Longyi Zhang (u3641195)
COMP7606B - Deep Learning

Assignment 1 Report

Introduction

Since the advent of convolutional neural networks (CNN) like AlexNet, deep learning has enabled rapid advancement in computer vision tasks such as image classification for the decade to come. Further design breakthroughs made by the likes of GoogLENet and ResNet allowed for deep and complex networks to achieve ever-higher predictive performance. For this assignment, my task is to train a CNN model for image classification on the CIFAR-10 dataset. I am specifically to implement and train a provided model architecture and experiment with different strategies to improve its accuracy.

Method

The baseline model consists of 5 convolutional layers with the 2nd and 4th layers having a stride of 2. Each layer employs same convolution using 5x5 kernels that double in number at the next layer. The number of kernels starts at 8 and grows to 128. A 5x5 max pooling layer follows the convolutional layers. There are 2 intermediary fully-connected (FC) layers with 120 and 84 neurons each. ReLU is used throughout the model. It has achieved an average accuracy of 60% out of 5 tests on the test set and broadly underperforms on animal labels compared to vehicles possibly due to the greater similarity and less distinguishable feature from background within the former categories.

To increase prediction accuracy, I proposed the following changes to the baseline model.

- Add batch normalisation to each convolutional layer. This method is given as an example.
- Eliminate biases in the convolution layers owing to batch normalisation negating them.
- Double the kernel number at the initial layer to 16.
- Increase the number of kernels by a factor of 3 instead of 2 at each layer on the previous one: another way to increase parameter count.
- Remove the intermediary FC layers as the network does not result in a high number of features.
- Switch the increased strides at layers 2 and 4 for 2x2 max pooling layers since they may result in better feature enhancement within the limited image dimensions.

Experiment & Analysis

I examined each of the strategies mentioned above by modifying the baseline model. The resulting feature models were compared against the baseline model on their predictive accuracy on the test set. For each test case, the average accuracy out of 5 experiments was taken. Models were also compared to each other on loss curves and training accuracy. All models were trained for 5 epochs with CCE loss function and SGD optimiser at a learning rate of 1e-2 and momentum 0.9. Changes that actively penalised or failed to improve model performance were omitted. Training losses and runtimes were logged using Tensorboard and were achieved on GPU Farm phase 2 nodes.

	test_accuracy	avg_time
baseline-test	60.00	38.48
batchnorm-test	73.00	44.48
bn_nobias-test	73.60	42.36
double-test	66.00	36.55
quad-test	69.80	57.20
simpl_fc-test	64.40	36.74
incr_by_3-test	64.40	62.12
max2-test	67.00	39.59
my_net-test	81.20	68.56

The included tables and graph contain each model's test accuracy and average training time, as well as the loss and training accuracy curves on their first training runs.

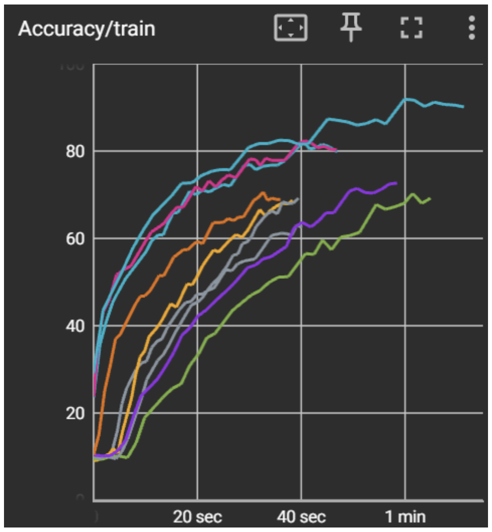
Overall all the tested improvements delivered some degree of improvement on test accuracy and, except those increasing the kernel number, did not incur substantial training time penalty. Moreover, certain architectural changes, namely batch normalisation and cutting FC layers, suppressed training losses as early as the first few minibatches.

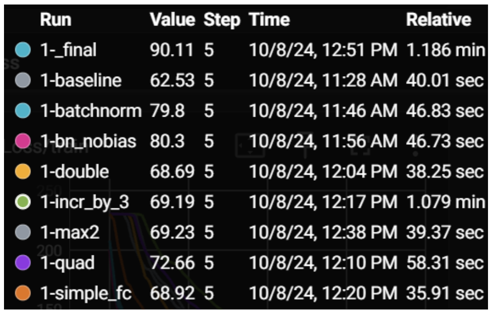
Batch normalisation proved to be the most effective improvement so far, attaining a 13-point average improvement at the cost of a few seconds. It also saw the steepest declining loss curve. This behaviour is expected as the standardised values at each layer allow for much stabler training downstream, thus avoiding the underfitting at the first few minibatches. It was unexpected that the elimination of biases did not produce perceptible changes accuracy or time-wise—the scale of the network on hand may not warrant a significant effect.

Next are the hyperparameter changes to boost kernel number in the network. Merely doubling the number of output kernels at the first convolutional layer yielded a 6-point advantage over the baseline. The feature model also is consistently trained faster, something I have no explanation for. I then performed another set of experiments by further increasing the initial kernel count to 32. This results in a smaller but still noticeable accuracy boost at a much higher time cost. Increasing the layer-on-layer kernel number multiplier to 3 while keeping the initial number constant, however, only returned a modest improvement with much longer training periods. Hence, I can infer that the number of lower level feature filters have a much more significant impact on accuracy than subsequent higher-level ones.

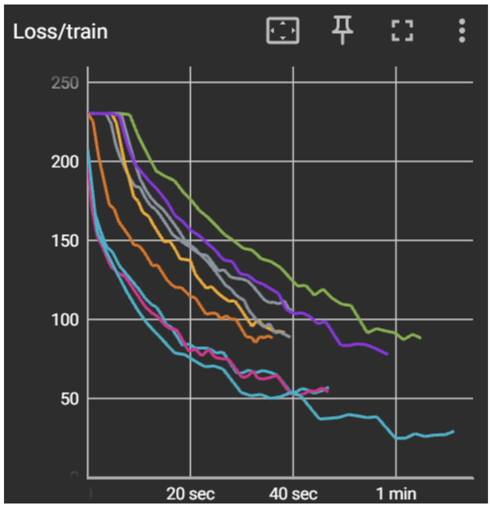
Somewhat surprisingly, removing the intermediate layers yielded a faster and more accurate model with better fitting early during training. For such smaller networks as the one for classifying CIFAR-10, the loss in precise feature location due to the additional layers may have overshadowed the benefit from compounding features. Meanwhile, exchanging the size-2 strides for 2x2 max pooling layers rewarded me with a 7-point accuracy increase with minimal training time penalty, owing to this pooling method accentuating the contrast between features.

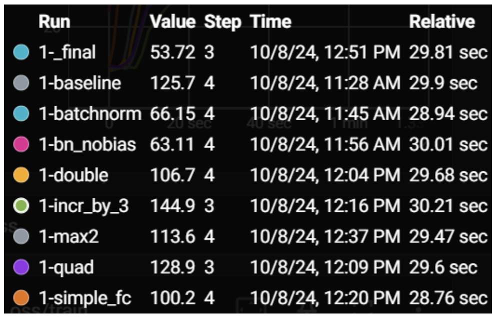
Learning from my experiments above, I utilised the most promising changes to redesign the provided network, aiming for a test accuracy of 80%. Changes included batch normalisation with no bias, quadruple the initial kernel count, 2 2x2 maxpool layers interjected into the convolutional layers, and the absence of in-between FC layers. In addition, I inserted 2 20% dropout layers after convolutional layers 4 and 5 to mitigate overfitting at higher epochs. The final model rewarded me with an accuracy of 81% at epoch 5 and 84% at epoch 10. Class accuracy was improved across the board, the lowest being cat at 73.3%. Thus concludes my task at image classification.





Run	Value	Step	Time	Relative
● 1-_final	90.11	5	10/8/24, 12:51 PM	1.186 min
● 1-baseline	62.53	5	10/8/24, 11:28 AM	40.01 sec
● 1-batchnorm	79.8	5	10/8/24, 11:46 AM	46.83 sec
● 1-bn_nobias	80.3	5	10/8/24, 11:56 AM	46.73 sec
● 1-double	68.69	5	10/8/24, 12:04 PM	38.25 sec
● 1-incr_by_3	69.19	5	10/8/24, 12:17 PM	1.079 min
● 1-max2	69.23	5	10/8/24, 12:38 PM	39.37 sec
● 1-quad	72.66	5	10/8/24, 12:10 PM	58.31 sec
● 1-simple_fc	68.92	5	10/8/24, 12:20 PM	35.91 sec





Run	Value	Step	Time	Relative
● 1-_final	53.72	3	10/8/24, 12:51 PM	29.81 sec
● 1-baseline	125.7	4	10/8/24, 11:28 AM	29.9 sec
● 1-batchnorm	66.15	4	10/8/24, 11:45 AM	28.94 sec
● 1-bn_nobias	63.11	4	10/8/24, 11:56 AM	30.01 sec
● 1-double	106.7	4	10/8/24, 12:04 PM	29.68 sec
● 1-incr_by_3	144.9	3	10/8/24, 12:16 PM	30.21 sec
● 1-max2	113.6	4	10/8/24, 12:37 PM	29.47 sec
● 1-quad	128.9	3	10/8/24, 12:09 PM	29.6 sec
● 1-simple_fc	100.2	4	10/8/24, 12:20 PM	28.76 sec