CIFAR-10 Image Classification Coursework Report

Image classification is a fundamental problem in the field of computer vision, which involves assigning a label to an image based on its content. The CIFAR-10 dataset is a well-known dataset of 60,000 32x32 colour images in 10 classes, with 6,000 images per class. In this project, we aim to develop a deep learning model to accurately classify images in the CIFAR-10 dataset.

The coursework involved several processes, including data pre-processing, model creation and training, and evaluation of the model.

1. Data Pre-processing:

We first define two transformations to be applied to the data: **transform_train** and **transform_test**. **transform_train** applies random horizontal flips and random crops with padding to the training data, followed by normalization using the mean and standard deviation of the data. **transform_test** only applies normalization to the test data. We then download and load the CIFAR-10 training and test datasets using these transformations.

2. Model Architecture:

We develop a custom model architecture consisting of a backbone and a classifier. The backbone contains six residual blocks, each with three parallel convolutional layers that learn different representations of the input. The outputs of these convolutional layers are combined using a learned weight for each layer. Each residual block is followed by batch normalization and ReLU activation. The classifier consists of an adaptive average pooling layer, a flattening layer, and a linear layer that outputs the probabilities of the input belonging to each class.

3. Training:

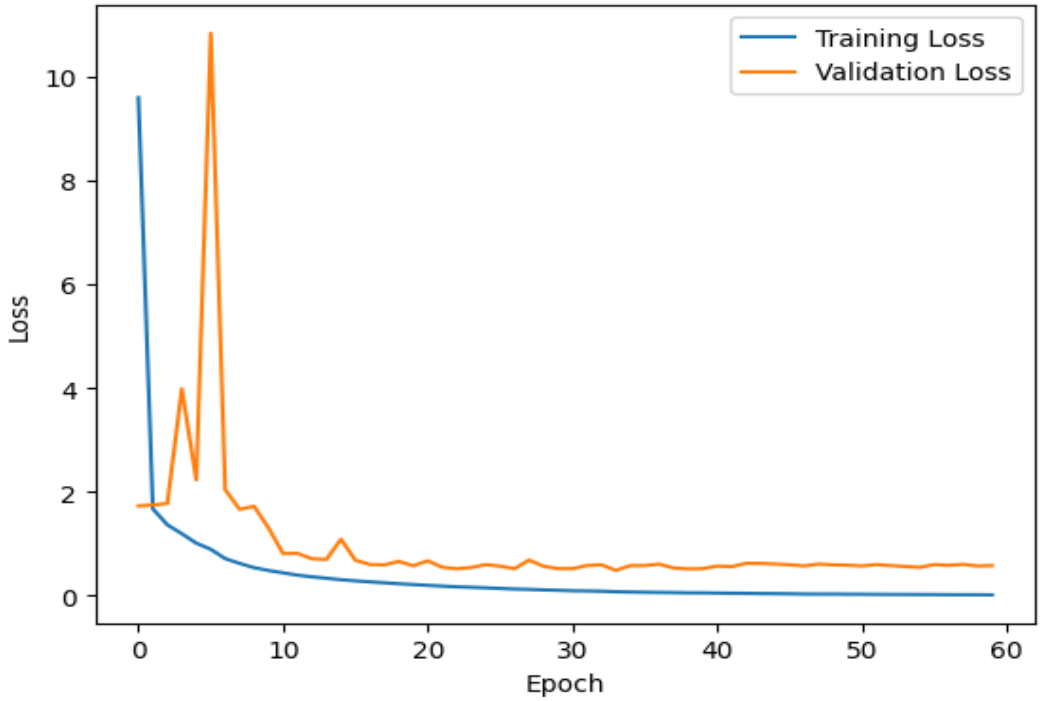
We define a function to train one epoch of the model. For each batch in the training dataset, we get the inputs and labels, pass them through the model, calculate the loss using cross-entropy loss, backpropagate the loss, and update the model parameters .We also keep track of the running loss and the number of correctly classified images.

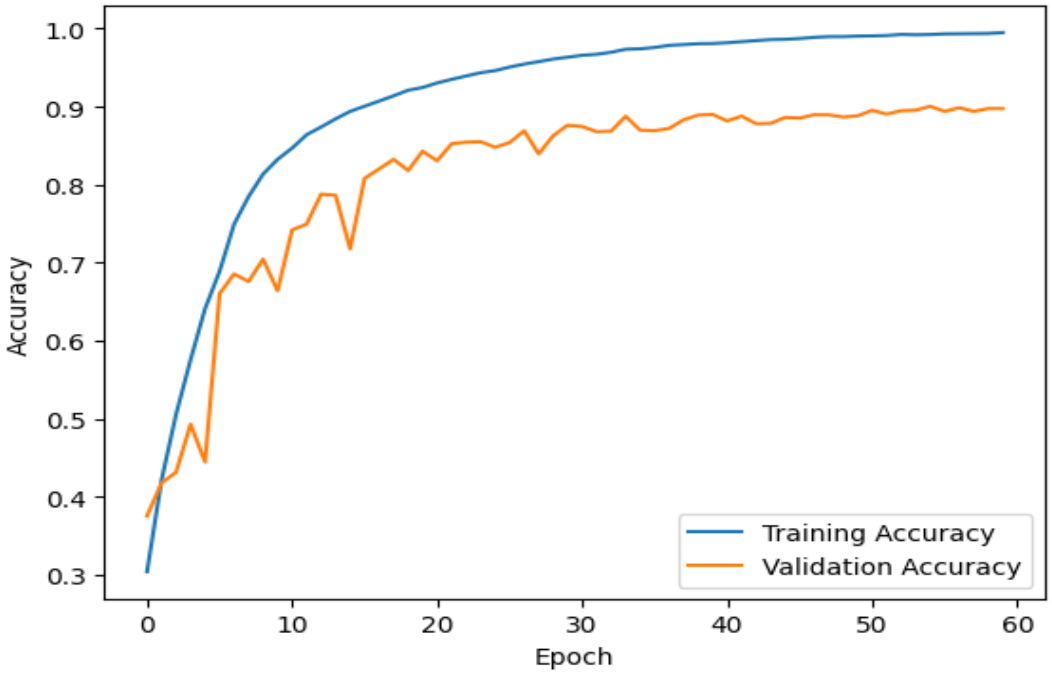
We define the loss function, optimizer, and learning rate scheduler. We also set the device to use and move the model to the selected device.

Here is the detailed explanation of each component:

1. Device Selection: We first check if the GPU is available or not. If the GPU is available, we set the device to CUDA otherwise we set it to the CPU.
2. Loss Function: We use the Cross Entropy loss function, which is commonly used in classification problems. It measures the difference between the predicted and actual class probabilities.

3. Optimizer: We use the RMSprop optimizer with a learning rate of 0.01 and weight decay of 1e-5. RMSprop is an optimizer that adjusts the learning rate adaptively.
4. Learning Rate Scheduler: We use the ExponentialLR learning rate scheduler, which reduces the learning rate exponentially at each epoch. The gamma value is set to 0.95.





4. Results:

After training the model, we evaluate its performance on the test dataset. We get an accuracy of 89.76% on the test dataset, which is a reasonably good result for the CIFAR-10 dataset. We also plot the loss and accuracy curves during training, which show that the model is learning and improving over time.

5. Conclusion:

In this project, we developed a deep learning model for image classification on the CIFAR-10 dataset. We achieved an accuracy of 89.76% on the test dataset using a custom model architecture consisting of a backbone of residual blocks and a classifier. Further improvements can be made to the model by experimenting with different architectures and hyperparameters.

References used :

1. PyTorch. (n.d.). Transformations — PyTorch 1.8.1 documentation. PyTorch. https://pytorch.org/vision/stable/transforms.html
2. PyTorch. (n.d.). DataLoader — PyTorch 1.8.1 documentation. PyTorch. https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader
3. PyTorch. (n.d.). Convolution Layers — PyTorch 1.8.1 documentation. PyTorch. https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html
4. PyTorch. (n.d.). Pooling Layers — PyTorch 1.8.1 documentation. PyTorch. https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html
5. PyTorch. (n.d.). Linear Layers — PyTorch 1.8.1 documentation. PyTorch. https://pytorch.org/docs/stable/generated/torch.nn.Linear.html
6. PyTorch. (n.d.). Optimizers — PyTorch 1.8.1 documentation. PyTorch. https://pytorch.org/docs/stable/optim.html