DS542 – Deep Learning for Data Science

Spring 2025 – Midterm Challenge Report

Samritha Aadhi Ravikumar
BU ID: U51656773

Part 1: SimpleCNN
Part 2: Sophisticated CNN
Part 3: Transfer Learning

AI Assistance Disclosure

- **ChatGPT**: Used to
  – Refine explanations of model architectures, fine-tuning strategies, regularization methods, and data augmentation techniques.
  – Improve the clarity, structure, and tone of the written report.
  – Suggest tuning strategies for learning rate schedules and batch size configuration.
- **Grammarly**: Used to
  – Proofread and enhance grammar, punctuation, and writing flow for the final report.
  – Ensure consistency in academic tone and eliminate stylistic errors.
- **GitHub Copilot / Cursor**: *Not used* for this assignment.
- **Code Implementation**: All modeling, training, fine-tuning, evaluation, and W&B integration code was written independently. No AI-generated code was used during development.

Part 1: SimpleCNN – Establishing a Baseline

1.1 Objective

The aim of part 1 was to carry out a performance examination of shallow convolutional neural networks on CIFAR-100 and construct a systematic training flow for future models that will be more powerful. At this phase, I created and trained two architectures of my own to test step by step: a basic 2-layer CNN (SimpleCNN) and a basic 3-layer CNN (SimpleCNN).

1.2 Approach and Progression

I initiated with SimpleCNN, a primitive form of architecture that employs only two basic building blocks in the form of convolutional layers followed by two fully connected layers. Even if this particular model was operating effectively for the purposes of error checking and validation of the training pipeline, its capacity of representation was not enough to deal with the complexity of CIFAR-100.

To solve this problem, I developed the architecture by adding the third convolutional layer and introducing Batch Normalization after each convolutional layer and Dropout before the final classification head. The outcome of this better model, SimpleCNN++, greatly raised the training stability and test performance.

1.3 Model Architectures

- **SimpleCNN (2-layer baseline)**

  Conv2d(3 → 32) → ReLU → MaxPool2d
  Conv2d(32 → 64) → ReLU → MaxPool2d
  Flatten → Linear(4096 → 256) → ReLU → Linear(256 → 100)

- **SimpleCNN++ (3-layer improved)**

  Conv2d(3 → 32) → BatchNorm → ReLU → MaxPool2d
  Conv2d(32 → 64) → BatchNorm → ReLU → MaxPool2d
  Conv2d(64 → 128) → BatchNorm → ReLU → MaxPool2d
  Flatten → Dropout(0.3) → Linear(2048 → 256) → ReLU → Linear(256 → 100)

1.4 Training Configuration

Both models were trained for 20 epochs on CIFAR-100 using a batch size of 64, with an 80/20 split for training and validation. The following configuration was used for each:

Component	**Simple CNN**	**Simple CNN ++**
Optimizer	SGD (momentum=0.9)	Adam
Learning Rate	0.01	0.001
Scheduler	StepLR (step_size=10, gamma=0.1)	
Loss Function	CrossEntropyLoss	
Data Augmentation	RandomCrop(32, padding=4), RandomHorizontalFlip, Normalize	
Regularization	None	Dropout(0.3) before FC, BatchNorm after each Conv

1.5 Results Overview

Model	**Train Acc**	**Train Loss**	**Val Acc**	**Val Loss**	**Test Acc**
SimpleCNN	47.77%	1.9653	41.29%	2.2686	45.67%
SimpleCNN++	50.31%	1.8283	48.43%	1.9257	**52.16%**

1.6 Analysis and Reflections

A 2-layer SimpleCNN model was a key initial point in the program development, which enabled the training process and found out where the data went wrong. Nevertheless, the lack of a deep network was a drawback for the model when dealing with the complex CIFAR-100 dataset. So, by adding one more convolutional layer along with Batch Normalization and Dropout in SimpleCNN++, the situation was almost totally reversed, and both test and training accuracy got a great boost. The model turned out to be more robust in the learning process without becoming a victim of overfitting, thanks, in part, to the augmentation of data. **SimpleCNN++** set a new record with a test accuracy of **52.16%** and a Kaggle score of **0.24442**, clearly exceeding the benchmark and laying down a solid foundation for more advanced models in Part 2.

Part 2: More Sophisticated CNN Models

2.1 Objective

Part 2 was intended to go beyond the basic models by adding more robust and deeper convolutional architectures using torchvision.models. The main goal was to expand the model capacity and also the training dynamics by utilizing the well-known, commonly used backbones and evaluating their performance on CIFAR-100.

2.2 Approach and Progression

I started with part 2 with ResNet18, a more complex model with lateral connections that allow the gradient to flow more smoothly. For the 100-class version, I changed the final layer only and trained it

with AdamW, label smoothing, and cosine LR scheduling from scratch. The model showed a clear improvement over SimpleCNN++, proving to be reliable, easy to train, and a strong first attempt among deeper architectures.
Then, I used DenseNet121, which is a neural network architecture that connects each layer to every other layer in a feed-forward fashion to improve feature reuse. The model which I trained using the same training configuration as ResNet18 was able to converge and reach the best possible accuracy faster. It outperformed all previous models on both validation and test accuracy, confirming the advantages of dense connectivity and deeper networks for CIFAR-100.

2.3 Model Architectures

- **ResNet18**

  [Conv → BN → ReLU → MaxPool]
  → Residual Block × 4 (2 blocks each)
  → Global AvgPool
  → Linear(512 → 100)

- **DenseNet121**

  [Conv → BN → ReLU → Pool]
  → DenseBlock × 4 with Transition Layers
  → Global AvgPool
  → Linear(1024 → 100)

2.4 Training Configuration

Both models were trained for 30 epochs on CIFAR-100 using a batch size of 64, with an 80/20 split for training and validation. The following configuration was used for each:

Component	**ResNet18 and DenseNet121**
Optimizer	AdamW
Learning Rate	0.001
Scheduler	CosineAnnealingLR
Loss Function	CrossEntropyLoss (label_smoothing=0.1)
Data Augmentation	RandomCrop, HorizontalFlip, ColorJitter, RandomRotation, Normalize
Regularization	Weight Decay (1e-4)

2.5 Results Overview

Model	**Train Acc**	**Train Loss**	**Val Acc**	**Val Loss**	**Test Acc**
ResNet18	73.78%	1.6341	49.05%	2.5379	52.87%
DenseNet121	82.17%	1.4123	52.42%	2.4265	**56.00%**

2.6 Analysis and Reflections

ResNet18 served as a solid entry point into deeper architectures. The residual connections in it were good for the stable training, and the model always performed better than my previous CNNs. Such deeper networks are more capable because of denser layers and can do even better than pre-training but still very good. It trained well and achieved strong accuracy, validating the benefits of deep networks without pretraining. DenseNet121 was the one that took the highest position in this competition. Since its layers are densely connected, which leads to feature reuse and gradient flow improvement, the model had a quicker training process and the accuracy was higher when testing. The test accuracy for **DenseNet121** was the highest, with **56.00%** and also it scored **0.34195**, in Kaggle. Therefore, DenseNet121 was deemed to be the strong basis for transfer learning in Part 3.

Part 3: Transfer Learning – Leveraging Pretrained Backbones

3.1 Objective

Part 3 focused on enhancing model performance and generalizability via the use of pretrained convolutional neural networks. Compared to the prior parts where models were trained from scratch, the goal was to employ high-performing pre-trained ImageNet backbones such as DenseNet121 and ConvNeXt-Tiny, and then fine-tune them on CIFAR-100. The objective was to boost in-distribution accuracy and out-of-distribution (OOD) robustness, culminating in the optimization of the Kaggle leaderboard score through advanced regularization and inference methods.

3.2 Approach and Progression

Looking at the performance of DenseNet121 in Part 2, I decided to retain the same architecture for Part 3 but this time with pre-trained ImageNet weights. I began by training only the classifier head (linear probing) for 10 epochs. Then I continued with full fine-tuning from the pre-trained DenseNet121 checkpoint in incremental stages: 20, 30, 40, and 50 epochs. This two-stage training procedure allowed the model to gradually adapt to the CIFAR-100 distribution. Clean test accuracy was progressively better across stages, with Epochs 30 and 50 performing the best.

Motivated by DenseNet121's trend performance, I performed experimentation on ConvNeXt-Tiny – a recent high-capacity architecture. I trained the model for 30 epochs from pre-trained weights, using MC Dropout and temperature scaling (T=1.5) at test time for improved calibration. The model achieved high performance, so I extended training to 50 epochs (continued from the checkpoint at Epoch 30). Both ConvNeXt-Tiny runs surpassed previous models, offering significant improvements in accuracy and leaderboard score.

3.3 Model Architectures

- **DenseNet121 (Pretrained)**

  [Conv → BN → ReLU → Pool]
  → DenseBlock × 4 with Transition Layers
  → Global AvgPool
  → Dropout(0.5)
  → Linear(1024 → 100)

- **ConvNeXt-Tiny (Pretrained)**

  [Conv → LayerNorm → GELU]

→ ConvNeXt Block × 4 stages
→ Global AvgPool
→ LayerNorm
→ Linear(768 → 100)

3.4 Training Configuration

Both pretrained models were fine-tuned on CIFAR-100 using a batch size of 64, with an 80/20 split for training and validation. The following configuration was used for each:

Component	**DenseNet121**(pretrained)	**ConvNeXt-Tiny**
Epochs	50 (in 5 phases)	30, 50
Pretrained Weights	ImageNet1K	
Optimizer	AdamW	
Learning Rate	0.001	0.001 (Ep30); 5e-5 (Ep50)
Scheduler	CosineAnnealingLR	
Loss Function	CrossEntropyLoss (label_smoothing=0.1)	
Data Augmentation	RandomCrop, HorizontalFlip, ColorJitter, RandomRotation, Normalize	
Regularization	Weight Decay (1e-4), Dropout(0.5)	

3.5 Results Overview

DenseNet121 showed steady improvement across fine-tuning phases, while ConvNeXt-Tiny performed strongly even with fewer epochs.

Model	**Epochs**	**Train Acc**	**Train Loss**	**Val Acc**	**Val Loss**	**Test Acc**
DenseNet121 (pretrained)	10	46.97%	2.47943	45.58%	2.49784	49.44%
	20	52.975%	2.27871	49.95%	2.36479	53.37%
	30	57.295%	2.14188	52.3%	2.29802	**55.57%**
	40	55.895%	2.19031	50.7%	2.3531	54.33%
	50	63.525%	1.95209	54.77%	2.23584	**57.83%**
ConvNeXt -Tiny	30	99.025%	0.8167	64.86%	2.25149	**67.43%**
	50	99.89%	0.78729	64.84%	2.32938	**67.89%**

3.6 Analysis and Reflections

The fine-tuning of **DenseNet121** across different stages was a successful strategy. From the pre-trained weights, the model was trained for 10, 20, 30, 40, and 50 epochs. With each iteration, there was always an increase in accuracy as well as loss. At **epoch 30**, the model provided a test accuracy of **55.57%** and a Kaggle leaderboard of **0.46325**. Further training through **epoch 50** improved the test performance to **57.83%**, and the terminal Kaggle performance of **0.43633** ensured adequate generalization. Such advancements justified the goodness of dense connection in improving feature reuse and permitting effective gradient propagation along deeper layers.

Following the improvement, I further ventured to evaluate the **ConvNeXt-Tiny**—a latest generation convolutional network whose prowess and scalability have triggered lofty expectations about its performance potential. I added MC Dropout and Temperature Scaling (T = 1.5) at inference time to improve calibration and OOD robustness. ConvNeXt-Tiny was trained for **30 and 50 epochs** with the corresponding test accuracies being **67.43%** and **67.89%** respectively. These runs achieved the highest Kaggle leaderboard scores of **0.52291** and **0.52028**, easily outperforming all our previous models. This confirmed that ConvNeXt-Tiny, with uncertainty-aware inference, was the optimal solution to this challenge.

Conclusion

This mid-term issue had a comprehensive tour of CNN-based image classification, from baseline models like SimpleCNN to state-of-the-art pre-trained architectures like DenseNet121 and ConvNeXt-Tiny. With each step, the performance got better and better with better architectures, regularization, and tuning habits. DenseNet121 was a strong transfer learning baseline, and ConvNeXt-Tiny achieved the best test accuracy and Kaggle score. In total, this rendered my deep learning workflow skills, model generalization, and the practical efficacy of calibration and fine-tuning practices sharper.

WandB Run Links

https://wandb.ai/samritha-boston-university/sp25-ds542-challenge?nw=nwusersamritha