SdcNet: A Computation-Efficient CNN for Object Recognition

Yunlong Ma

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Electrical and Computer Engineering) at

Concordia University

Montréal, Québec, Canada

March 2019

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Yunlong Ma

Entitled: SdcNet: A Computation-Efficient CNN for Object Recognition

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science

Complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. Weiping Zhu

_____ Examiner, External
Dr. Abdessamad Ben Hamza (CIISE) To the Program

_____ Examiner
Dr. Weiping Zhu

_____ Supervisor
Dr. Chunyan Wang

_____ Co-Supervisor

Approved by: _____
Dr. W. E. Lynch, Chair
Department of Electrical and Computer Engineering

_____20_____

_____
Dr. Amir Asif, Dean
Gina Cody School of Engineering and
Computer Science

Abstract

SdcNet: A Computation-Efficient CNN for Object Recognition

Yunlong Ma

In many computer-vision systems, object recognition is one of the most commonly-used operations. The challenging task in this operation is to extract sufficient critical features related to the targets from diverse backgrounds. Convolutional neural networks (CNNs) can be used to meet this challenge, which, however, often requires a large amount of computation resources.

In this thesis, a computation-efficient CNN architecture for object recognition is proposed. It aims at using the lowest computation volume to achieve a good processing quality. This is achieved by applying image filtering knowledge in the design of the CNN architecture. This work is composed of two parts, the design of a CNN module for feature extraction, and an end-to-end CNN architecture. In the module, in order to extract the maximum amount of high-density feature information from a given set of 2-D maps, successive depthwise convolutions are applied to the same group of data to produce feature elements of various filtering orders. Moreover, a particular pre-and-post-convolution data control method is used to optimize the successive convolutions. The pre-convolution data control is to organize the data to be convolved according to their nature. The post-convolution data control is to combine the critical feature elements of various filtering orders to enhance the quality of the convolved results. The CNN architecture is mainly composed of the cascaded modules. The hyper-parameters in the architecture can be adjusted easily so that each module is tuned to suit the signals in order to optimize the processing quality. The simulation results demonstrated that the architecture gives a better processing quality using a significantly lower computation volume, compared with existing CNNs of the similar kind. The results also confirm the computation efficiency of the proposed module, which enables more object recognition applications on embedded devices.

Acknowledgments

I would like to express my deep gratitude to my supervisor Dr. Chunyan Wang for her immense support and brilliant guidance during my study in Concordia University. She consistently allowed this thesis to be my work, but steered me in the right direction whenever she thought I needed it. I feel extremely privileged to be able to work under her supervision.

I would also like to thank my friends: Bao Zhu, Ameen Elsiddiq and Mohamed Hamid for the discussion of research and for all the fun we had in the last two years.

A very special gratitude goes out to Compute Canada and Calcul Quebec for providing the powerful servers for this work.

Last but not the least, I would like to thank my girl friend Suya Tang and my family in China, for all the love and support from them.

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

BN	Batch Normalization
CIFAR	Canadian Institute For Advanced Research
C-10	CIFAR-10 Dataset
C-100	CIFAR-100 Dataset
CNN	Convolutional Neural Network
DWConv	Depthwise Convolution
FC	Fully Connected Layer
G_Conv	Group Convolution
LR	Learning Rate
MSE	Mean Squared Error
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
Sdc	Successive Depthwise Convolutions
SdcBlock	A Successive Depthwise Convolutions Block
SdcNet	A CNN Built by SdcBlocks
SCR	Specific Color Reordering

List of Symbol

Ψ	The weights in a CNN.
ϕ	The gradients.
ζ	The momentum weight in an optimizer.
a	Input to a layer.
E	The expansion number in a SdcBlock.
Ep	The epoch number of training.
ER	The classification error rates.
g	Group number.
t_i	The ground truth label.
H	The height of a matrix.
I	Input data to a SdcBlock.
K	The number of convolution kernels in a layer.
m	The number of images in a mini-batch.
$Loss$	The loss value.
M	The height of a convolution kernel.
N_I	The channel number of data I.
N	The width of a convolution kernel.
P_e	The number of incorrectly predicted images.
T_N	The number of the test images.
W	The width of a matrix.
$a^l \in \mathbb{R}^{H^l \times W^l \times D^l}$	The input tensors for the l layer.

(i^l, j^l, d^l)	The element of the row i^l, the column j^l, the channel d^l in a tensor in the l layer.
X	Input data to the Sdc in a SdcBlock.
X'	Results of the first operations in a SdcBlock.
X''	Results of the second operations in a SdcBlock.
X_c	Concatenated results of X' and X'' in a SdcBlock.
y_i	The results of a CNN.

Chapter 1

Introduction

1.1 Background and Challenges

In recent years, computer-vision-based applications, such as autopilot , surveillance security systems and augmented reality systems [1, 2, 3, 4], have increased tremendously, which demands rapid development of techniques in this area. Object recognition is one of the most commonly-used operations in computer-vision-based systems [5]. It is to identify objects in an image or a video and the goal is to teach a computer vision system to gain a level of understanding of what an image contains [6]. In a car equipped with an autopilot system, object recognition is being used to recognize traffic signs, pedestrians, other vehicles and other traffic elements.

In general, an object recognition approach is performed by, (i) extracting various features related to the objects from diverse backgrounds, and (ii) classifying the objects according to the features. However, since there can be a huge amount of feature variations for the same object class, while the same feature elements appear in the different object classes, like the situations in Fig. 1.1, it is a critical challenge to achieve high performance of an object recognition system.

1.2 Motivation and Objective

Since the development in the domain of object recognition is highly demanded, a lot of research efforts have been made to achieve a good performance. Usually, the recognition systems can be designed by two kinds of approaches, knowledge-based filtering and machine learning. The

Figure 1.1: The dog on the left is similar to the cat on the middle, while the difference between the cat on the middle and that on the right is large.

approach of knowledge-based filtering [7, 8, 9, 10, 5, 11] is normally composed of three parts: data analysis, feature extraction and object recognition. The natures of the input images and the targeted objects are firstly analyzed, and then particular kinds of feature information related to the target objects are extracted by specifically designed filters. Finally, the recognition results are generated based on these extracted feature information. This approach is usually computation-efficient, but it has limitations in handling a huge number of variations in object features.

Machine learning approaches, in particular convolutional neural networks(CNNs), can provide a good solution to deal with the huge variations of features [12, 13, 14, 15]. This kind of networks uses a large number of samples to progressively determine the system parameters in order to generate various object features. The networks such as VGG [14] and ResNet [15] have been reported to deal with complex object recognition problems. However, CNNs normally require a large number of parameters, which, in consequence, leads to a huge computation volume in order to achieve a good performance [16]. The computation resource requirement is too strict for many embedded devices, which limits the implementation of object recognition by CNNs.

In fact, implementing object recognition functions in mobile devices is needed to improve the quality of life of ordinary people. It can be used, $e.g.$, to identify diseases by a photo, to simplify shopping by taking a picture of what you need, and also to start a task when the camera recognize a particular object [17, 4]. Thus, it is desirable to apply the image filtering knowledge in the design of CNN, in order to improve the computation efficiency by reducing its computation volume while without sacrificing the processing quality.

The objective of the work presented in this thesis is to develop a computation-efficient CNN architecture for object recognition tasks. Based on the image processing knowledge and the CNN

procedure, it seeks to use the lowest computation volume to achieve a good recognition quality with a view to enabling the implementation in more different applications.

1.3 Scope and Organization

The work presented in this thesis is focused on designing a low-computation object recognition system. It is done by using image processing knowledge to improve the computation efficiency of a CNN architecture and it is composed of two parts, a design of CNN module for feature extraction, since a general CNN architecture is mainly formed by a stack of feature-extracting modules, and an end-to-end CNN architecture for object recognition.

The work will start with an investigation of different convolution modes in CNNs and the analysis of the data in the different computation stages. Based on the results of the investigation, the module is designed following the image processing principles. In fact, the CNN module is a basic building block in the image filtering area. In order to extract extensive high-density features from a given set of feature maps, it is designed to use appropriate convolution modes and data flow control methods. Besides, the module is made to facilitate its use in different stages to extract features for various CNN applications. The other part is to develop a CNN architecture by an optimal use of the designed modules to achieve the high computation efficiency in object recognition.

The thesis is organized as follows. In Chapter 2, a brief review of the fundamentals of CNN is given, and the existing methods relevant to this work are also described. In Chapter 3, a design of the computation-efficient CNN module for feature extraction is proposed. A detailed description of the design ideas based on image processing principles and its implementation is presented. In Chapter 4, an end-to-end CNN architecture, based on the proposed CNN modules, for object recognition tasks is presented. Three detailed network configurations are given with different emphasis in performance. In Chapter 5, the performance of the three networks is evaluated, which is also used to assess the computation efficiency of the proposed module. The test results are compared with CNNs reported recently for the same tasks. Finally, Chapter 6 concludes the work and highlights its contributions.

Chapter 2

Background and Relevant Work

2.1 Introduction

Object recognition is highly demanded in the computer-vision-based applications. However, due to the limitation of computation requirements, the processing quality of object recognition can be degraded in the embedded devices, especially in the mobile devices. Therefore, the development of a computation-efficient architecture for object recognition tasks is necessary for most mobile computer-vision-based applications.

A variety of methods for object recognition have been developed, including the knowledge-based filtering and machine learning approaches. Among these methods, the knowledge-based filtering approach is usually computation-efficient, but it has limitations in handling a huge number of variations in object features. However, the machine learning approaches, in particular convolutional neural networks(CNNs) , can achieve a good processing quality requiring a large computational volume. Thus, it is desirable to apply the image filtering knowledge in the design of CNN, in order to improve the computation efficiency by reducing its computation volume while without sacrificing the processing quality.

In this chapter, an introduction of some fundamental CNNs for object recognition is given in Sub-chapter 2.2. Some recent CNNs for object recognition are reviewed in Sub-chapter 2.3. A brief summary is presented in Sub-chapter 2.4.

2.2 Fundamentals of CNN

A typical architecture of CNNs is illustrated in Fig. 2.1. From image processing point of view, it can be divided into three signal processing parts: early processing, feature extraction and decision processing. Each of these parts is composed of a stack of different computation layers, $i.e.$ the basic components in CNNs. However, once a CNN architecture is built using the basic components, in order to make it functional, all the weights need to be updated by the training procedure using training samples.

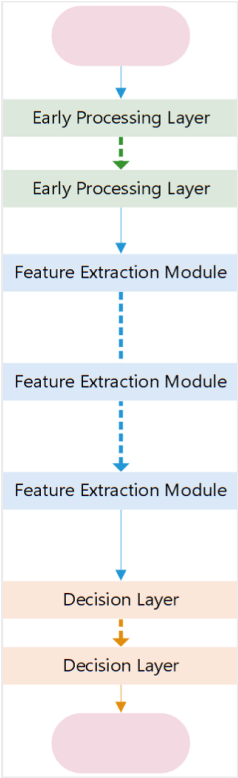


Figure 2.1: A typical CNN architecture

2.2.1 CNN Structures

A typical CNN architecture example is given in Fig. 2.2, and it is composed of a stack of different functional layers. Normally, these layers can be divided into convolution layers, pooling layers, fully-connected layers, non-linear layers and loss layers, in terms of functions.

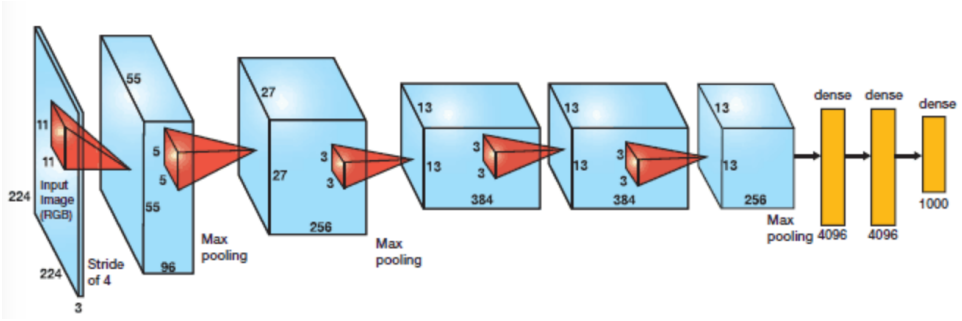


Figure 2.2: An example of a typical CNN architecture (AlexNet)

Convolution layer

In computer vision and object recognition, a convolution layer is a basic operation in CNNs to extract the features. The below is an example of a convolution layer.

Suppose the input data for the convolution layer l is $a^l \in \mathbb{R}^{H^l \times W^l \times D^l}$, the weights for this layer is $\Psi^l \in \mathbb{R}^{M \times N \times K^l}$. The three-dimension convolution layer expand the two-dimension convolution layer to all input channels, i.e. D^l.

After each training epoch, $\Psi^l_{m,n,d^l,k}$ will be updated according the update strategy and loss functions, $i.e.$, learnable weights. It is clear that the weights are same for different positions in the input data. This is called the weight sharing for the convolution layers. Furthermore, there are some other important hyper parameters such as stride, filter size, dilation and different styles of convolutions. Using proper convolution layers is important to extract feature information and contribute a good result.

- Standard convolution. In this mode, each of the K convolution kernels is applied to all the D^l input channels to generate one output channel.

- Group convolution (G-Conv). The D^l input channels is divided into g groups, and K convolution kernels are also divided into g sets. Each group of input data is convolved with a set of K/g kernels. The standard convolution can be seen as the specific case, with $g = 1$, of group convolution.

- Depthwise convolution (DW-Conv). It is, in fact, the conventional 2-D spatial convolution. In the context of CNN, it can be seen as another special case of group convolution, in which

$g = D^l$, $i.e.$, one channel per group, and each of the K kernels is applied only to one input channel, requiring $g = D^l = K$.

Pooling Layers

Pooling is a kind of down-sampling methods which can be done by specific pooling layers or stride convolution layers. It is to progressively reduce the spatial size of the feature maps to reduce the amount of weights and computation.

- Feature Down-Dimension. Because of down-sampling, an element of the pooling result represents for a sub-region of the input data. This means pooling is equivalent as spatially dimension reduction, therefore the network have an ability to extract feature information in a larger region. Meanwhile, the pooling layer can decrease the input data size, which can also decrease computation cost and the number of parameters.
- Scale Invariance. Pooling layer make the network focus more on feature itself other than the locations of features. This means that even a feature locates in a different location in the test case, the network can still extract enough feature information for the final classification.
- Lessening the Chance of Overfitting. According to our and the other researchers' experiment records, applying pooling can somehow lessen the chance of overfitting. It can make the network easy for training.

Fully connected layers

A fully connected layer [18] is usually used in the decision processing part to generate the final output by involving all the neurons in the previous layer, as shown in Fig 2.3. Neurons in a fully connected layer have full connections to all the inputs in the previous layer, as in regular Neural Networks. A fully connected layer can involve a huge computation volume but it can calculate a result based on all the neurons.

Normalization Layers

Batch Normalization (Batch Norm or BN) has been established as a very effective component in deep learning, largely helping push the frontier in computer vision. BN normalizes the features

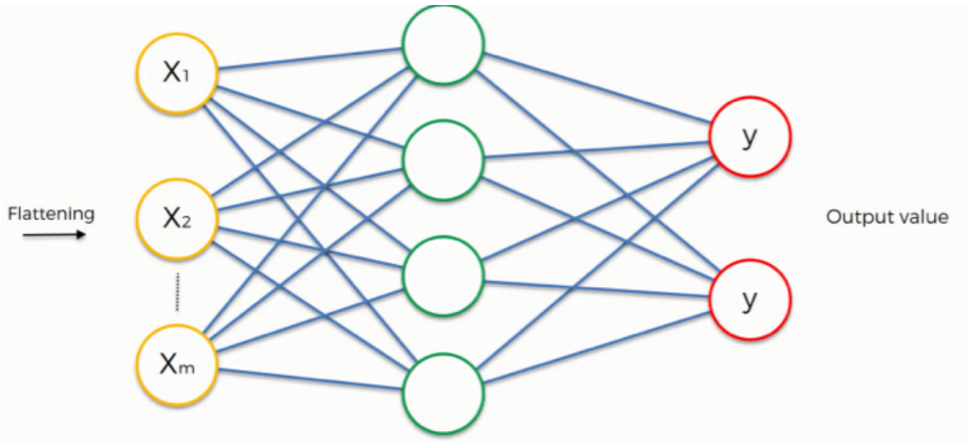


Figure 2.3: An example of fully connected layer

by the mean and variance computed within a batch [19]. This has been shown by many practices to ease optimization and enable deep networks to converge. The uncertainty of the batch statistics also acts as a regularizer that can benefit generalization.

Non-linear layers

A non-linear layer is used to introduce non-linearity to a network. It allows to model a class label that varies non-linearly with independent variables. Non-linear means the output cannot be replicated from a linear combination of inputs, which allows the model to learn complex mappings from the available data, and thus the network becomes a universal approximate. Another important aspect of the non-linear layer is that it should be differentiable. This is required during the backpropagation to compute gradients, and thus the weights can be tuned accordingly. The non-linear functions should be continuous and be able to transform the input into a desirable range.

(1) *Rectified Linear Unit (ReLU)*

Mathematically, ReLU [20] is given by this simple expression

$$\sigma(a) = max(0, a) \tag{1}$$

This means that when the input $a < 0$ the output is 0 and if $a > 0$ the output is a. This activation makes the network converge much faster. It does not saturate which means it is

resistant to the vanishing gradient problem at least in the positive region (when $a > 0$), so the neurons do not backpropagate all zeros at least in half of their regions. ReLU is computationally very efficient because it is implemented using simple thresholding. But there are few drawbacks of ReLU neuron :

- Not zero-centered: The outputs are not zero centered similar to the sigmoid activation function.
- The other issue with ReLU is that if $a < 0$ during the forward pass, the neuron remains inactive and it kills the gradient during the backward pass. Thus weights do not get updated, and the network does not learn. When $a = 0$ the slope is undefined at that point, but this problem is taken care of during implementation by picking either the left or the right gradient.

To address the vanishing gradient issue in ReLU activation function when $a < 0$, Leaky ReLU is given as an attempt to fix the dead ReLU problem.

(2) *Leaky ReLU*

Leaky ReLU [21] was an attempt to mitigate the dying ReLU problem. The function computes

$$\sigma(a) = max(0.1a, a) \tag{2}$$

The concept of leaky ReLU is when $a < 0$, it will have a small positive slope of 0.1. This function somewhat eliminates the dying ReLU problem, and it has all the characteristics of a ReLU activation function, but the results achieved with it are not consistent.

(3) *Sigmoid*

Sigmoid [22] is also known as Logistic Activation Function. It takes a real-valued number and squashes it into a range between 0 and 1. It is also used in the output layer where our end goal is to predict probability. It converts large negative numbers to 0 and large positive numbers to 1. Mathematically it is represented as

$$\sigma(a) = \frac{1}{1 + e^{-a}} \tag{3}$$

The three major drawbacks of sigmoid are:

- Vanishing gradients: The sigmoid function is flat near 0 and 1. In other words, the gradient of the sigmoid is 0 near 0 and 1. During backpropagation through the network with sigmoid activation, the gradients in neurons whose output is near 0 or 1 are nearly 0. These neurons are called saturated neurons. Thus, the weights in these neurons do not update.Furthermore, the weights of neurons connected to such neurons are also slowly updated. This problem is also known as vanishing gradient. So, if there was a large network comprising of sigmoid neurons in which many of them are in a saturated regime, the network would not be able to backpropagate.
- Not zero centered: Sigmoid outputs are not zero-centered.
- Computationally expensive: The exp function is computationally expensive compared with the other non-linear activation functions.

(4) *Tanh*

Tanh is also known as the hyperbolic tangent activation function. Similar to sigmoid, tanh also takes a real-valued number but squashes it into a range between -1 and 1. Unlike sigmoid, tanh outputs are zero-centered since the scope is between -1 and 1. In practice, tanh is preferable over sigmoid. The negative inputs considered as strongly negative, zero input values mapped near zero, and the positive inputs regarded as positive.

Mathematically it is represented as

$$\sigma(a) = tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}} \tag{4}$$

The main drawback is that the $tanh$ function also suffers from the vanishing gradient problem and therefore kills gradients when saturated. To address the vanishing gradient problem, another non-linear activation function known as the rectified linear unit (ReLU) is given which is a lot better than the previous two activation functions and is most widely used these days.

2.2.2 CNN Training Procedure

Once a CNN architecture is built using the basic components mentioned above, in order to achieve a good processing quality, all the weights in these components need to be updated by the training procedure using training samples. The training procedure includes forward-propagations and backward-propagations. The forward-propagations is to follow the network sequence to generate predicted results from the input samples, and then the loss is calculated based on the predicted results and the ground truth data. For the backward-propagations, the weights of each convolutional kernel is updated based on the loss calculated in order to gain a better result in the next epoch.

Loss function

The loss function is used to guide the training process of a neural network. For classification problem, cross entropy loss is widely used.

(1) *Cross-Entropy*

Cross-entropy loss [6], or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. A perfect model would have a log loss of 0.

The formula is shown below.

$$Loss(y,t) = \Sigma_i t_i log(y_i) \tag{5}$$

Where t_i is the ground truth label of training instance and y_i is the prediction result of the classifier for training instance. During training, minimize cross entropy $Loss(y,t)$.

Backward Propagation

The back-propagation algorithm was originally introduced in the 1970s, but its importance was not fully appreciated until a famous 1986 paper by David Rumelhart, Geoffrey Hinton, and Ronald Williams [23]. That paper describes several neural networks where back-propagations work far faster than other approaches, making it possible to use neural nets to solve problems which had

previously been insoluble. Today, the back-propagation algorithm is still the workhorse of learning in neural networks.

Algorithm 1 Back-propogation Algorithm

Input: DateSet: $(x_n^1, y_n), n = 1, ..., N$; Epoch Number: Ep
Output: $\omega^l, i = 1, ...L$

```
for ep = 1 → Ep do
    while (x_n^1, y_n) in DataSet do
        Forward propogation to calculate x^l for each layer and the loss z
        for l = L, ..., 1 do
            Compute the derivatives of the loss to the weights of the l layer;
            Compute the derivatives of the loss to the input of the l layer;
            Updates the weights;
        end for
    end while
end for
```

Optimizer

Optimization algorithm is used to minimize (or maximize) an objective function or loss function, and it is a mathematical function to modify the internal learnable weights used in computing the predicted values. Various optimization strategies and algorithms are designed to calculate appropriate and optimum updated-values of weights, which influences the learning process and also the outputs of a model.

(1) *Stochastic Gradient Descent(SGD)*

SGD [24] updates model weights (Ψ) in the negative direction of the gradient (ϕ) by taking a subset or a mini-batch of data of size (m):

$$\begin{aligned} \phi &= \frac{1}{m} \bigtriangledown_{\Psi} \Sigma_i Loss(f(a_i; \Psi), t_i) \\ \theta &= \Psi - \varepsilon \times \phi \end{aligned} \tag{6}$$

The network outputs is represented by $f(a_i; \Psi)$ where a_i are the training data and t_i are the training labels, the gradient of the loss $Loss$ is computed with respect to model weights Ψ. The learning rate ε determines the size of the step that the algorithm takes along the gradient (in the negative direction in the case of minimization and in the positive direction in the case of maximization).

Now due to these frequent updates, weight updates have high variances, which causes the loss function to fluctuate to different intensities. This is actually a good thing because it helps us discover new and possibly better than local minima , whereas Standard Gradient Descent will only converge to the minimum of the basin as mentioned above. But the problem with SGD is that due to the frequent updates and fluctuations, it complicates the convergence to the exact minimum and will keep overshooting due to the frequent fluctuations .

(2) *Momentum*

Momentum [25] accumulates exponentially decaying moving average of past gradients and continues to move in their direction:

$$\begin{aligned} v &= \zeta \times v - \varepsilon \bigtriangledown_{\Psi} (\frac{1r}{m} \Sigma_i Loss(f(a_i; \Psi), t_i)) \\ \theta &= \theta + v \end{aligned} \tag{7}$$

where ζ is the momentum weight. Thus step size depends on how large and how aligned the sequence of gradients are, common values of momentum parameter alpha are 0.5 and 0.9.

(3) *Nesterov Momentum*

Nesterov Momentum is inspired by Nesterov accelerated gradient method [26] :

$$\begin{aligned} v &= \zeta \times v - \varepsilon \bigtriangledown_{\Psi} (\frac{1}{m} \Sigma_i Loss(f(a_i; \Psi + \zeta \times v), t_i)) \\ \theta &= \theta + v \end{aligned} \tag{8}$$

The difference between Nesterov and standard momentum is where the gradient is evaluated, with Nesterov's momentum the gradient is evaluated after the current velocity is applied, thus Nesterov's momentum adds a correction factor to the gradient.

2.3 Relevant Work for Object Recognition

In this sub-chapter, serval widely-used CNN architectures for object recognition are investigated and the relevant work trying to improve the computation efficiency is more focused.

2.3.1 AlexNet

AlexNet [27] is the first widely focused and used convolution neural network(CNN) in image processing. It was proposed in 2012 and famously won the 2012 ImageNet LSVRC-2012 competition by a large margin (15.3% VS 26.2% (second place) error rates). Because of the good results achieved by AlexNet, many researchers focus on CNN to deal with problems in image processing.

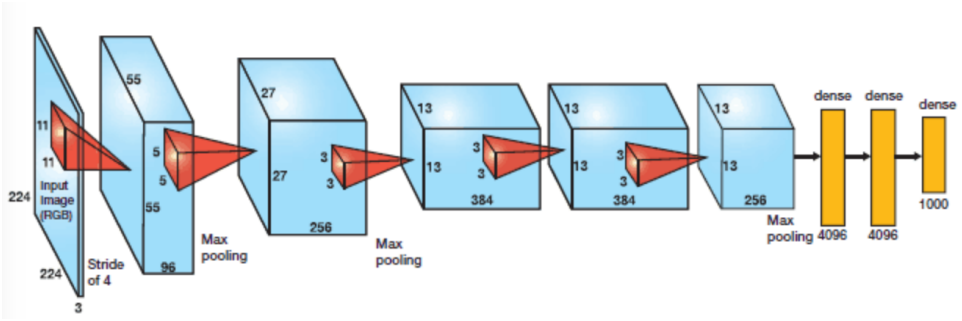


Figure 2.4: The Architecture of AlexNet [27]

AlexNet contains 5 convolution layers and 3 fully connected layers and the architecture is shown in Fig 2.4. A visualization of features in a trained AlexNet is also given in Fig. 2.5.

The main contributions of AlexNet are:

- It is the first CNN applied on the ImageNet dataset which contains of more than 1.5 million images in 1000 classes. This reveals the CNN has the ability of updating a large number of parameters to represent huge amounts of various features.
- GPUs were used to accelerate the network training and testing. For the research of neural network in 1980s, the limited computation resources blocked the development of the neural network. AlexNet decreased the training time from serval months to serval days, which allows that various networks and applications can be designed in a short time.
- The network building methods used in AlexNet are still useful today.
    - Using ReLU instead of $sigmod$ or $tanh$ to add non-linearity.
    - Using data augmentation and dropout to deal with overfitting.
    - Using normalization to reduce the error rates.

Figure 2.5: Visualization of features in a fully trained AlexNet [28]. For layer 1, the feature maps are easy to visualize. For layers 2-5, the top 9 activations in a random subset of feature maps, based on training dataset, are projected down to pixel space using deconvolutional methods.

2.3.2 VGGNet

This architecture is from VGG group, Oxford and the structure is shown in Fig. 2.6 [14]. It makes the improvement over AlexNet by replacing large kernel-sized filters(11 and 5 in the first and second convolutional layer, respectively) with multiple 3X3 kernel-sized filters one after another. In other words, all the convolution layers in VGGNet are uniformed with standard convolution layers with 3X3 kernels. VGGNets have the two versions, VGG-16 having 16 stacked convolution layers and VGG-19 having 19 similar layers. The multiple stacked smaller size kernel helps to extract more complex features using a lower computation cost, with respect to the larger size kernel.

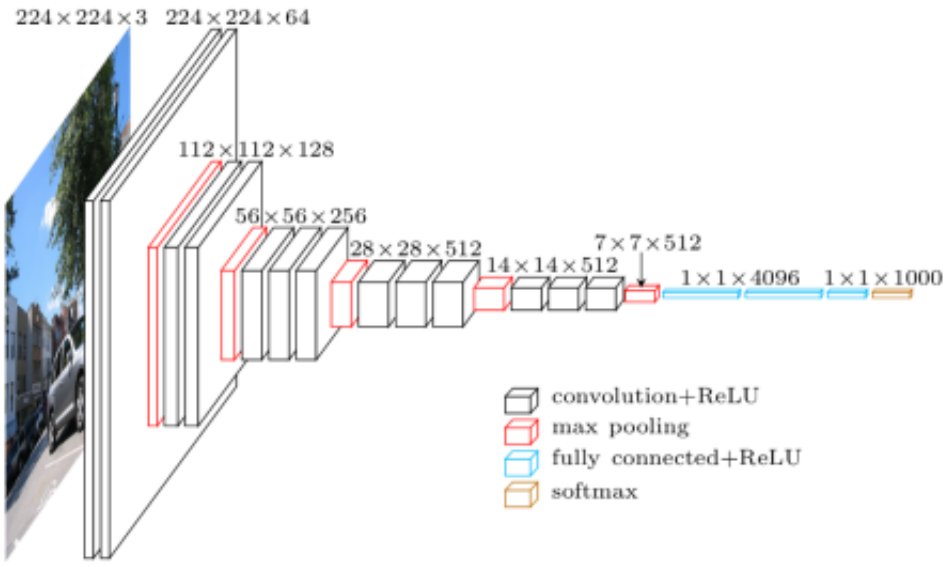


Figure 2.6: VGGNet Example [14].

2.3.3 ResNet

In 2015, most researchers think that increasing the depth should increase the accuracy of the network, as long as over-fitting is taken care of. However, in most cases, for plain networks, the classification error rate will be higher when the networks are more deeper. This is called degradation problem. The reason is that the signal required to modify the weights, which is from the end of the network by comparing ground-truth and prediction, becomes very small at the earlier layers, since the increase of the depth. In other works, the weights in the earlier layers are hard to be updated. This is called gradient vanishing.

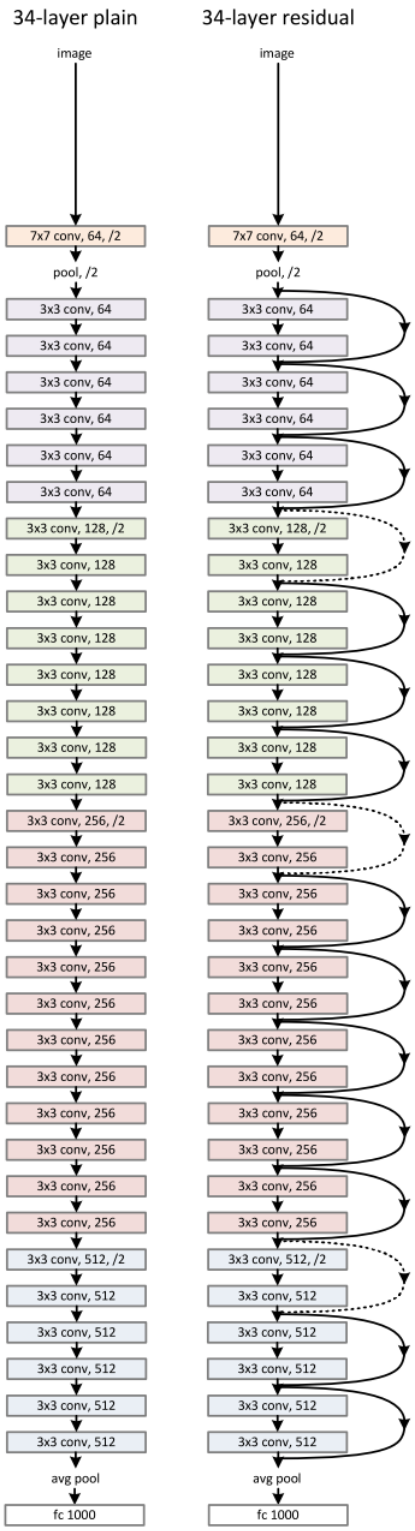


Figure 2.7: ResNet architectures for ImageNet [15]. **Left**: a plain network with 34 layers. **Right**: a residual network with 34 layers.

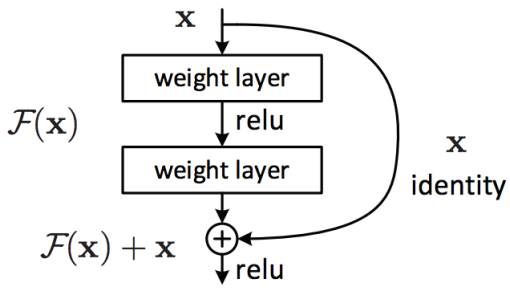


Figure 2.8: A residual module [15].

Residual networks (ResNets) [15] is proposed to allow training of such deep networks by constructing the network through modules called residual models. The comparison of Resent and a plain net is illustrated in Fig. 2.7. and the residual module is shown in Fig. 2.8. The intuition is to add an identify mapping between the input and output, thus the convolution layer only needs to learn the residual. Furthermore, this direct path between the input and the output allow the loss data directly pass to the earlier layers without vanishing. From signal processing view, the output result is enhanced by the addition with the original input information. Moreover, these input information can be further enhanced in the later layers. The identify mapping idea in the residual modules is critical to improve the computation efficiency because of the feature reusing. The residual path allows the CNNs to be deeper to achieve a good processing quality, however, normally a large a number of weights in these deep networks are redundant and can be reduced. Hence, some research effort, such as pruning, has been made to further improve the efficiency.

2.3.4 Pruning

Pruning neural networks [29, 30, 31] is an old idea to improve the computation efficiency. The idea is that among all the weights in a network, some are redundant and do not contribute a lot to the output. If one could rank the kernel weights in the network according to how much they contribute, the low ranking weights could be removed from the network, resulting in a smaller and faster network. Here are some examples of pruning VGGNets and ResNets [14, 15]. However, the problem is that the ranking methods are difficult to design and the pruning may result in a big accuracy drop. Moreover, the training of a large network and the fine-tuning of a pruning

network are a huge computation volume. Since these drawbacks of the pruning method, depthwise convolutions are used in CNN to build the computation-efficient CNN architectures.

2.3.5 Xception

Xception [32] is a convolutional neural network architecture based entirely on depthwise separable convolution layers. The detailed architecture is shown in Fig 2.9. It follows the hypothesis that the mapping of cross-channels correlations and spatial correlations in the feature maps of convolutional neural networks can be entirely decoupled.

The idea of using depthwise convolution to replace the full standard convolution is good and it does improve the computation efficiency. However, in most conditions, the cross-channels correlations and spatial correlations in the feature maps need to be calculated in a specific way according to the nature of data, which needs the cooperation with the data organization method.

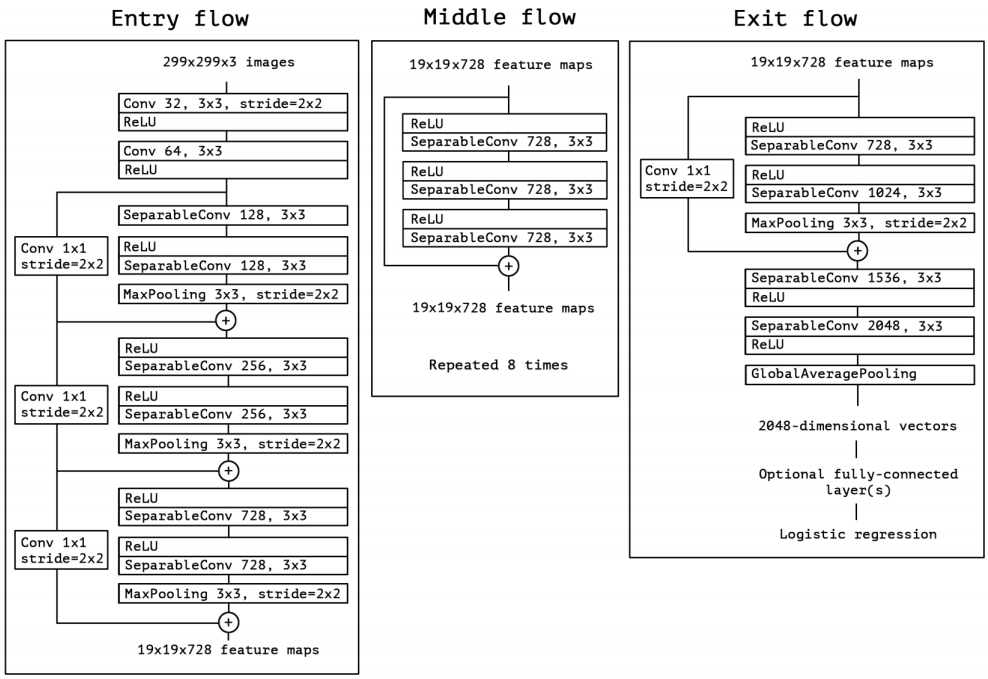


Figure 2.9: Detailed Xception Net [32].

2.3.6 MobileNetV2

MobileNetV2 [33] is designed to be a computation-efficient network trying to reduce the cost. The basic idea is to replace a full standard 3X3 convolution layer with an 1×1 convolution layer and a 3×3 depthwise convolution layer, indicated by DWConv in the Fig. 2.10. The structure of the MobileNetV2 module is also shown in Fig. 2.10. Moreover, it also introduces the residual path to improve the performance.

It is a good idea to improve the computation efficiency by replacing the standard convolution layers with several separate layers. A full standard convolution always combines two operations together: the data organization and the feature extraction, but the operation of data organization should be done in a specific way determined by the nature of data. Hence, using the depthwise convolutions to extract the features and using the 1×1 convolution to organize data can improve the computation efficiency. However, it should be also noted that the method of data organization can be further designed in a more careful way, and the computation efficiency should be even higher.

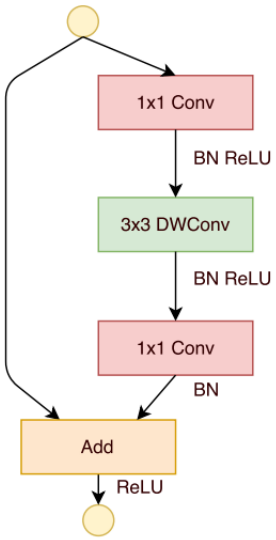


Figure 2.10: A MobileNetV2 module [33]. DWConv stands for depthwise convolution.

2.3.7 ShuffleNet

The structure of ShuffleNet [34] module is given in Fig. 2.11. The design idea is similar to the MobileNetV2. The main difference is that the 1×1 standard convolutions in MobileNet are replaced by the 1×1 group convolution and a channel shuffle.

With respect to MobileNetV2, ShuffleNet uses a more specific way to organize the input data. It is done by group convolution and channel shuffle, thus the computation volume can be reduced. However, if the data organization operation can be designed based on the nature of the data from image processing view, the expected result should be better.

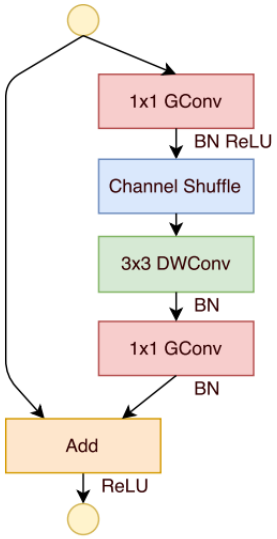


Figure 2.11: A ShuffleNet module [34]. DWConv stands for depthwise convolution.

2.4 Summary

In this chapter, a brief description of the fundamentals of CNNs is given. The basic components, $i.e.$ different computation layers, are presented. Moreover, since CNNs have a huge amount of learnable weights to extract various features, the training procedure including forward-propagation and backward-propagation is also introduced to determine the weights by using training samples.

Study on relevant CNN architectures for object recognition is also presented. A commonly used CNN architecture, such as VGGNet, uses multiple uniformed 3X3 kernels for simplicity. This network and its variations are widely used as references to evaluate the performance of other

networks. Based on VGGNet, ResNet was proposed to allow training a very deep network by using the residual modules. From image signal processing point of view, in each residual module, the input signals are combined with the convolved data thus the existing features can be enhanced for re-using and new features can also be generated. Since a large CNN architecture could not be implemented on mobile devices, two potential solutions have been tried to reduce the computation volume. Pruning a trained network trying to remain the same error rate was first used. However, it is difficult to determine which paths should be pruned, and a huge volume of computation is also required for training the un-pruned network and fine-tuning the pruned network. The second solution is to build a computation-efficient architecture by replacing full standard convolutions with depthwise convolutions, since standard convolutions have a large computation cost and some of the connections are unnecessary. XceptionNet is claimed to be the first CNN architecture based on entirely depthwise convolutions. MobileNetV2 and ShuffleNet used the combination of a 3X3 depthwise convolutions and 1X1 convolutions to replace a 3X3 standard convolutions trying to achieve the equivalent processing quality. They follow the similar idea $i.e.$ splitting a standard convolution layer into serval separate layers including the depthwise convolution layer. However, the motivation comes from the computation view rather than image signal processing view, thus the data organization methods are not optimized to suit the various natures of the data in each computation stage.

Based on these investigation results, our computation-efficient CNN will follow the idea that using depthwise convolutions and other CNN components to replace the full standard convolutions. The goal is to extract maximum feature information from a given group of data using the least computation volume. With respect to MobileNetV2 and ShuffleNet, the design of proposed CNN modules and architectures will start from the analysis of image data. Then the feature extraction part and the data organization part will be carefully designed to suit the nature of data. The modules and architectures will be presented in the following chapters.

Chapter 3

Design of SdcBlock

3.1 Introduction

As mentioned previously, extracting various features related a particular class of objects from diverse background is a great challenge. CNN-based approaches are considered effective to solve this kind of problem. It is to use huge amounts of samples to progressively determine the system parameters in order to detect various object features. Normally, it requires a large number of layers, which, in consequence, needs a huge number of computation volumes to achieve a good performance.

With a view to reducing significantly the computation volume without sacrificing the processing quality in object recognition tasks, in this chapter, the design of a convolution module, named SdcBlock, is presented. It aims at extracting feature information of high density from a given set of data. Since the core operations in CNNs are 2-D convolutions, the first stage of the research in developing the efficient feature extraction methods presented in this thesis is to study different convolution modes, and then the optimization of the convolved results supported by the suitable data control. Preliminary results of the proposed algorithm are presented in [35].

In subchapter 3.2, a discussion about different convolution modes is presented. The basic scheme of the proposed module, SdcBlock $i.e.$ Successive Depthwise Convolutions Block, is described in subchapter 3.3. In subchapter 3.4, a number of variations of SdcBlock for different computation purposes are presented.

3.2 Different Convolution Modes

The purpose of a convolution in CNN is to generate feature vectors based on the input data, in a way that the information relevant to the object features is extracted, composed, strengthened, and/or concentrated, while filtering out those irrelevant. To build effective convolution modules with a maximized capacity of extracting critical feature information, it's important to look into the different convolution modes and to direct the data to the appropriate convolution operations. In general, an input data of N_I channels can be transformed to an output data of K channels by a convolution with K kernels in one of the following three modes.

- Standard convolution. In this mode, each of the K convolution kernels is applied to all the N_I input channels to generate one output channel. It can generate an output result involving all the input information. It is often used when all the inputs are correlated.

- Group convolution (G-Conv). The N_I input channels are divided into g groups, and K convolution kernels are also divided into g sets. Each of the input data groups is convolved with a set of K/g kernels. The standard convolution can be seen as the specific case, with $g = 1$, of the group convolution. A group convolution can reduce computation volume by a factor of g, compared with the standard convolution. Furthermore, this convolution mode gives a flexibility in process because the inputs can be divided into serval groups according to their correlation levels to generate outputs containing different feature information.

- Depthwise convolution (DW-Conv). It is, in fact, the conventional 2-D spatial convolution with the weights progressively updated in the training process. In the context of CNN, it can be seen as another special case of the group convolution, in which $g = N_I$, *i.e.*, one channel per group, and each of the K kernels is applied only to one input channel, requiring $g = N_I = K$. Depthwise convolutions, if performed successively to a single input channel, can generate feature maps of different orders.

With given N_I and K, the standard convolution mode is the most computation-demanding, as it generates each of the output vectors based on the data sampled from all the input channels. On the contrary, the depthwise convolution mode is the least computation-demanding, and each of its output vectors is produced exclusively based on the data of a single channel. The group convolution

mode allows a flexibility for a desirable data organization. Improving the computation efficiency in this module is related to an appropriate choice of the convolution mode.

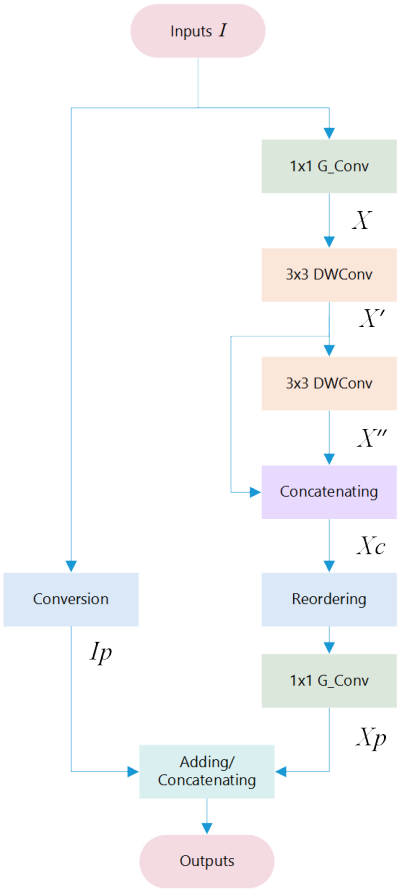


Figure 3.1: SdcBlock Modules. G_Conv denotes group convolution and DWConv denotes dethwise convolution.

3.3 SdcBlock

The basic scheme of the proposed convolution module, named SdcBlock (Successive Depthwise Convolution Block), is illustrated in Fig. 3.1. It is designed to make the best use of the input data to generate critical feature vectors of different orders. Appropriate convolutions supported by optimized data management are implemented to produce various feature information of high density.

In this module, three functions, Successive Depthwise Convolutions (Sdc), data preparation for the convolutions and arrangement of the convoluted data to form the output are performed. These functions are described in the following subsections.

3.3.1 Successive Depthwise Convolutions(Sdc)

Object recognition needs feature information of different natures that can be generated by filtering operations of different orders. Successive convolutions are used in this module to efficiently generate such features. These convolutions must be applied solely to the same set of data, for which only the depthwise convolution mode is suitable. Hence the Successive Depthwise Convolutions(Sdc) make the pivotal part in this module. They are are performed by the two boxes, indicated by *3×3 DWconv*, as illustrated in Fig. 3.1. I denotes the input of the module and X denotes the input of the depthwise convolution. X' and X'' denote the results of the first and second order filtering operations, respectively. It should be mentioned that each channel in X can be formed either from a single channel or a group of multiple channels in I.

If the module is placed in an early processing part of a network to handle raw image data, the successive depthwise convolutions will generate the first and second order gradient maps in order to obtain various low-level features. If the module is placed in the middle or final stages of a CNN, these operations will produce vectors containing higher order feature information. In each case, the extracted information of each of the two convolutions needs to be carefully preserved. Hence, the two sets of the convolution results are concatenated, instead of being summed up.

In the basic version of SdcBlock illustrated in Fig. 3.1, the kernel size of the two successive depthwsise convolutions is 3×3. It can, however, be extended to a larger size depending on the nature of the input data. Batch normalization and a non-linear function, such as ReLU, are applied after each of these two convolutions for the following operations.

3.3.2 Data Preparation for the Successive Depthwise Convolutions(Sdc)

Since the core operation in the proposed module consists of the successive depthwise convolutions, it is important to transform the input data I to a good form for the two depthewise

convolutions. Thus, a conversion in form of a 1×1 group convolution is applied to the input data I, as illustrated in Fig. 3.1, to covert I into the set X for the purposes stated as follows.

- Scaling of the input elements. This 1×1 convolution provides a scaling weight to the input channels of the module and these scaling weights can be adjusted to facilitate the feature extraction in the succeeding convolutions.

- Conversion of the input channel numbers. By applying the 1×1 group convolution, the input data I with N_I channels can be converted to X with $E \times N_I$ channels, a desirable number for the following successive depthwise convolutions.

- Grouping the input channels. As mentioned previously, the output of the two successive convolutions, namely X' and X'', are exclusively produced from the input set X. Each channel in X can be formed by a single channel, or a group of multiple channels from the input data I. In other words, by grouping multiple channels from I, more input information is involved in each channel in X and the succeed depthwise convolutions can generate feature maps containing information of higher density.

This component of data preparation converted the input data I to the data set X to meet the requirements of the successive depthwise convolutions. It allows an effective adjustment in the data format and makes it possible to involve multiple channels of data in depthwise convolutions to optimize the filtering process.

3.3.3 Data Arrangement to Generate the Output

This component in the SdcBlock is used to handle the two sets of data, $i.e.$ I, the input to the module and X_c, the concatenated data from the successive depthwise convolutions. In order to generate the output data of higher density and more varieties, an appropriate combination of these two sets is critical. In general, they can be combined in two different ways.

- *Addition*. The convolved data are added to the input data of the block, which can result in feature enhancements and/or a generation of new features.

- *Concatenation*. The two sets of data are concatenated. It is suitable in feature reusing and preserving feature information of the three different orders for further processing.

The decision to use the addition or concatenation is based on the nature of the input data I and the purposes of the processing in the block. In order to implement the combination, the two sets of data ,I and X_c, need to have the same dimensions and formats. Hence, an 1×1 convolution, is placed, as shown in Fig. 1 to compress or extend the concatenated data X_c to a desirable data form X_p. Meanwhile, it can also provide a scaling function to the data. Also, a similar process is applied to I, in order to meet the requirements of dimensions and channel sequences.

It should be mentioned that, as the data X' and X'' produced respectively by the different depthwise convolutions are concatenated, they are sequenced naturally in X_c. A box of reordering is placed to rearrange the sequence of these channels before the 1×1 group convolution. It is important, in many cases, to ensure that each group of the channels in this group convolution have the feature information of both the first and second orders.

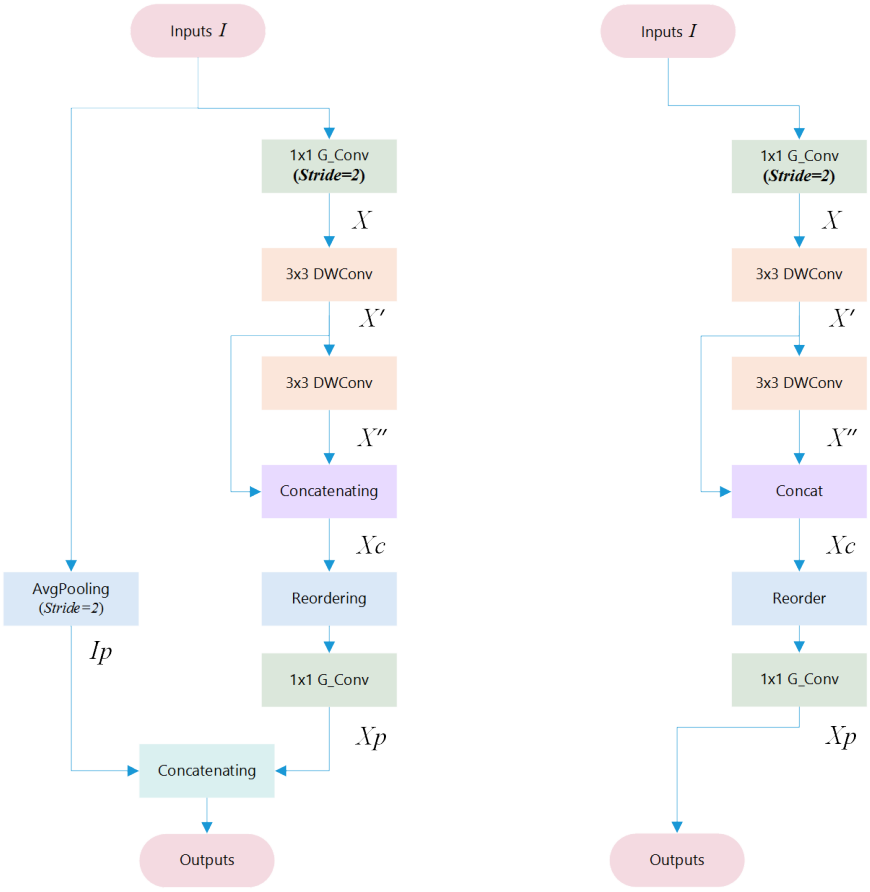
3.4 Deviations From the Basic SdcBlock

Some details of the SdcBlock can be specified by users for specific computation purposes. For example, the combination of the input data I and the convolved results X_c can be either addition or concatenation. Also, before being combined, I can be converted to match X_c by an 1×1 convolution or a particular way of pooling. Another important element to be determined is $Stride$, a hyper-parameter in convolutions. It can be $Stride = 1$ or $Stride = 2$.

As filtering operations with $Stride = 1$ normally give more precise results than those with $Stride = 2$, the convolution with $Stride = 1$ is used in the basic version of the SdcBlock. In this version, the dimension of feature map in each channel, $i.e.$ its height and width, remains the same during the whole processing of this module.

In some cases, the module with $Stride = 2$ is used and the main motivation is to reduce the dimension of feature maps in order to improve computation efficiency in object recognition tasks. Because there is information redundancy in the feature maps, the dimension size could be reduced without high risk of missing information. Meanwhile, the dimension reduction contributes to decrease computation cost and increase the convergence speed. SdcBlock with $Stride = 2$ can be used in CNN processes to simplify the computation and to improve the efficiency. To be specific, $Stride = 2$ is usually applied to the first 1×1 convolution. In the following examples, the size of

a input will be reduced half per side of its original height and width at the first convolution thus all the following computation cost can be also reduced. In Fig. 3.2(a) and Fig. 3.2(b), two examples of SdcBlock with $Stride = 2$ are illustrated, named SdcBlock-S2 and SdcBlock-S2-F, respectively.



(a) SdcBlock-S2 (b) SdcBlock-S2-F

Figure 3.2: SdcBlock Modules(*Stride=2*)

In case of SdcBlock with *stride = 2*, the SdcBlock-S2 shown in Fig. 3.2(a) is the most commonly used. The hyper-parameter $Stride = 2$ is used in the first group convolutions only while $Stride = 1$ is applied in the other convolutions. In order to re-use low-level feature information and compensate the eventual information lost due to dimension reduction, a concatenation of sampled input data and the rearranged convolved data is performed.

To perform this concatenation, the dimension of the input data should be converted to match that of the convolved results. In this case, an average pooling is applied to perform the conversion. The

average pooling can preserve most feature information of the input data while the size is reduced to one quarter. However, the average pooling can be replaced by another conversion approach.

In this version, the output contain both the sampled input information and the successive depthwise convolution results. Furthermore, the importance of the input data compared to the convolved results, $i.e.$ the proportion of the input data in the output, can be adjusted according to the data natures and computation purposes.

Another version of the SdcBlock($Stride = 2$), named SdcBlock-S2-F(Feature), is shown in Fig. 3.2(b). In some cases, the results of successive depthwise convolutions make the output of the block thus the concatenation and the conversion of the input data in SdcBlock-S2 can be removed. This version is called SdcBlock-S2-F, a special case of SdcBlock-S2.

In this version, the output are exclusively from the successive depthwise convolutions without being combined with the input. It can be useful in the later stages of a network, where almost all of low-level feature maps such have been converted to the high-level feature maps. There is almost no need for low-level feature reusing thus the concatenation of the input data can be removed.

3.5 Summary

In this chapter, a CNN module, named SdcBlock, has been proposed. In this module, optimized successive depthwise convolutions, supported by appropriate data managements, are applied to generate vectors containing varieties of feature information of high density. The successive depthwise convolutions can generate features of the first and the second order filtering. Meanwhile, the data management can not only organize the input data for the successive depthwise convolutions but also combine and compress the feature information of different orders. These methods can improve computation efficiency and reduce the computation volumes. Furthermore, some variations of this module has also been given for different computation purposes. This module can be adjusted easily to meet the computation requirements of different network stages in order to form an end-to-end network for object recognition tasks.

Chapter 4

Design of SdcNet

4.1 Introductation

As mentioned previously, SdcBlock has been proposed as a feature extraction module aiming at improving the computation efficiency of CNNs in object recognition. In this chapter, the design of SdcNet, a CNN architecture, is presented. This architecture is mainly composed of cascaded SdcBlocks. The objective of this work is to make good use of SdcBlocks to optimize the computation in order to obtain desired results with particular resource limitations. Since the SdcBlock is modularized, one can use a number of hyper-parameters in each SdcBlock to achieve the optimization.

In this chapter, the general structure of SdcNet, in terms of processing functions, is presented in Subchapter 4.2. Three different networks are described in detail, as design examples of SdcNet, in the following subchapter.

4.2 Processing in SdcNet

The general scheme of the proposed architecture, SdcNet, is shown in Fig. 4.1. The input of this network is images and the output is the data representing the recognition results, such as object classification labels. The network can be divided into three function parts:early processing, feature extraction and decision processing. The first and second parts are to extract low and high level feature information from the input images. This feature information is used in the third part to

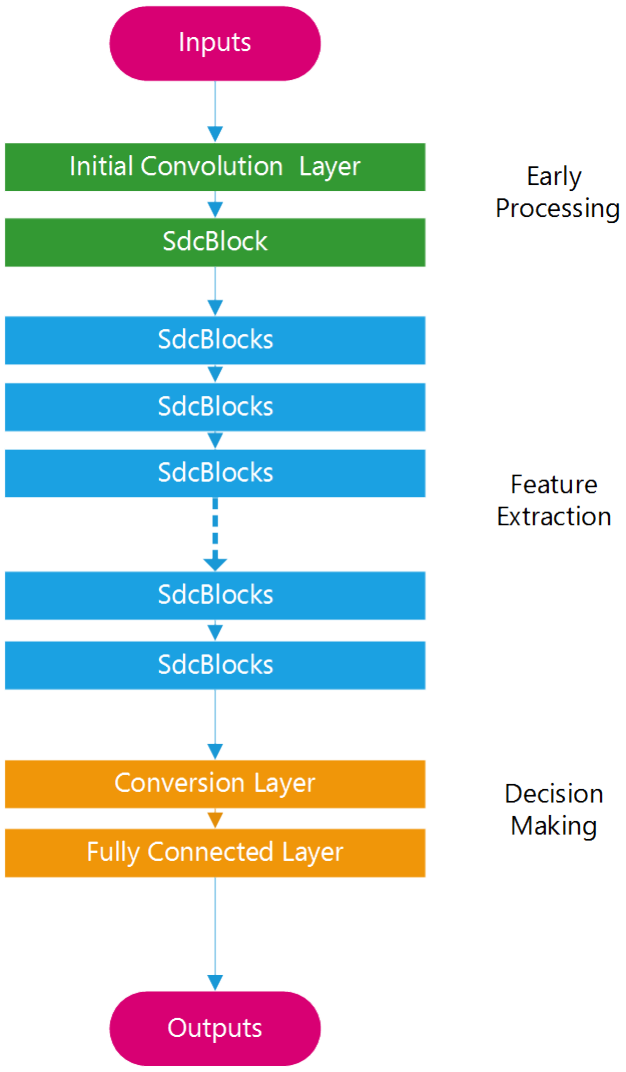


Figure 4.1: General scheme of SdcNet.

generate the final decisions. The decision processing part is composed of a conversion layer and a fully connected layer, as shown in Fig. 4.1.

Since the quality of the extracted feature information applied to the third part can make significant difference in the final decisions, it is critical to design the early processing and feature extraction parts to extract varieties of feature information and to present it in a high-density form. The design of SdcNet is to make good use of SdcBlock by determining a number of hyper-parameters in each block according to the input signal and the specific computation purpose. These hyper-parameters are listed as follows.

- The group number g. In a SdcBlock, it controls the partition of the data channels in the group convolution. It is used to organize the data and should be determined according to the nature and correlation of the data channels in each layer. Meanwhile, convolutions with a larger value of g decrease the amount of computation.

- The number of output channels. In each SdcBlock, the output channels should be sufficient to extract enough varieties of feature elements in order to decrease the risk of information lost. However, it should be mentioned that excessively increasing the number of output channels can not only result in a larger computation load, but also can reduce the information density and decrease the rate of convergence.

- The hyper-parameter $Stride$. It can be either $Stride = 1$ or $Stride = 2$ in a SdcBlock. A convolution with $Stride = 1$ can generate more precise results than that with $Stride = 2$. However, the latter downsizes the feature maps, which can concentrate the feature information and also reduce the computation volume by three quarter. It should be noted that convolutions with $Stride = 2$ may result in some data loss.

- The combinations of the data from different filtering operations. As mentioned previously, it can be either concatenation or addition.

- The way of reordering convolved data. The purpose is to reorder the data channels produced by the proceeding operations to match what required in the succeeding operations in order to obtain more kinds of feature information. In other words, these data are re-indexed based on the nature and correlation of the data in each SdcBlock. In the examples of SdcNet presented in Subchapter 4.3, details of the reordering are described.

- The expansion number E. As mentioned in Chapter 3, the first group convolution in a SdcBlock will convert the input data of N_I channels to $E \times N_I$ for the following successive depthwise convolutions.

The early processing part of a SdcNet contains an initial convolution layer and one or more SdcBlocks, as shown in Fig. 4.1. In most cases, the input of this part is a color image composed of three channels, such as RGB ones. In this processing part, early feature information is extracted,

preserved, organized and enhanced. In terms of the processing quality, it is necessary to generate a sufficient number of feature maps to catch most of the low-level features critical for the later processing. However, in order to improve the computation efficiency, it is essential to minimize the number of convolution kernels. In the initial convolution layer, if standard convolutions were used, a large number of kernels should been needed to capture various color features. In the design of SdcNet, based on the fact that the number of variations in a color channel is much smaller, the initial convolution is designed to be a group convolution with $g = 3$ so that each group of convolution kernels is applied to an individual color channel to generate monochromatic features. Thus one can use a smaller number of channels to perform a low-level feature extraction with a reduced risk of information lost. The SdcBlock in this part is used to further process these feature elements by successive depthwise convolutions. These elements are organized and enhanced by selecting suitable hyper-parameters for the later processing.

The feature extraction processing part is a stack of SdcBlocks as shown in Fig. 4.1 and it is the main processing part in the SdcNet. The input of this part is the data representing low-level features generated by the early processing, and the data are in the form of a small number of large-dimension channels. The processing in this part is to generate, from these low-level features, a sufficient number of small-dimension maps representing high-level features, ready to be applied to the decision processing. In this part, the early feature elements which might contribute to the final decisions are normally dispersed widely in the input channels. These elements should be transformed to the form of condensed information organized in different data channels each containing a specific kind of information. In order to achieve this target, the output channels of each processing stage increase progressively. SdcBlocks with $Stride = 2$, instead of pooling, are applied in some stages to downsize the feature maps, while the increase of the output channel number in these stages compensates for the eventual information lost. However, it should be noted that the successive use of SdcBlokcs with $Stride = 2$ should be avoided. Furthermore, other hyper-parameters, like the group number g, should be carefully chosen to optimize the processing performance, as group convolutions require less computation volume if g is larger, which suits the condition of low computation cost.

In the following subchapter, three SdcNets are presented, based on the design ideas described in this subchapter. In particular, the way of choosing hyper-parameters to better suit the application requirements is described.

4.3 Examples of SdcNet Design

In this subchapter, the design of three SdcNets, named SdcNet-S, SdcNet-M and SdcNet-L, with different emphases for the object classification tasks is presented in details. The network SdcNet-S is considered as the standard version of SdcNet and it has the lowest computation cost among the three and produce an acceptable processing quality. SdcNet-M and SdcNet-L are designed to achieve a higher processing quality. For these networks, the kernel size of all the depthwise convolutions is 3×3. In the design, the inputs of these SdcNets are the RGB images of 32x32 pixels in 10 classes from the CIFAR dataset.

4.3.1 SdcNet-S

Table 4.1: Details of SdcNet-S Configuration

Layer	Stride	Repeat Times	Group No. g	No. of Output Channels	Output size
Input image data sized 32x32, 3 channels.					
G-Conv*	1	1	3	36	32x32
SdcBlock	1	1	3	24	32x32
Stage 1	1	2	3	24	32x32
Stage 2	2	1	3	36	16x16
	1	2	3	36	16x16
Stage 3	2	1	3	72	8x8
	1	3	3	72	8x8
Stage 4	1	3	3	96	8x8
Stage 5	2	1	3	150	4x4
	1	2	3	150	4x4
Stage 6	1	1	3	300	4x4
Avg Pool**	2	1		300	1x1
FC		1		10	1x1
Complexity***	55.12 M Flops			No. of Weights: 1.04 M	

* G-Conv stands for group convolution with 3x3 kernel and $g = 3$.
** The kernel size of this average pooling is 4x4.
***The complexity is evaluated on 32×32 input images of 10 classes

The detailed configuration of SdcNet-S is shown in Table 4.1. As mentioned previously, this network can be divided into three parts, in terms of functions. The early processing part consists of a 3×3 group convolution layer and a SdcBlock. The succeeding feature extraction part is composed of sixteen SdcBlocks grouped into six stages, named from Stage 1 to Stage 6, as shown in Table 4.1. Each stage consists of one or more cascaded SdcBlocks and the number of output channel increases progressively stage by stage. A global average pooling layer and a fully connected layer are used to generate the final decisions.

SdcNet-S has the simplest structure among the three networks. It is designed to target a wide range of users. 17 SdcBlocks are used in this network. The majorities of them are the SdcBlock ($Stride = 1$), shown in Fig. 3.1. The remaining SdcBlocks are SdcBlock-S2 ($Stride = 2$), shown in Fig. 3.2, used at some stages where downsizing feature maps is needed. Among the hyper-parameters, the most important ones are determined based on the principles stated as follows.

- Group number g. The group number $g = 3$ is carefully chosen and kept identical for a simplicity through out the network. For the early processing, $g = 3$ is chosen to match the number of the input color channels so that three groups of low-level feature maps are generated, corresponding to the three kinds of monochromic information. For the feature extraction processing part, it is also appropriate to use group convolutions with $g = 3$, instead of standard convolutions, aiming at reducing the computation volume significantly. One may, however, have the concern that each convolution in this part needs to involve different data from a sufficient number of channels, in particular in later stages, to generate enough information varieties and to secure the processing quality. To address this concern, it should be noted that the correlation among the feature maps gets stronger stage by stage, while the number of channels in each group also increases progressively with the fixed group number $g = 3$. Thus there is a sufficient number of channels in each convolution in different stages.

- The numbers of data channels and the convolution $Stride$ in each stage. The initial group convolution with $Stride = 1$ generates 36 feature channels, 12 from each of the three input color channels. The 12 channels are sufficient to accommodate basic monochromic feature elements. The number of channels increases from 36 to 300 progressively, as shown in Table 4.1, meanwhile by using multiple SdcBlock-S2s, the size of each map decreases from 32×32

to 4×4, ready to produce the final decisions. The 300 channels for the decision part are chosen to produce sufficient kinds of feature information to secure a good processing quality without need for a excessive amount of computation. Furthermore, too many data channels may influence the information density and decrease the rate of convergence. SdcBlock-S2 are used as the first SdcBlocks in each of Stage 2, Stage 3 and Stage 5 to downsize the feature maps, meanwhile the numbers of channels at these stages are extended. There are at least 3 SdcBlocks with $Stride = 1$ between two SdcBlock-S2, following the principle of using SdcBlocks with $Stride = 2$ stated previously. In these stages, the data have been convolved multiple times and most of them have already been transformed to higher-level feature information sparsely distributed. The SdcBlock-S2 also contributes to produce concentrated information and improve the information density.

- Expansion number E. $E = 1$ is used in the first SdcBlock placed in the early processing part and $E = 6$ is applied to each of the other SdcBlocks in this network. The hyper-parameter E is used to expand the number of feature maps applied to the successive depthwise convolutions in each SdcBlock. The main purpose is to produce more varieties of information from the given input channels, as the depthwise convolutions need to involve more feature information. To this end, in this network, by using E greater than 1, the number of feature maps is multiplied and then applying to the depthwise convolutions. The expansion number E can be also used to adjust the computation volume to reach a appropriate trade-off between the processing quality and the computation volume. In this network, for a simplicity, $E = 6$ is used in the entire feature extraction part while it can be adjusted easily, if needed, according to computation conditions.

- Reordering data. In SdcNets, data reordering is applied to the concatenated convolved-data, and in some cases, also to the input data of a SdcBlock before combining the data of different orders. The purpose is to secure the information varieties for the succeeding processes. Shuffle is a general way to perform the reordering. It is to rearrange feature channels so that each group for convolutions has channels randomly taken from different sources. In SdcNet-S, the concatenated convolved-data are reordered by shuffle so that each group of the convolved channels contains information of different filtering orders.

- The combinations of the data from different filtering operations. For all the SdcBlock-S1, the convolved data are added to the input data in order to enhance the features and generate new feature maps. For all the SdcBlock-S2, the data of different orders are concatenated for feature-reusing.

- Data conversion in the decision part. A conversion layer should be placed before the fully connected layer and is used to convert the 4×4 feature maps to one element to facilitate the process in the fully connected layer. In this network, in order to include as much information in each map as possible when the size is reduced significantly, an average pooling layer is used for the conversion. However, it can be replaced by a convolution layer or another pooling method according to a specific computation purpose.

SdcNet-S has the simplest network configuration among the three examples presented in this subchapter. It uses a small amount of 55.12M Flops and a number of 1.03M weights. However, a good processing quality of the network is predicted since the hyper-parameters in this network is determined based on the principles of image processing. Nevertheless, if the hyper-parameters are chosen more carefully to better suit the signal natures in different stages, an even better performance can be expected.

4.3.2 SdcNet-M

The detailed configuration of SdcNet-M is shown in Table 4.2. This network is an updated version of SdcNet-S, aiming at improving the processing quality. Compared with SdcNet-S, in the design of SdcNet-M, more consideration is put in determining the hyper-parameters in different convolution layers. Since the high-level feature information for the decision part is produced from the data of basic feature elements, it is thus critical to improve the quality of information in these elements. The efforts in the design of SdcNet-M are made on the configuration of the first SdcBlock of the network and the first two stages of the feature extraction part, namely Stage 1 and 2. The rest of the network is identical to that in SdcNet-S. The updates in designing SdcNet-M are made on the principles stated below.

Table 4.2: Detail of SdcNet-M Configuration

Layer	Stride	Repeat Times	Group No. g	No. of Output Channels	Output size
Input image data sized 32x32, 3 channels.					
G-Conv*	1	1	3	36	32x32
SdcBlock	1	1	**12**	**36**	32x32
Stage 1	1	2	**6**	**36**	32x32
Stage 2	2	1	3	**66**	16x16
	1	2	3	**66**	16x16
Stage 3	2	1	3	72	8x8
	1	3	3	72	8x8
Stage 4	1	3	3	96	8x8
Stage 5	2	1	3	150	4x4
	1	2	3	150	4x4
Stage 6	1	1	3	300	4x4
Avg Pool**	2	1		300	1x1
FC		1		10	1x1
Complexity***	73.41 M Flops			No. of Weights: 1.11 M	

* G-Conv stands for group convolution with 3x3 kernel and $g = 3$.
** The kernel size of this average pooling is 4x4.
***The complexity is evaluated on 32×32 input images of 10 classes

(1) As the quality of the feature extraction is related to the number of feature maps produced in each stage, SdcNet-M has a relatively larger number of output channels in the early stages specified above, with respect to those in SdcNet-S, to accommodate more feature maps.

(2) In order to effectively use these channels, the group number g in the beginning stages of the network is carefully selected to improve the processing quality in convolutions, unlike a fixed $g = 3$ through-out the stages in SdcNet-S. Considering that the data channels generated in each convolution layer have certain correlation among them, it is important to group the channels having stronger correlation together for convolutions. A flexible group number g facilitates the data channel grouping to produce a better result. Moreover, g can also be adjusted in each stage to find the balance of the computation volume and the processing quality.

(3) In order to generate more varieties of features from given input data, the data channels are rearranged in different ways for the succeeding convolutions. In this network, with respect to

SdcNet-S, channel reordering by shuffle is used in most of the stages. However, in the first SdcBlock, the monochromic channels are rearranged in a specific way in order to control the data flow and to produce the feature maps containing specific combinations of color feature information. Moreover, it is also possible to rearrange the data channels in a specific way, in other designs, according to their natures to achieve particular purposes.

As shown in Table 4.2, the number of output channels for the first SdcBlock and Stage 1 is 36, instead of 24 in SdcNet-S. In Stage 2, there are 66 output channels in SdcNet-M, with respect to 36 in SdcNet-S. The group number $g = 12$ is used in the first SdcBlock and $g = 6$ is used in Stage 1, compared to $g = 3$ in the corresponding parts of SdcNet-S. The increase of g is to counter-balance the increase of the computation volume due to the increased number in data channels. In these stages, each convolution group does not have to involve a large number of channels. Thus applying $g = 12$ and $g = 6$ to the group convolutions in these stages is appropriate. Moreover, considering the data channels are from the three colormaps, the numbers of 12 and 6 are selected to facilitate the grouping.

In SdcNet-M, in all the SdcBlocks found in Stage 1 to 6, the data channels produced by the successive depthwise convolutions are rearranged by shuffle. However, a specific reordering approach, namely Specific Color Reordering (SCR) is proposed and applied to the first SdcBlock to control the data flow of monochromic channels in order to group them in a particular sequence, which can not be done by using shuffle. The monochromic feature maps produced by the successive depthwise convolutions are originated from the three color maps and the SCR reordering is designed to rearrange particularly these maps according their chromic characters to enhance the monochromic information as well as to produce varieties of full-color features. It is done by dividing these maps into two data sections for the succeeding convolutions. Each of the channel groups in the first section is composed of the same monochromic maps and that in the second section has mixed color maps. By doing so, the correlation among the feature channels produced from maps of the same color is well taken into consideration in producing high-level feature information, and that among the channels produced from maps of different colors is not ignored.

In this network, 36 channels are generated by the initial convolution layer from the three color maps and naturely, 12 from each color map. In the first SdcBlock, the successive

depthwise convolutions produce 72 monochromic feature maps, 36 from each of the two depthwise convolutions. These 72 channels are divided equally, in terms of numbers of channels, colors and filtering orders, into two sections. Then the 36 channels in the first section are divided into 3 groups, each has the channels of all the RGB information. While the other 36 channels in the second section are also grouped into 3, but monochromatically. By doing so, full-color feature maps are generated from the channels in the first section, and in the other section, monochromic feature information is produced and enhanced for feature re-using. These two kinds of feature information are included in the process to maximize the effective use of the early features and to generate more varieties of information. Furthermore, the input data of the block should also be rearranged in the same manner to have the same sequence before being combined with the rearranged filtered channels.

The SdcNet-M is designed to obtain a better processing quality, with respect to that of SdcNet-S, by improving the quality of the basic features in the early stages. The total computation amount of the network is 73.41M Flops with 1.11M weights, which is still a modest amount compared to those reported for similar tasks. In the following subsubchapter, another version of SdcNet is presented and it is expected to have a lower error rate without excessive computation load.

4.3.3 SdcNet-L

The configuration of SdcNet-L is shown in Table 4.3. Based on SdcNet-S, the objective in designing SdcNet-L is to obtain a better processing quality by extending feature channels in Stage 2 to 6. As more channels can accommodate more feature maps, more varieties of feature information can be extracted from the given input data. As shown in Table 4.3, the number of output channels of Stage 6 is 600, twice of that in SdcNet-S, providing the decision part with more precise feature information.

Two measures are taken to compensate for the increase of computation volume due to the extended channels. Firstly, the number of channels in the initial convolution or Stage 1 is kept the same as that in SdcNet-S before downsizing the map dimension. The channel extending starts from Stage 2, where the feature map size is reduced from 32×32 to 16×16. The second measure is to increase the group number g in all the stages, except the initial group convolution, from $g = 3$ to $g = 4$, reducing the number of parameters in order to decrease the computation volume. It should be

noted that this change does not reduce the numbers of channels per convolution group as the total numbers of the channels are increased, which avoid the degradation of the processing quality.

In this design, SdcNet-L is built to achieve the best processing quality among the three SdcNets by extending the channels. It requires 103.3M Flops and uses 2.53M weights to generate a good result. Compared with the reported CNNs for similar tasks, the computation volume is still a modest amount.

Table 4.3: Detail of SdcNet-L Configuration

Layer	Stride	Repeat Times	Group No. g	No. of Output Channels	Output size
Input image data sized 32x32, 3 channels.					
G-Conv*($g = 3$)	1	1	3	36	32x32
SdcBlock	1	1	**4**	24	32x32
Stage 1	1	2	**4**	24	32x32
Stage 2	2	1	**4**	**72**	16x16
	1	2	**4**	**72**	16x16
Stage 3	2	1	**4**	**96**	8x8
	1	3	**4**	**96**	8x8
Stage 4	1	3	**4**	**144**	8x8
Stage 5	2	1	**4**	**300**	4x4
	1	2	**4**	**300**	4x4
Stage 6	1	1	**4**	**600**	4x4
Avg Pool**	2	1		600	1x1
FC		1		10	1x1
Complexity***	103.3 M Flops			No. of Weights: 2.53 M	

* G-Conv stands for group convolution with 3x3 kernel and $g = 3$.
** The kernel size of this average pooling is 4x4.
***The complexity is evaluated on 32×32 input images of 10 classes

4.4 Summary

In Table 4.4, the three different SdcNets are summrized and the main differences among them are highlighted. SdcBlocks are used in all the networks to extract feature information. All of them requires a modest computation resource while they are expected to give a good processing performance. Hence they can be used in applications with restricted conditions of computations. Among the three networks, SdcNet-S has the simplest structure and requires the

smallest computation volume, SdcNet-L can achieve the best processing quality, and SdcNet-M is the best trade-off between processing quality and computation volume. Furthermore, since the SdcBlocks are modularized and are designed to be customized, a large variation of SdcNets can be formed to meet particular requirements, in terms of computation cost and/or processing quality by easily adjusting the hyper-parameters of the SdcBlocks.

Table 4.4: Comparison Among Three SdcNets

Network	Components	Group Numer g	Output Channels	Reordering	Flops	Weights	Comments
SdcNet-S	Initial Group Conv.	3	36	-	55.12M	1.04M	Simplest and standard version
	1-SdcBlock		24	Shuffle			
	Stage 1		24				
	Stage 2		36				
	Other Stages		72 $\sim$ 300				
SdcNet-M	Initial Group Conv.	3	36	-	73.41M	1.11M	Improved processing quality by optimizing the basic feature elements
	1-SdcBlock	12	36	SCR			
	Stage 1	6	36	Shuffle			
	Stage 2	3	66				
	Other Stages		72 $\sim$ 300				
SdcNet-L	Initial Group Conv.	3	36	-	103.3M	2.34M	Best processing quality by extending channels
	1-SdcBlock	4	24	Shuffle			
	Stage 1		24				
	Stage 2		72				
	Other Stages		96 $\sim$ 600				

Chapter 5

Performance Evaluations of SdcNets

5.1 Introduction

In order to evaluate the performance of SdcNet and to assess the computation efficiency of SdcBlock, the three SdcNets, namely SdcNet-S, SdcNet-M and SdcNet-L proposed in Chapter 4 are tested for object classification tasks. The computation complexity and the processing quality of these SdcNets have been measured and compared with the CNNs of the same tasks reported in the recent year.

In this chapter, the test conditions and the training elements of the SdcNets are described. The characteristics of the training and test process are also given. The test results of the SdcNets and the comparison with the existing CNNs are also presented.

5.2 Test Conditions

In this test, the CIFAR datasets have been used to train and test the networks to evaluate the network performance.

The CIFAR dataset [36] contains two sub-datasets, $i.e.$ CIFAR-10 and CIFAR-100. CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 testing images. The testing part contains exactly 1000 randomly-selected images from each class. The training part contain all the remaining images in random order

and exactly 5000 images from each class. Here are the classes in the dataset, as well as 10 random images from each:

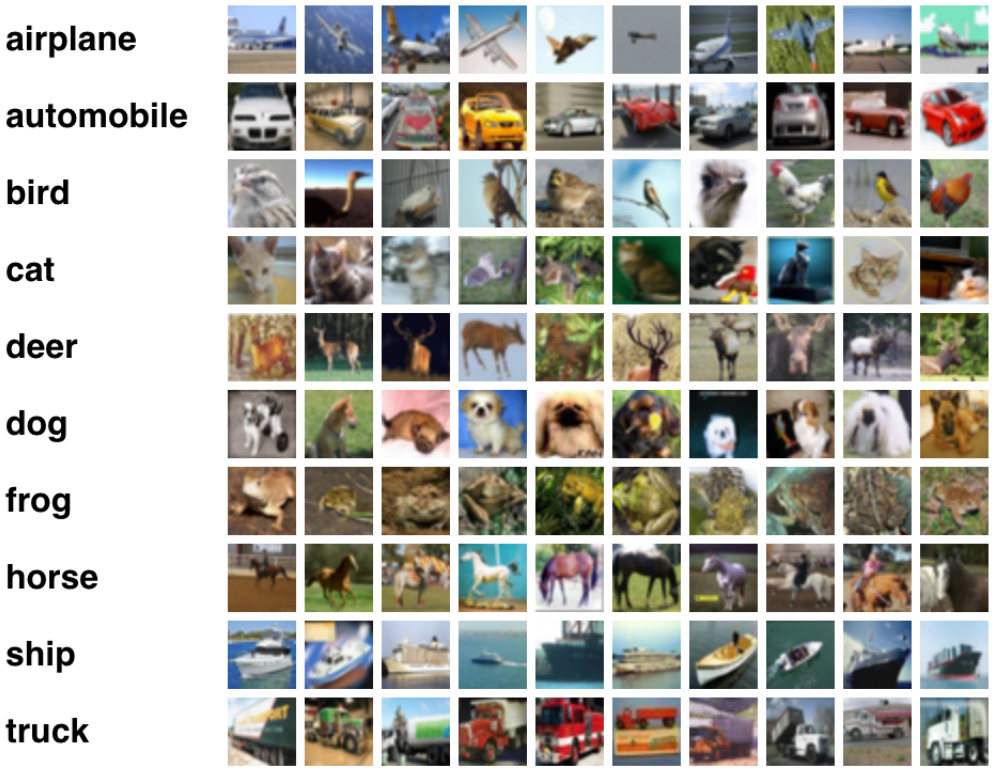


Figure 5.1: Examples of CIFAR-10 dataset

The object classes in the CIFAR datasets are completely mutually exclusive. It should be noted that automobiles and trucks are two separate classes. The class of "Trucks" contains exclusively big trucks and all the other kinds of automobiles are exclusively in the class "Automobile". Furthermore, each image sample contains exactly one object from one of the ten classes. CIFAR-100 dataset is similar as CIFAR-10 except that the number of classes in CIFAR-100 is 100 and there are 500 training images per class and 100 testing images per class.

In all the experiments, the SdcNets have been implemented using Python with Pytorch 0.4 and cuDNN 7.0 as the backends. The network training and testing processes have been done on the machines with two Nvidia Tesla P100s and an Intel Xeon E5-2683 v4 provided by Compute Canada.

Based on the CIFAR datasets and the computation conditions, the details of all the training elements have been taken into consideration to perform the tests of SdcNets in order to evaluate the network performance.

5.3 Training Detials

As the computation volume required in the SdcNets is modestly ranged from 55M to 103M Flops, a relatively small number of epochs to complete the training procedure can be expected. To this end, serval training elements should be taken into consideration in order to optimize the training procedure of these SdcNets.

- Training epochs and batch size. First of all, the number of training epochs is chosen to be 300 [37], which is commonly used for similar tasks. For the batch size, because the weight updating is relied on the statistic loss of each batch, the batch size should be large enough to secure the accuracy of the statistic loss, especially at the later part of the training procedure. Besides, with a given number of epochs, the batch size should be small enough to secure sufficient updates of the weights. Moreover, an excessively large batch size will lead to excessive computation complexity in the training process, which requires huge amounts of computation resources. Hence, after serval preliminary trials, a fixed batch size of 128, for a simplicity, has been chosen in the experiments and it has been proved to be appropriate in the computation of the given tasks.

- Optimizer. It is essential to control the magnitude of weight updating each time based on the learning rate and the statistic loss by selecting a suitable optimizer. As one of the most commonly used optimization methods, stochastic gradient descent (SGD) [24] has been chosen to minimize the loss in the experiments. Compared to the adaptive optimization methods such as Adam, SGD can achieve a smaller residual error and a better generalization performance. [38] Moreover, in order to increase the convergence rate, Nesterov momentum [26] with a momentum weight of 0.9 and a weight decay of 0.0001 has also been adopted.

- Learning rate. The learning rate should be variable in the course of the training. In the beginning epochs of the training procedure, the learning rate could be relatively large to

accelerate the loss reduction. While in the later epochs, a relatively small learning rate can minimize the loss with an appropriate optimizer such as SGD to gain a small residual error. Hence, in the training of SdcNets, the learning rate has been a variable, $i.e.$, from 0.1 using cosine-shape decreasing method [39] in the 300 epochs. The curve of the loss rate is illustrated in Fig 5.2.

- Loss function. Cross entropy loss function [6] has been used to calculate the loss in this work like many other networks to calculate the loss to solve the classification problems.

- Data augmentation. The purpose of data augmentation is for a better generalization performance by increasing the varieties of the training samples [40] In other words, by increasing the diversity in a given training samples used in different epochs, the data augmentation attempts to reduce the difference between the training loss and test loss. In the training process, the images have been padded with four zero pixels on each side, and then a 32x32 crop has been randomly sampled from the padded image. Then half of these images have been randomly selected and flipped horizontally. For the testing process, the original images of 32x32 have been used to evaluate the network.

- Weights initialization. In order to avoid the divergence in the training procedure, the weights of the network have been initialized by using the method reported in [41], $i.e.$, the weights being initialized in such a way that the standard deviation between inputs and outputs is the same in each layer.

Based these training details, the training behavior is given in the following sub-chapter.

5.4 Training Behavior

In order to validate a network, the first step is to obtain its training behavior. In general, the training behavior can be captured by the characteristics of the training loss versus training time, indicated by the index of the training epochs. An ideal training process should have a short convergence time, displayed by a quick decent of the loss, and a small residual error. In order to obtain the the training loss characteristics, 50,000 CIFAR-10 training samples containing 10 object classes have been applied to the three SdcNets, namely SdcNet-S, SdcNet-M and SdcNet-L.

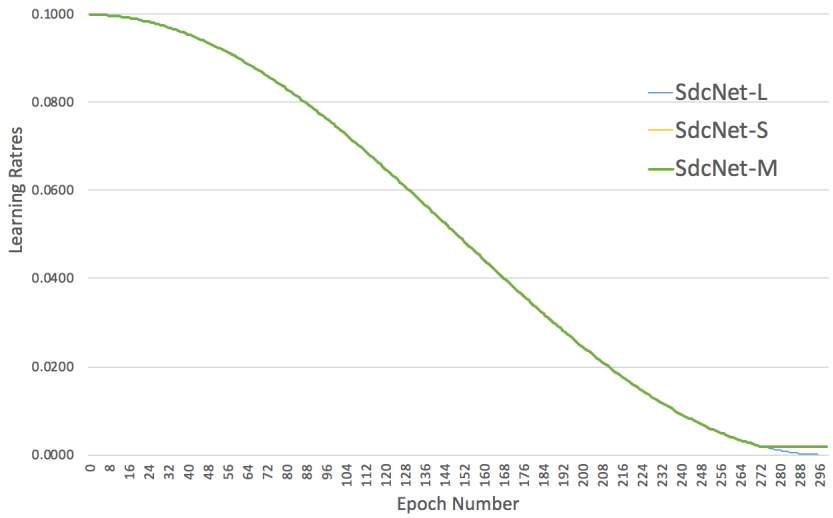


Figure 5.2: Variable Learning Rate in the Training Epochs. The minimum learning rate for SdcNet-S and SdcNet-M is 0.002 and that for SdcNet-L is 0.

The characteristics of training loss versus training epochs are given in Fig. 5.3. The three characteristics given by the three SdcNets are almost overlapped, and the training loss is around 1.5 initially. After 16 training epochs, the loss has decreased to a level below 0.3, $i.e.$, less than 20% of the initial loss. In other words, more than 80% loss has been reduced in the first 5% training time of the 300 epochs, which demonstrates a fast convergence in the network. It has been confirmed that the effectiveness of the relatively large learning rate and the appropriate set of the optimizer in the training lead to a quick loss decent. After the training of 256 epochs, all the networks have approached to their steady states with the loss of around 0.002. From the epoch 256 to the end, the network is fine-tuned to achieve the best processing quality. The final residual error is around 0.002. It has also been confirmed that the variable learning rate and the optimizer are suitable to reach a small residual error.

In different epochs during the training process, the classification error rate has also been measured. The classification error rate (ER) is defined as follows.

$$ER = \frac{P_e}{T_N} \tag{9}$$

where T_N is the number of the test images and P_e is the number of incorrectly predicted images. Fig. 5.4 illustrates the classification error rates of the training samples has decreased with the training

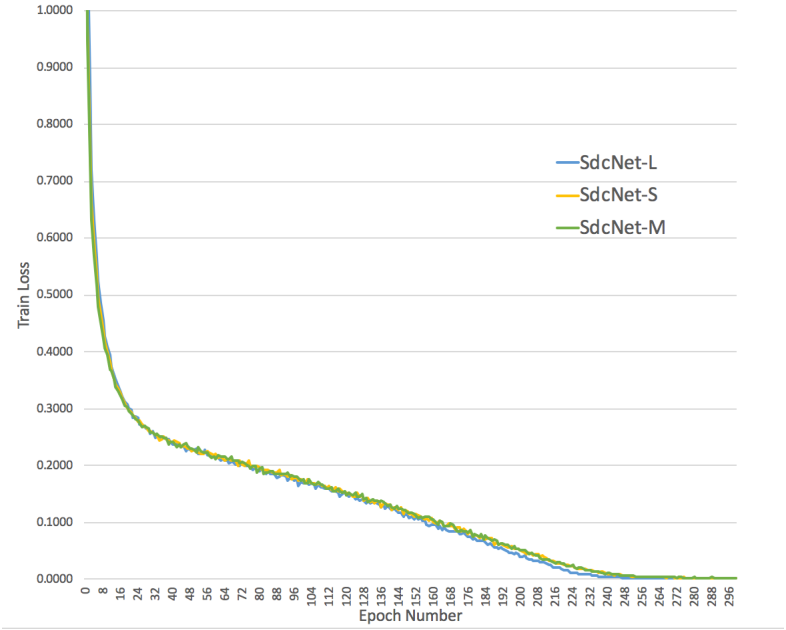


Figure 5.3: Loss curve for the training dataset of the three SdcNets

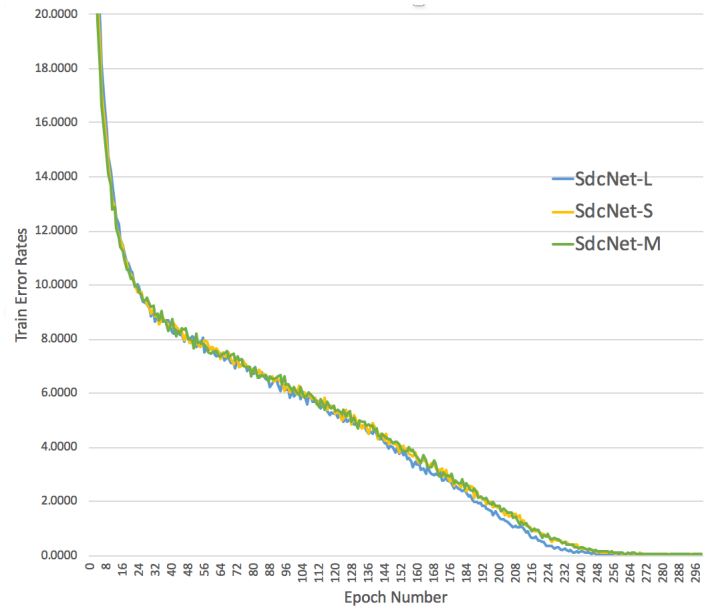


Figure 5.4: Error Rate Curve for the Training Dataset in the Training Epochs

loss reducing progressively. This curve is similar to the training loss curve, illustrating that the network is functional normally.

Based on these training details, it has been verified that SdcNet is effectively functional on the object classification tasks with a quick convergence and a small residual in the training process. The effectiveness of the training elements has also been confirmed. However, in order to evaluate the network performance, the test images are applied to the SdcNets in the following sub-chapter.

5.5 Testing Results of SdcNets

For the object classification tasks, the processing quality is measured by the classification error rate in the testing process. Fig. 5.5 shows the characteristics of the error rate versus the training epochs with the test samples from the CIFAR-10. The testing error rates are measured after a given number of training epochs are completed. The shape of this curve is similar to that of the training loss and the fluctuation is small after Epoch 255, which has confirmed the functions effectively of the network.

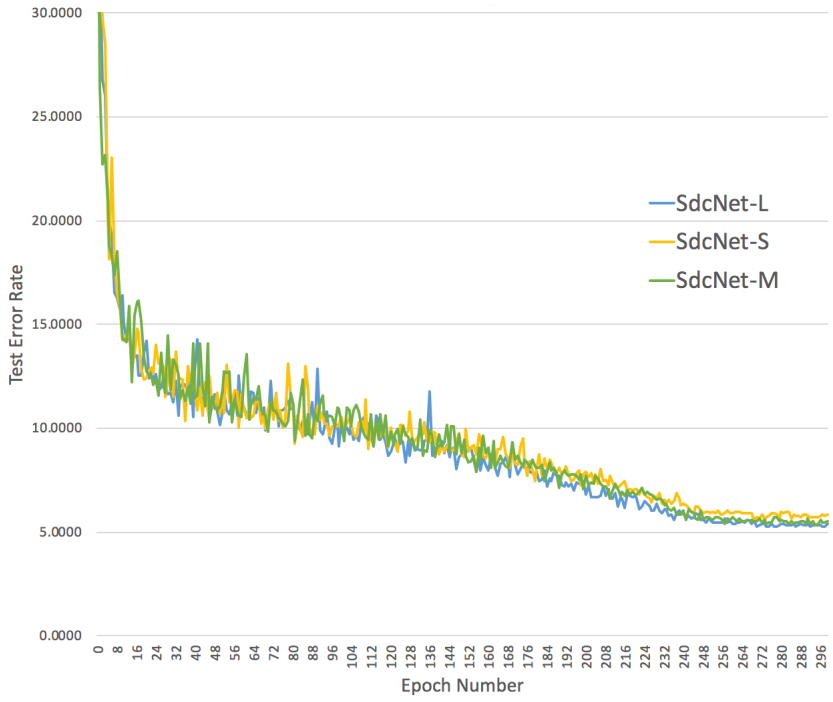


Figure 5.5: Error rate curve for the test dataset of the three SdcNets

Table 5.1: Comparison of classification error rate on CIFAR- 10 (C-10) and CIFAR-100 (C-100) with existing CNNs

Model	FLOPs	No. of Weights	ER (C10)*	ER (C100)**
VGG-16-pruned [29]	206M	5.40M	6.60%	25.28%
VGG-19-pruned [30]	195M	2.30M	6.20%	-
VGG-19-pruned [30]	250M	5.00M	-	26.20%
ResNet-56-pruned [31]	62M	-	8.20%	-
ResNet-56-pruned [29]	90M	0.73M	6.94%	-
ResNet-110-pruned [29]	213M	1.68M	6.45%	-
ResNet-164-B-pruned [30]	124M	1.21M	5.27%	25.28%
SdcNet-S	**55.12M**	**1.04M**	**5.60%**	**25.01%**
SdcNet-M	**73.41M**	**1.11M**	**5.41%**	**24.28%**
SdcNet-L	**103.3M**	**2.53M**	**5.24%**	**23.12%**

* The Error Rates(ER) for CIFAR-10 dataset.
* The Error Rates(ER) for CIFAR-100 dataset.

The performance matrix includes the processing quality error rates measured by the classification error rates (ER), and the computation complexity expressed by computation volume and the number of weights. In the CIFAR-10 test process, there are 10,000 samples in 10 classes while in 100 classes for CIFAR-100. In order to access the computation efficiency of SdcBlock and to evaluate the performance of SdcNet, these test results of the three SdcNets are compared with those reported recently using similar computation complexity, namely VGG-pruned and ResNet-pruned networks.

The test results are presented in Table 5.1. The error rates achieved by SdcNets are 5.6% or better in CIFAR-10 and 25.01% or better in CIFAR-100. To be more specific, the evaluation details are given below.

- SdcNet-S. The network requires the lowest computation volume, but has achieved an error rate of 5.60% in CIFAR-10 with 55 M Flops. This rate is better than those given by the VGG-pruned and ResNet-pruned networks listed for comparison, except the ResNet-164-B-pruned network, that yields 5.27% ,however, its computation volume is 2.24 times than that of SdcNet-S, in terms of Flops.

- SdcNet-L. It has achieved the best processing quality,$i.e.$ the error rate of SdcNet-L is 5.24% in CIFAR-10 using 103 M Flops with respect to 5.27% given by ResNet-164-B-pruned using 124 M Flops. The result has confirmed that the proposed network gives the best recognition

quality using a 20% less computation cost than the best processing quality network listed for comparaision.

- SdcNet-M. It achieves a low error rate of 5.41% using only 73.41 M Flops. It is a balance between the processing quality and the computation complexity.

Table 5.2: Error Rates of Per Class

Network	Bird	Automobile	Cat	Deer	Dog	Frog	Horse	Plane	Ship	Truck
SdcNet-S	6.8	2.3	13.6	5.2	8.8	3.2	3.7	4.9	3.5	3.9
SdcNet-M	6.3	2.1	13.0	5.1	8.6	3.4	3.7	5.1	3.3	3.5
SdcNet-L	6.2	2.1	11.2	5.1	8.2	4.1	3.8	5.2	3.1	3.4

The error rates of all the classes in CIFAR-10 of the three networks are given in Table 5.2. For the object classes with distinguished features such as "Automobile" and "Ship", a low error rate has been achieved. While for the objects having rich variations such as "Cat" and "Bird", the classification accuracy is less than that of the other classes. The results may be improved if more training samples of the objects having rich variations are provided in the training process.

In the test process, the performance of these SdcNets has been evaluated and the computation efficiency of SdcBlock has also been assessed. The test results provides a solid proof of that the proposed SdcBlock can extract features in high computation efficiency and SdcNet can achieve a good processing quality using a very modest computation volume.

5.6 Summary

In this chapter, the performance of SdcNet has been evaluated and compared with CNNs for the same tasks reported recently, which has also been used to assess the computation efficiency of SdcBlock. The network has been trained and tested using CIFAR dataset. In the training process, a batch size of 128 has been used to train the network for 300 epochs. An improved SGD optimizer with a cosine-shape descending learning rate has been applied to update the weights according to the loss calculated by the cross entropy loss function. For all the three SdcNets, the characteristics of the training loss versus training epochs illustrated that the architecture of SdcNet allows a short convergence time and a small residua error. In the test process, by applying 10,000 test samples in 10 classes from the CIFAR-10 dataset, SdcNet-S, the simplest SdcNet example, requires only 55

M Flops and its error rate has reached 5.60% in CIFAR-10. A better error rate of 5.24% has been achieved by SdcNet-L with a modest computation volume of 103 M Flops. SdcNet-M has been designed to achieve a balance of the processing quality and the computation volume, which gives a low error rate of 5.41% using only 73.41 M Flops. These results have confirmed that SdcBlock can efficiently extract extensive high-density feature information, and SdcNet, an end-to-end CNN architecture composed of SdcBlocks, yields a better performance, in terms of computation volume and processing quality, compared with CNNs for the same tasks reported recently.

Chapter 6

Conclusion

Object recognition is to identify objects in an image or a video. It is widely used in various applications such as autopilot and surveillance security systems. Extracting various features related to the objects from diverse backgrounds and classifying them is a challenging task, since there can be a huge amount of variations in the same object class while the same feature elements appear in the different classes. Hence, extracting sufficient high-density feature information for recognition is critical. Machine learning, in particular convolutional neural network(CNN), can be a good approach to solving this kind of problems. It uses a large number of samples to progressively determine the system parameters in order to detect various object features. Normally, CNN requires a large number of parameters, which, in consequence, leads a huge computation volume in order to achieve a good performance. Improving the computation efficiency of CNNs requires research effort.

The work presented in this thesis aims at developing a computation-efficient CNN architecture, $i.e.$ using the lowest computation volume to achieve a good processing quality, for object recognition. It is composed of two parts, a design of CNN module SdcBlock for feature extraction, and an end-to-end CNN architecture SdcNet for object recognition.

SdcBlock has been designed to be able to extract the maximum amount of high-density feature information from a given set of 2-D maps. To this end, filtering operations are applied successively to the same data to generate features of various orders, which can help to extract more varieties of image information from a given input channel. Based on this idea, in the design of SdcBlock, successive depthwise convolutions (Sdc) are applied to generate vectors containing various feature

information of high density. For each data channel, its first and second order filtering results are produced by the Sdc operations and then they are preserved by concatenation for further processing. Moreover, by using the depthwise convolutions, the computation volume can be significantly reduced, with respect to the standard convolutions. However, one may have the concern that the convolutions need to involve sufficient different kinds of data to secure the quality of the features extracted, especially for high-level features. In a CNN module as an implementation, data of multiple channels could be involved in each convolution, in particular in the mid-or-late stages. It should be mentioned that the concern has been addressed by a method of pre-and-post-convolutional data control method for the successive depthwise convolutions. Meanwhile, because the number of channels inputed to a module can be large, the data control in SdcBlock has been also used to optimize the data flow to improve the computation efficiency. The key issues of this data control method are given as follows.

Before the depthwise convolutions, an 1×1 group convolution with carefully selected hyper-parameters is applied to transform the input data to meet the requirements of the depthwise convolutions. In order to involve enough input channels required in one depthwise convolution, this group convolution indeed partitions the input channels, so that each depthwise convolution can be performed with information from multiple input channels of the SdcBlock. Moreover, this group convolution can not only help to scale the input data, but can also adjust the number of the data channels, and/or the map size by $Stride = 2$, which is required in some stages. The post-convolution data control is, in fact, a data arrangement applied to enhance useful features generated in the filtering operations of the two different orders. It is done by three steps, (i) rearranging the convolved data by a permutation, (ii) compressing the data by an 1×1 group convolution, and (iii) combining the compressed data with the input data by using addition or concatenation. By doing so, the information exchange among the feature channels produced by the depthwise convolutions is sufficient thus the extensive high-density feature information can be produced and enhanced for the succeeding computation stage. The pre-and-post-convolution data control is critical for the successive detphwise convolutions with a view to improving the computation efficiency and reducing the computation volume.

As mentioned previously, SdcBlock has been designed and proposed to be used in different stages of a CNN. To this end, this module is to permit the adjustment of hyper-parameters to handle

data of different natures. Besides the basic version of the SdcBlock, some variations of this module have also been given for different computation purposes. Among these hyper-parameters, serval of them should betaken into primary consideration. First of all, the group number g controls the input data channels, to prepare for the successive depthwise convolutions and also the convolved data, produced by the depthwise convolutions, to optimize the results. It should be determined according to the characters of the data channels in each layer and it is also related to the computation volume. Next, the number of the output channels should be sufficient to allocate enough varieties of feature elements while a larger of output channels means more computation volume. The parameter $Stride$ is also important as it is used to downsample the feature maps in order to condense the feature information and to reduce the computation volume. However, it should be used carefully to minimize the risk of information loss.

The end-to-end CNN architecture SdcNet, mainly composed of simply-cascaded SdcBlocks, has been proposed for object recognition tasks. Since SdcBlock is a modularized design, lots of different versions of SdcNets can be formed by easily adjusting the hyper-parameters to meet particular requirements, in terms of computation cost and/or processing quality. In this thesis, three detailed SdcNet configurations, namely SdcNet-S, SdcNet-M and SdcNet-L, have been given with different emphasis in performance. Among the three networks, SdcNet-S has the simplest structure and thus requires the smallest computation volume achieved by assigning an adequate number of output channels of each SdcBlock. Its group number is $g = 3$. SdcNet-L has been designed to achieve a better processing quality by increasing the number of output channels, meanwhile using a larger g to partially cancel the subsequent increased computation cost. SdcNet-M gives a balance between processing quality and computation volume. All of three SdcNets require a very modest computation resource, with respect to other CNNs reported recently, while they are expected to give a good processing quality.

The performance of the SdcNets mentioned above has been evaluated and compared with CNNs reported recently for the same tasks. CIFAR dataset has been used in the training and test process. The characteristics of the training loss versus the training epochs have been illustrated that the SdcNets yield a short convergence time and a small residual error. Ten thousand test samples in 10 classes from the CIFAR-10 dataset have been used to measure the classification error rates. The simplest SdcNet, SdcNet-S, which requires a very low computation cost of 55 M Flops, has given

an error rate of 5.60%. A better error rate of 5.24% has been achieved by SdcNet-L with a modest computation volume of 103 M Flops. SdcNet-M gives a low error rate of 5.41% using only 73.41 M Flops. These results have confirmed that SdcBlock can efficiently extract extensive high-density feature information, and SdcNet gives a better performance, in terms of computation volume and processing quality, compared with existing CNNs of similar kind.

The work presented in this thesis, the design of SdcBlock and the implementation of SdcNet, provides an effective solution to the problems of object recognition and helps to facilitate the design of CNNs. SdcBlock is a computation-efficient CNN module for feature extraction and it can be easily applied in varieties of CNNs for different image processing applications with a view to improving the efficiency of feature extraction with cooperation of other CNN layers. The end-to-end CNN architecture SdcNet can be applied in embedded systems such as mobile devices to recognize the objects using a limited computation resources.

References

[1] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proc. of the 40th International Conference on Software Engineering*, pp. 303–314, ACM, 2018.

[2] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *Proc. of British Machine Vision Conference*, vol. 1, p. 6, 2015.

[3] X. Cao, D. Wipf, F. Wen, G. Duan, and J. Sun, "A practical transfer learning algorithm for face verification," in *Proc. of the IEEE International Conference on Computer Vision*, pp. 3208–3215, 2013.

[4] O. Akgul, H. I. Penekli, and Y. Genc, "Applying deep learning in augmented reality tracking," in *Proc. of International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*, pp. 47–54, IEEE, 2016.

[5] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. of IEEE International Conference on Computer Vision*, vol. 2, pp. 1150–1157, Ieee, 1999.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[7] P. Viola and M. Jones, "Fast and robust classification using asymmetric adaboost and a detector cascade," in *Proc. of Advances in Neural Information Processing Systems*, pp. 1311–1318, 2002.

[8] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. of IEEE International Conference on Computer Vision*, vol. 2, pp. 1150–1157, 1999.

[9] O. Chapelle, P. Haffner, and V. N. Vapnik, "Support vector machines for histogram-based image classification," *IEEE transactions on Neural Networks*, vol. 10, no. 5, pp. 1055–1064, 1999.

[10] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," tech. rep., California Univ San Diego La Jolla Dept Of Computer Science and Engineering, 2002.

[11] P. F. Felzenszwalb and D. P. Huttenlocher, "Pictorial structures for object recognition," *International Journal of Computer Vision*, vol. 61, no. 1, pp. 55–79, 2005.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. of Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

[13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

[14] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[16] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, *et al.*, "Large scale distributed deep networks," in *Proc. of Advances in Neural Information Processing Systems*, pp. 1223–1231, 2012.

[17] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, p. 115, 2017.

[18] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 993–1001, 1990.

[19] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[20] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proc. of International Conference on Machine Learning*, ICML'10, (USA), pp. 807–814, Omnipress, 2010.

[21] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. of International Conference on Machine Learning*, vol. 30, p. 3, 2013.

[22] M. N. Gibbs and D. J. MacKay, "Variational gaussian process classifiers," *IEEE Transactions on Neural Networks*, vol. 11, no. 6, pp. 1458–1464, 2000.

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, p. 533, 1986.

[24] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[25] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145–151, 1999.

[26] Y. Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2)," in *Doklady AN USSR*, vol. 269, pp. 543–547, 1983.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. of Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

[28] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision – ECCV 2014*, pp. 818–833, Springer International Publishing, 2014.

[29] H. Li, A. Kadav, and I. Durdanovic, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[30] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proc. of IEEE International Conference on Computer Vision*, pp. 2755–2763, 2017.

[31] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. of IEEE International Conference on Computer Vision*, vol. 2, p. 6, 2017.

[32] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *arXiv preprint arXiv:1610.02357*, 2016.

[33] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proc. of 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, IEEE, 2018.

[34] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," *arXiv preprint arXiv:1707.01083*, 2017.

[35] Y. Ma and C. Wang, "Sdcnet: A computation-efficient cnn for object recognition," in *Proc. of 2018 IEEE 23rd International Conference on Digital Signal Processing (DSP)*, pp. 1–5, Nov 2018.

[36] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[37] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, p. 3, 2017.

[38] N. S. Keskar and R. Socher, "Improving generalization performance by switching from adam to sgd," *arXiv preprint arXiv:1712.07628*, 2017.

[39] I. Loshchilov and F. Hutter, "Sgdr: stochastic gradient descent with restarts," *Learning*, vol. 10, p. 3, 2016.

[40] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," in *Proc. of Artificial Intelligence and Statistics*, pp. 562–570, 2015.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proc. of IEEE International Conference on Computer Vision*, pp. 1026–1034, 2015.