CIFAR-10 Classification

Read dataset and create dataloaders

CIFAR-10 is a very popular database of images that can be used for machine and deep learning. It is made up of 50000 training images and 10000 test images, both of which are split into multiple batches.
Since the CIFAR-10 dataset is widely known, loading in the data was straightforward. PyTorch provides some very useful information on how to do this, so I went off their guidance to load in the data (PyTorch, 2023).

Create the model

Following the architecture laid out in the assignment slides, this model was based on a Convolutional Neural Network. The model could be split into two different sections: the backbone and the classifier. For the backbone, the model was made up of N blocks. Each block consisting of one MLP layer and numerous Convolutional layers. Starting with the MLP layer, the input tensor X (our images) are being passed through an initial Spatial Average Pooling layer which reduces the height and width of each image to 1x1. Then it is being flattened so it can be passed through a linear function that takes an input channel of 3 and outputs K, K being the number of elements inside the vector and the number of Convolutional layers. Next, a ReLU activation function is being passed over the vector to remove the non-activated pixels, leaving us with vector a, with K number of elements. For the convolutional layer, it was taking an input of 3 and an output of 32 channels. Next to get an output for an individual block, each element of vector a was being multiplied to each convolutional layer. Depending on the number of blocks, each output would be fed into a new block following the same process as described above. In my case I only used one block for the backbone, so the output was 25,088 features ready to go into the classifier.
For the classifier, it was defined by a final pooling layer on top of the final output of the last block before being passed to through multiple linear functions to reduce the inputted features to 10 which corresponded the number of classes.

Create loss and optimiser

This was very simple. I just used a Cross Entropy Loss function and used the Stochastic Gradient Descent (SGD) for my optimiser. The loss function will measure the dissimilarity between what the predicted distribution is and what the actual distribution is for the targets. I chose this over other functions such as MSE because it would provide greater detail and will help update the weights of the network and help minimise dissimilarity. For the optimiser, I chose SGD since it would greatly reduce the computational cost compared to other optimisers. Neural networks already bare a great cost, so minimising this was paramount. Furthermore, SGDs should aid with reducing the amount of overfitting since it updates the parameters of the model using randomly chosen subsets of the dataset. I set the learning rate at 0.001 to the manage convergence and avoid overshooting the weights for every time the model updates.

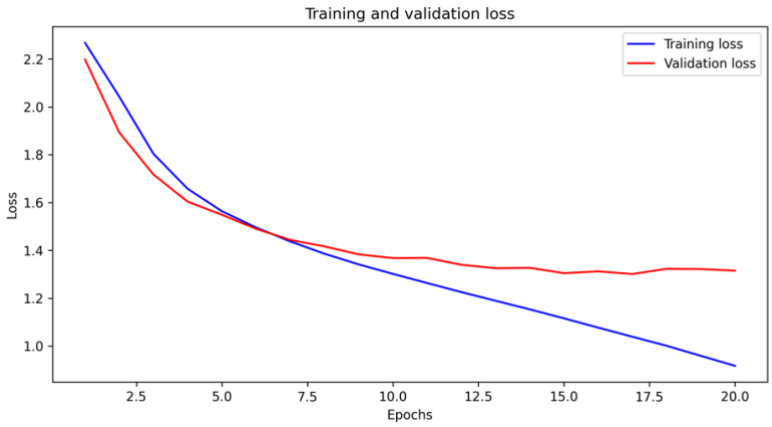
Training the model

The hyperparameters of my model are the following:

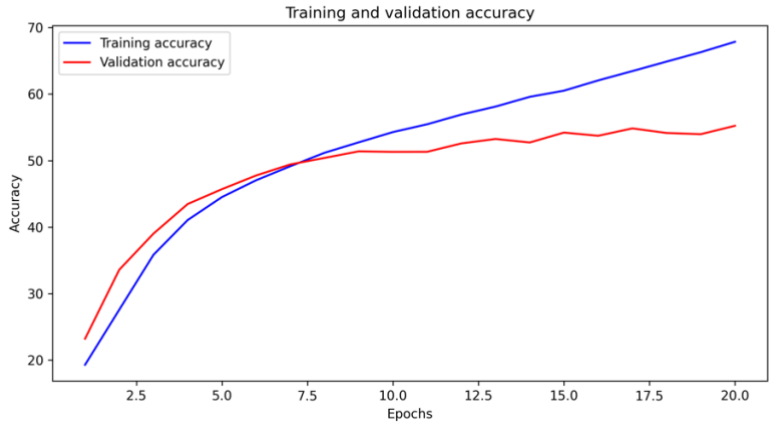
Batch Size	32
K Conv Layers	10
In channels	3 (from the RGB)
Out channels	32
Kernel Size	5
N blocks	1
g activation	ReLU

I opted to keep the batch size at 32. Firstly, because it would help avoid overfitting. Secondly, as you increase the batch size, the range of learning rates that lead to stable convergence and better performance on the validation set diminishes (Masters & Luschi, 2018). For the conv layers, I experimented with different values of K and found 10 to perform best. In channels is fixed at 3. Out channel is another parameter I experimented with, trying values of 64, 128, 256 etc before settling on 32 which provided my best result. The kernel size was set at 5. Since I was only using one block, I wanted a large enough reduction in image dimensions without losing too many features. The activation function I chose was ReLU.

For the training loop, I found guidance with PyTorch documentation (PyTorch, 2023), adapting it to my own use, to plot the following curves for both the loss evolution and training/validation accuracies:



For the first 5-7 epochs, both the training and validation loss function were decreasing at a similar rate. However, eventually as the training loss continues to decrease, the curve for the validation loss rapidly becomes flatter for the rest of the epochs. Tentatively, we can see that initial the model is generalising well for both datasets, before becoming too specialised to the training dataset.



The inferences from the loss evolution are backed up by the model accuracy evolution. While the loss function is tailing off, the accuracy of the validation curve is become flatter. This almost but certainly indicates the model is overfitting to the training data. It's possible that the level of complexity of my model isn't enough to stop it from becoming specialised to the training data. Even though initially the model was able to learn meaningful features from both datasets, the issue of specialisation to the training data is evident.

Final model accuracy on CIFAR-10 Validation Set

For the final accuracy of my model, I was able to achieve around 55% on the validation dataset but it's important to note that it varied between 53%-57%. On reflection, the model would've benefitted from an increase in the number of blocks making up the backbone to increase its complexity. Furthermore, implementing a form of batch normalisation for the output of the blocks in the backbone may have helped with the training stability, leading to faster convergence which is important for when the model updates the weights. I considered elements such as dropout but found it very difficult to implement into my architecture. The same goes for adding more blocks to my architecture, which on attempt, was returning a model with 10% accuracy, indicating a lack of integrity in the structure of the architecture, something of which I'd like to improve.

Bibliography

Masters, D., & Luschi, C. (2018). Revisiting Small Batch Training for Deep Neural Networks.

PyTorch. (2023). *TRAINING A CLASSIFIER*. Retrieved from https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html