ELEC 475 Lab 3

Ethan Parliament

Student Number: 20220776

## Training

There were 3 different models that were trained on the CIFAR-100 dataset, all using previously existing architectures: AlexNet, VGG16, and Resnet18. A summary of the hyper parameters used for them can be found below.

|  |  |
| --- | --- |
| Hyperparameter | Value used |
| Batch size | 1024 (VGG used size 512) |
| Learning rate | 0.001 |
| Weight decay | 0.001 |
| Transforms | Resize (224), ToTensor(), Normalize() |
| Optimizer | Adam |
| Scheduler | ReduceLROnPlateau |
| Loss function | Cross Entropy Loss |

These hyperparameters were experimented with in order to prevent overfitting from occurring in each of the models, which was a tough challenge. Despite trying different combinations of all hyperparameters, overfitting still occurred in each of the models.

Due to the size of the dataset and the size of the models being too large for my personal computer, models were trained using Google Colab’s T4 GPU. This greatly increased the speed at which the models trained.

Pytorch provides the ability to use pretrained weights of these models, which can decrease training time and increase accuracy. Because this was not explicitly mentioned in the lab instructions, the pretrained weights were not used, which resulted in poorer results than what these models originally achieved.

### AlexNet

Training AlexNet took about 1 minute 45 seconds per epoch, and needed 40 epochs to converge. The loss plot showed clear overfitting in this model, something that was a problem with all models. There is a minimum value hit for the validation loss around epoch 10, then keeps increasing until training finishes.

A graph of a line

Description automatically generated with medium confidence

### VGG16

Training VGG16 took longer to train than both AlexNet and Resnet18, at just over 2 minutes per epoch. This is because the model size was much larger, so a smaller batch size was used to stay within the GPU memory constraints. In a similar fashion to AlexNet, VGG16 also showed heavy overfitting, hitting a minimum validation loss early, and increasing every epoch after. This model converged much quicker, only taking 25 epochs.

A line graph with blue and orange lines

Description automatically generated

### Resnet18

Resnet18 converged in 30 epochs, and took 1 minute 50 seconds per epoch to train. Just like the previous 2 models, Resnet18 also overfitted, but to a lesser extend. Unlike the previous models, Resnet18 does not continue to increase the validation loss as much, instead the curve is much flatter, with a slight increase.

A graph of a line

Description automatically generated with medium confidence

## Testing

As each model tended to overfit, the error rate produced by each of them is not great, but shows small improvements between the 5 epoch and the converged models. The results can be seen in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Error rate | |
| Model | # of epochs | Top-5 | Top-1 |
| AlexNet | 5 | 0.3729 | 0.6695 |
| VGG16 | 5 | 0.3249 | 0.6245 |
| Resnet18 | 5 | 0.368 | 0.6835 |
| AlexNet | 40 | 0.2717 | 0.5316 |
| VGG16 | 25 | 0.31 | 0.5785 |
| Resnet | 30 | 0.2909 | 0.5808 |

In general, the 5 epoch models had a top-5 error rate between 0.37 and 0.32, and top-1 error rate between 0.68 and 0.62. The best model was VGG16, with error rates around 0.4 better than the others. The converged models showed some improvement, with top-5 error rates ranging from 0.31 to 0.27 and top-1 error rates ranging from 0.58 to 0.53. The model that showed the most improvement was AlexNet, going from the worst 5 epoch model to the best converged model.

## Ensemble Implementation

3 different ensemble methods were used for both the 5 epoch and the converged models: maximum probability, average probability, and majority voting. The models were loaded into a dictionary and looped through each model in the dictionary when making a prediction. Then, SoftMax was used to convert the output tensors into a probability of each class. These probabilities were used in each ensemble method to find the ensemble prediction using the max, mean, and mode of the probabilities from each of the models. Torch.stack was used to efficiently combine the output tensors into a single data structure that the ensemble could use to determine what the prediction was.

The majority voting method needed some extra work to handle the case where there was a tie in votes from each of the models. Without handling this edge case where there was a tie, torch.mode would return the output with the lowest class number, essentially a random guess. To deal with this, majority vote was set to default to whatever the max probability prediction was.

To verify that the ensemble methods were making the proper predictions based on the max, mean, and mode, predictions were manually verified by checking the output tensors of the SoftMax and comparing them to the predictions made. Seeing this also gave some insights to how the ensemble works to improve accuracy of the models.

## Ensemble Testing

The ensemble testing showed slight improvement for both the 5 epoch and converged models. Average probability showed the best results in the 5 epoch and converged models, while maximum probability had the worst results.

The 5-epoch ensemble improved error rates by 0.068 compared to the worst model, and 0.009 when compared to the best model. The converged ensemble improved error rates by 0.044 compared to the worst model, and ended up worsening the error rate compared to the best individual model, by a very small margin.

There was a larger improvement with the 5-epoch ensemble compared to the converged ensemble, this could be for a few reasons. The 5 epoch models are more diverse than the fully converged models, and likely did not learn the same patterns as each other. This can lead to more significant improvements compared to the converged models, where they learn the same patterns so combining them does not significantly impact the results. Another reason for this improvement could be due to the overfitting of the converged models. The 5-epoch models are less likely to be overfit, making their errors more random. By aggregating these errors, generalization of the ensemble can improve more than it would with the converged models.

A summary of the ensemble results can be found in the table below.

|  |  |  |
| --- | --- | --- |
| Ensemble method | # of epochs | Top-1 error rate |
| Maximum probability | 5 | 0.6392 |
| Average probability | 5 | 0.6153 |
| Majority voting | 5 | 0.6188 |
| Maximum probability | Convergence | 0.5582 |
| Average probability | Convergence | 0.5360 |
| Majority voting | convergence | 0.5499 |

When manually verifying the predictions coming out of the ensemble methods, some interesting results were seen. The 5 epoch models typically had a 3-way tie for the majority voting method, and defaulted to the maximum probability. However, that was not the case for the converged ensemble, and majority voting rarely saw a 3-way tie. This is confirmed the point made above that the converged models will learn the same patterns, and typically generate the same predictions, whereas the 5 epoch models show more randomness in their predictions.