Using ResNet-18 with Squeeze-and-Excitation and Data Augmentation

Pranav Bhatt, Kevin Mai, Riya Garg
https://github.com/maikzero/dl_project_1_public

Abstract

This paper presents a ResNet-inspired convolutional neural network (CNN) designed for image classification tasks under a strict parameter constraint of 5 million parameters. This project leverages ResNet-18, a lightweight convolutional neural network architecture, for image classification on the CIFAR-10 dataset through modifications in architecture to balance performance and computational efficiency. To enhance model generalization and performance, we incorporated advanced techniques such as a warm-up strategy to our learning rate, data augmentation techniques, Squeeze-and-Excitation (SE) blocks, dropout, and Mix-up augmentation. Our final model, with approximately 4.7 million parameters, achieved a test accuracy of 95%.

Introduction

ResNet-18 is one of the most widely adopted deep learning architectures for image classification. Its residual connections help mitigate the vanishing gradient problem, enabling the training of deeper networks. However, for real-world applications like embedded systems and robotics, achieving high accuracy while maintaining model efficiency is crucial. In this project, we aim to optimize ResNet-18 for the CIFAR-10 dataset while adhering to the constraint of utilizing no more than 5 million parameters. Our paper outlines the incorporation of various techniques such as a modified ResNet architecture, a comprehensive data augmentation pipeline to handle distorted images, and SE and dropout.

Methodology

The proposed architecture is designed to balance performance and efficiency while emphasizing generalization. During experimentation, it was found that the Kaggle dataset exhibited some distortions, making generalization crucial for maximizing model performance. The chosen block configuration, [3, 3, 4, 3], was the largest possible under 5 million parameters. Each layer captures different feature sets, with deeper layers learning more complex representations, ultimately improving the model's ability to generalize. We learned that a higher number of parameters resulted in better performance. The degree of regularization and robustness are further enhanced through the integration of dropout, Squeeze-and-Excitation (SE) blocks, and data augmentation techniques such as Mix-up.

Initial Convolution Layer

The network starts with a 3x3 convolution layer (conv1) for the initial feature extraction. The layer consists of: 32 filters, a stride of 1, and padding of 1. Batch Normalization is applied after the convolution operation to normalize activations and reduce internal covariate shift. ReLU activation introduces non-linearity, allowing the network to learn complex patterns.

Residual Blocks

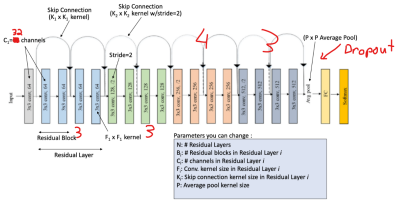

Figure 1: ResNet custom Architecture

Residual blocks form the backbone of the architecture. These blocks, with skip connections, allow gradients to flow more easily through the network, mitigating the vanishing gradient problem and enabling the training of deeper networks. Our ResNet model is inspired by the ResNet-18 architecture. Modifications include a smaller initial filter size of 32 instead of 64 with included SE layer placed near the end of each residual block. There exists one dropout layer at the end of the ResNet. The architecture consists of 4 layers with the following number of blocks:

- Layer 1: 3 residual blocks, 32 filters, stride of 1.
- Layer 2: 3 residual blocks with 64 filters and a stride of 2 (spatial resolution halved).
- Layer 3: 4 residual blocks with 128 filters and a stride of 2 (spatial resolution halved).
- Layer 4: 3 residual blocks with 256 filters and a stride of 2 (spatial resolution halved).

The residual blocks, or otherwise known as Basic Blocks, are composed of a skip connection and a two-convolutional-layer block. The skip connections fix the vanishing gradient problem by reintroducing older information, which allows for deeper networks to function. This configuration balances the depth and efficiency of the model, allowing it to extract complex features while maintaining a manageable number of parameters.

Squeeze-and-Excitation (SE) Blocks

Squeeze-and-Excitation (SE) blocks help the model focus on the most important features. They first squeeze the information from the entire feature map using Global Average Pooling (GAP). Then, two simple layers are used: one with ReLU activation to find patterns and another with Sigmoid activation to create attention weights. These weights help the model give more importance to useful features and reduce the impact of less important ones. SE blocks make the model better at learning important details and improve performance on datasets like CIFAR-10. However, they also increase the number of parameters and can make the model slower to train. In some cases, they might cause overfitting, especially when the dataset is small. Despite these downsides, SE blocks are helpful for boosting overall model performance.

Pooling and Fully Connected Layer

After passing through the residual blocks, the output feature map is pooled using Adaptive Average Pooling to reduce the spatial dimensions to 1x1. This pooling operation helps reduce the computational load and prepares the feature map for the fully connected layer. A Dropout layer follows to regularize the network, preventing overfitting by randomly dropping units (with a dropout probability of 0.5) during training. The final Fully Connected (FC) layer outputs the class probabilities (10 classes for CIFAR-10). The dropout layer before the FC layer ensures that the model generalizes better by reducing overfitting.

Dropout Layer

Dropout is a powerful regularization technique that helps prevent overfitting by randomly disabling 50% of the neurons during training. This forces the model to learn more robust features and prevents overreliance on specific connections. The benefit is improved generalization to new data. However, the drawback is that too much dropout can slow down learning and lead to underfitting, where the model fails to capture important patterns in the data.

Weight Initialization (He Initialization)

He initialization is a method used to set the starting values of the model's weights. It is especially useful when working with ReLU activation functions, as it prevents the problem of gradients becoming too small or too large during training. The weights are chosen from a normal distribution with a mean of 0 and a variance of $\frac{2}{n_{in}}$, where n_{in} is the number of inputs to the layer. This approach helps the model learn faster and improves stability during training. The main benefit of He initialization is that it allows deeper networks to train effectively without the issue of vanishing or exploding gradients. This leads to faster convergence and better performance. However, ResNets are generally not that sensitive to initial weights due to the skip connections inherently implemented by the residual blocks.

Mixup

Mixup is a data augmentation technique where pairs of training samples are combined to create synthetic training examples. This helps regularize the model and improve its robustness, especially in the presence of adversarial examples. The synthetic samples are created by linearly combining the images and their corresponding labels:

$$\text{mixed\_image} = \lambda \cdot \text{image\_a} + (1 - \lambda) \cdot \text{image\_b}$$

where λ is a parameter sampled from a Beta distribution. The labels are mixed in the same proportion, leading to improved model generalization. This approach helps the model generalize better and makes it more resilient to adversarial attacks and noisy data. By smoothing the decision boundaries, it reduces overfitting and improves performance on unseen data. However, Mixup can sometimes confuse the model, especially when the dataset has distinct and well-defined features, leading to slower learning and reduced accuracy in certain cases.

Optimizer and Hyper-Parameters

The optimizer used was Stochastic Gradient Descent with momentum. Our hyper-parameters include: learning rate, momentum, weight decay, epochs, batch size, as well as the various values for the data augmentation pipeline. The final values for our hyper-parameters were 0.02, 0.9, 0.0001, 300, and 64 for learning rate, momentum, weight decay, and batch size, respectively.

Learning Rate Scheduling (Warm-up)

Learning Rate Scheduling is utilized in the model to help the convergence of the model. It utilizes a linear warm-up period of 10 epochs where it linearly grows to our chosen learning rate of 0.02, which then follows cosine annealing. The initial small learning rate helps to stabilize the random weight initialization and prevents large disruptive updates. The model adapts more smoothly to the training data, which results in better convergence. Cosine annealing helps the model in escaping local minima by both increasing and decreasing the learning rate, resulting in both better convergence and generalization. The drawbacks of this approach are higher complexity and a longer training time.

Batch Size

Using a batch size of 64 strikes a balance between training stability and computational efficiency for the CIFAR-10 dataset. It allows the model to learn effectively without consuming excessive memory. The advantage of this batch size is that it smooths out gradient updates, leading to more stable convergence. However, the downside is that it may

miss some finer details in the data compared to smaller batch sizes, which can provide more variability and help the model generalize better. Additionally, larger batch sizes could speed up training, but they often require more memory and can lead to poor generalization.

Data Preprocessing Pipeline

The CIFAR-10 dataset includes 60,000 images with 10 classes, which we use to train our model. To improve the robustness of the model, we include a variety of transformations on the dataset. This includes random cropping, blurring and sharpness, rotations, flips, random erasing, auto augmentations, and color distortions. The dataset is also normalized using the standard normalization values from the ImageNet dataset. The pipeline helps to improve the diversity of the dataset, preventing overfitting.

Evaluation and Metrics

During each epoch, the model's performance is evaluated on the validation set. The following metrics are tracked: loss and accuracy. The loss function used is cross-entropy. Furthermore, at the end of training, the model is evaluated on the testing dataset, and its accuracy is recorded.

Model Checkpointing

Model checkpoints are saved at regular intervals during training. The best-performing model (based on validation accuracy) is selected for final testing. The checkpoint saves include the model's state dictionary, the optimizer's state dictionary, and loss/accuracy metrics for the current epoch.

Results Discussion

Our final model's performance exhibits the best accuracy on the Kaggle dataset of about 86%. The model validation accuracy was a value of 90.69%, with a test accuracy of 95.94%. The test accuracy is much higher because the transformation pipeline only included normalization, while the validation set went through much more diverse changes. The performance of the model on the accuracy of the training was significantly worse at a value of 73 41%. A low training accuracy could be an indication that the model has not fully learned the features of the dataset. Further improvements could be made, such as a longer training time, as the graphs indicate that the accuracy has not yet fully converged.

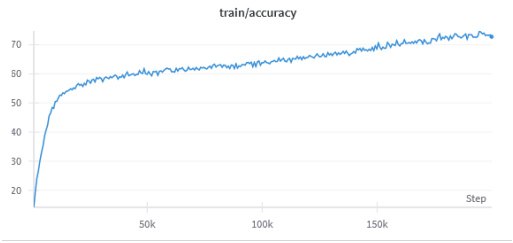

Figure 2: Training accuracy over epochs on the Kaggle dataset.

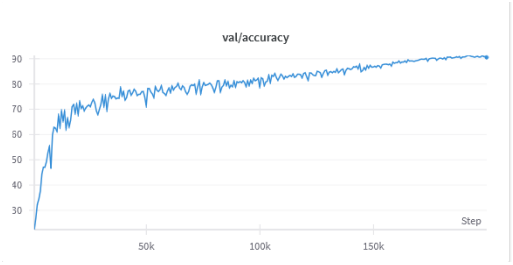

Figure 3: Loss

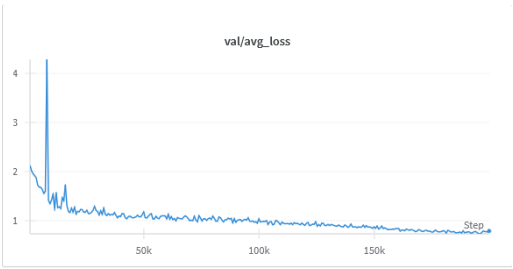

Figure 4: Validation accuracy over steps of final model

Lessons Learned

A crucial lesson learned is increasing the robustness of our model. The quality of the training dataset became increasingly important as we discovered the distortions in the Kaggle dataset. Our solution was an augmentation of the original dataset. Through modifications of the dataset, the model was able to learn general features of each class. Each iteration of the model was trained on increasingly more augmentation. The model started off with cropping and horizontal rotations. Gradually, we proceeded to further distortions such as erasure, color distortions, sharpness and blurring. Finally, we incorporated Mixup, which further enhanced the model's performance. Each training iteration leveraged progressively more augmentation, leading to a more resilient model.

References

1. He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385.
2. Hu, J.; Shen, L.; Albanie, S.; Sun, G.; and Wu, E. 2019. Squeeze-and-Excitation Networks. arXiv:1709.01507.
3. Thakur, A.; Chauhan, H.; and Gupta, N. 2023. Efficient ResNets: Residual Network Design. arXiv preprint arXiv:2306.12100.
4. Zhang, H.; Cisse, M.; Dauphin, Y. N.; and Lopez-Paz, D. 2017. mixup: Beyond empirical risk minimization. arXiv preprint arXiv:1710.09412.