RSKCNN – Introducing Randomly-Sparse-Kernel CNN

Zhexi Feng[1*] ,Yutao He[2] ,Yucheng Li[3]
[1] Department of Computer Science, Jilin University China, f18795465975zx@gmail.com
[2]Department of Computer Science, Nanjing University China, 221220102@smail.nju.edu.cn
[3]Department of Computer Science, Beijing Normal University -- Hong Kong Baptist University United International College China, yuchengli7625@gmail.com
[*]Corresponding author email: f18795465975zx@gmail.com

ABSTRACT

Convolutional neural networks (CNNs) perform excellently in many image processing and computer vision tasks. However, their complex structure and the vast number of parameters require substantial computational and storage resources. This makes them challenging to implement, especially on mobile or embedded devices. Furthermore, CNNs often face issues limited transferability and susceptibility to overfitting. Sparsity and pruning are commonly used techniques to address these issues. Current methods include SpatialDropout, block sparsity, structured pruning, dynamic pruning, and model-independent retraining-free sparsity. Our algorithm enhances CNNs by implementing a Dropout-like operation within the convolutional kernels. Drawing inspiration from sparse CNN and ROCKET methods, this approach employs randomly sparse convolutional kernels to reduce the data density processed during convolution operations. This novel method improves performance and efficiency, demonstrating its potential as a significant advancement in CNN architecture. The method is tested on serval popular datasets by adjusting parameters within the same model and on different hardware. It demonstrates improved training speed and accuracy reduced overfitting -- compared to traditional CNNs -- as measured by FLOPs and validation dataset accuracy.

Keywords: Convolutional neural networks, Sparsity, Randomly-Sparse-Kernel, Computer Vision, Algorithm

1. INTRODUCTION

Convolutional Neural Networks (CNNs) have been proven to perform exceptionally well in many classification tasks[1–3]. However, compared to earlier CNNs, state-of-the-art CNNs typically have higher model complexity and many parameters. For example, LeNet-5 has about 61,000 parameters[4], while the amount of parameters in VGG-16 has increased to more than 138,300,000[2]. The substantial growth in the number of parameters demands high memory capacity and computational power.

This has motivated the development of many optimisation methods for CNNs to reduce the number of parameters, and improve training speed, accuracy, and generalisation ability. In image classification tasks, Dropout methods are commonly applied to fully connected layers to prevent overfitting, but this approach is ineffective for images with spatial correlations. Tompson et al. introduced SpatialDropout[5]; it is used in convolutional layers. This method improves the model's generalization performance more effectively than standard Dropout methods, but it does not achieve Dropout at the single data level within convolutional layers. Subsequently, Wen et al. proposed a novel CNN architecture called BSCNN[6], which introduces sparse matrices to randomise convolutional kernels, reducing storage space requirements and time complexity. However, this method usually does not improve model accuracy.

The method proposed herein involves setting a probability p in the last convolutional layer and randomly initialising a mask matrix. Values below p in this mask matrix are set to zero. This mask matrix is then applied to the original convolutional kernel using the Hadamard product to obtain a randomly sparse kernel, which is then used for the subsequent convolution operation. The method represents a single-data-level Dropout operation within the convolutional kernel – different from SpatialDropout and IGCV[7, 8] – incorporating the concepts of sparse matrices and random kernels.

Our paper makes the following contributions:

- This method converts the convolutional kernel into a sparse convolutional kernel, reducing data density and computational complexity during convolution operations, thereby accelerating the training process.
- This method has a regularisation effect, which can reduce overfitting and thus improve model accuracy.
- This method is relatively easy to implement and provides an effective solution for models that overfit in a few iterations. Additionally, it offers extra hyperparameter tuning options for model trainers, potentially leading to further improvements in model accuracy.

There are some limitations to our method and the experiments reported in the ongoing: Firstly, in our sparse matrix, positions occupied by zeros in the final layer are not removed, indicating room for optimisation in the data structure. Secondly, our method leads to slower convergence and may require more iterations. Additionally, the datasets and models we have tested so far are small to medium-sized; we have yet to conduct conducted tests on large-scale datasets and models. Lastly, we have not employed self-learning techniques to determine the optimal probability p, which dictates the proportion of weight excluded from convolution operations.

The rest of this paper is organised as follows. Section 2 reviews related work by previous researchers; in section 3, we explain the specific implementation details and principles of the Randomly-Sparse-Kernel CNN (RSKCNN) method. Section 4 presents experiments conducted using our method on different datasets, network architectures, and hardware. Section 5 discusses key findings and some limitations of our work. Finally, section 6 gives a summary and introduces future work.

2. RELATED WORK

B. Liu et al. [9] propose a SCNN model, which reduces redundant parameters in CNNs through sparse decomposition of convolutional kernels. Initially, the decomposition is based on the reconstruction error of kernel weights, followed by fine-tuning the network while applying sparsity constraints. This process can zero out over 90% of the network's parameters, reducing computational complexity while maintaining accuracy. Also, an efficient sparse matrix multiplication algorithm is developed, which encodes the structure of sparse matrices as indices within the program, reducing indirect memory access. This algorithm achieves acceleration on CPUs.

J. Tompson et al. [5] propose a SpatialDropout method. This method considers the spatial correlation in CNNs by applying Dropout across the spatial dimensions, enforcing spatially discontinuous activation patterns. For a given convolutional feature tensor, SpatialDropout performs fewer Dropout trials across the entire feature map and extends the Dropout values across the entire feature map. This means that if a position in a feature map is dropped out, that position and all its neighboring pixels (within the feature map) will be set to zero. Models utilising SpatialDropout performed better on the FLIC dataset compared to models without SpatialDropout. It also helped reduce the spatial correlation within feature maps, enhancing the network's generalisation capability.

J. Park et al. [10] propose an efficient direct sparse convolution method, implemented through sparse matrix-dense matrix multiplication (SpMDM). This method is suitable for convolutional kernels with arbitrary sparsity patterns. A performance model based on the Roofline model is developed to predict the speedup of sparse convolution at different sparsity levels and to determine the valuable sparsity range. Additionally, the paper introduces the Guided Sparsity Learning (GSL) algorithm. The GSL algorithm combines insights from the performance model to incorporate acceleration-potential awareness into sparse learning, which improves inference speed by aggressively pruning selected layers during training while maintaining accuracy. Experiments were conducted on AlexNet and GoogLeNet in the paper, demonstrating that sparse convolution on different platforms -- including Intel Atom, Xeon, and Xeon Phi processors -- can achieve speedups ranging from 3.1-7.3x while maintaining inference accuracy.

S. Anwar et al. [11] introduce different levels of structured sparsity in CNNs, including channel-wise, kernel-wise, and intra-kernel strided sparsity. Channel level pruning removes all input and output weights of entire feature maps, directly resulting in a lightweight network. Kernel-level pruning removes complete k×k kernels, reducing the computational load of convolutional layers. Intra-kernel pruning prunes within the kernel, forcing certain weights to zero. The paper employs Particle Filter (PF) and Evolutionary Particle Filter (EPF) to identify candidate connections for pruning. PF uses a set of weighted particles to represent the filter distribution, simulating possible connection combinations to locate pruning candidates. Each particle simulates such a set of connections or masks, and importance weights are assigned by calculating each particle's misclassification rate (MCR). The pruned network is retrained to compensate for the losses due to pruning; meanwhile, it is further quantized through fixed-point optimisation, reducing the bit-width precision, significantly reducing the total storage size. Experiments were conducted on the MNIST and CIFAR-10 datasets, demonstrating that the pruned network, while reducing the number of parameters, exhibits better learning effects and superior generalisation capabilities. The experimental results indicate that through structured pruning, the network size can be reduced by 75% with minimal loss in accuracy.

G. Xie et al. [7] propose an IGCV2 method, that extends two structured sparse kernels to the product of more structured sparse kernels, each corresponding to a group convolution. This method introduces complementary conditions and balancing conditions to guide the design of structured sparse kernels. IGCV2 demonstrates advantages in model size, computational complexity, and classification performance.

K. Sun et al. [8] propose an IGCV3 method, which combines the composition of structured sparse kernels and the composition of low-rank kernels. Building on IGCV2, IGCV3 introduces a low-rank design pattern into group pointwise convolutions. The method forms convolutional kernels using the composition of structured sparse low-rank kernels. It employs a loose complementary condition applied to super-channels to guide the design of dense convolutional kernels. IGCV3 outperforms IGCV2 in image classification on ImageNet and object detection on COCO.

In [12], three model-agnostic sparsification methods -- Flat, Triangular, and Relative -- are developed, which can be run without the need for retraining. The Flat method defines a constant threshold for all layers, disregarding their weight distributions. The Triangular method introduces a varying threshold based on the size of the layers, with lower sparsity in earlier layers and higher sparsity in later layers. The Relative method defines a unique threshold for each layer based on the distribution of the weights in that layer. The model sparsity ratio and the drop in inference accuracy are used as evaluation metrics. The impact of different sparsification methods on various CNN models -- such as Inception-v3, MobileNet-v1, ResNet, VGG, and AlexNet -- is demonstrated. It is possible to reduce up to 73% of the weights -- with a compression factor of 3.7 times -- without exceeding a 5% loss in Top-5 accuracy.

To address CNNs' deployment limitations on embedded platforms, [13] proposes a new pruning algorithm that uses Linear Feedback Shift Registers (LFSRs) to locate non-zero weights, thereby reducing the resources needed to manage weights. Results show that this pruning method effectively reduces the model size of VGG, ResNet50, and InceptionV3 by 80%, 76%, and 65%, respectively.

N. Wen et al. [6] propose a Block-sparse CNN (BSCNN), which uses coefficient factors for convolutional kernels. Randomly generated sparse blocks based on certain scaling factors form a new convolutional kernel, maintaining the same effective parameters as the original dense kernel while expanding the receptive field of the topmost input layer. BSCNN integrates separable convolutions, which consist of depthwise convolution and pointwise convolution. This integration significantly reduces the number of parameters. Additionally, the paper introduces the use of the SUMMA algorithm to accelerate sparse matrix multiplication (SpMM) in BSCNN, reducing the need for temporary storage. In experiments comparing BSCNN with VGG16, BSCNN demonstrated reduced time and space complexity. When compared to im2col and MEC methods on GPUs, BSCNN showed lower memory requirements. Finally, using the VOC2007 + VOC2012 dataset, BSCNN's accuracy in object detection tasks was validated.

SRCNN, a sparse method for image detection tasks, is proposed in [14]. Specifically, the main goal of this architecture is to use small proposal boxes to replace the possible object regions searched by the Region Proposal Network, thereby avoiding the workload associated with label assignment. This architecture has demonstrated satisfactory accuracy and convergence speed on the COCO dataset.

In [15], it is emphasised that introducing sparsity in convolutional neural networks can accelerate execution. The paper introduces the 2:4 sparsity pattern and Sparse Tensor Cores deployed on NVIDIA GPUs. This pattern reduces storage requirements and optimises access efficiency. Mapping the 2:4 sparsity pattern onto Tensor Cores achieves a twofold performance improvement. The article provides a workflow demonstrating how to train sparse-introduced networks while ensuring accuracy and validating the effectiveness of these modifications.

The study [16] proposes a new loss function to incorporate sparsity into the cost function of CNNs, thereby reducing unnecessary neuron activations in hidden layers. This research applies this structure to the detection of bearing defects, demonstrating its effectiveness in accurately identifying defects and reducing overfitting.

B. Ji et al. [17] propose a highly efficient CNN inference architecture called FSCNN. It compresses fine-grained sparsity on a specially designed sparse data structure to accelerate the forward propagation process. Validated on architectures like YOLO and VGG16, FSCNN outperforms common deep-learning frameworks like PyTorch, demonstrating the potential of introducing sparsity for inference on CPUs. It also excels in fine-grained sparsity management.

3. METHODOLOGY

3.1. Method Overview

Our method involves applying the concept of Dropout to the convolutional kernels in the last convolutional layer. During each forward pass, a probability p is used to randomly mask out some weights in the convolutional kernels before performing the convolution operation. When utilising the method in the final application, all the weights in the convolutional kernels are incorporated and multiplied by the probability p to maintain consistency in the expected value. Theoretically, this method enables convolutional kernels to selectively retain important features, improving the model's generalisation ability and increasing accuracy. Additionally, reducing parameters provides a regularisation effect that helps reduce overfitting. Furthermore, since some parameters are set to zero, the computation speed on certain platforms can be improved. Our method is shown in Figure 1.

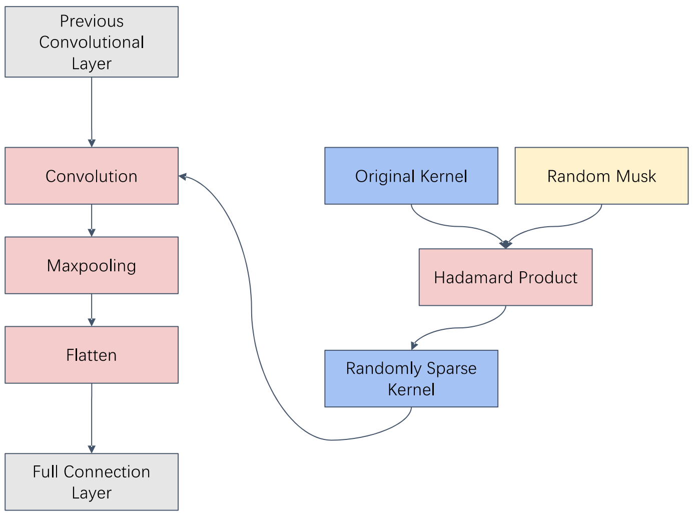

Figure 1: Flowchart of RSKCNN

3.2. Implementation Details

In practical implementation, a mask is generated during each forward pass where the elements are randomly distributed as 0s and 1s with equal probability. Values in this mask that are greater than p are set to 0, while those less than or equal to are set to 1, resulting in a binary mask matrix. This mask matrix is then element-wise multiplied (Hadamard product) with the convolutional kernels, producing the sparsely populated convolutional kernels used in the current convolution layer. During testing, the values in the mask matrix are set to p before performing the Hadamard product with the convolutional kernels, thus yielding the effective convolutional kernels. The operation of performing the Hadamard product between the mask matrix and the original kernel to obtain the Randomly Sparse Kernel is shown in Figure 2. Figure 3 illustrates how the convolution operation is performed using the Randomly Sparse Kernel in each iteration.

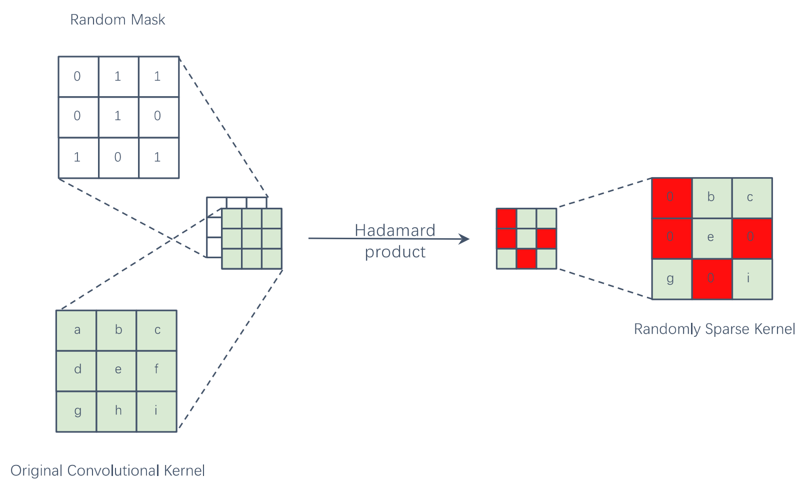

Figure 2: Details of RSKCNN (1)

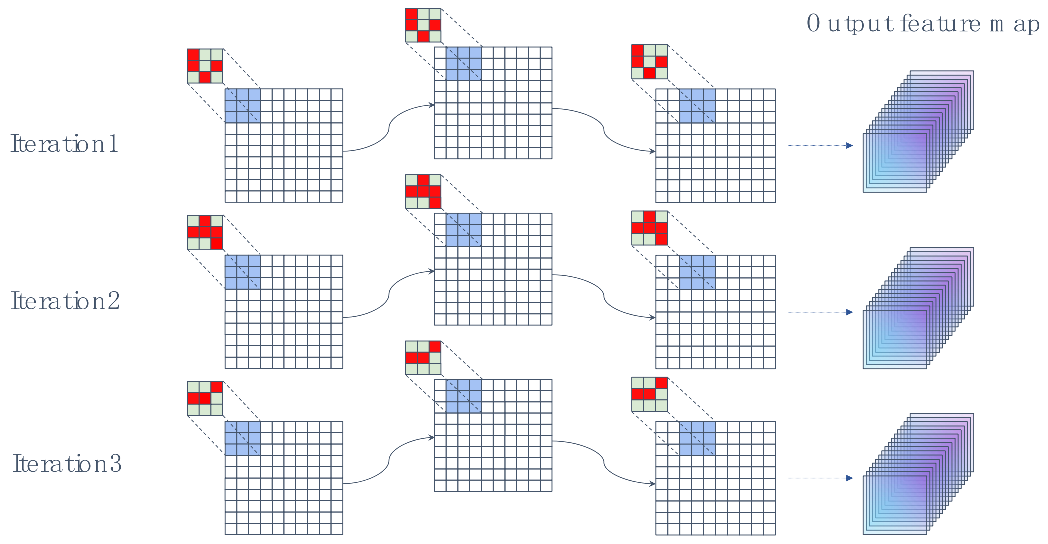

Figure 3: Details of RSKCNN (2)

3.3. Application Position

Our method only needs to be applied to the final convolutional layer before the fully connected layer. Applying it to earlier layers usually yields little benefit, as the sub-patterns/features captured in non-final layers are less meaningful. Additionally, the non-fixed sparse patterns in the earlier convolutional kernels may lead to unstable learning and convergence difficulties. Figure 4 shows the position where the RSKCNN method is used.

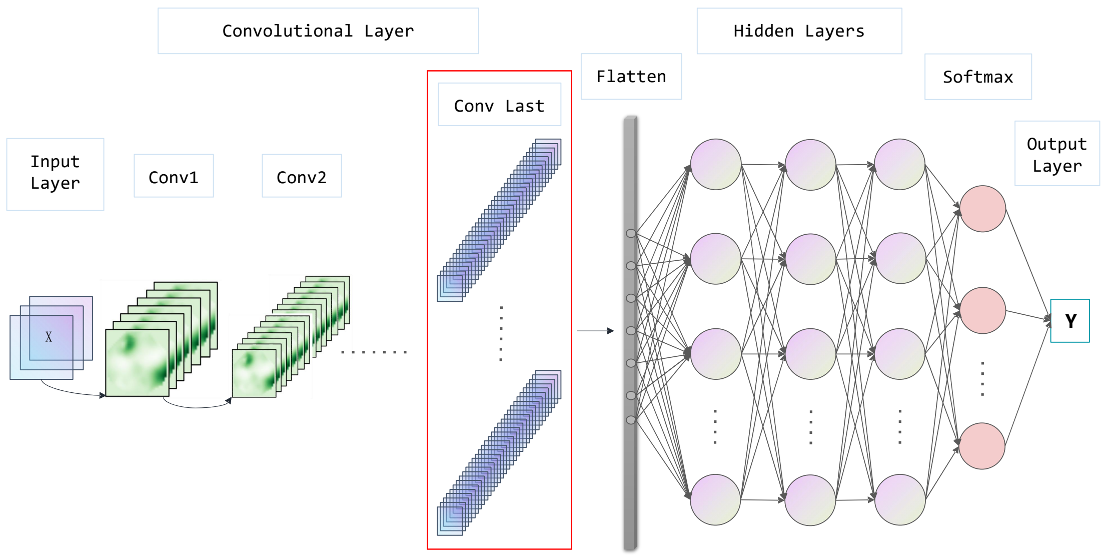

Figure 4: Application position of RSKCNN

3.4. Mathematical Description of the Model

This section introduces the mathematical process of our model.

3.4.1. Traditional CNNs Model

To explain the specific computational process of our method, we first introduce the basic convolution operation for multi-channel inputs as a background:

$$O(i,j) = \sum_{c=1}^{C} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I_c(i+m, j+n) \cdot K_c(m,n)$$

Where $O(i,j)$ is the value at position (i,j) in the output matrix O. $I_c(i+m, j+n)$ is the value at position $(i+m, j+n)$ in the input matrix I for channel c. $K_c(m,n)$ is the value at position (m,n) in the convolution kernel K for channel c. C is the number of channels in the input matrix. M and N are the height and width of the convolution kernel, respectively.

3.4.2. Our Model

Our model can be mathematically described as follows. Let $K(m,n)$ be the convolutional kernel and $M(m,n)$ be the mask matrix where each value is a Bernoulli distributed random variable with probability p of being 1, i.e.,

$$M(m,n) \sim \text{Bernoulli}(p)$$

The Hadamard product of the convolutional kernel K and the mask matrix M is given by:

$$\widetilde{K}(m,n) = K(m,n) \odot M(m,n)$$

where $K(m,n)$ is the resulting masked convolutional kernel and $\odot$ denotes the Hadamard product (element-wise multiplication).

3.4.3. Forward Propagation

During training, the output of our model can be expressed as:

$$O(i,j) = \sum_{c=1}^{C}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I_c(i+m, j+n) \cdot \widetilde{K_c}(m,n)$$

In the testing phase, the output is given by:

$$O(i,j) = \sum_{c=1}^{C}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I_c(i+m, j+n) \cdot K_c(m,n) \cdot p$$

To simplify the mathematical explanation, we merge and simplify the forward propagation during training as follows:

$$O(i,j) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I(i+m, j+n) \cdot \big(K(m,n) \odot M(m,n)\big)$$

3.4.4. Loss Calculation

The loss function L is computed based on the output O.

3.4.5. Backpropagation

The gradient of the loss with respect to the output O is denoted as $\frac{\partial L}{\partial O(i,j)}$.

To calculate the gradient of the loss function L with respect to the convolutional kernel K using the chain rule, we start with the following equation:

$$\frac{\partial L}{\partial K(m,n)} = \sum_i \sum_j \frac{\partial L}{\partial O(i,j)} \cdot \frac{\partial O(i,j)}{\partial K(m,n)}$$

Since the partial derivative of $O(i,j)$ with respect to $K(m,n)$ is given by $(I(i+m, j+n) \cdot M(m,n)$, we have:

$$\frac{\partial O(i,j)}{\partial K(m,n)} = I(i+m, j+n) \cdot M(m,n)$$

Substituting this into the gradient equation, the gradient of the loss with respect to the convolutional kernel K is computed as:

$$\frac{\partial L}{\partial K(m,n)} = \sum_i \sum_j \frac{\partial L}{\partial O(i,j)} \cdot I(i+m, j+n) \cdot M(m,n)$$

3.4.6. Weight Update

The kernel weights are updated using gradient descent:

$$K(m,n) \leftarrow K(m,n) - \eta \cdot \frac{\partial L}{\partial K(m,n)}$$

where η is the learning rate.

3.5. Theoretical Proof of Speed Accuracy and Regularisation

To demonstrate that sparse convolutional kernels can improve accuracy and provide regularisation, we need to prove the following two aspects mathematically.

3.5.1. Speed Improvement

Our method reduces the effective parameters involved in computations by setting some weights in the convolutional kernels to zero. For example, in networks like VGG, the parameter count can be as high as 27 million. In the last convolutional layer, there are 2.3 million parameters. This layer requires 512 convolutional kernels to perform convolution operations on an input feature map of size $14 \times 14 \times 512$ with padding = 1. Under these conditions, the number of operations required for this layer is $462{,}422{,}016$ addition operations and $462{,}422{,}016$ multiplication operations.

If $p = 0.5$, approximately half of these addition and multiplication operations involve zeros, which can speed up computation. This effect is more noticeable on CPUs. However, on parallel computing platforms like GPUs, due to block computation, if a block is not entirely zero, the zero elements still need to wait for the other elements to complete their computations before proceeding to the next operation. Therefore, GPUs have there is almost no performance improvemen.

3.5.2. Accuracy Improvement

Sparse convolutional kernels selectively retain important features, improving the model's generalisation ability and prediction accuracy. By applying the mask matrix M, unimportant weights in the convolutional kernel are zeroed out, and the retained weights are concentrated in positions that contribute more significantly to the output. This can also train convolutional kernels with the ability to capture sub-patterns/features. This selective feature extraction reduces the influence of noise and enhances the model's generalisation ability.

3.5.3. Regularisation Effect

Sparse convolutional kernels act as a regulariser by reducing the number of parameters and preventing overfitting. Consider a sparse convolutional kernel K and a corresponding mask matrix$M$, where $M(m, n) \sim \text{Bernoulli}(p)$. This means each element in the mask matrix is 1 with probability p and 0 with probability $1 - p$.

1) Forward Propagation: The output of the convolution operation is given by:

$$O(i,j) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I(i+m, j+n)\cdot\big(K(m,n)\odot M(m,n)\big)$$

2) Expected Loss Function: Assuming the loss function is L, we consider the expected value of the output:

$$E[O(i,j)] = E\left[\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I(i+m, j+n)\cdot\big(K(m,n)\odot M(m,n)\big)\right]$$

Since $M(m,n)$ follows a Bernoulli distribution and $E[M(m,n)] = p$, we have:

$$E[O(i,j)] = p\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I(i+m, j+n)\cdot K(m,n)$$

3) Regularisation Effect: Introducing the mask matrix M makes some weights in the convolutional kernel K zero, effectively reducing the number of parameters. Given N convolutional kernels, each of size $M \times N$, the expected number of non-zero parameters after applying the mask is $p \times M \times N \times N$. This sparsity introduces an L1 regularisation effect:

$$\|K\|_1 = \sum_{m,n} |K(m,n)|$$

L1 regularisation encourages sparsity, leading to some weights being exactly zero, which helps prevent overfitting.

3.6. Method Improvements

If a low p value is used with this method, it may result in slower convergence. We propose a new improvement to address this issue: setting different p values for each epoch. Given that a higher p value accelerates convergence, we can start with a higher p value, such as 1, and gradually decrease it throughout the training process until reaching the optimal p value.

3.7. Usage Considerations

1) The optimal p value may vary for complex models, necessitating manual parameter tuning and testing.

2) Ensure that the input size to the final convolutional layer is not smaller than the kernel size, as this would limit the optimisation potential (optimal p value approaches 1).

3) Initially test to find the best p value, then use a decremental approach, ideally reaching the target p value just before the peak performance iteration.

3.8. Steps for Adjusting Parameter p

1) Initially set p to 1 and run one epoch of the original model, observing when overfitting begins, or the improvement rate approaches zero.

2) Subsequently, different p values are tested to find the optimal value (this phase may have a slower fitting rate).

3) Finally, set the initial p value to 1, and decrement it evenly across iterations until reaching the target p value just before overfitting occurs. Switching directly to the target p value at the point of overfitting is not advisable, as this may prevent achieving the best performance.

4. EXPERIMENTS

We trained CNNs for image classification tasks across different domains and compared the performance with and without our method. Our findings show that convolutional neural networks using RSKCNN achieve higher accuracy and reduced overfitting. Extensive experiments were conducted to demonstrate the significant improvements brought by our approach across various software and hardware platforms, network architectures, datasets, and image tasks. The following sections present our validation tests. Each experimental result is averaged over multiple tests to minimise randomness and errors.

4.1. Datasets

We used several standard image classification task datasets, as shown in Table 1.

Data Set	Domain	Dimensionality	Training Set	Test Set
CIFAR-10	Object Recognition	32x32x3	50,000	10,000
CIFAR-100	Object Recognition	32x32x3	50,000	10,000
MNIST	Handwritten Digits	28x28x1	60,000	10,000
MNIST-fashion	Clothing Articles	28x28x1	60,000	10,000
Caltech101	Object Recognition	Varying	~9,144	~2,307
Caltech256	Object Recognition	Varying	~24,000	~6,084

Table 1: Information of datasets we have tested

4.1.1. CIFAR

The CIFAR-10 dataset is a standard benchmark dataset for image classification. It contains 60,000 32x32 colour images in 10 classes, with 6,000 images per class. The classes include airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The dataset is divided into 50,000 training images and 10,000 test images, which are widely used for evaluating image classification algorithms.

The CIFAR-100 dataset is an extended version of CIFAR-10. It containing 60,000 32x32 colour images in 100 classes, with 600 images per class. The classes are further grouped into 20 superclasses. Like CIFAR-10, the dataset is split into 50,000 training images and 10,000 test images, offering a more challenging classification task.

We conducted experiments using this dataset with CNNs (6 layers) on a Windows platform with a 4090 laptop; the results are shown in Figure 5 and Figure 6. On the CIFAR-10 dataset, our method increased the validation accuracy from 74% to 76%, and on the CIFAR-100 dataset, from 40% to 42%. The accuracy curves showed significant differences, indicating a substantial reduction in overfitting.

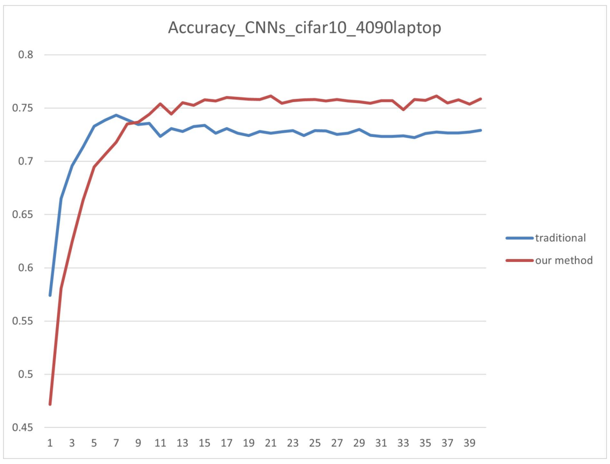

Figure 5: Accuracy of CNNs on cifar10 with 4090laptop

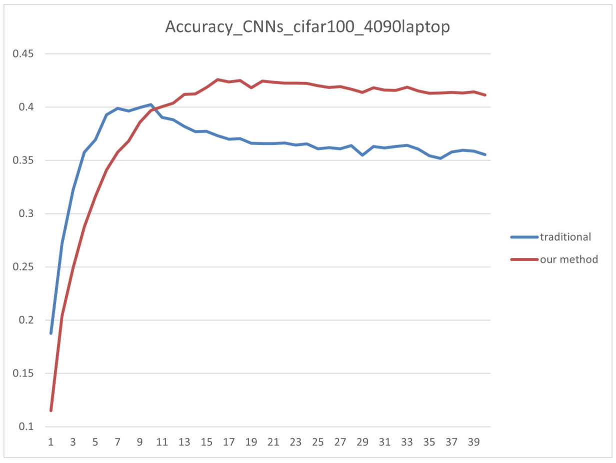

Figure 6: Accuracy of CNNs on cifar100 with 4090laptop

4.1.2. MNIST

The MNIST dataset is a standard benchmark for handwritten digit recognition. It contains 70,000 28×28 grayscale images in 10 classes (digits 0-9), 60,000 training images, and 10,000 test images. Due to its simplicity and wide application, it is extensively used for evaluating image classification algorithms.

The Fashion-MNIST dataset is an alternative to the MNIST dataset. It containing 70,000 28×28 grayscale images in 10 fashion categories (e.g., T-shirts, trousers, shoes). Like MNIST, it has 60,000 training images and 10,000 test images. Fashion-MNIST offers a more challenging dataset aimed at addressing the simplicity of MNIST.

We conducted experiments using this dataset with CNNs (6 layers) on a Windows platform with a 4090 laptop. The results are shown in Figure 7 and Figure 8. On the MNIST dataset, our method increased the validation accuracy from 99.3% to 99.4%, and on the Fashion-MNIST dataset, from 92% to 93%. The accuracy curves also showed significant differences, indicating reduced overfitting.

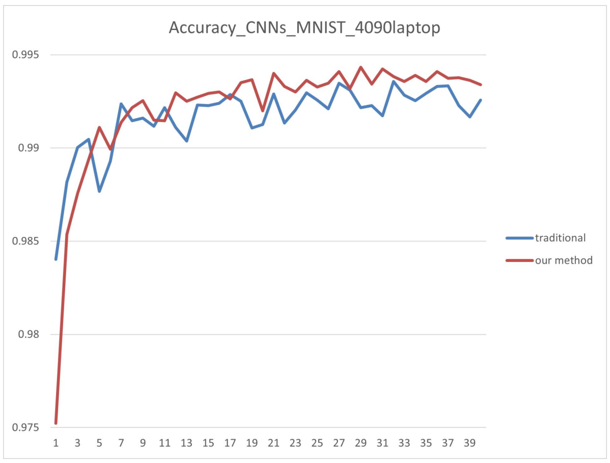

Figure 7: Accuracy of CNNs on MNIST with 4090laptop

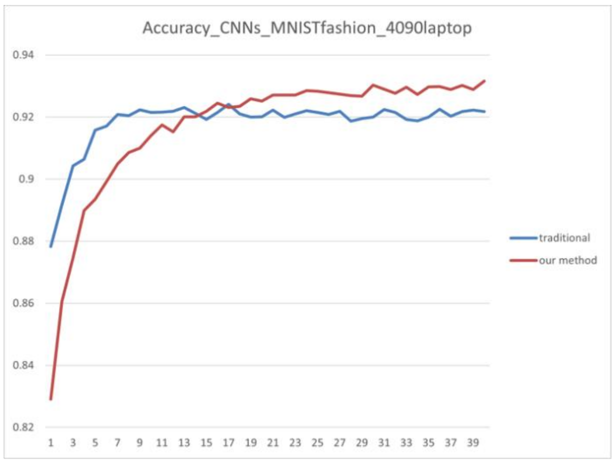

Figure 8: Accuracy of CNNs on MNISTfashion with 4090laptop

4.1.3. Caltech

The Caltech 101 dataset is used for object category recognition. It contains 101 categories plus a background category. Each category has 40 to 800 images, typically around 300×200 pixels, totaling approximately 9,146 images. The categories cover a range of objects such as animals, vehicles, and musical instruments, and are widely used in image classification and object detection research.

The Caltech 256 dataset extends Caltech 101, containing 256 categories plus a clutter category. Each category has at least 80 images, totaling 30,607 images. The images are typically around 300×200 pixels, providing more categories and a larger dataset for object category recognition and classification tasks.

We conducted experiments using the Caltech 101 dataset with VGG16 on an Ubuntu platform with A800. The results are shown in Figure 9 and Figure 10. Our method increased the validation accuracy from 63% to 66%. On the Caltech 256 dataset, using AlexNet on an Ubuntu platform with A800, our method increased the validation accuracy from 44% to 46%. The accuracy curves also showed significant differences, indicating reduced overfitting.

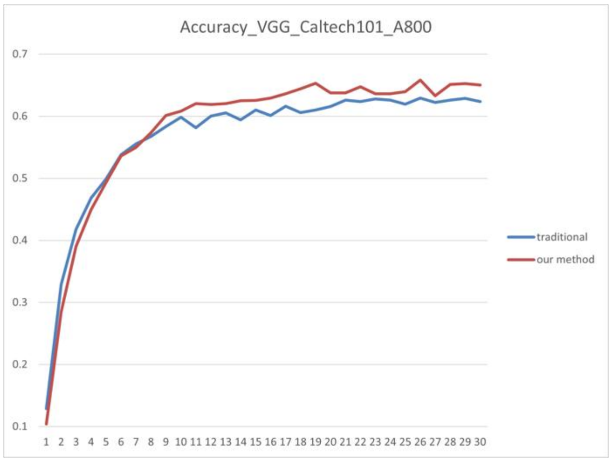

Figure 9: Accuracy of VGG on Caltech101 with A800

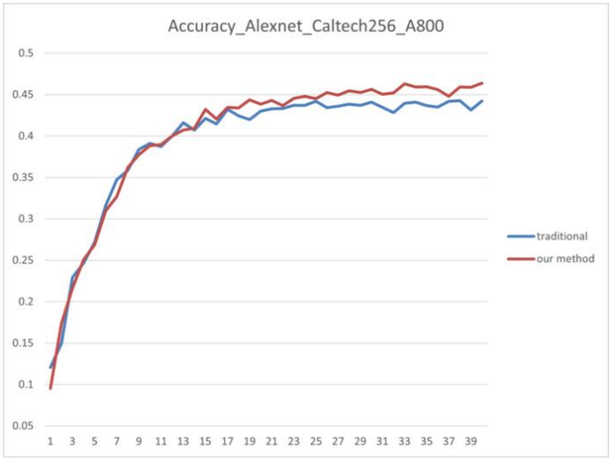

Figure 10: Accuracy of AlexNet on Caltech256 with A800

4.2. Network Architectures

We tested our method on various typical network architectures, as shown in the Table 2:

Model	Convolutional Layers	Fully Connected Layers	Total Layers
CNNs	3	3	6
AlexNet	5	3	8
VGG	13	3	16
ResNet	18	1	19

Table 2: Structure of models we have tested

4.2.1. CNNs (6 layers)

We conducted experiments on a 6-layer CNN using the CIFAR-100 dataset on a Windows 11 platform with a 4090 laptop. The results are as Figure 6. Our method provides immediate regularisation effects and accuracy improvements even with the simplest CNN.

4.2.2. AlexNet

We conducted experiments on AlexNet using the CIFAR-100 dataset on an Ubuntu platform with A800. The results are shown in Figure 11.

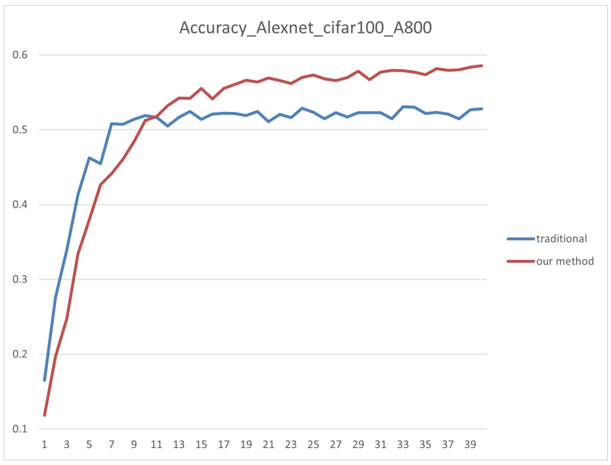

Figure 11: Accuracy of AlexNet on cifar100 with A800

Our method shows effectiveness even on these older and classic network architectures.

4.2.3. VGG16

We conducted experiments on VGG16 using the CIFAR-100 dataset on a Windows 11 platform with a 4090 laptop. The results are shown in Figure 12.

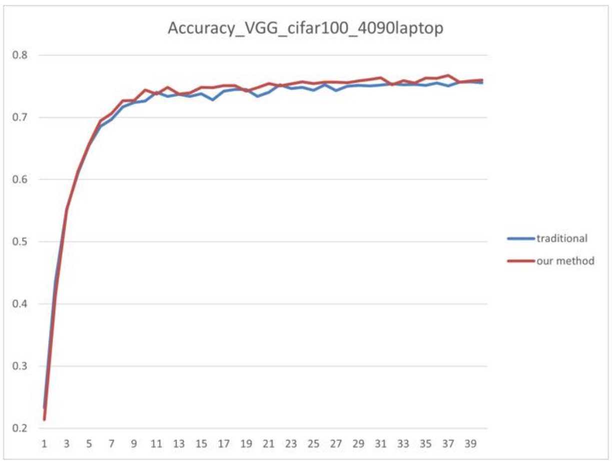

Figure 12: Accuracy of VGG on cifar100 with 4090laptop

It is noteworthy that our method shows minimal effect on VGG16. This may be due to the small input images and the excessive convolutional or pooling operations that reduce the image dimensions, leading to instability in the random sparsity of the final convolutional layer.

4.2.4. ResNet18

For ResNet18, the situation is slightly different. Since ResNet may skip specific convolutional layers, we designed a custom convolutional layer that cannot be skipped at the last layer. We tested two schemes: fixing the final convolutional layer and adding our method to the residual blocks. The results are shown in Figure 13 and Figure 14.

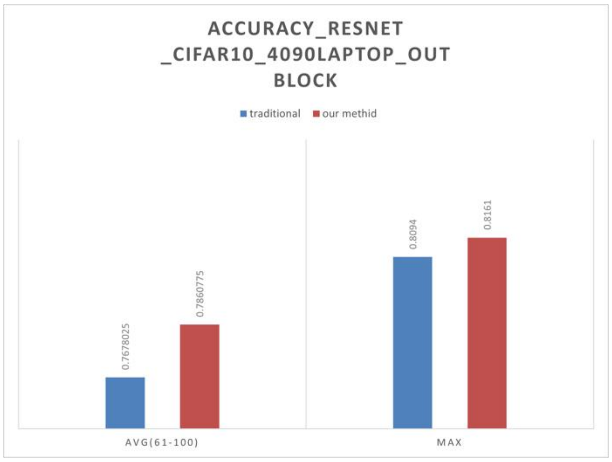

Figure 13: Accuracy of ResNet(outblock) on cifar10 with 4090laptop

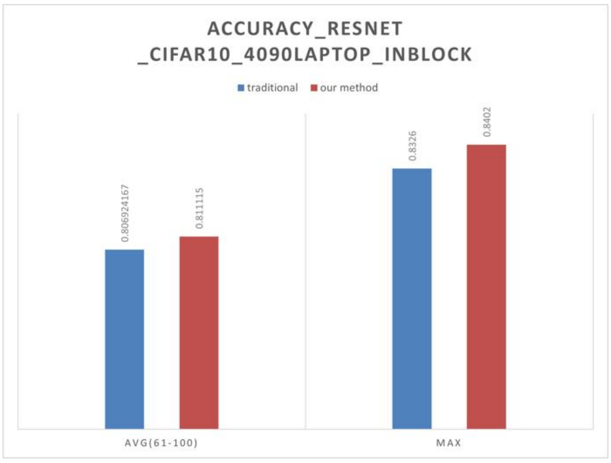

Figure 14: Accuracy of ResNet(outblock) on cifar10 with 4090laptop

When the final convolutional layer was fixed, the average accuracy over the last 40 epochs improved by about 2%. Adding our method to the residual blocks improved the average accuracy by about 0.4%, with the highest accuracy increasing by about 0.8%.

The minimal improvement in ResNet can be attributed to the following reasons:

- Residual connections effectively mitigate gradient vanishing, allowing very deep networks to train successfully. These connections themselves have a regularistion effect.
- Batch normalisation, widely used in ResNet, not only accelerates training but also has a regularistion effect by normalising each layer's input, reducing dependency on initial values and normalising small batches.
- ResNet's deep and complex structure makes it challenging to train with RSKCNNs or dropout as they may disrupt residual connections, affecting model convergence.
- In ResNet, residual modules add inputs directly to outputs, maintaining smooth information flow even in deep networks. Using RSKCNNs or Dropout might disrupt this flow, negatively impacting performance.

4.3. Software and Hardware Platforms

We tested our method on various software and hardware platforms, as shown in the Table 3:

Device	GPU/CPU	Manufacture	Operating System	Architecture

A800	GPU	Nvidia	Ubuntu	x86
4090 laptop	GPU	Nvidia	Windows 11	x86
i9-13980HX	CPU	Intel	Windows 11	x86
M3 Pro	CPU	Apple	macOS	ARM

Table 3: Information of hardware and platform we have tested

4.3.1. On CPU

Our method not only improves accuracy and reduces overfitting but also enhances training speed. The results are shown in Figure 15 Figure 16 Figure 17 and Figure 18.

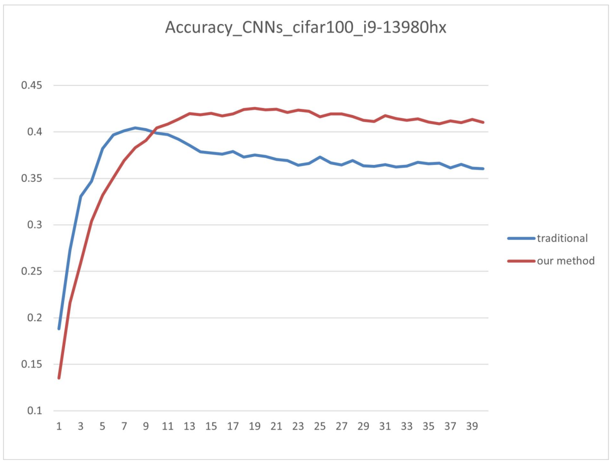

Figure 15: Accuracy of CNNs on cifar100 with i9-13980HX

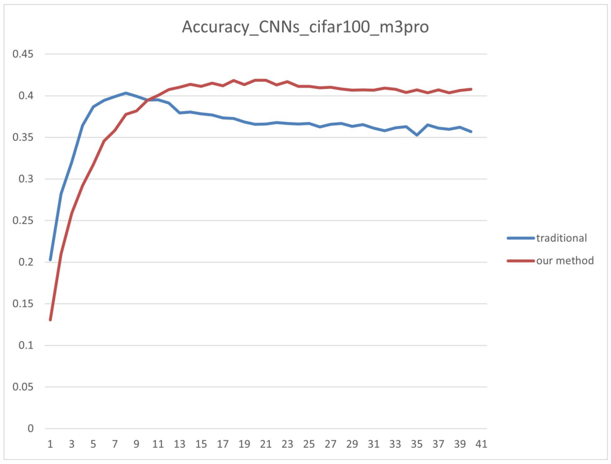

Figure 16: Accuracy of CNNs on cifar100 with m3pro

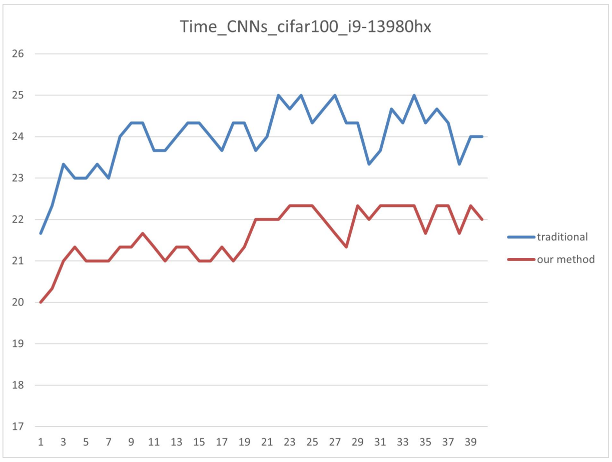

Figure 17: Time of CNNs on cifar100 with i9-13980HX

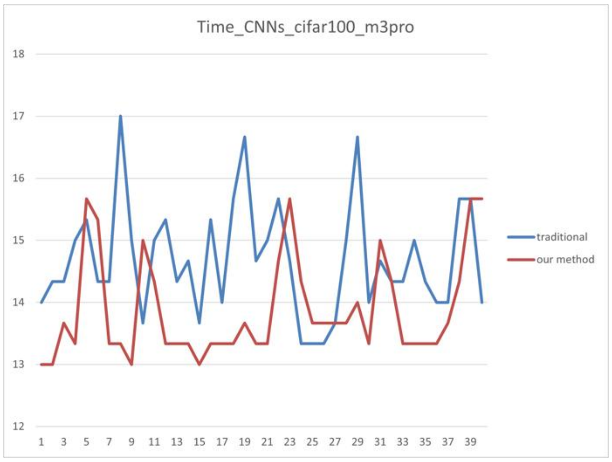

Figure 18: Time of CNNs on cifar100 with m3pro

This is because, in the final convolutional layer, which has the most data, $p \times 100\%$ of the nodes' computations become zero operations. In complex CPU architectures, specific optimisations can bypass ALU (Arithmetic Logic Unit) computations for zero operands, potentially enhancing specific computational speeds. However, the ALU's computational capabilities are still required for most complex arithmetic operations.

4.3.2. On GPU

Our method improves accuracy and reduces overfitting but does not significantly enhance computation speed. The results are shown in Figures 6 and 11.

In GPU matrix operations, even with zero operands, the GPU's parallel processing nature requires waiting for all operations in a batch to complete. Therefore, the speed gain from zero computations does not significantly impact the overall speed.

4.4. Experimental Results with the Improved Method

The improved method involves decrementing p from high to low. Higher p values result in faster convergence but are prone to overfitting, while appropriate p values reduce overfitting and improve accuracy. We utilised this principle by starting with higher p values and decreasing to the optimal p near the overfitting point. This approach achieves convergence speed comparable to traditional convolution methods with higher accuracy. The experimental results are shown in Figure 19.

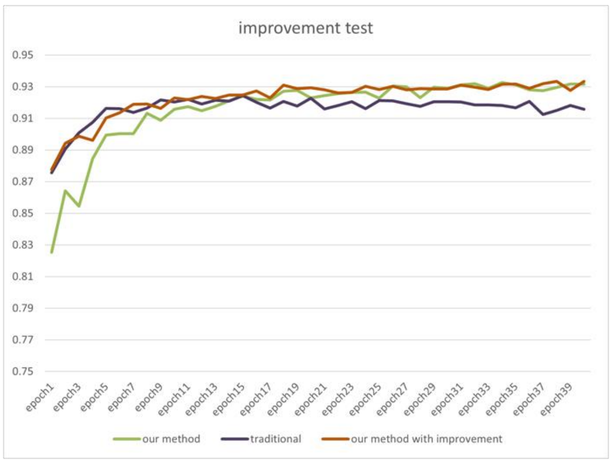

Figure 19: Accuracy of our method vs traditional vs our method with improvement

These results show that our method can be applied without any side effects by simply modifying the final convolutional layer, testing for the optimal p value, identifying the overfitting epoch, and applying our method to achieve improvements. The implementation is straightforward.

4.5. Other Metrics

In this section, we demonstrate the improvements our method offers on some other important metrics using the CIFAR-100 dataset with CNNs (6 layers) tested on a Windows platform with a 4090 laptop as an example.

4.5.1. Loss Curve

During the training process of machine learning and deep learning models, the loss function is a metric used to measure the difference between the model's predicted results and the actual labels. The loss curve shows the trend of loss values during training, usually including training and validation sets. By analysing the loss curve, one can intuitively understand the training process and performance of the model.

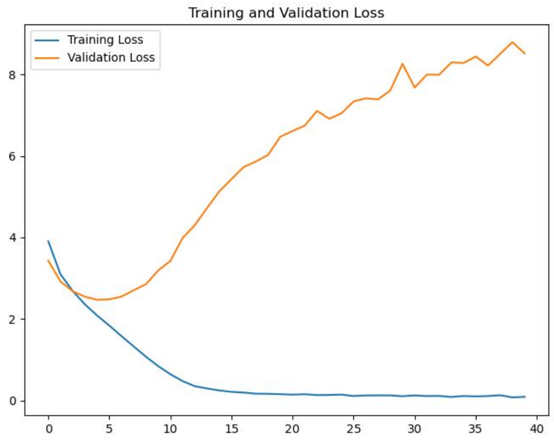

Figure 20: Loss of traditional

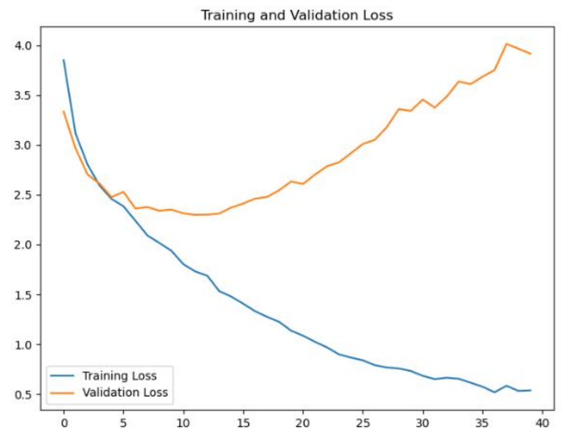

Figure 21: Loss of our method

As shown in Figure 20 and 21, our method slows down the rate of decline for the training set loss and the rate of increase for the validation set loss.

4.5.2. Top-1, Top-2, and Top-5 Curves

In multi-class classification tasks, the Top-K accuracy is a common evaluation metric used to measure whether the correct category is included among the top K most likely categories predicted by the model. Top-1 accuracy refers to the proportion of cases where the predicted category matches the true category exactly. Top-2 accuracy indicates the proportion of cases where the correct category is among the top two predicted categories. Top-5 accuracy refers to the proportion of cases where the correct category is among the top five predicted categories.

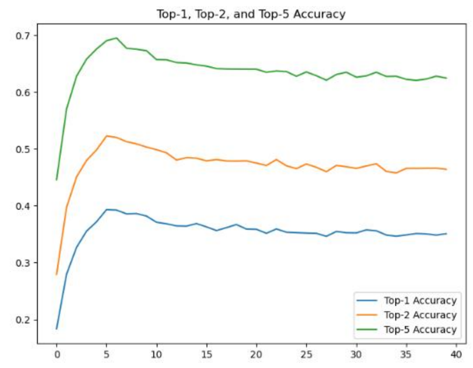

Figure 22: Top-K of traditional

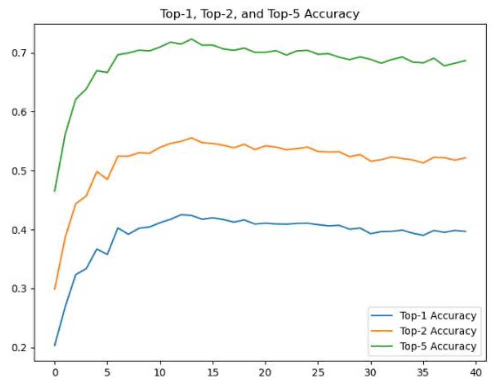

Figure 23: Top-K of our method

As shown in Figure 22 and 23, the peak values of the three Top-K accuracies all increased by about 2.5%, reducing the decline in accuracy caused by overfitting.

4.5.3. F1 Score

The F1 Score is an evaluation metric used in machine learning and statistics. It is particularly suitable for classification problems, especially in the case of imbalanced classes. It is the harmonic mean of precision and recall, taking into account the influence of both.

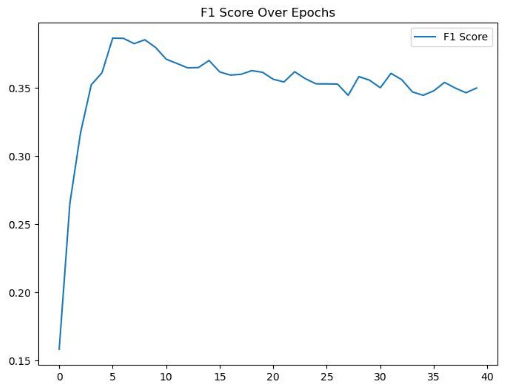

Figure 24: F1 score of traditional

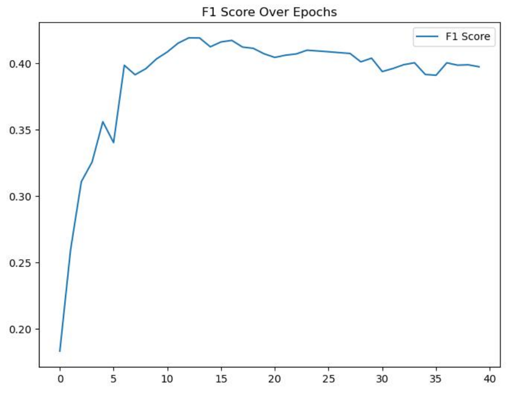

Figure 25: F1 score of our method

As shown in Figure 24 and 25, the peak F1 score increased from approximately 0.39 to about 0.42.

4.6. Other Tasks Using SRKCNN

We also tested other CNN-based tasks, such as speech recognition using the Google Speech Commands Datasets. This dataset is a standard benchmark for such tasks. Using our method on this dataset with a 6-layer CNN network on a Windows platform with a 4090 laptop, we improved accuracy from 88% to 90%. The results are shown in Figure 26.

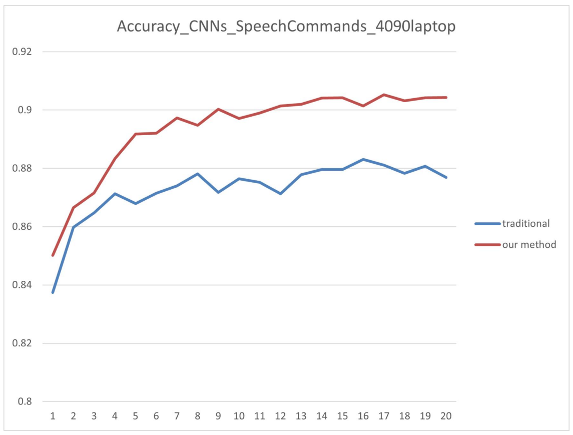

Figure 26: Accuracy of CNNs on Speech Commands with 4090laptop

This experiment demonstrates the potential of our method to enhance the performance of various neural networks across different tasks.

5. DISCUSSION

This method has contributed to solving the problem of the massive demand for training parameters of neural networks. At the same time, this method may also have some minor defects. For finding the best probability p, the current method still needs to manually initialise it to 1 and spend time testing different values, which cannot quickly find the best value through self-learning. At the same time, the 0 value generated by probability p acting on the mask matrix is not eliminated. Finally, due to the regularisation technology of Dropout, the requirement for the number of iterations in this method will increase, which will slow down the convergence rate.

6. CONCLUSION

In this paper, the Dropout operation at the single data level is applied to the last convolutional layer, and a sparse kernel is obtained for convolution operation through the mask matrix randomly initialised with parameter p. In addition, we propose an improved method to set different p values at each epoch to improve the convergence speed. It also describes how to find the best p value in the iterative process. Experimental results show that our method can improve the training accuracy and reduce the degree of overfitting in multiple data sets, network structures, and different hardware platforms, as well as improve the training speed on the CPU. In future work, we will continue to apply our method to the latest model and optimise its performance on more extensive and more complex datasets. Meanwhile, we hope that our work can be applied to medical impact analysis, large-scale image processing, etc., and can also be applied to CNN deployment in mobile devices and embedded systems.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” Advances in neural information processing systems, vol. 25, 2012.
[2] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv preprint arXiv:1409.1556, 2014.
[3] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath et al., “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups,” IEEE Signal processing magazine, vol. 29, no. 6, pp. 82–97, 2012.
[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.
[5] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler, “Efficient object localization using convolutional networks,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 648–656.

[6] N. Wen, R. Guo, B. He, Y. Fan, and D. Ma, “Block-sparse cnn: towards a fast and memory-efficient framework for convolutional neural networks,” Applied Intelligence, vol. 51, pp. 441–452, 2021.

[7] G. Xie, J. Wang, T. Zhang, J. Lai, R. Hong, and G. J. Qi, “Interleaved structured sparse convolutional neural networks,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 8847–8856.

[8] K. Sun, M. Li, D. Liu, and J. Wang, “Igcv3: Interleaved low-rank group convolutions for efficient deep neural networks,” arXiv preprint arXiv:1806.00178, 2018.

[9] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, “Sparse convolutional neural networks,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 806–814.

[10] J. Park, S. Li, W. Wen, P. T. P. Tang, H. Li, Y. Chen, and P. Dubey, “Faster CNNs with direct sparse convolutions and guided pruning,” in International Conference on Learning Representations, 2017. [Online]. Available: https://openreview.net/forum?id=rJPcZ3txx

[11] S. Anwar, K. Hwang, and W. Sung, “Structured pruning of deep convolutional neural networks,” ACM Journal on Emerging Technologies in Computing Systems (JETC), vol. 13, no. 3, pp. 1–18, 2017.

[12] A. H. Ashouri, T. S. Abdelrahman, and A. Dos Remedios, “Retraining-free methods for fast on-the-fly pruning of convolutional neural networks,” Neurocomputing, vol. 370, pp. 56–69, 2019.

[13] M. Qasaimeh, J. Zambreno, and P. H. Jones, “An efficient hardware architecture for sparse convolution using linear feedback shift registers,” in 2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP). IEEE, 2021, pp. 250–257.

[14] P. Sun, R. Zhang, Y. Jiang, T. Kong, C. Xu, W. Zhan, M. Tomizuka, L. Li, Z. Yuan, C. Wang et al., “Sparse r-cnn: End-to-end object detection with learnable proposals,” in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2021, pp. 14 454–14 463.

[15] A. Mishra, J. A. Latorre, J. Pool, D. Stosic, D. Stosic, G. Venkatesh, C. Yu, and P. Micikevicius, “Accelerating sparse deep neural networks,” arXiv preprint arXiv:2104.08378, 2021.

[16] A. Kumar, G. Vashishtha, C. Gandhi, H. Tang, and J. Xiang, “Tacho-less sparse cnn to detect defects in rotor-bearing systems at varying speed,” Engineering applications of artificial intelligence, vol. 104, p. 104401, 2021.

[17] B. Ji and T. Chen, “Fscnn: A fast sparse convolution neural network inference system,” arXiv preprint arXiv:2212.08815, 2022.