Midterm Challenge

Ziqi Tang <ztang20@bu.edu>

- **AI Disclosure:**

I used ChatGPT to figure out the parameters that needed to be modified during the implementation of predefined models. When fine-tuning the model, I referred to it for hyperparameter values that might fit our dataset.

1. Model Description

In this experiment, I tuned a pretrained **ResNet50** model on the CIFAR-100 dataset. ResNet50 is a deep convolutional network with residual connections. The following modifications were made:

- **Final Layer:**
  The original fully connected (fc) layer of ResNet50 (designed for 1000 ImageNet classes) was replaced by a new linear layer outputting 100 classes, which is more suitable for CIFAR-100.
- **Dropout Regularization:**
  Although the original ResNet50 does not include dropout in the final fc layer, I chose to wrap it with a dropout layer (dropout rate tuned as a hyperparameter) to further regularize the network. Specifically, I replaced the fc layer with a Sequential block that includes a Dropout layer followed by a Linear layer. This helps mitigate overfitting.

2. Hyperparameter Tuning

First step is to find the optimal batch size by running the batch size finder given in the starter code. Then I did a grid search over three hyperparameters for the Adam optimizer:

- **Learning Rate (lr):** Tested values: 0.001, 0.0005, 0.0001.
- **Weight Decay:** Tested values: 1e-5, 1e-4, 1e-3.
- **Dropout Rate:** Tested values: 0.1, 0.2, 0.3.

Process:

For each (lr, weight_decay, dropout rate) combination, the ResNet50 model was fine-tuned for 5 epochs on the training set. (I created a modified fc layer that includes each corresponding dropout rate.)

Then, the model was evaluated on a validation dataset, and the validation accuracy was computed after each short training run.

Then I selected the hyperparameter set giving the highest validation accuracy as the final choice for full training of 50 epochs.

The best hyperparameter combination was determined to be:

- **Learning Rate:** *best_lr* (0.0001)
- **Weight Decay:** *best_weight_decay* (0.001)
- **Dropout Rate:** *best_dropout_rate* (0.2)

3. Regularization Techniques

To improve generalization and reduce overfitting, the following regularization methods were employed:

- **Weight Decay:** This parameter is incorporated directly into the Adam optimizer. It adds an L2 penalty on the network weights to discourage the network from learning too large weights, thereby can help prevent overfitting.
- **Dropout:** I also added dropout before the final fc layer (and tested dropout rates of 0.1, 0.2, 0.3) to randomly deactivate neurons during training.
- **Early Stopping**: I also implemented an early stopping mechanism to stop training if the validation loss does not improve for 7 consecutive epochs, preventing overfitting to the training data.

4. Data Augmentation Strategy

To enhance the generalization ability of the model, I applied the following data augmentation techniques during training:

- **Random Crop:** Images are randomly cropped to 32×32 after a padding of 4 pixels. This is to simulate slight translations and improve spatial invariance.
- **Random Horizontal Flip:** Images are randomly flipped horizontally, increasing the effective size and diversity of the dataset.
- **Normalization:** Both training and test images are normalized using CIFAR-100 mean and standard deviation, ensuring that the data distributions are centered, scaled appropriately, and stable.

For the validation and testing process, no augmentation was applied. I only normalized the data to ensure consistent evaluation.

5. **Results Analysis** (for Part 3)

1. Model Results

a) Validation Loss (val_loss)

The validation loss starts relatively high (around 2.2) and quickly drops in the first few epochs, indicating that the model is learning effectively. After approximately 8 to 10 epochs, it slightly fluctuates. Then the val_loss increases consecutively and triggers the early_stopper.

b) Validation Accuracy (val_acc)

The validation accuracy shows a steep climb initially and then continues to rise more slowly, eventually leveling off around 63-64%. This indicates the model's generalization on the validation set has mostly stabilized.

c) Training Loss (train_loss)

The training loss falls quickly from the first epoch and continues to decline steadily during training, reaching a much lower level than the validation loss. The gap between training and validation losses by the later epochs suggests some overfitting, but I have used regularization techniques to help alleviate this problem.

d) Training Accuracy (train_acc)

The training accuracy starts around 30% and increases to about 92-93% by the final epochs. Similarly, the difference between the final training accuracy (~93%) and validation accuracy (~63%) indicates slight overfitting.

e) Learning Rate (lr)

Due to a learning rate scheduler, the learning rate drops at epoch 17, which helps fine-tune the model more gently.

f) Training vs. Validation Trends:

Although training accuracy steadily increases, the validation loss starts to rise around the seventeenth epoch, indicating early signs of overfitting, while both training and validation accuracy continues to improve.

2. Potential Improvements

a) Longer Training Epochs

As the validation accuracy is still slightly rising at epoch 20s, I consider extending training further.

b) Regularization Adjustments

We can also consider adding label smoothing to address the concern of overfitting.

c) Data Augmentation

I also tried augmentations like ColorJitter, but it would result in higher val_loss. However, other data augmentation methods are worth considering.

d) Architecture or Hyperparameters

I tried ResNet18, and it is doing considerably well compared to ResNet50. However, a more efficient architecture (e.g., EfficientNet-B0) might yield better results on this specific dataset CIFAR-100.

6. Experiment Tracking Summary

WandB Integration:

All experiments were tracked using [Weights & Biases (WandB)]. The tool was used to log:

- Training and validation losses
- Training and validation accuracies
- Learning rate values per epoch

Here I attached some screenshots from WandB for each part of the challenge.

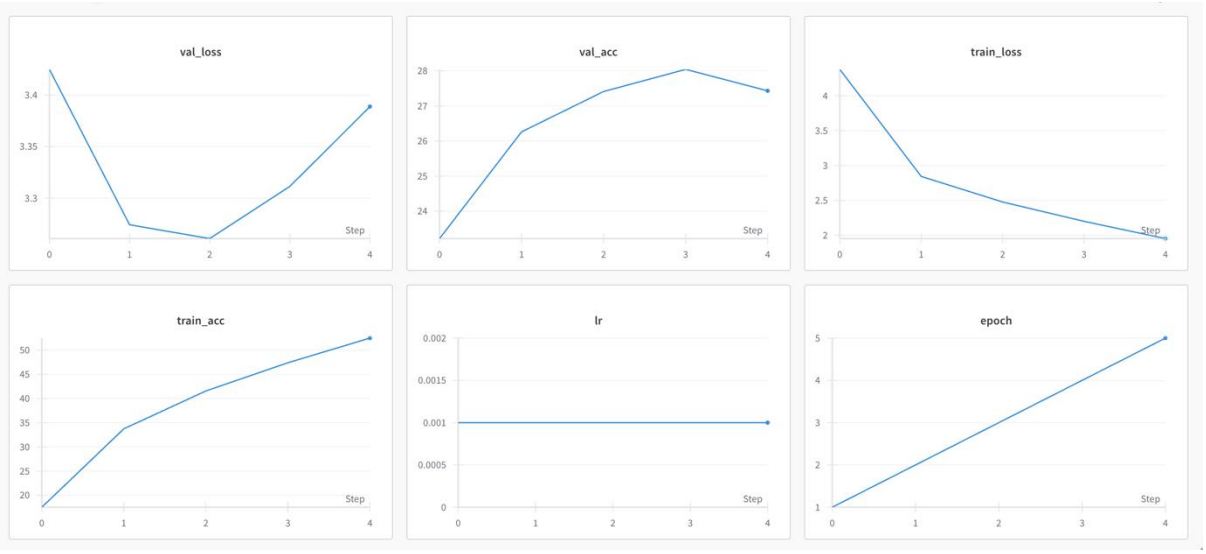

Figure 1. Part 1 Results

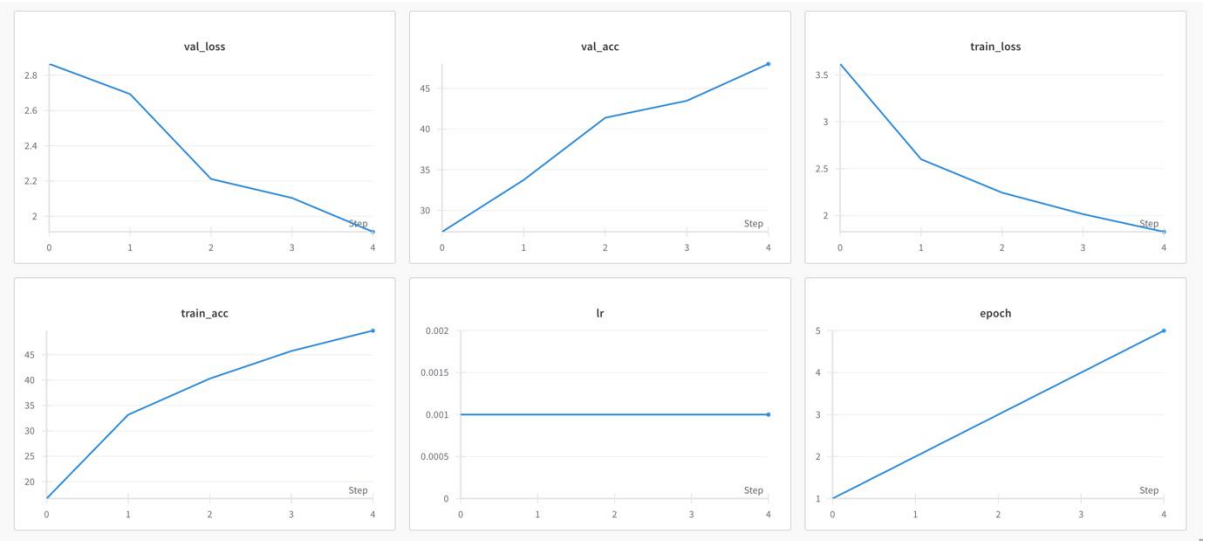

Figure 2. Part 2 Results

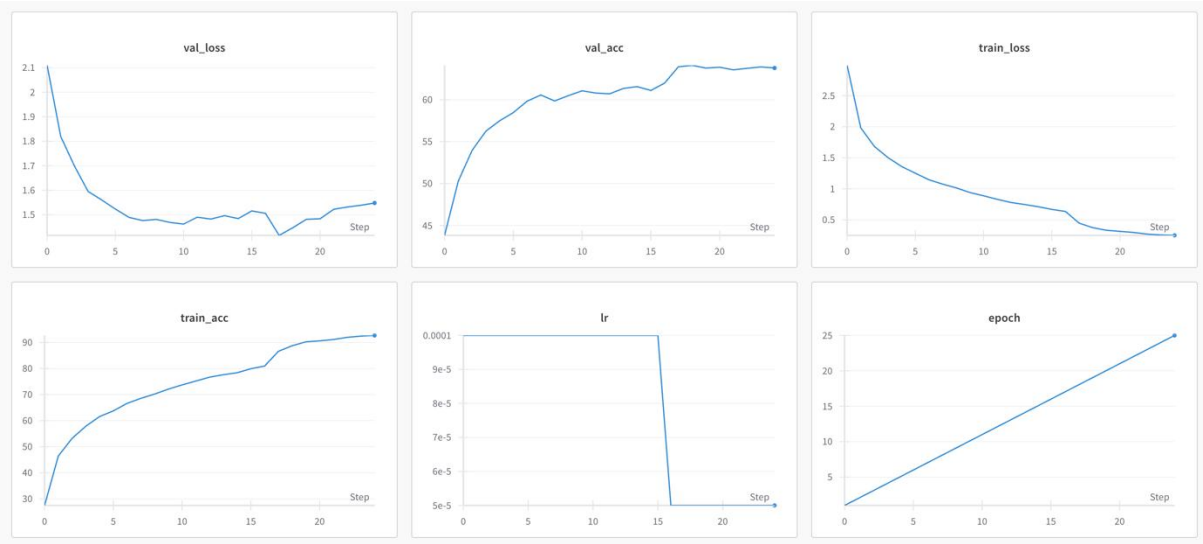

Figure 3. Part 3 Results