Deep Learning Mini-Project CS-GY 6953

Usaid Malik, Baiyu Guo, Haoyu Zhao

NYU Tandon
6 MTC, Brooklyn, NY

Overview

In our project we were tasked with implementing a ResNet below 5 million parameters. We began by adapting code from Git Hub [1] for the CIFAR 10 data set. The code had a prebuilt ResNet with an adjustable number of parameters, so we began by shrinking down the number of parameters until we were very close to the 5 million mark but not above it. We then trained this deep model and were able to achieve a test accuracy of 88% on the CIFAR-10 test data. We then trained a different model with less parameters ($\tilde{3}$ Million) and were able to get a test accuracy of 90.02% on the CIFAR-10 test data set. The relevant code for all models can be found at this GitHub repository *https://github.com/hzhao20/DLMiniproject*

Methodology

We based our model on the ResNet-18 architecture in Figure 1 and modified it to keep it below 5 million parameters while still keeping a high degree of nuance and specificity in the model.

We began by trying to get our model as close to the 5 million parameter mark as possible. Our reasoning for doing this was that the deeper the network would be the more accurate it would be in classifying the data.

We implemented a GeLU activation function instead of a ReLU as we belived this would give our model more nuance near the negative values and would allow it to pick up on more complex features. We had tried to remove the padding for our residual layers but in doing so our images shrank too much and we lost much of the information we were looking for so we kept the same padding on our residual layers and instead implemented an average pooling and max pooling layer that we believed would allow our model to have more nuance before we passed the results of the convolutional layer to the fully connected layer.

For our residual layers we had fewer residual layers in the beginning and gradually increased the number of channels from 32, 64, 115, 256. And we had a total of 3, 4, 11, 5 blocks respectively for each residual layer. We followed this pattern as it was a common pattern we found in many of the pre-built Residual networks and we assumed that it would allow us to capture the nuance present in the 32 x 32 x 3 images without overfitting too much on the images.

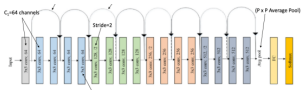

Figure 1: ResNet-18 Architecture diagram

We were extremely wary of overfitting with such a deep network and as such we incorporated two probabilities for our dropout layers, one with a probability of 0.287 and one with a probability of 0.432. We applied the lower dropout probability deeper in the network and the higher probability near the beginning. Our reasoning for doing this was that we would be able to still capture the more complex features that are present deeper in the network without losing too much accuracy and still being able to mitigate overfitting.

To further prevent over fitting with our deep network we also incorporated L2 Regularization when we were doing our training with a weight decay of $.0023$ between 0.001 at different steps throughout our training process.

Finally, since our model would benefit from having more data we implemented a series of random transforms on our training data such as Random Cropping, Random Zoom, Random Translations, Random Flips, and Color Transforms all in an attempt to get more data for our deeper model.

During training for this model we adjusted the hyperparameters during training such as the learning rate and epochs. Additionally, we implemented a gradient accumulation step in the hopes of speeding up our training. We utilized Automatic mixed precision from Pytorch as well with the hopes of decreasing our training time.

We started with a batch size of 128 for our training data and a learning rate of 0.001. Using this we were able to lower our Cross Entropy Loss to about 0.4 however this was still too low and our model was underfitting with a test accuracy of around 60%.

As such we decreased our learning rate as the loss was oscillating around the value of 0.4 indicating that our learning rate was too high and as such we also implemented a learning rate scheduler. We initially tried the Step learning rate scheduler from PyTorch but quickly found that it wasn't working as intended so we switched to the Reduce on Plateau learning rate and had it reduce the learning rate as the loss stopped improving for 2 epochs.

We additionally lowered our learning rate to about $1e-4$ before beginning training again and kept lowering it after about 50 epochs. We had tried the Stochastic Gradient Descent with a momentum of 0.9 but found that it wasn't lowering the test loss as well as the Adam optimizer so for our use case so we stuck with Adam and lowered our regularization penalty and increased our batch size to 256 to help lower the model loss and using this we were able to get our loss down to about 0.15 on the training data. After testing on the test data we were able to achieve an accuracy of 88% on the test data for the CIFAR 10 data set.

Since we were worried our model was overfitting we reasoned we could get a larger test accuracy so we changed our architecture to be less deep and lowered our total parameters to $\tilde{3}$ million. this architecture was similar to the deep architecture except that we had lowered the number of residual layers to 3 with each one having 2 blocks and 64, 128, and 256 channels each respectively. We also removed the dropout layers and returned to a ReLU activation function. Our fully connected layer was also 256 neurons instead of 256 * 3 * 3.

Our convolutional layers were the same but we had removed our max pooling and average pooling earlier in the network and instead placed an average pooling layer right before passing the inputs to the fully connected layer. We had done this as we were worried our model would now be underfitting so we removed the dropout layer and increase the amount of channels in each layer. to also ensure we don't go too deep in our network

Additionally during training, we had just done a random flip and random crop transform as we were worried about too much data our model would overgeneralize as it wasn't as deep as the previous one and wouldn't be able to accurately capture the specificities in the test data.

During training, we trained for just about 30 epochs at a learning rate of .001. This was much better than our 200+ epochs on the deeper model and we were able to get our training loss down to about .08. This resulted in an improved test accuracy of about 90%

We hadn't used any regularization or special techniques when training our shallower model indicating to us that it was better to use that model for faster training as we were able to finish training it much faster than the deeper model. Additionally, the accuracy on the test data was higher indicating to us that it was possible our deeper model was too deep and was overfitting on the data giving us a lower accuracy and higher training time indicating to us that we may have too little training data for the deep network and that the shallower network had worked better for the CIFAR 10 data set.

While working with the two models we were able to learn that the deeper the model is the more difficult it is to lower its loss without different techniques such as normalization increasing the batch size and lowering the learning rate. Due to the depth of the model updating the gradients may have been difficult as well as updating the weights resulting in a lot of epochs and training time.

The shallower model proved to work better with fewer parameters being better able to generalize to the test set due to the relatively lesser parameters being easier to update the weights resulting in a faster training time compared to the deeper model.

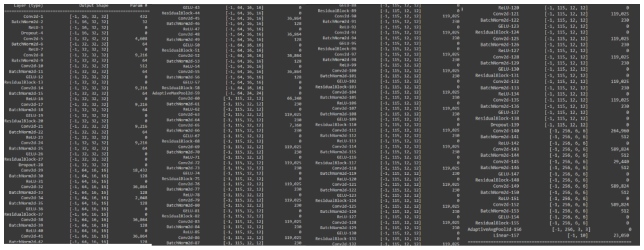

Figure 2: Model Architecture for the Deeper Model

Additionally, by lowering the learning rate for the models we were able to increase our accuracy as the gradient was better able to converge to a minimum for the loss without overfitting.

Results

For the deeper model we were able to achieve a test accuracy of 88% on the test CIFAR-10 data set. The model had 4,999,689 total parameters and its architecture is shown in Figure 2.

The shallow model had a final test accuracy of 90.04% and 2,777,674 parameters. The architecture of the model is shown in Figure 3.

Conclusion

Our loss curve is depicted in Figure 4 for the shallower model The graph presents the trend of training loss decrease over training epochs.

We observe from the training loss curve:

- Rapid initial decline: The training loss drops quickly within the first few epochs, indicating that the model is learning features from the training dataset at a high rate initially.
- A Steady decline trend: After the initial phase, the speed of loss reduction slows down but maintains a steady decline, typically suggesting that the model is gradually converging to a better solution.
- Gradual stabilization: As the number of epochs increases, the magnitude of loss reduction diminishes and begins to plateau. This suggests that the model may be approaching its performance limit on the current training set.

Such a loss curve is generally a positive sign as it indicates that there is no significant overfitting for the shallower model or other training issues occurring during the training process. The smooth decline of the loss curve could also imply that the learning rate is well-calibrated and the model capacity is sufficient to capture the features of the training data. However, this is only an indication of the model's performance on the training set; a comprehensive assessment of the model's performance would also require examining the loss and accuracy on a validation set. Moreover, to draw final

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 32, 32]	1,728
BatchNorm2d-2	[-1, 64, 32, 32]	128
ReLU-3	[-1, 64, 32, 32]	0
Conv2d-4	[-1, 64, 32, 32]	36,864
BatchNorm2d-5	[-1, 64, 32, 32]	128
ReLU-6	[-1, 64, 32, 32]	0
Conv2d-7	[-1, 64, 32, 32]	36,864
BatchNorm2d-8	[-1, 64, 32, 32]	128
ReLU-9	[-1, 64, 32, 32]	0
ResidualBlock-10	[-1, 64, 32, 32]	0
Conv2d-11	[-1, 64, 32, 32]	36,864
BatchNorm2d-12	[-1, 64, 32, 32]	128
ReLU-13	[-1, 64, 32, 32]	0
Conv2d-14	[-1, 64, 32, 32]	36,864
BatchNorm2d-15	[-1, 64, 32, 32]	128
ReLU-16	[-1, 64, 32, 32]	0
ResidualBlock-17	[-1, 64, 32, 32]	0
Conv2d-18	[-1, 128, 16, 16]	73,728
BatchNorm2d-19	[-1, 128, 16, 16]	256
ReLU-20	[-1, 128, 16, 16]	0
Conv2d-21	[-1, 128, 16, 16]	147,456
BatchNorm2d-22	[-1, 128, 16, 16]	256
Conv2d-23	[-1, 128, 16, 16]	8,192
BatchNorm2d-24	[-1, 128, 16, 16]	256
ReLU-25	[-1, 128, 16, 16]	0
ResidualBlock-26	[-1, 128, 16, 16]	0
Conv2d-27	[-1, 128, 16, 16]	147,456
BatchNorm2d-28	[-1, 128, 16, 16]	256
ReLU-29	[-1, 128, 16, 16]	0
Conv2d-30	[-1, 128, 16, 16]	147,456
BatchNorm2d-31	[-1, 128, 16, 16]	256
ReLU-32	[-1, 128, 16, 16]	0
ResidualBlock-33	[-1, 128, 16, 16]	0
Conv2d-34	[-1, 256, 8, 8]	294,912
BatchNorm2d-35	[-1, 256, 8, 8]	512
ReLU-36	[-1, 256, 8, 8]	0
Conv2d-37	[-1, 256, 8, 8]	589,824
BatchNorm2d-38	[-1, 256, 8, 8]	512
Conv2d-39	[-1, 256, 8, 8]	32,768
BatchNorm2d-40	[-1, 256, 8, 8]	512
ReLU-41	[-1, 256, 8, 8]	0
ResidualBlock-42	[-1, 256, 8, 8]	0
Conv2d-43	[-1, 256, 8, 8]	589,824
BatchNorm2d-44	[-1, 256, 8, 8]	512
ReLU-45	[-1, 256, 8, 8]	0
Conv2d-46	[-1, 256, 8, 8]	589,824
BatchNorm2d-47	[-1, 256, 8, 8]	512
ReLU-48	[-1, 256, 8, 8]	0
ResidualBlock-49	[-1, 256, 8, 8]	0
AdaptiveAvgPool2d-50	[-1, 256, 1, 1]	0
Linear-51	[-1, 10]	2,570

Figure 3: Architecture for the Shallow Residual Network

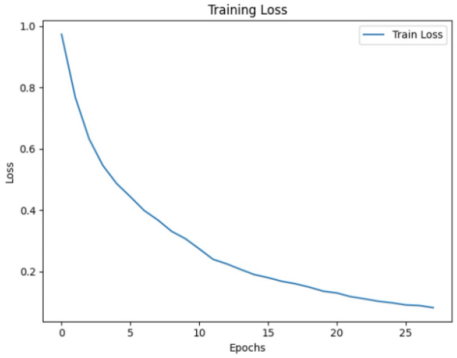

Figure 4: Loss Curve for the shallower Model

conclusions, it would be necessary to compare these results with those from other models or experiments under different configurations.

This is in contrast to the deeper model whos loss plateaus earlier at around 0.4 and requires different techniques to be able to get the loss lowered as the training epochs increase indicating that the deeper model is much harder to train despite the similair accuracy of the two models on the test set.

Going forward it is paramount to find a healthy balance between having a deeper model that is able to capture the complexity within the images and their classes as well as a shallower model that can be trained quicker and is less prone to overfitting.

Citations

[1] *https://github.com/drgripa1/resnet-cifar10*