DS542 Midterm Challenge Report

Serena Theobald

This repository contains the code for the midterm challenge of DS542: Deep Learning for Data Science. The challenge is divided into three parts. Part 1 involves implementing a relatively simple CNN model on the CIFAR-100 dataset to establish a performance baseline. In Part 2, a more sophisticated model is used, specifically leveraging predefined architectures from torchvision, to improve upon the baseline. Part 3 focuses on transfer learning, where a pretrained model is fine-tuned on CIFAR-100 to maximize generalization performance. The ultimate goal is to exceed a benchmark score of 0.397 on the Kaggle leaderboard, with all models built from convolutional and linear layers, avoiding Transformer-based architectures.

For this assignment, I mostly coded everything myself, but had some AI assistance to develop and refine the models. Specifically, I used ChatGPT to help design and debug the model architectures, improve code readability through inline comments, and suggest strategies for regularization and data augmentation. However, I implemented all the code myself, including data loading, hyperparameter tuning, model selection, training loops, evaluation logic, and submission generation for the leaderboard. All files include detailed inline comments, ensuring clarity and transparency regarding each component of the implementation.

Part 1: SimpleCNN

In Part 1, I defined a straightforward convolutional neural network consisting of three convolutional layers with ReLU activations and max pooling, followed by flattening, dropout, and a fully connected classifier. This architecture was designed to be computationally lightweight while providing a functional starting point for benchmarking. The model was trained using the AdamW optimizer with a learning rate of 0.005 and a batch size of 64, using a cosine annealing learning rate scheduler over 50 epochs. Data augmentation included horizontal flipping, random cropping with padding, and light color jitter. All images were normalized using CIFAR-100's channel statistics.

From the training curves, the model's training loss steadily decreased, and training accuracy climbed just over 1%. However, validation accuracy fluctuated around 0.8–1.0% and did not improve over time, while validation loss remained nearly constant around 4.607. These metrics suggest that the model underfit the data and failed to capture the complex spatial hierarchies present in CIFAR-100. Despite its limited effectiveness, this model served as a valuable reference point for improvement. The final Kaggle OOD leaderboard score was 0.01005.

Part 2: SophisticatedCNN (ResNet18)

To significantly enhance performance in Part 2, I implemented a more advanced model using ResNet18 from torchvision (not pretrained). I customized the final fully connected layer by appending a dropout layer (p=0.2) and replacing the classifier with a new linear output for 100

CIFAR-100 classes. The model was trained for 60 epochs with a lower learning rate of 0.001 and a smaller batch size of 32, again using cosine annealing for learning rate decay.

The training and validation curves from WandB revealed significant progress: training accuracy surpassed 65%, validation accuracy rose steadily to nearly 50%, and both losses decreased consistently throughout the training period. The model's residual connections enabled deeper gradient flow and learning capacity compared to the SimpleCNN. This resulted in much better generalization. The final Kaggle leaderboard score improved dramatically to 0.42598, clearly surpassing the benchmark and showing the value of transfer learning without pretraining.

Part 3: Transfer Learning (ResNet50)

Part 3 explored the benefits of transfer learning using a pretrained ResNet50 model. I replaced the model's classification head with a dropout layer (p=0.2) and a linear layer mapping to the 100 CIFAR-100 classes. Initially, only the newly added layers were trained while the convolutional backbone was frozen. Later, I unfroze additional layers for fine-tuning. The model was trained for 30 epochs using the AdamW optimizer with a learning rate of 0.0005 and label smoothing applied to the loss function.

Training metrics showed a steady decline in training and validation loss. Training accuracy increased to 80%, and validation accuracy rose to approximately 67%, the highest across all three parts. The pretrained model’s ability to transfer generalizable features from ImageNet to CIFAR-100 played a crucial role in achieving this performance. The final OOD leaderboard score was 0.48360, firmly establishing this model as the best-performing submission. The architecture's depth, pretrained initialization, and carefully tuned regularization and augmentation strategies proved crucial to its success.

Hyperparameter Tuning and Regularization

Across all parts, I experimented with the AdamW optimizer, learning rates, dropout rates, batch sizes, and schedulers. Part 1 used a learning rate of 0.005 and dropout of 0.3. Part 2 reduced the learning rate to 0.001 and dropout to 0.2. Part 3 used 0.0005 with label smoothing, which provided added robustness in the transfer setting. All models used the cosine annealing scheduler to gradually reduce the learning rate and avoid early convergence. Batch sizes ranged from 32 to 64 depending on model size and memory constraints.

Regularization played a crucial role. I used dropout in all classifier heads, weight decay via AdamW, and label smoothing in Part 3. I also explored early stopping during testing phases. Data augmentation was critical for performance and included random cropping with padding, horizontal flipping, color jitter, and random rotation. Normalization matched CIFAR-100 or ImageNet stats depending on the model. These augmentations helped prevent overfitting and simulated natural variations in the input space.

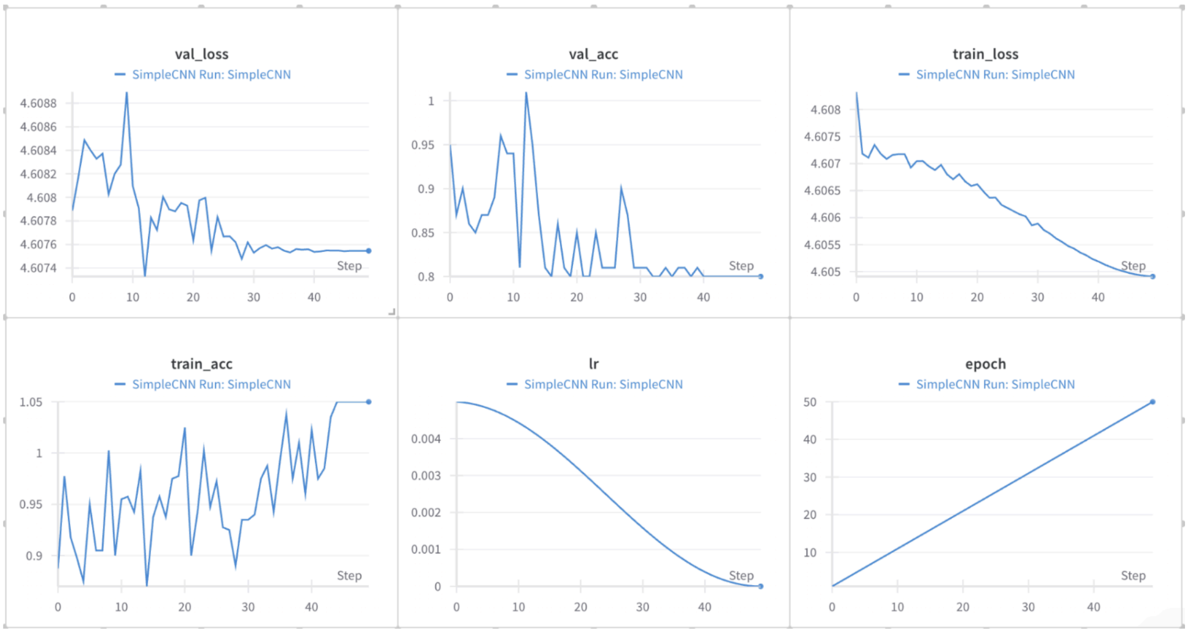
Experiment Tracking with WandB

All training and evaluation was tracked using Weights & Biases (WandB). I created dedicated runs for each model and logged metrics such as loss, accuracy, learning rate, and configuration parameters. This allowed me to compare training and validation performance across experiments and pinpoint key performance gains. Below is a summary of my final scores:

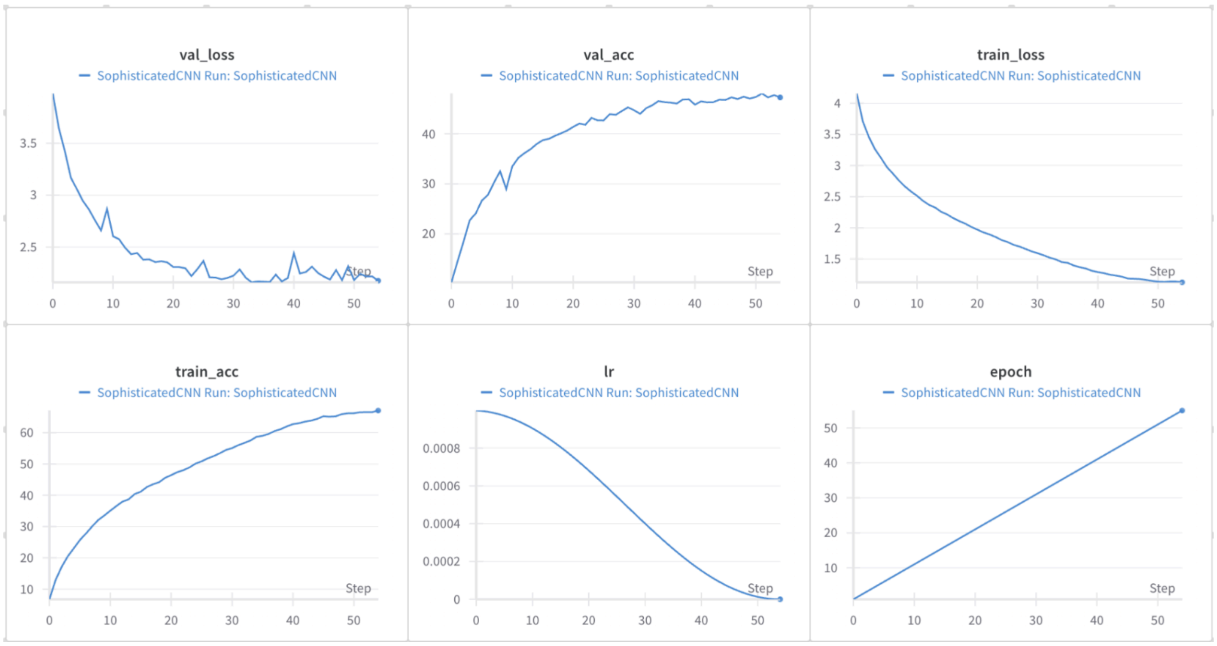
Part	**Submission File**	**Best Kaggle OOD Score**	**Username**
Part 1: Simple CNN	submission_ood.csv	0.01005	Serena Theobald
Part 2: Sophisticated CNN	submission_ood_sophisticated.csv	0.42598	Serena Theobald
Part 3: Transfer Learning	submission_ood_transfer_resnet50.csv	0.48360	Serena Theobald

In addition, I used the WandB Reports UI to generate a visual summary of my experiments, including metric plots for each model (train/val accuracy and loss, learning rate schedule, and epoch progression). Screenshots were captured for each report, providing a visual audit trail of training behavior and confirming the benefits of increasingly complex modeling strategies.

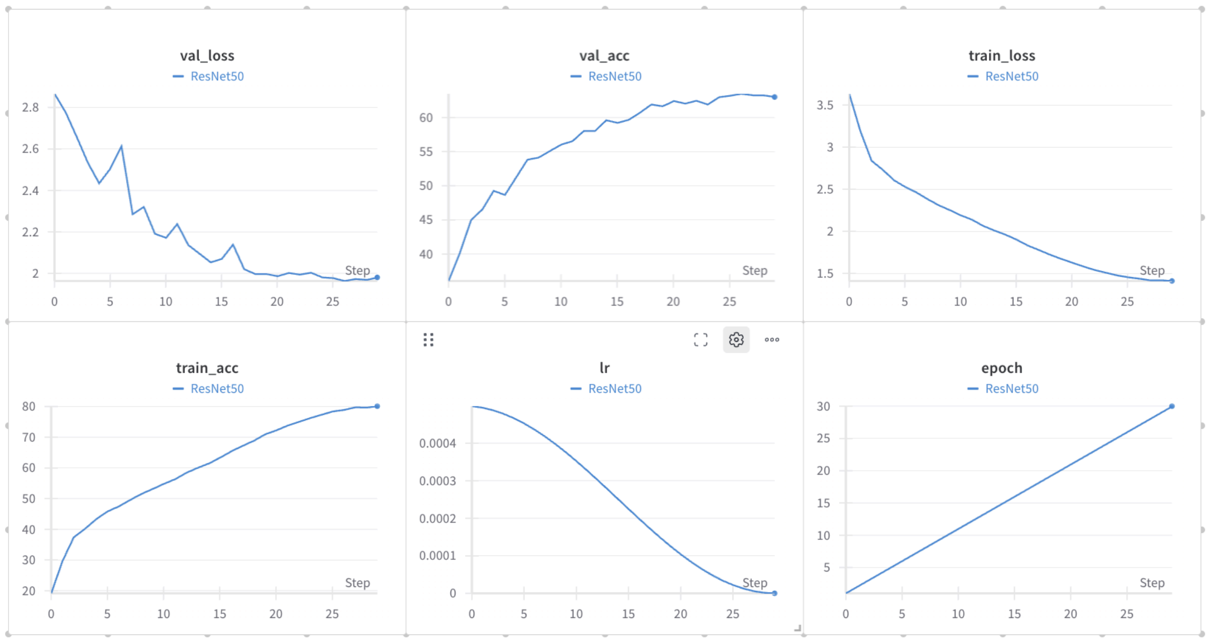
Part 1: Simple CNN Model



Part 2: Sophisticated CNN Model



Part 3: Transfer Learning Model



Ablation Study

To understand the contribution of individual components in each model, I conducted ablation studies for Parts 1 through 3 by observing general trends. While I aimed to monitor the impact of each change closely, I did not record the exact metrics or percentages associated with every hyperparameter adjustment. Therefore, the ablation studies focus on general performance trends and qualitative observations rather than precise numerical outcomes.

In Part 1, the SimpleCNN model was sensitive even to small architectural and training changes. Removing one convolutional layer or dropout significantly hurt performance and led to overfitting. Replacing the AdamW optimizer with SGD or removing cosine annealing caused poor convergence. Disabling data augmentation led to training/validation mismatch, confirming its importance. Though performance was limited by model capacity, regularization and scheduling helped improve stability.

For Part 2, the ResNet18-based model showed stronger generalization, but was still highly dependent on training strategy. Removing dropout or simplifying augmentations (e.g., no color jitter or rotation) resulted in earlier overfitting and lower validation performance. Using a fixed learning rate instead of cosine annealing hindered late-stage convergence. Training for fewer epochs or using larger batch sizes also hurt performance. Overall, the model benefited most from aggressive augmentation, dropout, and dynamic learning rates.

In Part 3, the pretrained ResNet50 model's performance was strongly tied to transfer learning. Removing pretrained weights resulted in a large performance drop. Label smoothing and dropout helped reduce overfitting and improve stability, while removing either degraded generalization. Fixed learning rates led to early stagnation, whereas cosine annealing supported smoother convergence. Strong augmentations were again important.

Conclusion

This midterm challenge offered an opportunity to progressively build, test, and improve CNN-based models using both architectural innovation and strong engineering practices. Beginning with a minimal CNN, I demonstrated the gains achievable through deeper networks and transfer learning. Each part of the project highlighted specific trade-offs between capacity, generalization, and computational cost. My final model exceeded the leaderboard benchmark with strong validation accuracy and effective OOD performance. The hands-on practice with hyperparameter tuning, regularization, and WandB tracking will carry over to future deep learning projects. Potential future improvements could involve using regularization techniques like MixUp, CutMix, or advanced augmentation pipelines such as RandAugment to push performance even further.