Report of Deep Learning Mini-Project

Wanqing Li, Jin Qin, Yihao Wang

New York University
wl3187@nyu.edu, jq2325@nyu.edu, yw7486@nyu.edu

Abstract

This project sought to enhance the ResNet architecture by methodically adjusting elements such as the number of blocks, stride size, and channel count, alongside various hyperparameters. These modifications led to notable improvements in computational efficiency and model accuracy, with test accuracies reaching up to 92% on standard batches and 80.5% on Kaggle competitions. Furthermore, these adjustments successfully reduced the model's trainable parameters from 11 million to 4.9 million. Through these systematic alterations in the structural and operational parameters, the project demonstrated an increased proficiency in fine-tuning the architecture for image classification tasks. The code is available on https://github.com/eus-lwq/NYUResnetChallenge.

Introduction

The objective of this project is to innovate upon the established Residual Network (ResNet) architecture to enhance its performance on the CIFAR-10 image classification dataset. CIFAR-10, a benchmark dataset in the field of machine learning, consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. The dataset is split into 50,000 training images and 10,000 testing images, examples are shown in figure x, CIFAR-10 is offering a standardized platform for evaluating the efficiency and effectiveness of image classification models. ResNet(He et al. 2016) is a revolutionary neural network architecture introduced by Kaiming He et al. in 2015, fundamentally changed the landscape of deep learning by enabling the training of extremely deep neural networks through the use of residual connections. These connections mitigate the vanishing gradient problem, allowing for the training of networks that are much deeper than was previously feasible. Despite its success, there is always room for improvement, especially in terms of model size and computational efficiency, which are crucial for deployment in resource-constrained environments.

Meanwhile, CIFAR-10 dataset is notably balanced, featuring approximately equal number of images across all classes shown in figure y. Consequently, there was no necessity to apply resampling techniques to achieve balance within the dataset.

Method

Hyper-parameters

Stride For adjusting stride size we find if we decrease the original stride size it will increase the computational cost and memory usage, it may also cause some overfitting due to the detailness of model detection. If we increase the stride size, this would decrease the resolution of feature maps more quickly, potentially decreasing overfitting on training data and lead to better generalization. However, it risks missing out on important local features, which might be crucial for our model.

Number of channels In the revised version of our ResNet architecture, channel configurations were adjusted from [64, 128, 256, 512] to [32, 64, 128, 256] to reduce the computational burden and increase model efficiency. This modification resulted in a significant reduction of the model's parameters to approximately 2.7 million. This decrease in parameter count not only reduces memory usage and accelerates processing but also maintains deployment feasibility in environments with limited resources. Importantly, the initial evaluations indicate that the model's efficacy is preserved, demonstrating that the optimized architecture continues to capture the critical features necessary for precise

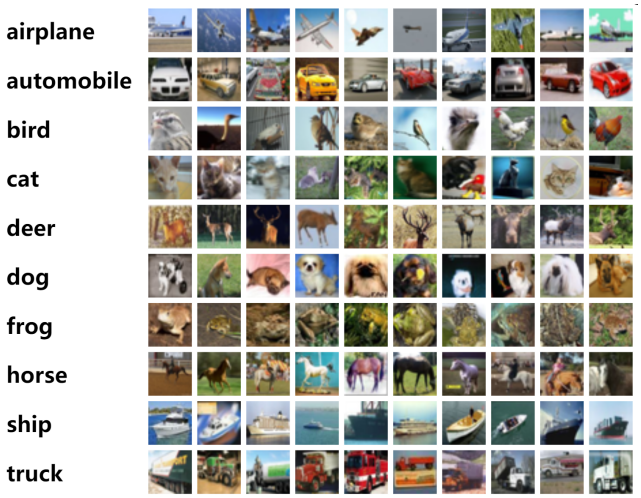

Figure 1: Visualization of CIFAR-10 dataset

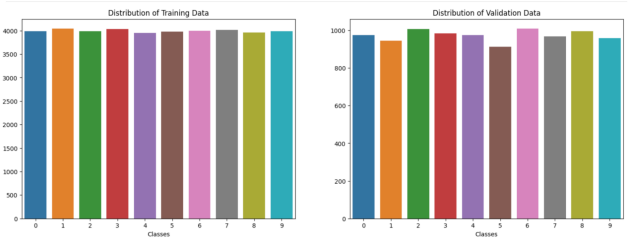

Figure 2: Classes distribution of CIFAR-10 train and validation dataset

predictions. This equilibrium between efficiency and accuracy highlights the success of the channel reduction approach in refining neural network models.

Number of Blocks The adjustment of the ResNet block configuration demonstrates its efficiency and robustness in handling parameter reductions while maintaining performance, particularly on the CIFAR-10 dataset. Reducing the ResNet configuration from four blocks of two layers each to four blocks of one layer each resulted in a significant reduction of the model's parameters from over 11M to 4.9M . Despite this reduction, the model continues to achieve relatively high accuracy on the test batch. Although a decrease in the number of blocks leads to fewer layers, the performance remains robust. This high level of performance may be attributed to inherent characteristics of the ResNet architecture, skip connection, to reduce overfitting and even with less layer still robust performance.

LR Optimizer and Weight decay Both SGD (Stochastic Gradient Descent) and ADAM (Adaptive Moment Estimation) utilize minibatches to process data, but they handle learning rate adjustments differently. SGD applies a uniform update across all parameters, adjusting the learning rates collectively based on the aggregate gradient descent. In contrast, ADAM enhances this approach by adapting the learning rates individually for each parameter. It does this through calculating exponential moving averages of past gradients and squared gradients, which allows it to make more tailored updates. This adaptive mechanism often results in faster convergence in practice, particularly in complex models involving large datasets, as it can fine-tune learning rates dynamically to suit different parameters throughout the training process.

For better convergence, we also tried AdamW(Loshchilov and Hutter 2019), which is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments with an added method to decay weights per the techniques discussed in the paper, 'Decoupled Weight Decay Regularization' by Loshchilov, Hutter et al., 2019.

For optimizer we also tuned the size of weight decay, range from 0.00001 to 0.001, we find if the weight decay size is too small, it brings up Insufficient Penalization: The model's weights can grow too large, which may make the model's output highly sensitive to the input data, potentially leading to overfitting. It also may bring poor generalization, a small weight decay may allow the model to fit the training data very closely, including the noise, which harms the model's ability to generalize to new data, as reflected in the erratic validation loss.

Scheduler For better convergence, we used CosineAnnealing learning rate scheduler.

Epochs We investigate the impact of varying epochs on model accuracy on the test-patch of original 'cifar-10' data and the 'nolabel' data, and achieve the higher performance. The experiment was structured around three distinct training intervals: 100 epochs, 50 epochs, and 200 epochs. The outcomes were as follows:

During the initial trial at 100 epochs, the model demonstrated a commendable accuracy of 91% when evaluated against the original test-patch. In contrast, its performance on the 'nolabel' dataset was slightly lower compared to the original test-patch, with an accuracy of 80%. The number of epochs appeared to yield satisfactory performance when evaluating the model on the 'cifar-10' test-patch. However, it became apparent that this number of epochs was insufficient to achieve desirable results when assessing the model against the 'nolabel' test patch. A reduction in training duration to 50 epochs resulted in a marginal decrease in accuracy to 89% on the original test-patch, accompanied by a significant drop in performance on the nolabel dataset to 34% accuracy. This outcome is emblematic of underfitting, wherein the model's limited exposure to the complexities of the data spectrum hampers its predictive accuracy and adaptability.

Extending the epoch count to 200 yielded a modest increase in accuracy to 92% on the original test-patch. However, the model exhibited signs of overfitting, as evidenced by a markedly low accuracy of 36.4% on the nolabel dataset. This phenomenon indicates an excessive optimization on the training data, to the detriment of the model's generalization to novel, unlabeled inputs.

Batch Size Increasing the batch size necessitates a corresponding increase in the number of epochs and intensifies the demand on computational resources such as GPU and CPU processing capabilities. Smaller batch sizes, while providing less precise gradient estimates, introduce variability that may facilitate the escape from local minima; however, they can also reduce the stability of the training process. On the other hand, larger batch sizes yield more accurate gradient directions, though the improvements in model training diminish as batch sizes become excessively large, which may result in a less effective convergence and a greater likelihood of remaining trapped in sharp minima. Research indicates that smaller batch sizes enhance the model's ability to generalize effectively to unfamiliar data, a benefit attributed to the regularization effect of the noise inherent in smaller batches, which assists in preventing overfitting to the training dataset.

Generalization

To reach better generalization, we also tried many different techniques, such as data augmentation, finding early stopping points, and utilizing semi-supervise.

Data Augmentation

CutMix is an augmentation technique where sections of one image are cut and pasted into another, mixing their re-

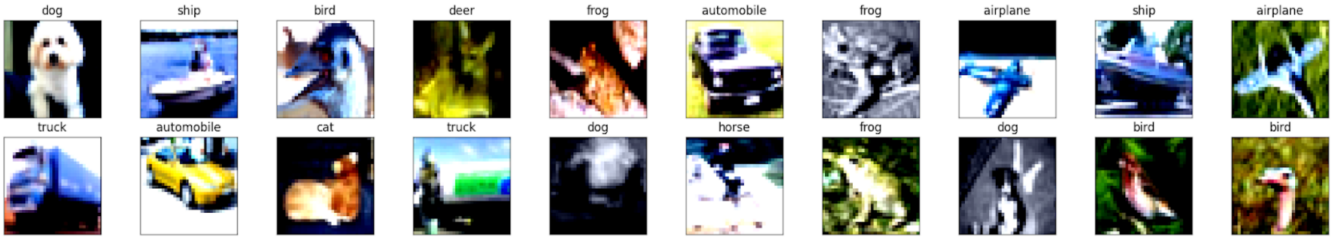

Figure 3: Visualization of training data with augmentation

spective labels proportionally to the area of the cut segments. This method helps in training models that are robust and less prone to overfitting by introducing more diverse data and encouraging the model to focus on non-dominant features. However, it can create complexity in how the model interprets mixed images and potentially introduce misleading label noise. MixUp blends two images together at the pixel level, averaging their pixels and labels. This technique smooths out the label space and extends the training distribution, promoting better generalization and lessening overfitting risks. The primary downside is that it can produce unrealistic images that might challenge the model's ability to learn relevant features from natural images, and it requires careful hyperparameter tuning to be effective. By randomly choosing between CutMix and MixUp for each batch, this approach increases input variability and combines the advantages of both methods, potentially compensating for their individual weaknesses. However, the added complexity and inconsistent training signals could complicate the training process and affect convergence. Other common transformations include random resizing, cropping, flipping, and color adjustments (such as color jittering and auto contrast), along with occasional conversions to grayscale. These transformations are designed to ensure that the model learns to recognize features independently of their orientation, size, or color, thus enhancing robustness. However, they might remove critical information and increase computational requirements, prolonging training times.

Semi-supervised / Self-training semi-supervised is usually used where obtaining a sufficient amount of labeled data is prohibitively difficult or expensive, but large amounts of unlabeled data are relatively easy to acquire (Reddy, Viswanath, and Reddy 2018). In this project test setting, we have 10,000 unlabeled data in nolabel.pkl, so we use a trained model to create pseudo-label on unlabeled data then feed it back for learning. But after a few test trials, we find the testing result doesn't generalize well on test data, nor see any significant improvement on test accuracy. We find it may possibly be due to semi-supervised learning its own property: it often requires a large amount of unlabeled data, but in this scenario, we have more labeled data rather than unlabeled data. If we are allowed to add another dataset's unlabelled data points to train our model, I think semi-supervised learning could be more efficient and more appropriate to use.

Figure 4: best train val loss, red is validation loss, blue is training loss

Result

In the evaluation of the ResNet model designed for this project, the best performance metrics were observed across various testing environments. The model achieved a peak accuracy of 92% on the designated test batch, while recording a lower accuracy of 80.5% on the Kaggle platform. Regarding loss metrics, the training loss was reported at 0.01272, with the validation loss significantly lower at 0.00032. The train val graph is shown in figure 4[1].

The architectural configuration of the ResNet model comprises four blocks, each containing a single residual unit, as indicated by the structure [1, 1, 1, 1]. The model utilizes a uniform stride of three across all layers and employs filters of size 3x3 in each block. Channel depth progression is established as follows: 64, 128, 256, and 512 across the respective blocks. The training process was conducted over 100 epochs with a batch size of 20, under the exclusive operation of a single computational process.

Optimization was facilitated through the use of a Stochastic Gradient Descent (SGD) optimizer, enhanced by a cosine annealing scheduler for dynamic adjustment of the learning rate and a modest weight decay set at 0.0001 to mitigate overfitting. This setup supports a model with approximately 4.9 million trainable parameters, indicating a substantial capacity for learning complex patterns in the dataset.

[1]Graph generated from wandb.log, to full see wandb report, check wandb report

Challenges

Bottleneck and resources allocation Initially, the training process was markedly slow due to the configuration of the number of subprocesses at zero, leading to the underutilization of the CPU. This inefficiency resulted in excessive GPU resources being idle, as they awaited the processing and transfer of data by the CPU. To address this issue, the number of subprocesses was increased to the maximum permitted by Kaggle. This adjustment significantly reduced the duration required to complete one epoch with a batch size of 256 to approximately 20 seconds.

Parameter Restriction The model initially exceeded the specified requirement of 5 million parameters, prompting the implementation of various techniques to reduce the size of the parameters. After adjusting the number of blocks and channels, we observed a significant reduction in the total number of parameters while maintaining high performance.

Debug with model architecture Initially, there was a significant issue with the number of channels; attempts to modify the channels independently led to numerous conflicts. However, through the study of relevant materials and examination of similar projects, we gradually acquired the knowledge necessary to effectively debug and resolve issues related to the number of channels.

Overfitting and generalization The challenge of overfitting the validation set and determining an appropriate early stopping point emerged prominently. We initially employed a stochastic gradient descent (SGD) optimizer to update each parameter with the same learning-rate at each epoch and It may cause the convergence to be quite slow. Meanwhile, many comparative analysis (Zhou et al. 2020) revealed that SGD operates at a slower rate than both Adam and AdamW optimizers but generalize better. Consequently, we shifted our strategy to adopt AdamW, which is the more recent and prevalently used optimizer in contemporary research and industry contexts.

Despite this change, establishing a precise early stopping point remained problematic, as our model consistently tended to overfit the training data. In response, we altered our approach to hyperparameter tuning by using the training data as a benchmark. This adjustment allowed us to incorporate test accuracy metrics directly into the training process. By doing so, we aimed to obtain a genuine measure of our model's performance, thereby facilitating the identification of a balanced and effective stopping point to mitigate the risk of overfitting.

Future Work

For future work, we considered utilizing Wandb(Biewald 2020) sweeps for hyper-parameter optimization but eventually chose manual tuning due to the protracted time requirements and limited resources available. Although Wandb Sweeper is adept at identifying correlations and assessing the significance of various hyper-parameters, facilitating the discovery of optimal settings, the extensive duration required to execute a sweep across a broad array of hyper-parameters — potentially spanning several days—was not feasible within our resource constraints. Consequently, we opted for manual tuning of the hyper-parameters to minimize redundancy and conserve resources.

Despite the project's restrictions against transfer learning and the addition of new data, future explorations might benefit from incorporating transfer learning techniques, introducing noise to the training dataset, or utilizing additional small datasets. Such strategies could potentially enhance the generalization capabilities of the model on unlabeled test data and increase its overall robustness.

We will also try other popular methods for regularization as well as improving generalization, such as label smoothing.Another promising avenue involves self-supervised learning techniques. To specify, we may try pre-training the model with DINO(Caron et al. 2021) or EMP-SSL(Tong et al. 2023)[2] in a self-supervised manner, and then fine-tuning the model on the CIFAR-10 dataset in a fully supervised manner.

Furthermore, another method to be explored in our future work involves the technique of teacher-student knowledge distillation. This approach leverages a pre-trained, typically larger and more complex 'teacher' model to guide the training of a smaller 'student' model. The student model learns to approximate the teacher's output distributions, which often captures more generalizable features and subtle data patterns than those obtained through direct training. Implementing this method could potentially streamline our model, improving performance and efficiency without sacrificing accuracy, thereby aligning with the project's goal of optimizing resource use while maintaining high model effectiveness.

References

Biewald, L. 2020. Experiment Tracking with Weights and Biases. Software available from wandb.com.

Caron, M.; Touvron, H.; Misra, I.; Jégou, H.; Mairal, J.; Bojanowski, P.; and Joulin, A. 2021. Emerging Properties in Self-Supervised Vision Transformers. In *Proceedings of the International Conference on Computer Vision (ICCV)*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Loshchilov, I.; and Hutter, F. 2019. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*.

Reddy, Y.; Viswanath, P.; and Reddy, B. E. 2018. Semi-supervised learning: A brief review. *Int. J. Eng. Technol*, 7(1.8): 81.

Tong, S.; Chen, Y.; Ma, Y.; and Lecun, Y. 2023. EMP-SSL: Towards Self-Supervised Learning in One Training Epoch. *arXiv preprint arXiv:2304.03977*.

Zhou, P.; Feng, J.; Ma, C.; Xiong, C.; Hoi, S. C. H.; et al. 2020. Towards theoretically understanding why sgd generalizes better than adam in deep learning. In *Advances in Neural Information Processing Systems*, volume 33, 21285–21296.

[2] which was claimed to work perfectly with ResNet backbone