CSCE 636 Project: CIFAR Classification using NFNets

Hema Deva Sagar Potala
hpotala@tamu.edu

Abstract

In the assignment 2, as per the instruction given there, we explored typical ResNets. As we know, one of the most important components that allows ResNets(or any CNN based model for image classification) to be deeper is Batch normalization. According to [1], it ensures good signal propagation which allows us to train deeper networks, without convolutions exploding. However, there are certain caveats with Batch-Normalization. It breaks the independence between training examples, memory overhead and also poses difficulty in replicating the trained models on different hardware. To counter the above disadvantages, I explored NFNet - Normalizer-Free ResNets, for the project, to free typical ResNets from batch normalization for good. Not only they make the training faster but according to [2] even the smaller models made of NFNets match the performance of EfficientNet (one of the SOTA models) on imagenet. NFNets is still an ongoing research field and I will talk about 2 significant factors that make up any variant of NFNets.

1 NFNets in Brief

Freeing the Normalization: According to [4], in typical ResNets, Batch normalization downscales the input to each residual block by a factor proportional to the standard deviation of the input signal. And each residual block increase the variance of the signal by an almost constant factor. Keeping these two findings in mind, authors in [3], proposes the following modified residual block that mimics the above 2 findings. That is,

$$x_{l+1} = x_l + \alpha f(\frac{x_l}{\beta_l})$$

where x_l denotes the input to the lth residual block, $f_l(.)$ denotes the residual block function, α denotes the hyperparameter (recommended value is 0.2) and β_l is choosen to be Var(x_l). Initializing the weights of the function f such that Var(f(x)) = Var(x), gives an analytical form to derive β_l. We achieve initialization that preserves variance through He or Kaiming initialization.
With the above initialization, β_l can be predicted with the following recurrence relation

$$\beta_l = Var(x_l) = Var(x_{l-1}) + \alpha^2$$

Since we normalize the data, $Var(x_o) = 1$. The above mentioned modified residual block, along with α and β_l will help in good propagation of signal without being exploded.

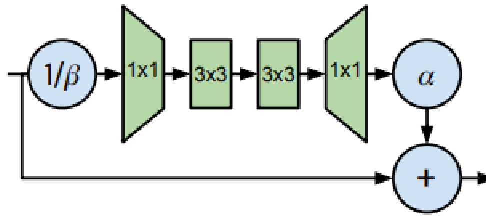
Recifying activation induced mean shifts: According to [3], it was observed that changing the residual block form, although helped, did introduced few practical challenges that arose from the mean shifts seen in hidden activations. To curb this mean shift and ensure that the variance in the residual branches are preserved (from exploding), scaled weight standardization, inspired from [5], is proposed by the authors in [3]. The authors suggest that we re-parameterize the weights of the convolution layers through the training in forward pass as below

$$\hat{W_{i,j}} = \gamma \frac{W_{i,j} - \mu_i}{\sigma_{W_i}\sqrt{N}}$$

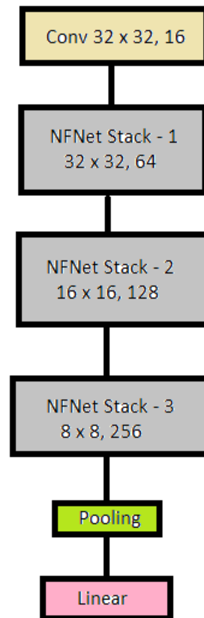
Where μ and σ are calculated across fan-in of the convolution filters. γ is the scaling dependent on the kind of activation the network uses. For my network I used ReLU as activations, for which $\gamma = \frac{\sqrt(2)}{\sqrt(1-\frac{1}{\pi})}$ (this value is derived in [3])
I designed a small version of NFNets in-corporating the above design choices.

2 Network choices and Design

According to [2], the NFNet residual blocks are designed slightly different from typical resnet bottleneck. Below picture, (take from [2]) shows a simple schematic of NFNet blocks.



Each of the convolution operation (not shown in the above picture but) is preceded by scaled ReLU activation. Although the paper [2] recommends to use Squeeze and Excite blocks after the last 1 x 1 convolution for faster training, I ignored them in my network design, since I didn't intend to train my network with larger batch size like 4096 and above.
My final network was moderately inspired from the Assignment 2 network. Below is the network schematic I used for final experimentation and reporting:



In each NFNet stack I used 5 NFNet blocks. This decision is mostly guided by the GPU and resource limitation. On my machine, which have a GPU of RTX 2060, 1 epoch takes 230 seconds to run for a stack size of 5. Anything above increases the run-time for each epoch to more than 230 seconds which is not feasible for effective hyper-tuning and experimentation (on my machine). So I present my results and experimentation observations with this architecture design. Note: For more details on the network, please refer to Network.py.

3 Data Pre-processing

There are few data pre-processing choices I adopted. Gaussian noise addition, image rotation (upto a maximum of 15 degrees) and horizontal flip.
The pre-processing function do not perform any processing to the original image 40% of the time. And does gaussian noise addition, rotation and horizontal flipping each 20% of the time.

4 Experimentation

I started fine-tuning the network on learning rate and batch size. I trained the model on every hyper-parameter combination for 40 epochs to assess which optimal setting to choose for final full blown training. Following is the hyper-parameter search space for learning rate and batch size:

$$learning_rate = [0.001, 0.01, 0.1], batch_size = [32, 64, 128, 256]$$

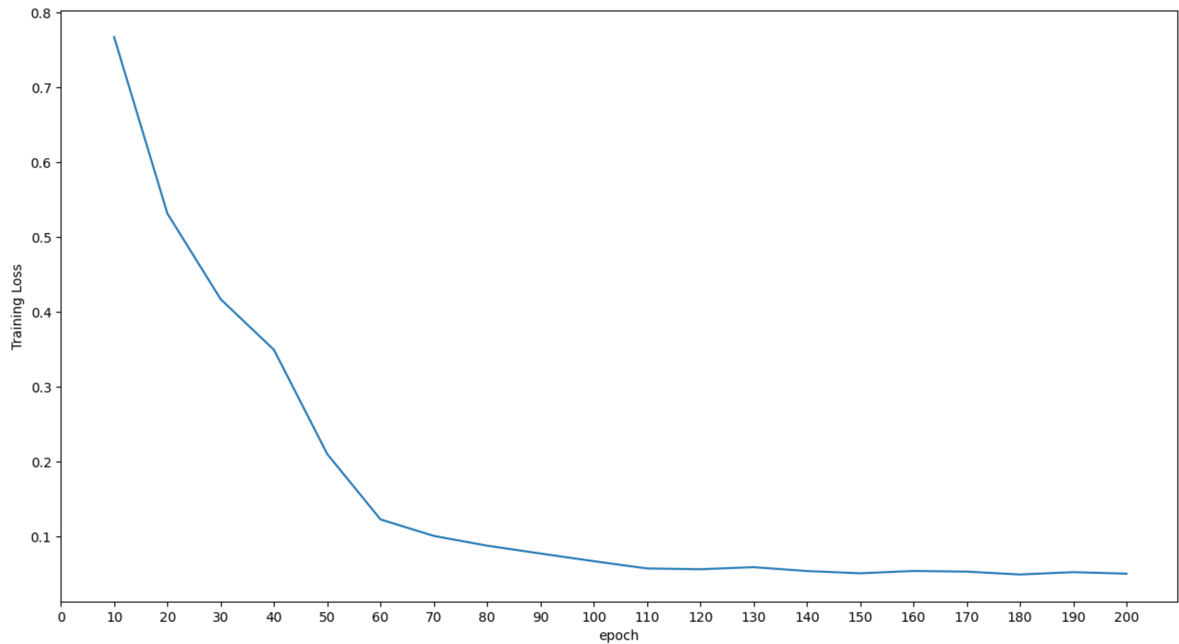
Below is the validation accuracy at every hyper-parameter combination:

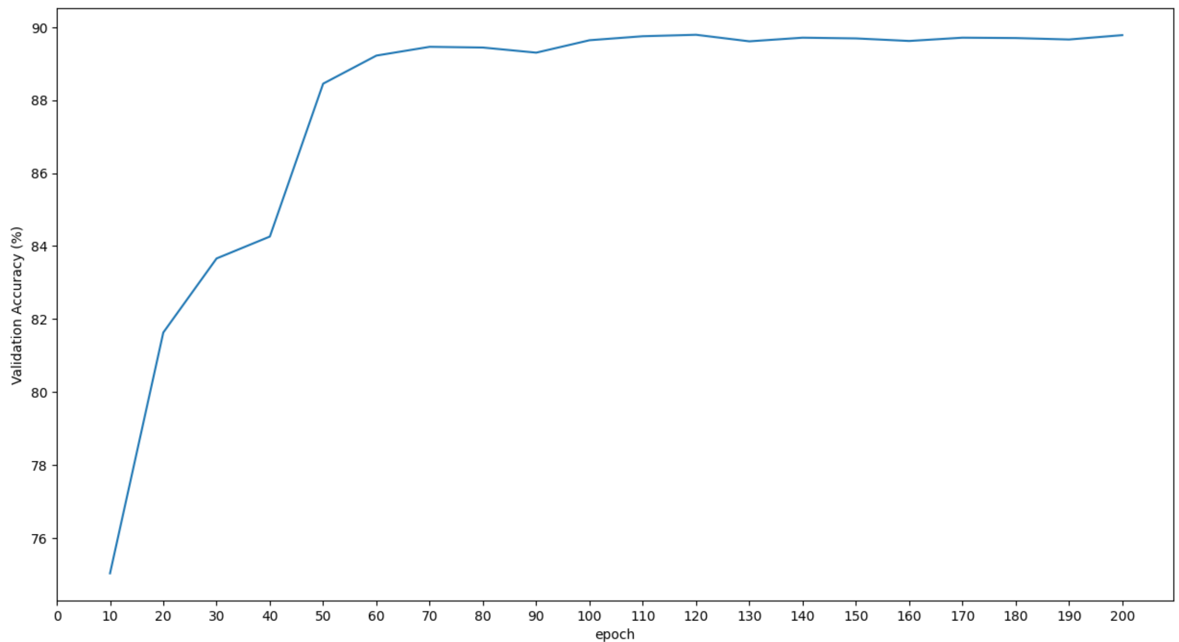
Batch Size	Learning Rate	Validation accuracy (%)
32	0.001	79.88
32	0.01	83.65
32	0.1	10.14
64	0.001	73.54
64	0.01	85.01
64	0.1	10.14
128	0.001	65.76
128	0.01	84.23
128	0.1	10.14
256	0.001	54.54
256	0.01	81.36
256	0.1	10.14

From the above table, the settings marked in green achieved better validation accuracy. So I trained my final full blown model with those settings, i.e, learning rate = 0.01 and Batch Size = 64 for max epochs = 200.
Note: I captured the entire code for hyper-tuning under "hypertune" mode (new mode I added to code apart from train, test, and predict). The command to run this hyper-parameter tuning is mentioned at the start of the report and also in the README file in code folder.

5 Results

I trained the model with the above optimal settings for 200 epochs and below are the training loss and validation accuracy plots at 10 epoch intervals.





The maximum validation accuracy observed is $\mathbf{89.78\%}$ and it was seen to be achieved at various stages. Like at epoch 110, 120 and 200. Test accuracy realized on the public test set with model after 200 epochs is $\mathbf{89.28\%}$. Important observation to note from the above is the improvement starts to plateau after 120 epochs. One potential way to get more improvement is to probably increase the model size.

6 Observation and conclusion

- While hyper-tuning, I observed 0.1 learning rate resulted in a very unstable model. Not exactly sure of the cause but the model didn't seem to learn there.

- Using He or kaiming initialization is very important for NFNets since they do good job at preserving the variance of the pre-activations across all the layers of the network. I experimented with other initialization like Xavier, uniform, etc. but the hidden activations and gradients, both exploded and prevented the model from learning.
- Also observed that the model's performance became a plateau after 120 epochs of training. Increasing the model capacity may help.

7 References

1 https://arxiv.org/pdf/2101.08692.pdf

2 https://arxiv.org/pdf/2102.06171.pdf

3 https://arxiv.org/pdf/2101.08692.pdf

4 https://proceedings.neurips.cc/paper/2020/file/e6b738eca0e6792ba8a9cbcba6c1881d-Paper.pdf

5 https://arxiv.org/pdf/1903.10520.pdf