ECS659U/ECS659P/ECS7026P Neural Networks and Deep Learning Coursework

CIFAR-10 Classification using Pytorch

Pranjali Hande, Student ID: 220707639

Task 1: Read Dataset and Create Dataloaders

CIFAR-10 is a dataset of images of 10 different classes. To import CIFAR-10 dataset we are using the torchvision library. We are splitting the data into train and validation sets using the "Train" parameter in tourchvision.dataset. With the help of torch.utils, loading the required data by passing various parameters such as batch size or shuffle.

Task 2: Create Model

Model Creation has 2 parts: Block and Backbone

1. **Block:**
    - Each block has a Linear layer and the number of convolution blocks, where all of them take the same input(x).
    - **A linear Layer: (ActivationFunction(SpatialAveragePool(X)W))**
      Input to the linear layer is passed through spatial average pooling. We are using AdaptiveAvgPool2d over the input signal, which is providing output with 8*8 size.
      This pooling output is then passed to the linear layer as input. We are using "nn.Linear" with 3(RGB) input channel and 3 Output channels (Which is equal to number of convolution layers in block). The output weights of Linear model are passed through the Activation Function.
      We are using Softmax function as an activation function in order to normalize the output. It returns a Tensor of the same dimension and shape as the input with values in the range [0, 1].
    - **Convolutional Layers:**
      We are using Convolution layers in a block that takes 3 Input channels and different output channels based on the architecture of the model.
      Padding is kept at 1 to preserve the spatial dimension of the image and Stride is equal to 1 to minimize the image pixels.

    The output of Linear layer is multipled with the output of each convolution layer:
    block output: a0*o0 + a1*o1 + a3*o3+…+ ak*ok

2. **Backbone:**
    - In the backbone, we are using the "add_module" method to add each block as a child module to the current module. We are adding the number of blocks equal to the length of the convolution architecture we need in our model.
    - The output channels are each block will be used as input channels for the next block. The output of each block with used as input to the next block.
    - The output channels of the last block i.e., the output of Backbone will be used as input channels for the classifier in the model.

Task 3: Create the loss and optimizer

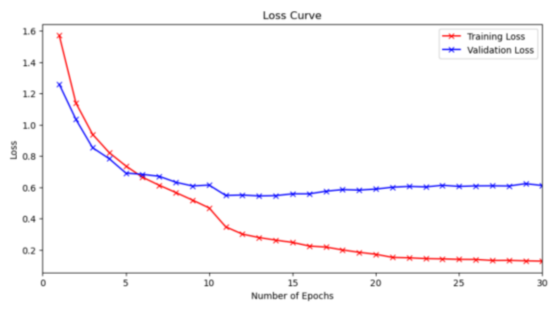
- **Loss**: In order to calculate the loss of the model we are using “CrossEntropyLoss()” from pytorch, which computes the cross-entropy loss between input logits and target. This is useful when training the classification problem.
- **Optimizer**: To update the parameters of our network we have tried multiple optimizers given by torch.optim package from pytorch. This has various optimization algorithms that update the parameters based on the computed gradients. We are using “AdamW()” optimizer with multiple parameters such as learning rate, weight decay.

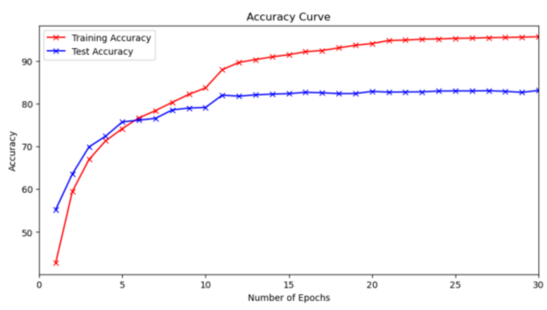
Task 4: Training Scripts to Train the Model, Hyperparameters

Training Scripts to Train the Model:

- We are training the model for several epochs so that the entire data set must be worked through the learning algorithm multiple times. Using the Train loader will get the inputs and labels for the model to train on. For each epoch, we will pass the inputs to the Model and get the predicted output from the model.
- We will calculate loss after each epoch using the loss function. We are using loss.backward() method to calculate gradients and call optimizer.step() makes the optimizer iterate over all parameters to update and use their internally stored grad to update their values.
- After this, we calculated the total training loss and accuracy for all epochs.
- We use model.eval() to test the validation set. Hers we get the inputs and labels from the Test Loader. By passing inputs to the model, we will get the predicted values.
- Loss is calculated by comparing them to labels from the test loader.
- We will calculate Validation accuracy for all the epochs.
- In order to plot the graphs for Training and Validation loss and Accuracy, this data is saved in a dictionary with several epochs.

Loss and Accuracy Curve:





Loss Curve: From the loss curve graph, we can observe that training loss and validation loss both are decreasing at the start. Somewhere around 5-7 epochs, training loss is decreasing but test loss is almost the same. We can say that model might be overfitting after this point.

Accuracy Curve: From the accuracy graph, we can observe that till 5-7 epochs, both training and validation accuracies are increasing. After this point, we can observe that training accuracy still increases but the validation accuracy is becoming stable.

Accuracy and Hyper Parameters Used:

Multiple hyperparameters were used to train the model, in order to increase the model's accuracy. Some of them are listed below:

Sr.No.	Hyperparameters Used	Validation Accuracy
1.	Conv Arch: (64, 128, 256), Without ReLU for block output, k = 2, Lr = 0.001, No.of Epochs = 30, Single FC classifier, AvgPool2D	52.00%
2.	Conv Arch: (64, 128, 256), With ReLU for block output, k = 2, Lr = 0.001, No. of Epochs = 30, AdaptivePool2D	72%
3.	Conv Arch: (64, 128, 256, 512), With ReLU for block output, k = 3, Lr = 0.001, No. of Epochs = 30, AdaptivePool2D	82.01%
4.	Conv Arch: (16,8,4), With ReLU for block output, k = 3, Lr = 0.001, No. of Epochs = 30, 1FC layer classifier	61%
5.	Conv Arch: (64, 128, 256, 512), With ReLU for block output, k = 3, Lr = 0.001, No. of Epochs = 15, dropout removed for linear	79.34%
6.	Conv Arch: (64, 128, 256, 512), With ReLU for block output, k = 3, Lr = 0.001, No. of Epochs = 15, He initialization, Scheduler Learning Rate, 3 FC layers, AdaptivePool2D.	82.97%

- Initially by using optimizers SGD, observed an accuracy of 10%. By updating the optimizer to Adam accuracy got improved and by using AdamW it increased more.
- Instead of ReLU, when using SoftMax as an activation function, we have seen a good increase in accuracy.
- We got the lowest accuracy when we didn't normalize the block output with ReLU. Hence, we added ReLU at the end of each block. After we have seen an increase in the accuracy to 72%.
- When using the Conv Arch: (16,8,4), we have observed a less accuracy of 61%
- When the model got dense by using architecture (64, 128, 256, 512), we have seen an increase in accuracy to 82.01%.
- We have also explored using a scheduler to update the learning rate on the run, and using He initialization, we have observed a good hike in accuracy.

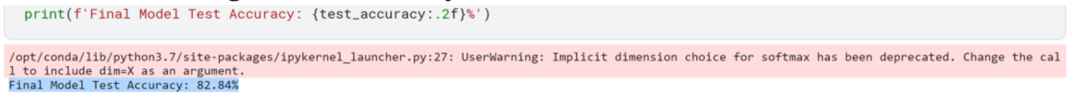
Task 5: Final Model Accuracy:

For the final model, we have used the below hyperparameters,

Convolution Architecture: (64, 128, 256, 512), k = 3, Learning rate= 0.001, LR with scheduler, Added He initialization, SoftMax as activation function, Regularizing block output with ReLU, Hidden Size: 4096, Optimizer: AdamW.

For the final classifier, we have used 3 FC layers in the sequential form passed through AdaptivePool2D(8) with nn.Linear as a classifier with dropout=0.5 with the final number of classes as 10.

The Final Model got Test Accuracy**: 82.84%**



```
print(f'Final Model Test Accuracy: {test_accuracy:.2f}%')
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:27: UserWarning: Implicit dimension choice for softmax has been deprecated. Change the cal
l to include dim=X as an argument.
Final Model Test Accuracy: 82.84%
```