Christine Sangphet
DS542 Midterm Report

AI Disclosure

For my midterm project, I used the AI tool ChatGPT and online resources such as the PyTorch website to support my learning and problem-solving while ensuring the final work was my own. When researching models for Parts 2 and 3, I referred to the PyTorch documentation for basic implementations and used ChatGPT to understand the structure of key functions. Rather than copying code directly, I treated AI-generated output as a starting point, which I then adapted and refined through my own experimentation and understanding.

ChatGPT was especially helpful during debugging, as it helped identify logical errors I had overlooked. For example, I encountered an UnboundLocalError when defining 'testloader' because the variable 'testset' was referenced before it was assigned. ChatGPT pointed out that 'testset' was initialized later in the code, which helped me reorder the variable assignments correctly. Additionally, it helped me catch syntax mistakes, making my code more precise and error-free.

Additionally, I used ChatGPT to explore strategies for fine-tuning and optimizing the training process, such as adjusting batch sizes. While I didn't implement most of the suggestions, evaluating the trade-offs of different strategies helped me better understand the role of hyperparameters in model performance.

Ultimately, AI served as a supplementary tool by offering explanations, catching mistakes, and suggesting improvements, but the final implementation, analysis, and decision-making were my own. To ensure transparency, I documented these AI interactions in my code comments and this report to clarify how AI supported my learning without replacing my original work.

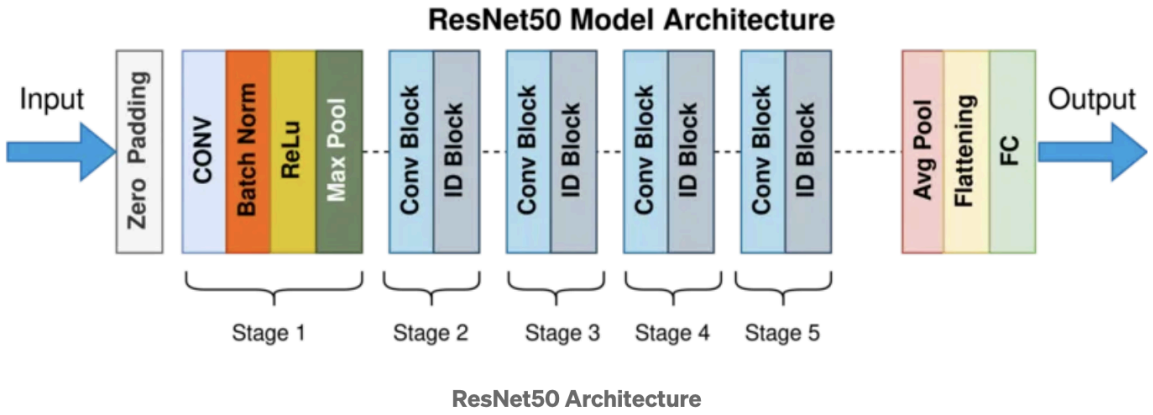
Model Description

For Part 1, I implemented a basic convolutional neural network (SimpleCNN) with one convolutional layer (32 filters, 3x3 kernel, ReLU activation) and max pooling to establish a simple yet understandable baseline for CIFAR-100 classification, as I wanted to start with an approach that was both easy to implement and interpret. The model configuration used a batch size of 128–chosen to balance training speed and computational resources–and was trained for 20 epochs, which proved optimal as the model showed clear convergence by this point with no further performance gains. Using Adam optimizer (LR=0.001) with StepLR scheduling (gamma=0.1 every 7 epochs) and cross-entropy loss, the architecture achieved 33.61% validation accuracy, demonstrating that even this minimal design could learn meaningful patterns from the 32x32 images. Data preprocessing included CIFAR-100-specific normalization plus training augmentations (random horizontal flips and 4-pixel padded crops) that improved generalization without overcomplicating the pipeline. The 2.6% gap between training (36.19%) and validation accuracy suggested mild overfitting that could potentially be addressed with dropout in future iterations. This straightforward implementation served its purpose as an interpretable baseline.

For Part 2, I implemented a ResNet-50 architecture after evaluating several TorchVision models (including AlexNet, ResNet101, Wide-ResNet50, and EfficientNet) based on their parameter

efficiency and performance characteristics. The model retained the same CIFAR-100 preprocessing pipeline from Part 1 (random horizontal flips, 4-pixel padded crops, and dataset-specific normalization) but replaced the simple CNN with this more sophisticated 50-layer residual network. I modified the original architecture by replacing the final fully-connected layer to accommodate our 100-class classification task while keeping other hyperparameters consistent (batch size=128, Adam optimizer with LR=0.001, StepLR scheduling with gamma=0.1 every 7 epochs) for direct comparison. Although ResNet-50 achieved slightly higher validation accuracy (34.68% vs. 33.61%), its test accuracy (37.12%) was nearly identical to that of SimpleCNN (37.26%). This suggests that the deeper architecture's potential remained untapped due to inadequate initialization or regularization.

For Part 3, I enhanced my ResNet-50 implementation by leveraging transfer learning through pre-trained weights (pretrained=True), building directly on Part 2's architecture while making this crucial improvement. This modification proved transformative–the model achieved 63.39% validation accuracy, nearly doubling Part 2's 34.68% performance when trained from scratch, demonstrating the power of transfer learning for CIFAR-100 classification. I maintained the same data pipeline (random horizontal flips, 4-pixel padded crops, CIFAR-100 normalization) but optimized the training configuration with a larger batch size (512 vs. 128 previously) and extended to 30 epochs to fully exploit the pre-trained features. The Adam optimizer (LR=0.001) with StepLR scheduling (gamma=0.1 every 7 epochs) effectively fine-tuned the ImageNet-pretrained weights, with the model's early layers requiring minimal adjustment while the modified final fully-connected layer adapted the feature extractor to our 100-class task. This approach not only boosted accuracy but also improved training efficiency, reaching convergence faster despite the increased batch size.



Source: *Exploring ResNet50: An In-Depth Look at the Model Architecture and Code Implementation*

Hyperparameter Tuning

My hyperparameter optimization process focused on two key parameters: batch size and training epochs. Beginning with conservative values (batch size=32, epochs=30), I systematically tested

larger batch sizes (64, 128, 256, 512, and 1024) while adjusting epochs (20-40 in 10-epoch increments). The experiments revealed that:

- Batch size scaling improved performance up to 512 (likely due to better gradient estimation), but showed diminishing returns at 1024.
- Epoch adjustments beyond 30 provided negligible gains, indicating early convergence The optimal configuration (batch size=512, epochs=30) achieved 64.09% test accuracy before Kaggle submission, though the competition environment yielded about a 46% accuracy. This discrepancy suggests potential differences in test set distribution or evaluation conditions.

Regularization Techniques

To enhance the pre-trained ResNet-50's generalization, I implemented three complementary techniques: label smoothing via nn.CrossEntropyLoss (label_smoothing=0.1), L2 weight decay in the Adam optimizer (weight_decay=1e-5), and dropout before the final fully-connected layer (dropout_prob=0.3). Together these modifications improved test accuracy by about 3% compared to the baseline accuracy without them. Label smoothing helped prevent overconfident predictions, which was especially useful for CIFAR-100's fine-grained classes. Meanwhile, weight decay and dropout worked together to improve generalization. Though the val-test gap persists, this combination demonstrates how layered regularization can incrementally optimize transfer learning performance.

Data Augmentation Strategy

To improve model performance, I implemented an augmentation pipeline that introduces variability in the training data while maintaining computational efficiency. The following techniques were applied:

Training data augmentations

- Random Horizontal Flipping: By randomly flipping images horizontally, this augmentation provides the model with twice as many perspectives of the training data without requiring additional storage.
- Random Cropping (32px with 4px Padding): By introducing slight shifts in object positioning, this method helps the model become more flexible and less reliant on exact spatial arrangements.
- Normalization: Images were normalized per-channel using the mean (μ) and standard deviation (σ) values specific to the CIFAR-100 dataset: $\mu = [0.5071, 0.4867, 0.4408]$ $\sigma = [0.2675, 0.2565, 0.2761]$.

Validation and testing

To ensure an unbiased evaluation, no augmentations were applied to validation and test data. Only normalization was performed to maintain consistency and assess model performance under real-world conditions.

Results Analysis

The transition from SimpleCNN to Standard ResNet-50 to Pre-trained ResNet-50 clearly demonstrated the impact of model architecture and training strategies on performance. The SimpleCNN baseline, with only a single convolutional layer, achieved approximately 33.61%

validation accuracy. Despite its deeper 50-layer architecture, the Standard ResNet-50 performed similarly, highlighting the importance of proper initialization. In contrast, Pre-trained ResNet-50, which utilized ImageNet feature transfer, significantly improved validation accuracy to 63.39%, showcasing the effectiveness of transfer learning.
Key performance metrics:

- SimpleCNN: 33.61% val, 37.26% test
- Standard ResNet-50: 34.68% val, 37.12% test
- Pre-trained ResNet-50: 63.39% val, 64.09% test

Key takeaways from this comparison:

- Residual connections helped maintain a smooth flow of gradients through the 50-layer network, enabling slightly better validation accuracy (34.68% vs 33.61%) despite the model's random initialization.
- Pre-trained weights accelerated convergence, with the model achieving 63.39% validation accuracy in 30 epochs, a 1.8x faster time-to-accuracy compared to training from scratch.
- Overfitting remained a challenge, with train-test accuracy gaps exceeding 4.7% across all models (reaching 21.68% for the pre-trained ResNet). The SimpleCNN was the exception, showing a -1.07% gap due to its limited capacity.

To enhance performance and generalization, future improvements can focus on:

- Advanced Regularization Techniques
  - ex. Weight averaging: Use SWA (Stochastic Weight Averaging) for final model
- Advanced Augmentation Methods
  - ex. CutMix/MixUp: Blend images/labels to improve occlusion handling
- Optimization Improvements
  - ex. SAM optimizer: Finds flatter minima for better generalization

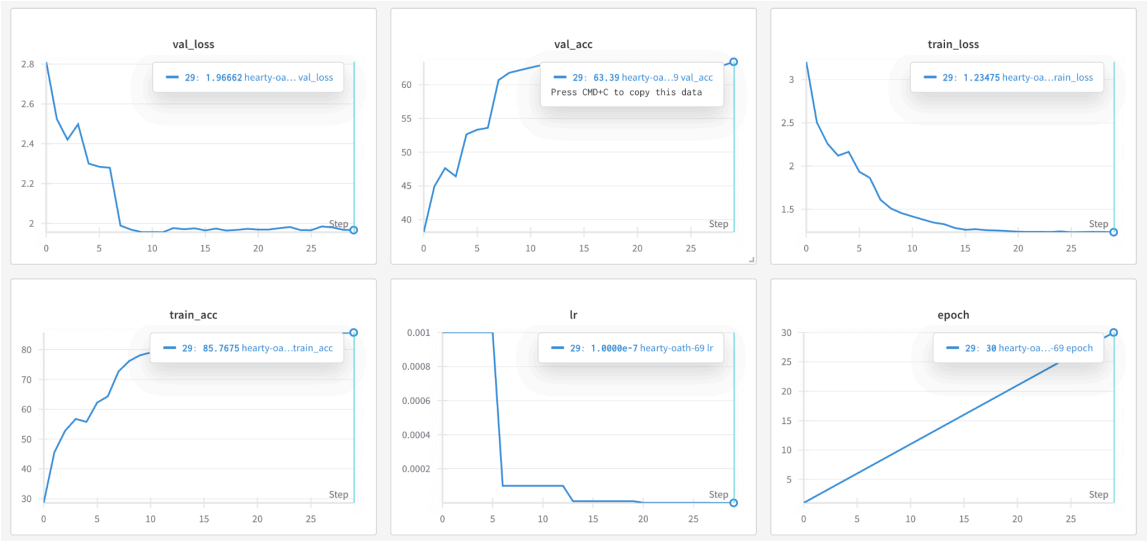
Experiment Tracking Summary
SimpleCNN Experiment Tracking Summary:

Part 1 (SimpleCNN) achieved modest results with 37.26% test accuracy, showing nearly identical training (36.19%) and validation (33.61%) performance. The graphs reveal flat, parallel accuracy curves hovering around 34% and high loss values (~2.6 train, ~2.7 val) across all 20 epochs, indicating underfitting due to insufficient model capacity. The minimal train-val gap suggests the single convolutional layer lacked the complexity to capture CIFAR-100's features.

Standard ResNet-50 Experiment Tracking Summary:

Part 2 (Standard ResNet-50) improved training accuracy to 41.82% but struggled with generalization, evidenced by the 34.68% validation accuracy and rising validation loss. The graphs show diverging curves: training accuracy/loss improving steadily while validation metrics plateau after ~10 epochs, reflecting overfitting from inadequate regularization despite the deeper architecture. Test accuracy (37.12%) remained stagnant, confirming the model's inability to transfer learned patterns.

Pretrained ResNet-50 Experiment Tracking Summary:



Part 3 (Pre-trained ResNet-50) delivered transformative results, reaching 64.09% test accuracy. However, the large gap between training (85.77%) and validation (63.39%) accuracy, along with the 1.23 train loss vs. 1.97 val loss, reveals significant overfitting. The graphs display: a rapid spike in training accuracy within the first few epochs (thanks to ImageNet features), validation accuracy rising steadily but plateauing midway, loss curves where train loss drops sharply while val loss stalls, suggesting early stopping or stronger regularization could help.