COMPSCI 574: Intelligent Visual Computing Spring 23

Prachi Jain

Assignment 2: Multi-View CNN for Shape Classification

1 Task A [70%]

Change starter code in "model.py" to define specified convnet:

1.1 Implementation

Please find attached code in *model.py* for the design of convnet.

```
import torch
import torch.nn as nn

#Utility function for calculating output size of previous convolution layer
    which can be used as a parameter
#for normalized_shape argument in nn.LayerNorm - "think about what the parameter
     normalized_shape should be"
import numpy as np
def conv2d_output_size(input_size, out_channels, padding, kernel_size, stride,
    dilation=None):
    """According to https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.
    html
    """
    if dilation is None:
        dilation = (1, ) * 2
    if isinstance(padding, int):
        padding = (padding, ) * 2
    if isinstance(kernel_size, int):
        kernel_size = (kernel_size, ) * 2
    if isinstance(stride, int):
        stride = (stride, ) * 2

    output_size = (
        out_channels,
        np.floor((input_size[0] + 2 * padding[0] - dilation[0] * (kernel_size[0]
     - 1) - 1) /
                 stride[0] + 1).astype(int),
```

```
        np.floor((input_size[1] + 2 * padding[1] - dilation[1] * (kernel_size[1] - 1) - 1) /
                 stride[1] + 1).astype(int)
    )
    return output_size

class CNN(nn.Module):

    def __init__(self, num_classes):
        super(CNN, self).__init__()
        """

        DEFINE YOUR NETWORK HERE

        """

        self.num_classes = num_classes

        # Convolution layers self.conv1, self.depthconv1, self.depthconv2 followed by norm layers and leaky relu
        # would not have biases since normalisation layer will re-center the data anyway, removing the bias and
        # making it a useless trainable parameter.

        #1 Convolution layer with 8 filters applying 7x7 filters on input image. Set stride to 1 and padding to '0'.
        self.conv1  = nn.Conv2d(in_channels = 1, out_channels = 8, kernel_size = 7, stride = 1, padding = 0, bias = False)
        #H,W of Input: 112, 112
        outputSize1 = conv2d_output_size([112, 112], out_channels = 8, kernel_size = 7, stride = 1, padding = 0)

        #2. Normalization layer
        self.norm1 = nn.LayerNorm(normalized_shape = [*outputSize1]) # <--- Normalize activations over C, H, and W shape of the Convolution Output

        #3, 7, 12 Leaky ReLu layer with 0.01 'leak' for negative input
        self.leakyReLU = nn.LeakyReLU(0.1)

        #4, 8, 13 Max pooling layer operating on 2x2 windows with stride 2
        self.MaxPool  = nn.MaxPool2d(kernel_size = 2, stride = 2)
        outputSize2 = conv2d_output_size(outputSize1[1:], out_channels = 8, kernel_size = 2, stride = 2, padding = 0)

        #5. Depthwise convolution layer with 8 filters applying 7x7 filters on the feature map of the previous layer.
        #Set stride to 2 and padding to 0. Groups should be 8 to enable depthwise convolution.
        self.depthconv1 = nn.Conv2d(in_channels = 8, out_channels = 8, kernel_size = 7, stride = 2, padding = 0, groups = 8, bias = False)
        outputSize3 = conv2d_output_size(outputSize2[1:], out_channels = 8, kernel_size = 7, stride = 2, padding = 0)

        #Max Pool outputsize calculation
        outputSize4 = conv2d_output_size(outputSize3[1:], out_channels = 8, kernel_size = 2, stride = 2, padding = 0)

        #6. Normalization layer
```

```
        self.norm2 = nn.LayerNorm(normalized_shape = [*outputSize3])

        #9. Pointwise convolution layer with 16 filters applying 1x1 filters on the feature map of the previous layer
        #(stride is 1, and no padding).
        self.pointconv1 = nn.Conv2d(in_channels = 8, out_channels = 16, kernel_size = 1, stride = 1, padding = 0, bias = True)
        outputSize5 = conv2d_output_size(outputSize4[1:], out_channels = 16, kernel_size = 1, stride = 1, padding = 0)

        #10. Depthwise convolution layer with 16 filters applying 7x7 filters on the feature map of the previous layer.
        #Set stride to 1 and padding to 0. Groups should be 16 to enable depthwise convolution
        self.depthconv2 = nn.Conv2d(in_channels = 16, out_channels = 16, kernel_size = 7, stride = 1, padding = 0, groups = 16, bias = False)
        outputSize6 = conv2d_output_size(outputSize5[1:], out_channels = 16, kernel_size = 7, stride = 1, padding = 0)

        #11. Normalization layer
        self.norm3 = nn.LayerNorm(normalized_shape = [*outputSize6])

        #14. Pointwise convolution layer with 32 filters applying 1x1 filters on the feature map of the previous layer
        #(stride is 1, and no padding).
        self.pointconv2 = nn.Conv2d(in_channels = 16, out_channels = 32, kernel_size = 1, stride = 1, padding = 0, bias = True)

        #15. Fully connected layer with K outputs, where K is the number of categories. Include the bias.
        #Implement the fully connected layer as a convolutional one (think how and why).
        self.fullconnect = nn.Conv2d(in_channels = 32, out_channels = num_classes, kernel_size = 3, bias = True)

        self.apply(self._init_weights)

    def _init_weights(self, module):
        """ Initialize the weights """
        if isinstance(module, nn.Conv2d):
            # Initialize weights according to the Kaiming Xe's uniform distribution scheme
            # for conv layers followed by leaky ReLU activations
            nn.init.kaiming_uniform_(self.conv1.weight)
            nn.init.kaiming_uniform_(self.depthconv1.weight)
            nn.init.kaiming_uniform_(self.depthconv2.weight)

            # Initialize weights according to the Xavier uniform distribution scheme for pointwise convolution layers
            # and fully connected layer (i.e., layers not followed by ReLUs).
            nn.init.xavier_uniform(self.pointconv1.weight)
            nn.init.xavier_uniform(self.pointconv2.weight)
            nn.init.xavier_uniform(self.fullconnect.weight)

            # Initialize any biases to 0.
            if module.bias is not None:
                module.bias.data.zero_()

```

```python
    def forward(self, x):
        """

        DEFINE YOUR FORWARD PASS HERE

        """
        h1 = self.conv1(x)
        h2 = self.norm1(h1)
        h3 = self.leakyReLU(h2)
        h4 = self.MaxPool(h3)

        #Combination of depthwise and pointwise convolution produces a block called "Depthwise Separable Convolution"
        h5 = self.depthconv1(h4)
        h6 = self.norm2(h5)
        h7 = self.leakyReLU(h6)
        h8 = self.MaxPool(h7)
        h9 = self.pointconv1(h8)

        #Combination of depthwise and pointwise convolution produces a block called "Depthwise Separable Convolution"
        h10 = self.depthconv2(h9)
        h11 = self.norm3(h10)
        h12 = self.leakyReLU(h11)
        h13 = self.MaxPool(h12)
        h14 = self.pointconv2(h13)

        out = self.fullconnect(h14)
        return out
```

2 Task B [30%]

Modify the function of "testMVImageClassifier.py" such that the function outputs category predictions per shape and test error averaged over all the test shapes with two strategies:

- Mean view-pooling
- Mean view-pooling

2.1 Implementation

Please find attached code of *testMVImageClassifier.py*.

```python
import os
import numpy as np
import torch
from torch.autograd import Variable
from data_utils import grayscale_img_load, listdir

```

```python
def testMVImageClassifier(dataset_path, model, info, pooling = 'mean', cuda=False, verbose=False):

    # save pytorch model to eval mode
    model.eval()
    if (cuda):
        model.cuda()

    test_err = 0
    count = 0
    print("=>Testing...")

    # for each category
    for idx, c in enumerate(info['category_names']):
        category_full_dir = os.path.join(dataset_path,c)
        shape_dirs        = listdir(category_full_dir)
        print('=>Loading shape data: %s'%(c))

        # for each shape
        for s in shape_dirs:
            if verbose: print('=>Loading shape data: %s %s'%(s, c))
            views = listdir(os.path.join(category_full_dir, s))
            scores = np.zeros((len(views),len(info['category_names'])))
            count  += 1

            # for each view
            for i, v in enumerate(views):
                image_full_filename = os.path.join(category_full_dir, s, v)
                if 'png' not in image_full_filename : continue
                if verbose: print(' => Loading image: %s ...'% image_full_filename)
                im  = grayscale_img_load(image_full_filename)/255.
                im -= info['data_mean']
                im  = Variable(torch.from_numpy(im.astype('float32')), requires_grad=False).unsqueeze(0)
                # get predicted scores for each view
                if (cuda):
                    im = im.cuda()
                    scores[i, :] = model(im).detach().cpu().numpy().squeeze()
                else:
                    scores[i, :] = model(im).detach().numpy().squeeze()

            '''
            YOUR CODE GOES HERE
            1) Get category predictions per shape and test error averaged over all the test shapes.
            2) Implement 2 strategies: 1) mean and 2) max view-pooling by specifying input arg 'pooling', like
               >> pooling = 'mean' or pooling = 'max'

             '''
            #predicted_label = 0 # obviously change this

            # Mean view-pooling
            if pooling == 'mean':
                predicted_label = np.argmax(np.mean(scores, axis = 0))
            # Max view-pooling
            elif pooling == 'max':
```

```
                predicted_label = np.argmax(np.max(scores, axis = 0))

            if predicted_label != idx:
                test_err += 1

            if verbose: print('predicted label:  %s, ground-truth label: %s\n'%(
info['category_names'][predicted_label] ,c))

    test_err = test_err / count
    print('Test error: %f%%\n'%(test_err * 100))
```

3 Performance Evaluation

- Test Error for Mean View Pooling = **14.000000%**
- Test Error for Max View Pooling = **17.000000%**

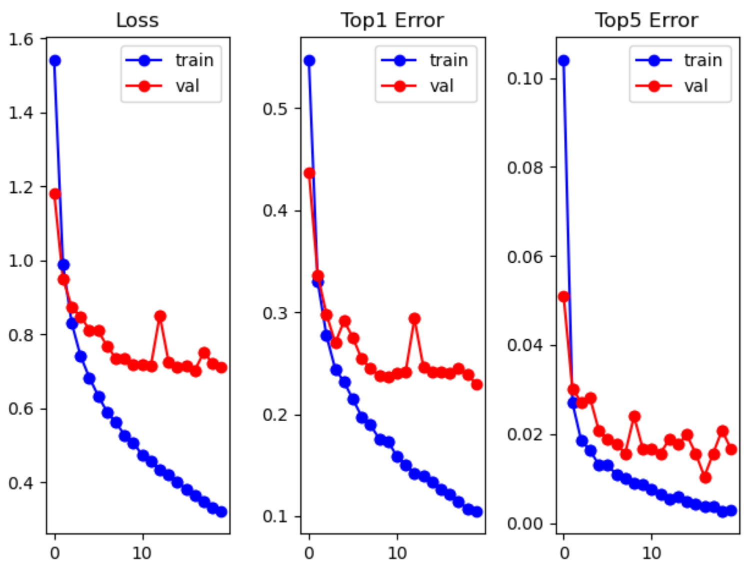

Figure 1: Plot showing a). Epoch vs Training and Validation loss, b) 2) Epoch vs Training and Validation Top1 Error and c). 2) Epoch vs Training and Validation Top1 Error