Learning Self-Choice using Deep Neural Networks with Combination of Tied and Untied Convolution

Master's Thesis im Fach Informatik

vorgelegt
von

Jinqing You

Geboren am 30.04.1994 in Jiangsu

Angefertigt am

Lehrstuhl für Mustererkennung (Informatik 5)
Department Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg.

Betreuer: M.Sc. Mathias Seuret, Dr.-Ing. V. Christlein, Prof. Dr.-Ing. habil. Andreas Maier

Beginn der Arbeit: August 29th, 2019

Abgabe der Arbeit: März 2nd, 2020

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Richtlinien des Lehrstuhls für Studien- und Diplomarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

Erlangen, den 2. März 2020

Übersicht

In praktischen Anwendungen der Computer Vision spielen Convolutional Neural Networks immer eine wichtige Rolle. Da diese 'Tied Convolution' in neuronalen Netzen weit verbreitet sind und die Ergebnisse recht gut sind, schenken die Menschen der 'Untied Convolution' normalerweise nicht viel Aufmerksamkeit. Diese Masterarbeit kombiniert Tied Convolution mit Untied Convolution durch eine Struktur namens ParallelBlock und beschreibt die Implementierung im Detail. Mithilfe von ParallelBlock können wir nicht nur neuronale Netze aufbauen und trainieren, sondern auch das Verhalten von 'Self-Choice' anhand verschiedener Datensätze untersuchen. Durch das Training der neuronalen Netze auf MNIST, CIFAR-10 und DIVA-HisDB erhalten wir eine Reihe von Ergebnissen. Diese Ergebnisse zeigen, dass Netzwerke mit ParallelBlock gute Erkennungsgenauigkeiten erreichen konnen. Dabei haben verschiedene Datensatze unterschiedliche Tendenzen zur Tied oder Untied Convolution. Am Ende wird ParallelBlock auch in ResNet eingebettet, um die Möglichkeit und Praktikabilität dieser Kombination weiter zu analysieren.

Abstract

In practical applications of computer vision, convolutional neural networks always play an important role. Because this 'Tied Convolution' is widely used in neural networks and the results are quite good, people typically do not pay much attention to 'Untied Convolution'. This thesis combines tied convolution with untied convolution together by a structure called ParallelBlock and describes the implementation in detail. With help of ParallelBlock, we can not only construct and train neural networks, but also research the 'Self-Choice' behaviors by different datasets. By training the neural networks on MNIST, CIFAR-10 and DIVA-HisDB, we get a series of results. These results show the networks with ParallelBlock can reach good recognition accuracies. Meanwhile, Different datasets do have different tendencies to tied or untied convolution. At the end, ParallelBlock is also embedded in ResNet to further analyze the possibility and practicality of this combination.

Contents

Chapter 1

Introduction

As a representative neural network [Has95] family for deep learning [LeC15a], convolutional neural networks [Yan15], which are also called shift-invariant [Roc97] artificial neural networks, play an important role in the field of computer vision [For02]. Convolutional neural networks are a collective term for a series of neural networks. They are all feedforward [Fin06] neural networks that contain convolution operations [Hir12]. They are designed by imitating biological visual perception and can be used for supervised and unsupervised learning [Zha07]. Convolutional neural networks are generally composed of different neural network layers. The most important parts are convolutional layers. These layers perform shift-invariant classification through convolution operation. A convolution operation is a type of method that generates the mathematical calculation result by two functions f and g, as it is described in Equation (1.1). For a tied convolution, which is written as $f * g$, it is defined as the integral of the product of the f and g, after g is reversed and shifted.

$$(f * g)(t) = \int_0^t f(\tau)g(\tau - t)dt \tag{1.1}$$

See Figure 1.1, taking a two-dimensional tensor [Hac12] as an example, the convolution operation is as shown in the figure, each pixel of input image covered by the kernel [ST04] is multiplied with each corresponding kernel value and the products are summated [Mac17]. The kernel is moved over the image performing the convolution as a sliding window.

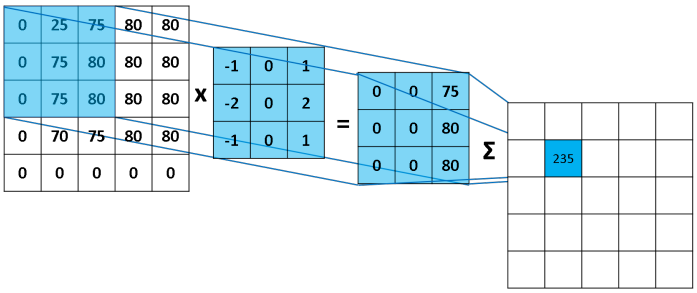


Figure 1.1: One step in the two-dimensional convolution process [Mac17].

By setting the weights of the convolution layer adequately [Thi97], the convolutional layer can perform feature extraction operations on the input data. The size of the weights determines the size of the field where the input data is convolved. The sliding window of weights moves field by field in order to perform matrix multiplication [Cop87] and sum operations on the elements contained in each field. Take 2D convolution as an example, according to Equation (1.2) and Equation (1.3), Z_l and Z_{l+1} represent the input and output of the current convolutional layer respectively. In these formulas, w is the weights and k is the number of kernels, which has an influence on the channels of output data, i.e. with k as kernel number, the output channels become k after this convolution layer. The variable b represents bias [Gem92], which is used to adjust the results during the training process.

$$Z_{l+1}(i,j) = [Z^l \otimes w^{l+1}](i,j) + b = \sum_{k=1}^{K_l} \sum_{x=1}^{f} \sum_{y=1}^{f} [Z_k^l(si+x, sj+y) w_k^{l+1}(x,y)] + b \tag{1.2}$$

$$L_{i+1} = \frac{L_i + 2p - f}{s} + 1 \tag{1.3}$$

According to Equation (1.3), we can get the output size of this convolutional layer, where L_i is defined as the size of the input data, L_{i+1} is defined as the size of the output data, f is the kernel size, p is the number of padding. s is number of stride, which represents the spacing between convolved fields. For the neural network structure with multiple convolutional layers [Kri12], the first convolutional layer typically only extract some low-level features such as edges, lines and corners. Then the following layers iteratively extract more complex features from lower-level features [Zho00]. So far, different variants of convolutional neural networks have achieved significant results, e.g. GoogLeNet [Sze15] won the ILSVRC(ImageNet Large Scale Visual Recognition Challenge) 2014 classification task, but VGGNet [Sim14] beated the GoogLeNet

and won the localization task. In 2015, ResNet [He16] took first place of classification task in ILSVRC.

However, how to further improve the performance of neural networks has become an interesting and challenging topic that many professionals and scientists are willing to make unremitting efforts. It is worth mentioned that many effective methods have already been invented to optimize the neural network, e.g. changing the neural network structure to extract and integrate patterns or using special designed loss functions [Ros04] to balance the impact between different classes. These methods can help to accelerate the process of recognition or improving the accuracy of result. According to the same considerations, the main idea of this report is based on the hypothesis of the combination of tied convolution and untied convolution [Fuk75] with verifying the authenticity of the hypothesis through neural network applications, in order to further study the influence of this kind of new structure on the deep learning recognition process.

Chapter 2

Untied Convolution

2.1 Introduction of Untied Convolution

For many people, 'Untied Convolution' is a relatively strange name, but this is not a mathematical concept that appeared recently. On the contrary, it appeared very early, even before the applications of neural networks became widespread. In 1975, Kunihiko Fukushima first proposed the concepts of untied convolution and pooling in his article 'Cognitron: A self-organizing multilayered neural network' [Fuk75], although these concepts were not called their present names at that time. Later in 1980, Kunihiko Fukushima proposed tied convolution in the network 'NeoCognitron' [Fuk80], which provided inspiration for the development of deep learning. In 1989, Yann LeCun proposed the first modern CNN: LeNet [LeC15b]. Since then, the neural networks based on tied convolution are the main research direction and concept of untied convolution faded out of people's vision gradually. But in recent years, people started paying attention again to this concept and understanding the potential value of this method.

2.2 Localization

For the tied convolutional layer, all convolved fields share the same kernel. See Figure 1.1, the tied convolutional layer ties the weights for the entire layer, i.e. in training period, one and only one kernel is multiplied and summated in different areas of input images, thus this kernel is a kind of combination among different areas and the kernel value after training represents entire image. However, this means the neural network can only extract features through one per filter transformation of the input, which provides the translational invariance. Sometimes this is not the optimal behavior, e.g. when it is more efficient to obtain the information provided by the

specific position of the input image. In this case, we can try to use untied convolution instead of tied convolution. The difference between untied convolution and tied convolution is that each predefined location of an input image in untied convolution shares the weights, but separate locations not. As it is showed in Equation (2.1), this local operation of untied convolution is able to extract different characteristics from different locations of input data, this tends to have more varying transformation operations and handle more variations of information than global operation. By different untied weights in different locations, the untied convolutional layer is capable of learning various linear function for different local maps, but it does not learn location-invariant features [Gao16], so it may lose some significant information that indicates the combinations of different areas.

$$Z_{l+1}(i,j) = [Z^l \otimes w^{l+1}](i,j) + b = \sum_{k=1}^{K_l}\sum_{x=1}^{f}\sum_{y=1}^{f}[Z_k^l(si+x, sj+y)w_{k,i,j}^{l+1}(x,y)] + b \qquad (2.1)$$

2.3 Amount of Parameters

Therefore, according to the localization of untied convolution, we can further calculate the number of parameters [Zou09]. Regardless of the number of channels, assuming that the width and height of input data are both *L*, size of kernel is *(*f), *p* is the number of padding and *s* is number of stride. According to Equation (1.3), we can calculate the number of parameters used by such an untied convolutional layer by Equation (2.2). In other words, the number of parameters used by the untied convolutional layer is *N* times more than a tied convolutional layer.

$$N = (\frac{L_i + 2p - f}{s} + 1) * (\frac{L_i + 2p - f}{s} + 1) \qquad (2.2)$$

2.4 Parallelization

In order to further accelerate the computing speed [Fan04], it is necessary to use GPU(Graphic Processing Unit). It is a microprocessor [Cha00] that specializes in image and graphics related computing work on personal computers, workstations, game consoles and mobile devices. However, GPUs can also be used for parallel computation [Akl97]. Compared with the CPU(Central Processing Unit), it has the advantage of focusing on computing-intensive tasks and its computation speed is much faster. After finishing an operation, the GPU does not send data back to memory, but performs the next operation based on previous data. Since a large amount of data are

prevented from being transmitted back and forth, the calculation speed of GPU is significantly accelerated. In current generations, the low end GPUs have about 400 cores and the high end GPUs have nearly 4,000 cores. For deep learning, a dataset is often divided into different batches, so with help of GPU can also make people fully utilize its feature of parallelization to process multiple data.

Chapter 3

ParallelBlock and its Implementation

3.1 Environment

After comparing the current popular deep learning frameworks, it is decided to base the project on PyTorch. The reasons are as follows: Firstly, automatic calculation of gradients [Rud16] is an advantage that cannot be ignored. Secondly, it is simple and easy to understand PyTorch code, people can implement with it very qucikly and flexibly. In addition, for the implementation of untied convolution layer, PyTorch libraries integrate key functions fold and unfold, namely *torch.nn.Unfold* and *torch.nn.Fold*, which provide a shortcut for the implementation.

3.2 Fold and Unfold

Because PyTorch does not provide an untied convolutional layer which is available for calling, it is necessary to implement it by inheriting from *torch.nn.Module*. Fortunately, PyTorch has encapsulated two classes named *torch.nn.Unfold* and *torch.nn.Fold*. PyTorch function *torch.nn.Unfold* can be used to extract the local block from the input data to achieve local connection sliding window. As an inverse operation of *torch.nn.Unfold*, *torch.nn.Fold* combines an array of sliding local blocks into a large containing tensor. The parameters of *torch.nn.Unfold* are *(kernelSize, dilation=1, padding=0, stride=1)* and the parameters of *torch.nn.Fold* are *(outputSize, kernelSize, dilation=1, padding=0, stride=1)*. By calling *torch.nn.Unfold*, then performing a matrix production between the outputs of *torch.nn.Unfold* and initialized weights, then let the results of matrix production [Gol88] pass *torch.nn.Fold*, we can the achieve same results of a tied convolution. For example, if we want to simulate *torch.nn.conv2d (inChannels=2, outChannels=3, kernelSize=(4,5))*, we set an input data with size *(channels=2, heights=10, width=12)*, the size

of *weights* should be *(2, 3, 4, 5)*. First we make the input data pass the unfold function whose *kernelSize* is *(4, 5)*. This result is recorded as *inpUnf* with size *(60, 56)*. After that, *inpUnf* and the reshaped *weights* with size *(2, 60)* are multiplied. New result is recorded as *outUnf* with size *(2, 56)*. Then we let *outUnf* go through a fold function with *outputSize* by *(7, 8)* and *kernelSize* by *(1, 1)*. The final output is recorded as *out* with size *(2, 7, 8)*. At the same time, we also record the output of *torch.nn.conv2d* as *out2d*. In order to prove that both processes are the same. We can initialize a random tensor with previous input size and get the corresponding *out* and *out2d*. After performing element-wise subtraction [Gol88] between *out* and *out2d*, we can find that maximum arithmetic difference does not exceed 1.9073e-06, which is most likely due to rounding errors [Wil94]. Therefore, the two kinds of processes can be regarded as equivalent.

3.3 Implementation of Untied Convolutional Layer

3.3.1 Avoid Loop

According to Equation (2.2), the number of untied convolution weights is N times bigger than tied convolution. In order to implement the untied convolutional layer efficiently, it is necessary to design the class structure before implementation in order to reduce the complexity [Pap98] of untied convolution as much as possible. One of the most effective ways is to avoid or reduce the usage of loops. The usage of loops may bring a series of problems. First and foremost, the complexity of the algorithm increases significantly, if we use too many loops in Python, the efficiency of our program would become low. Now we can use GPUs to accelerate the training speed, the loop unrolling feature of GPU is very important to parallelization. Each loop task can be executed separately without iteration. If loops are still used at will, the computation time might be out of control and the resources of GPUs are wasted. Besides, using loops may bring other problems such as reducing the readability [Bus09] of the code. Nested loop structure can also increase the difficulty of implementation and the line of codes. Fortunately, Python provides many library functions and matrix operations tools. By using these tools, we can avoid most loops [Lee90].

3.3.2 Matrix Trick Utilization

Instead of using loops, we choose to use some matrix operations [Lee90] to implement untied convolutional layer efficiently. We assume the input and output channels are recorded as *inChannels* and *outChannels*, the width and height of the kernel used at each independent area of input

tensor are recorded as *kernelWidth* and *kernelHeight* and the total number of kernels is counted as *kernelNum*. After the input tensor is submitted to function *torch.nn.Unfold*, we record a matrix *R*. Matrix *R* has this relationship with output tensor of *torch.nn.Unfold*: we first transpose the output, then replicate it by final dimension to the amount of *kernelNum*. Meanwhile, we initialize a zero matrix with the size (*kernelNum*, *inChannels* * *kernelWidth* * *kernelHeight* * *kernelNum*), then we set its corresponding diagonal element to 1, i.e. if *len* is equal to *inChannels* * *kernelWidth* * *kernelHeight*, for the i-th row, values from column (i * *len*) to column ((i +1) * *len*) need to be set to 1, we record this matrix as *O*. After this, we can describe the matrix trick in Equation (3.1) and Equation (3.2), where *M* is the hadamard product [Hor90] of matrix *O* and the output matrix *R* of function *torch.nn.Unfold*. *Z* is the matrix product of *M* and *W*.

$$M = O * R \tag{3.1}$$

$$Z = MW \tag{3.2}$$

3.3.3 Untied Convolutional Class

Untied convolutional class inherits from *torch.nn.Module*, so we only need to overwrite [Lip05] its constructor and forward function. For the constructor, as stated above, parameters are as follows: *(inChannels, outChannels, kernelWidth, kernelHeight, kernelNum, stride=1, padding=0)*. Meanwhile, we also need to initialize weights matrix *W*, matrix *O* and two variables *outputHeight* and *outputWidth*. These two variables represent the height and width of output tensor and can be calculated according to the Equation (1.3). These two variables are used as parameters of *torch.nn.Fold*. In the forward function, the input tensor passes through the unfold operation with parameters *((kernelWidth, kernelheight), 1, padding, stride)*, its output is recorded as *inpUnfold*. Using the matrix trick presented in 3.3.2, as *inpUnfold* plays the role of matrix *R*, we can get final output *outUnfold* as matrix *Z*. Then we let *outUnfold* pass the fold operation with parameters *((outputHeight, outputWidth), (1, 1))*. After adding bias on the output of fold operation, the process of forward propagation ends.

3.3.4 Time Memory Trade-Off

Although we reduce computing time through matrix tricks, we spend more memory. It is necessary to consider the balance between time and memory [Bir05]. In the field of computer science, this is a very classic problem. Engineers often need to weigh the computing time and the computing

resources, specially training a neural network is very time-consuming. In the case, using the matrix tricks allocates more memory, which means that variables like matrix *O* and matrix *R* occupy more resources. However, matrix *O* only needs to be initialized once in the constructor and the built-in function *repeat()* can be used to construct matrix *R*. These can help to implement more efficient classes.

3.4 ParallelBlock Class

3.4.1 Self-Choice with ParallelBlock

Given a unfamiliar dataset, it is hard to choose tied or untied convolutions to construct a neural network. Especially the training time of a neural network is relatively long. Thus we design a block structure which is called ParallelBlock. It takes inspiration from Highway Networks [Sri15]. ParallelBlock is constructed according to Equation (3.3), input data can pass through the untied and tied convolutional layers parallelly. At the same time, the data also need to pass through a function *w(x)*, the value range of *w(x)* is controlled between the interval]0, 1[. There are two reasons for using *w(x)*. Firstly, *w(x)* is used to further extract the data features as an input member of the next layer. Secondly, the untied and tied convolutional layer are effectively combined through *w(x)* and *1-w(x)*, which constructs a parallel structure. The values of *w(x)* can represent the data selection results, which are defined as Self-Choice of data. And the mean value of *w(x)* is defined as Self-Choice Factor. If Self-Choice Factor is greater than 0.5, it means that the data tend to untied convolution, otherwise the data tend to tied convolution.

$$y(x) = w(x) * uconv(x) + (1 - w(x)) * conv(x) \tag{3.3}$$

3.4.2 Implementation of ParallelBlock Class

The implementation of ParallelBlock is just like previous description. The input tensor pass a tied convolution layer and an untied convolution layer at the same time. Outputs are respectively recorded as *outConv* and *outUconv*. Meanwhile, the input tensor also needs to go through another tied convolutional layer, whose *outChannels* is set to be 1 and the remaining parameters are unchanged. The output of this layer is recorded as *outWeight*. In order to control Self-Choice results in interval]0, 1[, *outWeight* needs to go through a Sigmoid function [Han95] and its output are record as *uconvWeight*. After preparing these outputs, with the help of Python Broadcasting, the ParallelBlock can be constructed as Equation (3.3), where *uconWeight*, *outConv* and *outUconv*

represent the outputs of functions *w(x)*, *conv(x)* and *uconv(x)* respectively.

Chapter 4

Evaluation on Common Datasets

4.1 Experiments Goal

After completing the implementation of untied convolutional layer and ParallelBlock, the next step is to construct neural networks and evaluate them on some common datasets. This chapter describes training process and results of neural networks constructed by ParallelBlock on MNIST and CIFAR-10. This chapter has two purposes: First, to observe the performance of ParallelBlock through training to determine whether this structure can effectively extract features of data. Second, to observe the Self-Choice results of the data through the trained neural network parameters, which is also the original intention of designing ParallelBlock.

4.2 Training on MNIST

4.2.1 Description of Dataset

In the field of machine learning, MNIST [LeC13] is a classic dataset that is commonly used. This dataset comes from the National Institute of Standards and Technology (NIST). It contains 28x28 pixels images of handwritten numbers of the images which are from 0 to 9. It contains 60,000 training images and 10,000 test images. Figure 4.1 shows some samples of MNIST dataset.

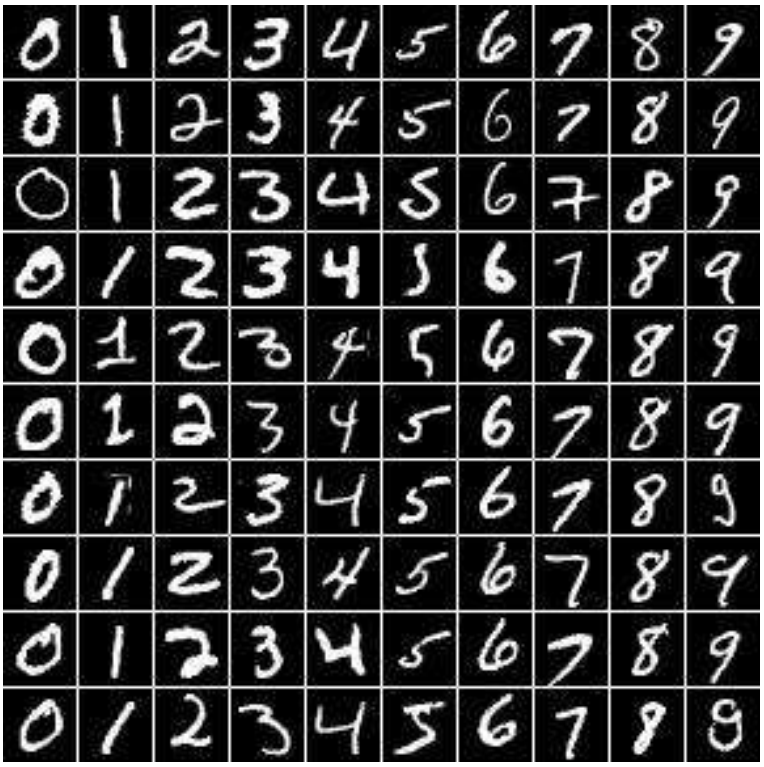


Figure 4.1: Samples of MNIST [Son17].

This experiment is the first attempt with the ParallelBlock. As a single-channel dataset, MNIST is suitable for testing the performance of ParallelBlock. Compared with other datasets, data composition of MNIST is relatively simple. By MNIST, we can easily judge whether the neural network can effectively learn the data features or not. This is very important for the practical judgment of ParallelBlock. If the performance on MNIST is poor, it might indicate that ParallelBlock has no research value or some ignored problems in implementation. All data from training file of MNIST are used. 2000 images randomly picked from the MNIST test file as validation set in this experiment. During the training process, after each 50 batches, the accuracy of the current model is judged by this validation set. The rest samples of the test file is used as test set.

4.2.2 Description of architecture

In order to research the performance of the ParallelBlock and compare it with common convolutional layer, two neural networks are designed. The structure of neural network with ParallelBlock is shown in Table 4.1. The neural network without ParallelBlock is constructed based on the first

network but with only a little difference, specifically replacing the ParallelBlock with a convolutional layer with same parameters. These two networks are trained with 0.001 as the learning rate [Smi17] of a stochastic gradient descent(SGD) [Bot18] optimizer, 16 as batch size [Kes16].

Layers	InChannels	OutChannels	KernelSize	Stride	Padding
Convolution1	1	16	5	1	2
ReLU	/	/	/	/	/
MaxPooling	/	/	2	1	0
ParallelBlock1	16	32	5	1	0
ReLU	/	/	/	/	/
MaxPooling	/	/	2	1	0
FullyConnected1	800	10	/	/	/

Table 4.1: Network Architecture with ParallelBlock.

4.2.3 Training result

Two neural networks are stopped after completing one epoch. The test set accuracy of network with ParallelBlock is 0.9811 and the network without ParallelBlock is 0.9826. The accuracies of both networks on validation set after each 50 batches are recorded and shown in Figure 4.2. As we can see from the figure, the two neural networks can achieve accuracies up to 98 percent, which can be inferred that the ParallelBlock has the value of continuing research.

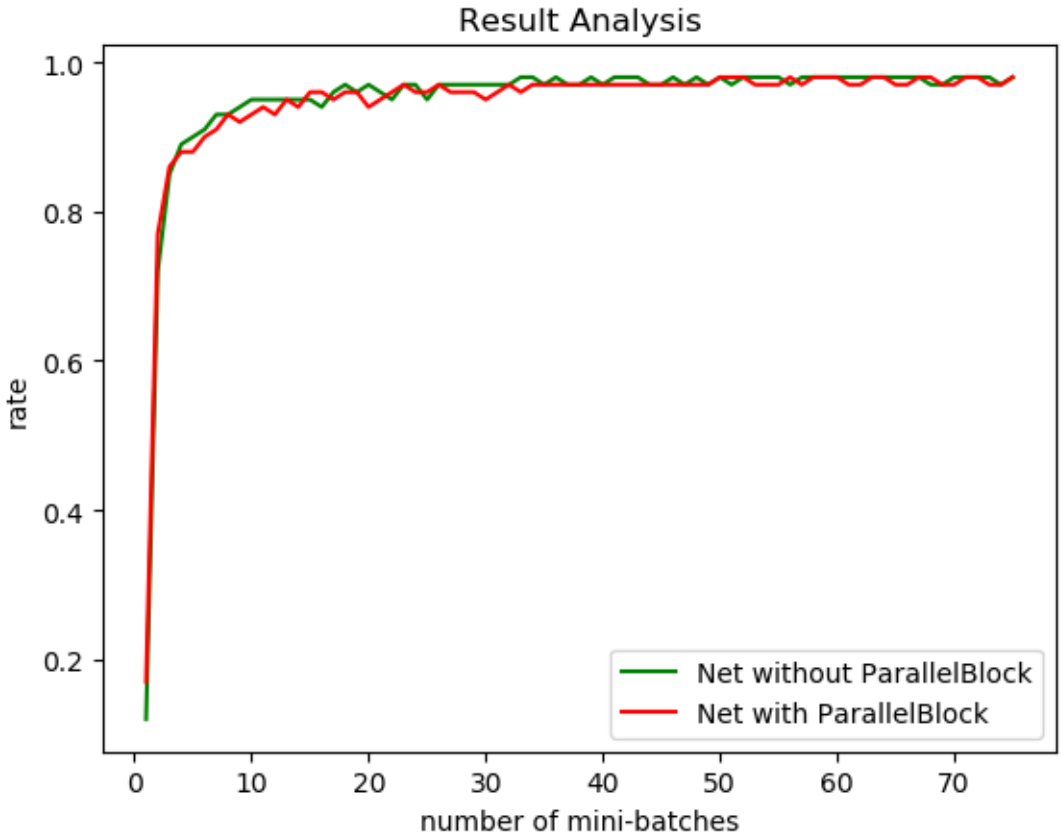


Figure 4.2: Training accuracies of two networks each 50 batches. The red line represents the accuracy of network with ParallelBlock, while the green line represents the accuracy of network without ParallelBlock.

4.2.4 Self-Choice on MNIST

Self-Choice of ParallelBlock is also need to be discovered in this experiment. After the training phase is completed, a new experiment is designed, in order to research the effect of the ParallelBlock and observe the Self-Choice result of MNIST. In the process of forward propagation, we can measure whether the tied convolution or the untied convolution is selected by comparing Self-Choice Factor in ParallelBock. The values of Self-Choice Factors in different intervals are counted, which are shown in Figure 4.3. According to the Figure 4.3, in this experiment, the Self-Choice Factors of almost all samples are concentrated in the interval]0.8, 1.0[, which shows that for MNIST dataset, the neural network prefer to use untied convolution at the end of its operational parts.

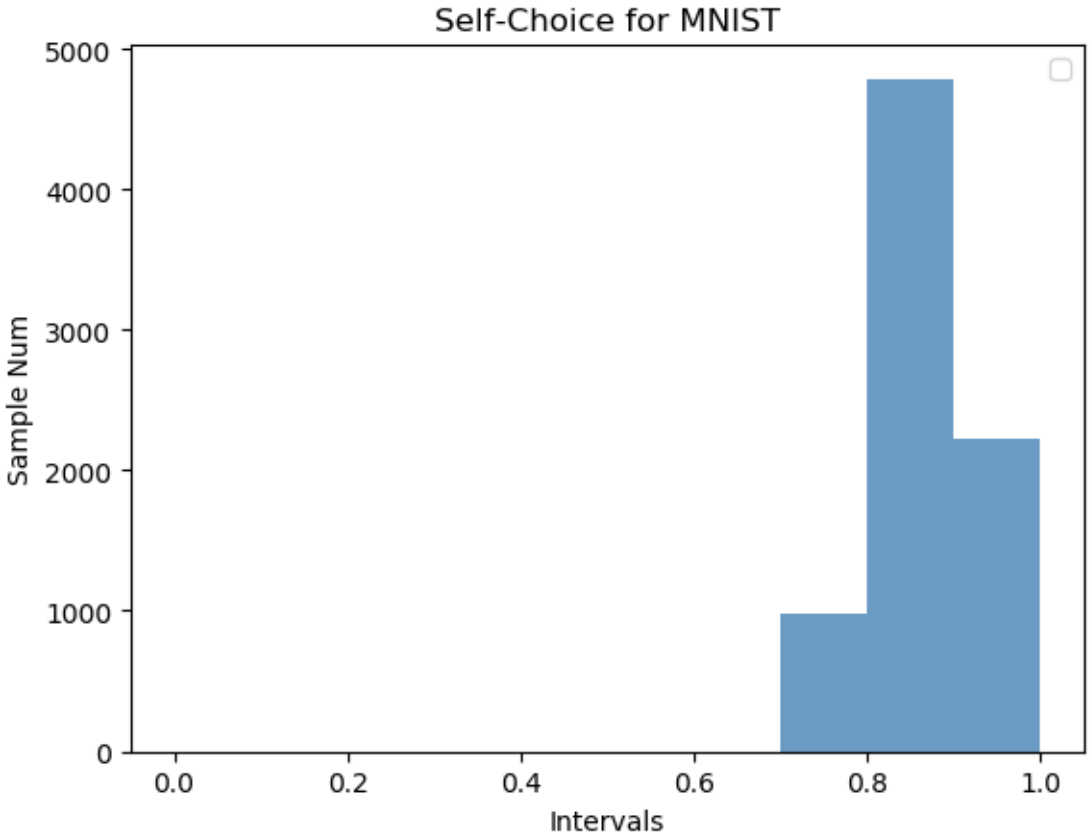
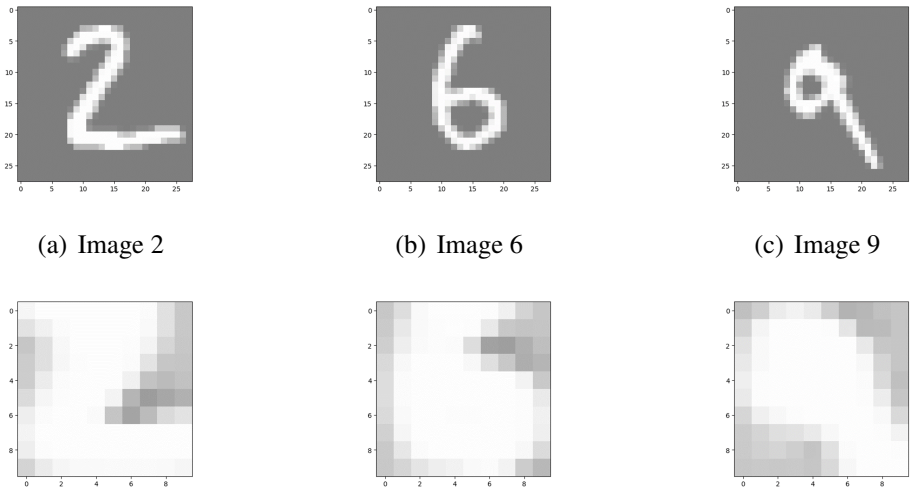


Figure 4.3: Self-Choice Factors of untied convolution on test set by ParallelBlock.

4.2.5 Heatmaps of Self-Choice Weights on MNIST

Parameters of tied convolution are affected by global information, but untied convolution is focused on locality. Figure 4.3 only shows the average values. In order to further explore what real happens during Self-Choice, it is necessary to visualize the whole parts of selection maps. In Figure 4.4, untied convolution selection maps in ParallelBlock from some correctly identified samples are shown. You can see more samples in Appendix A. The brightness is proportional to the value of selection maps. These highlighted parts can largely reflect the characteristics of the samples. In other words, the relevant weights of untied convolution in ParallelBlock do make a choice that reflects the characteristics of samples and the weights are formed during the training process, which represents the importance of united convolution during this training.

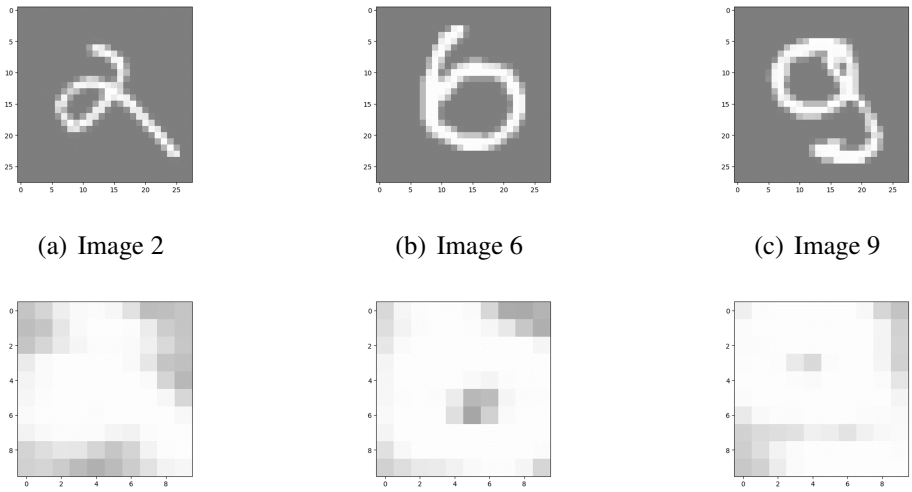


(a) Image 2 (b) Image 6 (c) Image 9

(d) Heatmap 2 (e) Heatmap 6 (f) Heatmap 9

Figure 4.4: Heatmaps of correctly identified samples 2,6,9.

Visualization for the heatmaps of some incorrectly identified samples have also been produced. According to Figure 4.5, although the recognition resluts are incorrect, heatmaps of these samples still show large areas of highlights, which proves the data in MNIST prefer untied convolution.



(a) Image 2 (b) Image 6 (c) Image 9

(d) Heatmap 2 (e) Heatmap 6 (f) Heatmap 9

Figure 4.5: Heatmaps of incorrectly identified samples 2,6,9.

4.3 Training a Block Neural Network on CIFAR-10 Dataset

4.3.1 Description of Dataset

After the experiments on MNIST, we can confirm that ParallelBlock contains potential experimental value. The previous experiments show that untied convolutions are prefered by the CNN for MNIST. The question now is whether it is always the case or just for some data. In order to further study the ParallelBlock, CIFAR-10 [Kri09] is selected as the next experimental dataset. CIFAR-10 comes from with a dataset of 80 million small images collected by Alex Krizhevsky, Vinod Nair, and Geoffrey samples Hinton [Kri09]. The CIFAR-10 dataset consists of 60,000 32x32 three channels colorful images of 10 classes, which are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. Figure 4.6 shows some samples from each class.

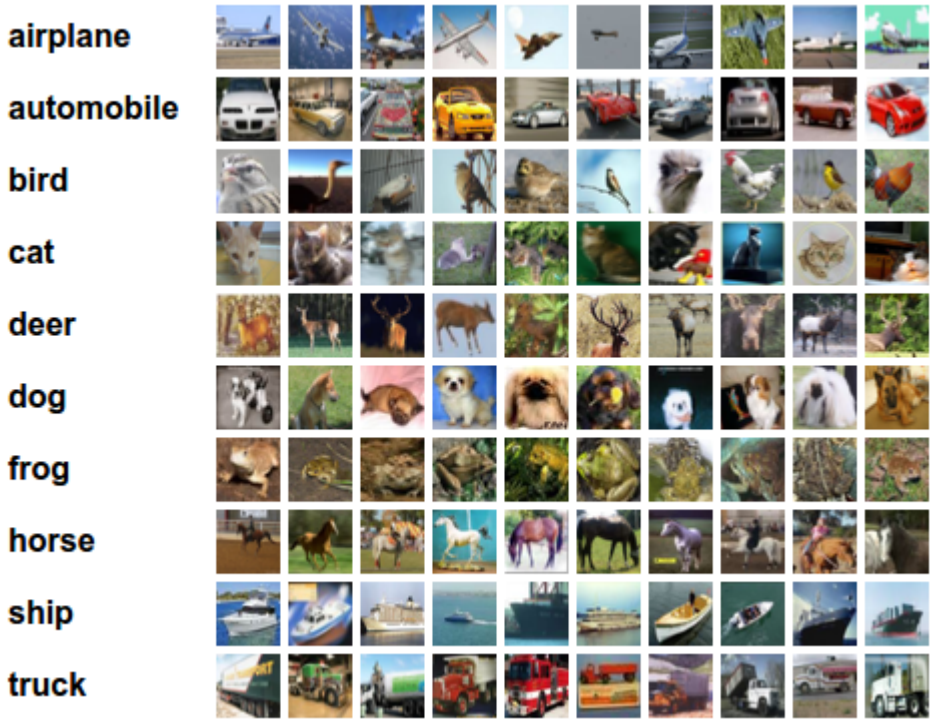


Figure 4.6: Samples of CIFAR-10 [Ape18].

4.3.2 Description of Architectures

According to the recommendation of the PyTorch Tutorial [PyT17], a new architecture is constructed. At the same time, the block version is constructed by copying PyTorch Tutorial network with replacing the second convolutional layer by a ParallelBlock with same parameters. It is

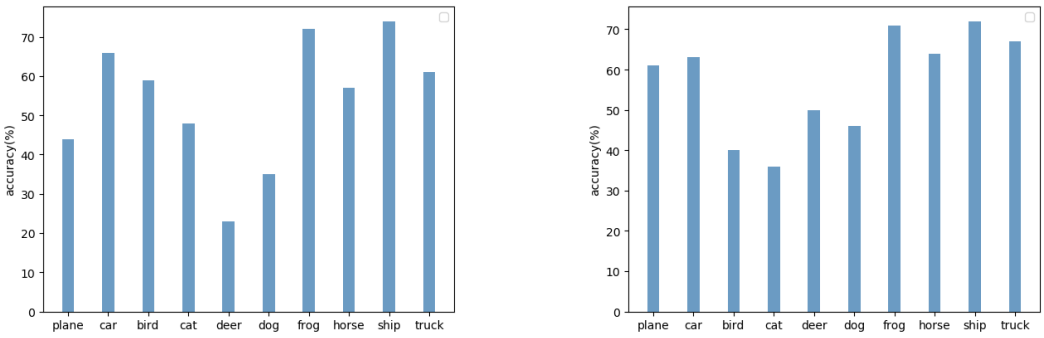
described in Table 4.2. Both neural networks are trained by 2 epochs according PyTorch Tutorial with the cross-entropy [DB05] as the loss function, SGD with learning rate of 0.001 as the optimizer and 4 as batch size.

Layers	InChannels	OutChannels	KernelSize	Stride	Padding
Convolution1	3	6	5	1	0
ReLU	/	/	/	/	/
MaxPooling	/	/	2	1	0
ParallelBlock2	6	16	5	1	0
ReLU	/	/	/	/	/
MaxPooling	/	/	2	1	0
FullyConnected1	400	120	/	/	/
ReLU	/	/	/	/	/
FullyConnected2	120	84	/	/	/
ReLU	/	/	/	/	/
FullyConnected3	84	10	/	/	/

Table 4.2: PyTorch Tutorial network with ParallelBlock [PyT17].

4.3.3 Training result

Neural network without ParallelBlock has the overall recognition accuracy of 54 percent. For the neural network with a ParallelBlock embedded, the overall recognition accuracy is 57 percent.The results of different classes are shown in Figure 4.7.



(a) Network without ParallelBlock

(b) Network with ParallelBlock

Figure 4.7: Training accuracies of CIFAR-10 test set by PyTorch tutorial networks with and wihout ParallelBlock.

4.3.4 Self-Choice on CIFAR-10

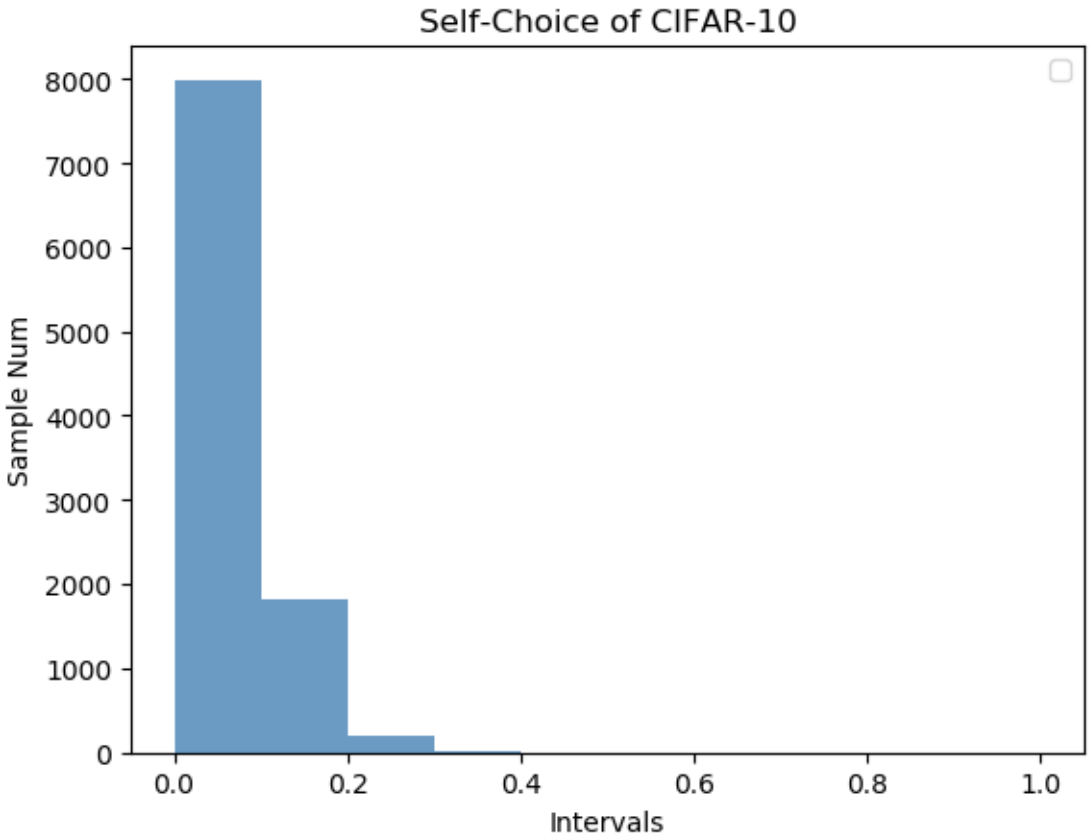
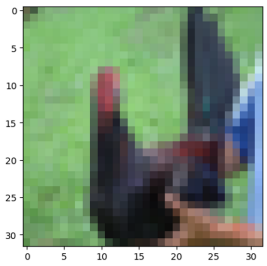


Figure 4.8: Self-Choice Factors on CIFAR-10 test set by ParallelBlock.

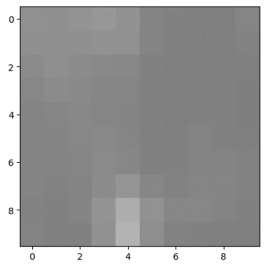
After showing the recognition accuracy, the same Self-Choice experiment as MNIST is also implemented on the CIFAR-10. According to Figure 4.8, the statistic result of CIFAR-10 is different from previous one, the vast majority of the Self-Choice Factors are distributed in the interval]0, 0.3[. It is worth to mention that none of a Self-Choice Factor is greater than 0.5. This result indicates that for most samples, the network is more inclined to use tied convolutional layer rather than untied convolutional layer. One possible explanation is that the objects can appear in any position of the image in CIFAR-10. Compared with untied convolution, tied convolution has the advantage of global feature extraction. Untied convolution can only play an auxiliary role in this training process, but this result proves that for CIFAR-10, tied convolution has a better adaptability..

4.3.5 Heatmaps of Self-Choice Weights on CIFAR-10

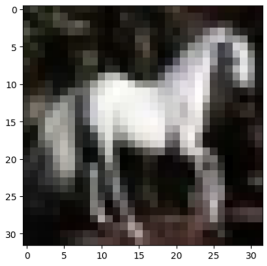
During the testing process, some of correctly identified and incorrectly identified samples are randomly selected and their Self-Choice weights are visualized. The results are shown in Figure 4.9 and Figure 4.10. You can see more samples in Appendix B. It can be seen that the Self-Choice selection maps of both correct and incorrect samples do not show obvious activity, indicating that the samples are more inclined to choose tied convolution as the way of identification, which is consistent with the previous statistical results.



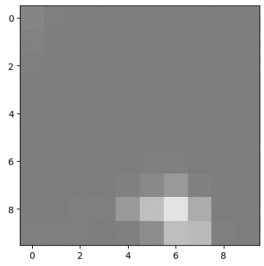
(a) Bird



(b) Bird Heatmap

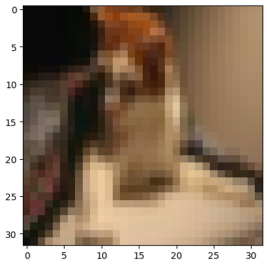


(c) Horse

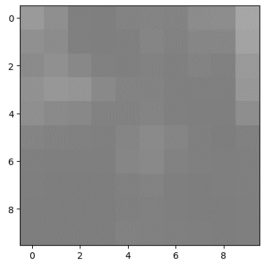


(d) Horse Heatmap

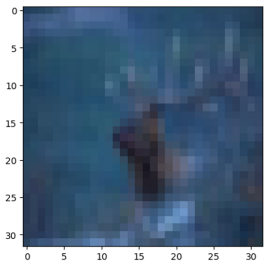
Figure 4.9: Heatmaps of correctly identified samples 'Bird' and 'Horse'.



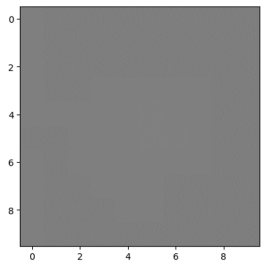
(a) Cat



(b) Cat Heatmap



(c) Deer



(d) Deer Heatmap

Figure 4.10: Heatmaps of incorrectly identified samples 'Cat' and 'Deer'.

Chapter 5

Evaluation on DIVA-HisDB

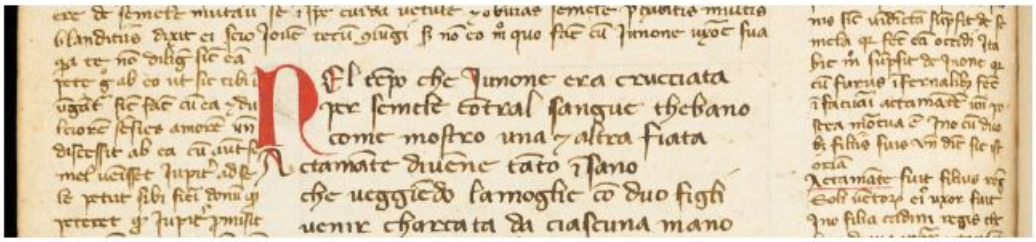
5.1 Experiments Goal

After observing the training results on the two datasets above, we can see that data does have a selection behavior in ParallelBlock. This chapter discuss the training and selection results of using ParallelBlock on a new dataset DIVA-HisDB [Sim16], Self-Choice results when multiple ParallelBlocks are used and the variants of ResNets with ParallalBlocks.

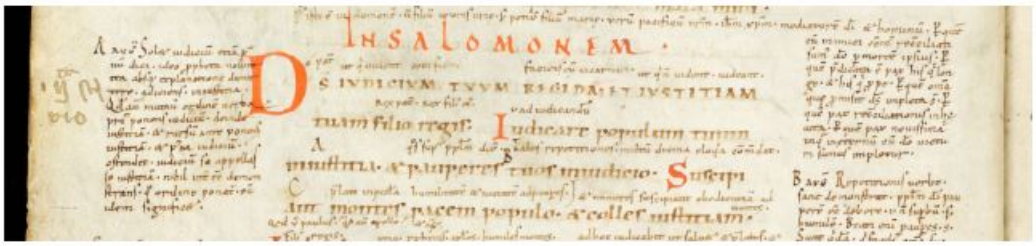
5.2 Training a BlockNet on DIVA-HisDB

5.2.1 Description of Dataset

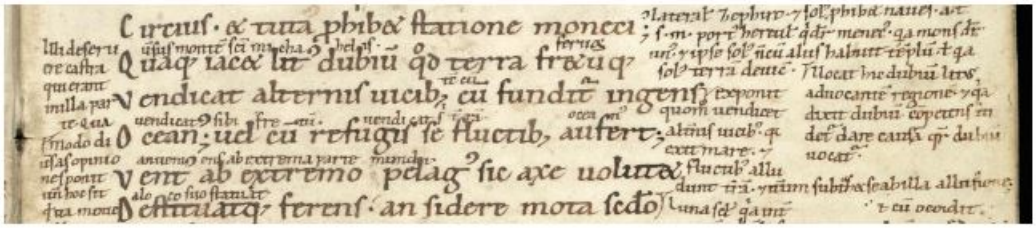
DIVA-HisDB is a historical manuscript database, which consists of 150 labeled pages whose manuscripts parts have been devided into three diffenent classes i.e. comments, decorations and main text. For the whole DIVA-HisDB, 63.97 percent of the total number of regions per annotated category are comments, 9.36 percent are decorations and 26.67 percent consists the main text body [Sim16]. It is a collection of three medieval manuscripts CB55, CSG18, and CSG863, which are shown in Figure 5.1.



(a) CB55, p.67v



(b) CSG18, p.116



(c) CSG863, p.17

Figure 5.1: CB55, CSG18 and CSG863 screenshots of DIVA-HisDB [Alb17].

5.2.2 Data Preparation

In data preparation period, since DIVA-HisDB does not store different classes in different images, it is necessary to pre-segment existing images to prepare exact training and test sets. The implementation way is to intercept parts of the image pixel by pixel with crop size. The labels of the newly generated images are defined by the position of the center point in the original image [Che16]. In the process of generating a new dataset, for the blank part of the original images, a new class has been added which is named as 'background' class, plus three classes in the original images, a total of four classes are utilized in training, validation and testing period. In this experiment, 6000 samples are generated according to the training, validation and test sets respectively 4000 samples, 1000 samples and 1000 samples.

5.2.3 Description of BlockNet Architecture

This neural network consists of four blocks, which are shown in Table 5.1. Network is trained with cross-entropy as loss function, SGD with learning rate 0.01 as optimizer and 50 as number of training epochs.

Layers	InChannels	OutChannels	KernelSize	Stride	Padding
ParallelBlock1	3	4	5	3	0
ParallelBlock2	4	6	3	2	0
ParallelBlock3	6	16	3	2	0
ParallelBlock4	16	32	3	2	0
FullyConnected1	32	4	/	/	/

Table 5.1: BlockNet with four ParallelBloks.

After each 50 mini batches, the validation set is used to evaluate the accuracy of currently trained neural network. At the initial time, an accuracy threshold is set to be 0.25, in the following training period, once the neural network reaches a higher accuracy than the threshold, the new model is preserved and the threshold is updated, so that after the training period, the best performing parameters of the neural network on the validation set is saved.

5.2.4 Training Result of BlockNet

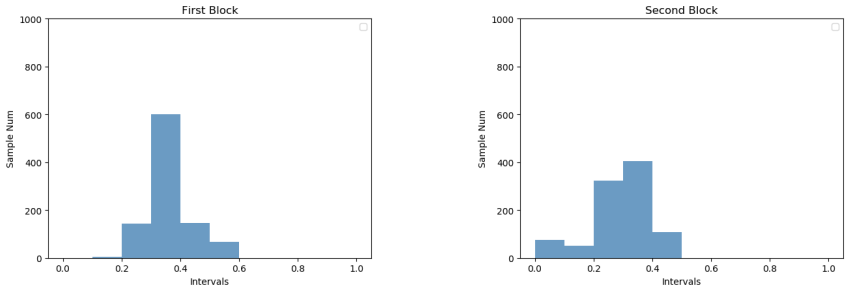
As can be seen from Table 5.2, the recognition accuracies of this network can eventually stabilize in the interval]0.8, 1.0[. In contrast, the accuracy of CB55 on validation set is 0.87188 and on test set is 0.84271, the accuracy of CS18 on validation set is 0.93803 and on test set is 0.91146, the accuracy of CS863 on validation set is 0.93018 and on test set is 0.88158.

Dataset	Validation	Test
CB55	0.87188	0.84271
CS18	0.93803	0.91146
CS863	0.93018	0.88158

Table 5.2: Accuracies of BlockNets on validation and test sets of CB55, CS18 and CS863.

5.2.5 Self-Choice on DIVA-HisDB by BlockNet

In the process of testing, the selection of these samples by a total of four blocks are counted and displayed in Figure 5.2. For test result of CS18, when the samples pass the first block, it is centered in the interval]0.3, 0.5[and it is similar to a normal distribution. It can be explained that the data show this kind of trend after normalization and regularization. The interval with the most samples after second block is]0.3, 0.4[. However, compared with the first one, the Self-Choice Factors of the second and third blocks show the trend that tilts to the right of middle point. This trend is even more obvious in the fourth block. The interval with the largest number of samples changes to]0.9, 1.0[. With the deepening of the neural network, the trends of the data become clear and the interval containing the most Self-Choice Factors moves toward the right side. Test results of the other two subsets also show similar trends.



(a) First ParallelBlock (b) Second ParallelBlock

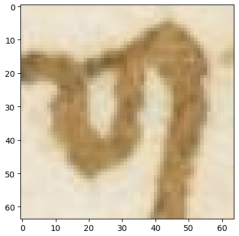
(c) Third ParallelBlock (d) Fourh ParallelBlock

Figure 5.2: Self-Choice Factors by ParallelBlocks on CS18 test set.

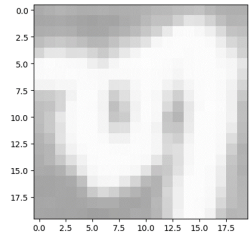
5.2.6 Heatmaps of Self-Choice Weights on DIVA-HisDB by BlockNet

The following Figure 5.3 and Figure 5.4 show the Self-Choice selection maps of four Parallel-Blocks). You can see more samples in Appendix C. These are two 'Comment' samples which are correctly and incorrectly identified. The map of the last block is not visualized because it is just a

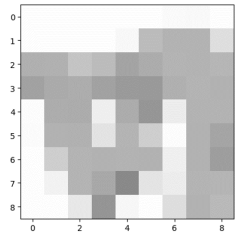
value.



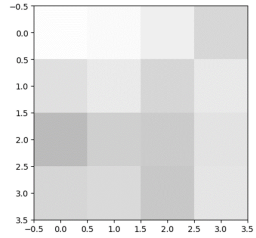
(a) Original Image



(b) First ParallelBlock

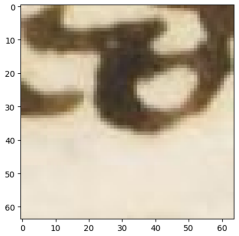


(c) Second ParallelBlock

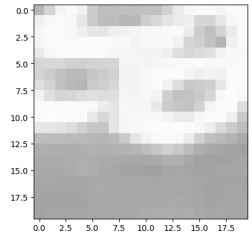


(d) Third ParallelBlock

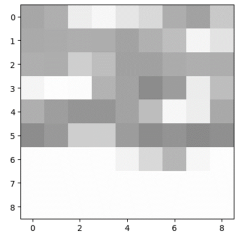
Figure 5.3: Heatmaps of a correctly identified sample 'Comment'.



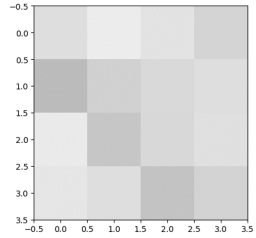
(a) Original Image



(b) First ParallelBlock



(c) Second ParallelBlock



(d) Third ParallelBlock

Figure 5.4: Heatmaps of an incorrectly identified sample 'Comment'.

According to the results of the visualization, it can be seen that the features are clearly extracted in the first and second blocks. It means that the selection maps can represent the characteristics of the samples, which further proves the map of the final layer can be seen as the sample selection. Although the sample in Figure 5.4 is identified incorrectly, selection maps still show clear features.

5.3 BlockNet with Different Input Size

5.3.1 Description of New Blocknet and Input Size

After the successful experiment with input size 64 by 64 in pixel, it is very interesting to try different sample size. Thus a new neural network is designed, it consists of five blocks, as their parameters are shown in Table 5.3. The dataset is still generated from DIVA-HisDB by same amount and proportion, the only difference is the size of each sample is 96 by 96 in pixel. Network starts training as before with cross-entropy as loss function, SGD with learning rate 0.01 as optimizer and 50 as number of training epoch.

Layers	InChannels	OutChannels	KernelSize	Stride	Padding
ParallelBlock1	3	6	1	2	0
ParallelBlock2	3	4	5	2	0
ParallelBlock3	4	6	5	2	0
ParallelBlock4	6	16	3	2	0
ParallelBlock5	16	32	3	2	0
FullyConnected	32	4	/	/	/

Table 5.3: BlockNet with five ParallelBloks.

5.3.2 Training Result of 96 by 96 BlockNet

As a subset of the dataset, CB55 is first used for training. However, the training results this time is not so good as expected. The accuracy on the validation set is only 25 percent, which is much lower than the previous training results. After adjusting parameters of the model, the accuracy is still not improved. This experiment is stopped after repeated training attempts, because accuracies cannot be improved significantly.

5.3.3 Possible Reasons of Failure

There is no doubt that this is a failed experiment. In summary, the reasons for failure may be as follows: Firstly, the structure of the neural network may not be able to extract the features. Secondly, the sample size is 96 by 96 in pixel and the new dataset is generated based on this size. Compared with the previous 64 in pixel, it may not be a suitable size.

5.4 Combination of ParallelBlock with ResNet

5.4.1 Aim of Combination

Combining the ParallelBlock with the current popular neural network structure is a exploratory experiment. According to previous experiments, we can know that the ParallelBlock can be used to train the neural network and show the Self-Choice of data. Once it is combined with the popular neural network frameworks, it may increase the accuracy of network and provide some new valuable information.

5.4.2 ResNet with Single ParallelBlock and Average Pooling

5.4.2.1 Network with ParallelBlock and Average Pooling

The datasets used in this experiment are the same as previous experiment, which are generated from DIVA-HisDB and 64 by 64 in pixel. ResNet18 [He16] is selected as the modified object. Because the input size should be 224 by 224 in pixel, considering previous failed experiments, the sample size are just resized from 64 to 224 instead of resampling. Specifically, there is an average pooling [Lin13] layer at the end of ResNet18. Some layers are inserted before it in the following order. It is shown in Table 5.4. Meanwhile, as a controlled trial, a derived network whose ParallelBlock is replaced by a tied convolution with same parameters is also designed. Just as previous experiment, training period starts after setting the threshold to 0.25 and the maximum epoch to 50.

Layers	InChannels	OutChannels	KernelSize	Stride	Padding
Convolution1	512	64	1	1	0
ReLU	/	/	/	/	/
ParallelBlock1	64	64	3	1	1
ReLU	/	/	/	/	/
AveragePooling	/	/	7	1	0
FullyConnected1	64	4	/	/	/

Table 5.4: Final parts of derived ResNet18 with single ParallelBlock and Average Pooling.

5.4.2.2 Training result of Network with Single ParallelBlock and Average Pooling

After combining ResNet18, the recognition rates are improved by both derived nets in Table 5.5. For ResNet18 with ParallelBlock, the accuracy of CB55 on validation set is 0.91041 and on test set is 0.90937, the accuracy of CS18 on validation set is 0.95512 and on test set is 0.94895, the accuracy of CS863 on validation set is 0.96959 and on test set is 0.96711. For ResNet without block structure, the accuracy of CB55 on validation set is 0.92916 and on test set is 0.91771, the accuracy of CS18 on validation set is 0.95726 and on test set is 0.93542, the accuracy of CS863 on validation set is 0.95608 and on test set is 0.95504.

Dataset	Validation	Test
Network with ParallelBlock		
CB55	0.91041	0.90937
CS18	0.95512	0.94895
CS863	0.96959	0.96711
Network without ParallelBlock		
CB55	0.92916	0.91771
CS18	0.95726	0.93542
CS863	0.95608	0.95504

Table 5.5: Accuracies of two derived ResNets on validation and test sets of CB55, CS18 and CS863, above one is the results with a ParallelBlock and below one is the results without ParallelBlock.

5.4.2.3 Self-Choice on DIVA-HisDB with Single ParallelBlock and Average Pooling

According to Figure 5.5, the distribution of Self-Choice Factors on ResNet18 with ParallelBlock is different from the previous neural network. This time, Self-Choice Factors of three data sets are concentrated in the left half of interval]0, 1[. It can be seen that the data still do choice, but the data reaching this layer are more inclined to tied convolution. A reasonable explanation is that after the data pass the previous layers, the features have been highly abstracted, which may not show the orignial features of DIVA-HisDB. The existence of average pooling layer may also influence the Self-Choice results. Since there is only one ParallelBlock, the trend of Self-Choice Factors between different ParallelBlocks cannot be performed.

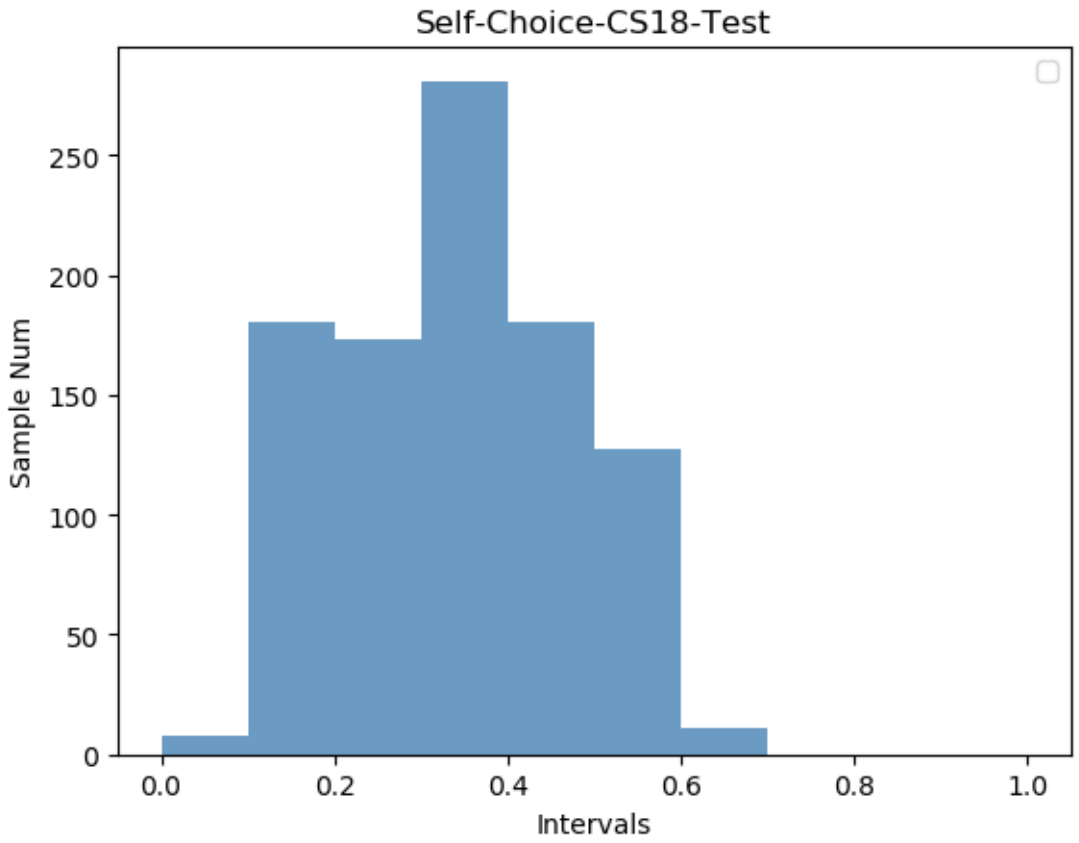


Figure 5.5: Self-Choice Factors by ParallelBlock on CS18 test set.

5.4.3 ResNet with ParallelBlocks

5.4.3.1 Network with ParallelBlocks and without Average Pooling

In order to explore the possible reasons for the previous Self-Choice result, as a comparison experiment to the previous two experiments, especially the ResNet18 with block structure, a new experiment is designed. As it is shown in Table 5.6, average pooling layer is replaced by another ParallelBlock. The input width and height of the block is 7 by 7, number of input channels is 64 and output channels is 64, kernel size of this layer is 7 by 7 with 1 as stride 0 as padding size, followed by a ReLU as activation function. The remaining experimental conditions and parameters are still unchanged.

Layers	InChannels	OutChannels	KernelSize	Stride	Padding
Convolution1	512	64	1	1	0
ReLU	/	/	/	/	/
ParallelBlock1	64	64	3	1	1
ReLU	/	/	/	/	/
ParallelBlock2	64	64	7	1	0
ReLU	/	/	/	/	/
FullyConnected1	64	4	/	/	/

Table 5.6: Final parts of derived ResNet18 with ParallelBlocks and without Average Pooling.

5.4.3.2 Training result of Network with ParallelBlocks

For derived ResNet with two ParallelBlocks, the recognition results are in Table 5.7.

Dataset	Validation	Test
CB55	0.91979	0.91979
CS18	0.93696	0.91354
CS863	0.96734	0.96931

Table 5.7: Accuracies of derived ResNet18 with ParallelBlocks and without Average Pooling on validation and test sets of CB55, CS18 and CS863.

5.4.3.3 Self-Choice on DIVA-HisDB with ParallelBlocks

It can be seen from Figure 5.6 that after the derived ResNet18 with two ParallelBlocks is trained, the samples of DIVA-HisDB show similar trends. After first block, the leftward intervals of]0, 1[

are selected, which means tied convolution is selected. But almost all samples are more inclined to the interval]0.9, 1[by second block, which means that without average pooling, data prefer untied convolution as forward operation, this result is in line with the result of the BlockNet.

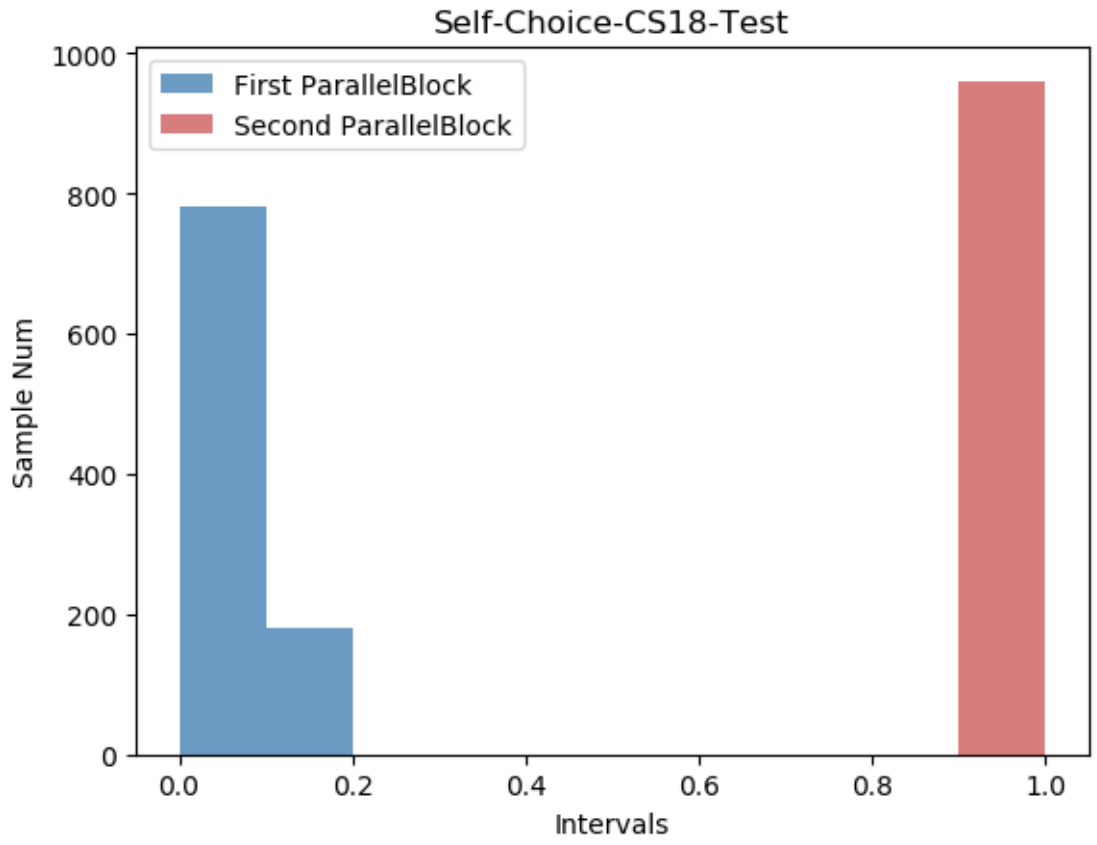
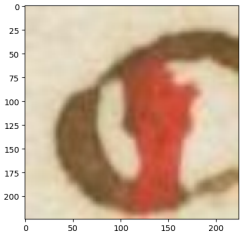


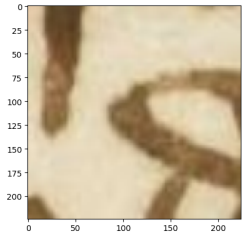
Figure 5.6: Self-Choice Factors by ParallelBlock on CS18 test set, while blue parts are the choice of first ParallelBlock and red parts are the choice of second ParallelBlocks.

5.4.3.4 Heatmaps of Self-Choice Weights on DIVA-HisDB by Derived ResNets

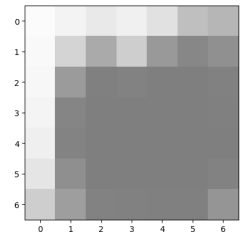
Figure 5.7 and Figure 5.8 shows visualization results of both derived ResNets. Compared with the previous Resnet18 with only one ParallelBlock, the features still can not be observed very clearly. However, after using ParallelBlock instead of average pooling, the visualization results seem to be smoother. Because the training accuracies of two derived ResNets are similar, it can be seen that the two ParallelBlocks replace the function of average pooling during training, which makes the middle Self-Choice Factors smoother. Usage of average pooling should be the reason why data in single ParallelBlock ResNet prefer tied convolution.



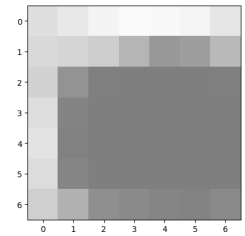
(a) Decoration



(b) Comment

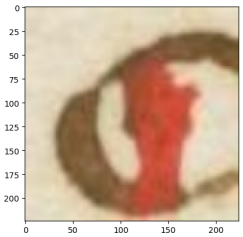


(c) Decoration Heatmap

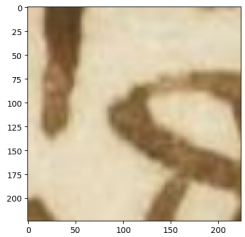


(d) Comment Heatmap

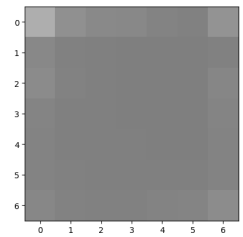
Figure 5.7: Heatmaps of correctly identified samples 'Decoration' and 'Comment' by derived ResNet18 with single ParallelBlock.



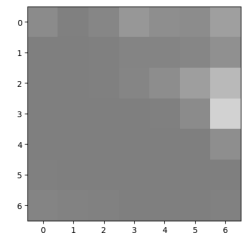
(a) Decoration



(b) Comment



(c) Decoration Heatmap



(d) Comment Heatmap

Figure 5.8: Heatmaps of correctly identified samples 'Decoration' and 'Comment' by derived ResNet18 with two ParallelBlock.

Chapter 6

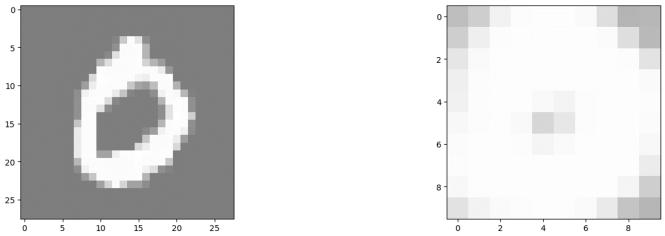
Conclusion

This thesis explores the possibility of combining tied and untied convolution by using ParallelBlocks to construct neural networks. According to the results of above experiments, we can draw the following conclusions: First of all, untied convolution class and ParallelBlock class can be easily formed into neural networks based on PyTorch. Besides, the neural networks constructed with ParallelBlocks can achieve at least the same accuracy comparing with its variants only with convolutional layers. Moreover, different datasets do make selection in the corresponding neural network, e.g. with similar neural network structure, MNIST prefers untied convolution while CIFAR-10 prefers tied convolution. In other words, Self-Choice by ParallelBlock really exists. In addition, we further explore the performance of ParallelBlock on the DIVA-HisDB, whose test results are better on the untied neural network. The Self-Choices by BlockNet can prove it from another perspective. Finally, combination of ParallelBlock with ResNet significantly improves test accuracy.

So far, there are still some points can be further studied. For example, further optimizing the performance of untied convolutional layer and ParallelBlock to make it more practical, not only a scientific tool. In addition, the last few experiments also prove that by using double-ParallelBlocks or multi-ParallelBlocks, the selection factors and maps of these ParallelBlocks become smooth and continuous. It is an effect which is similar to average pooling. The reason for this phenomenon is also worth exploring.

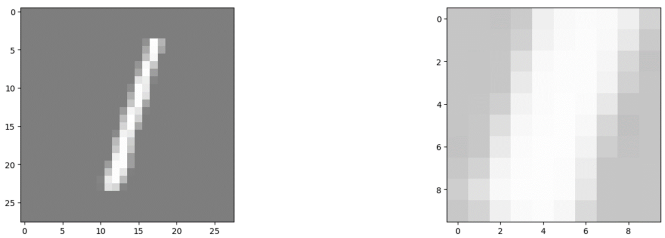
Appendix A

Heatmaps of Other Correctly Identified MNIST Samples



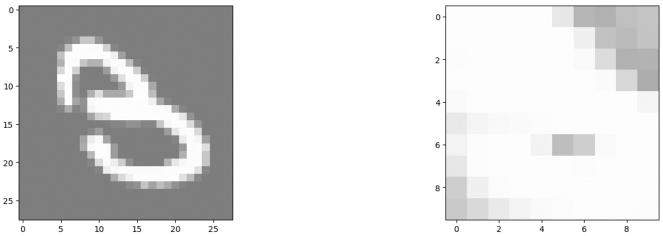
(a) Originla Image (b) Heatmap Image

Figure A.1: Heatmap of ’0’



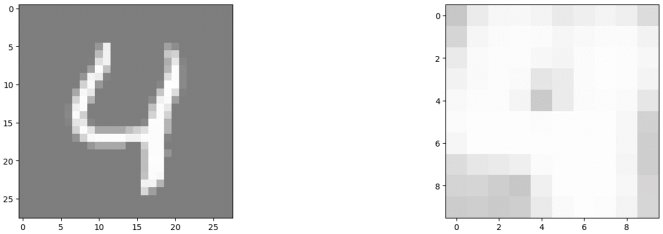
(a) Originla Image (b) Heatmap Image

Figure A.2: Heatmap of ’1’.



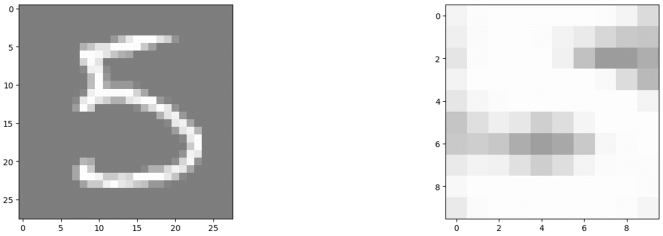
(a) Originla Image (b) Heatmap Image

Figure A.3: Heatmap of ’3’.



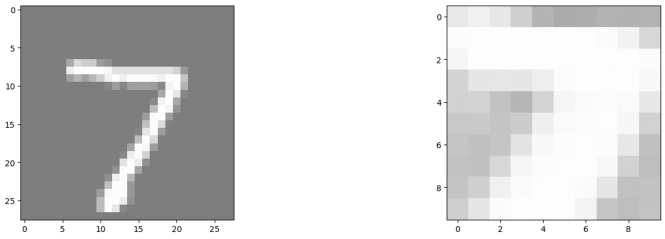
(a) Originla Image (b) Heatmap Image

Figure A.4: Heatmap of ’4’.



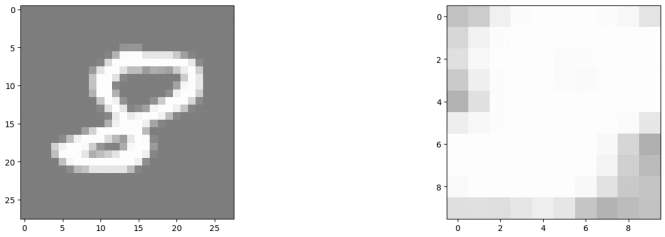
(a) Originla Image (b) Heatmap Image

Figure A.5: Heatmap of ’5’.



(a) Originla Image (b) Heatmap Image

Figure A.6: Heatmap of ’7’.

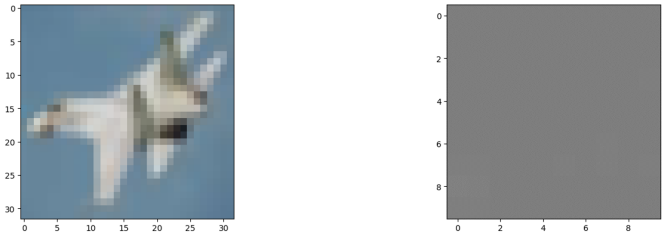


(a) Originla Image (b) Heatmap Image

Figure A.7: Heatmap of ’8’.

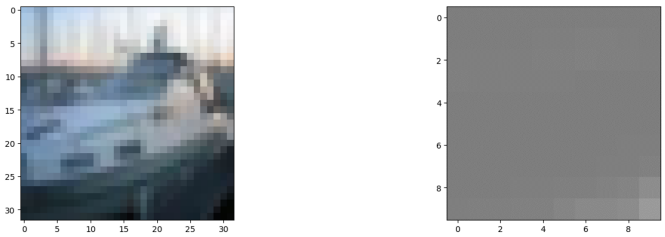
Appendix B

Heatmaps of Other CIFAR-10 Samples



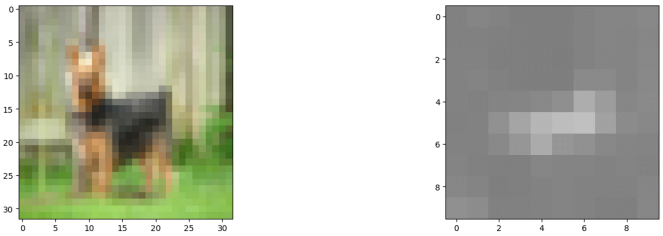
(a) Originla Image (b) Heatmap Image

Figure B.1: Heatmap of ’Plane’, which is correctly identified.



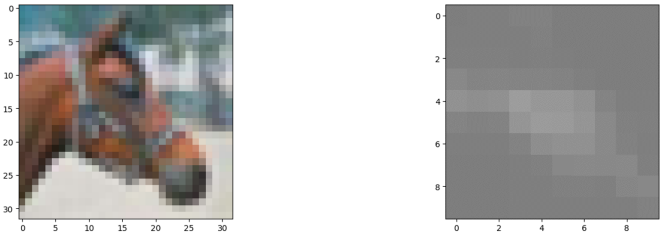
(a) Originla Image (b) Heatmap Image

Figure B.2: Heatmap of ’Ship’, which is correctly identified.



(a) Originla Image (b) Heatmap Image

Figure B.3: Heatmap of 'Dog', which is incorrectly identified.

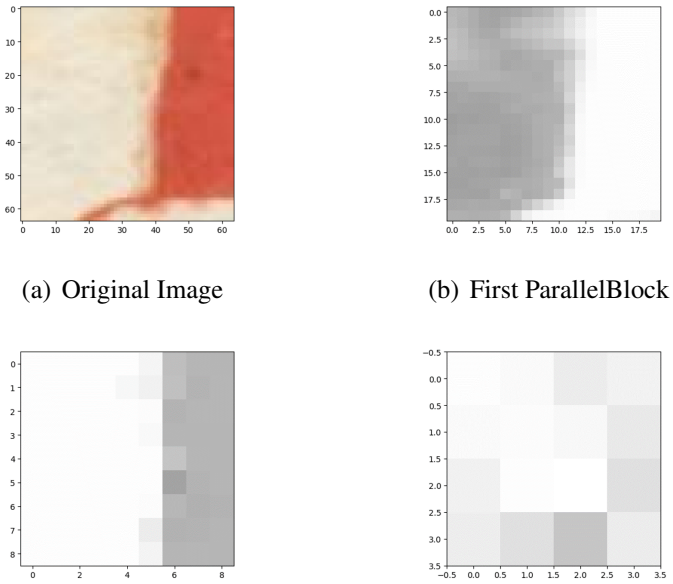


(a) Originla Image (b) Heatmap Image

Figure B.4: Heatmap of 'Horse', which is incorrectly identified.

Appendix C

Heatmaps by BlockNet



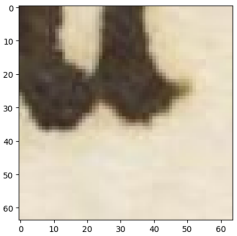
(a) Original Image

(b) First ParallelBlock

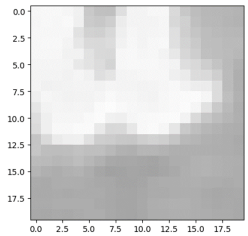
(c) Second ParallelBlock

(d) Third ParallelBlock

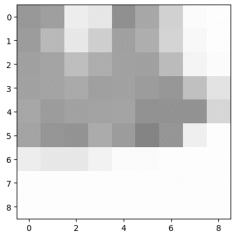
Figure C.1: Heatmaps of correctly identified sample 'Background' by BlockNet.



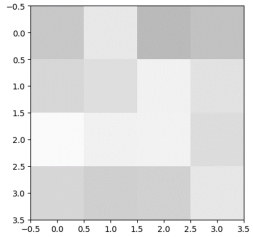
(a) Original Image



(b) First ParallelBlock



(c) Second ParallelBlock



(d) Third ParallelBlock

Figure C.2: Heatmaps of correctly identified sample 'Main Text' by BlockNet.

List of Figures

List of Tables

Bibliography

[Akl97] Selim G Akl. *Parallel computation: models and methods*, volume 4. Prentice Hall Upper Saddle River, 1997.

[Alb17] Michele Alberti, Mathias Seuret, Vinaychandran Pondenkandath, Rolf Ingold, and Marcus Liwicki. Historical document image segmentation with lda-initialized deep neural networks. In *Proceedings of the 4th International Workshop on Historical Document Imaging and Processing*, pages 95–100, 2017.

[Ape18] A Procedural Ape. Introduction of cifar-10. Website, 2018. `https://blog.csdn.net/qq_41185868/article/details/82793025`.

[Bir05] Alex Biryukov, Sourav Mukhopadhyay, and Palash Sarkar. Improved time-memory trade-offs with multiple data. In *International Workshop on Selected Areas in Cryptography*, pages 110–127. Springer, 2005.

[Bot18] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[Bus09] Raymond PL Buse and Westley R Weimer. Learning a metric for code readability. *IEEE Transactions on Software Engineering*, 36(4):546–558, 2009.

[Cha00] Anantha P Chandrakasan, William J Bowhill, and Frank Fox. *Design of high-performance microprocessor circuits*. Wiley-IEEE press, 2000.

[Che16] Kai Chen, Cheng-Lin Liu, Mathias Seuret, Marcus Liwicki, Jean Hennebert, and Rolf Ingold. Page segmentation for historical document images based on superpixel classification with unsupervised feature learning. In *2016 12th IAPR Workshop on Document Analysis Systems (DAS)*, pages 299–304. IEEE, 2016.

[Cop87] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 1–6, 1987.

[DB05] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

[Fan04] Zhe Fan, Feng Qiu, Arie Kaufman, and Suzanne Yoakum-Stover. Gpu cluster for high performance computing. In *SC'04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, pages 47–47. IEEE, 2004.

[Fin06] Terrence L Fine. *Feedforward neural network methodology*. Springer Science & Business Media, 2006.

[For02] David A Forsyth and Jean Ponce. *Computer vision: a modern approach*. Prentice Hall Professional Technical Reference, 2002.

[Fuk75] Kunihiko Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological cybernetics*, 20(3-4):121–136, 1975.

[Fuk80] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[Gao16] Yongbin Gao and Hyo Jong Lee. Local tiled deep networks for recognition of vehicle make and model. *Sensors*, 16(2):226, 2016.

[Gem92] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

[Gol88] Felix Goldberg and Abraham Berman. Linear algebra and its applications. 1988.

[Hac12] Wolfgang Hackbusch. *Tensor spaces and numerical tensor calculus*, volume 42. Springer, 2012.

[Han95] Jun Han and Claudio Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International Workshop on Artificial Neural Networks*, pages 195–201. Springer, 1995.

[Has95] Mohamad H Hassoun et al. *Fundamentals of artificial neural networks*. MIT press, 1995.

[He16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[Hir12] Isidore Isaac Hirschman and David V Widder. *The convolution transform*. Courier Corporation, 2012.

[Hor90] Roger A Horn. The hadamard product. In *Proc. Symp. Appl. Math*, volume 40, pages 87–169, 1990.

[Kes16] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[Kri09] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[Kri12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[LeC13] Yann LeCun. Mnist handwritten digit database, yann lecun, corinna cortes and chris burges, 2013.

[LeC15a] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[LeC15b] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, 20:5, 2015.

[Lee90] Peizong Lee and Zvi M Kedem. Mapping nested loop algorithms into multidimensional systolic arrays. *IEEE transactions on Parallel and Distributed Systems*, 1(1):64–76, 1990.

[Lin13] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[Lip05] Stanley B Lippman. *C++ Primer*. Pearson Education India, 2005.

[Mac17] MachineLearningNotebook. An introduction to cnns and deep learning. Website, 2017. https://mlnotebook.github.io/post/CNN1/.

[Pap98] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[PyT17] PyTorch. Training a classifier. Website, 2017. https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html?highlight=cifar.

[Roc97] Oliver Rockinger. Image sequence fusion using a shift-invariant wavelet transform. In *Proceedings of international conference on image processing*, volume 3, pages 288–291. IEEE, 1997.

[Ros04] Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. Are loss functions all the same? *Neural Computation*, 16(5):1063–1076, 2004.

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[Sim14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[Sim16] Foteini Simistira, Mathias Seuret, Nicole Eichenberger, Angelika Garz, Marcus Liwicki, and Rolf Ingold. Diva-hisdb: A precisely annotated large dataset of challenging medieval manuscripts. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 471–476. IEEE, 2016.

[Smi17] Leslie N Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472. IEEE, 2017.

[Son17] Cheng Song. From quantum matter states to deep learning. Website, 2017. http://toutiao.chinaso.com/dsj/detail/20170801/1000200033002801501562971931264919_1.html.

[Sri15] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385, 2015.

[ST04] John Shawe-Taylor, Nello Cristianini, et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[Sze15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[Thi97] Georg Thimm and Emile Fiesler. High-order and multilayer perceptron initialization. *IEEE Transactions on Neural Networks*, 8(2):349–359, 1997.

[Wil94] James Hardy Wilkinson. *Rounding errors in algebraic processes*. Courier Corporation, 1994.

[Yan15] LC Yan, B Yoshua, and H Geoffrey. Deep learning. *nature*, 521(7553):436–444, 2015.

[Zha07] Zheng Zhao and Huan Liu. Spectral feature selection for supervised and unsupervised learning. In *Proceedings of the 24th international conference on Machine learning*, pages 1151–1157, 2007.

[Zho00] Xiang Sean Zhou and Thomas S Huang. Cbir: from low-level features to high-level semantics. In *Image and Video Communications and Processing 2000*, volume 3974, pages 426–431. International Society for Optics and Photonics, 2000.

[Zou09] Hui Zou and Hao Helen Zhang. On the adaptive elastic-net with a diverging number of parameters. *Annals of statistics*, 37(4):1733, 2009.