Name: Andrew Porter

Neural Network Architecture for CIFAR-10 Image Classification

Dataset Loading:

The CIFAR-10 dataset was loaded into PyTorch DataLoaders with the use of torchvision datasets. Data augmentation techniques such as padding, random horizontal flips, and random crops were applied to the training dataset to improve generalization. Both training and testing datasets were normalized.

Various data augmentation techniques were experimented with. The augmentation which worked best was adapted from: https://medium.com/swlh/how-data-augmentation-improves-your-cnn-performance-an-experiment-in-pytorch-and-torchvision-e5fb36d038fb

Architecture:

Intermediate Blocks/Attention Mechanism:

At the heart of this architecture are the 'Intermediate Blocks'. These blocks contain convolutional layers which work in parallel whose output is weighted and summed through an attention mechanism. This mechanism is created as follows: The mean of each feature map across the spatial dimensions (height and width) is computed and stored in a vector 'm'. This vector 'm' is fed into a fully connected layer (softmax) which outputs a new vector 'a', containing l weights (l being equal to the number of layers within the block). This vector 'a' acts as an attention mechanism. The use of a softmax function ensures that these attention weights sum to one. This operation scales the contribution of each layer's output based on its relevance. This approach helps in fine-tuning the information flow through the network, ensuring that the most relevant features are highlighted, and less informative ones are subdued.

Each layer within a block receives the same input and has an output of equivalent spatial dimensions. Similarly, the hyperparameters for each layer within an intermediate block are equivalent (this was experimented with; discussed later). The output of each layer within the model goes through batch normalization and a ReLU activation function (inspired by the ResNet architecture).

Residual Macro Blocks:

The model also includes residual macro blocks, again inspired by the popular ResNet implementation. Within each of these macro blocks are two intermediate blocks of the same structure as mentioned before. However, the input to the first of these intermediate blocks is added to output of the second as a residual. This improves gradient flow, allowing for a deeper architecture in the face of vanishing gradients. In the same manner as ResNet, if the number of input channels to the first intermediate block differs from the output of the second, the input is converted via a 1x1 convolution.

Putting It Together: CIFAR-10 Classifier

After extensive experimentation with various hyperparameters (e.g. varied kernel sizes, in channels/out, number of layers/blocks/residual blocks, dropout, max pooling etc.), this final model was decided upon:

Intermediate Block 1:

- In channels: 3, out channels: 64, kernel size: 3, padding/stride: 1, layers: 2
- Intermediate Block 2:

- In channels: 64, out channels: 128, kernel size: 5, padding/stride: 2, layers: 2

Residual Macro Block 1:

- Intermediate Block 3: In channels: 128, out channels: 256, kernel size: 3, padding/stride: 1, layers: 2
- Intermediate Block 4: In channels: 256, out channels: 256, kernel size: 3, padding/stride: 1, layers: 2
- Input to Intermediate Block 3 converted by a 1x1 convolution and added to output of Intermediate Block 4.
- Max pool applied here (size=2, stride=2)

Intermediate Block 5:

- In channels: 256, out channels: 384, kernel size: 3, padding/stride: 1, layers: 2

Residual Macro Block 2:

- Intermediate Block 6: In channels: 384, out channels: 384, kernel size: 3, padding/stride: 1, layers: 2
- Intermediate Block 7: In channels: 384, out channels: 384, kernel size: 3, padding/stride: 1, layers: 2
- Input to Intermediate Block 6 added to output of Intermediate Block 7.
- Max pool applied here (size=2, stride=2)

Individual Layer:

- In channels: 384, out channels: 256, kernel size: 3, padding: 0, stride: 1
- Batch Normalization, ReLU

Fully Connected Layer 1:

- Adaptive Average Pooling
- In channels: 256, out channels: 128
- ReLU applied
- Dropout applied (p=0.5)

Fully Connected Layer 2:

- In channels: 128, out channels: 10 (number of classes)

The total number of channels was inspired by: https://github.com/BAIDU-USA-GAIT-LEOPARD/CIFAR10-Distributed-Training-BaiduNet9P/commit/df05b74663f21d4cbed894590535fab9a79d4d32#diff-364b5d93e0a68ffc381ba38e21e7cd4161d67733563dae9220eb2b4655785d8a

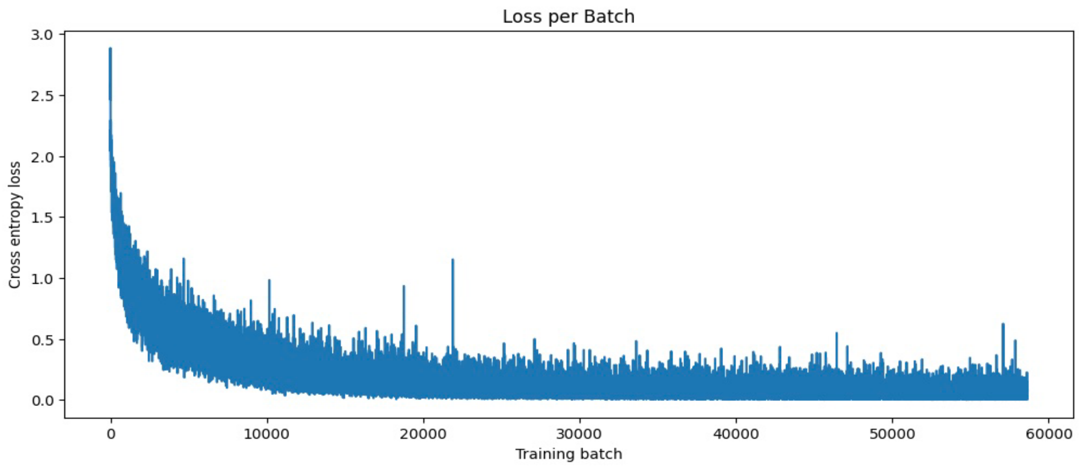
The additional block hyperparameters, as well as the use of an individual layer to reduce feature space after final residual block inspired by: https://github.com/wbaek/torchskeleton/releases/tag/v0.2.1_dawnbench_cifar10_release

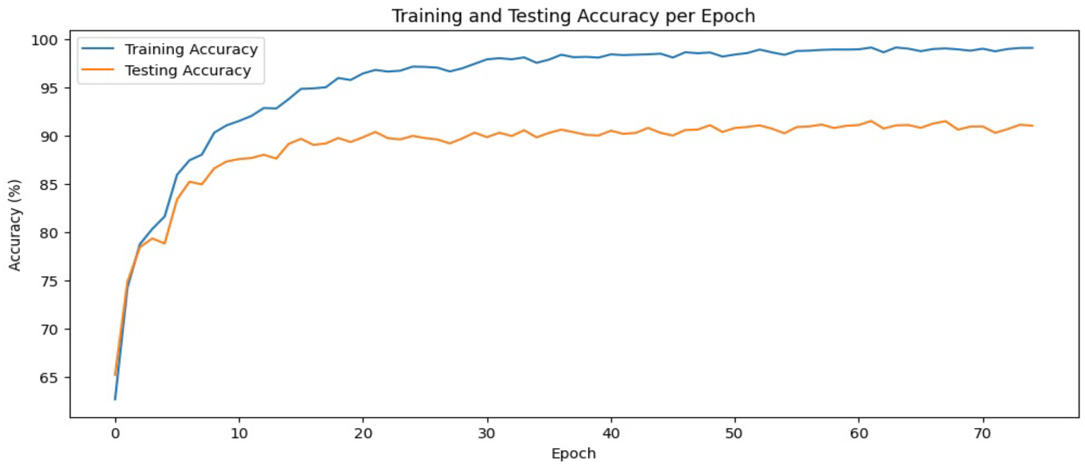
Deviations from the base architecture:

The requested architecture remains at the core of this model: It is made up of intermediate blocks with parallel layers , receiving the same input, whose outputs are weighted and summed by a vector 'a'. The process of how 'a' is created also remains in line with the requirements. However, the model deviates in a few ways: Firstly, by an increased number of intermediate blocks. Secondly, some of the intermediate blocks make up residual macro blocks. Thirdly, there is an individual layer at the end of all intermediate blocks for reducing the number of output feature maps. Lastly, there is an added fully connected layer before the final fully connected layer.

Training Procedure:

The model weights are initialised with Kaiming. Furthermore, the model was trained using the AdamW optimizer with a learning rate of 0.001. Additionally, a weight decay of 2e-2 (0.02) was applied to further combat overfitting, therefore AdamW was preferred over Adam. This weight decay caused some small volatility in later epochs, but the final accuracy achieved was marginally improved than using lower values or none. Cross entropy loss was used as the criterion. The **highest test accuracy** I achieved was 91.7 %, however when clearing outputs and retraining the model I could not replicate this. The highest test accuracy in the current wave of training shown in the code output is 91.5, 62nd epoch.





The Journey:

I began this architecture by incorporating a simplified VGG model within the required architecture, including consistent 3 by 3 filters, ReLU, max pooling etc. This was followed by the addition of batch normalisation. Upon receiving an accuracy of around 75 percent on the testing set, I introduced intermediate blocks with varying kernel sizes across layers. I believed the designed attention mechanism would benefit from utilising varying kernels within a block. However, as the kernel sizes and output channels became more optimised to the task, I found that throughout most of the model variations I tested, varied blocks did not improve testing performance sufficiently; in some cases, performance worsened/caused an increased degree of overfitting. Next, I implemented residual layers and residual blocks. I achieved improved performance when using residual blocks as opposed to individual layers in-between intermediate blocks, although both improved the model's ability to generalise to new data. I experimented with different activation functions (e.g. ELU), augmentation techniques, varying kernel sizes/stride/padding, output/input channels, optimizers, parameter initializations (etc.). Earlier iterations of the model also included sequentially increasing dropout across blocks as the model became deeper. However, the current combination of hyperparameters maximised performance. This includes optimizer parameters (e.g. weight decay, base learning rate). The current architecture displays an optimised combination of this unique parallel architecture and some of the ideas of many successful architectures of the past.