Midterm Report

AI Statement

I have annotated in detail in the code comments where AI is involved.

Here are the questions I asked AI:

1. **Standardized parameters for CIFAR-100?**

Answer: In most experiments and code implementations, the commonly used parameters are:

- Mean: (0.5071, 0.4867, 0.4408)
- Standard deviation: (0.2675, 0.2565, 0.2761)

2. **How can the model's generalization performance be improved without changing the dataset?**

Answer:

- **Regularization Techniques:** Dropout, Weight Decay, Label Smoothing
- **Data Augmentation:** Random Transformations, Mixup Augmentation
- **Optimizers and Learning Rate Schedulers**
- **Model Architecture Adjustments**
- **Early Stopping**

3. **What functions can we choose for the criterion and optimizer in the CIFAR-100 task?**

Answer: For criterion, it's better to choose nn.CrossEntropyLoss, because it automatically combines softmax and negative log-likelihood loss internally, which is suitable for classification tasks. For optimizer, it's better to choose SGD, because one of the most commonly used optimizers, which can be combined with momentum to help accelerate convergence and escape from local optimality. Example: optim.SGD(model.parameters(), lr=0.01, momentum=0.9, weight_decay=5e-4).

But in part 3, I compared two different optimizers, SGD and AdamW, and AdamW performed better in this case.

4. **How can we improve the diversity of the training data and the generalization ability of the model?**

Answer: Common improvement methods include: Color Jitter, RandomRotation, RandomAffine, RandomErasing, RandomHorizontalFlip, AutoAugment/RandAugment.

5. **What are the options for learning rate schedulers and what are their characteristics?**

Answer: StepLR, MultiStepLR, ExponentialLR, CosineAnnealingLR, ReduceLROnPlateau, CyclicLR, OneCycleLR.

6. **How can a Linear layer be replaced by a Sequential module to enhance the model's expressive capacity?**

Answer: You can use nn.Sequential to combine multiple layers together, replacing the original single Linear layer, so that the model can progressively extract more complex features. The specific approach is as follows:

- **Batch Normalization (BatchNorm):**
  Apply normalization before the input enters the fully connected layer to ensure a stable data distribution, which benefits subsequent training.
- **Dropout:**
  Add Dropout before and after each fully connected layer to randomly drop some neurons, thereby reducing the risk of overfitting.
- **Fully Connected Layer (Linear):**
  Stack multiple fully connected layers to gradually change the feature dimensions and construct a multi-layer nonlinear mapping.
- **Activation Function (ReLU):**
  Use an activation function between fully connected layers to introduce nonlinearity, enabling the model to capture more complex relationships.

7. **What is the range of values for the tuning parameters provided by OneCycleLR?**

Answer:

max_lr > 0, epochs $\in \mathbb{N}^+$, steps_per_epoch $\in \mathbb{N}^+$, pct_start $\in (0, 1)$, div_factor > 1, final_div_factor > 1.

Part 1 Model

Performance Score: 0.17869

Username: Shiyi Chen-2

Model Architecture: SimpleCNN

- **Convolutional Layers**: conv1, conv2, conv3
- **Pooling:** Max pooling layer (2×2)

- **Dropout:** A dropout layer with a probability of 0.5
- **Fully Connected Layers:** fc1, fc2
- **Activation Functions:** ReLU is used after each convolution and the first fully connected layer.

Key Hyperparameters

- **Batch Size:** 128
- **Learning Rate:** 0.01
- **Epochs:** 5
- **Optimizer:** SGD
- **Learning Rate Scheduler:** StepLR with a step size of 10 and a gamma of 0.5
- **Seed:** 42

Part 2 Model

Performance Score: 0.33902

Username: Shiyi Chen-2

Model Architecture: AdvancedCNN(Modified ResNet18---without weights)

- **First Convolution:** Changed to a 3×3 convolution
- **Pooling:** Removed the max pooling and replaced it with nn.Identity()
- **Fully Connected Layer:** Replaced to output 100 classes
- **Activation Functions:** Uses ReLU activations as in standard ResNet architectures

Key Hyperparameters

- **Batch Size:** 64
- **Learning Rate:** 0.005
- **Epochs:** 5
- **Optimizer:** SGD
- **Learning Rate Scheduler:** CosineAnnealingLR

Part 3 Model

Performance Score: 0.54101

Username: Shiyi Chen-2

Model Description:

The model is based on a pre-trained ResNet18 architecture and initialized with ImageNet pre-trained weights. However, since CIFAR-100 dataset images are 32×32, which is smaller than ImageNet images, it may affect model compatibility and reliability, so I adjusted some factors. The initial convolutional layer of the original ResNet18 was substituted with a 3×3 kernel, stride 1, padding 1 so that the input image has more spatial information at the beginning of processing. Additional enhancements involve eliminating the max pooling layer to maintain spatial resolution. Lastly, the original straightforward fully connected layer was substituted with a sequential framework. These architectural modifications better suit CIFAR-100, incorporating regularization and non-linear transformations to enhance model performance.

Hyperparameter Tuning:

- **Batch Size**: 128, I compared two different batch size and increased the batch size from 64 to 128 primarily to improve training efficiency and stability, thereby accelerating the overall training process
- **Learning Rate**: The initial learning rate is set to 5e-4
- **Weight Decay:** 5e-4, it helps suppress model overfitting
- **Epochs:** 50
- **Optimizer:** AdamW, I compared two different optimizers, SGD and AdamW, and AdamW performed better in this case
- **Learning Rate Scheduler:** Using the OneCycleLR scheduler, the learning rate is dynamically modified over 50 epochs. The scheduler sets the maximum learning rate to 10 times the initial learning rate and applies a 10% warmup phase at the start of training
- **Seed:** 42
- **Use_mixup:** Apply mixup data augmentation to improve the model's generalization ability

Regularization Technique:

1. **Dropout and batch normalization**: In the fully connected layer, we used different dropout rates (0.5, 0.3, 0.2) and batch normalization at different stages to ensure that the model does not rely on specific features during training, thereby improving the generalization ability of the model. This can help the model effectively prevent overfitting and retain more effective features
2. **Mixup Data Augmentation**: Setting a 50% probability to apply the mixup technique to the input data ensures that during the training process, the model can access both the original samples and obtain the regularization effect from the mixed samples
3. **Learning Rate Scheduler**: The OneCycleLR scheduler acts as a regularization technique by dynamically adjusting the learning rate

Data Augmentation Strategy:

- **RandomCrop:** Crop the image to 32×32 dimensions with a 4 padding along the edges to increase sample diversity

- **RandomHorizontalFlip:** By flipping the image horizontally, we can ensure that the model does not over-rely on the features in the left and right directions during training
- **RandAugment:** Randomly select 2 augmentation operations of magnitude 7 to further increase data diversity
- **ColorJitter:** Adjust brightness, contrast, saturation, and hue to make the model more robust to lighting variations
- **RandomRotation:** Rotate the image by up to 10 degrees to simulate different shooting angles, which in turn helps the model learn to recognize objects from slightly rotated perspectives
- **RandomErasing:** With a 5% probability, randomly erase a portion of the image to help the model learn more comprehensive features

Results Analysis:

Strengths:

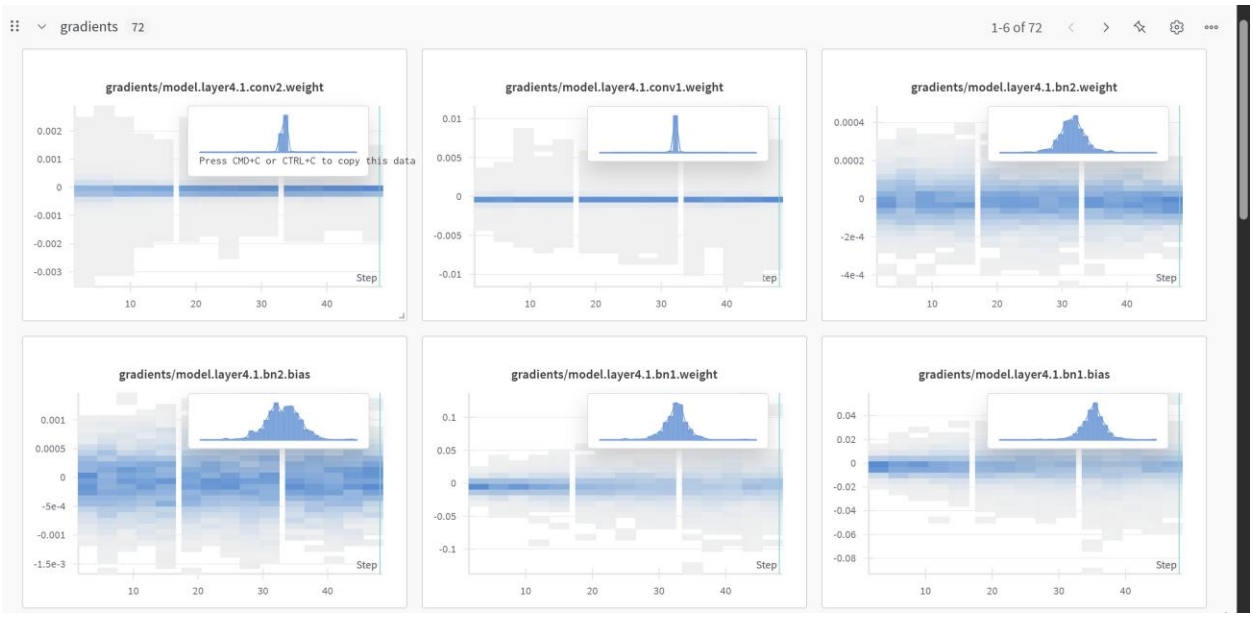
The model uses a pre-trained ResNet18 model as the backbone and initializes the ImageNet pre-trained weights to enable it to quickly learn general visual features. At the same time, my additional modifications to the initial convolutional layer and the removal of the max pooling layer help preserve smaller details in small-sized images. In addition, I applied a variety of regularization techniques (including dropout, batch normalization, Mixup Data Augmentation, etc.) and Data Augmentation Strategies (including RandomCrop, RandomHorizontalFlip, RandAugment, etc.) to effectively mitigate the risk of overfitting and enhance the generalization ability of the model, increase validation and train accuracy, and all these methods help build a more effective model architecture.

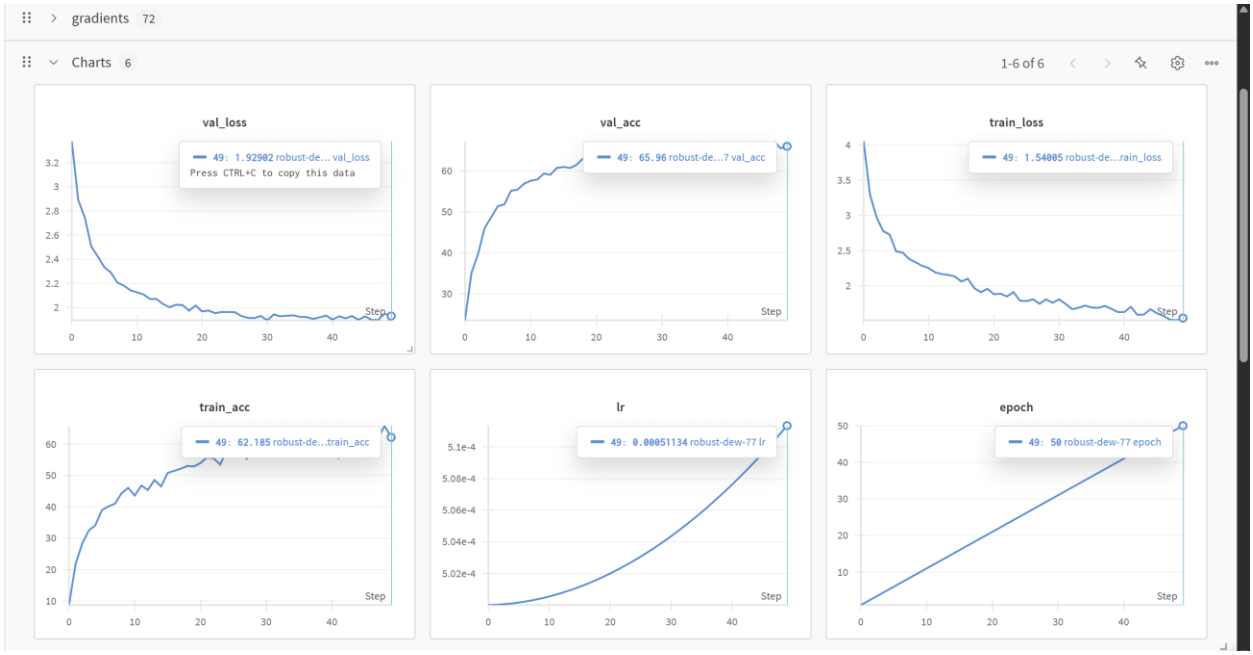
Weaknesses:

Although the model has many data augmentation methods and regularization techniques to improve generalization, due to the large number of parameters involved, I did not test every combination to determine the best settings. Therefore, there is still considerable room for improvement and each parameter needs to be carefully tuned to ensure a positive impact on model performance.

Additionally, from the WandB results, both training and validation accuracy curves had not completely plateaued after 50 epochs, suggesting potential for further improvement. Extending the number of training epochs maybe help to get the better results.

Experiment Tracking Summary:





From the training tracking charts of WandB, we can know:

Loss Function Decrease:

- Validation loss (val_loss) steadily decreases from approximately 3.3 to about 1.92902
- Training loss (train_loss) decreases from approximately 4.0 to about 1.54005
- Both exhibit smooth decrease trends without significant fluctuations, indicating stable training

Accuracy Increase:

- Validation accuracy (val_acc) increases from approximately 20% to about 65.96%
- Training accuracy (train_acc) increases from approximately 0 to about 62.185%
- Validation accuracy slightly higher than training accuracy, typically indicating good regularization effects

Learning Rate Changes:

- The learning rate (lr) curve follows the expected behavior of the OneCycleLR scheduler, starting at a low value, gradually increasing, and then decreasing again
- Final learning rate is 0.0005113

Gradient Distribution:

- Gradient distributions shown in the chart are concentrated and stable

Overall, from the training process showed in WandB, this implementation of a deep learning model can be considered successful. The training process remained stable without obvious problems such as gradient explosion, gradient vanishing, or severe overfitting. The final train and validation accuracy reached 62.185% and 65.96% respectively.