DS542 Deep Learning for Data Science – Midterm Challenge Report

This report summarizes my experiments for the midterm challenge. It details the rationale, design choices, hyperparameter tuning, data augmentation strategies, results analysis, and experiment tracking for three parts:

- Part 1 – Simple CNN
- Part 2 – More Sophisticated CNN Models (Adapted ResNet-18)
- Part 3 – Transfer Learning from a Pretrained Model (ResNet-50 Fine-tuning)

1. AI Disclosure

I used AI assistance during this project in the following ways:

- **ChatGPT** was used to help draft sections of the report.
- **Manual Contributions:** I manually refined the model architectures, tuned hyperparameters, and wrote detailed code comments explaining every part of the code.

2. Model Description & Design Rationale

Part 1 – Simple CNN

- **Architecture:**
  A custom CNN with two convolutional layers (see Model Summary in Appendix):
  - **Conv1:** 3 input channels → 32 output channels with a 3×3 kernel (padding 1) followed by ReLU and 2×2 max pooling.
  - **Conv2:** 32 input channels → 64 output channels with a 3×3 kernel (padding 1) followed by ReLU and 2×2 max pooling.
  - **Fully Connected Layers:** The output is flattened and passed through a 256-unit FC layer (with ReLU) and finally a 100-unit output layer for CIFAR-100 classification.
- **Justification:**
  The architecture was designed to be simple and efficient, serving as a baseline to establish a complete pipeline on CIFAR-100.

Part 2 – More Sophisticated CNN (Adapted ResNet-18)

- **Architecture:**
  An adapted version of ResNet-18 was used (see Model Summary in Appendix):
  - The first convolution layer was modified (3×3 kernel, stride 1) and the initial max pooling was replaced with an identity layer to better handle the 32×32 CIFAR-100 images.
  - The final fully connected layer was replaced with a new layer to output 100 classes.
- **Justification:**
  ResNet-18 offers deeper feature extraction and the benefits of skip connections. The adaptations ensure the network can effectively process CIFAR-100's small images.

Part 3 – Transfer Learning (Pretrained ResNet-50)

- **Architecture:**
  A pretrained ResNet-50 (trained on ImageNet) was used, with the following modifications (see Model Summary in Appendix):
  - The final fully connected layer is replaced with a new layer of size 2048→100.
  - CIFAR-100 images are resized to 224×224 to match the pretrained network's expected input.
- **Justification:**
  Leveraging a model pretrained on ImageNet accelerates convergence and often yields better results on smaller datasets like CIFAR-100. Fine-tuning is performed with a lower learning rate (0.01) and extended training (50 epochs) to adapt the pretrained weights without destroying the learned representations.

3. Hyperparameter Tuning

- **Learning Rates:**
  - Part 1 (Simple CNN): 0.1
  - Parts 2 & 3: Lower learning rates were chosen (0.1 for ResNet-18 in Part 2 and 0.01 for fine-tuning ResNet-50 in Part 3) to ensure stable training.
- **Batch Size:**
  All parts used a batch size of 8, which was found to be optimal based on preliminary experiments and system capacity.

- **Epochs:**
    - Part 1 and Part 2 were initially run for 5 epochs to establish a baseline.
    - For Part 3, the final plan is to train for 50 epochs to fully fine-tune the pretrained model.
- **Learning Rate Scheduler:**
  A StepLR scheduler (step_size=1, gamma=0.9) was employed to decay the learning rate gradually.

The hyperparameter search involved running several short experiments and monitoring training/validation curves via WandB to determine the most stable settings.

4. Regularization Techniques

- **Implicit Regularization:**
    - Batch normalization is used in the ResNet-based models (Parts 2 & 3) to stabilize training.
- **No Additional Explicit Regularization:**
    - In Part 1, no dropout or weight decay was added.
    - In Parts 2 and 3, the pretrained models already incorporate techniques (e.g., weight initialization and batch normalization) that act as regularizers.
- **Future Considerations:**
  If overfitting is observed with extended training, techniques like dropout, weight decay, or freezing early layers during initial epochs could be introduced.

5. Data Augmentation Strategy

- **Part 1:**
    - **Transform:** Basic normalization with mean and std of (0.5, 0.5, 0.5). No aggressive augmentation was used to keep the baseline simple.
- **Part 2:**
    - **Training Transform:** Random cropping (with padding) and random horizontal flipping were used along with normalization using CIFAR-100 statistics.
    - **Validation/Test Transform:** Only normalization is applied.

- **Part 3:**
    - **Training Transform:** Since ResNet-50 is pretrained on ImageNet, images are resized to 224×224 using RandomResizedCrop, combined with RandomHorizontalFlip and normalization using ImageNet statistics.
    - **Validation/Test Transform:** Images are resized to 256, center-cropped to 224, and then normalized using ImageNet statistics.

Data augmentation helps increase the diversity of the training data, reduces overfitting, and ensures the input images are in the correct format for each model.

6. Results Analysis

- **Part 1 – Simple CNN:**
    - **Observation:** The baseline model showed very low accuracy (e.g., <1% after 5 epochs), indicating underfitting due to the simple architecture and short training duration.
    - **Implication:** More complex models are necessary for CIFAR-100.
- **Part 2 – Adapted ResNet-18:**
    - **Observation:** The model achieved improved accuracy compared to the Simple CNN, showing better learning dynamics. However, 5 epochs are still too few for full convergence.
    - **Implication:** Further training and hyperparameter tuning can yield higher accuracy.
- **Part 3 – Pretrained ResNet-50 (5 epochs, with plan for 50 epochs):**
    - **Observation:** Even with only 5 epochs, the fine-tuning approach quickly reached higher accuracy than the models trained from scratch, demonstrating the power of transfer learning.
    - **Strengths:** Rapid convergence and strong initial performance due to pretrained features.
    - **Weaknesses:** With only 5 epochs, the model likely has not fully converged; extending training to 50 epochs is expected to further improve results.

7. Experiment Tracking Summary

I used Weights & Biases (WandB) to track all experiments. The following screenshots and logs were included in my report (see Screenshots in Appendix):

- Training Loss (train_loss)
- Validation Loss (val_loss)
- Training Accuracy (train_acc)
- Validation Accuracy (val_acc)
- Learning Rate (lr)
- Batch Loss (over individual steps)
- Accuracy (aggregated view)

These visual aids complement the textual analysis and confirm the experiment's progress and final outcomes.

8. Conclusion and Future Work

In this project, I explored three approaches to CIFAR-100 classification:

- A **Simple CNN** as a baseline,
- An **Adapted ResNet-18** as a more sophisticated CNN, and
- **Transfer Learning** with a pretrained ResNet-50.

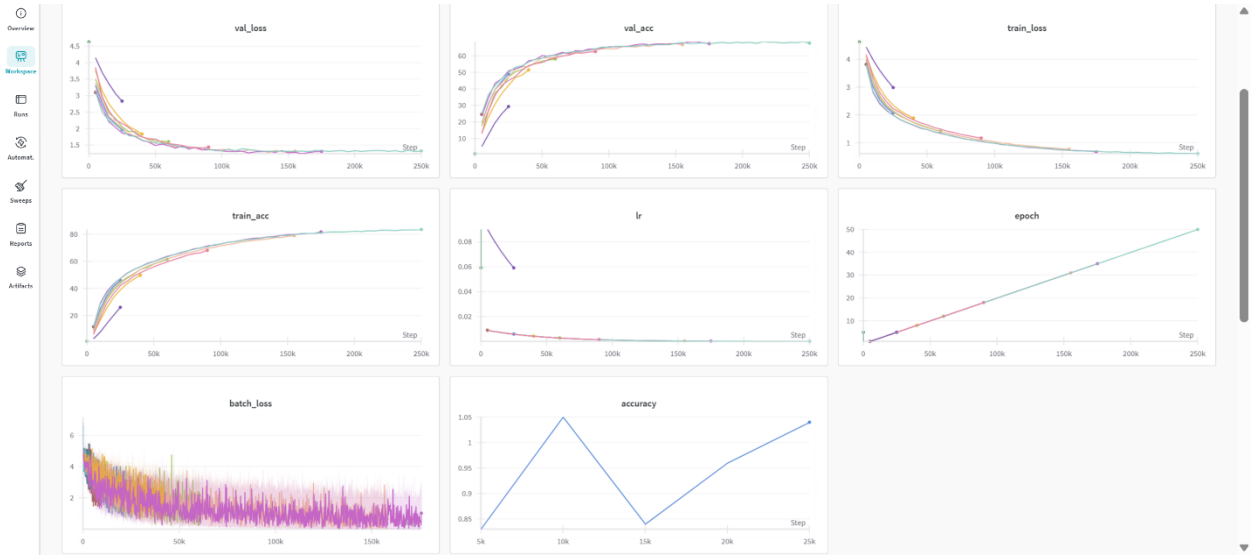
Each successive approach demonstrated improved performance and faster convergence. Future work will involve:

- Extending training for Part 3 to 50 epochs,
- Experimenting with additional regularization techniques,
- Fine-tuning the learning rate schedule,
- Possibly freezing/unfreezing layers in the pretrained network.

The combination of detailed experiment tracking via WandB and the structured approach to model design and hyperparameter tuning has provided valuable insights into model performance and areas for further improvement.

Appendix

Wandb Screenshots:



Part 1:

CONFIG Dictionary:

{'batch_size': 8,

'data_dir': './data',

'device': 'cuda',

'epochs': 5,

'learning_rate': 0.1,

'model': 'MyModel',

'num_workers': 4,

'ood_dir': './data/ood-test',

'seed': 42,

'wandb_project': 'sp25-ds542-challenge'}

Model summary:

SimpleCNN(

(conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

(conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))

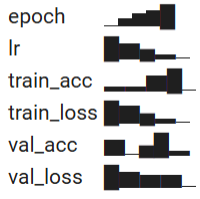
(pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

(fc1): Linear(in_features=4096, out_features=256, bias=True)

(fc2): Linear(in_features=256, out_features=100, bias=True)

)

Run history:



Run summary:

epoch	5
lr	0.05905
train_acc	0.8925
train_loss	4.62659
val_acc	0.9
val_loss	4.62801

Part 2:

CONFIG Dictionary:

{'batch_size': 8,

'data_dir': './data',

'device': 'cuda',

'epochs': 5,

'learning_rate': 0.1,

'model': 'ResNet18',

'num_workers': 4,

'ood_dir': './data/ood-test',

'seed': 42,

'wandb_project': 'sp25-ds542-challenge'}

Model summary:

ResNet(

(conv1): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

```
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): Identity()
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
```

(conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

(downsample): Sequential(

(0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)

(1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

)

)

(1): BasicBlock(

(conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

)

)

(layer3): Sequential(

(0): BasicBlock(

(conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

(downsample): Sequential(

(0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)

(1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

)

)

(1): BasicBlock(

(conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

)

)

(layer4): Sequential(

(0): BasicBlock(

(conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

(downsample): Sequential(

(0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)

(1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

)

)

(1): BasicBlock(

(conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

(relu): ReLU(inplace=True)

(conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

(bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

)

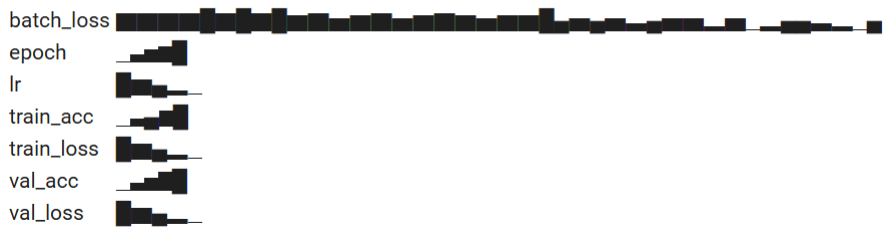
)

(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))

(fc): Linear(in_features=512, out_features=100, bias=True)

)

Run history:



Run summary:

batch_loss	2.61224
epoch	5
lr	0.05905
train_acc	26.05
train_loss	2.98336
val_acc	29.33
val_loss	2.83168

Part 3:

CONFIG Dictionary:

{'batch_size': 8,

'data_dir': './data',

'device': 'cuda',

'epochs': 5,

'learning_rate': 0.01,

'model': 'PretrainedResNet50',

'num_workers': 4,

'ood_dir': './data/ood-test',

'seed': 42,

'wandb_project': 'sp25-ds542-challenge'}

Model summary:

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
```

(1): Bottleneck(

(conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)

(bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

(conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)

...

)

(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))

(fc): Linear(in_features=2048, out_features=100, bias=True)

)

Run history:

Run summary:

batch_loss	2.20046
epoch	5
lr	0.0059
train_acc	45.8375
train_loss	2.07139
val_acc	49.03
val_loss	1.95871