Project Report

For Both Neural Network implementation, I have used PyTorch.

For data Loading for Neural Network Implementation, I have used Imageloader function from pytorch which automatically creates labels as the name of folder containing the images.

I loaded the training and testing dataset of batch size of 64 for having less computational resource.

Also while loading the training data as well as testing data I have transformed them to tensor also normalized them.

After Loading the dataset, I found that each tensor is of shape [3,32,32] which is, each image is 3 channels and of dimension 32*32.

Dataset is having 6 lasses

Fully Connected Neural Network:

For fully connected neural network I have defined a class **"Cifar_Lin"** to initialize layers as well to define forward function.

Here I have taken **3 Linear layers and Relu as activation function** to design the neural network.

Layer1: Its input size is of 3*32*32 which is 3072 and output size is of 768

Layer2: Its input size is of 768 and output size is of 192

Layer3 or Final Layer: Its input size is of 192 and output size is of 6 as no. of classes are 6 in the dataset

forward (): This method takes class object and dataset as input and feed the dataset to the 1st layer and output to the layer next to it with in between activation sequentially and finally returns the output of the final layer.

For this Network I have used **CrossEntropyLoss** as loss function and **Adam** as optimizer with learning rate as 0.00001

I have set to train the model for 30 epochs.

For each epoch Iteration I have set gradient to zero for better result and each dataset is flattened.

I set the model in training mode while training in training dataset and I evaluation mode while evaluating on validation dataset.

Finally, after successful execution of 30 epochs I found training accuracy as 59.18% and validation accuracy as 77.16%

CNN Network:

For fully connected neural network I have defined a class **"Cifar_CNN"** to initialize layers as well to define forward function.

Here I have taken **6 Convolution layers, 2 Linear Layers, 1 Maxpool2D layer of kernel size (2,2) and Relu as activation function**

Layer1: Convolution layer with input channels as 3, output channels as 12, kernel size=3, padding =1, stride=1.

Layer2: Convolution layer with input channels as 12, output channels as 32, kernel size=3,

padding =1, stride=1.

Layer3: Convolution layer with input channels as 32, output channels as 64, kernel size=3, padding =1, stride=1.

Layer4: Convolution layer with input channels as 64, output channels as 128, kernel size=3, padding =1, stride=1.

Layer5: Convolution layer with input channels as 128, output channels as 256, kernel size=3, padding =1, stride=1.

Layer6: Convolution layer with input channels as 256, output channels as 512, kernel size=3, padding =1, stride=1.

Linear1: This Layer is having input size as 512*16*16 which is 131072 and output size is of 1024

Linear2 or final Layer: This layer is having input size of 1024 and output size is of 6 as no. of classes are 6 in the dataset.

For each convolution layer I have used batch normalization which is having size of the output channel length for the respective convolution layer.

Note: Each sequential Convolution layer is having the below sequence

Dataset is fed to the convolution layer, Convolution layer output is fed to batch normalization, output of which is fed to the activation function

forward (): This method takes class object and dataset as input and feed the dataset to the 1st sequential convolution layer and output of this to the maxpool2d layer.

Output of this maxpoo2d layer is feed to the next sequential convolution layer and output of this layer to next sequential convolution layers.

Dataset size at different layers:

1st sequential layer input: [64, 3, 32, 32]

1st sequential layer output: [64, 12, 32, 32]

Maxpool2D layer input: [64, 12, 32, 32]

Maxpool2D layer output: [64, 12, 16, 16]

2nd sequential layer input: [64, 12, 16, 16]

2nd sequential layer output: [64, 32, 16, 16]

3rd sequential layer input: [64, 32, 16, 16]

3rd sequential layer output: [64, 64, 16, 16]

4th sequential layer input: [64, 64, 16, 16]

4th sequential layer output: [64, 128, 16, 16]

5th sequential layer input: [64, 128, 16, 16]

5th sequential layer output: [64, 256, 16, 16]

6th sequential layer input: [64, 256, 16, 16]

6th sequential layer output: [64, 512, 16, 16]

Output of the last sequential layer is flattened which makes the size of each dataset as 512*16*16 which is 131072 which is fed to 1st linear layer which activation function and output of this layer to final linear layer with activation function. Final Laer output is returned from the method.

For this Network I have used **CrossEntropyLoss** as loss function and **Adam** as optimizer with learning rate as 0.001 and weight decay as 0.0001

I have set to train the model for 15 epochs.

For each epoch Iteration I have set gradient to zero for better result.

I set the model in training mode while training in training dataset and I evaluation mode while evaluating on validation dataset.

Finally, after successful execution of 15 epochs I found training accuracy as 57.29% and validation accuracy as 55.63%

For classical model I have used SVM.

For SVM I used SVM from Scikitlearn.

SVM:

To load the dataset, I created one method get_dataset(dataset_dir), it takes the path of the dataset directory as argument and internally uses imread method from CV2 to load image files and convert it to NumPy array and created labels from the directory name. Finally, the get_dataset() returns the flattened NumPy array and labels.

For feeding the data to SVM model I used PCA from scikit learn to use dimensionality reduction technique, and reduced the features to 100.

For SVM model I used the SVM model from scikitlearn with kernel as polynomial.

Trained the model on training dataset, after which on validation dataset I got validation accuracy as 25.45 %

On the comparison for the 3 model's accuracy, I found that the SVM as the least accurate and CNN neural as most accurate.