GoogleNet

December 15, 2022

[ ]:
```
import torch
import torch.nn as nn
from torch.utils.tensorboard import SummaryWriter

class ConvBlock(nn.Module):
    def __init__(self, in_fts, out_fts, k, s, p):
        super(ConvBlock, self).__init__()
        self.convolution = nn.Sequential(
            nn.Conv2d(in_channels=in_fts, out_channels=out_fts, kernel_size=(k,␣
 ↪k), stride=(s, s), padding=(p, p)),
            nn.ReLU()
        )

    def forward(self, input_img):
        x = self.convolution(input_img)

        return x

class ReduceConvBlock(nn.Module):
    def __init__(self, in_fts, out_fts_1, out_fts_2, k, p):
        super(ReduceConvBlock, self).__init__()
        self.redConv = nn.Sequential(
            nn.Conv2d(in_channels=in_fts, out_channels=out_fts_1,␣
 ↪kernel_size=(1, 1), stride=(1, 1)),
            nn.ReLU(),
            nn.Conv2d(in_channels=out_fts_1, out_channels=out_fts_2,␣
 ↪kernel_size=(k, k), stride=(1, 1), padding=(p, p)),
            nn.ReLU()
        )

    def forward(self, input_img):
        x = self.redConv(input_img)

        return x

class AuxClassifier(nn.Module):
    def __init__(self, in_fts, num_classes):
        super(AuxClassifier, self).__init__()
        self.avgpool = nn.AvgPool2d(kernel_size=(5, 5), stride=(3, 3))
        self.conv = nn.Conv2d(in_channels=in_fts, out_channels=128,␣
 ↪kernel_size=(1, 1), stride=(1, 1))
        self.relu = nn.ReLU()
        self.fc = nn.Linear(4 * 4 * 128, 1024)
        self.dropout = nn.Dropout(p=0.7)
        self.classifier = nn.Linear(1024, num_classes)

    def forward(self, input_img):
        N = input_img.shape[0]
        x = self.avgpool(input_img)
        x = self.conv(x)
        x = self.relu(x)
        x = x.reshape(N, -1)
        x = self.fc(x)
        x = self.dropout(x)
        x = self.classifier(x)

        return x


class InceptionModule(nn.Module):
    def __init__(self, curr_in_fts, f_1x1, f_3x3_r, f_3x3, f_5x5_r, f_5x5,␣
 ↪f_pool_proj):
        super(InceptionModule, self).__init__()
        self.conv1 = ConvBlock(curr_in_fts, f_1x1, 1, 1, 0)
        self.conv2 = ReduceConvBlock(curr_in_fts, f_3x3_r, f_3x3, 3, 1)
        self.conv3 = ReduceConvBlock(curr_in_fts, f_5x5_r, f_5x5, 5, 2)

        self.pool_proj = nn.Sequential(
            nn.MaxPool2d(kernel_size=(1, 1), stride=(1, 1)),
            nn.Conv2d(in_channels=curr_in_fts, out_channels=f_pool_proj,␣
 ↪kernel_size=(1, 1), stride=(1, 1)),
            nn.ReLU()
        )

    def forward(self, input_img):
        out1 = self.conv1(input_img)
        out2 = self.conv2(input_img)
        out3 = self.conv3(input_img)
        out4 = self.pool_proj(input_img)

        x = torch.cat([out1, out2, out3, out4], dim=1)

        return x
```

```
class MyGoogleNet(nn.Module):
    def __init__(self, in_fts=3, num_class=1000):
        super(MyGoogleNet, self).__init__()
        self.conv1 = ConvBlock(in_fts, 64, 7, 2, 3)
        self.maxpool1 = nn.MaxPool2d(kernel_size=(3, 3), stride=(2, 2),␣
 ↪padding=(1, 1))
        self.conv2 = nn.Sequential(
            ConvBlock(64, 64, 1, 1, 0),
            ConvBlock(64, 192, 3, 1, 1)
        )

        self.inception_3a = InceptionModule(192, 64, 96, 128, 16, 32, 32)
        self.inception_3b = InceptionModule(256, 128, 128, 192, 32, 96, 64)
        self.inception_4a = InceptionModule(480, 192, 96, 208, 16, 48, 64)
        self.inception_4b = InceptionModule(512, 160, 112, 224, 24, 64, 64)
        self.inception_4c = InceptionModule(512, 128, 128, 256, 24, 64, 64)
        self.inception_4d = InceptionModule(512, 112, 144, 288, 32, 64, 64)
        self.inception_4e = InceptionModule(528, 256, 160, 320, 32, 128, 128)
        self.inception_5a = InceptionModule(832, 256, 160, 320, 32, 128, 128)
        self.inception_5b = InceptionModule(832, 384, 192, 384, 48, 128, 128)

        self.aux_classifier1 = AuxClassifier(512, num_class)
        self.aux_classifier2 = AuxClassifier(528, num_class)
        self.avgpool = nn.AdaptiveAvgPool2d(output_size=(7, 7))
        self.classifier = nn.Sequential(
            nn.Dropout(p=0.4),
            nn.Linear(1024 * 7 * 7, num_class)
        )

    def forward(self, input_img):
        N = input_img.shape[0]
        x = self.conv1(input_img)
        x = self.maxpool1(x)
        x = self.conv2(x)
        x = self.maxpool1(x)
        x = self.inception_3a(x)
        x = self.inception_3b(x)
        x = self.maxpool1(x)
        x = self.inception_4a(x)
        out1 = self.aux_classifier1(x)
        x = self.inception_4b(x)
        x = self.inception_4c(x)
        x = self.inception_4d(x)
        out2 = self.aux_classifier2(x)
        x = self.inception_4e(x)
        x = self.maxpool1(x)
        x = self.inception_5a(x)
        x = self.inception_5b(x)
        x = self.avgpool(x)
        x = x.reshape(N, -1)
        x = self.classifier(x)
        if self.training == True:
            return [x, out1, out2]
        else:
            return x


if __name__ == '__main__':
    # Temporary define data and target
    batch_size = 5
    x = torch.randn((batch_size, 3, 224, 224))
    y = torch.randint(0,1000, (batch_size,))
    num_classes = 1000

    # Add to graph in tensorboard
    writer = SummaryWriter(log_dir='logs/googlenet')
    m = MyGoogleNet()
    # print(m)
    # we have x,o1,o2 = m(x)
    # m(x)[0] means x; m(x)[1] means o1; m(x)[2] means o2
    # o1 and o2 are output from auxclassifier
    print(m(x)[0].shape)
    m.eval()
    print(m.training)
    writer.add_graph(m, x)
    writer.close()

    # Notice here! When you going to train your network
    # Put these loss value into train step of your model
    m.train()
    loss = nn.CrossEntropyLoss()
    loss1 = nn.CrossEntropyLoss()
    loss2 = nn.CrossEntropyLoss()
    discount = 0.3

    o,o1,o2 = m(x)

    total_loss = loss(o,y) + discount*(loss1(o1,y) + loss2(o2,y))
    print(total_loss)

    # And while inferencing the model, set the model into
    # model.eval() mode
    m.eval()
```