Neural Networks & Deep Learning
Coursework: CIFAR 10 Dataset- Image Classification using CNN

Natisha Mallick: 220867092

Aim:
Implement a specific model to solve the problem of CIFAR 10 classification ie. classify every image in terms of 1 out of the 10 classes.
Build a model on the training set & evaluate it on the test set.

Model should include:
- An architecture to process images based on Convolutional Neural Networks
- Model architecture consists of Backbone (B1, … BN) and classifier.

Step1: Read dataset and create data loaders:
Several transformations are applied to the training images.
1. **transforms.ToTensor() function**: Converts the images into PyTorch tensors.
2. **transforms.Normalize() function**: To normalize the tensors by subtracting the mean of 0.5 and dividing the standard deviation also 0.5.
3. **transforms.RandomAffine():** To randomly rotate and translate the images within a specific range and randomly scale the images.
4. **transforms.ColorJitter():** To adjust the brightness, contrast, saturation & hue of the images.
5. **transforms.Normalize():** To normalize the pixel values of the images in both training & test datasets using the mean & standard deviation values.

Batch Size Used-
a. Training Dataloader: 75
b. Test Dataloader: 50
Learning Rate: 0.001
epochs: 50

Step2: Creating the Model:
1. BLOCK: The block consists of the following implementtations:
a. Adaptive Average Pooling
b. Full Connected Linear Layer
c. K- Convolutional Layers
d. Residual Connection

2. Overall Architecture:
a. Backbone: (as per the requirements mentioned)

- Batch Normalization
- ReLu
- MaxPool2D

b. Classifier: To process output of last block.

- AdaptiveAvgPool2D
- Flatten
- Linear Layer

Step3: Loss Function, Optimizer & Learning Rate Scheduler:

a. **Activate GPU** for training:

Using the following, the GPU is activated:

```
[11]: device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
      print(device)

      cuda:0
```

b. **Loss Function: CrossEntropyLoss()**

c. **Optimizer: Adam** is used as it dynamically adjusts the learning rate of each parameter during training. Adam yields high accuracy compared to other optimizers.

d. **Learning Rate: 0.001**

e. **Learning Rate Scheduler: CosineAnnealingLR** is used as this will help the model to converge faster and achieve better accuracy by allowing to be able to jump out of the local minima during the training stage.

Step4: Training the Model:

a. **training_epoch():**
   Inputs: Model, Dataloader, Criterion, Optimizer- Adam_Optim, Device, Accumulation_Steps

   In the training stage, variables are assigned for total samples, correct predictions & running loss.
   After this, a ***for loop*** is used to iterate through the training data, to perform the following-

- Transferring images & labels back to device
- Forward pass
- Calculating the loss
- Computing gradients through backpropogation
- Updating model parameters
- Incrementing counter for total samples & correct predictons.

   The results are returned in the form of average loss & accuracy for each epoch.

b. **validation_epoch():**
   Inputs: Model, Dataloader, Criterion, Device

   In the validation stage, variables are assigned running loss, correct predictions & total samples.
   After this, **torch.no_grad()** is used to disable the gradient calculation.
   The ***for loop*** is used to perform the following-

- Transferring images & labels back to device
- Calculating the loss
- Updating counters for total samples & correct predictions

   The results are returned in the form of average loss & accuracy for each epoch.

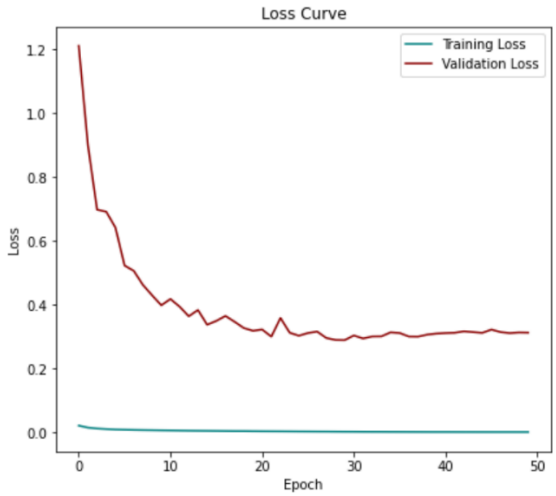
Step 5: Model Accuracy

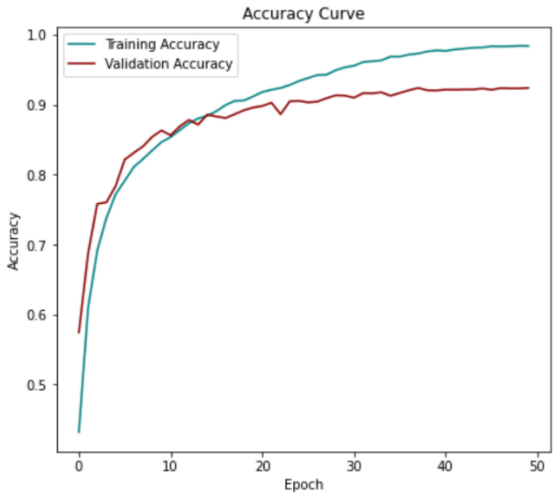
Training Accuracy & Validation Accuracy:

```
Epoch [5/50], Training Loss: 0.0087, Training Accuracy: 0.7717, Validation Loss: 0.6414, Validation Accuracy: 0.7834
Epoch [10/50], Training Loss: 0.0059, Training Accuracy: 0.8463, Validation Loss: 0.3979, Validation Accuracy: 0.8630
Epoch [15/50], Training Loss: 0.0044, Training Accuracy: 0.8840, Validation Loss: 0.3369, Validation Accuracy: 0.8856
Epoch [20/50], Training Loss: 0.0033, Training Accuracy: 0.9116, Validation Loss: 0.3179, Validation Accuracy: 0.8957
Epoch [25/50], Training Loss: 0.0025, Training Accuracy: 0.9337, Validation Loss: 0.3026, Validation Accuracy: 0.9053
Epoch [30/50], Training Loss: 0.0018, Training Accuracy: 0.9532, Validation Loss: 0.2887, Validation Accuracy: 0.9128
Epoch [35/50], Training Loss: 0.0012, Training Accuracy: 0.9686, Validation Loss: 0.3130, Validation Accuracy: 0.9125
Epoch [40/50], Training Loss: 0.0009, Training Accuracy: 0.9774, Validation Loss: 0.3093, Validation Accuracy: 0.9201
Epoch [45/50], Training Loss: 0.0007, Training Accuracy: 0.9816, Validation Loss: 0.3115, Validation Accuracy: 0.9229
Epoch [50/50], Training Loss: 0.0006, Training Accuracy: 0.9838, Validation Loss: 0.3122, Validation Accuracy: 0.9238
Training finished in 0.00 seconds.
```

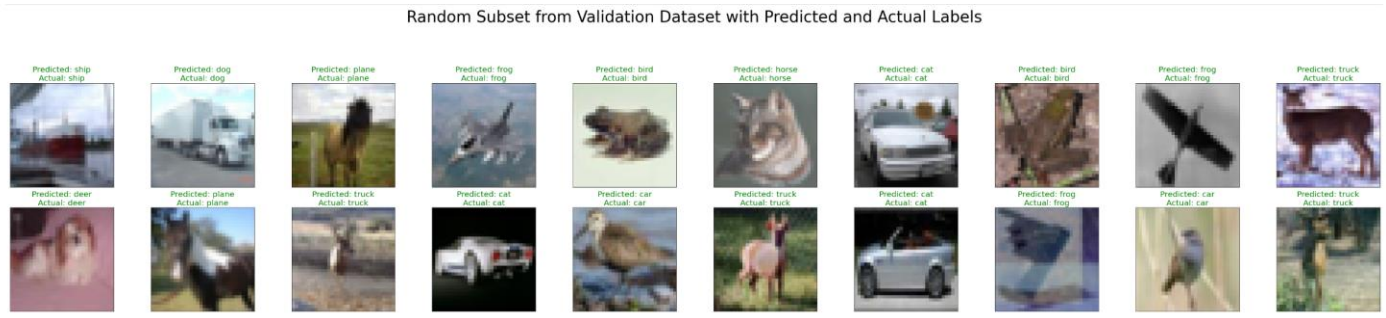
After completing 50 epoch, the model achieved-
The Final Training Accuracy- 98.38%
Validation Accuracy is 92.38%
Hence, the **Model Accuracy** as can be seen from above is **92.38%**

The Loss Curves & Accuracy Curves obtained for the model are as below:





The validation set visualization section presents 20 random images with their predicted and actual labels, of which the model has correctly classified all 20 images in the random selection.



In conclusion, the task to correctly classify images classes has been successfully achieved on the CIFAR-10 dataset.