Assignment3

Name: Ruichao Chen

NetID: rc38

Team name on Kaggle leaderboard: Ruichao Chen

Self-supervised Learning on CIFAR10

1) Rotation Training

For this part, the hyperparameters are listed below:

Number of epochs: 45

Decay epochs: 20

Initial learning rate: 0.01

Task: rotation

Optimizer: SGD

Loss function: CrossEntropyLoss

The rotation task achieved 79.1% accuracy.

The important implementations that improved the accuracy are listed below:

1. I used a learning rate decay that reduced the lr every 20 epochs

2. I implemented with gradient clipping to prevent exploding gradients during training

3. Using the entire ResNet18 architecture allowed the model to learn hierarchical features necessary for rotation prediction

2) Fine-tuning late layers

For this part, the hyperparameters are listed below:

Number of epochs: 20

Decay epochs: 10

Initial learning rate: 0.01

Task: classification

Optimizer: SGD with momentum=0.9 and weight_decay=5e-4

Loss function: CrossEntropyLoss

The fine-tuning process focused on the 'layer4' block and 'fc' layer, while freezing earlier layers.

And for the performance comparison:

- **Pre-trained model**: The model initialized with weights from rotation training achieved 66.22% accuracy on the test set after fine-tuning

- **Randomly initialized model**: The model with random initialization achieved 45.88% accuracy on the test set after fine-tuning

So the performance gap (20.34%) demonstrates the importance of self-supervised pre-training. The pre-trained model starts with useful visual representations, giving it a substantial advantage over random initialization. What's more, the pre-trained model shows faster convergence, reaching higher accuracy in early epochs.

3) Fully supervised learning

For this part, the hyperparameters are listed below:

Number of epochs: 20

Decay epochs: 10

Initial learning rate: 0.01

Task: classification

Optimizer: SGD with momentum=0.9 and weight_decay=5e-4

Loss function: CrossEntropyLoss

For the pre-trained model, the final accuracy is 83.94%, but for the randomly initialized model, the accuracy is 79.53%.

Comparing fine-tuning approaches:

1. **Late-layer fine-tuning (section 2)**: Only updates layer4 and fc layers, preserving earlier representations
2. **Whole model fine-tuning (section 3)**: Updates all layers, allowing further refinement of all representations

The difference between these approaches reveals insights about the transferability of features from the rotation task. Fine-tuning only late layers is more computationally efficient but potentially limited in performance, while fine-tuning the whole model allows for greater adaptation but might risk losing some useful representations from pre-training.

4) With a more advanced architecture

For 4.1, I only changed the hypermeters including the batch size, epoch, learning rate and weight decay.

And here are the values used in part4.1:

Batch_size: 512(larger than previous experiments)

Num_epochs: 65(compared to 45 in part 1)

Lr: 0.1(compared to 0.01 in part 1)

Weight_decay: 2e-4(for better regularization)

I also made some slight modifications on the learning rate decay, and the other code is the same as part1.

The most significant improvement came from implementing the cosine annealing learning rate scheduler, which gradually reduces the learning rate following a cosine curve. This approach provides a smoother learning rate decay compared to the step decay used in earlier experiments, allowing for better convergence.

The final accuracy of 4.1 is 83.58%, which is 3% higher than the accuracy of part1.

For 4.2, I also only changed the hypermeters like 4.1

And here are the values used in part4.2:

Batch_size: 512

Num_epochs: 50

Lr: 0.1

Weight_decay: 3e-4

The training code maintains the same structure as previous experiments but benefits from these optimized hyperparameters.

The performance improvement over earlier sections demonstrates that well-tuned hyperparameters and advanced training techniques can substantially impact model performance,

sometimes more dramatically than architectural changes. The cosine annealing scheduler helped prevent the model from getting stuck in local minima during training.

The final accuracy of 4.2 is 87.67%, which is 3% higher than the accuracy of part3.1.