**Neural Networks and Deep Learning (ECS659P/ECS7026P)**
**Utkarsh Kumar Sharma (230671946)**
**MSc Big Data science**

---

## Loading the CIFAR-10 dataset into appropriate DataLoaders

The code in task 1 cell of notebook file prepares the CIFAR-10 dataset for **training** and **testing**. It defines transformations for **data augmentation** and **normalization** for both training and testing sets. Then, it downloads the CIFAR-10 dataset, stores it in the 'data' directory, and creates DataLoader objects for the training and testing datasets. These DataLoader objects enable generating batches of examples for training and testing the model.

## Basic Architecture

The **IntermediateBlock** class represents a building block consisting of parallel convolutional layers followed by a fully connected layer to dynamically adjust the weights of convolutional outputs. It takes input channels, output channels, number of convolutional layers, kernel size, and activation function as parameters. The **OutputBlock** class represents the final block of the model, which computes the output predictions based on the aggregated features. It consists of fully connected layers for classification. The **BasicModel** class is composed of multiple IntermediateBlocks followed by an OutputBlock for classification. It takes input channels, hidden units, output shape, number of blocks, and number of convolutional layers as parameters.

The main variables in the code are **input_channels, hidden_units, output_shape, num_blocks, and num_conv_layers**. These variables control the architecture and behavior of the model.

The forward method of the BasicModel class defines the forward pass of the model. It passes the input through the first convolutional block, then through additional intermediate blocks, and finally through the output block to compute the final predictions.

**Computation of Vector 'm':** Initially, an image with 'C' colour channels is processed. Then, a 'C'-dimensional vector 'm' is computed by averaging each colour channel independently. This vector 'm' is passed through a fully connected layer, resulting in another vector 'a' of length 'L', corresponding to the number of convolutional layers within the block (e.g., [a1, a2, ..., aL]). This follows the formula:

$$a = f(Wm + b),$$

***where 'f' represents the activation function, 'W' denotes the weight matrix, and 'b' is the bias vector.***

**Weighted Sum of Convolved Images:** The same image is sequentially convolved through 'L' convolutional layers within the block (denoted as C1, C2, ..., CL). Each convolutional layer applies learned filters to the image, detecting features like edges, textures, and patterns. Following each convolutional operation, batch normalization enhances stability and speed, while ReLU activation introduces non-linearity for complex pattern learning. Max pooling reduces spatial dimensionality, and dropout regularization prevents overfitting. Each convolutional layer produces its own output image. The subsequent step involves computing the weighted sum of these images using weights derived from vector 'a', ensuring that the output image of each convolutional layer maintains the same shape.

## Advanced Architecture

The basic architecture is improved for its accuracy following these optimization:-

**Adding below attributes in convolution layer.**

- "BatchNorm2d"- It normalizes the activations of each channel across a mini-batch, reducing internal covariate shift.
- "activation"- It introduce non-linearity into the neural network, allowing it to learn complex patterns in the data.
- "MaxPool2d"- Max Pooling is a downsampling operation that reduces the spatial dimensions (width and height) of the input tensor. It applies a sliding window of size kernel_size over the input tensor and outputs the maximum value within each window. This helps in reducing computational complexity and controlling overfitting. Parameter used were **kernel_size=2, stride=2, padding=1.**
- "Dropout2d"- Dropout is a regularization technique used to prevent overfitting in neural networks. The **dropout_rate was kept at 0.4.**

**Applying Sigmoid activation function: -**

This is utilized to compute coefficients for the weighted sum in the IntermediateBlock class. These coefficients, obtained through a fully connected layer, determine the significance of each convolutional output in the final aggregated result. Applying Sigmoid ensures that the coefficients are bounded between 0 and 1, resembling probabilities, thereby regulating the influence of each output. This normalization fosters stability and effective learning within the neural network by facilitating a controlled combination of convolutional outputs.

**Added more intermediate bocks with variation in attributes: -** 4 intermediate with one output block is found to be better after running on multiple combinations.

## Description of hyperparameters and techniques employed for training

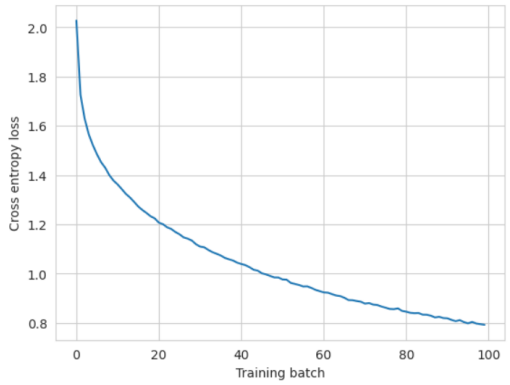
Various hyperparameters and training techniques were explored, including different learning rates (0.0005, 0.0015), batch sizes (60 and 128), numbers of blocks (7 and 10), numbers of convolutional layers (3 and 4), and the use of optemizers like Adam, AdamW, SGD, RMSprop. The improved neural network demonstrated enhanced performance compared to the basic model, achieving a test accuracy of 75.00%, marking a significant improvement from 45.00% test accuracy of basic model.

Few of the stats are captured in below table: -

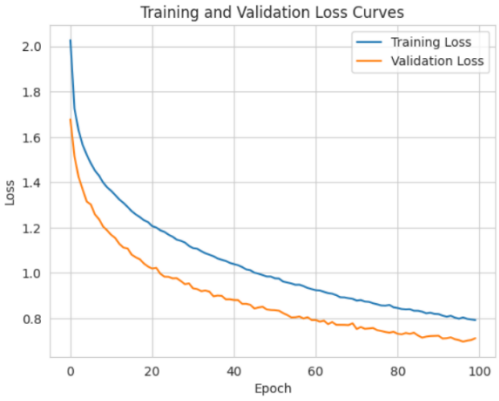
| | | |
|---|---|---|
| Epoch = 40<br>Batch size 128<br>Conv layer = 2'<br>Intermediate block 1<br>LR = 0.0015 | Training accuracy: 0.4521. | Testing accuracy: 0.4500. |
| Epoch = 40<br>Batch size 128<br>Conv layer = 2'<br>Intermediate block 1<br>LR = 0.0015<br>Kernel size for all conv layers = 8<br>Batchnormalization<br>ReLU activation | Training accuracy: 0.7245. | Testing accuracy: 0.7094. |
| Epoch = 40<br>Batch size 128<br>Conv layer for all block = 3''<br>Kernel size for all conv layers = 6<br>Intermediate block 3<br>Lr = 0.0015<br>Dropout rate for block 1 = 0.5<br>Dropout rate for block 2 = 0.5<br>Dropout rate for block 2 = 0.5<br>Batchnormalization<br>ReLU activation<br>Maxpooling<br>Optimizer = adam | Training accuracy: 0.7068. | Testing accuracy: 0.7526 |

Graph showing Cross Entropy loss on training batch.

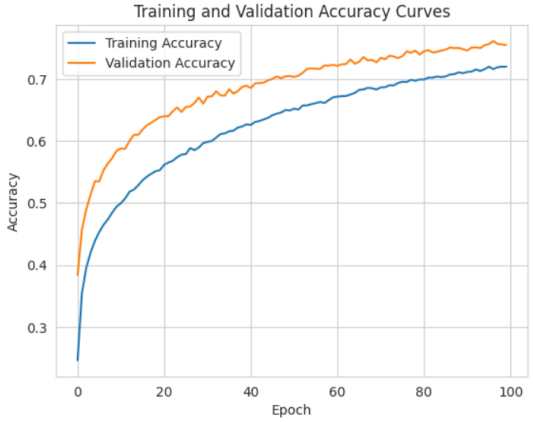
**Notes: - The values are normalized and therefore the graph looks different than that in lecture lab.**




Graph showing Training and Validation Loss Curves




Graph Showing Training and Validation Accuracy Curves




**Accuracy obtained in the testing dataset 0.7526**