Convolutional Neural Networks for Image Classification on CIFAR-10 and TinyImageNet

Abstract

This report presents a detailed study on applying Convolutional Neural Network (CNN) architectures for image classification using the CIFAR-10 dataset. For the study, we used a custom SimpleCNN and ResNet18 architectures. The aforementioned were implemented using PyTorch. In addition to standard cross-entropy loss, this report discusses the incorporation of label smoothing as a regularization method to reduce overfitting, thus improving generalization. We further optimized training using Adam and Stochastic Gradient Descent (SGD), weight decay, dropout regularization, and Learning Rate Scheduling (Cosine Annealing LR). We experimented in a comprehensive manner by tuning hyper-parameters across varied configurations. The results are represented via standard metrics, such as training/validation loss and accuracy, focusing on determining the impact of what architecture and regularization strategy was chosen. Our experiments revealed that ResNet18 with Adam optimizer and learning rate of 0.01 achieved the highest accuracy (81.28%), while the SimpleCNN architecture demonstrated competitive performance with appropriate hyperparameter settings, particularly with SGD at higher learning rates. We also found that batch size had minimal impact on performance, and standard data augmentation consistently outperformed advanced techniques across both architectures. This report provides a systematic account of the architecture choice and design, training procedure, and the outcomes, offering insights for CNN design for image classification tasks.

Introduction

Image classification is a primordial task in computer vision with lots of challenges. one of the most active and challenging problems in computer vision. With the rise of deep learning, convolutional neural networks (CNNs) have become pretty much the way to go for most of these tasks. In this report, we investigate several CNN architectures applied to the widely used CIFAR-10 dataset. The aforementioned dataset is a very established dataset in machine learning composed of 60,000 32 x 32 images divided into 10 classes, offering a very manageable testbed when wanting to test different algorithms. It was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

Recent advances in CNN architecture have demonstrated the benefits of using deeper and more complex networks such as ResNet. The design of a basic Convolutional Neural Network (CNN) starts with an Image Input Layer (dimensions: height, width, channels), subsequently convolved with a combination of some convolutional filters, max or average pooling layers followed by a fully connected layer and in the end, a Softmax or a Sigmoid output function for class label prediction. The channels of subsequent layers increase depending on the number of filters used in convolutional layers. This generic method of designing a CNN is common to every architecture, however the number and fashion of arrangement of these layers is what makes them different from each other in terms of performance and efficiency. (Sourced from: https://towardsdatascience.com/resnets-why-do-they-perform-better-than-classic-convnets-conceptual-analysis-6a9c82e06e53/). As we want to delve deeper however, a problem arises. Theoretically, the training and testing error should decrease as the network goes deeper. In contrast, after the network attains a minimum value, the error rate starts to rise again. Kind of counter-intuitive don't you think? This is because the model starts to learn the noise and other specifics of the training data which causes overfitting. This unique problem makes way for the Vanishing Gradient Problem. The Vanishing Gradient Problem is a phenomenon that occurs during the training of deep neural networks, such as CNN's, where the gradients that are used to update the network become extremely small, hence vanishing (as the name suggests) as they are back propagated from the output layers to the earlier layers. When using the chain rule, we must keep multiplying terms with the error gradient as we progress (or digress we should say). The issue is that if one does a lot of these multiplications with many values that are less than one, the resulting gradient will be very small. So as mentioned before, the gradient will become smaller and smaller (or even zero) nearing the early layers. In the event that it ends up being zero, it does not update the layers at all. (Source: https://chautuankien.medium.com/skip-connection-and-explanation-of-resnet-afabe792346c). ResNet fixes the issue with something called skip connection (Shortcut connections). The aforementioned allows the signal to skip one or more layers and just cut directly to the output of the network. This skip connection also helps to preserve the high level features that were learned by the network. Essentially, the main advantage and usefulness of skip connection is that if there is a layer that hurts the performance of the network, with regularisation it will be skipped. For this experiment we included a SimpleCNN with the intent of balancing model complexity and performance, and also as a baseline. We will be comparing the performance with ResNet18 for the image classification task on the CIFAR-10 Dataset. Usually, ResNets, especially ResNet18 are used for these kinds of tasks due to their accuracy and effectiveness. This holds especially true for the dataset we are using in this experiment. Due to this, we expect ResNet to have a robust performance in this task when compared to a simple CNN.

Method

Network Architectures

The code base offers two architectures:

SimpleCNN

The SimpleCNN model is a custom CNN designed to balance complexity and performance. It consists of three convolutional layers:

- **Layer 1:** Converts the 3-channel input to 32 channels using a 3×3 kernel with padding, followed by a ReLU activation.
- **Layer 2:** Expands the feature map to 64 channels, again using a 3×3 kernel and ReLU, followed by max-pooling.
- **Layer 3:** Further expands the feature map to 128 channels, applying a 3×3 convolution and ReLU, with a subsequent max-pooling operation.

A dropout layer is applied after the convolutional layers to mitigate overfitting, followed by two fully connected layers:

- **FC1:** Maps the flattened feature map (of size 128×4×4) to 256 hidden units with ReLU activation.
- **FC2:** Produces the final output logits corresponding to the number of classes (10 for CIFAR-10).

The dropout probability is configurable via the dropout_rate parameter. In general the architecture is designed to be very modular in order to allow for additional modifications that may be necessary.

ResNet18

ResNet18 is a variant of ResNet, a deep network that fixes the vanishing gradient problem through skip connections. The number represents how many layers it has.

Loss Function and Regularization

Label Smoothing

The report incorporates a custom loss function, LabelSmoothingCrossEntropy, which implements label smoothing. Label smoothing is a regularization technique that prevents the network from becoming too confident by softening the target labels. This approach reduces overfitting and may improve generalization by distributing the target probabilities to all other classes. If label smoothing is not applied, cross-entropy loss is used.

Data Preparation and Augmentation

CIFAR-10

The CIFAR-10 dataset is loaded using PyTorch’s torchvision.datasets module. Two sets of transformations are applied:

- **Advanced Augmentation:** When used (via advanced_augment), RandAugment is applied. This strategy automatically selects augmentation policies in order to improve the diversity of images during training.
- **Standard Augmentation:** This entails random cropping with padding and flipping horizontally.

Training and testing images are normalized using values (mean and standard deviation) from the CIFAR-10 dataset.

Training Procedure

The training process is as follows:

- **Device Selection:** The code checks for the availability of a CUDA-enabled GPU and uses it if available, otherwise defaults to CPU.
- **Optimizer Selection:** Depending on the user’s choice, the optimizer is selected between Adam and SGD. Both optimizers are configured with weight decay (L2 regularization) to reduce overfitting. Adam computes the decaying average of past squared gradients and past gradients. Stochastic Gradient Descent updates all the parameters for each training example and the label individually.
- **Learning Rate Scheduling:** A CosineAnnealingLR scheduler is used to adjust the learning rate over epochs, with a smoother convergence in mind.
- **Epoch Loop:** For each epoch, the training loop iterates through the training dataset, calculates the loss using the chosen criterion, and updates model weights. Validation is performed on the test set after each epoch, and the metrics of interest (loss and accuracy) are then recorded.

Metrics and Visualization

During training, the following metrics are tracked:

- **Training Loss:** The average loss over the training set.
- **Validation Loss and Accuracy:** They are computed on the test/validation set at the end of every epoch.

Training progress is showcased with plots that are generated using Matplotlib

- **Loss Curves:** A comparison of training and validation losses over epochs.

- **Accuracy Curve:** A plot showcasing the evolution of validation accuracy over epochs.

Experiments and Results

Experiment Setup

We conducted our experiments using PyTorch on the CIFAR-10 dataset, which consists of 60,000 32×32 color images across 10 classes (50,000 for training, 10,000 for testing). All models were trained for 15 epochs to balance computation time with sufficient learning. We systematically varied hyperparameters across 52 different configurations to thoroughly investigate the impact of architecture choices and training strategies.

Architecture Performance Comparison

Both SimpleCNN and ResNet18 achieved a maximum validation accuracy of approximately 81% (Figure 6). These findings were somewhat surprising, as it demonstrated that a well-designed simple architecture with appropriate hyperparameters can rival more complex models on the CIFAR-10 dataset.

The learning curves in Figure 7 provide additional insights:

- ResNet18 typically displayed faster initial convergence than SimpleCNN
- ResNet18 showed more stable learning trajectories, particularly with the Adam optimizer
- Both architectures showed signs of continued improvement at epoch 15, suggesting longer training could yield further gains

Optimizer Effects

As shown in Figure 2, both Adam and SGD optimizers achieved similar peak performances when properly tuned. However, they exhibited architecture-dependent behaviors:

- For ResNet18, Adam consistently outperformed SGD across most configurations
- For SimpleCNN, SGD performed exceptionally well with higher learning rates (0.01)

This interaction between architecture and optimizer highlight the importance of jointly considering these choices rather than selecting them independently.

Learning Rate Impact

Learning rate emerged as one of the most influential hyperparameters, with different optimal values depending on the architecture-optimizer combination (Figure 5):

- ResNet18 with Adam reached its peak performance (81.28% accuracy) with a learning rate of 0.01
- SimpleCNN with SGD also excelled with a 0.01 learning rate, achieving 80.54% accuracy
- SimpleCNN with Adam performed better with a lower learning rate of 0.001 (80.90% accuracy)

These results demonstrate the critical importance of learning rate tuning and its interaction with both architecture and optimizer selection.

Batch Size Analysis

Interestingly, our results indicated that batch size had minimal impact on final model performance (Figure 4). Across the tested batch sizes (64, 128, 256), validation accuracies remained relatively consistent for both architectures. This suggests that for CIFAR-10, practitioners can select batch sizes based on computational constraints without significantly compromising accuracy.

Data Augmentation Effectiveness

Contrary to our initial expectations, standard data augmentation techniques (random cropping and horizontal flipping) consistently outperformed the more advanced RandAugment approach across both architectures (Figure 3). This suggests that:

1. Simpler augmentation techniques may be sufficient for CIFAR-10
2. More advanced techniques might require longer training periods to realize their benefits
3. Parameter tuning for advanced augmentation might be necessary

Learning Rate Scheduling Impact

The effect of cosine annealing learning rate scheduling was relatively modest (Figure 1), with no significant difference in the final accuracy for either architecture. However, the learning curves (Figure 7) show that scheduling contributed to smoother convergence, particularly in later epochs.

Best Performing Configurations

The top-performing configurations from our experiments were:

1. **ResNet18 + Adam + LR=0.01**: 81.28% validation accuracy
2. **SimpleCNN + Adam + LR=0.001**: 80.90% validation accuracy
3. **SimpleCNN + SGD + LR=0.01**: 80.54% validation accuracy

These results demonstrate that while ResNet18 achieved the highest peak performance, SimpleCNN could approach comparable performance with appropriate hyperparameter settings. This finding is valuable for deployment scenarios with computational constraints where simpler architectures might be preferred.

Conclusion

Our comprehensive study of CNN architectures and hyperparameters for image classification on CIFAR-10 produced several significant findings:

1. **Architecture Choice**: While ResNet18 achieved the highest overall accuracy (81.28%), our custom SimpleCNN demonstrated surprisingly competitive performance (up to 80.90%) with appropriate hyperparameter settings. This suggests that for relatively simple datasets like CIFAR-10, properly tuned simpler architectures can be viable alternatives to more complex models, especially when computational resources are limited.
2. **Optimizer-Architecture Interaction**: We found that the choice of optimizer should be made in conjunction with the network architecture. ResNet18 performed best with Adam, while SimpleCNN showed excellent performance with SGD at higher learning rates. This interaction highlights the importance of jointly optimizing these choices rather than treating them as independent decisions.
3. **Learning Rate Significance**: Learning rate emerged as the most influential hyperparameter, with its optimal value strongly dependent on the architecture-optimizer combination. This reinforces the critical importance of learning rate tuning in CNN training.
4. **Batch Size Flexibility**: The minimal impact of batch size on final performance suggests that practitioners can select batch sizes based on computational constraints rather than accuracy concerns for datasets like CIFAR-10.
5. **Augmentation Strategy**: Simpler augmentation techniques outperformed more complex ones in our experimental setup. This indicates that advanced augmentation methods may require longer training periods or more careful parameter tuning to realize their benefits.

These findings have several practical implications for CNN design and implementation:

- For datasets of moderate complexity like CIFAR-10, starting with simpler architectures and proper hyperparameter tuning can be a cost-effective approach
- The architecture-optimizer-learning rate relationship should be considered holistically rather than in isolation
- The diminishing returns from advanced techniques (complex augmentation, scheduling) suggest that computational resources might be better allocated to more extensive hyperparameter searches with simpler methods

Future work could explore:

- Longer training durations to determine whether the observed trends persist
- Transfer of these findings to more complex datasets like TinyImageNet or ImageNet
- Additional architectures such as EfficientNet or MobileNetV2 that might offer better efficiency-performance trade-offs
- More advanced regularization techniques like MixUp or CutMix

Our study demonstrates the importance of systematic hyperparameter optimization when designing CNNs for image classification tasks and provides practical insights for practitioners seeking to balance performance with computational efficiency.

Sources

https://viso.ai/deep-learning/resnet-residual-neural-network/#:~:text=The%20Importance%20Of%20ResNet%20in%20Computer%20Vision,-Deep%20Neural%20Networks&text=These%20additional%20layers%20help%20solve,to%20get%20highly%20accurate%20results.

https://www.geeksforgeeks.org/image-classifier-using-cnn/

https://chautuankien.medium.com/skip-connection-and-explanation-of-resnet-afabe792346c

https://www.engati.com/glossary/vanishing-gradient-problem#:~:text=exploding%20gradient%20problem%3F-,What%20is%20vanishing%20gradient%20problem%3F,layers%20to%20the%20earlier%20layers.

https://www.analyticsvidhya.com/blog/2021/08/all-you-need-to-know-about-skip-connections/

https://towardsdatascience.com/resnets-why-do-they-perform-better-than-classic-convnets-conceptual-analysis-6a9c82e06e53/

https://debuggercafe.com/implementing-resnet18-in-pytorch-from-scratch/

https://medium.com/analytics-vidhya/resnet-understand-and-implement-from-scratch-d0eb9725e0db

https://www.geeksforgeeks.org/resnet18-from-scratch-using-pytorch/

https://medium.com/data-science/what-is-label-smoothing-108debd7ef06

https://medium.com/mdr-inc/from-sgd-to-adam-c9fce513c4bb