MIDTERM CHALLENGE REPORT

Viktoria Zruttova

03/27/2025
Deep Learning For Data Science DS542

AI DISCLOUSRE

I utilized AI assistance, specifically the free version of ChatGPT. It helped with my understanding of the model architecture, data augmentation techniques, and training loop implementations. Detailed code comments, data transformation strategies, and the implementation of regularization methods were written collaboratively with ChatGPT. I wrote the initial model setup, hyperparameter tuning strategies, and the overall training and evaluation loop, while AI support helped me with understanding and provided mostly debugging support.

MODEL DESCRIPTION

Part 1: Simple CNN

For Part 1 of the project, I designed and trained a simple CNN from scratch on the CIFAR-100 dataset. The architecture consists of 3 convolutional layers with 32, 64, and 128 filters, respectively, each followed by a ReLU activation and 2x2 max pooling to reduce spatial dimensions. After the convolutional layers, the output is flattened and passed through a fully connected layer with 256 units and another ReLU activation, followed by a final output layer with 100 units corresponding to the 100 CIFAR-100 classes.

The model was trained using the Stochastic Gradient Descent (SGD) optimizer with a learning rate of 0.01 and momentum of 0.9. A learning rate scheduler (StepLR) was used to reduce the learning rate by a factor of 0.9 after every epoch. The training was done for 5 epochs using a batch size determined dynamically using a batch size finder. The loss function used was cross-entropy loss, which is suitable for multi-class classification tasks.

In terms of regularization, this model did not use dropout, weight decay, or data augmentation.

Overall, this simple CNN served as a baseline, achieving a Kaggle score of **0.16158.**

Part 2: ResNet18

In Part 2, I significantly upgraded the model architecture by using ResNet18, a deeper and more powerful CNN compared to Part 1. I initialized the model from scratch to train directly on the CIFAR-100 dataset. To adapt ResNet18 for this task, I replaced the final fully connected layer with a custom head consisting of a dropout layer (p=0.5) followed by a linear layer outputting 100 classes. The dropout layer serves as a regularization technique to reduce overfitting by randomly deactivating neurons during training.

Images were resized to 224x224, matching the input dimensions expected by ResNet18. I also applied data augmentation to improve generalization (random horizontal flips, random cropping with padding, and color jittering for brightness, contrast, saturation, and hue). The test and validation data were only resized and normalized.

For optimization, I used the AdamW optimizer, which combines the fast convergence of Adam with weight decay (1e-4) to help penalize large weights, further reducing overfitting. I used a CosineAnnealingLR scheduler to smoothly decrease the learning rate over epochs, allowing the model to explore larger updates at first and fine-tune gradually as training progressed. The model was trained with a batch size of 128, a learning rate of 0.001, and for 5 epochs. This setup achieved a Kaggle score of **0.30093**, showing a clear performance improvement over the baseline model in Part 1.

Part 3: ResNet5

For Part 3, I chose **ResNet-50**, a deep convolutional neural network pre-trained on ImageNet. My main reason for selecting ResNet-50 was its depth and proven ability to learn hierarchical features, which I believed would allow it to generalize better on a complex dataset with 100 diverse classes. Additionally, I wanted to **leverage transfer learning**. Since ResNet-50 was pre-trained on a large-scale dataset, I expected it to provide better performance even with limited epochs by adapting the learned weights to the new task. I initially **froze all but the last 50 params** to retain the learned representations in the early layers while allowing the deeper layers to adapt to CIFAR-100. The final fully connected layer was replaced with a **Dropout (0.5)** layer, a **BatchNorm1d** layer for better convergence, and a **Linear layer** outputting 100 classes.

HYPERPARAMETER TUNING

I selected these values based on experimentation and practical recommendations from the internet/ChatGPT. I did not use a full grid or Bayesian search due to time and compute constraints, but I played around with the learning rate, smoothing factor, weight decay, and batch size.

what	value	why
Batch size	32	chosen based on memory constraints and stable gradient estimates
Learning rate	0.001	selected after testing smaller and larger values. Smaller values converged too slowly, and larger ones led to instability.
Epochs	200	with early stopping applied (patience = 40 epochs) to avoid overfitting and long training
Optimizer	AdamW	handles adaptive learning rates and includes built-in weight decay for better regularization
Scheduler	2 phases	**Linear warmup** to gradually increase learning rate from a small value. **CosineAnnealingWarmRestarts** to cyclically decay the learning rate and help the model escape local minima.

REGULARIZATION TECHNIQUES

The primary regularization methods include:

1. **Dropout (50%)** in the fully connected layer, reducing overfitting.
2. **Weight Decay**: Set at 1e-4 through AdamW optimizer to penalize large weights.
3. **Early Stopping**: Implemented to halt training upon validation loss stagnation, preventing overfitting and unnecessary computation. These techniques
4. **Dropout (50%)** before the final fully connected layer to randomly disable neurons

during training.

5. **Batch Normalization** before the final classification layer to stabilize learning.
6. **Label Smoothing** with CrossEntropyLoss (set to `0.1`) to reduce overconfidence in predictions.
7. **Gradient clipping** with a max norm of 1.0 to prevent exploding gradients during backpropagation.

DATA AUGUMENTATION STRATEGY

I noticed that image augmentation played a huge role in improvement from part 2.

For part 3, in the first 30 epochs, I did basic data augmentation. Then, for the model to increase its diversity, I added stronger augmentation. Data augmentation included these transformations:

- Random cropping, horizontal flipping, and rotation for spatial variability.
- Color jitter for robustness against lighting and color variations.
- Random erasing as an additional regularization method. These augmentations significantly improved the model's ability to generalize by artificially expanding the dataset diversity.
- I used the **Mixup** technique during training, which blends input images and their labels. This helps improve generalization and model robustness. I found that Mixup **helped slightly.**

I aimed to increase diversity in the training images after the model had already started to converge. The stronger augmentation technique helped slightly reduce overfitting but only led to **small improvements in accuracy**.

RESULT ANALYSIS

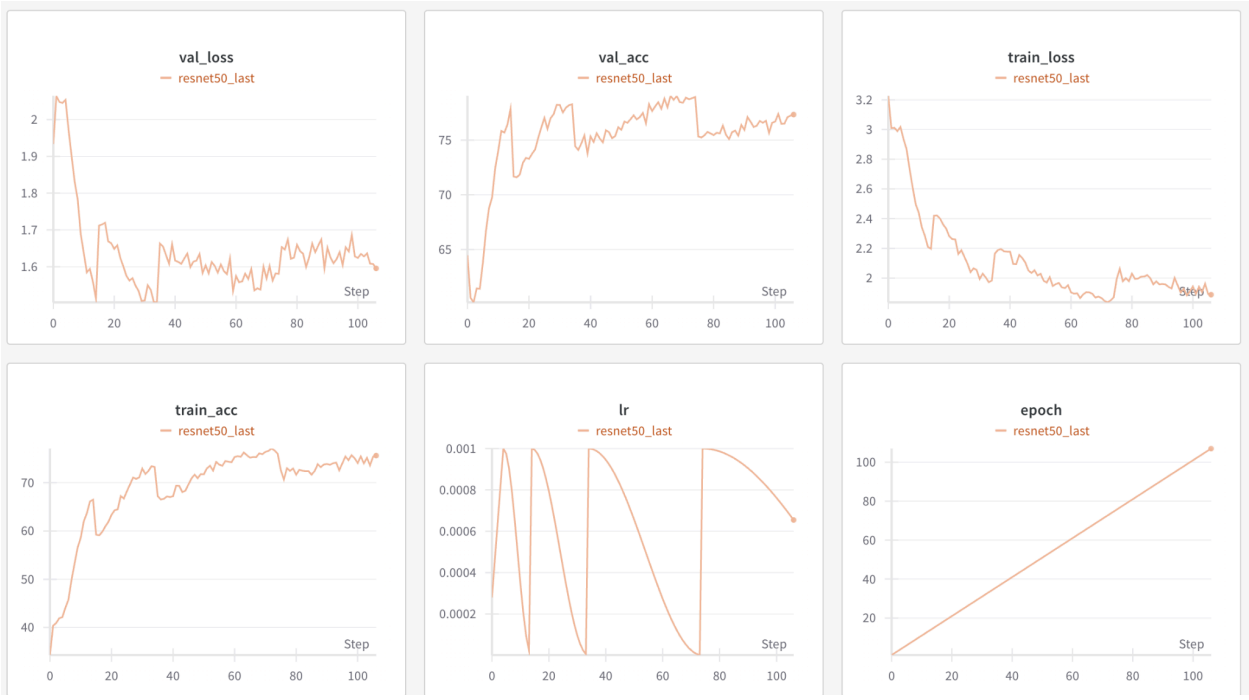
This model achieved the **best Kaggle score** (0.57876) out of all three parts.
ResNet-50 was able to learn **complex hierarchical features** thanks to its deep architecture and transfer learning. In my training process, I used the **AdamW** optimizer over **SGD** because it adapts the learning rates for each parameter, decouples weight decay for better regularization, and converges faster, which is particularly beneficial for training complex models like ResNet50. Regularization and strong augmentation helped

the model perform well on validation and unseen test data. However, after a lot of fine-tuning and different augmentation techniques, I only saw small improvements in the accuracy.

The validation accuracy consistently improved, but the model appeared to plateau after a certain point. While **CosineAnnealingWarmRestarts** as a learning rate scheduler helped with smoother convergence, I had difficulty fine-tuning a large model like ResNet-50 to achieve better results. Other limitations were longer training time and possible overfitting. In the future, I could try to unfreeze more layers gradually or try hyperparameter tuning via a grid or Bayesian search.

EXPERIMENT TRACKING

All training and evaluation runs were tracked using **Weights & Biases (wandb)**. I logged metrics such as training/validation accuracy, loss, and learning rate for every epoch. This made it easier to visualize training curves, detect overfitting, and compare different runs. WandB also stored the best model checkpoints and provided interactive plots.



The first metric, **validation loss**, begins with a sharp drop, which indicates that the model quickly learns to minimize errors. However, after the initial decrease, the curve

fluctuates, suggesting that while the model is improving, it faces some instability during later training steps because of the cyclic learning rate. The final value of the validation loss hovers around 1.6, which reflects the model's struggle to improve more. In contrast, the **validation accuracy** graph displays a steady upward trend, with some minor plateaus. This consistent improvement in accuracy suggests that the model is effectively generalizing to unseen data, even though the rate of improvement slows down toward the end. By the conclusion of the training, the model achieves an accuracy of about 75%, which indicates reasonable success in learning the task at hand. The Mixup augmentation used during training likely helps the model generalize better to unseen data.

The **train_loss** has a dramatic decrease early on as the model adjusts its weights to fit the training data. It becomes more stable over time. This is expected since the model has already learned to generalize well after the first few epochs. The **training accuracy** follows a similar trend, steadily increasing over time and reaching nearly 70%.

The **learning rate (lr)** graph shows an oscillating pattern because of the cyclic learning rate scheduler. The validation loss and accuracy graphs show smoother and more stable progress over time compared to the steep LR scheduler I used before.

KAGGLE

Name: Viktoria Zruttova

PART	KAGGLE SCORE
1	0.16158
2	0.30093
3	0.57876