A New Classifier Based on CIFAR10 Data Set

Yuncong Cao[*]
Department of Computer Science
Fudan University
220 Handan Rd, Shanghai
`22307130424@m.fudan.edu.cn`

Abstract

This study explores the enhancement of image classification on the CIFAR-10 dataset by modifying and optimizing convolutional neural network architectures. We first developed a custom model from scratch, progressively integrating advanced techniques such as additional convolutional layers, batch normalization, dropout, and self-attention mechanisms. Subsequently, we optimized the ResNet18 architecture by adjusting hyperparameters and introducing modifications like dropout, adaptive learning rates, and depthwise separable convolutions. Our findings indicate that while certain enhancements significantly improve model performance, others, such as self-attention, may lead to overfitting. The final optimized model, with 722,186 parameters, achieved a test accuracy of 90.04%, demonstrating the effectiveness of strategic architectural adjustments and training optimizations in improving the accuracy and efficiency of image classification systems.

1 Introduction

The proliferation of deep learning in image classification has led to significant advancements in computer vision, yet the challenge to enhance model interpretability and performance persists. Convolutional Neural Networks (CNNs) have been the cornerstone in this domain, renowned for their proficiency in feature extraction and representation learning. However, as the complexity of tasks escalates, CNNs often require deeper architectures, which leads to increased computational costs and model complexity. This paper addresses the issue by proposing a synergistic approach that integrates technologies like self-attention mechanisms into CNNs to refine feature processing and improve classification outcomes.

1.1 Objective

The primary goal of this study is to significantly enhance the accuracy and efficiency of image classification on the CIFAR-10 dataset. We aim to achieve this by refining the architecture and training process of convolutional neural networks (CNNs), focusing on the integration of advanced techniques to address the challenges of high variability and complexity in image data.

1.2 Technological background

CNNs have been foundational in progressing image classification tasks, yet they face limitations in handling intricate patterns and ensuring model generalization. Recent advancements, such as new network architectures, auto-adapted learning rate, batch normalization, and attention mechanisms, have shown promise in overcoming these hurdles. Our study draws on these developments, aiming to explore their combined potential to boost classification performance on the CIFAR-10 dataset.

[*]Corresponding author. For additional information, including alternate contact details and ancillary affiliations, please use the provided email address.

1.3 Experimental Approach and Principal Findings

Approach: Our strategy involved a comprehensive enhancement of the CNN architecture for the CIFAR-10 image classification through various methods.

Findings: The implementation of these strategies resulted in a model that not only achieved significant accuracy improvements on the CIFAR-10 dataset but also showed enhanced generalization capabilities, outperforming conventional CNN models. This underscores the effectiveness of our approach in pushing forward the capabilities of image classification systems.

2 Methodology

Research Problem Definition: Our study aims to tackle the challenge of enhancing image classification accuracy and efficiency on the CIFAR-10 dataset, characterized by its small-sized images and high variability across ten distinct classes.

Experimental Approach:

1. **Model Architecture:**
   - *Depth Enhancement:* Increment in the number of convolutional layers to deepen feature extraction capabilities.
   - *Advanced Blocks:* Integration of residual and inception blocks for facilitating the training of deeper architectures.
   - *Self-Attention Mechanism:* Embedding a self-attention module within the architecture to dynamically focus on salient features.
2. **Training Optimizations:**
   - *Batch Normalization:* Application post each convolutional layer to stabilize and expedite the training process.
   - *Regularization:* Implementation of dropout and L2 regularization to mitigate overfitting.
   - *Pooling Adjustments:* Experimentation with various pooling strategies to find the optimal downsampling approach.
   - *Learning Rate and Optimizer:* Selection of an adaptive learning rate and an advanced optimizer to ensure effective model convergence.

Experimental Procedure: The model undergoes training on the CIFAR-10 training dataset with validation splits to monitor overfitting. Hyperparameters are fine-tuned based on validation performance, and the efficacy of the model is ultimately evaluated on the CIFAR-10 test dataset to measure classification accuracy and generalization.

Evaluation Metrics: Classification accuracy on the test set is the primary metric, with loss reduction during training serving as an indicator of model efficiency and learning effectiveness.

This methodology is crafted for reproducibility and scalability, aiming to strike a balance between model complexity and computational feasibility, with the ultimate objective of advancing the state-of-the-art in CIFAR-10 image classification.

3 Experiments

Improve the basic model step by step by checking the model test accuracy and the number of parameters.

3.1 The basic model

Here is our basic CNN model:

Table 1: The Basic Model Structure

Layer Name	Output Size	Layer Configuration
Input	32x32	-
conv1	28x28	3x3, 6
pool	14x14	2x2
conv2	10x10	3x3, 16
pool	5x5	2x2
fc1	-	120 Neurons
fc2	-	84 Neurons
fc3	-	10 Neurons

Table 2: Model Hyperparameters

Hyperparameter	Value
Epoch Number	20
Learning Rate	0.001
Momentum	0.9
Batch Size	8
Optimizer	SGD

Table 3: Model Performance

Metric	Value
Number of Parameters	62,006
Test Accuracy	62.06%

3.2 Changing the convolutional layer configurations

Table 4: Net Model Structure

Layer Name	Output Size	Layer Configuration
Input	32x32	-
conv1	32x32	3x3, 64, padding=1
pool	16x16	2x2
conv2	16x16	3x3, 128, padding=1
pool	8x8	2x2
conv3	8x8	3x3, 256, padding=1
fc1	-	120 Neurons
fc2	-	84 Neurons
fc3	-	10 Neurons

Table 5: Model Hyperparameters

Hyperparameter	Value
Epoch Number	100
Learning Rate	0.001
Momentum	0.9
Batch Size	8
Optimizer	SGD

Table 6: Model Performance

Metric	Value
Number of Parameters	2,348,030
Test Accuracy	77.91%

(1) Initial Model: Uses simpler configurations with two convolutional layers (conv1 and conv2) having 6 and 16 output channels, respectively, and a kernel size of 5 with no padding mentioned. This setup is straightforward but might be less effective at preserving spatial dimensions and capturing finer details due to the larger kernel size and absence of padding.

(2) Optimized Model: Employs three convolutional layers with more output channels (64, 128, and 256) and a smaller kernel size of 3, along with padding set to 1. This configuration allows the network to capture more complex features with increased depth while preserving spatial dimensions, leading to potentially better performance on complex image classification tasks. And we increase the epoch number to 100.

(3) Potential for Overfitting: With increased complexity and capacity, the optimized model may be more prone to overfitting, especially if trained on a small dataset without adequate regularization techniques (e.g., dropout, batch normalization) or data augmentation.

```
[EPOCH 041 | IT 04000] LOSS:  0.001
[EPOCH 041 | IT 06000] LOSS:  0.001
[EPOCH 042 | IT 02000] LOSS:  0.000
[EPOCH 042 | IT 04000] LOSS:  0.000
[EPOCH 042 | IT 06000] LOSS:  0.000
............
............
[EPOCH 099 | IT 06000] LOSS:  0.000
[EPOCH 100 | IT 02000] LOSS:  0.000
[EPOCH 100 | IT 04000] LOSS:  0.000
[EPOCH 100 | IT 06000] LOSS:  0.000
```

(4) Training process data: Based on the provided training process data, where the loss progressively decreases with each epoch and eventually approaches zero, it might be concluded that the number of epochs could be excessively high. When the loss value consistently decreases and stabilizes or nears zero, it usually indicates that the model has learned as much information as possible from the training data. Further training may not significantly improve the model's performance. Moreover, continuing the training could increase the risk of overfitting, where the model performs well on the training data but poorly on unseen data.

3.3 Reduce the model's complexity and avoids overfitting

Table 7: Net Model Structure

Layer Name	Output Size	Layer Configuration
Input	32x32	-
conv1	32x32	3x3, 64, padding=1
bn1	32x32	BatchNorm2d
pool	16x16	2x2
conv2	16x16	3x3, 128, padding=1
bn2	16x16	BatchNorm2d
pool	8x8	2x2
conv3	8x8	3x3, 256, padding=1
bn3	8x8	BatchNorm2d
glob avg pool	1x1	AdaptiveAvgPool2d
fc1	-	256 Neurons
dropout	-	Dropout(0.5)
fc2	-	120 Neurons
fc3	-	84 Neurons
Output	-	10 Neurons

Table 8: Model Hyperparameters

Hyperparameter	Value
Epoch Number	50
Learning Rate	0.001
Momentum	0.9
Batch Size	8
Optimizer	SGD

Table 9: Model Performance

Metric	Value
Number of Parameters	413,566
Test Accuracy	82.14%

(1) Batch Normalization (BN)

Improvement: It adds batch normalization layers after each convolutional layer (bn1, bn2, bn3). Batch normalization can accelerate deep network training by reducing internal covariate shift, improving training stability, and allowing for higher learning rates.

Impact: Helps in faster convergence and makes the network more stable through normalization of the layers' inputs.

(2) Global Average Pooling (GAP)

Improvement: It introduces a global average pooling layer before the fully connected layers. This layer reduces each feature map to a single value by averaging over all units, decreasing the total number of parameters and effectively reducing overfitting.

Impact: Reduces the model's complexity and spatial dimensions, making it less prone to overfitting compared to models with more fully connected layers. Also, it adapts the network to accept input images of varying sizes.

(3) Dropout

Improvement: A dropout layer with a rate of 0.5 is added before the final fully connected layers. Dropout prevents overfitting by randomly setting a fraction of input units to 0 at each update during training time, which helps to prevent neurons from co-adapting too much.

Impact: Enhances the model's generalization ability by reducing overfitting, making the network's predictions more robust to slight variations in the input data.

(4) Epoch number: Based on the training process data of the last model, we reduce the epoch number from 100 to 50.

3.4 Adding self-attention mechanisms and self-adjusting learning rate

Table 10: Net Model with Self-Attention Structure

Layer Name	Output Size	Layer Configuration
Input	32x32	-
conv1	32x32	3x3, 64, padding=1
bn1	32x32	BatchNorm2d 64
Self-Attention1	32x32	In_dim=64
pool	16x16	2x2
conv2	16x16	3x3, 128, padding=1
bn2	16x16	BatchNorm2d 128
Self-Attention2	16x16	In_dim=128
pool	8x8	2x2
conv3	8x8	3x3, 256, padding=1
bn3	8x8	BatchNorm2d 256
Self-Attention3	8x8	In_dim=256
Global Avg Pool	1x1	AdaptiveAvgPool2d
fc1	-	256 Neurons
dropout	-	Dropout(0.5)
fc2	-	120 Neurons
fc3	-	84 Neurons
Output	-	10 Neurons

Table 11: Model Hyperparameters

Hyperparameter	Value
Epoch Number	50
Learning Rate	0.001
Momentum	0.9
Batch Size	8
Optimizer	SGD

Table 12: Model Performance

Metric	Value
Number of Parameters	521,646
Test Accuracy	81.77%

Analysis: After adding self-attention mechanisms and self-adjusting learning rate, we can find that the test accuracy of the model drops a little instead of enhancing. I think there is two possible reasons.

(1)Self-adjusting learning rate: If the learning rate is reduced too late, the model may overfit during the initial stages of training. By the time the learning rate is finally lowered, the model may have already lost its ability to generalize to unseen data.

(2)Self-attention mechanism Self-attention adds additional parameters and complexity to the model. This can lead to overfitting, especially if the training dataset is not large enough to support the increased complexity.

3.5 Tuning ResNet18

(1) Adding Dropout layer and weight decay

add a dropout layer before the final full-connected layer

Table 13: Model Hyperparameters

Hyperparameter	Value
Epoch Number	135
Learning Rate	0.001
Momentum	0.9
Batch Size	8
Optimizer	SGD

Table 14: Model Performance

Metric	Value
Number of Parameters	11,173,962
Test Accuracy	90.92%

Conclusion Adding Dropout layer and weight decay can enhance the performance of the model slightly.

(2) Changing optimizer from SGD to Adam

Table 15: Model Hyperparameters

Hyperparameter	Value
Epoch Number	135
Learning Rate	0.001
Batch Size	8
Optimizer	Adam

Table 16: Model Performance

Metric	Value
Number of Parameters	11,173,962
Test Accuracy	89.66%

Conclusion Adam needs different hyperparameters from SGD.

(3) Set learning rate to 0.0001

Table 17: Model Hyperparameters

Hyperparameter	Value
Epoch Number	100
Learning Rate	0.0001
Batch Size	128
Optimizer	Adam

Table 18: Model Performance

Metric	Value
Number of Parameters	11,173,962
Test Accuracy	91.27%

(4) Reduce the number of residual blocks in each stage to 1 (originally 2)

Table 19: Model Hyperparameters

Hyperparameter	Value
Epoch Number	135
Learning Rate	0.0001
Batch Size	128
Optimizer	Adam

Table 20: Model Performance

Metric	Value
Number of Parameters	4,903,242
Test Accuracy	90.03%

Conclusion The performance become a little bit worse than the original model because the number of parameters of the model decrease.

(5) Use adaptive learning rate

```
scheduler = ReduceLROnPlateau(optimizer, 'min', patience=5,
    factor=0.5)
```

Table 21: Model Hyperparameters

Hyperparameter	Value
Epoch Number	135
Initial Learning Rate	0.0001
Batch Size	128
Optimizer	Adam

Table 22: Model Performance

Metric	Value
Number of Parameters	4,903,242
Test Accuracy	92.09%

Conclusion The performance become a little bit worse than the original model because the number of parameters of the model decrease.

(6) Use depthwise separable convolutions

This included initializing convolutional layers, four residual layers, and a fully connected layer.

Table 23: Detailed ResNet Model Structure

Layer (Type)	Output Size	Layer Configuration
Input	32x32	-
Conv1	32x32	Conv2d(3, 64, 3x3, stride=1, padding=1, bias=False) BatchNorm2d(64), ReLU
Layer1 - ResidualBlock	32x32	
DepthwiseSeparableConv	32x32	Conv2d depthwise (64, 3x3, stride=1, padding=1, groups=64) Conv2d pointwise (64, 64, 1x1) BatchNorm2d(64), ReLU
DepthwiseSeparableConv	32x32	Conv2d depthwise (64, 3x3, stride=1, padding=1, groups=64) Conv2d pointwise (64, 64, 1x1) BatchNorm2d(64), ReLU
Shortcut	32x32	Identity
Layer2 - ResidualBlock	16x16	
DepthwiseSeparableConv	16x16	Conv2d depthwise (64, 3x3, stride=2, padding=1, groups=64) Conv2d pointwise (64, 128, 1x1) BatchNorm2d(128), ReLU
DepthwiseSeparableConv	16x16	Conv2d depthwise (128, 3x3, stride=1, padding=1, groups=128) Conv2d pointwise (128, 128, 1x1) BatchNorm2d(128), ReLU
Shortcut	16x16	Conv2d(64, 128, 1x1, stride=2), BatchNorm2d(128)
Layer3 - ResidualBlock	8x8	
DepthwiseSeparableConv	8x8	Conv2d depthwise (128, 3x3, stride=2, padding=1, groups=128) Conv2d pointwise (128, 256, 1x1) BatchNorm2d(256), ReLU
DepthwiseSeparableConv	8x8	Conv2d depthwise (256, 3x3, stride=1, padding=1, groups=256) Conv2d pointwise (256, 256, 1x1) BatchNorm2d(256), ReLU
Shortcut	8x8	Conv2d(128, 256, 1x1, stride=2), BatchNorm2d(256)
Layer4 - ResidualBlock	4x4	
DepthwiseSeparableConv	4x4	Conv2d depthwise (256, 3x3, stride=2, padding=1, groups=256) Conv2d pointwise (256, 512, 1x1) BatchNorm2d(512), ReLU
DepthwiseSeparableConv	4x4	Conv2d depthwise (512, 3x3, stride=1, padding=1, groups=512) Conv2d pointwise (512, 512, 1x1) BatchNorm2d(512), ReLU
Shortcut	4x4	Conv2d(256, 512, 1x1, stride=2), BatchNorm2d(512)
Average Pooling	1x1	AvgPool2d(4x4)
Fully Connected	10	Linear(512, 10)

Table 24: Model Hyperparameters

Hyperparameter	Value
Epoch Number	135
Learning Rate	0.0001
Batch Size	128
Optimizer	Adam

Table 25: Model Performance

Metric	Value
Number of Parameters	722,186
Test Accuracy	90.68%

Conclusion The performance become a little bit worse than the original model because the number of parameters of the model decrease.

4 Conclusion

In this study, we approached the enhancement of CIFAR-10 image classification through two distinct experimental phases: initially by constructing a custom model from scratch and then by optimizing the ResNet18 architecture.

Custom Model Development and Iterative Improvement:

- **Initial Model:** Started with a basic CNN featuring two convolutional layers. Despite its simplicity, this model achieved a test accuracy of 62.06%.
- **First Improvement:** Increased the number of convolutional layers and adjusted their configurations, which improved the test accuracy significantly to 77.91%.
- **Second Improvement:** Introduced batch normalization and dropout techniques, which further enhanced the model's performance, achieving an accuracy of 82.14%.
- **Third Improvement:** Added self-attention mechanisms to focus on salient features. Surprisingly, this resulted in a slight decrease in accuracy to 81.77%, suggesting a potential for overfitting or inadequate training adaptation to the increased complexity.

Optimizing ResNet18:

- **Adding Dropout and Weight Decay:** Initially, integrating dropout and weight decay slightly improved the performance, boosting accuracy to 90.92%.
- **Changing the Optimizer to Adam:** This change unexpectedly decreased the model's performance to 89.66%, highlighting that Adam might require different parameter tuning compared to SGD.
- **Adjusting Learning Rate and Epochs:** By reducing the learning rate and adjusting the number of epochs, the accuracy improved to 91.27%.
- **Modifying the Number of Residual Blocks:** Reducing the number of residual blocks decreased the model's complexity but also reduced its accuracy to 90.03%.
- **Implementing an Adaptive Learning Rate:** Using a learning rate scheduler improved the performance, culminating in a test accuracy of 92.09%.
- **Employing Depthwise Separable Convolutions:** While this reduced the number of parameters significantly, it slightly dropped the accuracy to 90.04%.

Each experimental phase provided valuable insights into the delicate balance required between model complexity, computational efficiency, and classification accuracy. The initial construction phase emphasized the importance of incremental enhancements, while the ResNet18 optimizations highlighted the critical role of hyperparameter tuning and architectural adjustments in achieving optimal performance. These experiments underscore the potential for tailored architectures and refined training strategies to significantly improve image classification outcomes.