Project 1 - CIFAR 10 Image Classification using ResNet

Abhishek Adinarayanappa (aa12037), Harsha Mupparaju (sm12754), Nived Damodaran (nd2746)

GitHub Repository Link: https://github.com/Zeudon/Cifar_10_Image_Classification

Abstract

This study explores the implementation of a ResNet-based convolutional neural network for image classification, leveraging residual learning to improve feature propagation and gradient flow. The model is designed with three residual layers, each composed of multiple BuildingBlocks, which incorporate skip connections to mitigate the vanishing gradient problem. The network is trained and evaluated on a dataset preprocessed with an extensive data augmentation pipeline, including random cropping, perspective distortion, rotation, horizontal flipping, and color jittering. These transformations enhance generalization by introducing natural variations in training samples.

The model achieves high training (97.68%) and validation (93.11%) accuracies, demonstrating strong learning capabilities. However, the test accuracy drops to 82.16%, indicating potential overfitting to the training distribution. This discrepancy suggests that the model may have learned dataset-specific patterns that do not generalize well to unseen data. Despite applying normalization based on CIFAR-10 statistics, further regularization strategies—such as increased dropout, weight decay, or more aggressive data augmentation—may be required to enhance generalization. The findings highlight the importance of balancing model complexity and generalization to optimize performance across unseen datasets.

Methodologies

Our methodology involved designing and refining the resnet-18 architecture to effectively classify images in our CIFAR 10 dataset. We explored various model architectures, optimizers, and data augmentation strategies to achieve optimal performance.

Model Design and Architecture Choices

Our residual block implements:

$$ReLU(S(x) + F(x))$$

where $S(x)$ is the skipped connection and $F(x)$ is a block that implements:

$$Conv \rightarrow\ BN \rightarrow\ ReLU \rightarrow\ Conv \rightarrow\ BN$$

Our initial experiments included configurations with residual block arrangements of (1,1,1,1) with 64, 128, 256, and 512 channels; (3,3,3) with 64, 128, and 256 channels; and (2,1,1,1) with 64, 128, 256, and 512 channels. Additionally, we tested narrower architectures that began with 16 or 32 channels and scaled up to 128 or 256 channels. However, these smaller models delivered lower accuracy and utilized far fewer parameters than the 5 million parameter ceiling.

Ultimately, we found that a structure featuring four residual blocks in the first two layers and three residual blocks in the final layer — i.e., the (4,4,3) configuration with 64, 128, and 256 channels — achieved the highest accuracy and proved to be the most effective design. This configuration utilized approximately 4.7 million parameters, effectively balancing model complexity and performance. Its increased depth and width enabled the network to extract robust features while efficiently utilizing the parameters. The deeper architecture effectively captured complex patterns in the dataset while avoiding overfitting.

Training Strategy and Optimizer Selection

During training, we evaluated multiple optimizers, including Adam, RMSprop, and SGD. Although Adam and RMSprop provided faster convergence during early epochs, they struggled to generalize well on the unseen test data. Ultimately, SGD with momentum proved to be the most effective, yielding superior test accuracy and more stable convergence. SGD's consistent updates with momentum helped the network converge efficiently without overfitting.

Additionally, we experimented with StepLR, MultiStepLR, and ReduceLROnPlateau as learning rate schedulers. Among these, MultiStepLR performed the best, effectively balancing convergence speed and generalization when training for approximately 100 epochs. This combination of SGD with momentum and MultiStepLR significantly improved test accuracy and ensured stable learning throughout the training process.

Data Augmentation Techniques

A key observation during our experiments was the significant impact of data augmentation. Since the test data distribution appeared slightly different from the training and validation datasets, effective augmentation was critical. We applied the following augmentations to enhance model generalization:

- Random cropping and resizing to ensure spatial invariance and introduce diversity in object scales.
- Random perspective distortion and small rotations to improve robustness against viewpoint changes.
- Horizontal flipping to introduce additional variance in object orientation.
- Brightness, contrast, and saturation adjustments to increase robustness to lighting variations.
- Sharpness adjustments to improve texture representation.

Additionally, we experimented with Gaussian blurring to smoothen the highly pixelated training images, but this did not result in a significant improvement in accuracy.

Furthermore, all images across the training, validation, and test sets were normalized using the mean and standard deviation values of the CIFAR-10 dataset. This normalization step ensured consistent data distribution and helped the model generalize effectively rather than overfitting to the CIFAR-10 dataset's specific patterns.

Therefore, augmentation techniques, coupled with proper normalization, significantly boosted our model's test accuracy from approximately 75% to 82%.

Observations and Key Lessons Learned

Through extensive experimentation, we identified the following insights:

- Deeper networks like the (4,4,3) architecture consistently outperformed shallower designs, suggesting that our dataset benefited from additional layers to capture complex patterns.
- Data augmentation significantly improved generalization performance, reinforcing the need to simulate real-world data variations.
- SGD with momentum and MultiStepLR scheduler offered the most stable and effective convergence behavior, despite slower initial learning.

By combining these findings, we successfully developed a model that achieved strong performance on the dataset while maintaining robust generalization to unseen data.

Results

The ResNet-based model, constructed with three residual layers consisting of 4, 4, and 3 residual blocks (Figure 1), was trained for 100 epochs. Each residual block contains two convolutional layers with ReLU activations and batch normalization. The total number of trainable parameters in this architecture is 4,697,162. The model achieved the following performance metrics:

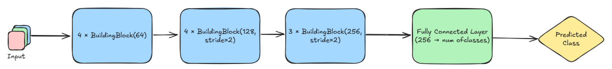
Input
4 × BuildingBlock(64)
4 × BuildingBlock(128, stride=2)
3 × BuildingBlock(256, stride=2)
Fully Connected Layer (256 → num ofclasses)
Predicted Class

Figure 1: Final Model Architecture Block Diagram.

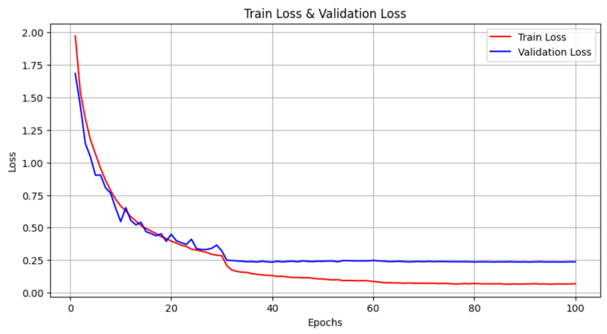

Figure 2: Train and Validation Loss.

- Training Accuracy: 97.68%
- Validation Accuracy: 93.11%
- Final Training Loss: 0.06
- Final Validation Loss: 0.237
- Test Accuracy: 82.162%

Training and Validation Loss Analysis

The training and validation loss curves (Figure 2) show a steady decline over the epochs, converging to low final values (0.06 and 0.237, respectively). The loss starts relatively high, but drops significantly within the first 20 epochs, indicating rapid learning. The smooth decline suggests effective optimization with minimal oscillations, and no evidence of severe overfitting during training. However, the small but noticeable gap between training and validation loss towards the end suggests minor overfitting, which may explain the lower test accuracy (82.16%) compared to validation accuracy.

Train and Validation Accuracy Analysis

The training and validation accuracy curves (Figure 3) illustrate that the model reaches a high accuracy quickly, sur-

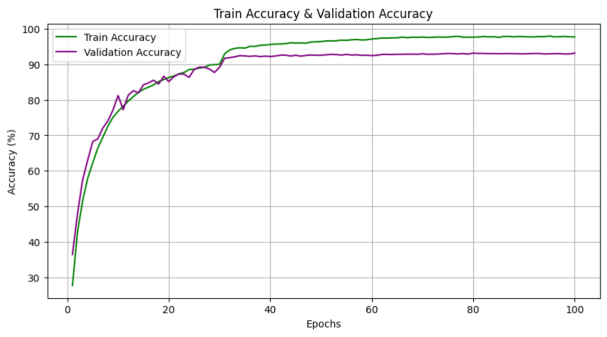

Figure 3: Train and Validation Accuracy.

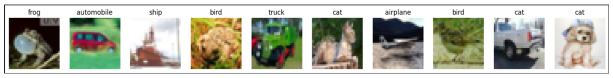

Figure 4: Sample Outputs.

passing 80% within the first 20 epochs. The final training accuracy is 97.68%, while validation accuracy stabilizes at 93.11%. The small gap between the two suggests that while overfitting is present, it is not extreme. However, the test accuracy of 82.16% suggests that the model's generalization to completely unseen data is limited, potentially due to dataset-specific learning.

Potential Improvements

To address the observed generalization gap, the following strategies could be considered:

- Stronger regularization techniques (e.g., dropout layers or increased weight decay).
- More aggressive data augmentation, including random erasing or mixup to encourage feature robustness.
- Fine-tuning hyperparameters, such as learning rate schedules or batch size adjustments.

References

https://github.com/kuangliu/pytorch-cifar

https://pytorch.org/vision/main/transforms.html