $8 \times 8$, respectively, and two fully connected layers are used at the end with 256 neurons. The dataset is preprocessed into size $134 \times 134$ and random crops of size $128 \times 128$ are used as input. Considering the size of the dataset, batch size is set to 16. Other settings are the same as in Subsection 3.3.2, where SGD with Nesterov momentum [57] of 0.9 was used and the initial learning rate was set to 0.01. Average pooling, max pooling and the proposed RNN-based pooling are evaluated for all networks. Moreover, the VGG16 model [3] pre-trained on ImageNET is also fine-tuned on this dataset with the original max pooling
Table 3.6: Classification result comparison on PASCAL VOC 2012 in terms of mAP (%) using different networks.

<table>
<thead>
<tr>
<th>Network</th>
<th>Max pooling</th>
<th>Average pooling</th>
<th>Proposed pooling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv.32</td>
<td>40.9</td>
<td>35.1</td>
<td>41.5</td>
</tr>
<tr>
<td>Conv.64</td>
<td>43.1</td>
<td>34.6</td>
<td>44.9</td>
</tr>
<tr>
<td>VGG16 (pre-trained on ImageNET)</td>
<td>84.15</td>
<td>N/A</td>
<td>84.59</td>
</tr>
</tbody>
</table>

and the pre-trained LSTM-based pooling (replacing the max pooling), respectively. For the fine-tuning process, the last layer of 1000 neurons with softmax function in the VGG16 model is replaced by 20 neurons with sigmoid function for the classification. The dataset is preprocessed into size $256 \times 256$ and random crops of size $224 \times 224$ as in [3] are used as input. Initial learning rate of 0.01 is used first to train the last layer with other layers fixed and then dropped by a factor of 10 to train all the layers.

The performance is shown at Table 3.6 in terms of mean Average Precision (mAP) [62]. For the fine-tuning process, the result on the test set (evaluated by the PASCAL VOC server) is presented while for the others, the result on the validation set is presented for simplicity. From Table 3.6, it can be seen that the proposed RNN-based pooling consistently improves the performance over max pooling and average pooling. Mover, even for the fine-tuning process where the parameters are first learned based on max pooling, the proposed RNN-based pooling still improves the performance of the overall model by fine-tuning on the dataset (relatively small compared with ImageNET).

### 3.3.4 Classification performance on CIFAR-10 and CIFAR-100

In this Subsection, the widely used CIFAR-10 and CIFAR-100 datasets were used to evaluate the performance of the proposed FTN (the second architecture). A similar network as the VGG16 architecture [3] was used for classification due to its popularity. Fig. 3.5 shows the detailed architecture. It is composed of 5 stacks of convolutional layers with a $2 \times 2$ pooling layer at the end of each stack, and 2 layers of fully connected layers in the end. Since the spatial size of the input to the fully connected layers is $1 \times 1$, the fully connected layers works in the
**Figure 3.5:** Illustration of the network architecture used for classification on CIFAR-10 and CIFAR-100.

**Table 3.7:** Comparison of the proposed FTN and CNNs on CIFAR-10 and CIFAR-100 in terms of test error rate (%).

<table>
<thead>
<tr>
<th>Network</th>
<th>CIFAR-10</th>
<th>CIFAR-100</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSN[63]</td>
<td>7.97</td>
<td>34.57</td>
</tr>
<tr>
<td>NIN[64]</td>
<td>8.81</td>
<td>35.68</td>
</tr>
<tr>
<td>Maxout[65]</td>
<td>9.38</td>
<td>38.57</td>
</tr>
<tr>
<td>All-CNN[23]</td>
<td>7.25</td>
<td>33.71</td>
</tr>
<tr>
<td>Highway Network[66]</td>
<td>7.60</td>
<td>32.24</td>
</tr>
<tr>
<td>ELU[58]</td>
<td>6.55</td>
<td>24.28</td>
</tr>
<tr>
<td>LSUV[59]</td>
<td>6.06</td>
<td>29.96</td>
</tr>
<tr>
<td>LSUV* [59]</td>
<td>5.84</td>
<td>N/A</td>
</tr>
<tr>
<td>LEAP[24]</td>
<td>7.17</td>
<td>29.80</td>
</tr>
<tr>
<td>Stochastic Pooling[20]</td>
<td>15.13</td>
<td>42.51</td>
</tr>
<tr>
<td>Rank based Pooling[21]</td>
<td>13.84</td>
<td>43.91</td>
</tr>
<tr>
<td>Mixed Pooling[25]</td>
<td>10.80</td>
<td>38.07</td>
</tr>
<tr>
<td>Tree Pooling[26]</td>
<td>6.67</td>
<td>33.13</td>
</tr>
<tr>
<td>Tree+Max-Avg Pooling[26]</td>
<td>6.05</td>
<td>32.37</td>
</tr>
<tr>
<td>Baseline</td>
<td>6.29</td>
<td>27.09</td>
</tr>
<tr>
<td><strong>Proposed FTN</strong></td>
<td><strong>5.79</strong></td>
<td><strong>26.89</strong></td>
</tr>
<tr>
<td>With extreme data augmentation[67]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>All-CNN[23]</td>
<td>4.41</td>
<td>N/A</td>
</tr>
</tbody>
</table>

LSUV* is obtained with deep residual network using maxout as activation function. N/A represents the result is not provided in the corresponding paper.
Figure 3.6: Test error comparison between the proposed FTN and the traditional CNN, (a) over the entire training process, (b) over the first 20 epochs.
CHAPTER 3. FULLY TRAINABLE NETWORK

same way as convolutional layers. In the figure, the convolutional layer includes the convolution operation, batch normalization and activation function, and the size of the input to the convolutional layer and the number of neurons are also shown in the figure. A fixed 10/100 units fully connected layer together with a softmax output layer are added in the end for classification of CIFAR-10 and CIFAR-100, respectively. The leaky ReLU unit with leakiness of 0.1 was used as activation functions for both the convolutional layers and fully connected layers. Dropout was used after the pooling layers (dropping rate 30%) and fully connected layers (dropping rate 50%) to regularize the training. The preprocessing of the dataset and the training procedure are the same as in Subsection 3.3.2. For the proposed FTN, the pooling layers were replaced with LSTM units, one for each pooling layer. It is worth noting that one LSTM unit only introduces 12 parameters. On the contrary, one convolutional unit generally introduces $N \times N \times C_{j-1} + 1$ units where $N$ is the filter size, $C_{j-1}$ is the number of channels of the input to the current unit, and $+1$ indicates the bias. Compared to the large amount of parameters used in CNN, the increased number of parameters due to a LSTM unit is negligible.

The results of the FTN and the comparison to the existing methods using the similar plain CNN architectures are shown in Table 3.7. It can be seen that under similar training conditions (without the extreme data augmentation [67]), the proposed FTN outperforms the baseline network with max pooling and achieves better performance than other pooling methods using the similar network architecture. Although LSTM has been reported to be difficult to train in the literature, it is found that the proposed FTN converges very fast in the experiments, even faster than a CNN with traditional max pooling. The test errors of the proposed FTN and CNN versus the training epochs are shown in Fig. 3.6(a). Fig. 3.6(b) shows the zoomed-in curve of the testing errors of the first 20 epochs. It can be clearly seen that the proposed FTN achieved a relatively higher performance in less iterations than the CNN.
3.4 Summary

In this Chapter, a fully trainable network (FTN) is proposed, where the handcrafted pooling layer in the traditional CNNs is replaced with RNN in the proposed FTN. Due to the capability of a LSTM or RNN in general in modelling sequential data, the proposed learnable pooling can be trained to capture patterns of the data in the pooling regions. Experimental results have verified the efficacy of the proposed RNN based pooling and FTN. Specifically, we have shown that only one LSTM unit based pooling can approximate the existing pooling functions with a very high accuracy. Moreover, the performance on image classification tasks including the CIFAR-10, CIFAR-100 and PASCAL VOC demonstrates that the proposed FTN with the LSTM based pooling achieves better performance than the existing pooling methods.
Chapter 4

Independently Recurrent Neural Network (IndRNN)

4.1 Introduction

Recurrent neural networks (RNNs) [29] have been widely used to solve problems of sequential data such as action recognition [68] and language processing [39], and have achieved impressive results. Compared with the feed-forward networks such as the convolutional neural networks (CNNs), a RNN has a recurrent connection where the last hidden state is an input to the next state. The update of states can be described as follows:

\[ h_t = \sigma(Wx_t + Uh_{t-1} + b) \]  

where \( x_t \in \mathbb{R}^M \) and \( h_t \in \mathbb{R}^N \) are the input and hidden state at time step \( t \), respectively. \( W \in \mathbb{R}^{N \times M} \), \( U \in \mathbb{R}^{N \times N} \) and \( b \in \mathbb{R}^N \) are the weights for the current input and the recurrent input, and the bias of the neurons. \( \sigma \) is an element-wise activation function of the neurons, and \( N \) is the number of neurons in this RNN layer. By unfolding RNN in time, this can be illustrated as in Fig. 4.1(a).

Training of the RNNs suffers from the gradient vanishing and exploding problem due to the repeated multiplication of the recurrent weight matrix. Several RNN variants such as the long short-term memory (LSTM) [40], [43] and the gated recurrent unit (GRU) [39] have been proposed to address the gradient problems. However, the use of the hyperbolic tangent and the
sigmoid functions as the activation function in these variants results in gradient decay over layers. Consequently, construction and training of a deep LSTM or GRU based RNN network is practically difficult. By contrast, existing CNNs using non-saturated activation function such as ReLU can be stacked into a very deep network (e.g. over 20 layers using the basic convolutional layers and over 100 layers with residual connections [8]) and be still trained efficiently. Although residual connections have been attempted for LSTM models in several works [53], [54], there has been no significant improvement (mostly due to the reason that gradient decays in LSTM with the use of the hyperbolic tangent and the sigmoid functions as mentioned above).

Moreover, the existing RNN models share the same component $\sigma(Wx_t + Uh_{t-1} + b)$ in (4.1), where the recurrent connection entangles all the neurons. This makes it hard to interpret and understand the roles of the trained neurons (e.g., what patterns each neuron responds to) since the outputs of individual neurons is affected by others.

In this chapter, a new type of RNN, referred to as independently recurrent neural network (IndRNN), is proposed. In the proposed IndRNN, the gradient backpropagation through time can be regulated to effectively address the gradient vanishing and exploding problems. It can work well with a non-saturated function such as ReLU as activation function and be trained robustly. Multiple layers of IndRNNs can be efficiently stacked, especially with residual connections over layers, to increase the depth of the network. Long-term memory can be kept with IndRNNs to process long sequences. Behaviour of IndRNN neurons in each layer are easy to interpret due to the independence of neurons in each layer. Experiments have demonstrated that an IndRNN can well process sequences over 5000 steps while LSTM could only process less than 1000 steps. An example of a deep (21 layer) IndRNN is demonstrated in the experiments for language modelling. The experimental results have demonstrated that IndRNN performs much better than the traditional RNN and LSTM models on the tasks of the adding problem, sequential MNIST classification, language modelling and action recognition.
4.2 Proposed Independently Recurrent Neural Network

The proposed independently recurrent neural network (IndRNN) can be described as:

$$h_t = \sigma(Wx_t + u \odot h_{t-1} + b)$$

(4.2)

where recurrent weight $u$ is a vector and $\odot$ represents Hadamard product. Each neuron in one layer is independent from others and connection between neurons can be achieved by stacking two or more layers of IndRNNs as presented later. IndRNN can be illustrated as in Fig. 4.1(b) by unfolding it in time. By comparing with RNN in Fig. 4.1(a), it can be clearly seen neurons of IndRNN in one layer are independent from each other while their correlation is explored in the following layers.

For the $n$-th neuron, the hidden state $h_{n,t}$ can be obtained as

$$h_{n,t} = \sigma(w_n x_t + u_n h_{n,t-1} + b_n)$$

(4.3)

where $w_n$ and $u_n$ are the $n$-th row of the input weight and recurrent weight, respectively. Each neuron only receives information from the input and its own hidden state at the previous time step. That is, each neuron in an IndRNN deals with one type of spatial-temporal pattern inde-
pendently. Conventionally, a RNN is treated as multiple layer perceptrons over time where the parameters are shared. Different from the conventional RNNs, the proposed IndRNN provides a new perspective of recurrent neural networks as independently aggregating spatial patterns (i.e. through $w$) over time (i.e. through $u$). The correlation among different neurons can be exploited by stacking two or multiple layers. In this case, each neuron in the next layer processes the outputs of all the neurons in the previous layer as shown in Fig. 4.1(b).

4.2.1 Backpropagation Through Time for An IndRNN

For a traditional RNN network, its hidden state in (4.1) can be simply represented as $h_t = \sigma(Wx_t + Uh_{t-1})$ where the bias $b$ is included in the input weight and input is extended with an extra dimension. Suppose that the input to the network is $T$ time steps and the objective trying to minimize at time step $T$ is $J$. Then the gradient back propagated to time step $t$ is

$$\frac{\partial J}{\partial h_t} = \frac{\partial J}{\partial h_T} \frac{\partial h_T}{\partial h_t} = \frac{\partial J}{\partial h_T} \prod_{k=t}^{T-1} \frac{\partial h_{k+1}}{\partial h_k}$$

$$= \frac{\partial J}{\partial h_T} \prod_{k=t}^{T-1} \text{diag}(\sigma'(h_{k+1}))U^T$$

(4.4)

where $\text{diag}(\sigma'(h_{k+1}))$ is the Jacobian matrix of the element-wise activation function. As the time step $T$ grows, the gradient may grow exponentially large or small depending on the recurrent weight matrix $U$ and the activation function $\sigma$, leading to the gradient exploding or vanishing problems mentioned above. With the involvement of the recurrent weight matrix, it is very difficult to directly constrain the gradient in a desirable range. Moreover, if the eigenvalues are ill conditioned [69], a small perturbation to the recurrent matrix would cause a large change to the eigenvalues and accordingly a large change to the gradient value. Therefore, the training of RNN may become unstable.

For the gradient backpropagation through time in each layer, the gradients of an IndRNN can be calculated independently for each neuron since there are no interactions among them in one layer. For the $n$-th neuron $h_{n,t} = \sigma(w_n x_t + u_n h_{n,t-1})$ where the bias is ignored, suppose the objective trying to minimize at time step $T$ is $J_n$. Then the gradient back propagated to the time
where $\sigma'_{n,k+1}$ is the derivative of the element-wise activation function. It can be seen that the gradient only involves the exponential term of a scalar value $u_n$ which can be easily regulated, and the gradient of the activation function which is often bounded in a certain range. Compared with the gradients of an RNN ($\frac{\partial J_n}{\partial h_{n,T}} \prod_{k=t}^{T-1} \text{diag}(\sigma'(h_{k+1}))U^T$ where $\text{diag}(\sigma'(h_{k+1}))$ is the Jacobian matrix of the element-wise activation function), the gradient of an IndRNN directly depends on the value of the recurrent weight. Accordingly, it is changed by a small magnitude according to the learning rate instead of via a matrix product which is mainly determined by its eigenvalues and can be changed significantly even though the change to each matrix entries is small [69]. Thus the training of an IndRNN is more robust than a traditional RNN. To solve the gradient exploding and vanishing problem over time, we only need to regulate the exponential term “$u_n^{T-t} \prod_{k=t}^{T-1} \sigma'_{n,k+1}$” to an appropriate range. This is further explained in the following together with keeping long and short memory in an IndRNN.

To keep long-term memory in a network, the current state (at time step $t$) would still be able to effectively influence the future state (at time step $T$) after a large time interval. Consequently, the gradient at time step $T$ can be effectively propagated to the time step $t$. By assuming that the minimum effective gradient is $\epsilon$, a range for the recurrent weight of an IndRNN neuron in order to keep long-term memory can be obtained. Specifically, to keep a memory of $T-t$ time steps, $|u_n| \in \left( \frac{\epsilon}{\prod_{k=t}^{T-1} \sigma'_{n,k+1}}, \infty \right)$ according to (4.5) (ignoring the gradient backpropagated from the objective at time step $T$). That is, to avoid the gradient vanishing for a neuron, the above constraint should be met. In order to avoid the gradient exploding problem, the range needs to be further constrained to $|u_n| \in \left( \frac{\epsilon}{\prod_{k=t}^{T-1} \sigma'_{n,k+1}}, \frac{\gamma}{\prod_{k=t}^{T-1} \sigma'_{n,k+1}} \right)$ where $\gamma$ is the largest gradient value without exploding. For the commonly used activation functions such as ReLU and tanh,
their derivatives are no larger than 1, i.e., $|\sigma'_{n,k+1}| \leq 1$. Especially for ReLU, its gradient is either 0 or 1. Considering that the short-term memories can be important for the performance of the network as well, especially for a multiple layers RNN, the constraint to the range of the recurrent weight with ReLU activation function can be relaxed to $|u_n| \in (0, \frac{(T-t)^2}{\sqrt{Y}})$. Note that the regulation on the recurrent weight $u$ is different from the gradient clipping technique. For the gradient clipping or gradient norm clipping [70], the calculated gradient is already exploded and is forced back to a predefined range. The gradients for the following steps may keep exploding. In this case, the gradient of the other layers relying on this neuron may not be accurate. On the contrary, the regulation proposed here essentially maintains the gradient in an appropriate range without affecting the gradient backpropagation through this neuron.
4.3 Multiple-layer IndRNN

As mentioned above, neurons in the same IndRNN layer are independent of each other, and cross channel information over time is explored through multiple layers of IndRNNs. To illustrate this, we compare a two-layer IndRNN with a traditional single layer RNN. For simplicity, the bias term is ignored for both IndRNN and traditional RNN. Assume a simple $N$-neuron two-layer network where the recurrent weights for the second layer are zero which means the second layer is just a fully connected layer shared over time. The Hadamard product $(\mathbf{u} \odot \mathbf{h}_{t-1})$ can be represented in the form of matrix product by $\text{diag}(u_1, u_2, \ldots, u_N)\mathbf{h}_{t-1}$. In the following, $\text{diag}(u_1, u_2, \ldots, u_N)$ is shortened as $\text{diag}(u_i)$. Assume that the activation function is a linear function $\sigma(x) = x$. The first and second layers of a two-layer IndRNN can be represented by (4.6) and (4.7), respectively.

\[
\mathbf{h}_{f,t} = \mathbf{W}_f \mathbf{x}_{f,t} + \text{diag}(u_{f,i})\mathbf{h}_{f,t-1} \tag{4.6}
\]
\[
\mathbf{h}_{s,t} = \mathbf{W}_s \mathbf{h}_{f,t} \tag{4.7}
\]

Assuming $\mathbf{W}_s$ is invertible, then

\[
\mathbf{W}_s^{-1}\mathbf{h}_{s,t} = \mathbf{W}_f \mathbf{x}_{f,t} + \text{diag}(u_{f,i})\mathbf{W}_s^{-1}\mathbf{h}_{s,t-1} \tag{4.8}
\]

Thus

\[
\mathbf{h}_{s,t} = \mathbf{W}_s \mathbf{W}_f \mathbf{x}_{f,t} + \mathbf{W}_s \text{diag}(u_{f,i})\mathbf{W}_s^{-1}\mathbf{h}_{s,t-1} \tag{4.9}
\]

By assigning $\mathbf{U} = \mathbf{W}_s \text{diag}(u_{f,i})\mathbf{W}_s^{-1}$ and $\mathbf{W} = \mathbf{W}_s \mathbf{W}_f$, it becomes

\[
\mathbf{h}_t = \mathbf{W}_s \mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} \tag{4.10}
\]

which is a traditional RNN. Note that this only imposes the constraint that the recurrent weight ($\mathbf{U}$) is diagonalizable. Therefore, the simple two-layer IndRNN network can represent a traditional RNN network with a diagonalizable recurrent weight ($\mathbf{U}$). In other words, under linear activation, a traditional RNN with a diagonalizable recurrent weight ($\mathbf{U}$) is a special case of a
two-layer IndRNN where the recurrent weight of the second layer is zero and the input weight of the second layer is invertible.

It is known that a non-diagonalizable matrix can be made diagonalizable with a perturbation matrix composed of small entries. A stable RNN network needs to be robust to small perturbations (in order to deal with precision errors for example). It is mathematically possible to find an RNN network with a diagonalizable recurrent weight matrix to approximate a stable RNN network with a non-diagonalizable recurrent weight matrix. Therefore, a traditional RNN with a linear activation is a special case of a two-layer IndRNN. For a traditional RNN with a nonlinear activation function, its relationship with the proposed IndRNN is yet to be established theoretically. However, we have shown empirically in Section 4.4 that the proposed IndRNN can achieve better performance than a traditional RNN with a nonlinear activation function.

Regarding the number of parameters, for a \( N \)-neuron RNN network with input of dimension \( M \), the number of parameters in a traditional RNN is \( M \times N + N \times N \), while the number of parameters using one-layer IndRNN is \( M \times N + N \). For a two-layer IndRNN where both layers consist of \( N \) neurons, the number of parameters is \( M \times N + N \times N + 2 \times N \), which is of a similar order to the traditional RNN.

In all, the cross-channel information can be well explored with a multiple-layer IndRNN although IndRNN neurons are independent of each other in each layer.

### 4.3.1 Deeper and Longer IndRNN Architectures

In the proposed IndRNN, the processing of the input \((Wx_t + b)\) is independent at different time steps and can be implemented in parallel as in [51], [52]. The proposed IndRNN can be extended to a convolutional IndRNN where, instead of processing input of each time step using a fully connected weight \((Wx_t)\), it is processed with convolutional operation \((W \ast x_t, \text{where } \ast \text{ denotes the convolution operator})\).

The basic IndRNN architecture is shown in Fig. 4.2(a), where “weight” and “Recurrent+ReLU” denote the processing of input and the recurrent process at each step with ReLU as the activation function. By stacking this basic architecture, a deep IndRNN network can be constructed.
CHAPTER 4. INDEPENDENTLY RECURRENT NEURAL NETWORK (INDRNN) 45

Compared with an LSTM-based architecture using the sigmoid and hyperbolic tangent functions decaying the gradient over layers, a non-saturated activation function such as ReLU reduces the gradient vanishing problem over layers. In addition, batch normalization, denoted as “BN”, can also be employed in the IndRNN network before or after the activation function as shown in Fig. 4.2(a).

Since the weight layer \( \mathbf{Wx}_t + \mathbf{b} \) is used to process the input, it is natural to extend it to multiple layers to deepen the processing. Also the layers used to process the input can be of the residual structures in the same way as in CNN [8]. With the simple structure of IndRNN, it is very easy to extend it to different networks architectures. For example, in addition to simply stacking IndRNNs or stacking the layers for processing the input, IndRNNs can also be stacked in the form of residual connections. Fig. 4.2(b) shows an example of a residual IndRNN based on the “pre-activation” type of residual layers in [11]. At each time step, the gradient can be directly propagated to the other layers from the identity mapping. Since IndRNN addresses the gradient exploding and vanishing problems over time, the gradient can be efficiently propagated over different time steps. Therefore, the network can be substantially deeper and longer. The deeper and longer IndRNN network can be trained end-to-end similarly as other networks.

4.4 Experiments

In this Section, evaluation of the proposed IndRNN on various tasks is presented. The algorithms for all the tasks are implemented using Theano [55] and Lasagne [56], and run on GPU (Titan XP).

4.4.1 Adding Problem

The adding problem [30], [36] is commonly used to evaluate the performance of RNN models. Two sequences of length \( T \) are taken as input. The first sequence is uniformly sampled in the range \((0, 1)\) while the second sequence consists of two entries being 1 and the rest being 0. The output is the sum of the two entries in the first sequence indicated by the two entries of 1 in the second sequence. Three different lengths of sequences, \( T = 100, 500 \) and \( 1000 \), were used for
Figure 4.3: Results of the adding problem for different sequence lengths. The legends for all figures are the same and thus only shown in (a).
the experiments to show whether the tested models have the ability to model long-term memory.

The RNN models included in the experiments for comparison are the traditional RNN with tanh, LSTM, IRNN (RNN with relu). The proposed IndRNN was evaluated with relu activation function. Since GRU achieved similar performance as LSTM [40], it is not included in the report. RNN, LSTM, and IRNN are all one layer while the IndRNN model is two layers. 128 hidden units were used for all the models, and the number of parameters for RNN, LSTM, and two-layer IndRNN are $16K$, $67K$ and $17K$, respectively. It can be seen that the two-layer IndRNN has a comparable number of parameters to that of the one-layer RNN, while many more parameters are needed for LSTM. As discussed in Subsection 4.2.1, the recurrent weight is constrained in the range of $|u_n| \in (0, \sqrt{2})$ for the IndRNN.

Mean squared error (MSE) was used as the objective function and the Adam optimization method [71] was used for training. The baseline performance (predicting 1 as the output regardless of the input sequence) is mean squared error of 0.167 (the variance of the sum of two independent uniform distributions). The initial learning rate was set to $2 \times 10^{-3}$ for models with tanh activation and set as $2 \times 10^{-4}$ for models with relu activations. However, as the length of the sequence increases, the IRNN model does not converge and thus a smaller initial learning rate ($10^{-5}$) was used. The learning rate was reduced by a factor of 10 every 20K training steps. The training data and testing data were all generated randomly throughout the experiments, different from [36] which only used a set of randomly pre-generated data.

The results are shown in Fig. 4.3(a), 4.3(b) and 4.3(c). First, for short sequences ($T = 100$), most of the models (except RNN with tanh) performed well as they converged to a very small error (much smaller than the baseline). When the length of the sequences increases, the IRNN and LSTM models have difficulties in converging, and when the sequence length reaches 1000, IRNN and LSTM cannot minimize the error any more. However, the proposed IndRNN can still converge to a small error very quickly. This indicates that the proposed IndRNN can model a longer-term memory than the traditional RNN and LSTM.

From the figures, it can also be seen that the traditional RNN and LSTM can only keep a mid-range memory (about 500 - 1000 time steps). To evaluate the proposed IndRNN model for
very long-term memory, experiments on sequences with length 5000 were conducted where the result is shown in Fig. 4.3(d). It can be seen that IndRNN can still model it very well. Note that the noise in the result of IndRNN is because the initial learning rate \(2 \times 10^{-4}\) was relatively large and once the learning rate dropped, the performance became robust. This demonstrates that IndRNN can effectively address the gradient exploding and vanishing problem over time and keep a long-term memory.

**Analysis of Neurons’ Behaviour**

In the proposed IndRNN, neurons in each layer are independent of each other which allows analysis of each neuron’s behaviour without considering the effect coming from other neurons. Fig. 4.4(a) and 4.4(b) show the activation of the neurons in the first and second layers, respectively, for one random input with sequence length 5000. It can be seen that neurons in the first layer mainly pick up the information of the numbers to be added, where the strong responses correspond to the locations to be summed indicated by the sequence. It can be regarded as reducing noise, i.e., reducing the effect of other non-useful inputs in the sequence. For the second layer, one neuron aggregates inputs to long-term memory while others generally preserve their own state or process short-term memory which may not be useful in the testing case (since only the hidden state of the last time step is used as output). From this result, we conjecture that only one neuron is needed in the second layer to model the adding problem. Moreover, since neurons in the second layer are independent from each other, one neuron can still work with the others removed (which is not possible for the traditional RNN models).

To verify the above conjecture, an experiment was conducted where the first IndRNN layer is initialized with the trained weights and the second IndRNN layer only consists of one neuron initialized with the weight of the neuron that keeps the long-term memory. Accordingly, the final fully connected layer used for output is a neuron with only one input and one output, i.e., two scalar values including one weight parameter and one bias parameter. Only the final output layer was trained/fine-tuned in this experiment and the result is shown in Fig. 4.5. It can be seen that with only one IndRNN neuron in the second layer, the model is still able to model the
CHAPTER 4. INDEPENDENTLY RECURRENT NEURAL NETWORK (INDRNN) 49

Figure 4.4: Neurons’ behaviour in different layers of the proposed IndRNN for long sequences (5000 time steps) in the adding problem.

Figure 4.5: Result of the adding problem with just one neuron in the second layer for sequences of length 5000.

adding problem very well for sequences with length 5000 as expected.

4.4.2 Sequential MNIST Classification

Sequential MNIST classification is another problem that is widely used to evaluate RNN models. The pixels of MNIST digits [1] are presented sequentially to the networks and classification is performed after reading all pixels. To make the task even harder, the permuted MNIST classification was also used where the pixels are processed with a fixed random permutation. Since an RNN with tanh does not converge to a high accuracy (as reported in the literature [31]), only
Table 4.1: Results (in terms of error rate (%)) for the sequential MNIST and permuted MNIST.

<table>
<thead>
<tr>
<th>Method</th>
<th>MNIST</th>
<th>pMNIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRNN [31]</td>
<td>5.0</td>
<td>18.2</td>
</tr>
<tr>
<td>uRNN [36]</td>
<td>4.9</td>
<td>8.6</td>
</tr>
<tr>
<td>RNN-path [33]</td>
<td>3.1</td>
<td>-</td>
</tr>
<tr>
<td>LSTM [36]</td>
<td>1.8</td>
<td>12.0</td>
</tr>
<tr>
<td>LSTM+Recurrent dropout [72]</td>
<td>-</td>
<td>7.5</td>
</tr>
<tr>
<td>LSTM+Recurrent batchnorm [73]</td>
<td>-</td>
<td>4.6</td>
</tr>
<tr>
<td>LSTM+Zoneout [74]</td>
<td>-</td>
<td>6.9</td>
</tr>
<tr>
<td>LSTM+Recurrent batchnorm+Zoneout</td>
<td>-</td>
<td>4.1</td>
</tr>
<tr>
<td><strong>IndRNN (6 layers)</strong></td>
<td>1.0</td>
<td>4.0</td>
</tr>
</tbody>
</table>

IndRNN with ReLU was evaluated. As explained in Section 4.3.1, IndRNN can be stacked into a deep network. Here we used a six-layer IndRNN with batch normalization inserted after each layer, and each layer has 128 neurons. The Adam optimization was used with the initial learning rate $2 \times 10^{-4}$ and reduced by a factor of 10 every 600K training steps. The results are shown in Table 4.1 in comparison with the existing methods. It can be seen that IndRNN achieved better performance than the existing RNN models.

4.4.3 Language Modeling

Char-level Penn Treebank

In this subsection, we evaluate the performance of the proposed IndRNN on the language modelling task using the character-level Penn Treebank (PTB-c) dataset. The test setting is similar to [73]. A six-layer IndRNN with 2000 hidden neurons is used for the test. To demonstrate that the IndRNN network can be very deep with the residual connections, a 21-layer residual IndRNN as shown in Fig. 4.2(b) in Subsection 4.3.1 was adopted. The frame-wise batch normalization [75] is applied, and the batch size is set to 128. Adam was used for training with initial learning rate set to $2 \times 10^{-4}$ and dropped by a factor of 5 when performance on the validation set was no longer improved (with patience 20). Dropout [76] with a dropping probability of 0.25 and 0.3 were used for the 6-layer IndRNN and the residual IndRNN. The sequences are non-overlapping and length $T = 50$ and $T = 150$ were both tested in training and testing.

The results are shown in Table 4.2 in comparison with the existing methods. Performance
Table 4.2: Results of char-level PTB for our proposed IndRNN model in comparison with results reported in the literature, in terms of BPC.

<table>
<thead>
<tr>
<th>Model</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN-tanh [34]</td>
<td>1.55</td>
</tr>
<tr>
<td>RNN-relu [33]</td>
<td>1.55</td>
</tr>
<tr>
<td>RNN-TRec [34]</td>
<td>1.48</td>
</tr>
<tr>
<td>RNN-path [33]</td>
<td>1.47</td>
</tr>
<tr>
<td>HF-MRNN [77]</td>
<td>1.42</td>
</tr>
<tr>
<td>LSTM [74]</td>
<td>1.36</td>
</tr>
<tr>
<td>LSTM+Recurrent dropout [72]</td>
<td>1.32</td>
</tr>
<tr>
<td>LSTM+Recurrent batchnorm [73]</td>
<td>1.32</td>
</tr>
<tr>
<td>LSTM+Zoneout [74]</td>
<td>1.27</td>
</tr>
<tr>
<td>HyperLSTM + LN [78]</td>
<td>1.25</td>
</tr>
<tr>
<td>Hierarchical Multiscale LSTM + LN [79]</td>
<td>1.24</td>
</tr>
<tr>
<td>Fast-slow LSTM [80]</td>
<td>1.19</td>
</tr>
<tr>
<td>Neural Architecture Search [81]</td>
<td>1.21</td>
</tr>
<tr>
<td><strong>IndRNN (6 layers, 50 steps)</strong></td>
<td>1.26</td>
</tr>
<tr>
<td><strong>IndRNN (6 layers, 150 steps)</strong></td>
<td>1.23</td>
</tr>
<tr>
<td><strong>res-IndRNN (21 layers, 50 steps)</strong></td>
<td>1.21</td>
</tr>
<tr>
<td><em><em>res-IndRNN (11 layers</em>, 150 steps)</em>*</td>
<td>1.19</td>
</tr>
</tbody>
</table>

*Note that due to the limitation of GPU memory, an 11-layer residual IndRNN was used for time step 150 instead of 21 layers.

was evaluated using bits per character metric (BPC). It can be seen that the proposed IndRNN model achieved better performance than the traditional RNN and LSTM models. It can also been seen that with a deeper residual IndRNN, the performance can be further improved. Also an improvement can be achieved with longer temporal dependencies (from time step 50 to 150) as shown in Table 4.2.

**Word-level Penn Treebank**

In this subsection, the performance of the proposed IndRNN on the word-level Penn Treebank dataset is evaluated. The test setting is similar to [74]. A 11-layer residual IndRNN was used for test and the weight tying [82], [83] of the input embedding and the final output weight is also adopted. The frame-wise batch normalization [75] is applied, and the batch size is set to 128. Adam was used for training with initial learning rate set to $5 \times 10^{-4}$ and dropped by a factor of 5 when performance on the validation set was no longer improved (with patience 20).
Table 4.3: Results of word-level PTB for our proposed IndRNN model in comparison with results reported in the literature, in terms of perplexity.

<table>
<thead>
<tr>
<th>Model</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN-LDA + KN-5 + cache [84]</td>
<td>92.0</td>
</tr>
<tr>
<td>Deep RNN [44]</td>
<td>107.5</td>
</tr>
<tr>
<td>CharCNN [85]</td>
<td>78.9</td>
</tr>
<tr>
<td>LSTM [74]</td>
<td>114.5</td>
</tr>
<tr>
<td>LSTM+Recurrent dropout [72]</td>
<td>87.0</td>
</tr>
<tr>
<td>LSTM+Zoneout [74]</td>
<td>77.4</td>
</tr>
<tr>
<td>LSTM+Variational Dropout [76]</td>
<td>73.4</td>
</tr>
<tr>
<td>Pointer Sentinel LSTM [86]</td>
<td>70.9</td>
</tr>
<tr>
<td>RHN [87]</td>
<td>65.4</td>
</tr>
<tr>
<td>Neural Architecture Search [81]</td>
<td>62.4</td>
</tr>
<tr>
<td><strong>res-IndRNN (11 layers)</strong></td>
<td><strong>65.3</strong></td>
</tr>
</tbody>
</table>

The sequences are non-overlapping and length \( T = 50 \) was used in training and testing. Dropout [76] with a dropping probability of 0.35 was used among IndRNN layers (including embedding) while 0.8 was used after the last IndRNN layer. The recurrent weights are initialized with \( \text{Normal}(0.4, 0.2) \), which makes the network starts with learning more mid-range memory.

The results are shown in Table 4.3 in comparison with the existing methods. It can be seen that the proposed IndRNN model achieved better performance than most of the traditional RNN and LSTM models except the neural architecture search [81]. It is worth noting that the neural architecture search constructs new models through learning, which can be regarded as a simple way of implementing an exhaustive search of models and parameters and extremely computational expensive. On the other hand, the proposed method is tested with empirical parameters without heavy optimization.

### 4.4.4 Complexity Evaluation

As we described in Section 4.3, a two-layer IndRNN has the same order of number of parameters as the traditional RNN. Moreover, the computational complexities of the traditional RNN and the proposed KINN are also similar. From the point of view of matrix multiplication, they both involve two matrix product operations \((M \times N \text{ and } N \times N)\). Additionally, IndRNN requires another two elementwise vector product operations. On the other hand, LSTM has many more
(about 4 times) parameters than RNN and IndRNN, and it takes much more computation than RNN and IndRNN models.

To evaluate the computational complexity of different models, the adding problem for sequence of length 100 is used. The settings are the same as described in 4.4.1. The program is implemented based on Theano [61] and Lasagne, and runs on a TITAN X GPU. The training and testing time (seconds) for RNN, one-layer IndRNN, two-layers IndRNN and LSTM are shown in Fig. 4.6. For the RNN and IndRNN models, ReLU is used as the activation function. It can be seen that the results are consistent with our argument that two-layers IndRNN takes a similar time as RNN while LSTM takes much more time.

4.5 Summary

In this Chapter, we presented an independently recurrent neural network (IndRNN), where neurons in one layer are independent of each other. The gradient backpropagation through time process for the IndRNN has been explained and a regulation technique has been developed to effectively address the gradient vanishing and exploding problems. Compared with the existing RNN models including LSTM and GRU, IndRNN can process much longer sequences. The basic IndRNN can be stacked to construct a deep network especially combined with residual connections over layers, and the deep network can be trained robustly. In addition, independence among neurons in each layer allows better interpretation of the neurons. Experiments on
multiple fundamental tasks have verified the advantages of the proposed IndRNN over existing RNN models.
Chapter 5

Application to Skeleton based Activity Recognition

5.1 Introduction

Human action recognition has received increasing interest in the past due to its wide range of applications in video analytics, robotics, health monitoring and autonomous driving. The success of deep learning in computer vision has driven the development of many deep models [88]–[96] for action recognition. Among these models, recurrent neural network (RNN) [68], [97]–[99] is one of the popular ones because of its capability of modeling sequential data. Recently, RNNs are further augmented with attention models [100], [101] to explicitly model the observation that discriminative information presents in different body parts at different time steps. Noticeable improvement in performance has been attained [102], [103].

This chapter is concerned with two fundamental and challenging issues in an attention-based RNN for action recognition from skeleton data, where the attention weights are associated with joints of the skeletons. First, the state-of-the-art attention models, such as those presented in [102], [103], lacks proper regularization on attention weights to enforce that same class of actions would share similar attention weights and the attention weights of different actions would be sufficiently different, but at the same time the attention weights for the same class should be also allowed to vary to accommodate different performing styles. For example, the joints of legs in action “kicking” would have higher attention weights than other joints, so are the joints
of arms in action “boxing”. Therefore, the attention on joints for different actions are different, that is, the attention weights between two skeleton samples of “kicking” should be more similar than the attention weights between a skeleton sample of “kicking” and a skeleton sample of “boxing”. This makes the regulation of attention weights for different action categories possible. In addition, multiple sets of joints may be discriminative for different samples of a same class of actions. For example, one subject may perform the “hand waving” with their left hand while another one may perform it with their right hand. Therefore, while similarities exist in the attention weights for one action class, there may also be differences, which should also be considered in the attention regularization.

Second, the general principle that the deeper the network the better in extracting discriminative features is hardly implementable using a conventional RNN, such as the Vanilla RNN and long short-term memory (LSTM), due to the notorious gradient vanishing and exploding problems. Attention based RNNs for action recognition usually only include one or two fully connected layers to obtain attention and one or two LSTM layers for the classification as in [102], [103]. Such shallow networks are hardly able to explore the long range dependency both temporally and spatially and a deep (e.g. multiple layers) RNN is expected to improve the performance as observed in the last chapter. In addition, one fully connected layer in estimating the attention weight tends to trap the end-to-end training to a local optimum as shown in the experiments. Such a local optimum issue cannot be resolved by the double stochastic attention regularization [101] which aims to encourage the model to pay equal attention to every joints over a sequence of skeletons.

To address these two issues, this chapter proposes

- a new deep attention architecture in which the IndRNN model is adopted to build up a deep RNN for classification and multiple fully connected layers are employed to estimate the attention weights for each joint at each time step. An ablation study has shown that the proposed deep attention architecture provides much more stable and better performance than the shallow counterparts.
• a new triplet loss function to regulate the attention among different action categories. This triplet loss function is further extended with a sample to class distance to enforce the intra-class attention distances to be no larger than the inter-class distances and at the same time to allow different sets of attention weights within the same class.

Experimental results have shown that the proposed deep attention architecture and the new loss function improves significantly the performance of classification and that the attention learned is much more stable compared with the traditional attention models [102], [103]. In addition, the double stochastic attention regularization [101] is no longer required to train the network.

5.2 Related Work

A large number of skeleton-based action recognition methods have been proposed in the literature. Among them, many methods employ the deep learning models, including both the convolutional neural networks (CNN) and the recurrent neural networks (RNN). CNN based methods [104]–[109] usually summarizes the information from all frames into one image and then apply CNN for classification on this single image. On the contrary, RNN based methods [102], [103], [110]–[114] sequentially process the frames and classify the sequences after all the frames are given. In addition to the deep learning based methods, there are also some traditional methods based on handcrafted features and view geometry [115], [116] if multiple views are employed. Since the method proposed in this chapter is a RNN-based model, only related RNN-based methods are reviewed here.

Most of the RNN-based methods employ the long short-term memory network (LSTM) which can better maintain the long-term memory. On top of the original LSTM models, many models have been developed to take advantage of the specific features of skeletons. Considering that body joints move together in groups, in [110], a hierarchical recurrent neural network was proposed where different parts of the body are first processed with different RNNs and then concatenated together for the whole body action. Similarly, in [112], a part-aware LSTM model
was proposed where the LSTM cell is split into different parts for different groups of joints of the body in order to explore the joint groups. Since some skeleton data may be noisy, a trust gate is further added in the LSTM model in [113]. In [117], the co-occurrence of joints is explored by adding a fully connected layer before LSTM to learn joint connections. There are also methods exploring other types of features in the skeleton data. In [114], instead of processing the joint coordinates, a geometric feature was proposed for skeletons to explore the geometric relationships between different joints. In [111], a differential gating scheme was proposed for LSTM, which emphasizes the change in information gain caused by the salient motions between different frames. These methods consider and process all joints equally at all time steps for all actions, which is against the intuition that different joints may contribute differently to the classification of actions.

RNN based methods are recently augmented by the incorporation of attention models [100], [101], [118], [119] to explicitly model the observation that, for different actions, different joints may show different degrees of importance in classification. In [102], an attention model was proposed to assign different weights to different joints at different time steps. An additional temporal weight is assigned to the features obtained at different time steps for the final classification. Since no ground truth is available for the attention, the attention weights are often treated as latent variables and trained by the classification objective. The doubly stochastic attention regularization [101] is most widely used to encourage the model to pay equal attention to every joint over the sequence in order to avoid attention weights only being assigned to one or two joints. In [103], the attention model was also used where the global context over the whole sequence is employed to obtain the attention weights. These attention models are similar as those used in the image-based action recognition [101], where no direct loss is applied to regulate the attention weights other than the doubly stochastic attention regularization, and they often fail to meet the requirement on similar and multiple intra-class attention weights and different inter-class attention weights.

The independent recurrent neural network (IndRNN) proposed in the last chapter provides an effective solution to the gradient vanishing and exploding problem in training a multiple-
layer RNN, which allows deep networks to be constructed and to learn long-term dependency. Specifically, preliminary experiments using multiple layers of the basic IndRNN have shown that better performance than LSTM based networks on the skeleton-based action recognition can be attained. Therefore, IndRNNs are adopted in this chapter to construct a deep attention network for action recognition and a new regularization is developed to train the network.

5.3 Proposed Method

5.3.1 IndRNN-based Deep Attention Model

In this chapter, the independently recurrent neural network (IndRNN) is used as a basic RNN component to construct a deep RNN for classification to leverage IndRNN’s capability of learning deeper and longer features than LSTM.

Fig. 5.1 shows the framework of the IndRNN based deep attention model for skeleton-based action recognition. It consists of a main classification network and an attention network. The main classification network is composed of several IndRNN layers, and batch normalization layers are inserted after each IndRNN (ignored in Fig. 5.1 for simplicity) layer. Residual connections are also used to further facilitate the gradient propagation across layers and each residual block consists of two IndRNN layers. A fully connected layer (FC layer) is added at the final time step for classification (also ignored in Fig. 5.1 for simplicity). Due to the use of hidden states at each time step to obtain the attention, the statistics for the batch normalization is estimated for each time step, while the parameters for the affine mapping is shared over time.

The attention for each time step ($s_t$) is obtained using an attention network based on the current input and the hidden state of the last IndRNN layer at the previous time step. Unlike the conventional attention network that consists of only one fully connected (FC) layer, a multiple-layer network is proposed as the attention network to avoid the issue that the conventional one-layer attention network cannot robustly estimate attention leading to the attention being mainly placed on joints with larger movements. Specifically, since the hidden state of the last IndRNN layer captures the high level information after going through several IndRNN layers, a few
CHAPTER 5. APPLICATION TO SKELETON BASED ACTIVITY RECOGNITION

Figure 5.1: The framework of the spatial IndRNN based attention models for skeleton-based action recognition. The attention network obtains the attention based on the last hidden state and the current input. The main classification network is a deep residual IndRNN network where each residual block contains two IndRNN layers.
CHAPTER 5. APPLICATION TO SKELETON BASED ACTIVITY RECOGNITION

Figure 5.2: Illustration of the attention network used to process the input and the last hidden state from IndRNN.

FC layers are first used to extract efficient features of the input skeleton before concatenating it with the hidden state of IndRNN. Fig. 5.2 illustrates the attention network, where residual networks are used with residual connections across every two FC layers. The residual network used to extract the features of the input skeleton, referred to as feature extractor, consists of 5 FC layers, while the residual network used to process the concatenated features, referred to as attention estimator, consists of 4 FC layers. Details on the architecture will be further explained in Section 5.4. A softmax function over all the joints is added at the end to produce the attention weights.

Let the input to the IndRNN net be $x'_t = (x'_{t,1}, x'_{t,2}, ..., x'_{t,K})$ with $x'_{t,i} = s_{t,i} \cdot x_{t,i}$ where $x_{t,i}$ and $s_{t,i}$ represent the feature and attention weight of the $i$-th joint. Cross-entropy loss is used as the classification objective. Conventionally, the doubly stochastic attention regularization [101], [102] would be used to regularize the attention weights. However, the proposed deep attention model can be trained robustly without this regularization term and thus is not employed in the training probably because of the deeper networks for classification and attention estimation.
address the first issue discussed in the introduction, a new loss function, referred to as *Triplet Attention Loss*, is developed to guide the learning of attention weights.

### 5.3.2 Triplet Attention Loss

It is known that different joints can be of different degrees of importance to different action classes. Therefore, the attention weights for different joints are different for different action classes. For example, “Kicking” mostly focuses on the leg while “Punching” mostly focuses on the hand. Accordingly, the attention weights assigned to the informative joints are supposed to be larger than others, and the attention weights for samples from one class are supposed to be more similar than those from different classes. However, this cannot be achieved in the current attention model framework learned completely by the final objective since there are no direct constraints on the attention.

To address the above problem, a new triplet loss function is proposed to guide the learning of the attention weights in addition to the final classification objective. This new loss function enforces the intra-class attention distance to be smaller than the inter-class attention distance. As mentioned before, for skeleton based attention models, the attention weights are assigned to the joints and the mapping between attention and joints is fixed. Therefore, for different samples in the same class and different classes, the attention distance over the joints can be calculated and thus the triplet loss can be implemented. The details on the new triplet loss functions are defined as follows.

For a skeleton sample $v_i$ (*anchor*), let the attention weights on the sample at time step $t$ be $s_{a_i}^t$. A sample from the same action class represents the *positive* sample whose attention weights is $s_{p_i}^t$, and one sample from different action classes represents the *negative* sample whose attention weights by $s_{n_i}^t$. The following constraint stands.

$$||s_{a_i}^t - s_{p_i}^t||_2^2 + \alpha < ||s_{a_i}^t - s_{n_i}^t||_2^2$$

(5.1)

where $\alpha$ is a margin that is enforced between positive and negative pairs. This constrains the intra-class attention distance to be smaller than the inter-class attention distance by at least $\alpha$. 

Figure 5.3: Illustration of the new triplet loss with sample to class distance.

Accordingly, a triplet loss \( L_{tri,t} \) [120], [121] for the spatial attention in each frame can be defined as

\[
L_{tri,t} = \sum_i N \left[ \| s_{a,t} - s_{p,t} \|_2^2 - \| s_{a,t} - s_{n,t} \|_2^2 + \alpha \right] + \quad (5.2)
\]

where \( N \) is the number of samples. When the triplet loss function is used to obtain embeddings for different classes [120], [121], \( \alpha \) is set to a positive value to avoid the trivial solution that embeddings for different classes are the same. However, in the attention models for skeleton-based action recognition, the attention weights are guided by the class classification objective in addition to the above triplet loss. Therefore, \( \alpha \) can be set to zero, which only encourages the distance between the attention weights of anchor and the positive sample to be no larger than that between anchor and the negative sample. This allows different action classes to share similar attention weights.

On the other hand, since multiple sets of attention weights may exist for one action as mentioned in the introduction, the direct implementation of the above triplet loss function may not be appropriate as it may reduce the number of plausible attention weights for one action. To overcome this issue, a new triplet loss is proposed based on a sample to class distance, instead of the distance between the attention weights of anchor and one positive.

Fig. 5.3 illustrates the new triplet loss. The anchor to the positive class distance is defined as
the minimum distance between anchor and multiple positive samples, \( \min_{m=1,2,...,M} ||s^a_{i,t} - s^{p_m}_{i,t}||^2_2 \), where \( M \) is the number of samples used in the positive class. Through learning, the attention of the anchor sample is pushed to be closer to the attention of the positive classes. In this way, the attention weights for the anchor only needs to be closer to any sample in the positive class than the negative sample, which allows for multiple sets of attention weights for each action. This also compensates, to some extent, the differences of attention (if there is any) due to the styles of performing the actions or other factors.

To reduce the computation, the positive class is represented by a few randomly selected positive samples. Accordingly, the new triplet loss \((L_{tri,t})\) follows

\[
L_{tri,t} = \sum_{t}^N \left[ \min_{m=1,2,...,M} ||s^a_{i,t} - s^{p_m}_{i,t}||^2_2 - ||s^a_{i,t} - s^n_{i,t}||^2_2 + \alpha \right]_{+} 
\]  

(5.3)

Assuming the probability of the attention set used by the current anchor sample is \( p_a \), the probability of at least one sample in the selected positive class samples sharing the attention set as the current anchor sample is \( 1 - (1 - p_a)^M \). Comparing with the probability \((p_a)\) of drawing one random positive sample sharing the attention set, the probability is higher with \( M > 1 \), and much higher when \( M \) is large.

Notice that different action samples may be not well aligned in the time domain in practice because of varying starting point and speed of an action being performed. Accordingly, the frame-by-frame distance between the anchor sample and the positive sample could be large even if they may share similar attention weights on some key frames. This issue has been widely studied in the literature and could be addressed with preprocessing the sequences by time dynamic warping if the increased computation complexity is affordable. In most of the training datasets for action recognition, each video sample is a short video clip containing a single action. In the training process, each video is first divided to \( T \) sub-sequences with the same length and one frame is randomly selected from each sub-sequence [112]. The attention weights can be averaged over a few frames to make each segment roughly aligned. Since the triplet loss function is only used in training, this processing does not affect the testing.
In all, the final objective function used for training is as follows.

\[ L = L_c + \lambda_2 \sum_{t=0}^{T} L_{tri,t} \]  

where \( L_c \) represents the class classification loss using the typical cross-entropy loss \( -\sum_{i=1}^{C} y_i \log \hat{y}_i \), where \( y_i \) and \( \hat{y}_i \) are the groundtruth label and the predicted label, respectively). \( \sum_{t=0}^{T} L_{tri,t} \) represents the triplet loss on the attention over time where \( T \) is the length of the sequence.

### 5.4 Experiments and Analysis

The proposed deep attention model have been evaluated in three widely used datasets, i.e. the NTU RGB+D dataset [112], SBU Kinect Interaction Dataset (SBU) [122] and Berkeley MHAD Dataset [123], which covers a wide range of actions. The algorithms are implemented using PyTorch [124], and run on GPU (Titan XP).

#### 5.4.1 Results on NTU RGB+D dataset

The NTU RGB+D dataset [112] is currently the largest available action recognition dataset with skeletons. It contains 56880 sequences of 60 action classes. It was collected by three Kinect v2 cameras with 17 different setups. Two evaluation protocols are suggested for this dataset including Cross-Subject (CS) and Cross-View (CV) settings. In the training, 5\% of the training data was reserved as evaluation data as suggested in [112]. Two skeletons (25 joints per skeleton) were used as input and if only one is present in the sample, the second was set as zero. For this dataset, when multiple skeletons are present in the scene, the skeleton number captured by Kinect may be changed over time, especially when the number of the skeletons is changed. Therefore, an alignment process, by comparing the distance of all the joints between different skeletons, is first applied to keep the same number assigned to the skeleton of the same subject. This is only performed once as preprocessing and the processed skeleton data was used for training and testing the network. For both training and testing, each sequence was first divided to 20 segments of the same length and one frame is randomly selected from each
The hyperparameters of the deep attention based IndRNN model used in the experiments are as follows. For the main IndRNN classification network, seven IndRNN layers with residual connections are used and each layer contains 512 neurons. Five and four layers are used for extracting the features of the input skeleton and processing the concatenated features in the attention network respectively. The joint coordinates of two persons (of dimension $25 \times 2 \times 3$) are used as input. The batch size was set to 128. The Adam optimization is used with the initial learning rate $2 \times 10^{-4}$ and decayed by 10 once the evaluation accuracy does not increase (with patience 20). Dropout is applied after each layer with a dropping probability of 0.45 and 0.3 for the CS and CV settings, respectively.

The proposed triplet loss function was evaluated using the above deep attention network. The overall performance of the proposed model in comparison with the existing methods is shown in Table 5.1, where the proposed deep attention based IndRNN model is denoted by DA-IndRNN. It can be seen that the proposed model significantly improves the performance over the traditional attention methods.

**Evaluation of the deep attention network**

It is known that the skeleton data can be noisy. Therefore, in addition to the hidden state of the previous time step, the current input information is also used as input to obtain the attention as shown in Section 5.3.1. Moreover, we show that it is important to explore relatively deep features to provide robust performance. Fig. 5.4 shows comparisons among different configurations of the feature extractor and attention estimator for the attention network. Each figure shows the performance over 10 training processes of the corresponding configuration of the attention network. It can be clearly seen that with only a one-FC-layer attention estimator and without feature extractor, the training process is likely to be trapped to a local optimum with poor performance as shown in Fig. 5.4(a). With the number of layers in the attention estimator and the feature extractor increases, the training process becomes stable and less likely to be trapped to a local optimum. For a network with a four-FC-layer attention estimator and
**Table 5.1:** Results of the skeleton based action recognition on NTU RGB+D dataset in comparison with the existing methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>CS</th>
<th>CV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep learning on Lie Group [125]</td>
<td>61.37%</td>
<td>66.95%</td>
</tr>
<tr>
<td>JTM+CNN [104]</td>
<td>73.40%</td>
<td>75.20%</td>
</tr>
<tr>
<td>Res-TCN [106]</td>
<td>74.30%</td>
<td>83.10%</td>
</tr>
<tr>
<td>SkeletonNet(CNN) [105]</td>
<td>75.94%</td>
<td>81.16%</td>
</tr>
<tr>
<td>JDM+CNN [108]</td>
<td>76.20%</td>
<td>82.30%</td>
</tr>
<tr>
<td>Clips+CNN+MTLN [107]</td>
<td>79.57%</td>
<td>84.83%</td>
</tr>
<tr>
<td>Enhanced Visualization+CNN [109]</td>
<td>80.03%</td>
<td>87.21%</td>
</tr>
<tr>
<td>1 Layer RNN [112]</td>
<td>56.02%</td>
<td>60.24%</td>
</tr>
<tr>
<td>2 Layer RNN [112]</td>
<td>56.29%</td>
<td>64.09%</td>
</tr>
<tr>
<td>1 Layer LSTM [112]</td>
<td>59.14%</td>
<td>66.81%</td>
</tr>
<tr>
<td>2 Layer LSTM [112]</td>
<td>60.09%</td>
<td>67.29%</td>
</tr>
<tr>
<td>1 Layer PLSTM [112]</td>
<td>62.05%</td>
<td>69.40%</td>
</tr>
<tr>
<td>2 Layer PLSTM [112]</td>
<td>62.93%</td>
<td>70.27%</td>
</tr>
<tr>
<td>JL_d+RNN [114]</td>
<td>70.26%</td>
<td>82.39%</td>
</tr>
<tr>
<td>ST-LSTM + Trust Gate [113]</td>
<td>69.20%</td>
<td>77.70%</td>
</tr>
<tr>
<td>GCA-LSTM [103]</td>
<td>74.4%</td>
<td>82.8%</td>
</tr>
<tr>
<td>STA-LSTM [102]</td>
<td>73.40%</td>
<td>81.20%</td>
</tr>
<tr>
<td>IndRNN</td>
<td>81.80%</td>
<td>87.97%</td>
</tr>
<tr>
<td>Proposed DA-IndRNN</td>
<td>82.80%</td>
<td>88.40%</td>
</tr>
<tr>
<td>Proposed DA-IndRNN with the attention loss</td>
<td>83.24%</td>
<td>88.70%</td>
</tr>
</tbody>
</table>
a five-FC-layer feature extractor, the training converges almost monotonically with improving performance.

**Comparison of the DA-IndRNN against the shallow LSTM based attention model**

To demonstrate the advantage of the proposed model against a shallow LSTM based attention model [102], the difference between the normalized confusion matrices (%) of the proposed model minus that of the shallow LSTM based attention model is shown in Fig. 5.5(a). Since the number of classes (60) is too large to show the entire difference matrix, Fig. 5.5(a) only shows part of the difference confusion matrix. The 10 rows represent the classes with the 10 largest differences between the proposed model and the shallow LSTM based attention model. The columns represent the classes that are confused with one of the classes in the rows and the difference between the proposed model and the shallow LSTM based attention model is at least 2%. The positive values of the main highlighted diagonal elements (where the true label equals the predicted label) represent the improvement in percentage points achieved by the proposed model compared to the shallow model for these classes. The negative values of the other elements represent that error in percentage points reduced by the proposed model. From the figure, it can be seen that the proposed model significantly improves the performance of these classes with up to 29 percentage points, and reduces the confusion among classes, especially for classes performed with same joints but small differences, such as Class 13 (“teat up paper”) and Class 11 (“reading”), or Class 31 (“pointing to something with finger”) and Class 32 (“taking a selfie”), both involving the movement of the hands.

To further show detailed performance of the proposed model, Fig. 5.5(b) shows part of the confusion matrix of the proposed model. The top 10 most confused classes are shown in the 10 rows. The columns represent the classes that are confused with one of the classes in the rows by at least 2%. The values of the main highlighted diagonal elements represent the accuracy of the proposed model, and the larger the better. The values of other elements represent the errors and the smaller the better. Despite the performance improvement of the proposed model, it seems that the proposed model still suffers from distinguishing the order of each movement.
Figure 5.4: Performance comparison of different attention models obtained with 10 training processes on the CS setting of the NTU RGB+D dataset [112]. Fig. 5.4(a), 5.4(b), 5.4(c) and 5.4(d) represent the attention model using a one-FC-layer attention estimator with no feature extractor, a one-FC-layer attention estimator with a one-FC-layer feature extractor, a two-FC-layers attention estimator with no feature extractor, and a two FC layer attention estimator with a one-FC-layer feature extractor, respectively. Fig. 5.4(d) represents the proposed deep attention model using a four-FC-layer attention estimator with a five-FC-layer feature extractor.
Taking Class 16 (“wear a shoe”) and Class 17 (“take off a shoe”) for example, these two classes with similar movements but different orders cannot be classified well by the proposed model. Therefore, RNN models with explicit considering of the orders may be needed, which will be investigated in the future.

**Verification of the learned attention**

Fig. 5.6 shows the attention on the joints of hands and feet over time for different classes. The feet are represented by joints 15, 16, 19 and 20 (left ankle, left foot, right ankle and right foot), and hands include joints 7, 8, 22, 23, 11, 12, 24 and 25 (left wrist, left hand, tip of the left hand, left thumb, right wrist, right hand, tip of the right hand and right thumb). The average percentages of the joints being actively focused (where the joint attention weight is larger than 0.04 which represents the case that all 25 joints of a skeleton are equally weighted if the second person is not attended at all) are shown. Fig. 5.6(a) shows the attention on the joints of feet over time for action “kicking something” and action “hand waving”. It can be seen that for “kicking something”, with the progress of the actions in time, more joints of feet are being focused, until it reaches the midpoint of the action and towards the end of the action, it starts to shift back to the original state. On the other hand, for “hand waving”, the attention on the joints of feet decreases over time. This agrees with the intuition that “kicking something” focuses on the joints of feet while “hand waving” does not. Fig. 5.6(b) shows the attention on the joints of hands over time for action “playing with phone/tablet” and action “kicking something”. Similar behaviour as Fig. 5.6(a) can be observed where attention on the joints of the hands improves for “playing with phone/tablet”.

To further illustrate that even for one action class, the attention patterns may vary for different samples due to different performing styles, Fig. 5.7 further shows two examples of “playing with phone/tablet” performed using left and right hands, respectively. It can be clearly seen that when the action is performed by different hands, the attention on the joints of different hands are clearly different. When playing using left hand, the attention on the left hand is much larger than that of playing using right hand, and the attention on the right hand is accordingly much
CHAPTER 5. APPLICATION TO SKELETON BASED ACTIVITY RECOGNITION

(a) Difference of the confusion matrix between the proposed model and the shallow LSTM based attention model.

Figure 5.5: Illustration of the normalized confusion matrices (%) for the proposed model and its difference with the shallow LSTM base attention model on the NTU dataset. Note that there are 60 classes in total, which is too large to show the entire difference matrix. Therefore, only part of the difference confusion matrix is shown. For the difference of the confusion matrix (confuse matrix of the proposed model minus that of the shallow LSTM based attention model), the 10 rows represent the classes with the 10 largest differences between the proposed model and the shallow LSTM based attention model. The columns represent the classes that are confused with one of the classes in the rows and the difference between the proposed model and the shallow LSTM based attention model is at least 2%. For the confusion matrix of the proposed model, the top 10 most confused classes are shown in the 10 rows. The columns represent the classes that are confused with one of the classes in the rows by at least 2%.
(a) Progress of the attention on the joints of feet over time for “hand waving” and “kicking something”

(b) Progress of the attention on the joints of hands over time for “playing with phone/tablet” and “kicking something”

Figure 5.6: Comparison of the attention on the joints of feet and hands over time for different action classes.
smaller than that of playing using right hand. This indicates that with different performing
styles, different attention patterns exist and the proposed model is able to accommodate such
variation well.

One of the objectives of the triplet loss functions developed for the attention is to make the
attention weights more consistent for the same actions, but at the same time to allow them to
vary within a certain degree. Therefore, the distance of the attention weights between different
samples in the same class is presented in Fig. 5.8. It can be seen that the attention distance
regulated by the proposed triplet loss is smaller than that of the traditional attention. Also, the
variance of the distances obtained by the proposed triplet loss is smaller (i.e. consistency for
the same actions), but variations does exist (i.e. accommodation of different performing styles
of the same actions).

5.4.2 Results on SBU Kinect Interaction Dataset (SBU)

The SBU Kinect Interaction Dataset (SBU) [122] contains 8 classes of two-person interaction.
It includes 282 skeleton sequences. The experiment protocol of the 5-fold cross validation
as suggested in [122] was used. The data was processed as in [102], [110], [117] using the
Svaitzky-Golay filter to smooth each joint’s position in the temporal domain. We further nor-
malized the coordinates of the joints using the mean and variance of all the coordinates of all
joints. This scales the coordinates into a range for processing by IndRNN. Since this dataset is
relatively small, a four-layer network was used as the main classification network, and one FC
layer and two FC layers were used for feature extractor and attention estimator, respectively.
The batch size was set to 16. The training was performed in the similar way as in [113]. Specif-
ically, Adam was used to optimize the training and the initial learning rate was set to $6 \times 10^{-4}$
and decayed by a factor of 0.99. Since this data is small compared with the NTU RGB+D
dataset [112], only the IndRNN model and the DA-IndRNN are tested. The performance is
shown in Table 5.2, which demonstrates that the proposed method achieved better performance
than the existing spatio-temporal attention models.
(a) Progress of the attention on the left hand joints over time when the action “playing with phone/tablet” is performed by the left (red) and right (blue) hand respectively.

(b) Progress of the attention on the right hand joints over time when the action “playing with phone/tablet” is performed by the left (red) and right (blue) hand respectively.

Figure 5.7: Comparison of the attention on the joints of left and right hands over time for “playing with phone/tablet” performed by left and right hands.
CHAPTER 5. APPLICATION TO SKELETON BASED ACTIVITY RECOGNITION

5.4.3 Results on Berkeley MHAD Dataset

Berkeley MHAD Dataset [123] includes 659 sequences from 11 actions. The experiment protocol in [123] was used for evaluation. Specifically, 384 sequences corresponding to the first 7 subjects were used for training and 275 sequences of the remaining 5 subjects were used for testing. The data preprocessing and the training process is the same as the one used above. Our method achieves accuracy of 100%.

Table 5.2: Results on SBU dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>HBRNN [110] (reported by [117])</td>
<td>80.4%</td>
</tr>
<tr>
<td>Deep LSTM (reported by [117])</td>
<td>86.0%</td>
</tr>
<tr>
<td>Co-occurrence LSTM [117]</td>
<td>90.4%</td>
</tr>
<tr>
<td>ST-LSTM + Trust Gate [113]</td>
<td>93.3%</td>
</tr>
<tr>
<td>GCA-LSTM [103]</td>
<td>94.1%</td>
</tr>
<tr>
<td>STA-LSTM [102]</td>
<td>91.51%</td>
</tr>
<tr>
<td>IndRNN</td>
<td>92.29%</td>
</tr>
<tr>
<td>Proposed DA-IndRNN</td>
<td>95.19%</td>
</tr>
</tbody>
</table>

Figure 5.8: Distance of the attention weights between different samples in the same class.
5.5 Summary

This chapter presents a new IndRNN based deep attention model, termed as DA-IndRNN, for skeleton-based action recognition to effectively model the facts that different joints are usually of different degrees of importance to different action categories. The model consists of (a) a deep IndRNN as the main classification network to overcome the limitation of a shallow RNN network in order to obtain deeper and longer features, and (b) a deep attention network with multiple fully connected layers to estimate reliable attention weights. To train the DA-IndRNN, a new triplet loss function is proposed to guide the learning of the attention among different action categories. Specifically, this triplet loss enforces intra-class attention distances to be smaller than inter-class attention distances and at the same time to allow multiple attention weight patterns exist for a same class. The proposed DA-IndRNN can be trained end-to-end. Experiments on the widely used datasets have demonstrated that the proposed method can achieve better and stable performance than the state-of-the-art attention models.
Chapter 6

A Fusion Framework for Camouflaged Moving Foreground Detection in the Wavelet Domain

6.1 Introduction

Despite the success of computer vision methods shown in the literature, most of them deal with the relatively easy cases where the background is relatively simple. However, there are cases of camouflage in color [126] where the person shares a similar color as the background. For example, in Fig. 6.1, a person wears clothes that have similar colors as the background wall. In such a case, the intensity difference between the person and background are very small and the person cannot be easily detected by the existing methods, which greatly increases the difficulty of the following processing based on the person. Considering that moving foreground object detection is an important pre-processing step for further in-depth analysis in many video processing and computer vision systems including activity recognition such as those based on silhouette in [127], in this chapter we investigate the moving foreground object detection problem in camouflage scenes to explore possible computer vision applications in complex scenes. With reasonable performance of this foreground detection task in complex scenes, decent performance on complex tasks such as activity recognition can be expected.

A commonly used approach for foreground detection is background subtraction, where a
background model is first formulated and then the foreground object is detected by comparing the current frame and the background model. There are many background subtraction methods [128]–[140] in the literature. Most of them are developed under the assumption that foreground and background show visually distinct characteristics and thus the foreground can be detected once a good background model is obtained.

However, as mentioned above, there are camouflage cases where the intensity differences between the foreground and background are very small and the foreground cannot be easily detected by the existing methods. This chapter addresses the problem of “camouflage in color” and it is simply referred to as “camouflage” in the rest of the chapter.

6.2 Related work

Over the past decades, many methods have been reported for background subtraction, including statistical models, fuzzy models, neural models, subspace models, low rank models, sparse models and transform domain models. Here we only review the most relevant works including the statistical background models especially the Gaussian mixture models (GMM), the transform domain methods especially those employing the wavelet transform, and the existing background models specifically designed for camouflaged scenes. Other approaches, which go beyond the scope of this chapter, can be found in several survey papers [128]–[140].

6.2.1 Statistical Background Models

One of the most popular background modelling methods is the Gaussian mixture models (GMM) [141], [142]. It models the intensity distribution of each pixel using a summation of weighted Gaussian distributions. Specifically, a predefined number of Gaussian components are used [143], and the Gaussian components are ordered based on the weight and the variance. The first $B$ components with total probabilities over a defined threshold are used as the current background model. When a new pixel is processed, it is considered to be a foreground pixel if its value cannot be well described by the background Gaussian distributions. These distributions are then updated with the current pixel using an online expectation-minimization algorithm. It
is also known as MOG in openCV [144]. In [145], an improved adaptive Gaussian Mixture
Model was proposed which can adaptively estimate the number of the Gaussian components
needed, known as MOG2 in openCV. In [146], 3D multivariate Gaussian distributions are used
to formulate a multiple layers model and the mean and variance of the distributions are obtained
based on a recursive Bayesian learning approach.

In addition to the above statistical models based on parametric distributions, there are also
methods using non-parametric models such as kernel density estimation [147]–[153]. The ker-
nel density estimation (KDE) based methods [148] estimate the probability density function for
each pixel from many recent samples (N) over time using a kernel function. These methods can
be memory consuming considering that N frames have to be kept in memory. Many improve-
ments have been made on the KDE methods, including changing the kernel function. In [149],
a sequential kernel density approximation method was proposed based on the mean-shift mode
finding algorithm and the density modes are sequentially propagated over time. In [153], the
foreground is also modelled and further tracked over time based on a particle filter. A selective
analysis strategy was further proposed to reduce the computation. Other than the above KDE
based methods, there are also non-parametric methods [154]–[156] using a history of recently
observed pixel values for background modelling instead of using the distribution of the pix-
els. The foreground decision is made based on the number of the pixels in the background set
that the current pixel is close to. Different background update rules have been proposed in the
literature. In SACON (sample consensus) [154], the history of background image is updated
using a first-in first-out strategy, while in ViBe (visual background extractor) [155] and PBAS
(pixel-based adaptive segmenter) [156], the history is updated by a random scheme based on
fixed or adaptively changed randomness parameters.

Another category of statistical models employ subspace learning methods [157], [158]. In
[157], subspace learning using principal component analysis (SL-PCA) is applied on N images,
and the background model is represented by the mean image and the projection matrix compris-
ing the first p significant eigenvectors of PCA. The foreground is obtained by computing the
difference between the input image and its reconstruction. Recently, lots of methods have been
CHAPTER 6. EXPLORATION OF APPLICATIONS IN COMPLEX SCENES

proposed to formulate the background model via a robust subspace using a low-rank and sparse
decomposition such as robust principal component analysis [159]. The background is modelled
by the low-rank subspace that gradually changes over time while the foreground consists of the
correlated sparse outliers. In order to consider the spatial connection of the foreground sparse
pixel, in [160], a class of structured sparsity-inducing norms were introduced to model the mov-
ing objects in videos. In [161], a framework named detecting contiguous outliers in the low-rank
representation (DECOLOR) was proposed where the object detection and background learning
are integrated into a single optimization process solved by an alternating algorithm. In [162],
an online sequential framework named contiguous outliers representation via online low-rank
approximation (COROLA) was proposed to detect moving objects and learn the background
model at the same time. It works iteratively on each image of the video to extract foreground
objects by exploiting the sequential nature of a continuous video of a scene where the back-
ground model does not change discontinuously and can therefore be obtained by updating the
background model learned from preceding images.

The above approaches mainly focus on constructing the background models from the inten-
sity of pixels. There is also considerable research carried out to investigate new features to assist
in background subtraction. In [163], local binary patterns (LBP) [164] was used to formulate
the background model in order to consider the local texture. In [165], the intensity and LBP
are used together to formulate the background model based on PBAS [156]. In [166], multiple
features, including intensity, gradient, and Haar-like features, are combined for the foreground
detection. In [167], the SILTP texture (an extension of LBP) in addition to the gray and color
features is used. However, these kinds of features are often effective for a certain type of texture
in a small predefined region that the current pixel is located in. When the current pixel is located
in a relatively flat or poor texture region, it may fail and thus the background subtraction based
on these features may not work as desired.

Other than the above models, there are also methods based on support vector machines and
other approaches which can be found in several survey papers [128]–[140] as mentioned in the
beginning of this Section.
6.2.2 Transform Domain Background Models

Another class of background models are the transform domain models where the background formulation and/or foreground detection is performed in a different domain. Methods using different transformations have been proposed in the literature, including the wavelet transform, the fast Fourier transform [168], the discrete cosine transform [169], [170], the Walsh transform [171] and the Hadamard transform [172]. Here we focus on reviewing the methods using wavelet transform.

Wavelet transform has been widely used in computer vision tasks as it may capture some desired characteristics in the frequency domain. It decomposes the signal into different levels of wavelet frequency bands, which allows for multi-resolutional analysis. There have been a few methods in the literature using the wavelet transform for background subtraction. In [173], a moving object segmentation method was developed in the wavelet domain, where the change detection method with different thresholds is used in four wavelet bands. The canny edge operator is applied on the results to get the edge maps of the bands. The edge result of the current frame is obtained as the union of different edge maps or the inverse transform of the different edge maps, and the foreground is detected as the video object planes based on the edges. This was further improved in [174] by using a double edge detection method. In [175], the Daubechies complex wavelet transform is used and the foreground edges are obtained using the double change detection method similarly as in [174]. The final detection results are obtained by post-processing the edge map with some morphological operations. This method was further improved in [176], where the approximate median filter is applied to detect the frame difference. In [177], an undecimated wavelet transform is used and the foreground is detected in each wavelet band by frame differencing. The final foreground is obtained as the union of the detection results from all the bands. This method was further improved in [178], [179] where, instead of using frame differencing for foreground detection in each wavelet band, a modified z-score test was used. The moving foreground object is detected as outliers. The above methods generally detect the foreground based on two or three consecutive frames using techniques such as frame differencing without maintaining a background model.
In [180], the orthogonal non-separable wavelet is used where the approximation coefficient is used to construct the background model in order to filter out the noise. A running average scheme is used to maintain the background model when the background has a gradual change, while when there is a sudden change, the background is replaced by the current frame. In [181]–[183], a background model based on Marr wavelet kernel was proposed and the foreground was detected in each sub-band based on binary discrete wavelet transforms. Specifically, the background model keeps a sample of intensity values for each pixel in the image and uses this sample to estimate the probability density function of the pixel intensity. The density function is estimated using the Marr wavelet kernel density estimation technique. In [184], [185], the Daubechies Complex Wavelet Transform was used and the background model is formulated based on the low frequency wavelet bands. The foreground is determined by comparing the difference between the current frame and the background model. In [186], the background model is first formulated using the low frequency wavelet bands in the discrete wavelet transform, then the inverse transform is used to obtain the background model in the image domain. The foreground is detected by comparing the difference between the background and the current frame. In [187], a modified directional lifting-based 9/7 discrete wavelet transform structure was proposed in order to reduce the computation and preserve fine shape information in low resolution images. The low frequency bands are used for background modelling and the double change detection method is used to detect the foreground. In [188], the high frequency wavelet bands are used as a feature for foreground decision and the low frequency wavelet bands are used for the background modelling with an improved GMM model. The above methods generally construct the background model based on the low frequency wavelet bands to reduce the effect of the noise.

In [189], a foreground detection method was proposed to process wavelet compressed video. The background model and foreground detection is implemented for each wavelet band. A running average scheme is used to formulate the background and the foreground in each band is determined by comparing the difference between the current band image and the background. Finally, the foreground of the image is detected as the union of the detected foreground in all
wavelet bands. In [190], the HSV (hue, saturation, value) components are used as features and the value component is used for foreground detection while the saturation component is used to suppress moving shadows. The foreground is detected by comparing the difference between the current frame and the background model, and the threshold used is determined based on the standard deviation of the difference image after the wavelet transform. The region connectivity with chromatic consistency is further used in the background update to overcome the aperture problem in [191]. This method was further improved in [192], where the RGB color space is used and the foreground segmentation is further developed based on the motion variation and chromatic characteristics. In [193], [194], a color ratio difference was further adopted to suppress shadow. The foreground classification is further developed in [195] by fusing mode filtering, connectivity analysis and spatial-temporal correlation. In [196], a region based background model was proposed where the background for each region is modelled with GMM in the wavelet domain. The distance between the current frame and the background model is calculated as the summation of the distance in different wavelet bands. Accordingly, the background model is updated with the new distance and so is the foreground detection. The above methods generally construct the background models in all wavelet bands or construct one background model based on all the wavelet bands.

The existing wavelet transform based methods generally only take advantage of the noise resilience capability of the transform in order to construct a reliable background model or detect reliable foreground to lessen the effect of noise. Generally only one or two levels of wavelet decomposition are employed. The foreground result is usually obtained as the union of the results in different bands. Compared with these methods, our approach targets the foreground detection in camouflaged scenes where the wavelet transform is used to reveal detailed differences in different frequencies. Usually multiple levels (6 for example) of wavelet transform are required to show the small differences in the camouflaged scene as demonstrated in the next Section. Also we propose a fusion scheme to efficiently combine the results from different bands based on the characteristics of the wavelet transform.
6.2.3 Background Models for Camouflaged Scenes

The foreground detection methods discussed above are developed to deal with the general foreground objects which show distinct intensity or texture changes. However, there are situations where the foreground objects may share similar intensity and texture as the background, such as the camouflaged foreground, especially those with poor texture as shown in Fig. 6.1. These cases pose great challenges to foreground detection and needless to say, methods that can properly handle such cases are highly desirable for robust foreground detection. Currently there are only a few methods developed for such cases. Here we only review the existing methods specifically designed for the camouflaged cases. Note that some methods working on regular scenes may also work in the camouflaged cases to some extent, but only the methods with a specific component designed for the camouflaged cases are described in this Subsection.

In [197], a post-processing phase was used after foreground detection to recover camouflaged errors based on object detection. The paper focused on the detection of the moving people and the method is limited by the performance of person detection. Similarly in [198], the object information is also used and formulated over time using a Markov random field model, which assists the detection of the camouflaged part of the object in the following frames. In [199], a camouflage model was proposed, which formulates spatial global models on the foreground and background in addition to the pixel models over time. It can also be regarded as building a model of the foreground object. All these methods require part of the object always being non-camouflaged or the object being non-camouflaged when the object first moves into the scene, in order to detect the object or formulate the global foreground model. These constraints limit their use in practical applications. By contrast, the method proposed in this chapter detects the camouflaged region by highlighting the small differences using wavelet transform.
6.3 Motivation and overview

6.3.1 Motivation

Most of the background subtraction methods work on the cases where foreground object and background show distinct changes in either intensity or texture, but there are camouflage cases such as Fig. 6.1 where the foreground objects share similar color to the background. Fig. 6.2 shows an example of differences in intensity and texture (measured by LBP) between the foreground and background (obtained by MOG2) where the values in the image are properly scaled for display. It can be seen that the differences in the visually similar regions are very
Figure 6.3: Differences of wavelet coefficients between the current image and the background image in each wavelet band at different decomposition levels.
Figure 6.4: Block diagram of the stationary wavelet transform.

small. Consequently the existing methods based on intensity or LBP features may fail in such cases.

By decomposing the signal into different bands, a wavelet transform can capture and separate different characteristics of the signal into each band. The key idea of this chapter is that for different parts of a true foreground there is a detectable difference between the foreground and background in one or more bands. There are many types of wavelets including non-redundant and redundant transforms using Haar wavelet, Daubechies wavelet, etc. In this chapter, we employ the stationary wavelet transform (translation-invariant) [200], [201] based on the Haar wavelet due to its simplicity. The stationary wavelet transform is a non-decimated redundant wavelet transform which contains the coefficients of the $\epsilon$-decimated wavelet transform for every choice of $\epsilon$. Since it is non-decimated, each wavelet band image is of the same size as the original image. Fig. 6.4 shows the block diagram of the 2D stationary wavelet transform. $LL_j$, $LH_j$, $HL_j$ and $HH_j$ represent the low frequency approximation of the signal and the details (high frequency) of the signal in horizontal, vertical, and diagonal directions at level $j$, respectively. $F_j$ and $G_j$ represent the low-pass and high-pass filters, respectively. As shown in Fig. 6.4, the LL band (or the signal if it is the first decomposition level) is first filtered along rows using the low-pass and high-pass filters and then filtered again along columns. Compared with the discrete wavelet transform, there is no decimation (downsample) after the convolution with the filters in the stationary wavelet transform.

Fig. 6.3 shows the difference of wavelet coefficients in each band at different levels between the current frame and background model. It can be seen that although no single band can per-
fectly extract the entire foreground person, different parts of the person are able to be extracted
by a few and different wavelet bands. For example as shown in Fig. 6.3, the LH bands seem to strongly respond to the changes in the body of the person. Compared to the intensity and texture differences in the image domain as shown in Fig. 6.2, we can see that the differences are greatly highlighted in a few wavelet bands (in relatively higher levels such as levels higher than 3), which makes the foreground detection from the visually similar background possible. Note that for cases where foreground and background are both texture-less such as rigid objects with extremely smooth surfaces, the wavelet transform may not effectively highlight the differences between them and thus they may not be detectable. For other cases where the color difference between the foreground and background is small but foreground or background are textural surfaces, such as clothes, the difference between them can become apparent or detectable in one or more wavelet bands as shown in Fig. 6.3.

6.3.2 Overview of the proposed framework

The input image is first decomposed into different levels of wavelet bands. The foreground detection based on the wavelet bands can be considered as a decision making process from
different representations. Therefore, the foreground decision can be made by

\[
D = \begin{cases} 
1 & \text{if } p(F|x_{A,N};x_{H,1},\ldots,x_{H,N};x_{V,1},\ldots,x_{V,N};x_{D,1},\ldots,x_{D,N}) \\
> p(B|x_{A,N};x_{H,1},\ldots,x_{H,N};x_{V,1},\ldots,x_{V,N};x_{D,1},\ldots,x_{D,N}) & \text{else} 
\end{cases} 
\tag{6.1}
\]

where \(x_{d,l}\) represents the wavelet coefficient of the band \(d\) at level \(l\), while \(A, H, V\) and \(D\) represent the “LL”, “LH”, “HL” and “HH” bands, respectively. \(p(F|x_{A,N};x_{H,1},\ldots,x_{H,N};x_{V,1},\ldots,x_{V,N};x_{D,1},\ldots,x_{D,N})\) and \(p(B|x_{A,N};x_{H,1},\ldots,x_{H,N};x_{V,1},\ldots,x_{V,N};x_{D,1},\ldots,x_{D,N})\) represent the probabilities of the pixel belonging to foreground and background, respectively. \(N\) is the decomposition level used in the wavelet transform. The pixel is detected as foreground (\(F\)) when \(D = 1\) and background (\(B\)) otherwise.

According to the Bayesian theorem, the probabilities of the pixel belonging to foreground and background can be further represented as

\[
p(F|x_{A,N};x_{H,1},\ldots,x_{H,N};x_{V,1},\ldots,x_{V,N};x_{D,1},\ldots,x_{D,N}) = \frac{p(x_{A,N};x_{H,1},\ldots,x_{H,N};x_{V,1},\ldots,x_{V,N};x_{D,1},\ldots,x_{D,N}|F)p(F)}{p(x_{A,N};x_{H,1},\ldots,x_{H,N};x_{V,1},\ldots,x_{V,N};x_{D,1},\ldots,x_{D,N}|F)p(F) + p(x_{A,N};x_{H,1},\ldots,x_{H,N};x_{V,1},\ldots,x_{V,N};x_{D,1},\ldots,x_{D,N}|B)p(B)} \tag{6.2}
\]

Foreground can appear in the scene at any time and any location, and the foreground typically occupies less than 50% of the image. However, for simplicity (and as commonly adopted in the literature), we assume the prior foreground and background probabilities to be the same, i.e., \(p(F) = p(B)\). Therefore, with (6.2), (6.1) can be simplified as

\[
D = \begin{cases} 
1 & \text{if } p(x_{A,N};x_{H,1},\ldots,x_{H,N};x_{V,1},\ldots,x_{V,N};x_{D,1},\ldots,x_{D,N}|F) \\
> p(x_{A,N};x_{H,1},\ldots,x_{H,N};x_{V,1},\ldots,x_{V,N};x_{D,1},\ldots,x_{D,N}|B) & \text{else} 
\end{cases} \tag{6.3}
\]

It can be seen that in order to make the foreground decisions, the likelihood functions over
all the wavelet bands need to be obtained first.

Since the “LL”, “LH”, “HL”, “HH” bands capture different characteristics of the signal, they can be regarded as independent from each other. Thus the following can be obtained.

\[
p(x_{A,N}; x_{H,1}, ..., x_{H,N}; x_{V,1}, ..., x_{V,N}; x_{D,1}, ..., x_{D,N}|F) = p(x_{A,N}|F) \cdot p(x_{H,1}, ..., x_{H,N}|F) \cdot p(x_{V,1}, ..., x_{V,N}|F) \cdot p(x_{D,1}, ..., x_{D,N}|F)
\]

\[
p(x_{A,N}; x_{H,1}, ..., x_{H,N}; x_{V,1}, ..., x_{V,N}; x_{D,1}, ..., x_{D,N}|B) = p(x_{A,N}|B) \cdot p(x_{H,1}, ..., x_{H,N}|B) \cdot p(x_{V,1}, ..., x_{V,N}|B) \cdot p(x_{D,1}, ..., x_{D,N}|B)
\]

(6.4)

The foreground detection problem is now reduced to the estimation of the likelihood functions for each type of wavelet bands, which will be shown in the next Section. Considering that the wavelet bands are used as features instead of reconstructing the image, in this chapter, all the “LL” wavelet bands from level 1 to level N are used.

The framework of the proposed method is shown in Fig. 6.5. A wavelet transform is first applied to decompose the input image into different levels of wavelet bands, and background models are formulated for each band. Instead of making foreground decisions for each wavelet band, the likelihood functions of the coefficients being foreground and background are estimated, respectively. Considering the properties of the wavelet transform, the likelihood values calculated on each type of band are fused over different levels, shown as “level fusion” in Fig. 6.5, and then the results of different types of bands are fused, shown as “band fusion” in Fig. 6.5. The final foreground decision is determined based on the fused result. In this chapter, we only consider foreground detection using the grayscale video and thus only one component is used. It can be easily extended to color images.

### 6.4 The proposed method

#### 6.4.1 Foreground and background formulation for each wavelet band

After video frames are decomposed into different wavelet bands, background and foreground models are updated for each wavelet band. In this chapter, the widely used GMM model is used.
In the conventional GMM model, each pixel in the image domain is formulated as a mixture of Gaussian distributions, while here the wavelet coefficients are modelled instead. Similarly as in [145], the GMM model for the wavelet coefficients in each band over time can be represented by

\[
p(x_{d,l}) = \sum_{m=1}^{M} \pi_m N(x_{d,l}; \mu_m, \delta^2_m) \tag{6.5}
\]

where \(d\) and \(l\) represents the wavelet band type and level. \(M\) is the number of Gaussian components, \(\pi_m, \mu_m, \delta^2_m\) are the weight, mean and variance of the \(m\)-th Gaussian component for the wavelet band \(d\) at level \(l\) (where the subscript \(d,l\) is omitted for simplicity). The samples may contain both foreground objects and background. Usually, if the foreground object presents in the scene, it will be captured by the distribution as a new Gaussian component with a small weight (\(\pi_m < 0.1\) for example). Since the Gaussian components are ordered by the magnitude of the weight, the background model can be approximated by the first \(B\) largest Gaussian components, which is

\[
p(x_{d,l}|B) \approx \sum_{m=1}^{B} \pi_m N(x_{d,l}; \mu_m, \delta^2_m).
\]

For natural images, foreground objects are generally assumed to follow a uniform distribution [202], \(p(x|F) = c_F\). According to the central limit theorem, the (weighted) summation of identically distributed random variables can be approximated by a Gaussian distribution. Since the wavelet coefficient in each band is obtained with a filter operation which can be regarded as a linear transform, the wavelet coefficients of the foreground can be assumed to follow a Gaussian distribution as follows.

\[
p(x_{d,l}|F) = N(x_{d,l}; \mu_F, \delta^2_F) \tag{6.6}
\]

where \(\mu_F\) and \(\delta^2_F\) are the mean and variance of the foreground distribution for the wavelet band \(d\) at level \(l\) (where the subscript \(d,l\) is removed for simplicity as above). For the high frequency wavelet bands including “LH”, “HL” and “HH”, the mean is zero due to the properties of the wavelet transform. In our experiments, the variances of different bands are estimated using sample video sequences and fixed through all the tests.
Fusion of the likelihood from different wavelet bands

One way of obtaining the likelihood functions for each type of wavelet band is using the Bayesian theorem in the same way as (6.4) by assuming they are independent from each other. Denote \( p(x_{d,1}, \ldots, x_{d,N}|F) \) and \( p(x_{d,1}, \ldots, x_{d,N}|B) \) by \( p(x_d|F) \) and \( p(x_d|B) \), respectively. The likelihood functions for one type of wavelet bands can be obtained as follows.

\[
p(x_d|F) = p(x_{d,1}|F)p(x_{d,2}|F) \ldots p(x_{d,N}|F)
\]
\[
p(x_d|B) = p(x_{d,1}|B)p(x_{d,2}|B) \ldots p(x_{d,N}|B)
\] (6.7)

However, one type of wavelet bands over different levels all capture details in the same direction and thus are related to each other. Therefore, the above decomposition may not be able to well represent the likelihood functions. In the following, we propose a weighted fusion way to obtain the likelihood functions for each type of wavelet bands.

It is known that each wavelet coefficient is obtained based on a number of pixels in a block and the block size increases as the level increases. Thus the result obtained for the wavelet coefficient from each band is a noisy result corresponding to a block. Therefore, from the perspective of noise reduction, the likelihood functions from one type of wavelet bands can be obtained by

\[
p(x_d|F) = \frac{1}{N} \sum_{l=1}^{N} f(p(x_{d,l}|F))
\]
\[
p(x_d|B) = \frac{1}{N} \sum_{l=1}^{N} f(p(x_{d,l}|B))
\] (6.8)

where \( f(\cdot) \) maps the result from the wavelet domain to the image domain. In this chapter, a linear mapping process is used, which is shown in the following.

It is clear that if all the pixels related to a coefficient belong to the same object, the result obtained based on this coefficient can well represent the result on these pixels. On the contrary, if pixels related to a coefficient belong to different objects, the result obtained based on this coefficient may not be correct for all these pixels. In order to characterize this relationship, the
correlation among pixels needs to be modelled first. It is known that the image signal can be modelled as a first-order autoregressive process \( x(i) = c + \alpha \cdot x(i-1) + \varepsilon \), where \( \alpha (0 < \alpha < 1) \) is the autoregressive coefficient, \( c \) is a constant, and \( \varepsilon \) is a white noise process with zero mean. Based on this model, in this chapter, the correlation among pixels is simply represented as

\[
\rho(\delta) = \alpha^\delta
\] (6.9)

where \( \delta \) represents the distance between two pixels. When the distance between two pixels increases, their correlation (\( \rho \)) reduces. Therefore, as the wavelet decomposition level increases and accordingly the number of pixels related to one coefficient increases, more pixels are becoming far away from the central pixel. Consequently, the correlation between these pixels and the central pixel gets smaller, and the result obtained based on the coefficient is becoming less representative for all its related pixels.

To account for this difference among different levels, a translation weight \( \omega_{t,l} \) is introduced. It is determined as the average correlation of all the pixels related to the coefficient.

\[
\omega_{t,l} = \frac{1}{N_p} \sum_{i=1}^{N_p} \rho(\delta_i)
\] (6.10)

where \( N_p \) is the total number of the pixels related to one coefficient in the wavelet bands at level \( l \) and \( \delta_i \) represents the distance between the pixel and the center of the block.

On the other hand, noise exists in images and thus in the wavelet bands. The results obtained at each level are all affected by the noise to some extent. Assume it is white Gaussian noise, which has zero mean and the same energy at different frequencies. Since the energies of the signal at different wavelet bands of different levels can be quite different, the effects of the noise on the results of different bands are different. To take this into account, a noise-induced weight factor is introduced as follows.

\[
\omega_{n,d,l} = \frac{\sigma_{d,l} - \sigma_n}{\sigma_{d,l}}
\] (6.11)

where \( \sigma_{d,l} \) and \( \sigma_n \) are the standard deviation of the wavelet band of type \( d \) at level \( l \) and noise, respectively.

The mapping function \( f(\cdot) \) in (6.8) is defined by combining the translation weight in (6.10) and the noise induced weight in (6.11) as \( f(p(x_{d,l} | \cdot)) = \omega_{d,l} p(x_{d,l} | \cdot) \), where \( \omega_{d,l} = \omega_{t,l} \cdot \omega_{n,d,l} \).
Accordingly the likelihood functions for each wavelet band in (6.8) can be obtained as

\[
p(x_d|F) = \frac{1}{N} \sum_{l=1}^{N} \omega_{d,l} p(x_{d,l}|F)
\]

\[
p(x_d|B) = \frac{1}{N} \sum_{l=1}^{N} \omega_{d,l} p(x_{d,l}|B)
\]

(6.12)

The above process fuses the results from one type of wavelet bands among different levels. Then according to (6.4), the result from different types of wavelet bands can be further fused. Accordingly, the foreground decisions can be made via (6.3).

In the implementation, since the “LL” wavelet bands focus on the intensity while the other bands focus on the texture, the “LL” wavelet bands are processed separately and the final foreground detection result is obtained by combining the fused result from the high frequency wavelet bands and the result from the low frequency bands. Also for the “LL” wavelet bands, the foreground distribution is regarded as a uniform distribution in the same way as in the conventional GMM model for simplicity.

### 6.5 Experiments

#### 6.5.1 Experimental setting

Since we focus on the camouflaged foreground detection problems, camouflaged videos are used for evaluation. The camouflaged videos in [203] and SBM-RGBD dataset [204] are both used in our experiments. The video in [203] was artificially generated by computers and the 5 videos in [204] are recorded as RGB-D dataset using the Microsoft Kinect. In the experiments, the depth channel is not considered. Moreover, we further recorded 10 videos captured in real scenes including both in-house and out-of-house cases. The dataset is named as “CAMO_UOW” and the details of the dataset including the resolution, number of frames and the format (grayscale or RGB) are shown in Table 6.1. In our recorded sequences, the foreground person wears clothes in a similar color as that of the background. Groundtruth are manually labelled for all frames in the collected “CAMO_UOW”. Example frames of the sequences in CAMD_UOW is available at https://www.uow.edu.au/~wanqing/#Datasets.
Table 6.1: Details on the collected camouflaged video dataset (CAMO_UOW). The groundtruth is available for all frames.

<table>
<thead>
<tr>
<th>Video number</th>
<th>Resolution</th>
<th>Frames</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video 1</td>
<td>1600 × 1200</td>
<td>371</td>
<td>Grayscale</td>
</tr>
<tr>
<td>Video 2</td>
<td>1600 × 1200</td>
<td>176</td>
<td>Grayscale</td>
</tr>
<tr>
<td>Video 3</td>
<td>1600 × 1200</td>
<td>371</td>
<td>Grayscale</td>
</tr>
<tr>
<td>Video 4</td>
<td>1600 × 1200</td>
<td>371</td>
<td>Grayscale</td>
</tr>
<tr>
<td>Video 5</td>
<td>1600 × 1200</td>
<td>371</td>
<td>Grayscale</td>
</tr>
<tr>
<td>Video 6</td>
<td>1600 × 1200</td>
<td>373</td>
<td>Grayscale</td>
</tr>
<tr>
<td>Video 7</td>
<td>1920 × 1080</td>
<td>272</td>
<td>RGB</td>
</tr>
<tr>
<td>Video 8</td>
<td>1920 × 1080</td>
<td>466</td>
<td>RGB</td>
</tr>
<tr>
<td>Video 9</td>
<td>1920 × 1080</td>
<td>288</td>
<td>RGB</td>
</tr>
<tr>
<td>Video 10</td>
<td>1920 × 1080</td>
<td>458</td>
<td>RGB</td>
</tr>
</tbody>
</table>

“CAMO_UOW” are shown in Fig. 6.6 along with the groundtruth foreground masks.

In our implementation, the images are decomposed into six levels of wavelet bands. Accordingly, the sequences are cropped into the nearest size that can be divided by $2^6$ due to the requirement of wavelet transform. Moreover, all the sequences are converted into the grayscale format which makes the foreground detection problem more difficult. Since the wavelet coefficients in each band are modelled using the GMM model, the MOG2 implementation in openCV to be specific, several parameters are involved, including the initial variance, minimum and maximum variance of the Gaussian distributions. In our implementation, the parameters are set based on those used in the MOG2 implementation, in a proportional way according to their variance. That is to say, $\sigma^2_{\text{ini,prop}} = \sigma^2_{\text{ini,MOG2}} \cdot \frac{\sigma^2_w}{\sigma^2_o}$, where $\sigma^2_{\text{ini,prop}}$, $\sigma^2_{\text{ini,MOG2}}$, $\sigma^2_w$ and $\sigma^2_o$ represent the initial variance used for the proposed method of a wavelet band, the initial variance used in MOG2 for the natural image, the variance of the wavelet band and the variance of the natural image, respectively. The rest of the parameters are set in the same way. For the high frequency wavelet bands including “LH”, “HL”, “HH”, when the magnitude of the coefficient deviates from zero, it usually represents edges and the Gaussian distribution used to formulate such coefficients may express a large change over time. Therefore, the initial variance, minimum and maximum variance of the Gaussian are set to be larger when the mean deviates from 0. Taking the initial variance for example, it is set as $\sigma^2_{\text{ini,prop}} + \alpha \cdot m^2$, where $m$ represents the mean of the Gaussian distribution and $\alpha$ is a small weight. In the end, the detection results are
processed with some morphological operations to remove the isolated pixels. To be specific, to remove the small holes inside a foreground object, the result is first processed with a closing operation and an eroding operation. Then combining with the original result, it is further processed with a closing and opening operation to remove noise. The final result is further eroded considering that the edges of the foreground objects may get dilated due to the wavelet transform.

6.5.2 Performance evaluation

The performance of the proposed “FWFC” is shown both qualitatively and quantitatively. Several popular and representative methods are compared, including “MOG2” [145], “FuzzyChoquetIntegral” [205], “LBAAdaptiveSOM” [206], “MultilayerBGS” [208], “SuBSENSE” [207],

Figure 6.6: Example frames and groundtruth in our collected video dataset “CAMO_UOW”.

(a) Video 1  (b) Video 2
(c) Video 3  (d) Video 4
(e) Video 5  (f) Video 6
(g) Video 7  (h) Video 8
(i) Video 9  (j) Video 10
Table 6.2: Performance comparison on different tested videos in terms of F-measure.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Video 1</td>
<td>0.79</td>
<td>0.88</td>
<td>0.8</td>
<td>0.9</td>
<td>0.89</td>
<td>0.89</td>
<td>0.92</td>
<td>0.8</td>
<td>0.94</td>
</tr>
<tr>
<td>Video 2</td>
<td>0.82</td>
<td>0.79</td>
<td>0.8</td>
<td>0.82</td>
<td>0.88</td>
<td>0.8</td>
<td>0.83</td>
<td>0.58</td>
<td>0.96</td>
</tr>
<tr>
<td>Video 3</td>
<td>0.88</td>
<td>0.86</td>
<td>0.85</td>
<td>0.91</td>
<td>0.9</td>
<td>0.8</td>
<td>0.82</td>
<td>0.9</td>
<td>0.94</td>
</tr>
<tr>
<td>Video 4</td>
<td>0.89</td>
<td>0.9</td>
<td>0.76</td>
<td>0.93</td>
<td>0.78</td>
<td>0.88</td>
<td>0.8</td>
<td>0.72</td>
<td>0.94</td>
</tr>
<tr>
<td>Video 5</td>
<td>0.84</td>
<td>0.86</td>
<td>0.82</td>
<td>0.83</td>
<td>0.82</td>
<td>0.8</td>
<td>0.82</td>
<td>0.75</td>
<td>0.91</td>
</tr>
<tr>
<td>Video 6</td>
<td>0.93</td>
<td>0.87</td>
<td>0.77</td>
<td>0.95</td>
<td>0.92</td>
<td>0.95</td>
<td>0.97</td>
<td>0.72</td>
<td>0.94</td>
</tr>
<tr>
<td>Video 7</td>
<td>0.76</td>
<td>0.83</td>
<td>0.88</td>
<td>0.91</td>
<td>0.87</td>
<td>0.79</td>
<td>0.91</td>
<td>0.83</td>
<td>0.96</td>
</tr>
<tr>
<td>Video 8</td>
<td>0.83</td>
<td>0.87</td>
<td>0.85</td>
<td>0.87</td>
<td>0.93</td>
<td>0.86</td>
<td>0.86</td>
<td>0.68</td>
<td>0.96</td>
</tr>
<tr>
<td>Video 9</td>
<td>0.89</td>
<td>0.9</td>
<td>0.87</td>
<td>0.84</td>
<td>0.92</td>
<td>0.87</td>
<td>0.86</td>
<td>0.78</td>
<td>0.88</td>
</tr>
<tr>
<td>Video 10</td>
<td>0.89</td>
<td>0.86</td>
<td>0.89</td>
<td>0.91</td>
<td>0.92</td>
<td>0.9</td>
<td>0.94</td>
<td>0.85</td>
<td>0.96</td>
</tr>
<tr>
<td>Camouflage[203]</td>
<td>0.72</td>
<td>0.72</td>
<td>0.71</td>
<td>0.82</td>
<td>0.82</td>
<td>0.78</td>
<td>0.84</td>
<td>0.76</td>
<td>0.75</td>
</tr>
<tr>
<td>colorCam1[204]</td>
<td>0.15</td>
<td>0.1</td>
<td>0.82</td>
<td>0.42</td>
<td>0.08</td>
<td>0.09</td>
<td>0.26</td>
<td>0.59</td>
<td>0.61</td>
</tr>
<tr>
<td>colorCam2[204]</td>
<td>0.76</td>
<td>0.22</td>
<td>0.6</td>
<td>0.52</td>
<td>0.74</td>
<td>0.28</td>
<td>0.48</td>
<td>0.94</td>
<td>0.63</td>
</tr>
<tr>
<td>Cespatx_ds[204]</td>
<td>0.71</td>
<td>0.69</td>
<td>0.86</td>
<td>0.81</td>
<td>0.9</td>
<td>0.6</td>
<td>0.74</td>
<td>0.86</td>
<td>0.92</td>
</tr>
<tr>
<td>Hallway[204]</td>
<td>0.59</td>
<td>0.62</td>
<td>0.43</td>
<td>0.62</td>
<td>0.69</td>
<td>0.58</td>
<td>0.67</td>
<td>0.63</td>
<td>0.67</td>
</tr>
<tr>
<td>Office2[204]</td>
<td>0.33</td>
<td>0.36</td>
<td>0.79</td>
<td>0.58</td>
<td>0.48</td>
<td>0.47</td>
<td>0.89</td>
<td>0.76</td>
<td>0.9</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>0.74</td>
<td>0.71</td>
<td>0.78</td>
<td>0.79</td>
<td>0.78</td>
<td>0.71</td>
<td>0.8</td>
<td>0.76</td>
<td>0.87</td>
</tr>
</tbody>
</table>
Figure 6.7: The average performance of all the test videos in terms of Recall, Precision and F-measure for different methods.

“PBAS” [156], “DECOLOR” [161], “COROLA” [162]. The “LSD” method [160] is similar to “DECOLOR” [161] and “COROLA” [162] methods in terms of low-rank representation and performance on the camouflaged videos and so is not further included in the comparison. The results of these methods are obtained based on the bgslibrary [209], [210] and the LRSLibrary [211], [212]. The default settings in these libraries are used which is explained in the survey paper [135] and [140], respectively. No post-processing such as morphological operations is employed for the comparison methods since the result in our experiments shows no noticeable improvement. Note that “DECOLOR” [161] cannot directly work on the sequences as the memory required is larger than 32GB and takes a very long time. Therefore, its results are first obtained on downsampled videos (the largest dimension being 320 with the same aspect ratio) and then upsampled to its original size. There is a block-by-block implementation available in the LRSLibrary enabling the direct processing of sequences with large resolutions. However, its performance is worse than processing with downsampling and upsampling in our experiments.

The results obtained with different methods for some example frames are shown in Fig. 6.8. The images in each row from left to right represent the original frame, the groundtruth foreground, and the result of different methods.
Figure 6.8: Example of foreground detection results obtained using different methods on all the videos. From left to right: the original frame, the groundtruth foreground mask, and the results obtained by different methods including “MOG2” [145], “FuzzyChoquetIntegral” [205], “LBAAdaptiveSOM” [206], “PBAS” [156], “SuBSENSE” [207], “MultiLayerBGS” [208], “DECOLOR” [161], “COROLA” [162], the proposed FWFC. From top to bottom: Video 1-10 in our collected “CAMO_UOW” dataset, the camouflaged video in [203], colorCam1[204], colorCam2[204], Cespatx_ds[204], Hallway[204], Office2[204].
CHAPTER 6. EXPLORATION OF APPLICATIONS IN COMPLEX SCENES

ground mask, and the results obtained by different methods including “MOG2” [202], “Fuzzy-ChoquetIntegral” [205], “LBAAdaptiveSOM” [206], “PBAS” [156], “SuBSENSE” [207], “MultiLayerBGS” [208], “DECOLOR” [161], “COROLA” [162], the proposed FWFC. Each row of images from top to bottom represent Video 1-10 in our collected “CAMO_UOW” dataset, the camouflaged video in [203], colorCam1[204], colorCam2[204], Cespatx_ds[204], Hallway[204], Office2[204]. From the results, it can be clearly seen that FWFC achieves the best result around the camouflaged foreground area for most of the sequences. Taking the “Video 1” shown in the first row of Fig. 6.8 for example, the existing methods cannot detect the body of the foreground person who wears clothes in the similar color as the background wall. On the contrary, FWFC works very well as most of the body has been detected. However, for colorCam1[204] and colorCam2[204], parts of the camouflaged foreground objects are not able to be detected. This is mainly because the foreground object and background in these two videos are both texture-less objects.

The performance is also quantitatively measured using Recall, Precision and F-measure. For “Video 1” - “Video 10” in our collected dataset “CAMO_UOW” and the camouflaged video in [203], groundtruth is available for all frames and thus the Recall, Precision and F-measure of the static quality metrics in [213] is used, and computed by the BMC Wizard software. For the sequences in [204], groundtruth is only available for a few frames and the quality metrics are computed with the scripts provided by [204]. Table 6.2 shows the result of the proposed method in terms of F-measure compared with the existing methods. “Average” represents the average result of all the videos. The notations of the existing methods are simplified due to the limited space, and their full notations can be found in the beginning of this Subsection. It can be seen that the performance of FWFC is much better than the existing methods, with an average F-measure of 0.87, compared to values between 0.71 and 0.8 for the existing methods.

It is worth noting that the performance on the camouflaged video in [203] is slightly worse than some of the existing methods. This is mainly because FWFC is implemented based on MOG2 [145] which performs slightly worse in the case of swaying background such as the leaves of the tree. As shown in Fig. 6.8, part of the trees are still detected as the foreground which is the
Table 6.3: Complexity comparison in terms of running time (seconds) for one frame of resolution 640*384.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOG2[145]</td>
<td>0.007</td>
</tr>
<tr>
<td>FCI[205]</td>
<td>0.251</td>
</tr>
<tr>
<td>LBA-SOM[206]</td>
<td>0.076</td>
</tr>
<tr>
<td>PBAS[156]</td>
<td>0.449</td>
</tr>
<tr>
<td>SuBSENSE[207]</td>
<td>1.176</td>
</tr>
<tr>
<td>ML-BGS[208]</td>
<td>0.293</td>
</tr>
<tr>
<td>DECOLOR[161]</td>
<td>4.999</td>
</tr>
<tr>
<td>COROLA[162]</td>
<td>0.619</td>
</tr>
<tr>
<td><strong>FWFC</strong></td>
<td><strong>0.275</strong></td>
</tr>
</tbody>
</table>

same as the MOG2 result. However, it can be seen that FWFC works very well in capturing the camouflaged foreground such as the car. Also, FWFC still achieves better performance than MOG2, which FWFC is developed upon.

The average result of the tested videos in terms of Recall, Precision and F-measure is shown in Fig. 6.7. It can be clearly seen that the Recall of FWFC is much higher than others, which indicates that FWFC can detect most of the foreground objects. In terms of the Precision, the performance of FWFC is comparable to other methods. Therefore, combined with a higher Recall, FWFC achieves the best result. Experiments on regular (not camouflaged) videos were also conducted using the sequences from the CDW-2014 (2014 IEEE Change Detection Workshop) dataset [214], [215]. The results obtained by the proposed method are better than or comparable to those obtained by the conventional MOG2.

The computational complexity (in terms of CPU time) of FWFC in comparison with the existing methods are shown in Table 6.3. The time is for processing a frame of resolution 640*384 and is obtained on a personal computer (i7-4790 CPU with 32GB memory). Note that “DECOLOR” [161] and “COROLA” [162] are implemented in Matlab while others are implemented in C++. FWFC takes more time than MOG2 [145] but less time than most of the existing methods as listed in Table 6.3. Currently, FWFC is programmed in a sequential manner without optimization. Among the time (0.275 seconds) used to process a frame, the wavelet decomposition takes 0.12 seconds, the background modelling (MOG2) on the wavelet
bands takes 0.04 seconds and the fusion from the results obtained from the wavelet bands takes around 0.11 seconds. Note that FWFC is highly parallelable considering the result from each wavelet band can be obtained in parallel, which can further reduce the time.

6.6 Summary

This chapter investigates the camouflage moving foreground detection problem as an exploration of the applications in complex scenes since moving foreground detection can be considered as a very basic problem in computer vision tasks. A fusion framework is presented, which first transfers the foreground detection problem into the wavelet domain and shows that the small differences in the image domain can be detected in certain wavelet bands. Then foreground and background models are formulated for all the wavelet bands. The results from different bands are fused by considering the properties of different wavelet bands. Experimental results have shown that the proposed method performs significantly better than the existing methods in terms of camouflaged foreground detection.
Chapter 7

Conclusion

7.1 Summary of Contributions

This thesis addresses some challenges in deep learning including CNNs and RNNs, and also considers their applications to activity recognition.

A literature survey is presented in Chapter 2 which outlines some limitations of existing state-of-the-art methods and highlights the challenges of deep learning with CNNs and RNNs addressed in this research. Specifically, different CNN architectures are described where the existing pooling functions used in CNNs are explained in detail showing their limitation of not being adaptive to data. Moreover, different RNN variants are reviewed including simple RNN and LSTM, and their problems in terms of processing long sequences and constructing deep networks are discussed.

In Chapter 3, a fully trainable network (FTN) is proposed where RNNs are used to perform the pooling function instead of the handcrafted pooling functions used in traditional CNNs. Since RNNs are universal approximators (Turing-complete), RNNs can be trained to function as any pooling function. In this way, the pooling function can be trained together with the other convolutional layers to make the whole network trainable without any handcrafted function. Different FTN architectures are explored and the LSTM (one type of RNN) is extended with ReLU to align with the convolutional layers. Experimental results show the efficiency of the proposed RNN-based pooling by demonstrating that only one LSTM unit based pooling can approximate the existing pooling functions with a very high accuracy, making it appropriate
CHAPTER 7. CONCLUSION

to be used in CNNs. Then the performance of FTN is verified by comparing with CNNs with the existing pooling methods in image classification tasks. The effect of RNN based pooling in different sizes of networks has also been studied along with analysis of the learned pooling functions.

In Chapter 4, an independently recurrent neural network (IndRNN) is proposed, where neurons in one layer are independent of each other. The structure of IndRNN is explained and the gradient backpropagation through time process is illustrated. It effectively solves the long-standing gradient vanishing and exploding problems in traditional RNNs by regulating the recurrent weights, making IndRNN appropriate for processing long sequences. Moreover, different multiple-layer IndRNN architectures are explored including directly stacking IndRNN and stacking IndRNN through residual connections, making the network very deep. With multiple layers stacked, the cross-channel information of the neurons in one layer is explored in the following layers. Moreover, the relationship between the IndRNN and the RNN is demonstrated where under linear activation, a traditional RNN is a special case of a two-layer IndRNN and they share a similar number of parameters and running complexities. Experiments on multiple tasks including the adding problem, sequential MNIST classification and language modelling (both char-level and word-level) have been conducted, where IndRNN achieves better performance than the tradition RNN methods. Moreover, experiments on processing 5000 time steps for the adding problem and the use of a 21-layer IndRNN show that IndRNN can be deeper and longer.

In Chapter 5, the IndRNN model is applied to skeleton based activity recognition, where a IndRNN based deep attention model is developed. The proposed model employs a deep network to obtain the attention instead of just one or two layers in the conventional attention models, and a deep IndRNN network for classification. Moreover, a new loss function is proposed based on the triplet loss to regulate the attention among different action categories. The new loss function explicitly enforces the intra-class attention distance to be no larger than the inter-class attention distance. Experiments have demonstrated that the proposed model can attain more robust attention weights and better performance than the existing methods.
In Chapter 6, the camouflaged foreground detection problem is investigated to explore the performance of different computer vision applications in complex scenes. A fusion framework is presented to address the camouflaged foreground detection problem in the wavelet domain. The motivation and overview of the proposed method is described where the foreground detection problem is first transformed into the wavelet domain and shows that the small differences in the image domain can be detected in certain wavelet bands. Therefore, the detection is first performed in the wavelet domain by reformulating the foreground and background models for all the wavelet bands. The results from different bands are then fused by considering the properties of different wavelet bands. The experimental results show the proposed method performs significantly better than existing methods in camouflaged foreground detection.

7.2 Future Work

Research in this thesis can be extended in many ways. The concept of independently recurrent neural network in this thesis has pioneered a new direction for the exploration of recurrent neural networks. The proposed fully trainable network also extends the frontier of traditional CNN architectures. Specifically, some extensions are listed as follows:

- The effect of pooling in CNNs has been questioned for a long time despite the fact that lots of research has been conducted on it and proved its usefulness in CNNs. Its disadvantage of losing the geometric information in a local pooling region makes it difficult for CNNs to learn different setups of objects as this requires large amounts of data covering different perspectives of objects for training. The recent proposed Capsule Net [216] is also dedicated to reducing the effect of pooling and improving the efficiency of representation. The proposed FTN with the RNN based pooling can keep the local information by aggregating information at all the pooling locations through RNN. Therefore, it should alleviate the loss of information which occurs in traditional pooling. However, the current proposed RNN based pooling employs LSTM which usually consumes lots of memory, leading to a loss in training efficiency. Moreover, the current RNN based pooling uses
one unit for each channel, limiting the capacity of the learned function. On the contrary, CNNs use multiple neurons and multiple layers to explore the structure of the data. Therefore, in the future, RNN based pooling can be improved by using efficient RNN structures and better RNN based pooling architectures.

- The independently recurrent neural network (IndRNN) proposed in this thesis opens a door to new research such as visualization and understanding, network pruning and deeper RNN architectures, which has been very difficult for traditional RNNs. In addition to the experiments and applications conducted in this thesis, there are still many new applications available to be explored with IndRNN such as speech recognition and neural machine translation. Moreover, although in the thesis IndRNN has been extended to the convolutional form, the convolutional IndRNN architectures have not yet been fully investigated and validated. In the future, different IndRNN architectures and different applications can be explored.

- The exploration of camouflaged foreground detection shows that while the problem becomes harder in complex scenes, it is still solvable. Therefore, different applications in complex scenes could be further explored. For example, human activity recognition with different types of backgrounds can be investigated to illustrate the effect caused by different backgrounds.
Acknowledgments

First and foremost, I would like to express my most sincere gratitude to my supervisor and co-supervisor, A/Prof. Wanqing Li and Prof. Chris Cook, for their advice and support. In the last three years, they have taught me a lot, both knowledge and research methods. I am also very grateful that they take their after-hours, early morning, late evening or weekends, to help me revise reports and paper drafts. Most of all, I want to thank them for supporting me to do the research I like. Their constant encouragement and thoughtful considerations make my PhD journey very delightful.

Secondly, I want to thank my colleagues and my friends, Prof. Philip Ogunbona from my lab, Prof. Ce Zhu from UESTC, Dr. Dinei Florencio from Microsoft Research, Lijuan Zhou, Yuyao Zhang, Pichao Wang, Jing Zhang, Zewei Ding, Huogen Wang, Chuankun Li, Jun Hu, et al., for their constructive discussions, kind help and all the joyful memories.

I would like to thank my family for their supports and backing my choices, especially my wife Yanbo Gao for keeping me company.

With great appreciation I shall acknowledge the University of Wollongong for providing Scholarships which makes it possible for me to come in the first place. It is a very beautiful school and I enjoyed my time here.
Bibliography


