IE 534: Deep Learning, Fall 2018

Homework 4: Deep Residual Neural Network on CIFAR100

Lecturer: Justin Sirignano *Submitted by: Rachneet Kaur, Oct. 5th, 2018*

Part A: Training a Resnet for CIFAR100

Implementation

The objective is to **classify** the input 32×32 dimensional coloured image into the corresponding one of the 100 classes. The CIFAR100 data set has 50000 coloured training images and 10000 images in the test set of dimensions $3 \times 32 \times 32$.

Data Loading

To load the data set, I used *torchvision.datasets.CIFAR100* and *torch.utils.data.DataLoader* functions in Pytorch.

Hence, we have:

- Dimensions of training set input X = (50000, 3, 32, 32) and labels Y = (50000, 1)
- In my code, I store training set in data_train.
- Dimensions of test set input X = (10000, 3, 32, 32) and labels Y = (10000, 1)
- In my code, I store testing set in data_test.
- I use batch size=100 both while loading the training and the testing set.
- I shuffle the training data randomly while loading whereas I do not shuffle the data while loading it in testing phase.
- While loading the training data, I perform data augmentation step (discusses further afterwards) and normalize the data.
- While loading the testing data, no data augmentation is performed but I normalize the data.

- While using transfer learning from the pretrained Resnet18 model, I re-size the training and testing images to 224×224.

- Since this is a $k = 100$ class classification problem, I have 100 units in the output layer.

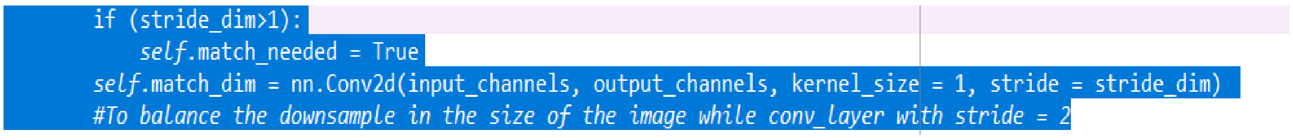
Model Architecture

Basic Block

I implemented a class *BasicBlock* to define the basic block architecture. The architecture of basic block with 2 convolution layers, batch normalization and ReLU activation is outlined below.
Note that Kernel Size, Input Channel dimension, Output channel dimension, Stride and Padding dimension are input variables to the basic block depending on the Residual network architecture.

- 2D Convolution layer with input channels as 'Input Channel dimension', output channels as 'Output Channel dimension', Kernel size = 'Kernel size dimension', Stride = 'Stride dimension' and Padding = 'Padding dimension'.
- Batch Normalization with 'Output Channel dimension' channels
- ReLU activation
- Convolution layer with input channels as 'Input Channel dimension', output channels as 'Output Channel dimension', Kernel size = 'Kernel size dimension', Stride = 1 and Padding = 'Padding dimension'.
- Batch Normalization with 'Output Channel dimension' channels
- Down-sampling the residual image through a 2D Convolution layer, if needed
- Adding the residual image to the new image which is obtained from passing the image from the basic block

Note that in the above architecture, if 'Stride dimension' > 1, say for example 'Stride dimension' $= 2$, then the size of the image will decrease by two (in this case). Hence to balance the mismatch in size of the image in basic block, I introduce an artificial convolution layer with stride dimension = 2 (in this case) in the code to reduce the size of the image before adding the residual image to the new image from the basic block. The code to achieve the same is in Figure 1.



```
        if (stride_dim>1):
            self.match_needed = True
        self.match_dim = nn.Conv2d(input_channels, output_channels, kernel_size = 1, stride = stride_dim)
        #To balance the downsample in the size of the image while conv_layer with stride = 2
```

Figure 1: Down sampling the image size depending on stride dimension in basic block

- Residual image is passed through a 2D Convolution layer with input channels as ‘Input Channel dimension’, output channels as ‘Output Channel dimension’, Kernel size = 1, Stride = ‘Stride dimension’ and Padding = ‘Padding dimension’, before adding it to the new image as an output from the basic block.

Residual Network Architecture

I implemented a class *RESNETModel* to define the architecture. The input image is $3 \times 32 \times 32$. My architecture has a convolution layer, 12 basic blocks and a fully connected layers with dropout, batch normalization and Max pooling. The detailed architecture is as follows:

- 2D Convolution layer with input as 3 channels and output as 32 channels, with Kernel $= 3 \times 3$, Stride = 1 and Padding = 1.
- ReLU activation
- 2D Batch Normalization with 32 channels
- Dropout with probability p = 0.3
- Basic Block with input as 32 channels and output as 32 channels, with Kernel $= 3 \times 3$, Stride = 1 and Padding = 1.
- Basic Block with input as 32 channels and output as 32 channels, with Kernel $= 3 \times 3$, Stride = 1 and Padding = 1.
- 2D Convolution layer with input as 32 channels and output as 64 channels, with Kernel $= 1 \times 1$ to up sample the no. of channels of the image.
- Basic Block with input as 64 channels and output as 64 channels, with Kernel $= 3 \times 3$, Stride = 2 and Padding = 1.
- Basic Block with input as 64 channels and output as 64 channels, with Kernel $= 3 \times 3$, Stride = 1 and Padding = 1.
- Basic Block with input as 64 channels and output as 64 channels, with Kernel $= 3 \times 3$, Stride = 1 and Padding = 1.

- Basic Block with input as 64 channels and output as 64 channels, with Kernel = 3×3, Stride = 1 and Padding = 1.
- 2D Convolution layer with input as 64 channels and output as 128 channels, with Kernel = 1×1 to up sample the no. of channels of the image.
- Basic Block with input as 128 channels and output as 128 channels, with Kernel = 3×3, Stride = 2 and Padding = 1.
- Basic Block with input as 128 channels and output as 128 channels, with Kernel = 3×3, Stride = 1 and Padding = 1.
- Basic Block with input as 128 channels and output as 128 channels, with Kernel = 3×3, Stride = 1 and Padding = 1.
- Basic Block with input as 128 channels and output as 128 channels, with Kernel = 3×3, Stride = 1 and Padding = 1.
- 2D Convolution layer with input as 128 channels and output as 256 channels, with Kernel = 1×1 to up sample the no. of channels of the image.
- Basic Block with input as 256 channels and output as 256 channels, with Kernel = 3×3, Stride = 2 and Padding = 1.
- Basic Block with input as 256 channels and output as 256 channels, with Kernel = 3×3, Stride = 1 and Padding = 1.
- Max pooling with stride = 4 and kernel size = 4
- Flattening the convolution output
- Fully connected layer with 256 input and 100 output features.

Note: No. of output units in a convolution layer = $\frac{\text{No. of input units - Filter Size + 2(Padding)}}{Stride} + 1$
Next, we calculate the predicted label as argmax$(f(x, \theta))$ for the training sample which is compared with true label y to compute the accuracy of the training set.

- Notice that since the output of the first basic block contains 32 channels, whereas the input for the next basic block should have 64 channel, so an artificial 2D Convolution layer is added between the two basic blocks with input channels as output of first basic block and output channels as the input for next basic block, and Kernel = 1×1 to up sample the no. of channels of the image.

- For implementation in Python, I used the ReLU activation function as $\sigma(z) = max(0, z)$.

- For implementation, I trained the Resnet for 50 epochs in Part (1) and 5 epochs in Part (2) to achieve the target accuracy of 60% and 70% resp.
- I used the batch size = 100.

Hence, we have an artificial 2D convolution layer to up sample the number of channels and another 2D convolution layer in the basic block to downsample the size of the image.

Data Augmentation and Optimizer

I implemented the following data augmentation techniques in the training set while loading:

- Randomly flip the image horizontally with a probability of 0.2
- Randomly flip the image vertically with a probability of 0.2
- Normalization (Done both while training and testing as well)

I used the ADAM optimizer, implemented by *torch.optim.Adam(model.parameters(), lr=LR)* to minimize the cross entropy loss function, implemented by *nn.CrossEntropyLoss()*.

- For implementation, I used ADAM optimizer to minimize the cross entropy loss function with learning rate $\alpha = 0.001$ and $\alpha = 0.0001$ after 30 epochs.

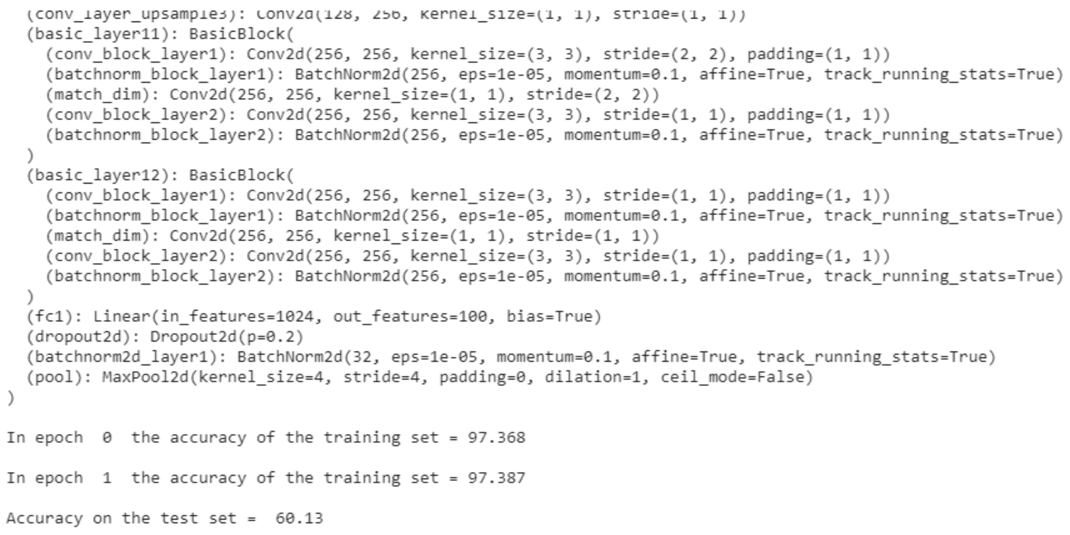
A snapshot of the training and test accuracy's for self trained Residual network is given in Figure 2.
Note that the accuracy on the training set after 50 epochs reaches approx. 0.973, but clearly since the test accuracy is 0.613, we can notice that the model **overfits** the training set.

Part B: Fine tuning a pretrained Resnet Model for CIFAR100

I used the pretrained Residual Network on ImageNet data-set to load the initial weights for the network. The following piece of code in Figure 3 loads the pretrained Resnet18 model in Pytorch:

Also, I fine-tune the model by changing it's last fully connected layer and setting the number of outputs as 100, as CIFAR100 has 100 classes.
Also, since the model is pretraiend on ImageNet, with image size = 224×224 and CIFAR100 images are 32×32, we re-size the train and test images to 224×224.

- Change the number of outputs in the last layer of the pretrained network to 100.



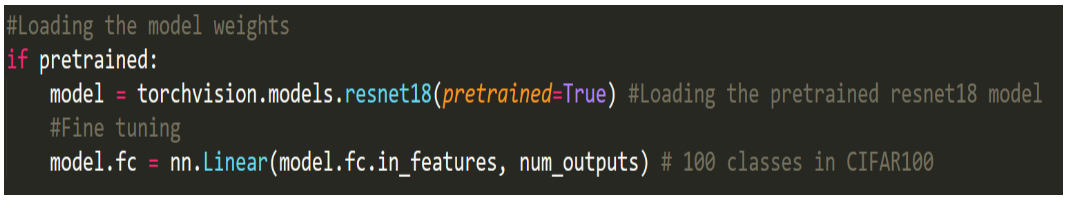
```
  (conv_layer_upsample3): Conv2d(128, 256, kernel_size=(1, 1), stride=(1, 1))
  (basic_layer11): BasicBlock(
    (conv_block_layer1): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (batchnorm_block_layer1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (match_dim): Conv2d(256, 256, kernel_size=(1, 1), stride=(2, 2))
    (conv_block_layer2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (batchnorm_block_layer2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (basic_layer12): BasicBlock(
    (conv_block_layer1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (batchnorm_block_layer1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (match_dim): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
    (conv_block_layer2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (batchnorm_block_layer2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (fc1): Linear(in_features=1024, out_features=100, bias=True)
  (dropout2d): Dropout2d(p=0.2)
  (batchnorm2d_layer1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
)

In epoch  0  the accuracy of the training set = 97.368

In epoch  1  the accuracy of the training set = 97.387

Accuracy on the test set =  60.13
```

Figure 2: Results for the RESNET architecture mentioned in HW notes (Part 1)



```python
#Loading the model weights
if pretrained:
    model = torchvision.models.resnet18(pretrained=True) #Loading the pretrained resnet18 model
    #Fine tuning
    model.fc = nn.Linear(model.fc.in_features, num_outputs) # 100 classes in CIFAR100
```

Figure 3: Code for loading pretrained model

- To resize the images in the training and testing set, *transforms.Resize(size=(224, 224))* function is used in the data loading step for both training and testing images loader.
- In this case, I use the Stochastic Gradient Descent optimizer with learning rate 0.01.

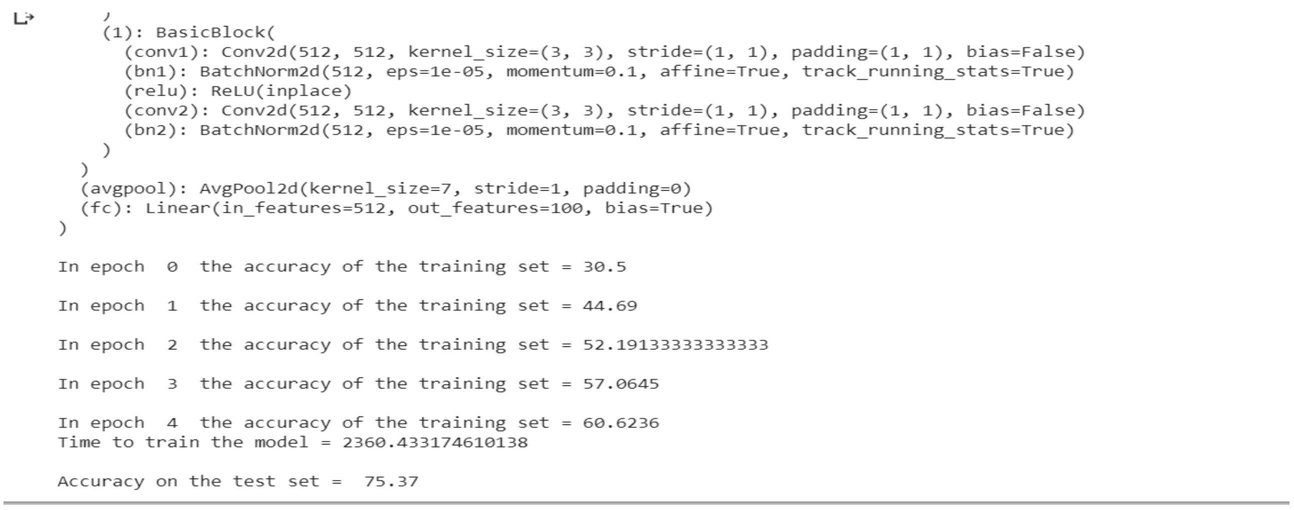
Once the pretrained model is loaded, I train it on the CIFAR100 dataset for 5 epochs. Figure 4 outlines the results for the same.

Results

Part 1: The results for self trained Residual Network are as follows:

- The accuracy on the test set is = 60.13 %
- The estimated time to train the model for 50 epochs is about 3.5 hours.

Part 2: The results for pretrained Residual Network are as follows:



```
      )
      (1): BasicBlock(
        (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (avgpool): AvgPool2d(kernel_size=7, stride=1, padding=0)
    (fc): Linear(in_features=512, out_features=100, bias=True)
  )

  In epoch  0  the accuracy of the training set = 30.5

  In epoch  1  the accuracy of the training set = 44.69

  In epoch  2  the accuracy of the training set = 52.19133333333333

  In epoch  3  the accuracy of the training set = 57.0645

  In epoch  4  the accuracy of the training set = 60.6236
  Time to train the model = 2360.433174610138

  Accuracy on the test set =  75.37
```

Figure 4: Results for the Pretrained RESNET model (Part 2)

- The accuracy on the test set is = 75.37 %
- The estimated time to train the model for 5 epochs is about 1.5 hours.

I predict the label based on each element in test set and check if the true label is same as the predicted label to calculate the accuracy.