Problem Set 4

Due: December 6, 2023

Programming Exercise: In this exercise, you will implement a ResNet structure and will use it for MNIST database classification. You can use the given PyTorch tutorial for training and replace the model with ResNet by using the following ResNet9.

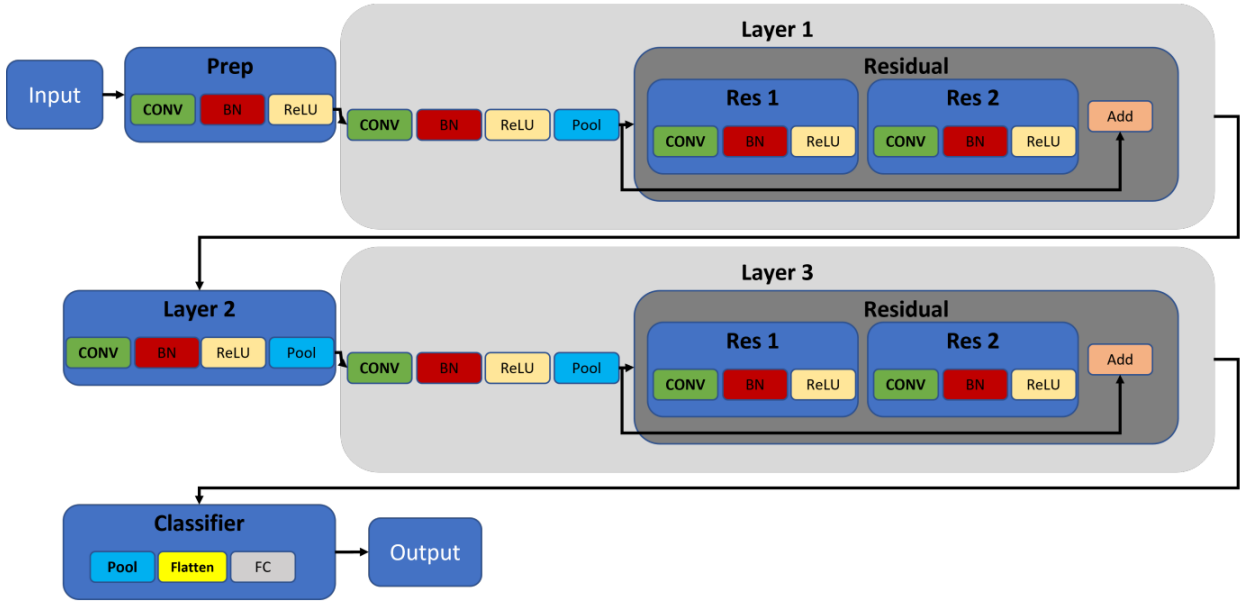

Figure 1: ResNet9 Architecture

1) [3 pts] To implement this architecture, you need to start with a convolutional block consist of a 2D convolution (kernel size = 3×3) followed by a batch normalization and a ReLU activation. Note that, the first preparation block (and layers inside the Residual connections) does not have a pooling that needs to be taken into account while handling the function. Following code snippet would be helpful to define this convolution block.

```
def conv_block(in_channels, out_channels, pool=False,poll_size=2):
layers = [
#define 2D Convolutions
#define batch normalization
#define activation
]
if pool: #add the pooling
return nn.Sequential(*layers)
```

2) [6 pts] In this part, you will implement the ResNet9 class using the convolutional block from the previous step.

```
class ResNet9(nn.Module):
   # when you call ResNet9, you need to give # of input channels and # of
      classes
   # by using ResNet9(inp_c=,out_c=)
   def __init__(self, in_channels, num_classes):
      super().__init__()

      #(1)define 1st convolutional layer using conv_block(), the ouput channel
         number should be 32
      #(2)define 2nd convolutional layer using conv_block(), input channels
         will be 32, the ouput channels will be 64 with a pooling of 2

      #(3)define 1st residual connection using two consequitive convolutional
         blocks such as nn.Sequential(conv_block(64, 64), conv_block(64, 64))

      #repeat steps 1-3 but with twice the input-output channel sizes
      #such as 3rd convolutional layer's input will be 64 and output will be
         256
      #this layer should end up with 256 channels

      #define the classifier layer using max pooling followed by flattening
         and a #fully connected layer. Output channel number should be the
         num_classes


   def forward(self, xb):

      #connect all the convolutional and residual layers
      #Note: do not forget to add residual connections

      return out
```

3) [6 pts] Using the ResNet9, train the model over 10 epochs and report the accuracy on the test database along with example images and the predicted labels.

Note : Similar to PyTorch, if you choose to use TensorFlow, following two snippets are examples for convolutional blocks and ResNet structure:

```
def conv_layer(x, Ws, is_training, is_pool):
#x is the input and Ws are in the format of [kernel size,kernel size,#input
   #channel,#output channel]
#first you need to initialize the W by using
W = tf.get_variable('W', shape=Ws,initializer=tf.random_normal_initializer(0,
   0.05))
#define the 2D convolution. Note: Do not forget the strides and padding
   information
```

```
#define bacth normalization
#define activation

if (is_pool):
#define pooling
return x
```

```
def ResNet(inp, is_training):

    nw = {} #initialization for the layers
    Ws = {} #initialization for the kernels and hannels sizes

    #define the [kernel size,kernel size,#input channel,#output channel] of all
        layers
    #such as Ws[1] = (3,3,1,32) and so on

    #(1)define the first layer without the pooling
    with tf.variable_scope('FirstLayer'):
       #nw['c1'] = conv_layer(...)

    #(2)define the 2nd voncolutional later using the conv_layer but with pooling
    #(3) define the residual connection using two consequtive convolutional
        blocks
       #with tf.variable_scope('ResBlock1'):
          #conv_layer1 = conv_layer(...)
          #conv_layer2 = conv_layer(conv_layer1,...)
          #nw['c2'] = conv_layer2 + ...

    #repeat steps 1-3 with the proper #of input-output channels. The final
        output channel size should be 256

    #define the classifier layer using pooling followed by flattenning and a
        fuclly connected layer

    #Note: Do not forget to add residual at the end

    return out
```