Midterm report

midterm report

Kunshu Yang

Created on March 30 | Last edited on March 30

AI Disclosure

In this project covering Parts 1, 2, and 3, I used AI tools such as ChatGPT and GitHub Copilot to assist in brainstorming ideas and gathering suggestions for code organization and hyperparameter tuning. However, all core aspects of the project—including the design and implementation of the model architectures, training and evaluation pipelines, and data augmentation strategies—were developed, refined, and rigorously tested by me.

For Part 1, I independently designed and implemented the SimpleCNN model and its complete training loop. Although AI provided initial ideas on layer arrangement and hyperparameter settings, every decision was ultimately made based on my experiments and observations. In Part 2, I transitioned to a more sophisticated approach by adopting the ResNet18 model, which I modified to suit the CIFAR-100 dataset. AI suggestions helped me confirm best practices for adapting pretrained architectures, but the modifications and fine-tuning were carried out by myself. For Part 3, I further expanded the methodology by integrating transfer learning strategies and advanced training techniques, again building upon insights generated through iterative experimentation.

Overall, while AI-assisted tools offered valuable recommendations, the final code and design decisions are the result of my original work, ensuring that the project meets the required standards.

Part1

Model Description

Model Architecture

The SimpleCNN model adopted in this section is intentionally kept straightforward and is divided into two primary components: feature extraction and the classifier. In the feature extraction stage, the network consists of three convolutional blocks. The first block begins with a convolutional layer that applies a 3×3 kernel to the input RGB image (of size 3×32×32), converting it into 32 feature maps while maintaining the spatial dimensions by using a padding of 1. This is immediately followed by batch normalization, which standardizes the output to stabilize the activations, and a ReLU activation to introduce nonlinearity. Finally, a 2×2 max pooling operation reduces the spatial dimensions from 32×32 to 16×16.

The second convolutional block operates similarly; it uses a 3×3 convolution to transform the 32 feature maps into 64 feature maps (with the same padding settings), followed by batch normalization and a ReLU activation. A 2×2 max pooling then reduces the spatial dimensions further from 16×16 to 8×8. In the third convolutional block, the network applies another 3×3 convolution (with padding set to 1) to convert the 64 feature maps into 128 feature maps. This block is also followed by batch normalization and a ReLU activation, but no pooling is applied here so that more local features are preserved.

The classifier component begins by flattening the output from the convolutional blocks. This flattened vector, originally of size 128×8×8, is then passed through a fully connected layer that maps it to a 512-dimensional feature vector. A ReLU activation introduces further nonlinearity, and a dropout layer with a dropout probability of 0.5 is incorporated to reduce the risk of overfitting. Finally, another fully connected layer maps the 512 features to 100 outputs, corresponding directly to the 100 classes of the CIFAR-100 dataset.

Design Motivation and Limitations

The design of the SimpleCNN model was primarily driven by the need for a simple and easily debuggable architecture that could quickly establish a full training, validation, and testing pipeline. This baseline model allowed for rapid experimentation and provided valuable insights into performance limitations, which serve as a foundation for further improvements. Batch normalization was utilized to stabilize and accelerate training, and dropout was integrated to help mitigate overfitting.

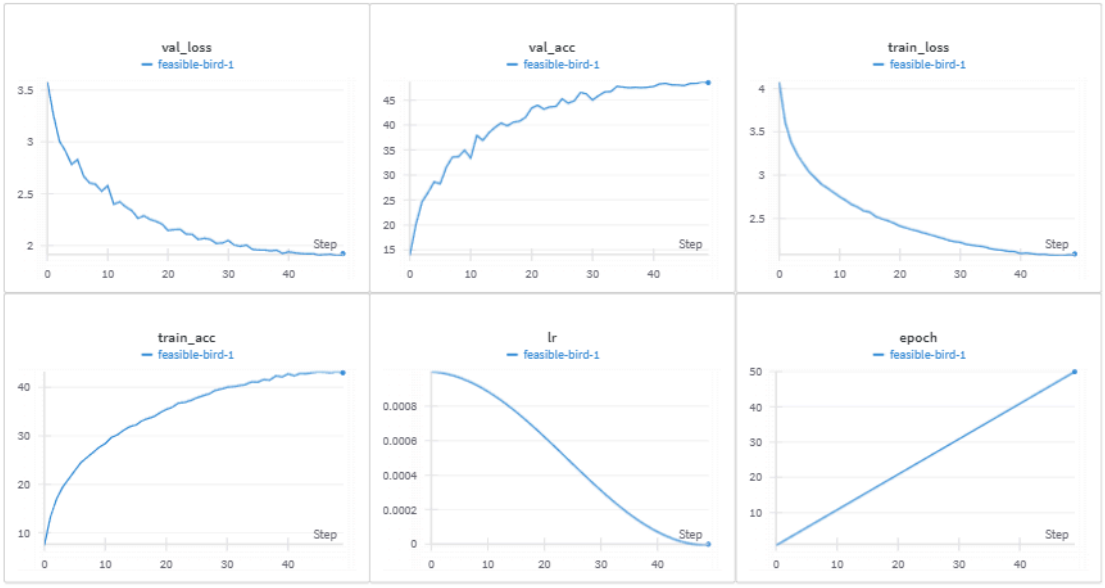
However, the simplicity of this CNN also introduces limitations. Due to its shallow architecture, the model's capacity to extract and represent complex features is inherently limited, which contributes to its lower accuracy. Additionally, the relatively low validation accuracy observed during training indicates that the model is underfitting, underscoring the need for more sophisticated architectures. These shortcomings have motivated the exploration of more complex, pretrained models in Parts 2 and 3, where advanced architectures and transfer learning strategies will be employed to achieve higher performance.

Hyperparameter Tuning

In this experiment, the primary hyperparameter settings were as follows: a batch size of 128 was chosen, an initial learning rate of 0.001 was used, and the AdamW optimizer was employed with a weight decay of 0.01. The model was trained for 50 epochs, and we utilized the CosineAnnealingLR scheduler to gradually reduce the learning rate throughout the training process. Preliminary experiments indicated that an initial learning rate of 0.001 provided stable training dynamics, while the combination of the AdamW optimizer and weight decay helped to control model complexity and mitigate overfitting.

Observations from the Wandb plots further reinforced these findings: the training loss steadily decreased, indicating that the

model was effectively fitting the training data, yet the validation accuracy remained limited—hovering between 36% and 40%. This discrepancy suggests that the simple model architecture might be insufficient in capturing the complex features required for higher performance. These insights have provided a clear direction for future work in Parts 2 and 3, where more sophisticated models and more refined hyperparameter tuning strategies will be employed to further enhance performance.



Part2

Model Description

Model Architecture

In Part 2, I switched from the simpler CNN used in Part 1 to a more sophisticated ResNet18 architecture provided by the torchvision library. ResNet18 introduces residual blocks that help mitigate the vanishing gradient problem in deeper networks, thus enabling more effective feature extraction for CIFAR-100. Specifically, I replaced the original fully connected layer with a linear layer that outputs 100

classes, ensuring the model aligns with CIFAR-100's classification requirements.

The ResNet18 architecture consists of an initial convolutional layer followed by a series of residual blocks and finally a fully connected classifier. The network begins with a convolutional layer that uses a 7×7 kernel with a stride of 2 and a relatively large number of filters, followed by batch normalization and a ReLU activation. However, since the CIFAR-100 dataset consists of 32×32 images, I modified the initial convolution layer to use a 3×3 kernel with a stride of 1 and a padding of 1. This change ensures that the spatial dimensions are preserved, which is crucial for such small input images. Additionally, I replaced the original max pooling layer with an identity operation to prevent excessive downsampling of the already small images.

Following the modified initial layers, ResNet18 is organized into four groups of residual blocks. Each group contains two residual blocks, and each block is designed with two convolutional layers that have 3×3 kernels. The key aspect of these blocks is the presence of skip connections, which add the input of the block to its output after the convolutional operations. This mechanism allows the network to learn residual functions, thereby improving the flow of gradients during backpropagation. Batch normalization and ReLU activation functions are applied after each convolution within the blocks, ensuring stable and efficient training.

After passing through all residual blocks, the network applies a global average pooling operation that reduces each feature map to a single value. The output of this pooling layer is a vector that is then fed into the final fully connected layer. In the standard ResNet18 architecture, this final layer is configured for 1000-class classification (for ImageNet), but I replaced it with a new linear layer tailored for CIFAR-100, which has 100 output classes.

▾ Design Motivation and Limitations

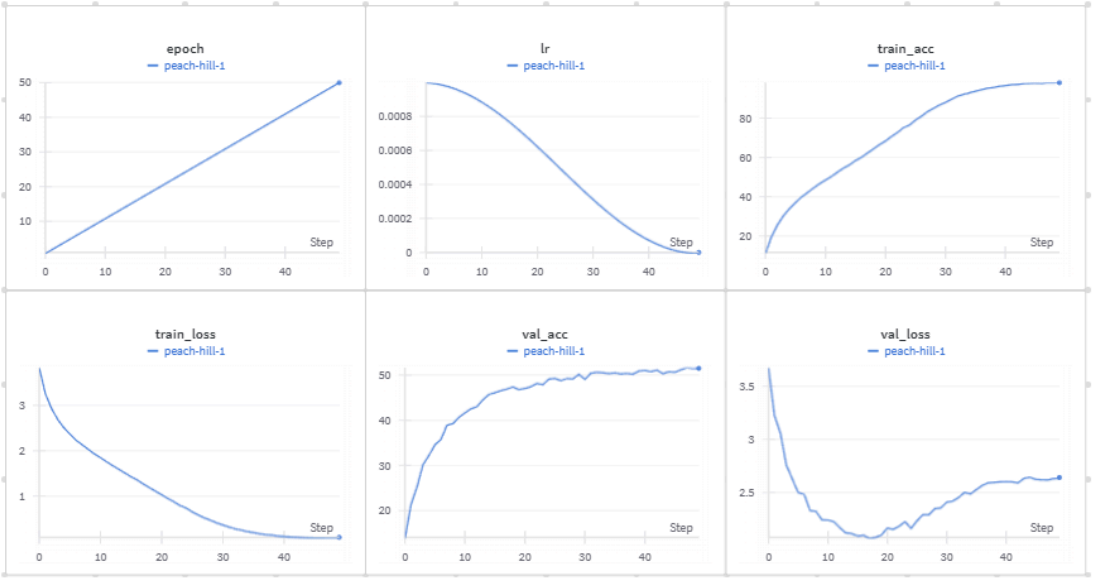
The primary motivation for using ResNet18 lies in its balance between depth and computational efficiency. Its residual structure not only enables training a deeper network but also improves the

network's ability to generalize by facilitating better gradient flow during training. The improvements observed in the training logs—such as a faster decrease in training loss and a higher validation accuracy compared to the simple CNN used in Part 1—underscore the benefits of employing a more advanced architecture.

However, despite these advantages, the model's performance is not without limitations. The experiments revealed that while the ResNet18 model extracts richer features, its performance is still bounded by the current hyperparameter settings and regularization strategies. The validation accuracy, although improved, indicates that further fine-tuning and possibly the use of more advanced models (or transfer learning in Part 3) could lead to even better results. These insights have guided subsequent work and provided a clear pathway for improvement in the later parts of this project.

Hyperparameter Tuning

For hyperparameter tuning, I established a set of key parameters based on preliminary experiments. The model was trained with a batch size of 128 and an initial learning rate of 0.001. I employed the AdamW optimizer with a weight decay of 0.01 to help control model complexity and mitigate overfitting. The training process was carried out for 50 epochs, during which I utilized a CosineAnnealingLR scheduler to smoothly decrease the learning rate over time, thereby promoting stable convergence during the later stages of training. Through extensive experimentation and tracking via WandB, it was observed that the training loss consistently decreased, while the validation accuracy improved significantly compared to the baseline established in Part 1. However, the final accuracy, though improved, still indicated that further optimizations are necessary—a realization that has motivated the plan to employ more advanced techniques in Part 3.



Part3

Model Description

Model Architecture

In Part 3, I leveraged transfer learning by adopting a pretrained ResNet50 model from the torchvision library and adapting it specifically for the CIFAR-100 classification task. Since the original ResNet50 is designed for ImageNet and expects large input images (typically 224×224), I modified the initial convolution layer to use a 3×3 kernel with a stride of 1 and padding of 1, ensuring that the network can process the smaller 32×32 CIFAR-100 images without losing critical spatial details. Additionally, I replaced the subsequent max pooling layer with an identity mapping; this adjustment prevents the early layers from downsampling the images excessively, which is crucial for preserving the fine-grained information present in small images.

Furthermore, I replaced the final fully connected layer with a new linear layer that outputs 100 classes, directly aligning the model's predictions with the CIFAR-100 dataset. To fully exploit the pretrained features while adapting to the new domain, I employed a staged unfreezing strategy. Initially, all layers except the final fc layer

were frozen to allow the classifier to learn the new task without disturbing the established feature representations. Then, during training, I gradually unfroze deeper layers in stages: first, I unfroze layer4, which is closer to the output and responsible for higher-level features; next, I unfroze layer3 to allow for more mid-level feature adaptation; and finally, I unfroze layer2, enabling the network to fine-tune even earlier features. This step-by-step unfreezing strategy helps balance training stability during the early epochs while gradually increasing the model's capacity for adaptation as training progresses.

Hyperparameter Tuning

The hyperparameters were meticulously selected based on a series of preliminary experiments aimed at achieving stable and effective training during the transfer learning phase. Initially, I experimented with training the model for 50 epochs using a batch size of 128 and an initial learning rate of 0.01. However, the results were unsatisfactory, as the model converged too quickly on the training set but exhibited low validation accuracy, suggesting that the pretrained features were not being effectively fine-tuned.

Subsequently, I extended the training duration to 100 epochs in hopes of achieving further improvement. Although this extension allowed the training loss to decrease more significantly, the validation accuracy still failed to meet expectations. These early experiments indicated that the simple freeze-and-unfreeze strategy required further refinement and that a longer training schedule might be necessary to fully adapt the pretrained features.

To address these issues, I divided the training process into two distinct phases. In the freeze phase (3 epochs), only the final fully connected (fc) layer was trained, allowing it to adapt to the CIFAR-100 task without disturbing the robust pretrained features. For the subsequent unfreeze phase (147 epochs, for a total of 150 epochs), layers were gradually unfrozen—first layer4, then layer3, and finally layer2. During the unfreeze phase, I employed the OneCycleLR scheduler with a maximum learning rate of 0.025 and a pct_start of

0.15 to provide an initial warmup period followed by a gradual decay in the learning rate. The AdamW optimizer was used in the freeze phase and maintained throughout the unfreezing process.

These refined settings, validated through extensive WandB logging, resulted in a steady decrease in training loss and improved validation accuracy, although further optimization remains possible.

Hyperparameter Tuning

To mitigate overfitting and enhance generalization, several regularization methods were incorporated. I applied batch normalization after each convolution layer to stabilize and speed up training. A dropout layer with a probability of 0.5 was used in the classifier to reduce overfitting in the fully connected layers. Weight decay was integrated into the optimizer (set to 5e-5 during the transfer learning phase) to constrain the model complexity. Additionally, label smoothing with a factor of 0.1 was used in the loss function to prevent the model from becoming overconfident in its predictions. Furthermore, the MixUp augmentation technique was employed during training to encourage the model to generalize better by blending samples and their corresponding labels.

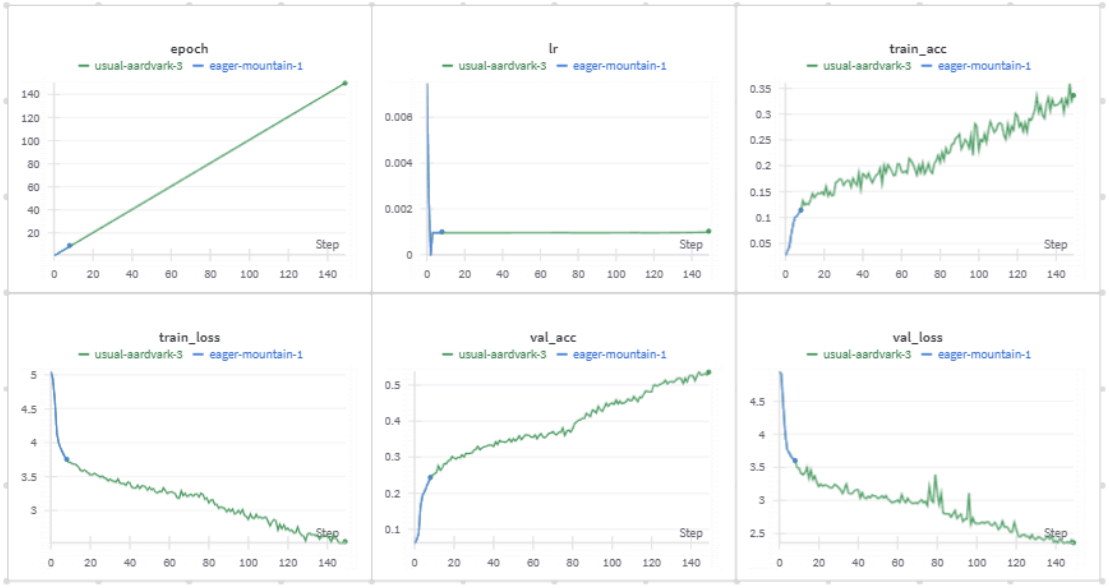
Data Augmentation Strategy

During training, I employed a series of augmentation techniques to significantly improve the model's generalization ability. First, the images underwent a random resized crop that resizes and then randomly crops them to 32×32 pixels, simulating variations in object scale and position. In addition, each image is randomly flipped horizontally with a probability of 0.5, which effectively increases the dataset's diversity. To introduce further variability, I integrated RandAugment, applying one augmentation operation with a magnitude of 7; this addition injects extra randomness without being overly disruptive. I also applied ColorJitter to modify brightness, contrast, and saturation, further enriching the input variations. Finally, the images are normalized using the specified mean and standard deviation, and RandomErasing is applied to randomly mask

portions of an image, thereby enhancing robustness against occlusion. For the testing set, only the essential transformations—conversion to tensor and normalization—are applied to ensure consistency during evaluation.

Results Analysis

The experimental results for Part 3 indicate that the transfer learning approach using the pretrained ResNet50 model has markedly improved performance over the baseline established in Part 1. During training, the loss consistently decreased, and the validation accuracy improved steadily, demonstrating that the pretrained features were effectively adapted to the CIFAR-100 task. However, despite these improvements, the final accuracy suggests that there remains room for further optimization. One strength of the current approach is the robust feature extraction provided by ResNet50's residual blocks, which capture complex patterns in the data. Moreover, the staged unfreezing strategy—where layer4 was unfrozen first, followed by layer3 and then layer2—helped maintain training stability while gradually fine-tuning deeper layers. Effective regularization techniques, including dropout, batch normalization, and weight decay, further contributed to reducing overfitting. On the downside, the learning rate schedule, while generally stable, appears to be sensitive and may benefit from additional fine-tuning or alternative scheduling strategies to fully exploit the model's capacity. These insights pave the way for future experiments in which more advanced hyperparameter tuning and regularization methods will be explored to push the accuracy even higher.



Created with ❤️ on Weights & Biases.

https://wandb.ai/luckykun-boston-university/ds542-part1-cnn/reports/Midterm-report--VmlldzoxMjA1MDYwMA