Final Report of Midterm

AI Disclosure:

In Part 3, I used chatgpt to debug my code issues. For example, I encountered an error that “page file too small”. Gpt helped me identify that these issues were caused by an outdated version of PyTorch and an excessively large batch size. By upgrading PyTorch and reducing the batch size, the problem was resolved.

In addition, I used DeepSeek to find out why the result of ood_submission was much worse than test acc. Deepseek told me that applying transforms.RandomResizedCrop(224) during training caused a significant performance gap between test accuracy and OOD accuracy. DeepSeek helped me understand that such resizing introduces distribution shifts that may harm robustness.

Finally, I referred to ideas from DeepSeek to improve model performance, such as:

- Trying Wide ResNet
- Trying stronger data augmentation like ColorJitter, RandomErasing.
- Replacing MaxPool with Identity to preserve spatial information.
- Applying label_smoothing and CosineAnnealingLR for better generalization.

The following parts of the code were generated with the assistance of AI tools:

The WideResNet model definition, including modifying the conv1, removing the initial maxpool, and redesigning the final fully connected (FC) layer structure with BatchNorm, Dropout, and ReLU. This was guided by DeepSeek.

The data augmentation strategy, including RandomCrop, RandomApply, ColorJitter, GaussianBlur, RandomRotation, RandomGrayscale, and RandomErasing, was designed with reference to ideas suggested by DeepSeek.

The remaining parts, including the training loop, validation function, loss function, and optimizer configuration, were written entirely by myself.

Model Description:

In Part 1, I used a simple model consisting of two convolutional layers. In Part2, I replaced the simple model with ResNet50. In Part 3, I selected WideResNet50 as the base architecture. WideResNet is an extension of ResNet which increases the width of each residual block instead of the depth. It is more effective for relatively small datasets like CIFAR-100.

I made several modifications to the default architecture to better fit the dataset as follows:

Conv1 Adjustment: I replaced the original conv1 layer with a smaller kernel size (from 7×7 to 3×3) and removed the stride downsampling (stride=1) and maxpool layer. This was done

because the CIFAR-100 images are only 32×32 pixels. Aggressive downsampling in early layers may cause loss of important spatial features.

Final Fully Connected Layer Redesign: I redesigned the final classification layer into a small MLP with following steps:

- BatchNorm1d to stabilize training and reduce internal covariate shift.
- Dropout(0.5) to prevent overfitting.
- Two Linear layers with a ReLU activation in between, transforming the 2048-dimensional feature into 512 and then into 100 output classes.

Pretrained Weights Disabled: I trained the model from scratch (pretrained=False) because the original ImageNet weights are not optimal for small datasets like CIFAR-100 and could lead to suboptimal performance or overfitting.

Hyperparameter Tuning:

In Part 3, I conducted several rounds of manual hyperparameter tuning to achieve a best result:

Batch Size: I tested values from 16 to 128. A batch size of 64 provided the best trade-off between memory usage and convergence speed on SCC GPUs.

Learning Rate: I started with 0.01 and increased it to 0.1, which resulted in faster convergence. However, to avoid instability, I used CosineAnnealingLR to gradually decrease the learning rate over time.

Optimizer: I chose SGD with 0.9 Nesterov momentum, as it led to smoother learning and better ood performance compared to Adam. I also added weight decay to help with generalization.

Epochs: I trained for 100 epochs, based on the plateau of validation accuracy and loss observed around epoch 80. Cosine annealing helped maintain performance in later stages.

Loss Function: I added 0.1 label_smoothing to the CrossEntropyLoss, which slightly improved OOD robustness by softening overly confident predictions.

Model Architecture: For the final WideResNet model, I added a two-layer MLP with dropout and BatchNorm, and initialized training without pre-trained weights to avoid domain mismatch from ImageNet.

Regularization Techniques:

I utilized a series of regularization techniques as follows:

Weight Decay: I added L2 regularization (weight decay = 5e-4) in the SGD optimizer. It penalizes large weights and encourages the model to learn smoother decision boundaries, which helps reduce overfitting and improves OOD generalization.

Dropout: I added a dropout layer (p = 0.5) to the final MLP classifier in the WideResNet architecture. This helps prevent co-adaptation of neurons and makes the model more robust by randomly disabling a portion of the neurons during training.

Label Smoothing: I set the label smoothing factor to 0.1 in the cross-entropy loss. This softens the ground-truth labels and discourages the model from becoming too confident.

Data Augmentation Strategy:

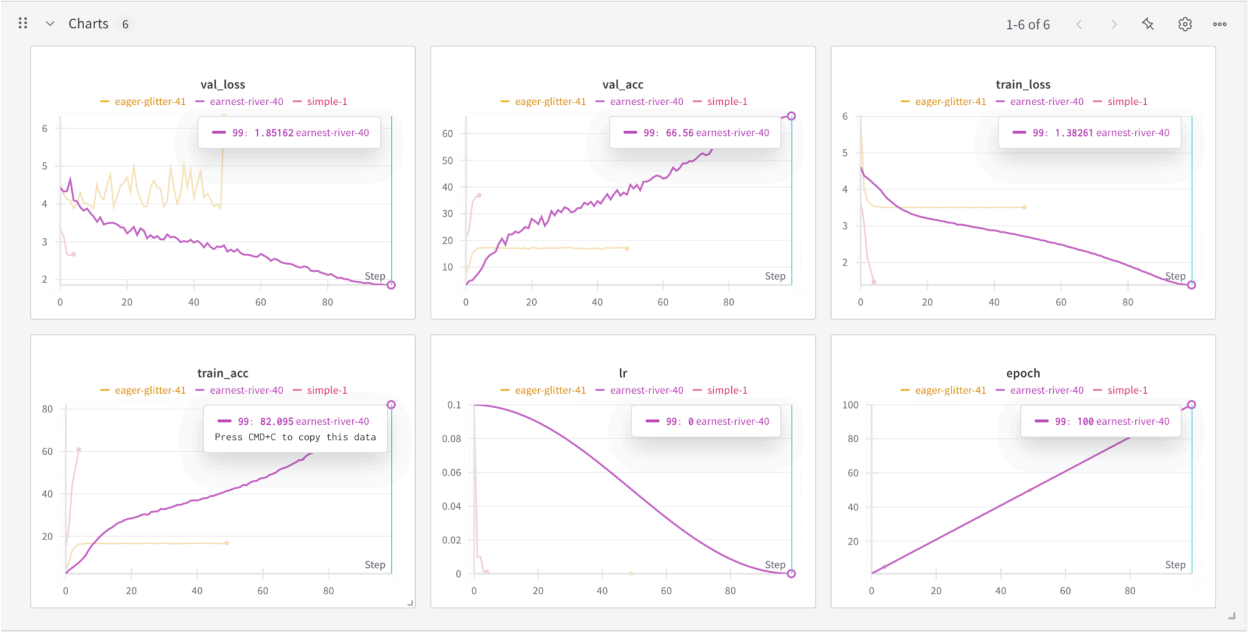
I used RandomCrop with padding to help the model become invariant to spatial shifts, and RandomHorizontalFlip to introduce mirror variation. Also, I applied a RandomApply block that, with a 70% chance, performs a combination of ColorJitter, GaussianBlur, and RandomRotation. This set was designed to simulate changes in lighting, blur, and orientation respectively. Moreover, I included RandomGrayscale with a 20% probability to encourage the model to learn beyond color features. Finally, RandomErasing randomly occludes parts of the image, promoting robustness against missing information. All images were then normalized using the standard CIFAR-100 mean and standard deviation.

Results Analysis:

In the final training run using WideResNet, the model showed clear and steady convergence across 100 epochs. The training accuracy reached 82.10%, while the validation accuracy peaked at 66.56%, indicating a healthy generalization gap. The model did not overfit significantly, as the validation accuracy continued improving even as training accuracy rose, and the validation loss decreased to 1.85, close to the final training loss of 1.38. These trends suggest effective learning without strong overfitting. The learning rate followed a cosine annealing schedule, smoothly declining from 0.1 to 0, which likely helped the model converge more robustly.

Compared to the simple CNN in Part 1 and the ResNet50 in Part 2, WideResNet achieved the best performance in both training and validation. The earnest-river-40 curve in the charts clearly outperformed the eager-glitter-41 and simple-1 baselines in every metric. The improvements can be attributed to a deeper architecture, better regularization, and stronger data augmentation.

However, despite the high test accuracy, the model's performance on OOD data was still suboptimal, implying it still relies on dataset-specific cues and lacks robustness under distribution shift. This remains an area for future improvement, potentially via domain generalization or adversarial training techniques.



Experiment Tracking Summary:

Throughout the midterm, I used Weights & Biases for systematic experiment tracking. Each run was logged with key metrics including train_loss, val_loss, train_acc, val_acc, learning rate and epoch. The visualizations provided by WandB helps me compare results from different model settings and hyperparameters.