CIFAR-10 Classification

Jayesh Kumar Soni
School of Electronic Engineering and Computer Science
Queen Mary University of London
Mile End Road, London E1 4NS, UK
ec22068@qmul.ac.uk

Introduction

The goal of this project is to build a specific model on the CIFAR10 image data set and test the accuracy of the model on the basis of image classification.

1 Read Data Set and Create data loaders.

This is a 32*32 color image data set and consist of 60,000 images containing one of 10 object classes, with 6000 images per class. The 'torchvision.datasets.CIFAR10' class provides an interface to load and preprocess the CIFAR-10 dataset in PyTorch. There are 50,000 training images and 10,000 test images.

2 Create the model.

This model consists of Backbone and Classifier. There were initially 2 and then 4 Blocks in the model. Each block consists of Linear/MLP layers predicting a vector. This vector is a product of a non-linear activation function and spatial average pool. Block also consists of convolutional layer which are combined with the MLP. Both MLP and convolutional layers are provided with the same input and runs in parallel.

The classifier takes as input the output of the last block. It can be a softmax regression classifier or an MLP.

3 Create the loss and optimizer

Here we have used RMSprop(Root Mean Square Propagation) for optimization. RMSprop is an optimization algorithm used for stochastic gradient descent (SGD). Training loss and test loss are calculated in train_epoch and test_epoch. It can be seen in both 2 and 4 blocks that mostly as the number of epocs increase there is a decrease in the training and test loss.

4 The training script to train the model

This model consists of the blocks and the classifiers. Each block is a combination of the special average pool, MLP layer, convolutional layers and residual connections. Initially I had run this model with the combination of two blocks where input, output provided were (3, 64) and (64, 64) respectively in each block. In blocks, firstly special average pool is used to reduce the dimensions to 1*1. After which the same input is provided to the linear and convolutional layer. The linear layer has 3 input and output. As there are 3 outputs coming from linear layer, that's why from for loop 3 convolutional layers will be generated, which has 3 inputs and 64 outputs, kernel of 3*3, stride of 1*1 and padding of 1*1. After which kaiming normalization is performed to initialize the weights of the convolutional layer.

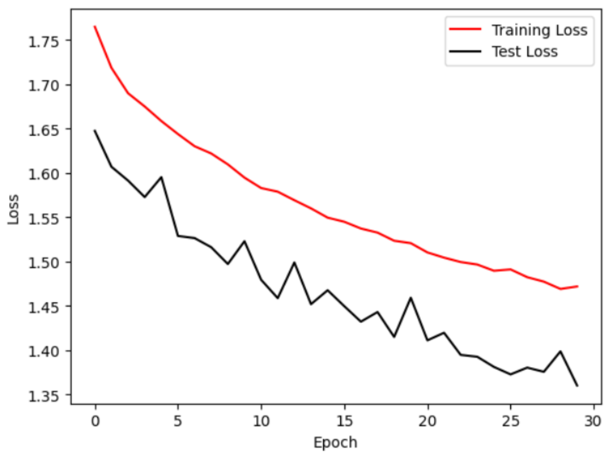
After that the residual connection is added using a skip connection with a 1x1 convolution followed by batch normalization. It is applied only when the input and output channels of the Block are not equal, as indicated by the condition if in_channels != out_channels:. The weights of the residual connection are initialized using the Kaiming normal initialization, which is a popular weight initialization technique for ReLU-based activation functions. This finishes the block after which Batch normalization and ReLU is used to improve the training stability and performance of deep neural networks by normalizing the inputs to each layer in a mini-batch during training.

Similar function is performed in the second block with entries (64, 64). After which max pooling is performed as a down-sampling operation in the backbone.

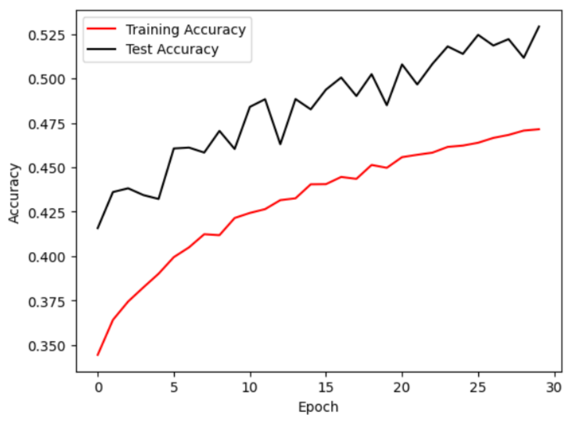
Output of all the blocks is combined together using linear layer to produce a single output.

This output is provided as input to the classifier which consist of special average pool to reduce dimension, flatten function to flatten the tensor into a one dimension vector and a fully connected linear layer to produce the final class score.

This model was run with 30 epochs, learning rate of 0.01 and batch size of 64.



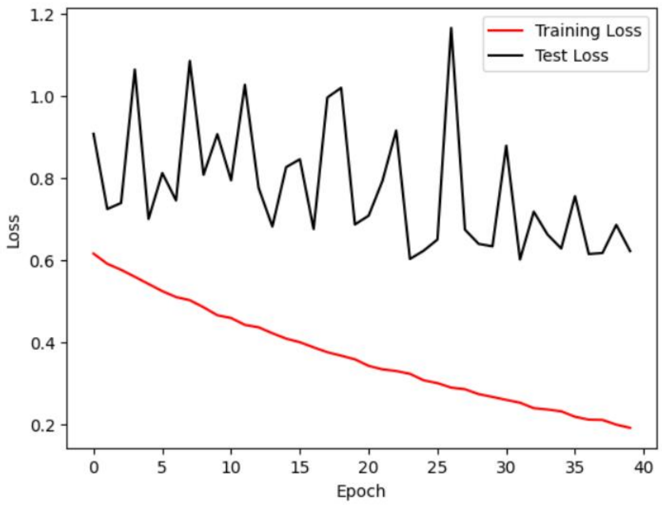
After running this model through the training and the test set, it showed that the training loss and test loss keps on reducing as the number of epoch increased.



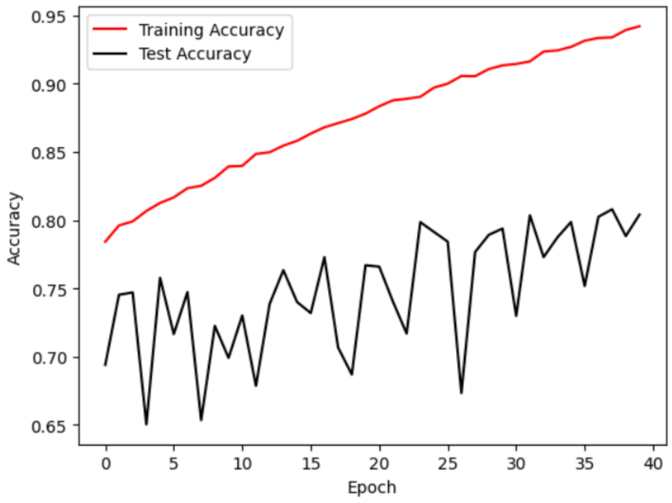
And the training and test accuracy keeps on increasing as the number of epochs increase. The best accuracy scored through this model is 52.45% when epoch is 26 which is very less.

Thus in order to improve the accuracy of this model I increased the number of blocks from 2 to 3 and then to 4. Because increasing the number of blocks will help in setting a bigger batch size. At 4 blocks I increased the input channel and the output channel entries from 64 to 128. Reduced the learning rate from 0.01 to 0.001, this is done is order so that the optimizer used for training the model will update the model's parameters with smaller steps during each iteration. Added a dropout function in classifier, this helps in preventing the model from relying too heavily on any particular neuron, forcing it to learn more robust and generalized representations. Increased the epoch from 30 to 35 and then to 40, so that the training process will iterate over the entire dataset for a larger number of times during training. The new model1 was again trained in the training set

Thus this model was run with 40 epochs, with a learning rate of 0.001, and a batch size of 128.



After running this model through the training and the test set, it showed that the training loss and test loss kept on reducing as the number of epoch increased, even more compared to the previous model.



And the training and test accuracy keeps on increasing as the number of epochs increase. There is a lot of fluctuations in the test accuracy. The best accuracy scored through this model is 80.79% when epoch 38 which is much better compared to the previous model.

Training Details including hyper-parameters used.

The models were trained and validated in the train_epoch and validate_epoch respectively. In Training function the data is entered in the model, loss is accumulated, total number of samples are updated, and the correct number of predictions are counted. Loss and accuracy is calculated of the training data. In the validation epoch function, same tasks are performed for the test data. The loss and the accuracy are calculated and accumulated for the test data set, the number of samples are updated, and the number of correct predictions for the test data set is counted.

The batch size, learning rate, epoch number are the hyper parameters used in both the models. Initially in first model it batch size and number of epochs are not very much and learning rate is very high due to which the accuracy of the model does not turn out to be that good. Thus in second model there has been a change in the entries of the hyper parameters, epoch size and the number of blocks are increased and the learning rate is further decreased which has further increased the accuracy of the model.

5 Model accuracy on CIFAR-10 Validation set

Model first accuracy on cifar 10

	Epoch	Train Loss	Train Acc	Test Loss	Test Acc
Min training loss	29	1.4692	0.4707	1.3987	0.5116
Max training accuracy	30	1.4720	0.4714	1.3602	0.5292
Min test loss	30	1.4720	0.4714	1.3602	0.5292
Max test accuracy	30	1.4720	0.4714	1.3602	0.5292

Model second accuracy on cifar 10

	Epoch	Train Loss	Train Acc	Test Loss	Test Acc
Min training loss	40	0.1903	0.9420	0.6214	0.8040
Max training accuracy	40	0.1903	0.9420	0.6214	0.8040
Min test loss	38	0.2096	0.9338	0.6167	0.8079
Max test accuracy	38	0.2096	0.9338	0.6167	0.8079

Conclusion

On first model I trained with a batch size of 64, learning rate of 0.01 and epoch of 30. It was visible that maximum training, testing accuracy and minimum training and testing loss was coming close to 30th epoch, which stated that performance could be increased with increase in that number of epoch. Thus in second model I increased the number of batches so that batch size could be increased and number of epochs, further reduced the learning rate, which increased the accuracy of the model.

References

1. **https://www.upgrad.com/blog/basic-cnn-architecture/**
2. **https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html#sphx-glr-beginner-blitz-cifar10-tutorial-py**
3. https://www.kaggle.com/datajameson/cifar-10-object-recognition-cnn-explained