Tin Luu
Computer Vision
Kaggle Competition

Report

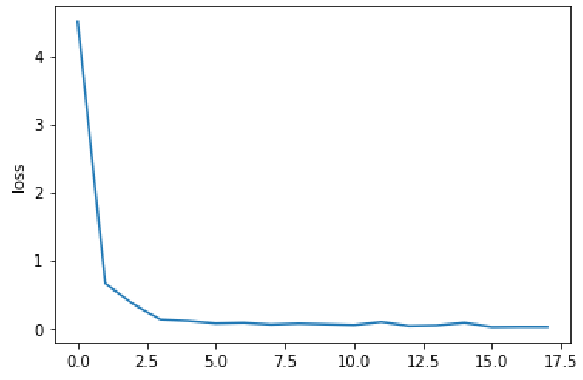
Here’s the link of the Kaggle Competition: https://www.kaggle.com/c/nyucvfall2019

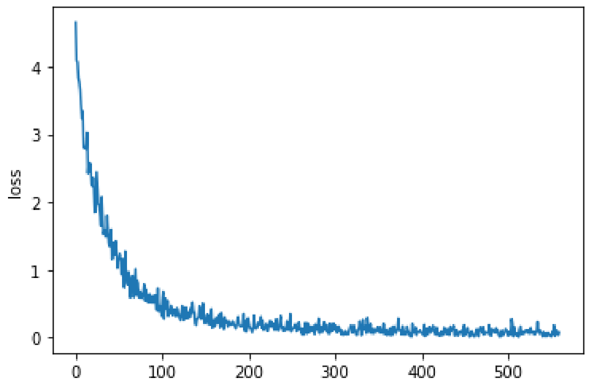
My best model at submission to Kaggle achieved 0.98907 accuracy on the test set on the Public Leaderboard. On the Private (test data) Leaderboard, my model achieved 0.99097 accuracy.

To achieve this model, here are the steps I took:

- [Baseline Model] Noted the architecture of the baseline (provided) model and run it.
  - This model uses SGD optimizer with momentum
  - Its architecture is deep convolutional NN
    - 2 layers of convolution and 2 fully connected layers
    - ReLu nonlinearity, max pool, regularization with drop out at rate 0.5
  - Performance: 86% validation accuracy after 10 epochs.
- [Model 1] To achieve immediate improvement, momentum SGD is replaced with AdamW optimizer
  - Why AdamW? This optimizer still maintains momentum, has decoupled weight decay to assist with regularization and adaptive learning rate to different parameters, all these allow for faster training of the model
    - Default parameters from Pytorch were used
  - Performace: the model is able to achieve the same validation accuracy as the baseline model’s in just 5 epochs. After 10 epochs, validation accuracy is 92%.
- [Model 2] Aside from tuning the parameters of AdamW, to improve the model performance on the training data set, I implemented a convolution NN inspired by AlexNet in the aspects of depth, padding and pooling.
  - Architecture:
    - 3 convolutional and 3 fully connected layers
      - Padding to prevent losing information of the image boundary
    - Overlapping max pool
      - To prevent overfitting
    - Drop out at the 2nd conv layer, and at 1st and 2nd fc layers.
  - Using with architecture with AdamW (lr = 0.0005, weight_decay = 0.01), the validation performance reached 92% after 4 epochs, and then peaks at 95% from epoch 6th onward. Overfitting is observed as the training accuracy stays at 97%.
  - Test accuracy is 96.26%
- [Model 3] A deeper conv NN with more regularization and specifications.
  - Architecture
    - 4 convolutional and 3 fully connected layers
    - Increase regularization: drop out (p = 0.6) at 2nd and also 3rd conv layers

        - Batch normal at every hidden layer to increase training efficiency
        - Leak ReLu is used as the model is now slightly deeper
        - “Kaiming He” initialization for weights is used for the 1st hidden layer to be compatible with (leak) Relu
    - Train for 10 epochs.
    - Performance: worse than the much simpler Model 2. Validation accuracy plateaus at 94%. With this, I return to Model 3.
- [Model 4] Used the smaller Model 2 but added weight adjustments to fine tune by
    - applying Kaiming He initialization – as it works well wit ReLu nonlinear – and batch normalization for all hidden layers only. This allows for more efficient training (initialization and speed)
    - Learning rate is chosen at 0.0005 and the weight decay rate at 0.5.
    - The model is slightly enlarged – more channels are used to capture more edges
    - Performance: 88% validation accuracy by 2nd epoch, 92% accuracy by 3rd epoch, 96% by 5th epoch and 97% by the 10th epoch.
- [Model 5] Now, I train Model 4 for much longer, with 20 epochs, and anneal the learning rate by trying both step (gamma = 0.8) and exponential learning rate schedulers. Both produce similar results and so I chose to work with step scheduler. In addition, I also increased the training batch from 64 to 128 as my GPU can handle it and doing this will produce less noise in the loss. This model is able to achieve 98% validation accuracy at the end of the training.
- [Model 6] After tweaking Model 5 and experiment with different parameters and hyper-parameters of the model. Model 6 is my best model to date as it was able to achieve 0.98907 test accuracy. Please find Model 6 in the attachment.
    - Train set: Average loss: 0.000005, Accuracy: 35297/35339 (99.00%)
    - Validation set: Average loss: 0.000763, Accuracy: 3809/3870 (98.00%)
    - To get to best model again, please run the attached main.py and model.file along with the provided data.py file. The CSV file uploaded to kaggle is attached. To get this CSV file again, run the model.pth file against the provided evaluate.py.
    - Here are its convergence plot (loss vs. epochs). Left side is smoother because it was trained on 128-size batch. Right side on 64-size batch.


loss


loss