CSC413 Programming Assignment Two

ZhenDi Pan 1003241823

2020-02-26

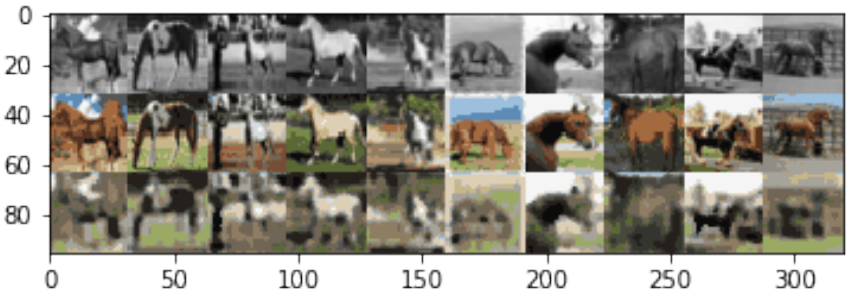
Part One

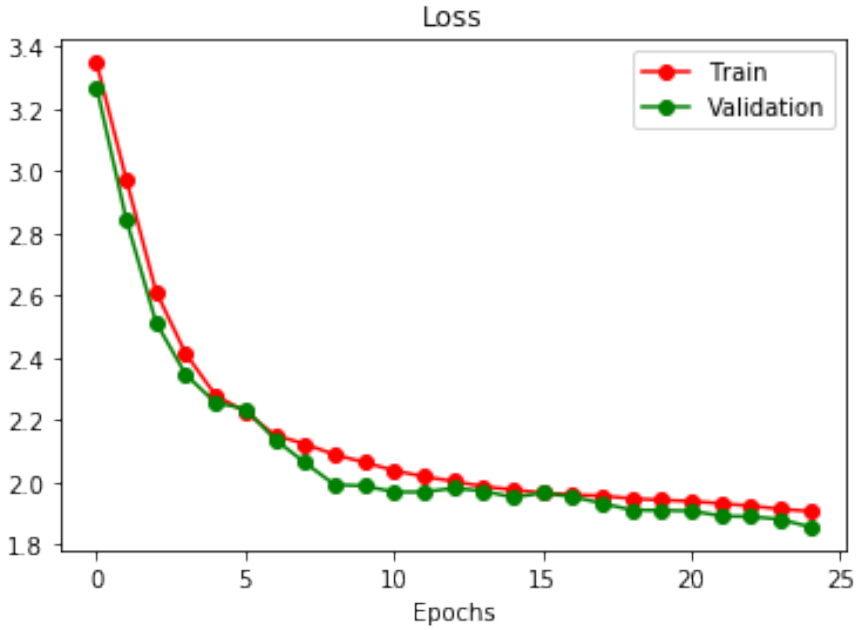
Question 1

In code file.

Question 2

The result is shown below (see code file for specifics):





The results look good in general, however they can definitely be better. Most of the colours are approximately correct compared with the target and the validation error seems OK from the plot. But we can still some colours that are wrong in the results.

Question 3

The number of weights for the first myConv2D-NF: $(2\times2\times+1)NF = 5NF$(added bias). For myConv2D-2NF: $10NF$. For For myConv2D-NC: $5NC$. So the total number of weights as for convolutional layers: $30NF + 10NC$. For batch-normal: $4 \times 5NF = 20NF$, $4 \times 10NF = 40NF$, $4 \times 10NF = 40NF$, $4 \times 5NF = 20NF$, $4 \times 5NF = 20NC$, $4 \times 5NF = 20NC$. So the total number of weights $150NF + 50NC$. Doubling the input dimension does not affect the number of weights.

The number of connections for the first myConv2D-NF: $((32 \times 32 \times 4) + 1) \times NF$. For myConv2D-2NF: $((16 \times 16 \times 4) + 1) \times 2NF$. For the rest of the myConv2D-2NF layers: $((8 \times 8 \times 4) + 1) \times 2NF$, $((8 \times 8 \times 4) + 1) \times NF$, for myConv2D-2NC: $((8 \times 8 \times 4) + 1) \times NC$. So total number of connections $4097NF + 2050NF + 514NF + 257NF + 514NC = 6918NF + 514NC$. Doubling the input dimension would also double the connections: $13836NF + 1028NC$.

The number of outputs: $6NF + 2NC$. Doubling the input dimension does not affect the total number of outputs, it only affectS the size of the outputs.

Question 4

Assuming that the operation does result in overflows, since the operation is linear and convolutional layers are linear as well, this pre-processing step will translate the output of our conv net from question 1 and 2 by the the same amount.

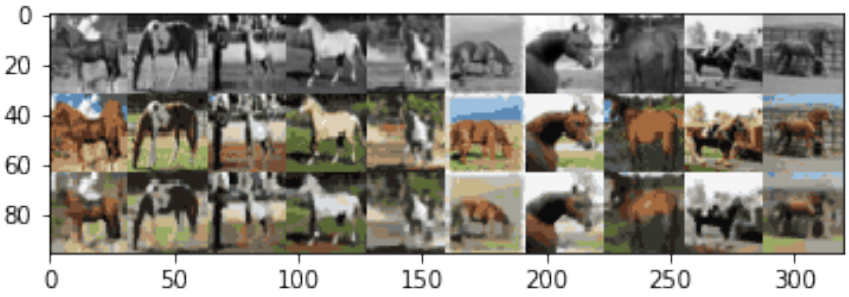
Part Two

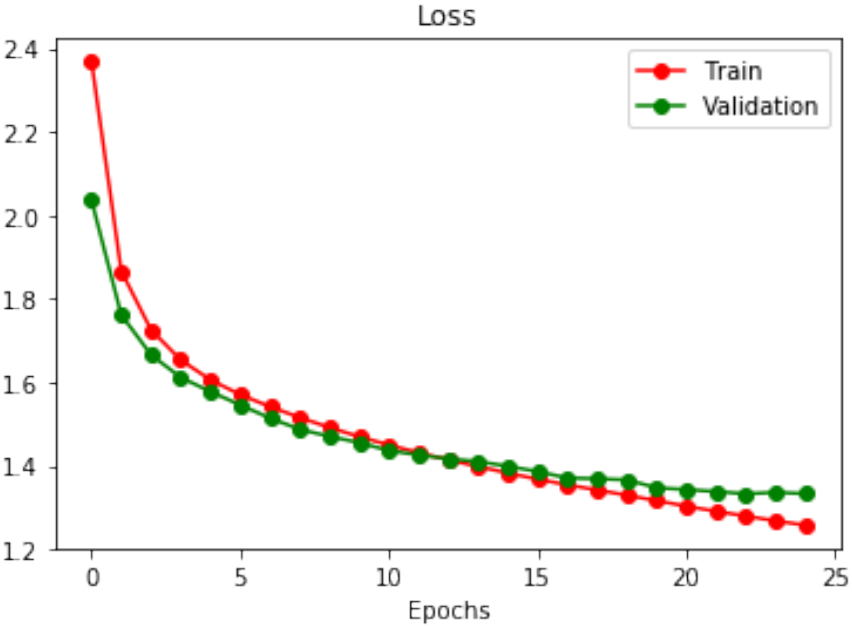
Question 1

In code file.

Question 2

Train the model for 25 epochs and the training curve:





Question 3

The result improved quite a bit compared to the previous model. From the output, we see that the accuracy rate improved from about 32 percent to about 49 percent. The validation error also decreased from around 1.85 to 1.33. In addition, we can see the output images are visibly better. One reason that skip connections improves the result is because during the pooling layers, some of the information passed is reduced, thus adding a skip connection gain back some of the information. Another reason is that passing the first image directly to the last layer helps our model captures the information better, which fills in the colours better and also maintain the original image content.

Question 4

I used a fixed number of 25 for epoch. The different mini batch sizes I used are [10, 50, 100, 200]. The best performance is when the batch size is set to 10. The validation error is the lowest and the accuracy rate is the highest. With larger batch sizes, the performance gets worse, the accuracy rate drops and errors are increased. However, when batch size is set to 10, the training takes a long time to run, almost three times as long as when the batch size is set to 200 (Please see the code file for specific output).

Part Three

Question 1

In code file

Question 2

In code file. The result mIOU: 0.3528.

Question 3

In code file (Too many pictures to include in write-up)

Question 4

If we freeze all layers but the last layer, we only need to back-propagate the last layer. So if we need to fine-tune the entire pre-trained model, the complexity is $\mathcal{O}(n)$, but if we freeze layers, then it becomes $\mathcal{O}(1)$. For computational complexity, we still have to compute the forward pass, even if we freeze all layers but the last layer. Thus the complexity should be $\mathcal{O}(n)$, but we can almost always be sure that the training will be faster if only have to fine-tune the last layer.

Question 5

If we increase the height and the width of the input image by a factor of 2, the number of units in our model increases by the same scale. Thus the memory complexity of fine-tuning will be increased by the same scale as well. The number of parameters is only dependent on filter sizes and feature maps, the size of the image has no affect on it, so the number of parameters remains unchanged.

$$
\begin{aligned}
\bar{y} &= \frac{\partial L}{\partial y} = y - t \\
\bar{W}^{(2)} &= \frac{\partial L}{\partial \bar{W}^{(2)}} = (y-t)\bar{W}^{(2)} \\
\bar{z_2} &= (y-t)\bar{W}^{(2)} \\
\bar{h} &= (y-t)\bar{W}^{(2)} \\
\bar{z_1} &= (y-t)\bar{W}^{(2)}\sigma'(z) \\
\bar{W}^{(1)} &= (y-t)\bar{W}^{(2)}\sigma'(z)x \\
\bar{x} &= \bar{z_1}\frac{\partial z_1}{\partial x_1} + \bar{z_2}\frac{\partial z_2}{\partial x_1} = (y-t)(\bar{W}^{(2)})^2(\bar{W}^{(1)})\sigma'(z)
\end{aligned}
$$