Deep Learning Mini-Project
ResNet for CIFAR-10

Team GradeIsAllYouNeed

Ansh Sarkar[1], Princy Doshi[2], Simran Kucheria[3]

[1]as20363@nyu.edu, [2]pd2672@nyu.edu, [3]sk11645@nyu.edu

New York University

Abstract

The project aims to train/test a ResNet model on the CIFAR10 dataset with less than 5 million parameters. There were other constraints as well such as no pre-trained models allowed.

We created and trained multiple models that are variants of the implementation of the ResNet model on the CIFAR10 dataset to classify images. Our best model achieves a test accuracy of 96.95% with 4,992,586 parameters.

Implementation can be found here: https://github.com/anshsarkar/ECE-GY-7123-Deep-Learning-Project-1

Overview

A residual network (ResNet) architecture is any convolutional network with skipped connections.
The key component in ResNet models is a residual block that implements:

$$ReLu(S(x) + F(x)) \tag{1}$$

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

$$conv \rightarrow BN \rightarrow relu \rightarrow conv \rightarrow BN \tag{2}$$

where "BN" stands for batch normalization.

Chaining such blocks layer by layer gives a deep ResNet. Hyperparameters (design variables) in such architectures include:

- Ci − the number of channels in the ith layer.
- Fi − the filter size in the ith layer
- Ki − the kernel size in the ith skip connection
- P − the pool size in the average pool layer.

The model is a variant of the ResNet model that consists of blocks called BasicBlocks. Each BasicBlock has two convolutional layers with batch normalization and ReLU activation. It consists of multiple layers of these blocks, with each of these block layers having the same channel size.

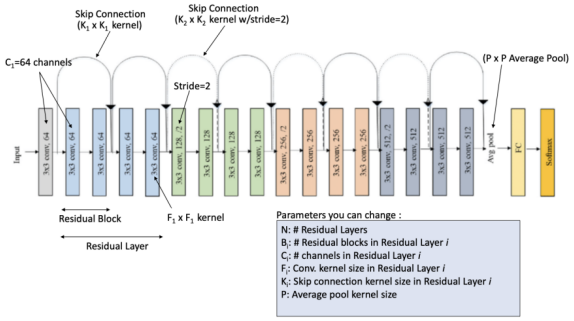

Figure 1: Standard ResNet Architecture

As such we decided to experiment with a variety of techniques explained in later sections of the report to come to our final configurations.

The final configurations of our model are:

- Number of layers: 3
- Blocks in each layer: [4,5,3]
- Input channels to each layer: [64,128,256]
- Filter Size: 3x3
- Kernel Size: 1x1
- Average pooling size: 8

With the following hyperparameters:

- Batch Size: 128
- Optimizer: Lookahead + SGD
- Learning Rate: 0.1
- Momentum: 0.9
- Weight Decay: 0.0005
- Annealing Cosine: 200 epochs
- Learning Rate Decay: by 0.1
- Total Epochs: 200
- Lookahead Alpha: 0.5

combined with other advanced techniques, resulting in a train accuracy of 97.65% and a validation accuracy of 96.95%
Apart from changing the model architecture, we also experimented with data augmentation and generalization techniques that are explained further in the next section.

Methodology

Our methodology focused on optimizing model training through architecture exploration, data augmentation, and regularization.
To perform experiments we divided the dataset into training and validation datasets using an 84:16 split with the unlabeled data hosted on Kaggle acting as our test set.
Initially, we experimented with various architectures under similar hyperparameters and data augmentations, identifying a few that performed well on the test dataset.
We then tested different augmentation techniques, including *Random Cropping* and *Random Horizontal Flip*, and introduced a learning rate scheduler to address stagnation in training accuracy.
Analysis of misclassified samples revealed generalization issues, prompting the use of regularization methods like weight decay, label smoothing, and dropout.
Additionally, advanced techniques such as *Cutout* and *Lookahead* were employed, further improving generalization and model performance.

Architectural Changes

We experimented with 3-layer and 4-layer architectures with block ranges that started from 1 to 4. Also experimented with different channel sizes ranging from 64 to 512. Tabulating some of the more noteworthy results here.

Model Architectures			
Channel Sizes	**Blocks**	**Val. Accuracy (%)**	**No. of Params**
(64,128,256,512)	(1,1,1,1)	89.8	4,903,242
(64,128,230,270)	(2,2,2,2)	95.7	4,995,102
(64,128,192,230)	(4,3,2,1)	**96.03**	**4,236,302**
(64,128,256)	(4,5,3)	**96.95**	4,992,586

Table 1: Comparison of Model Architectures based on Channel Sizes, Block Configuration, Validation Accuracy, and Parameter Count

The results of the experiment showed that an increase in the number of layers may not necessarily lead to an overall increase in the test or validation accuracy.
Hence we decided to go ahead with a 3 layer architecture.

Data Augmentations

We experimented with a set of data augmentation techniques to enhance model performance. These included *Random Crop*, which randomly crops a region from the image with padding to introduce spatial variability, and *Random Horizontal Flip*, which flips the image horizontally with a given probability to increase diversity. We also applied *Auto Augment*, an automated augmentation policy tailored for CIFAR-10 to optimize performance.
The implementation of *Color-Jitter* and *Random-Rotation* transformations did not yield any notable improvement in validation accuracy. Instead, the accuracy plateaued and the loss curves became less discernible, suggesting that these specific augmentations were not helping the model converge and that model training was not getting stable.

In addition to our general transforms and data augmentation techniques, we also explored some of the advanced techniques that might help the model to generalize better:

- **Mixup**: is a data augmentation technique that enhances model generalization by blending pairs of training samples and their labels. By interpolating feature vectors and labels, it reduces overfitting and improves performance across various machine-learning tasks.
  We tested different α values, including 0.5 and 0.75, which initially improved our model. However, after incorporating additional techniques, we ultimately decided not to include Mixup in our final implementation.
- **Cutout**: was another technique that masks regions and could help the model generalize better. This technique was helpful to us in our implementation. By masking random square regions in the input, this technique compels the model to focus on broader contextual patterns, thereby enhancing its ability to generalize. This approach consistently achieved the highest validation accuracy among all methods tested, underscoring its strength as a powerful regularization tool.
- **Cutout** and **MixUp**: Combining the two techniques was not very beneficial for us. After some epochs, there was no improvement in train and validation accuracies suggesting that maybe it was too complex for the model to learn.

Despite the potential of these methods, neither **MixUp** nor the **MixUp-Cutout** combination yielded the significant improvements we had anticipated. The results were inconsistent and failed to consistently exceed the performance of our baseline model. For our final implementation, we went only with Cutout in addition to other techniques.

Data Augmentations	
Augmentation	**Val. Accuracy (%)**
Random Crop + Random Horizontal Flip + AutoAugment (1)	95.42
(1) + Cutout	**95.94**
(1) + ColourJitter + Random Roatations	94.9

Table 2: Comparison of Data Augmentation Techniques on the Basis of Validation Accuracy

Optimisers/Regularisers

To better optimize the performance of the model, in terms of Optimisations and Regularisation techniques, we experimented with a variety of optimisers including *SGD*, *Adam*, *Lookahead* whilst also employing learning rate schedulers such as cosine annealing. We also incorporate regularization

methods like *Dropout*, *L2 regularization*, and *label smoothing*.
We tested these strategies to find the best combination that would improve training stability, enhance the model's ability to generalize, and ultimately boost its overall performance.
Doing a basic comparison between SGD and ADAM, we found that SGD outperformed ADAM during initial training, so we decided to go ahead with SGD. This correlates with existing media where SGD converges better for image classification tasks.
Other than these steps we also tried advanced regularization techniques that could help our model generalize better:

- **Lookahead**:
  We used the **Lookahead** optimization algorithm to improve the stability and convergence speed of our ResNet model on the CIFAR-10 dataset. Lookahead works by maintaining two sets of weights: fast weights, which are updated frequently using Adam, and slow weights, which are updated every k iteration by moving towards the best-performing fast weights. This method helped stabilize training and speed up convergence. We set $k = 5$ and $\alpha = 0.5$ to balance exploration and stability. Our results showed better training stability and strong test performance, demonstrating Lookahead's effectiveness in optimizing complex models.

- **Cosine Annealing LR**: The **CosineAnnealingLR** scheduler was implemented to dynamically adjust the learning rate over epochs. We used a starting learning rate of 0.1. This scheduler was applied to the **SGD** optimizer wrapped with the **Lookahead** algorithm, ensuring stable learning rate adjustments throughout the training process.

- **Dropouts**: Additionally, dropouts were also implemented with a rate of 0.3, 0.25, and 0.2 but found that it did not yield the best results for our ResNet model on the CIFAR-10 dataset. Hence, we opted for a dropout rate of 0.0, effectively disabling dropout, as it provided better performance when combined with other regularization techniques like data augmentation and label smoothing.

- **L2 Regularization**: We used L2 regularization through weight decay in the **SGD** optimizer, setting it to 5×10^{-4}, to prevent overfitting by penalizing large weights and promoting more generalizable models. This approach helped stabilize training and improved the model's ability to generalize to unseen data, contributing to better performance on the CIFAR-10 dataset.

- **Label Smoothing**: Label smoothing is a regularization technique that modifies the target labels used during training to improve model generalization and calibration. Instead of using hard one-hot encoded labels (e.g.,), label smoothing creates "soft" labels. We achieved better performance using this technique as using this results in creating tighter representations and handle noisy labels more robustly.

Learning Rate and Epochs

We trained our ResNet model for 200 epochs, which helped it converge and perform well on the CIFAR-10 dataset. This duration was chosen after testing different training lengths to avoid overfitting or underfitting. Training for 250 or 300 epochs led to overfitting, with a large gap between training and validation performance. On the other hand, training for just 100 epochs resulted in underfitting, where the model didn't capture the data patterns effectively.

The 200-epoch training duration proved to be the best, ensuring that the model generalized well to unseen data, as shown by the competitive test set performance. The training saw rapid progress in the early epochs, followed by stabilization in the later stages, which is typical in deep learning. Overall, choosing 200 epochs was key to achieving a balanced and effective model, with improved stability and accuracy throughout the training process.

We experimented with 0.1 and 0.01 learning rates as well to see if it can help our model converge better. For initial models that were not using *Cosine Annealing*, we preferred to use a 0.01 learning rate with more epochs to see how the loss and accuracy curves turnout. After adding **Cosine Annealing** we finalized our learning rate as 0.1 as the learning rate will reduce with epochs so we were able to increase our learning rate to 0.1.

Results

After testing our model with different configurations we achieved a training accuracy of 97.65 and a final test accuracy of 87.39.
The model configurations are as follows:

- Number of layers: 3
- Blocks in each layer: [4,5,3]
- Input channels to each layer: [64,128,256]
- Filter Size: 3x3
- Kernel Size: 1x1
- Average pooling size: 8
- Number of parameters: 4,992,586

With the following hyperparameters:

- Batch Size: 128
- Optimizer: Lookahead + SGD
- Learning Rate: 0.1
- Momentum: 0.9
- Weight Decay: 0.0005
- Annealing Cosine: 200 epochs
- Learning Rate Decay: by 0.1
- Total Epochs: 200
- Lookahead Alpha: 0.5

We can see that the model begins to converge at around 200 epochs by looking at Epoch/Loss for train plot. The Epoch/Accuracy curves also tell a similar story with the accuracy plateauing around 200 epochs.

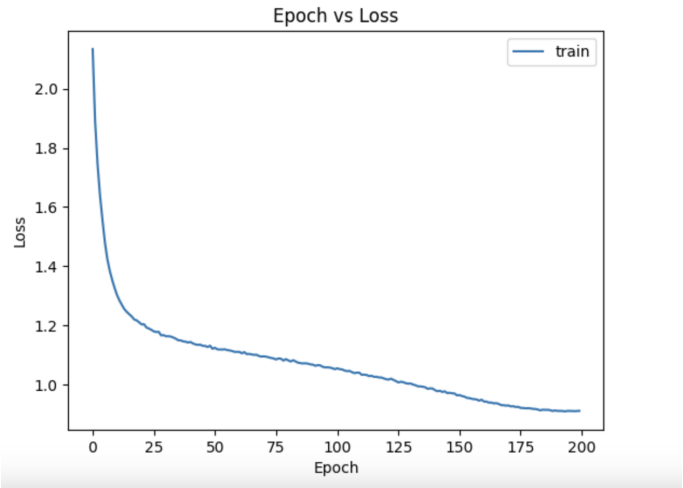

Figure 2: Training Loss wrt to Epochs trained

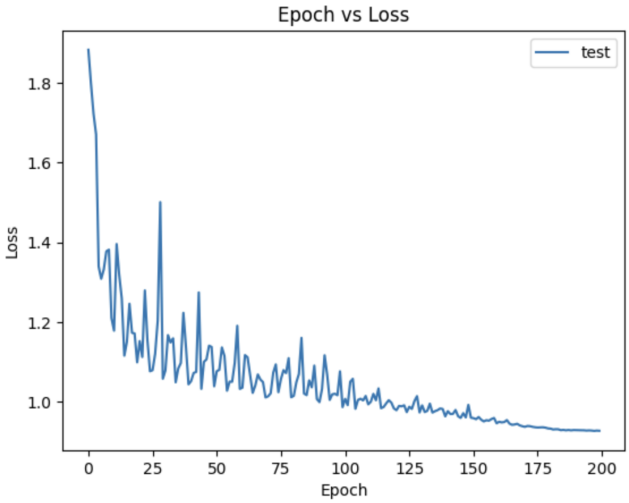

Figure 3: Testing Loss wrt to Epochs trained

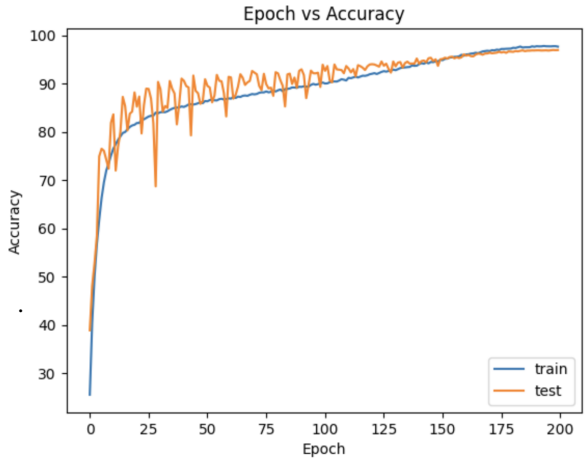

Figure 4: Accuracy wrt to Epochs trained

References

DeVries, T.; and Taylor, G. W. 2017. Improved Regularization of Convolutional Neural Networks with Cutout. arXiv:1708.04552.

facebook-research . 2017. "Mixup". https://github.com/facebookresearch/mixup-cifar10/blob/main/train.py.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385.

kuangliu. 2021. "Train CIFAR10 with PyTorch". https://github.com/kuangliu/pytorch-cifar/tree/master.

lonePatient . 2019. "LabelSmoothing". https://github.com/lonePatient/label_smoothing_pytorch.

Müller, R.; Kornblith, S.; and Hinton, G. 2020. When Does Label Smoothing Help? arXiv:1906.02629.

uoguelph-mlrg . 2021. "Cutout". https://github.com/uoguelph-mlrg/Cutout/tree/master.

Zhang, H.; Cisse, M.; Dauphin, Y. N.; and Lopez-Paz, D. 2018. mixup: Beyond Empirical Risk Minimization. arXiv:1710.09412.

Zhang, M. R.; Lucas, J.; Hinton, G.; and Ba, J. 2019. Lookahead Optimizer: k steps forward, 1 step back. arXiv:1907.08610.