Neural Networks and Deep Learning Coursework (ECS7026P)
CIFAR-10 Classification

Name: Aman Naredi
Student ID Number: 230290873

Task 1: Reading Dataset and Creating Data Loaders
For this task, I have defined a function, **load_data_CIFR**. This function handles both the downloading of the dataset and the application of necessary transformations to enhance the model's learning ability.

Transformations:
1. **Training Data Transformations:** Includes **random horizontal flipping**, **random cropping** and converting to a **tensor** and **normalizing** using mean and standard deviation values of (0.5, 0.5, 0.5) across the RGB channels.
2. **Testing Data Transformations:** Consists of converting images to **tensors** and **normalizing** them in the same manner as the training images.

Data Downloading and Loading:
The CIFAR-10 dataset is split into training and testing sets and downloaded using **torchvision.datasets.CIFAR10** and transformations are applied.
Then **Data Loaders** for both training and test datasets are created, which are configured to **shuffle** the training data to ensure random distribution of batches in the batch size defined in the parameters of the function.
The function returns the Data Loaders of train and test datasets.

Task 2: Creating the Model
Basic Architecture
This model **BasicNet** is built as per the architecture (Fig1, Fig2 and Fig3) for the CIFAR dataset classification. It consists of intermediate blocks and an output block. Each intermediate block contains a sequence of convolutional, batch normalization, and ReLU activation layers. The output block finalizes the predictions by averaging features and applying a fully connected layer.
In this **architecture** the **output channels** are **fixed** and a limited number of convolutional and fully connected layers are used.
Batch size was kept 64

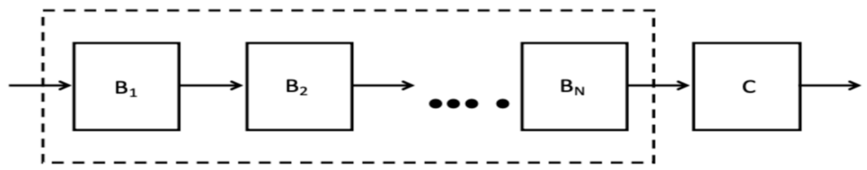
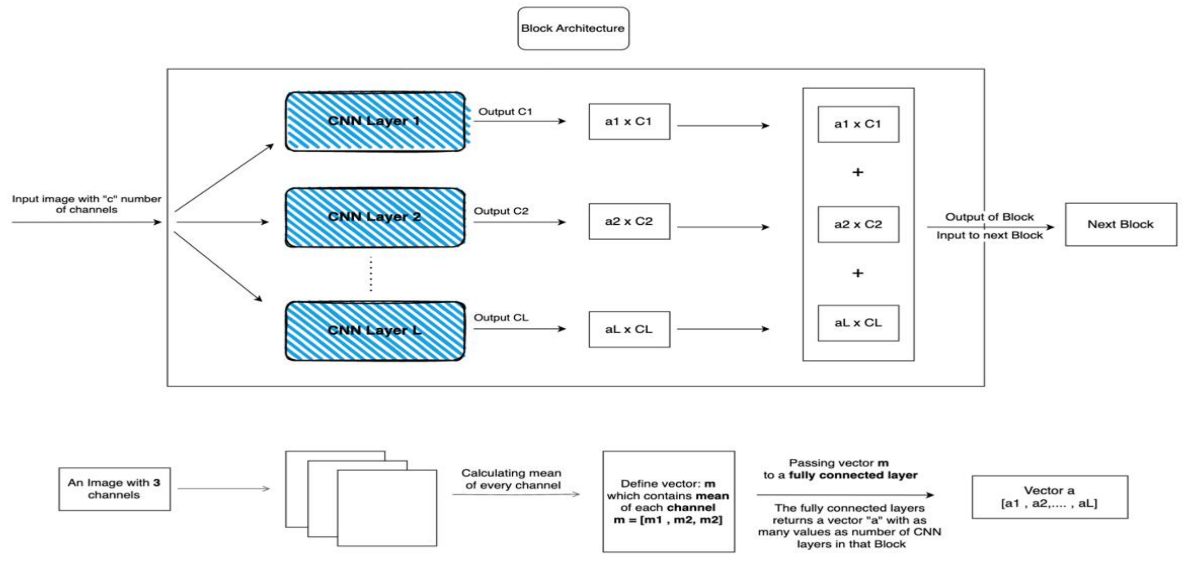
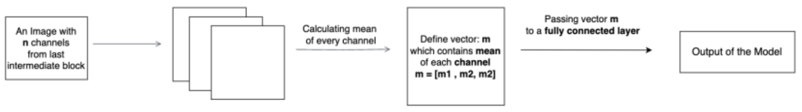


Fig1. Architecture of the Model



Block Architecture
Input image with "c" number of channels
CNN Layer 1
CNN Layer 2
CNN Layer L
Output C1
Output C2
Output CL
a1 x C1
a2 x C2
aL x CL
+
Output of Block
Input to next Block
Next Block
An Image with 3 channels
Calculating mean of every channel
Define vector: m which contains mean of each channel m = [m1 , m2, m2]
Passing vector m to a fully connected layer
The fully connected layers returns a vector "a" with as many values as number of CNN layers in that Block
Vector a [a1 , a2,.... , aL]

Fig2. Block Architecture



An Image with n channels from last intermediate block
Calculating mean of every channel
Define vector: m which contains mean of each channel m = [m1 , m2, m2]
Passing vector m to a fully connected layer
Output of the Model

Fig3. Out Put Block Architecture

2.2 Advanced Net Model

The model Adv_model is the enhanced version of BasicNet. It is also built as per the architecture (Fig1, Fig2 and Fig3). To improve the efficiency of the BasicNet, several changes in the Intermediate and the Output Blocks are done.

Advance Net Architecture:
Increased Batch Size to 128 as it enhances computational efficiency and stability during training.

Intermediate Block : The `Adv_IntermediateBlock` class defines the Block architecture of my advance model. It takes the number of input channels, output channels (required from the second block) and the desired number of convolutional layers. This block first computes the mean of input tensors (m) using **adaptive average pooling** then applies a fully connected layer to generate coefficients (a), and then weights the output of each convolutional layer using these coefficients.
Advancements and changes made as compared to BasicNet Intermediate Block.

- **Adaptive Average Pooling**: Added an Adaptive Average Pooling Layer**.** It ensures that the output tensor has a **fixed size**. This helped in handling input tensors of variable sizes.
- **Output Channels:** Instead of using a fixed number of output channels (3) they are set to be dynamic. It helped in extracting more features from the images.
- **Max Pooling and Dropout:** Added **`MaxPool2d`** and **`Dropout`** layers after each convolutional layer. Max pooling down sampled the feature maps and dropout introduced regularization to prevent overfitting.

Output Block: The `Adv_OutputBlock` Class defines the output block architecture of my Advance Net model.
This block takes the input from the last intermediate block and then it computes the mean of the input tensor across the width and height dimensions using **adaptive average pooling**. Then, it passes the resulting tensor through the **sequence** of **linear layers** to generate logits as the final output of the model.
Advancements and changes made as compared to BasicNet Output Block

- It utilizes Adaptive Average Pooling to ensure a fixed-size output tensor regardless of the input size.
- Constructed a **sequence** of 3 **linear** layers with **leaky ReLU** activations and 2 **dropout layers (p = 0.2 and p = 0.3)** and each layer's **output** tensor shape is set to the **double** of its **input** tensor. This allowed for more complex transformations and feature extraction before producing the final output. I had experimented with more layers and various dropout fractions.
- Dropout layers to enhance regularization and prevent overfitting during training. To improve the model's generalization ability.

Advance Net Model: The `AdvanceNet` class defines my complete neural network model.
It takes several parameters:

- **`conv_arch`:** A list specifying the architecture of the convolutional layers in intermediate blocks. Each element of `conv_arch` is a tuple (out_channels, num_convs) indicating the number of output channels and the number of convolutional layers for each block.
- **`outBlock_out_channels`:** Number of output channels for the output block.
- **`num_classes`:** Number of classes. Default is set to 10
- **`in_channels`:** Number of input channels. Default is set to 3.

It iterates through each intermediate block , applying them sequentially to the input tensor `x`. Then, it passes the resulting tensor through the output block to get the final output, which represents the logits for each class.

Task 3: **Training the Neural Network**
Training of the neural network is done in 2 steps.

Step1. Defining Hyperparameters, Optimizer and Loss function
The hyperparameters set for Adv_model were:

- **`IN_CHANNELS`** --> Number of Input channels = **3**, as an image as 3 channels.
- **`OUT_BLOCK_OUT_CHANNELS`** = **32**, Defines the number of output channels for the first linear layer of the output block network.
- **`NUM_CLASSES`** --> Number of classes = **10.**
- **`CONV_ARCH`** --> It specifies the architecture of the convolutional layers in the intermediate blocks network. It's a list of tuples, where each tuple contains two values:
  - The first value represents the number of convolutions per block.
  - The second value indicates the number of blocks.
  - For example, (512, 1) means there's one block with 512 convolutions.
  - After experimenting with different combinations such as `[(64, 1), (128, 1), (256, 2), (512, 2), (512, 2)]`, `[(128, 2), (128, 2), (256, 4), (512, 2), (512, 4)]`. But these architectures did not give an accuracy of more than 86%.**Used [(512, 1), (512, 1), (512, 2), (512, 2), (512, 4)]**

The hyperparameters set for BasicNet_model were:

- `IN_CHANNELS = 3  # Number of channels in CIFAR-10 images`
- `NUM_BLOCKS = 7  # Number of intermediate blocks`
- `NUM_CONVS_PER_BLOCK = 5  # Number of convolutions per block`
- `NUM_FCS_OUT = 1 # Number linear layers in output block`
- `NUM_CLASSES = 10  # Number of classes in CIFAR-10`

The model utilizes the **Cross-Entropy Loss function** due to its effectiveness in multi-class classification tasks. **Adam optimizer** was chosen for its adaptive learning rate capabilities, helping in quicker convergence. In this case, it's used to optimize the parameters of the model with a learning rate (lr) of 0.001.

Step2. Defining Training and Testing functions

Training Function (train):

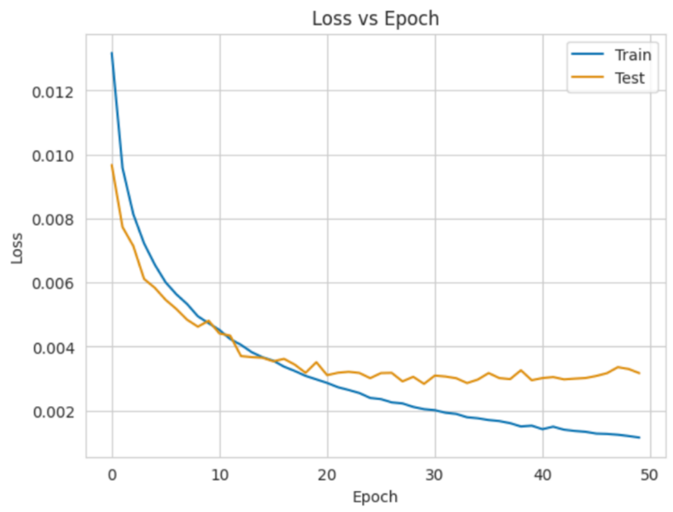
- It sets the model to training mode and initializes variables to track total loss and correct predictions.
- It iterates over the training data, moving the data and labels to the appropriate device (CPU or GPU).
- For each batch, it performs a **forward pass** through the model, **computes the loss**, performs **backpropagation**, and **updates** the model's parameters.
- It calculates the **accuracy** by comparing the model's predictions to the true labels and accumulates the loss.
- Finally, it computes the train loss and accuracy over the entire training dataset and returns these values.

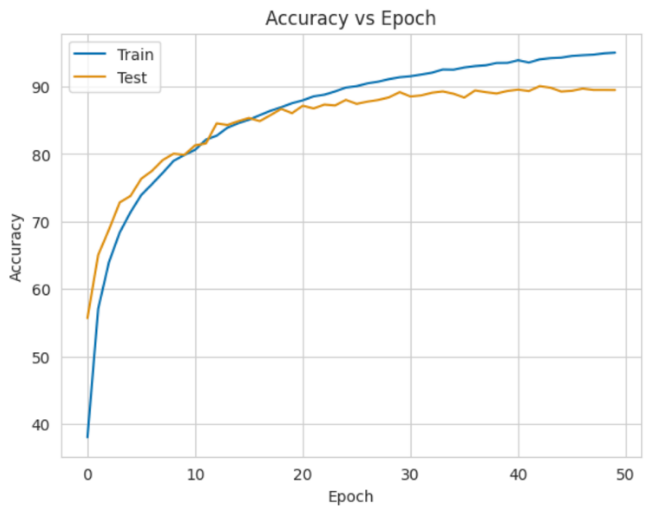
Testing Function (test):

- It sets the model to evaluation mode, disabling operations like dropout for consistency.
- It iterates over the test data, moving it to the device and computing the model's output.
- It calculates the loss and accuracy similarly to the training function but without backpropagation.
- Finally, it computes and returns the test loss and accuracy over the entire test dataset.

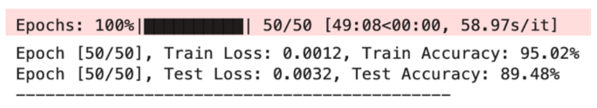
Task 4: Training and Testing Loop for the Neural Network Model and Plotting

I have trained my Adv_Net model for 50 epochs and that of Basic Net for 35 epochs as its accuracy was not increasing.
The **train and test loss** for each epoch and **train and test accuracy** for each epoch were **stored** in **lists** for the plotting purpose.

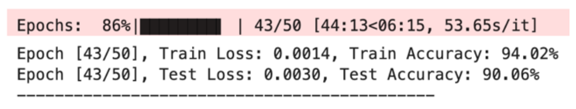




The `BasicNet` model achieved a respectable accuracy, but with the advanced architecture adjustments in `AdvanceNet`, the model's performance significantly improved, demonstrating high accuracy in both training and testing phases. The final model achieved **training accuracy as high as 95.02%** and a **test accuracy of 89.48%.** The **highest test accuracy** achieved by Adv_model was **90% at 43rd epoch**.



```
Epochs: 100%|██████████| 50/50 [49:08<00:00, 58.97s/it]
Epoch [50/50], Train Loss: 0.0012, Train Accuracy: 95.02%
Epoch [50/50], Test Loss: 0.0032, Test Accuracy: 89.48%
--------------------------------------------
```



```
Epochs:  86%|████████  | 43/50 [44:13<06:15, 53.65s/it]
Epoch [43/50], Train Loss: 0.0014, Train Accuracy: 94.02%
Epoch [43/50], Test Loss: 0.0030, Test Accuracy: 90.06%
--------------------------------------------
```

`Note`: Because of space limits I was not able to provide architectures of my BasicNet model and Adv_Net model. In my code file you can find the architectures of both the models. Also the Loss vs Epoch curve and Accuracy vs Epoch curve for Basic Net model can be found in my code file.