ECS7026P: CIFAR-10 Image Classification

Name: Eleanor Prashamshini
Student ID: 220772291
Course: Neural Networks and Deep Learning
Course Code: ECS7026P

Dataset

The CIFAR-10 dataset is a collection of 32 x 32 colour images belonging to one of the following ten categories – {“airplane”, “automobile”, “bird”, “cat”, “deer”, “dog”, “frog”, “horse”, “ship”, “truck”}. There is no image that is mapped to more than one class, that is, they are mutually exclusive. The 60,000 images are divided into training and testing data in the ratio 5:1. Both, training and testing datasets, are downloaded and loaded into memory from `torchvision.datasets.CIFAR10`.

Components of the Model

Linear Layer: A fully connected layer, where every neuron is connected to another neuron in the next layer. A Multi-Layer Perceptron (MLP) is a combination of multiple linear layers. They are often used in combination with an activation function.

Convolution Layer: This layer is most used on 2D input such as images. A sliding window of fixed dimensions, called a filter, traverses the input image computing the product of the filter and the local image patch. This results in a feature map. This layer is also often followed by an activation function.

Pooling Layer: This is a subsampling layer used on 2D data, to summarise the local neighbours. It is a generalization mechanism and helps avoid overfitting the network. There are two types of pooling used in this model –

- Average pooling: Computes the average value of the given window.
- Max Pooling: Takes the maximum of values in the window.

Activation Functions: A non-linear function applied after the Linear or Convolution layer. This is useful in building complex relationships between input and output.

- Sigmoid: Maps input to value between 0 to 1.
- ReLU: Maps non-positive values to zero and leaves positive values as is.
- Leaky ReLU: Similar function to ReLU, however, instead of mapping negative values to zero it allows a small gradient (0.01 to 0.3 typically) for them.

Architecture

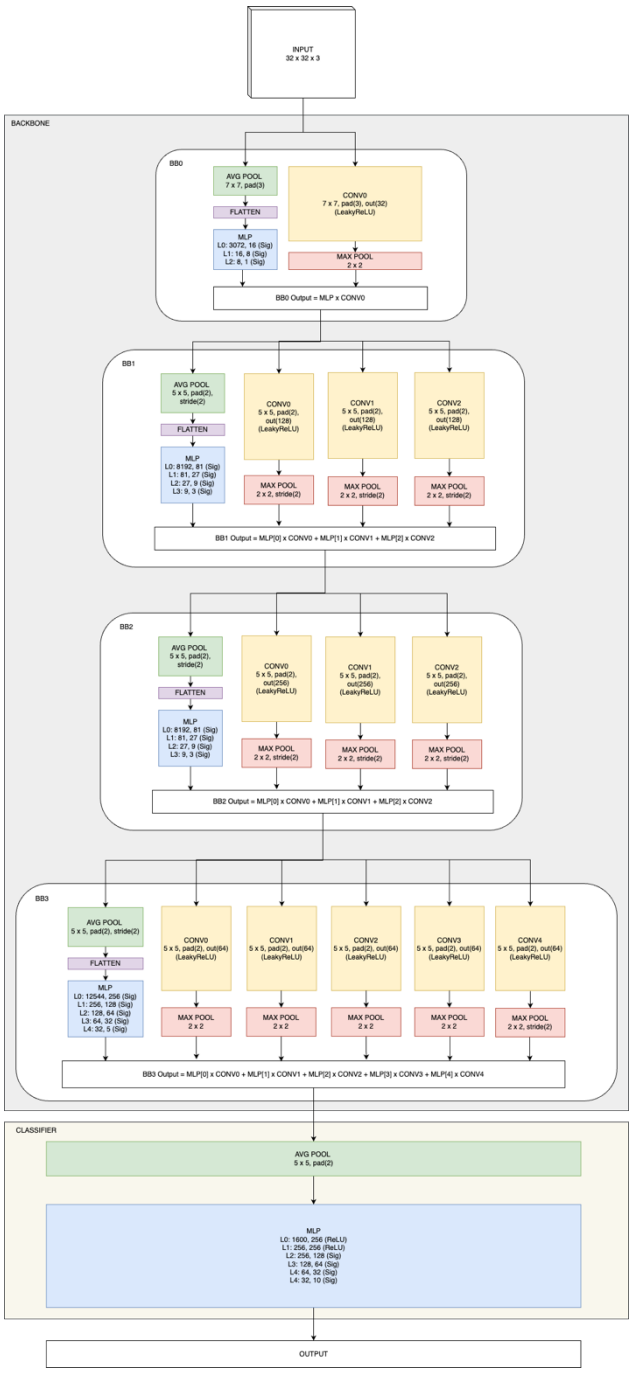

Fig.1: Neural Network Architecture

There are two main components in the neural network are –

1. Backbone: There are four blocks within this component. Each block takes the input and processes it in parallel through an MLP and a chosen number of convolutions. The MLP outputs a tensor with length same as the number of convolutions. Further, each value in this tensor is multiplied by the output of each convolution block. This is then taken as input to the next block.
2. Classifier: This layer takes a spatial average pool of the final output from the backbone, flattens it, and passes it through an MLP.

There are several combinations of architectures that have been tried within the blocks. As well as varieties of activation functions and normalization methods that have been tried before concluding on this as the final model.

Loss and Optimization

Loss is an objective function that is minimized during the training process. It tells us how accurate the network predictions are compared to the actual values. It works in combination with an optimizer that updates the parameters based on the loss calculated. We use the cross-entropy loss (`CrossEntropyLoss()`) to evaluate the performance of the model and stochastic gradient descent (`SGD()`) to update the parameters during each minibatch. Additionally, we employ a scheduler to decrease the learning rate according to a cosine annealing scheduler (`CosineAnnealingLR()`).

Data Augmentation

Augmentation is a technique to generate modified versions of the data. Adding these variations to the actual training set helps increase the dataset and improve generalization. We use the following transformation methods – `RandomCrop()`, `RandomHorizontalFlip()`. It is important to note that these transformations are only applied to the training dataset and not the testing.

Normalisation

This is used as a pre-processing step to standardize the range of the pixel values. This is applied to both the training and the testing dataset. The parameters for mean and standard deviation are calculated from the natural color images of the CIFAR-10 dataset.

```
transforms.Normalize(mean=[0.4914,
0.4822, 0.4465], std=[0.2023,
0.1994, 0.2010])
```

Dropout

This is a regularization method, where random nodes are dropped in every iteration during training to avoid overfitting the network to the training data. This is used in the classifier MLP and is set to a probability of 20%.

Hyperparameters

Training

- `num_epochs = 60 or 100`
- `batch_size = 256`

Optimizer

- `lr = 0.1`
- `momentum = 0.9`
- `weight_decay = 1e-4`
- `nesterov = True`

Scheduler

- `T_max=200`
- `eta_min=0.0001`

Training

During the training of each epoch, the following steps are executed looping over minibatches –

- First the network is set to training mode and the optimizer sets gradients of all parameters to zero.
- Perform the forward pass through the network and calculate the loss between the predicted and expected outputs.
- Using backpropagation, compute the gradient of loss with respect to the parameters and update them.
- Add the training loss, training accuracy and testing accuracy to the accumulator object.
- Calculate the average training loss and accuracy of current epoch.

At the end of every epoch, evaluate the model with the testing dataset and run the scheduler to update the learning rate.

Result

Training Loss	**0.164**
Training Accuracy	**0.955**
Testing Accuracy	**0.852**

Table.1: Results at 60 epochs

Training Loss	**0.091**
Training Accuracy	**0.977**
Testing Accuracy	**0.879**

Table.2: Results at 100 epochs

Table.1 and 2 records the results obtained after training the network for 60 and 100 epochs respectively.

The graphs in Fig.2 and Fig.3, shows the evolution of the training loss (blue), training accuracy (pink) and test accuracy (green) over 60 and 100 epochs. The x-axis is the number of epochs.

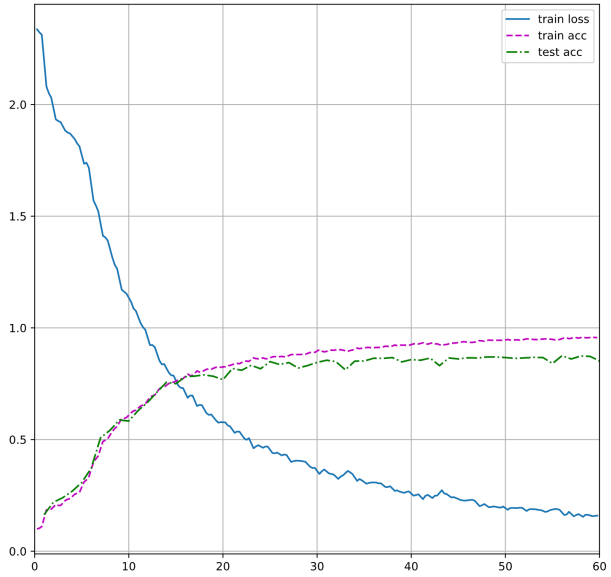

Fig.2: Training loss, training accuracy and testing accuracy curves for 60 epochs

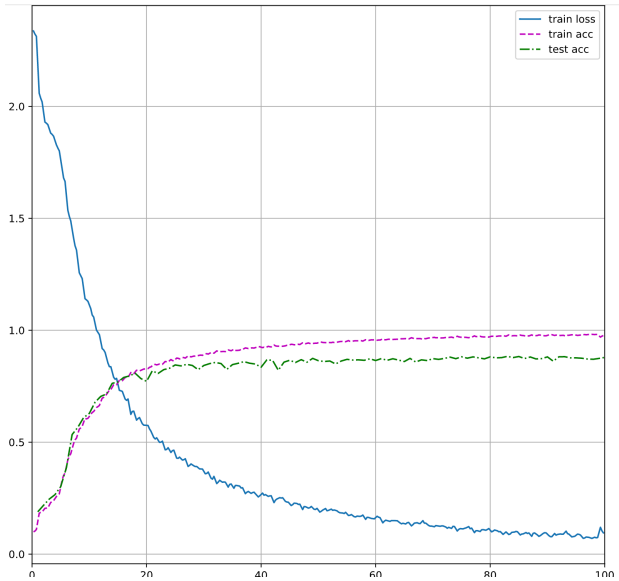

Fig.3: Training loss, training accuracy and testing accuracy curves for 100 epochs

Reflection

This project was a great opportunity to experiment with the building blocks of neural networks and understand how they could be leveraged in image processing. The combination of a complex architecture, data augmentation and regularisation methods enabled the model to achieve an accuracy of 87.9% on the testing dataset.