Deep Learning Mini-Project – Spring-24 – NYU – Image Classification: Enhancing CIFAR-10 Performance With Modified ResNet

Ark Pandey[1]*, **Durga Avinash Kodavalla**[1]*, **Priyangshu Pal**[2]*

[1]New York University, Tandon School of Engineering, Department of Electrical and Computer Engineering (ECE)
[2]New York University, Tandon School of Engineering, Department of Computer Science and Engineering (CSE)
Mini-Project GitHub Repository: click here

Abstract

In this report, we will train a modified residual network (ResNet) architecture with the highest test accuracy on the CIFAR-10[1] image classification dataset, under the constraint that our model has no more than 5 million parameters.

Introduction

Deep convolutional neural networks have led to a series of breakthroughs for image classification. ResNet-18 is a convolutional neural network (CNN) architecture that has 18 layers. Building upon the idea of classifying images into one of ten classes, the project involves detailed analysis of how a ResNet-18 network can be updated to obtain $<10\%$ error on the CIFAR-10 image classification dataset. We present a residual learning framework with significantly fewer trainable parameters than those used previously. The performance of the trained model was evaluated on a separate test dataset consisting of previously unseen images from the CIFAR-10 dataset. The evaluation metrics used were test accuracy and test loss. The final test accuracy achieved by the model was $\approx 91.5\%$ with just **2.7M** trainable parameters.

Related Work

Deeper neural networks have always been more challenging to train due to vanishing gradients and optimization difficulties. In [2], the authors have proposed the architecture of a residual learning framework to ease the training of networks that are substantially deeper than those used previously. Instead of directly learning the desired underlying mapping, they learn residual functions with reference to the layer inputs. This modification in the architecture of a standard deep neural network allows for easier optimization of deep networks. Further in their research, empirical evidence shows that Residual Networks (ResNets) are easier to optimize and achieve higher accuracy with increased depth. On the ImageNet dataset, ResNets with up to 152 layers and a top-5 error of 3.57% outperforms networks like VGG.

Efficient ResNets: Residual Network Design[5] discusses architectural designs of residual networks which have less than 5 million parameters. They show that their ResNet with $<$5M parameters achieves a test accuracy of 96.04% on CIFAR-10 which is much higher than ResNet18 (which has greater than 11 million trainable parameters) when equipped with a number of training strategies and suitable ResNet hyperparameters.

*These authors contributed equally.

Architecture

In the CIFAR-10 dataset, the images are small in size (32x32) compared to other datasets like ImageNet (224x224). Therefore, a modified version of the ResNet architecture is used for these datasets, known as CifarResNet[3]. Our ResNet architecture has drawn inspiration from such CifarResNets. Our focus is to get the best behavior with minimal parameters, but not on pushing state-of-the-art results, so we intentionally use a simple architecture. Our model consists of an initial convolutional layer, followed by batch normalization. Considering the accuracy and time trade off[4], we chose the ReLU activation. It includes a global average pooling layer, which computes the average of the feature maps across the spatial dimensions. In total, the model has 22 layers as shown in Figure 1

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 32, 32]	864
BatchNorm2d-2	[-1, 32, 32, 32]	64
Conv2d-3	[-1, 32, 32, 32]	9,216
BatchNorm2d-4	[-1, 32, 32, 32]	64
Conv2d-5	[-1, 32, 32, 32]	9,216
BatchNorm2d-6	[-1, 32, 32, 32]	64
BasicBlock-7	[-1, 32, 32, 32]	0
Conv2d-8	[-1, 32, 32, 32]	9,216
BatchNorm2d-9	[-1, 32, 32, 32]	64
Conv2d-10	[-1, 32, 32, 32]	9,216
BatchNorm2d-11	[-1, 32, 32, 32]	64
BasicBlock-12	[-1, 32, 32, 32]	0
Conv2d-13	[-1, 64, 16, 16]	18,432
BatchNorm2d-14	[-1, 64, 16, 16]	128
Conv2d-15	[-1, 64, 16, 16]	36,864
BatchNorm2d-16	[-1, 64, 16, 16]	128
Conv2d-17	[-1, 64, 16, 16]	2,048
BatchNorm2d-18	[-1, 64, 16, 16]	128
BasicBlock-19	[-1, 64, 16, 16]	0
Conv2d-20	[-1, 64, 16, 16]	36,864
BatchNorm2d-21	[-1, 64, 16, 16]	128
Conv2d-22	[-1, 64, 16, 16]	36,864
BatchNorm2d-23	[-1, 64, 16, 16]	128
BasicBlock-24	[-1, 64, 16, 16]	0
Conv2d-25	[-1, 128, 8, 8]	73,728
BatchNorm2d-26	[-1, 128, 8, 8]	256

```
        Conv2d-27          [-1, 128, 8, 8]         147,456
   BatchNorm2d-28          [-1, 128, 8, 8]             256
        Conv2d-29          [-1, 128, 8, 8]           8,192
   BatchNorm2d-30          [-1, 128, 8, 8]             256
    BasicBlock-31          [-1, 128, 8, 8]               0
        Conv2d-32          [-1, 128, 8, 8]         147,456
   BatchNorm2d-33          [-1, 128, 8, 8]             256
        Conv2d-34          [-1, 128, 8, 8]         147,456
   BatchNorm2d-35          [-1, 128, 8, 8]             256
    BasicBlock-36          [-1, 128, 8, 8]               0
        Conv2d-37          [-1, 256, 4, 4]         294,912
   BatchNorm2d-38          [-1, 256, 4, 4]             512
        Conv2d-39          [-1, 256, 4, 4]         589,824
   BatchNorm2d-40          [-1, 256, 4, 4]             512
        Conv2d-41          [-1, 256, 4, 4]          32,768
   BatchNorm2d-42          [-1, 256, 4, 4]             512
    BasicBlock-43          [-1, 256, 4, 4]               0
        Conv2d-44          [-1, 256, 4, 4]         589,824
   BatchNorm2d-45          [-1, 256, 4, 4]             512
        Conv2d-46          [-1, 256, 4, 4]         589,824
   BatchNorm2d-47          [-1, 256, 4, 4]             512
    BasicBlock-48          [-1, 256, 4, 4]               0
        Linear-49                [-1, 10]           2,570
================================================================
Total params: 2,797,610
Trainable params: 2,797,610
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 5.63
Params size (MB): 10.67
Estimated Total Size (MB): 16.31
----------------------------------------------------------------
```

Figure 1: Model summary

Overall, this architecture is a variant of the ResNet architecture designed for the CIFAR-10 dataset. It includes shortcut connections and residual blocks to enable the training of deeper networks, and a global average pooling layer to capture spatial information.

Methodology

Data augmentation is an approach applied to the images of a training dataset to add diversity and increase its quantity by manipulating them with random transformations. In our code, various data augmentation techniques are implemented. This enables the model to make stronger predictions without overfitting. They imitate those which could be encountered with real-life events, and therefore providing a generalization better from the given data set to unseen new data. Below is an overview of the methods used and their goals:

- **Random Crop and Padding:** This involves cropping a random part of the input image and padding it if necessary. Here, it randomly crops a region of the image and then pads it back to its original size. This can simulate the effect of zooming in on parts of the image, helping the model to focus on details.
- **Random Horizontal Flip:** It gives us the image that is horizontally mirrored with a 50% chance. This reflects the possibility of objects facing either direction in real-life images, making the model invariant to horizontal orientation.
- **Color jitter:** It randomly changes the image's brightness, contrast, saturation, and hue by making the parameters to brightness=0.2, contrast=0.2, saturation=0.2, and hue=0.1. This mimics diverse lighting situations and color changes that happen in nature, helping with the identification of objects in varying environments
- **Random Rotation:** : It rotates the image by a degree randomly chosen in the range [-15, 15]. This helps to learn to recognize objects regardless of their orientation.
- **Random Affine Transformations:** It applies random affine transformations to the images, which include translation but not rotation (since degrees=0). This simulates a shift in the position of objects within the image frame, making the model adaptable to object location variations.
- **Normalization:** It is applied to normalize the pixel values of the image. Of course, this is a must for the training to be accelerated as we have to keep the input values approximately equal to normal distribution parameters(mean=0 and std=1). This decreases the overweighting of certain groups and leads to better solution, which improves the learning efficiency.

These data augmentation techniques are used to create a more versatile and generalized dataset, leading to improved performance on unseen data. By simulating a variety of real-world conditions, these transformations help to focus on the invariant features that define the classes of objects in the CIFAR-10 dataset, irrespective of variations in orientation, position, lighting, or color.

In our project, specific components have been selected for optimization, loss computation, and learning rate scheduling. Understanding why these choices are optimal for our task can help in tuning our model for better performance. Let's discuss the rationale behind these selections:

- **Loss Function - Cross Entropy Loss:** Cross Entropy Loss is commonly chosen for classification tasks as it evaluates the performance of a model that outputs a probability value between 0 and 1. Cross entropy loss grows as the predicted probability moves away from the actual label, which is ideal for tasks such as classifying the CIFAR-10 dataset, where each image is categorized into one of ten classes. It excels in imposing heavier penalties on incorrect classifications, thereby steering the model towards more precise predictions.
- **Optimizer - Adam:** The Adam optimizer is an extension to stochastic gradient descent that has been adopted widely in the field of deep learning. It combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. Adam is computationally efficient, requires little memory, and is well suited for problems that are large in terms of data and/or parameters. Its adaptive learning rate makes it suitable for handling the nuances of different parameters and their gradients, facilitating faster convergence.
- **Learning Rate Scheduler - ReduceLROnPlateau:** Learning rate scheduling adjusts the learning rate during training by reducing it according to a set strategy. ReduceLROnPlateau reduces the learning rate when a metric has stopped improving, which is ideal for fine-tuning

a model. By monitoring a particular performance metric (e.g., validation loss), this scheduler reduces the learning rate by a factor (defined in our code) when the metric stops decreasing, preventing overfitting and helping to escape local minima. This strategy is particularly useful in deep learning models like the one used in our project, where finding the right balance between exploration (via higher learning rates) and exploitation (via lower learning rates) is crucial for efficient training and achieving higher performance.

Results

In Figure 2, we plot the training and test loss curves against each epoch. Our ResNet model converged at a training loss of 0.1905 and a test loss of 0.2672.

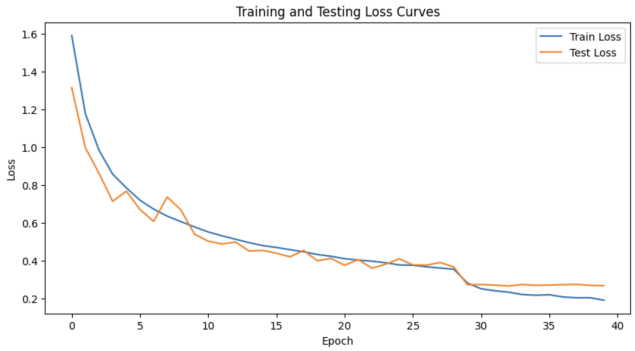

Figure 2: Loss curves

In Figure 3, we plot the training and test accuracy curves against each epoch. Our ResNet model converged at a training accuracy of 93.30% and a test accuracy of 91.48%.

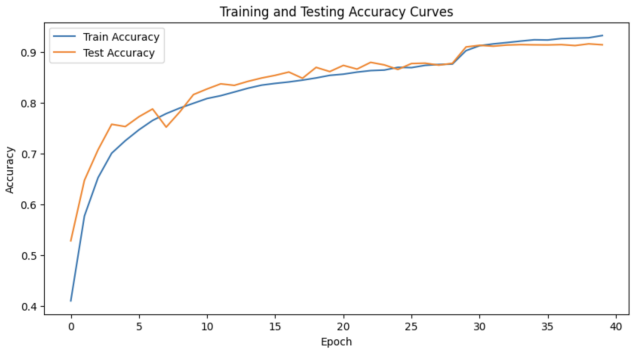

Figure 3: Accuracy curves

Our model trained for 40 epochs as we observed convergence at that point. We don't want our model to overfit to the training data so we didn't train it any longer and stopped at 40 epochs.

References

[1] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images.

[2] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. CoRR, abs/1512.03385.

[3] Shuvam Das (2023). Implementation of ResNet Architecture for CIFAR-10 and CIFAR-100 Datasets.

[4] Liu, W. et al. (2021). Improvement of CIFAR-10 Image Classification Based on Modified ResNet-34. In: Meng, H., Lei, T., Li, M., Li, K., Xiong, N., Wang, L. (eds) Advances in Natural Computation, Fuzzy Systems and Knowledge Discovery. ICNC-FSKD 2020. Lecture Notes on Data Engineering and Communications Technologies, vol 88. Springer, Cham. https://doi.org/10.1007/978-3-030-70665-4_68

[5] Thakur, A., Chauhan, H., Gupta, N. (2023). Efficient ResNets: Residual Network Design. arXiv preprint, arXiv:2306.12100.