Programming Assignment 2 - CAINE, Wilbert (20584260)

Q1

----------------------------------------------------------------

Layer (type) Output Shape Param #

================================================================

Conv2d-1 [-1, 64, 64, 64] 9,408

BatchNorm2d-2 [-1, 64, 64, 64] 128

ReLU-3 [-1, 64, 64, 64] 0

MaxPool2d-4 [-1, 64, 32, 32] 0

Conv2d-5 [-1, 64, 32, 32] 36,864

BatchNorm2d-6 [-1, 64, 32, 32] 128

ReLU-7 [-1, 64, 32, 32] 0

Conv2d-8 [-1, 64, 32, 32] 36,864

BatchNorm2d-9 [-1, 64, 32, 32] 128

Identity-10 [-1, 64, 32, 32] 0

ReLU-11 [-1, 64, 32, 32] 0

ResBlock-12 [-1, 64, 32, 32] 0

Conv2d-13 [-1, 64, 32, 32] 36,864

BatchNorm2d-14 [-1, 64, 32, 32] 128

ReLU-15 [-1, 64, 32, 32] 0

Conv2d-16 [-1, 64, 32, 32] 36,864

BatchNorm2d-17 [-1, 64, 32, 32] 128

Identity-18 [-1, 64, 32, 32] 0

ReLU-19 [-1, 64, 32, 32] 0

ResBlock-20 [-1, 64, 32, 32] 0

Conv2d-21 [-1, 128, 16, 16] 73,728

BatchNorm2d-22 [-1, 128, 16, 16] 256

ReLU-23 [-1, 128, 16, 16] 0

Conv2d-24 [-1, 128, 16, 16] 147,456

BatchNorm2d-25 [-1, 128, 16, 16] 256

Conv2d-26 [-1, 128, 16, 16] 8,192

BatchNorm2d-27 [-1, 128, 16, 16] 256

ReLU-28 [-1, 128, 16, 16] 0

ResBlock-29 [-1, 128, 16, 16] 0

Conv2d-30 [-1, 128, 16, 16] 147,456

BatchNorm2d-31 [-1, 128, 16, 16] 256

ReLU-32 [-1, 128, 16, 16] 0

Conv2d-33 [-1, 128, 16, 16] 147,456

BatchNorm2d-34 [-1, 128, 16, 16] 256

Identity-35 [-1, 128, 16, 16] 0

ReLU-36 [-1, 128, 16, 16] 0

ResBlock-37 [-1, 128, 16, 16] 0

Conv2d-38 [-1, 256, 8, 8] 294,912

BatchNorm2d-39 [-1, 256, 8, 8] 512

ReLU-40 [-1, 256, 8, 8] 0

Conv2d-41 [-1, 256, 8, 8] 589,824

BatchNorm2d-42 [-1, 256, 8, 8] 512

Conv2d-43 [-1, 256, 8, 8] 32,768

BatchNorm2d-44 [-1, 256, 8, 8] 512

ReLU-45 [-1, 256, 8, 8] 0

ResBlock-46 [-1, 256, 8, 8] 0

Conv2d-47 [-1, 256, 8, 8] 589,824

BatchNorm2d-48 [-1, 256, 8, 8] 512

ReLU-49 [-1, 256, 8, 8] 0

Conv2d-50 [-1, 256, 8, 8] 589,824

BatchNorm2d-51 [-1, 256, 8, 8] 512

Identity-52 [-1, 256, 8, 8] 0

ReLU-53 [-1, 256, 8, 8] 0

ResBlock-54 [-1, 256, 8, 8] 0

Conv2d-55 [-1, 512, 8, 8] 1,179,648

BatchNorm2d-56 [-1, 512, 8, 8] 1,024

ReLU-57 [-1, 512, 8, 8] 0

Conv2d-58 [-1, 512, 8, 8] 2,359,296

BatchNorm2d-59 [-1, 512, 8, 8] 1,024

Conv2d-60 [-1, 512, 8, 8] 131,072

BatchNorm2d-61 [-1, 512, 8, 8] 1,024

ReLU-62 [-1, 512, 8, 8] 0

ResBlock-63 [-1, 512, 8, 8] 0

Conv2d-64 [-1, 512, 8, 8] 2,359,296

BatchNorm2d-65 [-1, 512, 8, 8] 1,024

ReLU-66 [-1, 512, 8, 8] 0

Conv2d-67 [-1, 512, 8, 8] 2,359,296

BatchNorm2d-68 [-1, 512, 8, 8] 1,024

Identity-69 [-1, 512, 8, 8] 0

ReLU-70 [-1, 512, 8, 8] 0

ResBlock-71 [-1, 512, 8, 8] 0

AvgPool2d-72 [-1, 512, 1, 1] 0

Input

Conv+BN+ReLU = 64 x 64 x 64

Max pooling = 64 x 32 x 32

ResBlock = 64 x 32 x 32

ResBlock = 64 x 32 x 32

ResBlock-D = 128 x 16 x 16

ResBlock = 128 x 16 x 16

ResBlock-D = 256 x 8 x 8

ResBlock = 256 x 8 x 8

ResBlock = 512 x 8 x 8

ResBlock = 512 x 8 x 8

Average pooling = 512 x 1 x 1

Output

Q2

Kernel size = 8 x 8

Stride = 1

Q3

Convolution is a possible alternative to the pooling layer. By applying convolutional layer with the certain settings, such as the number of output channels equals to the number of input channels and the stride equals to the kernel size, we will obtain a 1D vector out of the output feature map. In the case of average pooling, we could define a convolutional layer where the weights would compute the average of the receptive field with the same kernel size and stride and without bias. More specifically, the weights are all equal to one another with sum of one. Thus, we obtain that the weights equal 1/kernel\_side\_length^2. This process would mimic the process of average pooling. With this understanding, we could view the average pooling layer as a special case of convolutional layer. One thing to note is that average pooling layer do not have trainable parameter unlike convolutional layer.

Q4

The first ResBlock consists of:

Conv2d-5 [-1, 64, 32, 32] 36,864

BatchNorm2d-6 [-1, 64, 32, 32] 128

ReLU-7 [-1, 64, 32, 32] 0

Conv2d-8 [-1, 64, 32, 32] 36,864

BatchNorm2d-9 [-1, 64, 32, 32] 128

Identity-10 [-1, 64, 32, 32] 0

ReLU-11 [-1, 64, 32, 32] 0

with a total of 73984 trainable parameters.

Q5

None of the layers of the modified ResNet-18 model cannot load the pretrained weights

The following layers of the pretrained weights are discarded: fc

* fc.weight param cannot be found in our model
* fc.bias param cannot be found in our model

Q6

The output dimensionality of the last FC layer depends on the number of candidates, i.e. 230 and 49 for training and validation stage respectively.

Q7

Using dropout layer, we randomly set neurons to zero with the pre-specified probability during the forward pass. So, there is a factor of randomness during the training of the neurons. Another way to interpret the model with dropout in training mode is that we are training a large ensemble of models that share parameters.

Dropout is useful because it prevents co-adaptation of features, and it forces the network to have redundant representation. Thus, we could prevent overfitting the model.

Q8

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generatedChart, line chart

Description automatically generated

In the above plots, we observe that the accuracy increases overall while the training and the training loss per iteration decreases overtime as we train for larger epoch number. The validation loss per epoch is not affected as mentioned in the project guideline. The accuracy order from highest to lowest is R10, R5, and R1 which is expected considering the way they are measured where it is generally more difficult to predict correctly given fewer prediction candidates. The training losses are already considerably close to zero at the later epochs. Further hyperparameter tuning and training epochs could potentially yield better results.

Q9

Chart, scatter chart

Description automatically generated

Q10

There are 230 \* (10 \* 9) \* 229 \* 10 = 47403000 possible triplets in the training dataset

Q11

Chart, line chart

Description automatically generatedChart

Description automatically generatedChart

Description automatically generated

In the above plots, we observe that the accuracy for R1 drops considerably in the beginning before increasing and stabilizes around 0.67, R5 increases significantly in the beginning and remains around 0.83, and R10 overall could not increase as the training process continues further. As in the previous model, the accuracy order from highest to lowest is R10, R5, and R1. The training loss per epoch and the training loss per epoch decreases overtime as we train for larger epoch number. The losses are already considerably close to zero at the later epochs.