Enhanced Convolutional Neural Networks for Image Classification: Implementation and Extensions

# Introduction

This report presents our implementation and extension of the research paper "Enhanced Convolutional Neural Networks for Improved Image Classification" by Xiaoran Yang, Shuhan Yu, and Wenxi Xu (2023). Our work focuses on reproducing the core methodologies proposed in the original paper and extending them with several architectural enhancements to improve image classification performance. We evaluate these enhancements on both CIFAR-10 and CIFAR-100 datasets, providing a comprehensive analysis of their effectiveness.

# Original Paper Summary

## Overview

The original paper by Yang et al. introduced enhancements to traditional CNN architectures for image classification tasks. The authors proposed a baseline CNN model and demonstrated how specific architectural modifications could lead to significant improvements in classification accuracy. Their work highlighted the importance of feature extraction capabilities and efficient information flow within CNN architectures.

## Key Contributions of the Original Paper

* Development of an enhanced CNN architecture optimized for image classification tasks
* Introduction of effective training strategies to improve model convergence
* Demonstration of state-of-the-art performance on benchmark image classification datasets
* Analysis of the impact of various architectural components on overall model performance

## Baseline Architecture

The original paper proposed a baseline CNN architecture consisting of three convolutional blocks, each containing two convolutional layers followed by batch normalization, ReLU activation, and max pooling. This architecture served as the foundation for our implementation and subsequent enhancements.

# Implementation Methodology

## Datasets

We evaluated our implementations on two standard benchmark datasets:

1. **CIFAR-10**: A dataset consisting of 60,000 32x32 color images across 10 classes, with 50,000 training images and 10,000 test images.
2. **CIFAR-100**: A more challenging extension of CIFAR-10 with 100 classes, maintaining the same image dimensions and train/test split proportions.

Both datasets were preprocessed using standard normalization techniques, and data augmentation methods (random cropping and horizontal flipping) were applied during training to improve generalization.

## Baseline Implementation

We first implemented the baseline CNN architecture as described in the original paper. This implementation served as our control model against which we compared all subsequent enhancements. For CIFAR-100, we achieved a baseline test accuracy of 67.5%, consistent with the original paper's findings when adapting their model to this more challenging dataset.

## Training Details

All models were trained using:

* Cross-entropy loss function
* Adam optimizer with a learning rate of 0.001
* Batch size of 128
* Training for 100 epochs
* Learning rate reduction on plateau
* Early stopping to prevent overfitting

# Our Contributions and Extensions

We implemented several architectural extensions to the baseline model to explore their impact on classification performance:

## Increased Network Depth (5-Block CNN)

We extended the network depth from the original three convolutional blocks to five blocks, maintaining the same structure for each block. This modification aimed to enhance the model's representational capacity, enabling it to learn more complex features from the input data.

**Implementation details:**

* Added two additional convolutional blocks with the same structure as the original blocks
* Adjusted the channel dimensions to gradually increase feature complexity (64→128→256→512→512)
* Maintained consistent dropout rates (0.25) across all blocks

## Graph Convolutional Network Integration (5-Block + GCN)

### **What are Graph Convolutional Networks?**

Graph Convolutional Networks (GCNs) are a class of neural networks designed to operate on graph-structured data. Unlike standard neural networks that process data in Euclidean space (like images or sequences), GCNs can handle data represented as graphs with nodes and edges. The key innovation of GCNs is their ability to learn representations by aggregating information from a node's neighborhood within the graph.

### How GCNs Work?

The core operation in a GCN is the graph convolution, which can be understood as:

1. **Feature Propagation**: Each node aggregates information from its neighbors.
2. **Feature Transformation**: The aggregated information is transformed through learnable parameters.
3. **Non-linear Activation**: A non-linear function is applied to the transformed features.

Mathematically, a simple GCN layer can be expressed as:

H^(l+1) = σ(D^(-1/2) Â D^(-1/2) H^(l) W^(l))

Where:

* H^(l) is the feature matrix at layer l
* Â is the adjacency matrix with self-connections
* D is the degree matrix of Â
* W^(l) is the weight matrix at layer l
* σ is a non-linear activation function

### Relationship to CNNs and Our Implementation

While CNNs excel at capturing local spatial patterns in grid-structured data like images, they don't explicitly model relationships between features in the latent space. GCNs complement CNNs by modeling these feature relationships as a graph after the feature extraction phase.

We introduced Graph Convolutional Network (GCN) layers to enhance feature learning by modeling relationships between features. The GCN layers were incorporated after the CNN feature extraction stage.

**Implementation details:**

class SimpleGCNLayer(nn.Module):

def \_\_init\_\_(self, in\_features, out\_features):

super(SimpleGCNLayer, self).\_\_init\_\_()

self.linear = nn.Linear(in\_features, out\_features)

def forward(self, x):

# x shape: [batch, features]

A = torch.eye(x.size(1), device=x.device) # identity matrix as adjacency

A\_hat = A + torch.eye(x.size(1), device=x.device) # self-connections

D\_hat = torch.diag(A\_hat.sum(1))

D\_hat\_inv\_sqrt = torch.inverse(torch.sqrt(D\_hat))

norm\_A = D\_hat\_inv\_sqrt @ A\_hat @ D\_hat\_inv\_sqrt # normalization

x = torch.matmul(x, norm\_A)

return self.linear(x)

This approach allowed us to leverage graph-based feature learning within our CNN architecture, potentially capturing more complex feature relationships.

Our implementation creates a simple graph structure where features are treated as nodes. The adjacency matrix (initialized as an identity matrix) defines connections between these features. The normalized graph convolution operation then allows features to share information according to the graph structure, potentially capturing more complex feature relationships than traditional fully connected layers.

This CNN-GCN hybrid approach aimed to combine the spatial feature extraction capabilities of CNNs with the relational feature learning of GCNs to improve overall classification performance.

## Squeeze-and-Excitation with Residual Connections (SE + Residuals)

### Squeeze-and-Excitation Networks

Squeeze-and-Excitation (SE) Networks introduce a channel attention mechanism that enhances the representational power of CNNs by explicitly modeling interdependencies between channels. SE blocks adaptively recalibrate channel-wise feature responses by:

1. **Squeeze Operation**: Global information embedding by aggregating feature maps across spatial dimensions using global average pooling, producing a channel descriptor.
2. **Excitation Operation**: Channel-specific weights are generated using a small fully connected network with a bottleneck layer, capturing channel-wise dependencies.
3. **Rescaling**: The original feature maps are rescaled using the generated weights, emphasizing important channels and suppressing less useful ones.

### Residual Connections

Residual connections, introduced in ResNet architectures, address the degradation problem in deep networks by creating shortcut connections that bypass one or more layers:

y = F(x) + x

Where F(x) represents the residual mapping to be learned and x is the identity mapping (input). This approach offers several benefits:

* Mitigates the vanishing gradient problem by providing direct paths for gradient flow
* Makes it easier for the network to learn identity mappings when optimal
* Enables training of much deeper networks with improved performance

We incorporated Squeeze-and-Excitation (SE) blocks along with residual connections to enhance channel-wise feature recalibration and facilitate gradient flow during training.

**Implementation details:**

* SE blocks were added to each convolutional block to perform channel-wise attention
* Residual connections were implemented to improve information flow through the network
* The modified blocks were structured as follows:
  + Input → Conv1 → BN → ReLU → Conv2 → BN → SE → Add(Identity) → ReLU → MaxPool → Dropout

This architecture combines the benefits of attention mechanisms (focusing on important features) with residual learning (preserving feature information and improving gradient flow).

The integration of SE blocks with residual connections creates a powerful architecture that:

1. Enhances feature discrimination by weighting channels according to their importance (via SE blocks)
2. Maintains information flow throughout the network (via residual connections)
3. Allows for effective training of deeper networks with improved gradient flow
4. Balances the focus between spatial features (convolutions) and channel relationships (SE blocks)

This approach aims to improve the network's ability to focus on the most informative features while maintaining stable training dynamics through residual connections.

## Residual Attention Network

### What is a Residual Attention Network?

The Residual Attention Network is an advanced neural network architecture that combines attention mechanisms with residual learning. This approach introduces attention modules that can be stacked to form very deep networks with enhanced feature learning capabilities. The key innovation is the use of a trunk-and-mask attention mechanism that focuses on relevant spatial regions while maintaining the benefits of residual connections.

### How Residual Attention Networks Work

The Residual Attention Network is built around attention modules that consist of two parallel branches:

1. **Trunk Branch**: This branch performs feature processing using standard residual blocks, maintaining the original feature information.
2. **Mask Branch (Softmax Branch)**: This branch generates spatial attention masks through a series of operations:
   * Down-sampling (reducing spatial dimensions)
   * Residual processing (feature refinement)
   * Up-sampling (restoring spatial dimensions)
   * Sigmoid activation (generating attention weights between 0 and 1)
3. **Attention Refinement**: The final output is computed as:

Output = (1 + Mask) ⊙ Trunk

Where ⊙ represents element-wise multiplication. This formulation can be rewritten as:

Output = Trunk + Mask ⊙ Trunk

Which shows the residual connection nature of the attention mechanism.

This architecture offers several advantages:

* The attention masks learn to focus on important spatial regions
* The residual connection ensures that information flows even when attention weights are small
* The structure can be stacked to form very deep networks with stable training dynamics

### Relationship to CNNs and Our Implementation

The Residual Attention Network enhances traditional CNNs by integrating spatial attention mechanisms with residual learning. Unlike standard CNNs that treat all spatial locations equally, this architecture learns to focus on relevant regions through the attention mechanism.

We implemented a Residual Attention Network featuring a trunk-and-mask attention mechanism integrated with residual learning. This approach combines spatial attention with residual connections for enhanced feature learning.

**Implementation details:**

* Trunk branch: Standard residual blocks for feature processing
* Softmax branch: Down-sampling → Residual processing → Up-sampling → Sigmoid activation
* Feature refinement through multiplicative attention: Output = Trunk \* Mask + Trunk

class ResidualAttentionModel(nn.Module):

def \_\_init\_\_(self):

super(ResidualAttentionModel, self).\_\_init\_\_()

self.pre\_layers = nn.Sequential(

nn.Conv2d(3, 64, kernel\_size=3, padding=1, bias=False),

nn.BatchNorm2d(64),

nn.ReLU(inplace=True)

)

self.residual\_block1 = ResidualBlock(64, 128, stride=2)

self.attention\_module1 = AttentionModule(128, 128)

self.residual\_block2 = ResidualBlock(128, 256, stride=2)

self.attention\_module2 = AttentionModule(256, 256)

self.residual\_block3 = ResidualBlock(256, 512, stride=2)

self.attention\_module3 = AttentionModule(512, 512)

self.avg\_pool = nn.AdaptiveAvgPool2d((1, 1))

self.fc = nn.Linear(512, 100)

def forward(self, x):

out = self.pre\_layers(x)

out = self.residual\_block1(out)

out = self.attention\_module1(out)

out = self.residual\_block2(out)

out = self.attention\_module2(out)

out = self.residual\_block3(out)

out = self.attention\_module3(out)

out = self.avg\_pool(out)

out = out.view(out.size(0), -1)

out = self.fc(out)

return out

class AttentionModule(nn.Module):

def \_\_init\_\_(self, in\_channels, out\_channels):

super(AttentionModule, self).\_\_init\_\_()

self.trunk\_branch = nn.Sequential(

ResidualBlock(in\_channels, out\_channels),

ResidualBlock(out\_channels, out\_channels)

)

self.softmax\_branch = nn.Sequential(

nn.MaxPool2d(3, stride=2, padding=1),

ResidualBlock(in\_channels, out\_channels),

nn.MaxPool2d(3, stride=2, padding=1),

ResidualBlock(out\_channels, out\_channels),

nn.Upsample(scale\_factor=2, mode='bilinear', align\_corners=True),

ResidualBlock(out\_channels, out\_channels),

nn.Upsample(scale\_factor=2, mode='bilinear', align\_corners=True),

nn.Sigmoid()

)

def forward(self, x):

trunk = self.trunk\_branch(x)

mask = self.softmax\_branch(x)

out = trunk \* mask + trunk

return out

This architecture creates a sophisticated attention mechanism that can focus on relevant spatial regions while maintaining the benefits of residual connections.

# Experimental Results and Analysis

## Performance on CIFAR-100

Our experimental results on the CIFAR-100 dataset are summarized below:

|  |  |  |  |
| --- | --- | --- | --- |
| Approach | Description | Test Accuracy | Comments |
| Paper's CNN | 1:1 reproduction on harder dataset | 67.5% | Baseline |
| 5-Block CNN | Deeper model | 71.82% | Solid improvement from baseline |
| 5-Block + GCN | Graph-based feature learning | 69.09% | Interesting, but not best performing |
| 5-Block + SE + Residuals | SE attention and skip connections | 72.50% | Stable accuracy, good generalization |
| Attention Modules + Residual Attention | Combined spatial/channel attention + residuals | 75.00% | Best accuracy, some overfitting observed |

## Analysis of Results

**1. Impact of Network Depth:** The extension from 3 to 5 convolutional blocks resulted in a significant 4.32% improvement in accuracy on CIFAR-100 (67.5% → 71.82%). This demonstrates that increased network depth allows for more complex feature hierarchies, which is beneficial for the fine-grained classification required by CIFAR-100's 100 classes.

**2. Effectiveness of GCN Integration:** While the GCN-enhanced model showed improvement over the baseline (69.09% vs. 67.5%), it underperformed compared to the simple 5-block architecture. This suggests that the simple identity-based graph structure may not be optimal for capturing feature relationships in this context. More sophisticated graph structures might yield better results.

**3. Benefits of Attention Mechanisms:** Both attention-based approaches showed significant improvements:

* The SE with Residual connections model achieved 72.50% accuracy, demonstrating the value of channel-wise attention.
* The Residual Attention Network achieved the best performance at 75.00%, highlighting the effectiveness of combining spatial attention with residual learning.

**4. Trade-offs Between Architectures:**

* The Residual Attention Network achieved the highest accuracy but showed signs of overfitting, suggesting a need for stronger regularization.
* The SE + Residuals model offered a good balance between performance improvement and generalization.
* The GCN approach, while theoretically interesting, may require further refinement to realize its full potential.

# Conclusion and Future Work

## Conclusion

Our implementation and extensions of the original paper by Yang et al. demonstrate several effective strategies for enhancing CNN architectures for image classification. The experimental results confirm that:

1. Increased network depth provides substantial benefits for complex classification tasks.
2. Attention mechanisms, especially when combined with residual connections, significantly improve classification accuracy.
3. The combination of spatial attention and residual learning (as in the Residual Attention Network) offers the most substantial performance gains, achieving a 7.5% improvement over the baseline.

These findings extend the original paper's contributions by providing a comparative analysis of different architectural enhancements and their impact on classification performance.

## Future Work

Based on our findings, several promising directions for future work include:

1. **Optimizing GCN Integration**: Exploring more sophisticated graph structures and adjacency matrix definitions to better capture feature relationships.
2. **Regularization Techniques**: Investigating advanced regularization methods to mitigate the overfitting observed in the Residual Attention Network.
3. **Hybrid Architectures**: Combining the most effective elements from different approaches, such as integrating GCN with attention mechanisms.
4. **Transfer Learning**: Evaluating the transferability of the enhanced architectures to other datasets and domains.
5. **Efficiency Optimization**: Analyzing computational efficiency and exploring model compression techniques to enhance real-world applicability.

Our work provides a foundation for further exploration of CNN architectural enhancements for image classification tasks, with the potential to advance the state-of-the-art in this field.

# References

1. Yang, X., Yu, S., & Xu, W. (2023). Enhanced Convolutional Neural Networks for Improved Image Classification.
2. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).
3. Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-excitation networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 7132-7141).
4. Wang, F., Jiang, M., Qian, C., Yang, S., Li, C., Zhang, H., Wang, X., & Tang, X. (2017). Residual attention network for image classification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 3156-3164).
5. Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.