# Deep Learning Mini-Project

**Raghuram(rc5428), Tanmay(tr2452), Vishnu (vb2409)**

New York University
Access the Project Code on GitHub Here!

## Abstract

In this project, we undertake a detailed exploration of the residual network (ResNet) architecture tailored for the CIFAR-10 image classification challenge, with the goal of maximizing test accuracy while adhering to a strict constraint of under 5 million parameters. Our approach involved a systematic modification of the ResNet architecture through meticulous adjustments of various hyperparameters including the number of channels, filter sizes, kernel dimensions, and pooling strategies. Additionally, we incorporated advanced training techniques and optimization strategies to enhance model performance. Through iterative experimentation and validation, we successfully developed a modified ResNet model that not only achieves an impressive test accuracy of over 90% on the CIFAR-10 dataset but also remains within the parameter budget. The final architecture demonstrates a significant improvement in handling overfitting and generalization compared to traditional ResNet models. This report details the iterative process of our experimental design, the challenges encountered, the solutions implemented, and the significant outcomes that contribute to the field of efficient neural network design for image recognition.

## Resnet18 Architechture

The core building block of the ResNet18 architecture is the Residual Block, which is designed to address the problem of vanishing gradients that can occur in very deep neural networks. The Residual Block consists of the following components:

**Convolutional Layer:** The first convolutional layer takes the input and applies a set of learnable filters to produce a feature map.

**Batch Normalization:** The batch normalization layer is used to normalize the output of the previous convolutional layer, which helps to stabilize the training process and improve the model's performance.

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$
$$y_i = \gamma \hat{x}_i + \beta$$

**Activation Function:** The rectified linear unit (ReLU) activation function is applied to the output of the batch normalization layer, introducing non-linearity to the model.

$$f(x) = \max(0, x)$$

**Skip Connection:** The key component of the Residual Block is the shortcut connection, which bypasses the two convolutional layers and directly adds the input to the output of the second batch normalization layer. This shortcut connection allows the network to learn the residual mapping between the input and output, rather than the full mapping, which can be easier to optimize.

To fully develop a model to understand the performance of ResNet18 on the CIFAR10 Dataset, we have to understand the idea behind the resnet.The ResNet18 is built on an idea called the Skip/Residual, this connection takes the activations from an (n-1) convolution layer and adds it to the convolution output of (n+1) layer and then applies ReLU on this sum, thus Skipping the n layer.The below diagram explains how a skip connection works. (Here we are using f(x) to denote Relu applied on x where x is the output after applying Convolution operation). The ResNet18 architecture

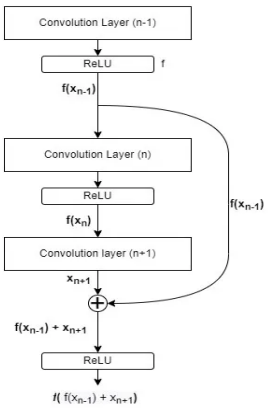


Figure 1: One Residual Layer

is composed of multiple Residual Blocks stacked together, with some additional layers and pooling operations.

## Dataset-CIFAR10

The CIFAR-10 dataset is a widely used benchmark, the small image size and limited number of classes may not fully represent the complexity of real-world image classification tasks. Size: The CIFAR-10 dataset consists of 60,000 color images, each with a resolution of 32x32 pixels. Classes: The dataset contains 10 mutually exclusive classes, each with 6,000 images. The classes are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Data Split: The dataset is split into a training set of 50,000 images and a test set of 10,000 images. Image Format: The images are stored in the RGB color format, with each pixel represented by a 3-byte value (one byte for each color channel).


Figure 2: CIFAR-10 dataset classes

## Implementation

We began our project by implementing the standard ResNet-18 architecture, which typically consists of a series of BasicBlock layers. To meet the stringent 5 million parameter constraint, we made several significant modifications. Initially, we reduced the number of channels in each layer, starting from 64 channels in the first layer and incrementing up to 256 channels in the final layer. Our model, which we have named ResNet4m, boasts a total of 4,962,922 trainable parameters, thus fitting within the project's parameter limit.

### ResNet4m Architecture Overview

Our final model architecture exclusively utilizes the BasicBlock as the foundational building unit, contrary to some versions of ResNet that incorporate Bottleneck blocks. The BasicBlock facilitates the standard residual connection, where the output from a series of convolutional and batch normalization layers is added to a shortcut connection. This configuration enables the model to learn residual functions and train effectively even with deep network structures.

Each BasicBlock in our ResNet4m model consists of two convolutional layers with 3x3 kernel sizes, designed to maintain spatial dimensions while altering the depth of feature maps. Accompanying these are batch normalization layers that help stabilize learning by normalizing the activations, promoting faster and more stable training. The blocks also incorporate a shortcut connection that employs a 1x1 convolution when a change in dimensions is necessary due to increased planes or a change in stride. This structure allows the model to learn residual functions effectively and enhances the ability to train deep neural networks without degradation in performance.

### Data Preparation and Augmentation Techniques

Data handling was a crucial aspect of our project, aiming to optimize the model's performance on the CIFAR-10 dataset. We developed a custom dataset handler, CIFAR10Custom, which unpacks the dataset from its original pickled format and prepares it for use with PyTorch. This handler enables on-the-fly transformations of the images during training and validation, which is essential for effective learning.

For data augmentation, our approach included random crops and horizontal flips to introduce variability into the training process, simulating different scenarios that the model might encounter in real-world applications. Specifically, we used a random crop of 32x32 pixels with a padding of 4 pixels and randomly flipped the images horizontally. These transformations were applied only to the training dataset to simulate a more varied data environment, which helps in improving the model's generalization capabilities.

Normalization was applied to both the training and validation datasets with mean values of [ 0.4914 , 0.4822 , 0.4465 ] [0.4914,0.4822,0.4465] and standard deviations of [ 0.2023 , 0.1994 , 0.2010 ] [0.2023,0.1994,0.2010], aligning the data distribution to be more consistent and centered around zero. This normalization is crucial for maintaining numerical stability during the learning process and ensuring that the gradient updates occur smoothly across all layers of the network.

Our training strategy also incorporated a scheduled learning rate adjustment using a cosine annealing schedule, which gradually reduces the learning rate as training progresses. This method helps in fine-tuning the model parameters towards the latter stages of training to avoid overshooting minima in the loss landscape. The entire training and validation processes were conducted using CUDA-enabled devices, with the model and data being loaded onto GPU memory to expedite computations.

### Advanced Techniques Explored

As part of our exploration to maximize the model's performance, we experimented with the concept of teacher-student distillation. We trained a larger ResNet model with approximately 11 million parameters to serve as the teacher. The idea was to distill its knowledge into our more parameter-efficient model. However, our modified 5 million parameter model, which we named ResNet4m, already achieved high validation accuracy. This performance made the distillation process redundant, as the simpler model was sufficient for our objectives without the need for additional complexity.

In addition to the distillation experiment, we also evaluated a PreResNet architecture. The PreResNet model, which modifies the order of operations in the residual block, was tested to see if it could offer better generalization capabilities. We aimed to make a Kaggle competition submission with this model. Despite our efforts, the PreResNet model

did not outperform our modified ResNet architecture and, in some cases, even yielded lower test accuracies. Consequently, we decided to continue with our customized ResNet configuration for the final implementation.

These combined strategies of architectural design and data handling have been pivotal in achieving over 90% validation accuracy on the CIFAR-10 dataset, adhering to the strict parameter constraints set out by the project while ensuring robustness and efficiency in model training.

Our model stacks these BasicBlocks in four sequential layers with arrangements [2, 2, 2, 2], maintaining depth and reducing spatial dimension progressively. This setup is preceded by an initial convolutional layer and batch normalization, and concludes with an average pooling layer and a final linear layer for classification.

## Results and Observations

### Training and Validation Performance

The performance of our model over training epochs is illustrated in Figure 3. The left subplot shows the training and validation losses as the model progresses through the epochs, indicating a decrease in loss and thus an improvement in the model's fit to the data. The right subplot depicts the training and validation accuracies, highlighting the model's increasing accuracy, which suggests effective learning and generalization.

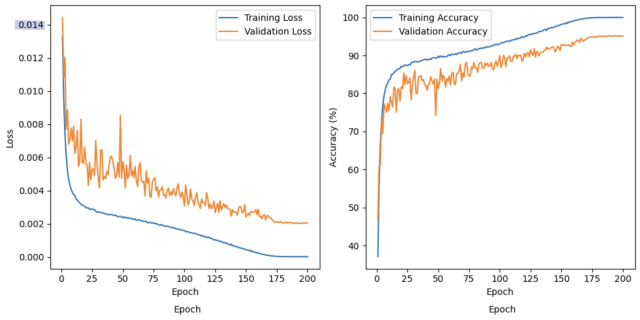


Figure 3: Training and Validation Losses and Accuracies over epochs. The left graph shows the losses decreasing over time, which indicates that the model is learning effectively. The right graph shows the accuracies, both training and validation, which increase over time, demonstrating the model's ability to generalize well to new, unseen data.

Throughout our iterative process, we achieved impressive internal validation accuracies, including a peak of 95.11% on our validation data and nearly 99% on training data by epoch 198:

- **Training Data:** Epoch 188, Loss: 0.002, Accuracy: 99.992% (49996/50000)
- **Validation Data:** Epoch 188, Loss: 0.202, Accuracy: 95.200% (9520/10000)

### Results of Our Experiments

Below are the results of the submissions our team has made, showcasing the progress and improvements over various training epochs and configurations:

| File Name | Author | Date | Description | Score |
|---|---|---|---|---|
| predictions_3.csv | Vishnu Beji | 1d ago | 4.9 mil 200 epochs | 0.852 |
| predictions_2.csv | Vishnu Beji | 1d ago | 4.6 mil after 200 epochs | 0.857 |
| predictions.csv | Raghuram_2309 | 3d ago | | 0.851 |
| predictions (1).csv | Vishnu Beji | 3d ago | 4 mil model after 100 epochs | 0.770 |
| predictions.csv | Vishnu Beji | 4d ago | | 0.798 |
| predictions.csv | Tanmay Anil Rathi | 4d ago | 4.1M parameters Test1 | 0.848 |
| submission.csv | Raghuram_2309 | 4d ago | | 0.096 |
| submission.csv | Raghuram_2309 | 4d ago | Error | |
| submission.csv | Raghuram_2309 | 9d ago | | 0.099 |
| submission.csv | Raghuram_2309 | 9d ago | | 0.099 |
| submission.csv | Raghuram_2309 | 9d ago | | 0.099 |

Table 1: Submission results and their respective performance metrics.

Despite these promising results, when we submitted our predictions on the Kaggle competition's test set, which lacks labels, our accuracy hovered around 85%. This discrepancy suggests a potential shift in data distribution between the training/validation sets and the Kaggle test set, or possibly that the model is still underfitting. It indicates that while our model performs exceedingly well on familiar data, it struggles to generalize to new, unseen data from the same task.

### Analysis and Future Directions

The observed performance gap points to the need for strategies that could enhance the model's generalization ability. One such strategy could be knowledge distillation, where a more complex model (teacher) transfers knowledge to our simpler model (student). This technique might help our model learn more robust features that are not tied as closely to the training data distribution, thus improving performance on external test sets.

## Conclusions

Through this mini-project, we developed a modified ResNet architecture, named ResNet4m, which achieves over 90% validation accuracy on the CIFAR-10 dataset while adhering to a 5 million parameter constraint. Our key design choices involved utilizing BasicBlock layers exclusively, tuning the dimensions of channels strategically, and employing appropriate pooling techniques. The iterative process of experimenting with different configurations and evaluating their impact was crucial to achieving both the performance and parameter goals. This project underscored the importance of balancing model complexity and efficiency, a critical consideration for deploying deep learning models within resource-constrained settings.

## Acknowledgements

We extend our deepest gratitude to Professor Chinmay Hegde for his expert guidance and unwavering support throughout this deep learning mini-project. His invaluable advice significantly influenced our approach and strategies in modifying the ResNet architecture.

Our thanks also go to the teaching assistants whose assistance was instrumental in clarifying critical concepts and enhancing our project execution. We appreciate the collaborative efforts of our classmates whose insights and discussions contributed to a rich learning environment.

Additionally, we are grateful for the computational resources provided by Kaggle, which enabled us to efficiently train our models and conduct extensive experiments. The use of Kaggle's platforms allowed us to leverage advanced GPU capabilities essential for our project's success.

Our project implementation benefited greatly from the PyTorch community, particularly the repository maintained by Kuang Liu, which provided an excellent starting point for training Residual Networks on the CIFAR-10 dataset. Here is the citation of the repository we used:

## References

[1] Liu, K. (2023). *PyTorch CIFAR-10 Training Scripts*. Available at: https://github.com/kuangliu/pytorch-cifar. Accessed 04/08/2024.

Our project implementation benefited greatly from the PyTorch community, particularly the repository maintained by Kuang Liu, which provided an excellent starting point for training Residual Networks on the CIFAR-10 dataset.