Deep Residual Learning: Analyzing Model Performance on CIFAR-10 Images

Aaron Bengochea
Repository: https://github.com/aaronbengochea/cifar10-evaluation

Abstract

In this paper, we explore the performance of residual convolutional neural networks on the CIFAR-10 classification task. We conduct a comprehensive study comparing shallow, mid-depth, and deep residual architectures—ranging from 10 to 102 layers—focusing on the BasicBlock design originally proposed in ResNet18 [1]. Our investigation explores the impact of various hyperparameters including optimizer selection (SGD vs. ADAM), learning rate scheduling, and regularization through pre-training and training-time image augmentations. Experimental results reveal that while deeper networks generally achieve improved accuracy, careful tuning of the optimization parameters is crucial to address challenges such as overfitting and degradation in training performance. Furthermore, our study highlights that the interplay between block count and channel size significantly influences the network's capacity and convergence behavior. The insights from our findings provide practical guidelines for efficiently training deep residual networks in scenarios with constrained computational resources focusing on networks with less than 5M trainable parameters.

Overview

In this study, we extend the ideas presented in the original "Deep Residual Learning for Image Recognition" [1] paper by conducting an in-depth analysis of residual networks on the CIFAR-10 dataset. Our investigation is centered on the use of BasicBlocks as defined in ResNet18 [1], and we evaluate three variants of residual convolutional architectures:

- **ResNet10**: A shallower network designed to test the lower bound of network depth.
- **ResNet40**: A mid-depth architecture intended to balance model complexity and performance.
- **ResNet102**: A deep variant that explores the benefits and challenges of extreme depth.

To assess model generalizability, we also evaluated the performance of each network variant on a hidden test set. The primary focus of our experiments was to determine how variations in network depth affect convergence behavior, overfitting, and overall classification accuracy. In our experimental setup, we trained models using a variety of model hyperparameter choices, including:

- **Optimizer Selection**: Comparison between SGD and ADAM to determine the impact on convergence speed and final accuracy.
- **Learning Rate Scheduling**: Analysis of different strategies for adjusting the learning rate throughout the training process.
- **Regularization Techniques**: Examination of pretraining and training-time image augmentations, including techniques such as CutMix, to enhance model generalizability.

Our findings indicate that among the three architectures tested, the ResNet40 variant delivered optimal results on the hidden test set, slightly outperforming both the shallow ResNet10 and the deep ResNet102 variants. This optimal performance suggests that a mid-depth architecture effectively balances model capacity with generalizability when using the BasicBlock design. The comprehensive experiments validate the hypothesis that while shallower networks may underfit and excessively deep networks may suffer from optimization challenges, a well-tuned medium-depth network such as the ResNet40 we propose, achieves the best trade-off between accuracy and computational efficiency.

Methodology

1. Architecture

We compare three control models based on residual convolutional neural network architectures which include: ResNet10, ResNet40, and ResNet102. All architectures are constructed using the BasicBlock used in ResNet18 [1], ensuring consistency in building block design across all variants, allowing us to gauge which model performs best for the CIFAR-10 classification task. Below are the key architectural details for each model:

- **ResNet10:**
  - *Blocks per Layer:* [1, 1, 1, 1]
  - *Channels per Layer:* [64, 128, 256, 512]
  - *Convolutional Kernel Size:* [3x3]
  - *Convolutional Skip Kernel Size:* [1x1]
  - *Adaptive Pooling Size:* 1
  - *Total Trainable Parameters:* 4.90M
  - *Hidden Testset Accuracy:* 83.18%

- **ResNet40:**
  - *Blocks per Layer:* [5, 7, 4, 3]
  - *Channels per Layer:* [32, 64, 128, 256]
  - *Convolutional Kernel Size:* [3x3]
  - *Convolutional Skip Kernel Size:* [1x1]
  - *Adaptive Pooling Size:* 1
  - *Total Trainable Parameters:* 4.99M
  - *Hidden Testset Accuracy:* 84.37%
- **ResNet102:**
  - *Blocks per Layer:* [8, 13, 17, 12]
  - *Channels per Layer:* [16, 32, 64, 128]
  - *Convolutional Kernel Size:* [3x3]
  - *Convolutional Skip Kernel Size:* [1x1]
  - *Adaptive Pooling Size:* 1
  - *Total Trainable Parameters:* 4.99M
  - *Hidden Testset Accuracy:* 83.39%

Each architecture adheres to a consistent design framework while varying in depth and complexity. The ResNet10 model serves as our shallow network baseline, ResNet40 serves as our mid-depth network baseline and the ResNet102 model serves as our deep network baseline. Choosing this set of control models with varying depths and similar trainable parameters allows us to evaluate the isolated effects that increased network depth has on model training efficiency and CIFAR-10 classification performance.

2. Optimizers

We evaluated two popular optimization methods: Stochastic Gradient Descent (SGD) and ADAM; In order to determine their effects on model convergence and performance. These optimizers differ in their update mechanisms and hyperparameter sensitivities, which in turn influence the training dynamics and final accuracy. Below are the specific configurations used for each throughout our study:

- **SGD:**
  - *Learning Rate:* 0.1
  - *Momentum:* 0.9
  - *Weight Decay:* 5e-4
- **ADAM:**
  - *Learning Rate:* 0.1
  - *Weight Decay:* 1e-4

By comparing these two optimization methods, we aim to assess how adaptive learning rates (as in ADAM) and momentum-driven updates (as in SGD) impact the convergence behavior and overall accuracy of the residual networks on CIFAR-10.

2. Learning Rate Scheduler

In addition to the choice of optimizer, the scheduling of the learning rate plays a crucial role in model convergence and overall performance. We evaluated two learning rate schedulers to adjust the learning rate during training:

- **ReduceLROnPlateau:**
  - *Mode:* 'max' (monitors an evaluation metric and reduces the learning rate when it ceases to improve)
  - *Factor:* 0.5 (multiplies the learning rate by 0.5 upon plateau)
  - *Patience:* 5 (number of epochs with no improvement before reducing the learning rate)
- **CosineAnnealingLR:**
  - *T_max:* EPOCHS (the maximum number of epochs for the cosine annealing cycle)

ReduceLROnPlateau dynamically reduces the learning rate when performance plateaus, allowing the model to fine-tune its parameters as training progresses. CosineAnnealingLR, on the other hand, follows a cosine function to gradually decrease the learning rate over the training period, often leading to smoother convergence. By experimenting with these two learning rate schedulers, we aim to understand how dynamic adjustments to the learning rate can further optimize model performance and convergence during training.

3. Regularization

To improve model generalizability and prevent overfitting, we applied minor random data augmentations during pre-training. These are the same base regularization techniques that were employed in the original ResNet18 study [1]. The following strategies were uniformly implemented across all models:

- **Random Horizontal Flips:** Images were randomly flipped horizontally, introducing variation that mimics real-world scenarios.
- **Random Crop:** A random crop of size 32 with a padding of 4 was applied to each image, further diversifying the training data.

By adopting these standard augmentation methods for all of our control models, we ensure that our models learn robust features and maintain consistency with the regularization approaches proven effective in ResNet18. We also ensure that all control models have the same regularization techniques applied, again allowing us to focus on model performance as a function of depth alone.

Control Model Findings

We trained and evaluated the ResNet10, ResNet40, and ResNet102 architectures under two distinct sets of assumptions. The first set of assumptions is detailed below.

1. Two Distinct Assumption Sets

Assumption Set One (AS1):

In this configuration, our training setup leverages the following hyperparameter choices:

- **Optimizer: SGD**
  - *Learning Rate:* 0.1
  - *Momentum:* 0.9

  - *Weight Decay:* 5e-4
- **Learning Rate Scheduler: LinAnnealLR**
  - *T_max:* EPOCHS
- **Regularization:**
  - Random horizontal flips
  - Random cropping with a crop size of 32 and a padding of 4

This first set of assumptions serves as a baseline configuration, allowing us to assess how well each architecture performs under standard training conditions using SGD with a linear annealing learning rate schedule and consistent base regularization.

Assumption Set Two (AS2):

In this configuration, our training setup employs the ADAM optimizer combined with a ReduceLROnPlateau learning rate scheduler, alongside the same base regularization techniques. The specific hyperparameter choices are as follows:

- **Optimizer: ADAM**
  - *Learning Rate:* 0.1
  - *Weight Decay:* 1e-4
- **Learning Rate Scheduler: ReduceLROnPlateau**
  - *Mode:* 'max' (monitors an evaluation metric and reduces the learning rate when it ceases to improve)
  - *Factor:* 0.5 (multiplies the learning rate by 0.5 upon plateau)
  - *Patience:* 5 (number of epochs with no improvement before reducing the learning rate)
- **Regularization:** As in Assumption Set One, all models were pre-trained with standard data augmentation methods, including:
  - Random horizontal flips
  - Random cropping with a crop size of 32 and a padding of 4

This second set of assumptions allows us to evaluate the effect of using an adaptive optimizer (ADAM) in conjunction with a dynamic learning rate reduction strategy on model convergence and overall performance.

2. Outcomes

In our experiments, **Assumption Set Two (AS2)**, which utilized ADAM and the ReduceLROnPlateau scheduler, exhibited signs of faster initial convergence. However, managing the learning rate proved challenging; despite multiple parameter adjustments, the learning rate often dropped too quickly, limiting the network's capacity to continue improving. As a result, the overall performance under AS2 was suboptimal for deeper training runs.

On the other hand, **Assumption Set One (AS1)**, which used SGD and the LinAnnealLR scheduler, provided a smoother, more gradual decrease in the learning rate. This steady annealing process allowed the networks to learn more consistently over time, leading to higher final accuracies across the ResNet10, ResNet40, and ResNet102 architectures.

Figures 1 and 2 illustrate these outcomes for ResNet40. In Figure 1, the faster drop in learning rate under ADAM + ReduceLROnPlateau (AS2) can be seen to coincide with an earlier plateau in test accuracy, while Figure 2 demonstrates the more gradual decrease in learning rate under SGD + LinAnnealLR (AS1), allowing the model to continue improving over a longer training period.

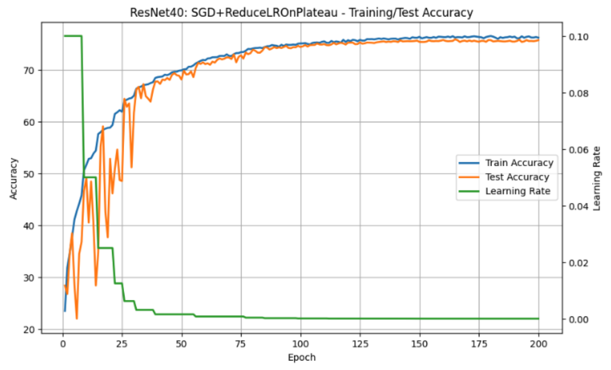


Figure 1: ResNet40 trained with ADAM + ReduceLROnPlateau (AS2). Note the early and rapid decline in learning rate.

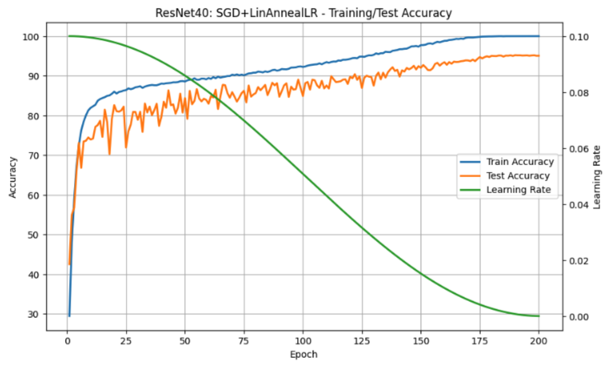


Figure 2: ResNet40 trained with SGD + LinAnnealLR (AS1). The smoother decrease in learning rate contributes to more consistent training progress.

Given these observations, we will focus our subsequent analysis on the models trained under **AS1**, as it has proven to be the more effective set of hyperparameter choices for achieving robust performance on CIFAR-10, mainly due to our control of the learning rate. We achieve a total of 95.05% on the test set accuracy using AS1, while only achieving a total of 75.71% using AS2, which shows that using SGD in combination with LinAnnealLR provides a smoother convergence landscape compared to ADAM and ReduceLROnPlateau.

To further illustrate the impact of AS1 on all three depth variants, Figures 3, 4, and 5 show the training and test accuracy curves for ResNet10, ResNet40, and ResNet102 respectively under SGD + LinAnnealLR.

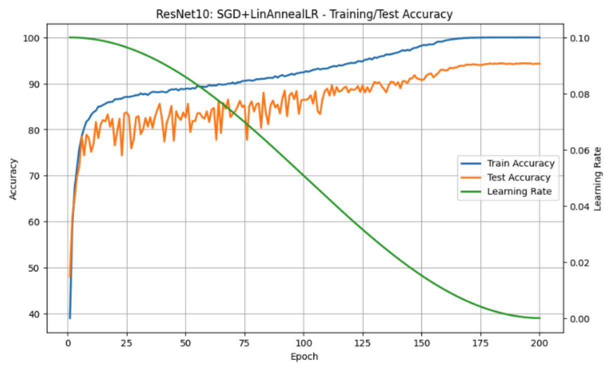


Figure 3: ResNet10 trained with SGD + LinAnnealLR (AS1).

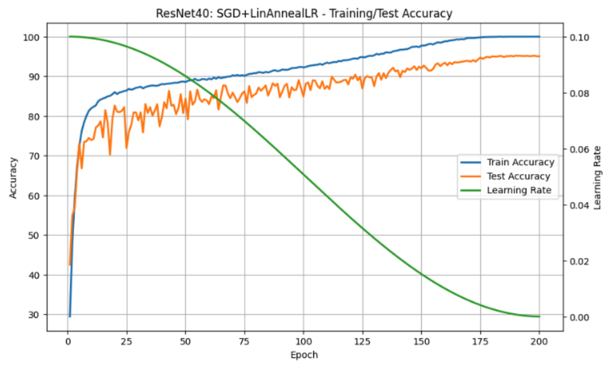


Figure 4: ResNet40 trained with SGD + LinAnnealLR (AS1).

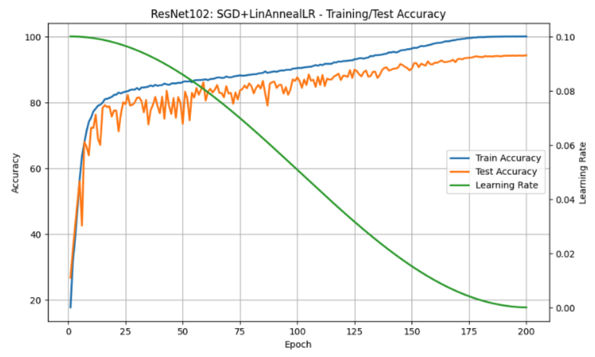


Figure 5: ResNet102 trained with SGD + LinAnnealLR (AS1).

Table 1: Maximum Test and Hidden Test Accuracies

Model	**Test Acc (%)**	**Hidden Test Acc (%)**
ResNet10	94.41	83.18
ResNet40	95.20	84.33
ResNet102	94.23	83.39

Among these three depth variants, **ResNet40** demonstrated the most potential, yielding the highest performance on both the standard test set and the hidden test set. Thus, the mid-depth **ResNet40** architecture, in combination with **SGD** and **LinAnnealLR**, provided the most promising results overall. Consequently, we will focus our fine-tuning efforts and further analyses on the ResNet40 model.

Most Performant Model – Results

Building on our previous observation that **ResNet40** provided the best overall performance among the three depth variants, we further refined this model to maximize its accuracy and generalization. Specifically, we:

- Extended the training duration to **300 epochs** (an additional 100 epochs compared to the control setup).
- Retained the **AS1** hyperparameter assumptions (i.e., SGD + LinAnnealLR + base regularization).
- Added two additional regularization strategies: **CutMix** [2] and **RandomAffine**.
- We name this model **ResNet40-FT**

Regularization - CutMix

CutMix is a data augmentation technique that cuts and pastes patches from one image onto another while also combining the labels proportionally. By replacing a random patch of pixels from one image with a patch from another, the network learns to identify objects even under occlusion or partial views. This forces the model to pay attention to less discriminative regions, thereby enhancing its robustness and reducing overfitting. Additionally, by blending the labels, CutMix affects the cross-entropy loss during training, preventing the model from becoming overly confident on a single class and encouraging a more balanced and generalized learning process.

Regularization - RandomAffine

We also applied a `RandomAffine` transformation with the following parameters:

- *Degrees:* 10 (random rotation in the range of ±10 degrees)
- *Translate:* (0.1, 0.1) (random horizontal and vertical translation up to 10% of the image size)
- *Scale:* (0.9, 1.1) (random scaling between 90% and 110%)
- *Shear:* 10 (random shearing up to 10 degrees)

This affine transformation further diversifies the training data by randomly rotating, shifting, scaling, or shearing each image. By training on such distorted images, the network becomes more robust to geometric variations in the input data.

Architecture and Assumptions

For our most performant model, we focus on the ResNet40 architecture—identified in our previous analysis as the optimal depth variant. In this experiment, we build upon the base architectural framework and control assumptions described earlier, with additional regularization techniques and extended training duration. The experimental setup is defined by the following key points:

- **Architecture: ResNet40**
  - *Block Type:* BasicBlock
  - *Blocks per Layer:* [5, 7, 4, 3]
  - *Channels per Layer:* [32, 64, 128, 256]
  - *Convolutional Kernel Size:* [3x3]
  - *Convolutional Skip Kernel Size:* [1x1]
  - *Adaptive Pooling Size:* 1
  - *Total Trainable Parameters:* 4.99M
- **Assumptions:**
  - **Training Duration:** 310 epochs (110 epochs of additional fine-tuning over our control experiments)
  - **Train/Test Batch Sizes:** Train = 128, Test = 100
  - **Optimizer:** SGD
    - *Learning Rate:* 0.1
    - *Momentum:* 0.9
    - *Weight Decay:* 5e-4
  - **Learning Rate Scheduler:** LinAnnealLR
    - *T_max:* EPOCHS = 310
  - **Base Regularization:** Standard data augmentations including random horizontal flips and random cropping (crop size 32 with padding 4) as used in the original ResNet18 study [1].
  - **Additional Regularization:**
    - **CutMix:** Applied randomly during training-time to our training batches
    - **RandomAffine:** An affine transformation with parameters which introduces random rotations, translations, scaling, and shearing. This further diversifies the training data and makes the model more robust to geometric variations.

Final Results

The incorporation of additional regularization techniques, Randomized CutMix and RandomAffine, provided considerable improvements in the learning capacity of our models. In particular, our experiments reaffirm that careful management of the learning rate is one of the key components in extracting the most performance gains from our networks. The smooth convergence facilitated by the LinAnnealLR scheduler in combination with the enhanced data augmentations enabled our fine-tuned ResNet40 model to slightly outperform the control models.

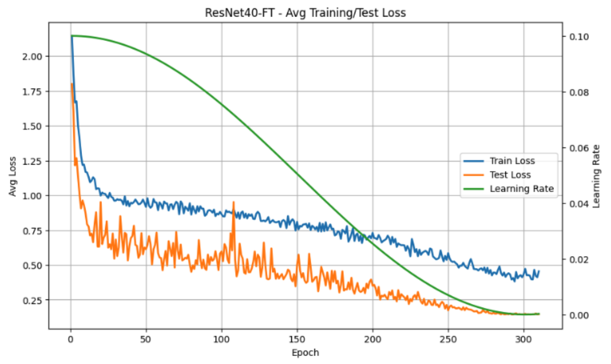


Figure 6: ResNet40-FT: Training and Test Loss Curves. The green line indicates the learning rate.

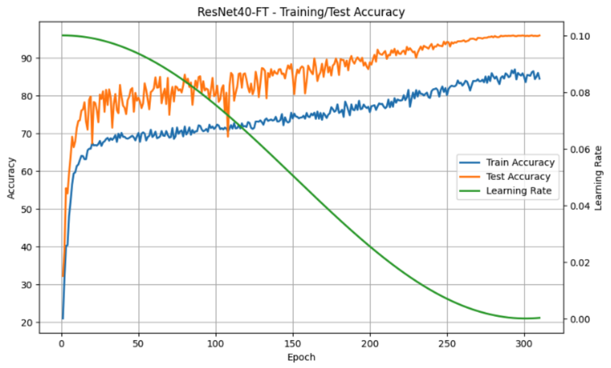


Figure 7: ResNet40-FT: Training and Test Accuracy Curves. The green line indicates the learning rate.

Table 2 summarizes the performance metrics for the control models as well as the fine-tuned ResNet40 model (ResNet40-FT). Notably, the fine-tuned ResNet40 model achieved a test accuracy of 96.0% and a hidden test accuracy of 85.09%, outperforming the other variants.

Table 2: Maximum Test and Hidden Test Accuracies

Model	**Test Acc (%)**	**Hidden Test Acc (%)**
ResNet10	94.41	83.18
ResNet40	95.20	84.33
ResNet40-FT	96.01	85.09
ResNet102	94.23	83.39

These final results clearly demonstrate that the ResNet40 mid-depth architecture, when combined with SGD, LinAnnealLR, and the enhanced regularization strategies, provides the most promising performance. ResNet40-FT represents our final model. A detailed summary of ResNet40-FT can be found in the appendix section, Figure 6.

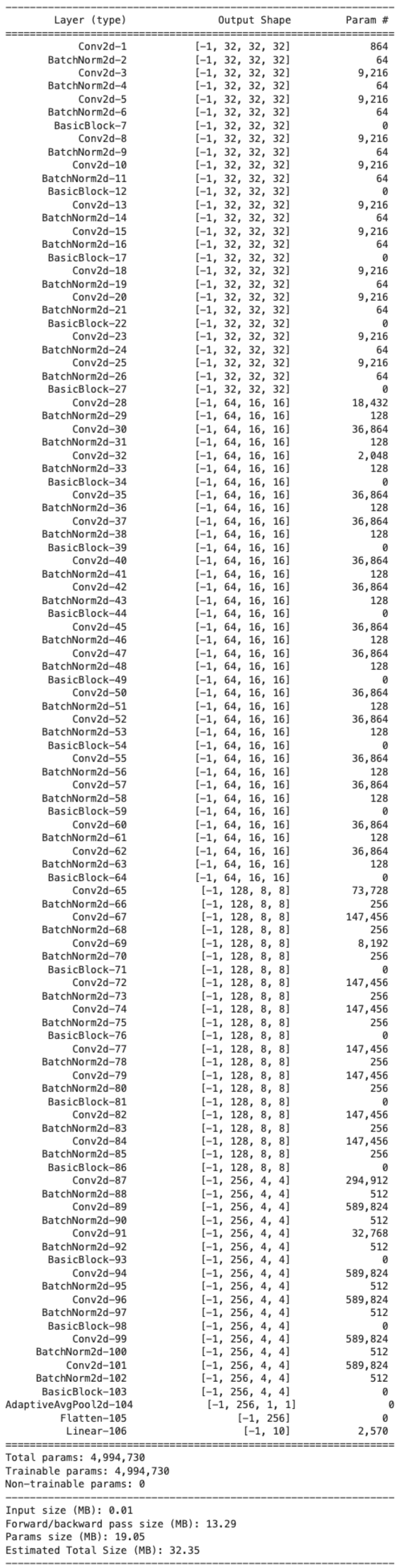
ResNet40 Architecture Summary

Figure 8 displays the ResNet40 architecture summary as generated by `torch.summary`. This overview provides detailed insights into the layer configurations, output shapes, and the number of trainable parameters in our ResNet40 model.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.

[2] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo. CutMix: Regularization strategy to train strong classifiers with localizable features. arXiv preprint arXiv:1905.04899, 2019.



```
----------------------------------------------------------------
        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, 32, 32, 32]           9,216
      BatchNorm2d-14           [-1, 32, 32, 32]              64
           Conv2d-15           [-1, 32, 32, 32]           9,216
      BatchNorm2d-16           [-1, 32, 32, 32]              64
       BasicBlock-17           [-1, 32, 32, 32]               0
           Conv2d-18           [-1, 32, 32, 32]           9,216
      BatchNorm2d-19           [-1, 32, 32, 32]              64
           Conv2d-20           [-1, 32, 32, 32]           9,216
      BatchNorm2d-21           [-1, 32, 32, 32]              64
       BasicBlock-22           [-1, 32, 32, 32]               0
           Conv2d-23           [-1, 32, 32, 32]           9,216
      BatchNorm2d-24           [-1, 32, 32, 32]              64
           Conv2d-25           [-1, 32, 32, 32]           9,216
      BatchNorm2d-26           [-1, 32, 32, 32]              64
       BasicBlock-27           [-1, 32, 32, 32]               0
           Conv2d-28           [-1, 64, 16, 16]          18,432
      BatchNorm2d-29           [-1, 64, 16, 16]             128
           Conv2d-30           [-1, 64, 16, 16]          36,864
      BatchNorm2d-31           [-1, 64, 16, 16]             128
           Conv2d-32           [-1, 64, 16, 16]           2,048
      BatchNorm2d-33           [-1, 64, 16, 16]             128
       BasicBlock-34           [-1, 64, 16, 16]               0
           Conv2d-35           [-1, 64, 16, 16]          36,864
      BatchNorm2d-36           [-1, 64, 16, 16]             128
           Conv2d-37           [-1, 64, 16, 16]          36,864
      BatchNorm2d-38           [-1, 64, 16, 16]             128
       BasicBlock-39           [-1, 64, 16, 16]               0
           Conv2d-40           [-1, 64, 16, 16]          36,864
      BatchNorm2d-41           [-1, 64, 16, 16]             128
           Conv2d-42           [-1, 64, 16, 16]          36,864
      BatchNorm2d-43           [-1, 64, 16, 16]             128
       BasicBlock-44           [-1, 64, 16, 16]               0
           Conv2d-45           [-1, 64, 16, 16]          36,864
      BatchNorm2d-46           [-1, 64, 16, 16]             128
           Conv2d-47           [-1, 64, 16, 16]          36,864
      BatchNorm2d-48           [-1, 64, 16, 16]             128
       BasicBlock-49           [-1, 64, 16, 16]               0
           Conv2d-50           [-1, 64, 16, 16]          36,864
      BatchNorm2d-51           [-1, 64, 16, 16]             128
           Conv2d-52           [-1, 64, 16, 16]          36,864
      BatchNorm2d-53           [-1, 64, 16, 16]             128
       BasicBlock-54           [-1, 64, 16, 16]               0
           Conv2d-55           [-1, 64, 16, 16]          36,864
      BatchNorm2d-56           [-1, 64, 16, 16]             128
           Conv2d-57           [-1, 64, 16, 16]          36,864
      BatchNorm2d-58           [-1, 64, 16, 16]             128
       BasicBlock-59           [-1, 64, 16, 16]               0
           Conv2d-60           [-1, 64, 16, 16]          36,864
      BatchNorm2d-61           [-1, 64, 16, 16]             128
           Conv2d-62           [-1, 64, 16, 16]          36,864
      BatchNorm2d-63           [-1, 64, 16, 16]             128
       BasicBlock-64           [-1, 64, 16, 16]               0
           Conv2d-65            [-1, 128, 8, 8]          73,728
      BatchNorm2d-66            [-1, 128, 8, 8]             256
           Conv2d-67            [-1, 128, 8, 8]         147,456
      BatchNorm2d-68            [-1, 128, 8, 8]             256
           Conv2d-69            [-1, 128, 8, 8]           8,192
      BatchNorm2d-70            [-1, 128, 8, 8]             256
       BasicBlock-71            [-1, 128, 8, 8]               0
           Conv2d-72            [-1, 128, 8, 8]         147,456
      BatchNorm2d-73            [-1, 128, 8, 8]             256
           Conv2d-74            [-1, 128, 8, 8]         147,456
      BatchNorm2d-75            [-1, 128, 8, 8]             256
       BasicBlock-76            [-1, 128, 8, 8]               0
           Conv2d-77            [-1, 128, 8, 8]         147,456
      BatchNorm2d-78            [-1, 128, 8, 8]             256
           Conv2d-79            [-1, 128, 8, 8]         147,456
      BatchNorm2d-80            [-1, 128, 8, 8]             256
       BasicBlock-81            [-1, 128, 8, 8]               0
           Conv2d-82            [-1, 128, 8, 8]         147,456
      BatchNorm2d-83            [-1, 128, 8, 8]             256
           Conv2d-84            [-1, 128, 8, 8]         147,456
      BatchNorm2d-85            [-1, 128, 8, 8]             256
       BasicBlock-86            [-1, 128, 8, 8]               0
           Conv2d-87            [-1, 256, 4, 4]         294,912
      BatchNorm2d-88            [-1, 256, 4, 4]             512
           Conv2d-89            [-1, 256, 4, 4]         589,824
      BatchNorm2d-90            [-1, 256, 4, 4]             512
           Conv2d-91            [-1, 256, 4, 4]          32,768
      BatchNorm2d-92            [-1, 256, 4, 4]             512
       BasicBlock-93            [-1, 256, 4, 4]               0
           Conv2d-94            [-1, 256, 4, 4]         589,824
      BatchNorm2d-95            [-1, 256, 4, 4]             512
           Conv2d-96            [-1, 256, 4, 4]         589,824
      BatchNorm2d-97            [-1, 256, 4, 4]             512
       BasicBlock-98            [-1, 256, 4, 4]               0
           Conv2d-99            [-1, 256, 4, 4]         589,824
     BatchNorm2d-100            [-1, 256, 4, 4]             512
          Conv2d-101            [-1, 256, 4, 4]         589,824
     BatchNorm2d-102            [-1, 256, 4, 4]             512
      BasicBlock-103            [-1, 256, 4, 4]               0
AdaptiveAvgPool2d-104            [-1, 256, 1, 1]               0
         Flatten-105                  [-1, 256]               0
          Linear-106                   [-1, 10]           2,570
================================================================
Total params: 4,994,730
Trainable params: 4,994,730
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 13.29
Params size (MB): 19.05
Estimated Total Size (MB): 32.35
----------------------------------------------------------------
```

Figure 8: ResNet40 architecture summary as generated by `torch.summary`.