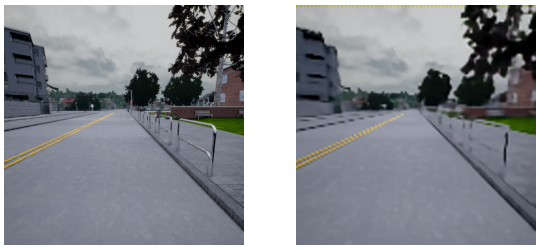
Model Pruning: Weekly Report 6

Patricia Gschoßmann

1. Weekly Progress

In this week additional tests regarding time and average memory consumption for last week's finished model were executed.

Moreover, I started to prune the RGB autoencoder from task 1. The goal is to reduce 65% of selected convolutional layers in the model's en- and decoder. The original model's results can be seen in fig. 1b. Input images are of size $128 \times 128 \times 3$. The images at the bottleneck are of size $16 \times 16 \times 256$, which in fact is not a real reduction. Unfortunately, this issue was only noticed during this week's pruning process. Training of a new RGB autoencoder, which reduces the images to a size of $16 \times 16 \times 128$ at the bottleneck, was started right away. Next week's goal is then to prune this model.



(a) Original image (b) Original model output

Figure 1: Original and reconstructed image data.

2. Results

2.1. Test results of last week's experiments

I executed additional tests for last week's model, which was trained with weight updates for the pretrained model from the beginning. The average time taken as well as the average memory consumption per batch was calculated. The test set contained 10000 images and each batch consisted of a single image.

2.1.1 On GPU

The tests were executed on a NVIDIA RXT 2070 SUPER GPU. On average, the original model took ≈0.0228 seconds for each batch, while the pruned model took ≈0.0059 seconds, which means that the smaller model was almost four times faster. However, no such improvement could be observed with regard to the average GPU consumption: The smaller model utilized only 0.02% less of it, than the original model; more precisely 49.88% compared to 49.90%.

2.1.2 On CPU

The tests were executed on an Intel i5-8250U (8) @ 3.400 GHz CPU. On average, the original model took ≈0.0452 seconds for each batch, while the pruned model took ≈0.0394 seconds. The difference is not as much as compared to the results on the GPU, however, it is still a noticeable improvement. Interestingly, the pruned model showed bigger improvements in terms of memory consumption: The original model utilized 17.83% of the CPU, while the smaller model only took up 17.25% - 0.58% less.

2.2. Results of this week's experiments

The original autoencoder uses transposed convolutions during the decoding process. I decided to not prune them, and consequently the previous convolutional layer neither. In this way the usual pruning approach can be applied, before venturing into transposed convolutions.
However, this actually leads to the same issue as with the original model: Since the layer before transposed convolutions is not pruned, especially the last layer before the bottleneck, the pruned model does as well not truely encode the images.

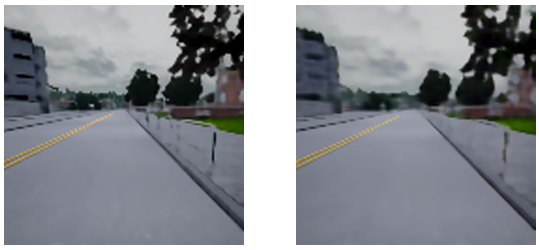
2.2.1 Pruning

Pruning with weight updates of the pretrained model turns out to be the most successful approach in the last weeks, which is why the same was applied for the following results. Again, a learning rate of 0.001 was used, as well as a step scheduler, which reduces α by 0.1 at each iteration. I used the validation loss as performance indicator, since no accuracy is available for this task. The original model reached a minimum validation loss of 1.69.
Except for $\alpha = 0$, the validation loss was the highest for the model with $\alpha = 0.9$ with $val\_loss = 2.37$. With each

α-decay, the performance improved - a different development compared to the experiments with the VGG, where the performance dropped with each iteration. At $\alpha = 0$, the model reached a minimum validation loss of 2.57. The training was resumed with a learning rate of 0.01. Unfortunately, this did not improve the performance. Currently, the training is further resumed with a learning rate of 0.1.

2.2.2 Model output

Even though the validation loss is yet not as low as desired, I was interested in how the model output looks. Intermediate results for $\alpha = 0.1$ and $\alpha = 0$ before pruning are shown in figure 2. One can see, that the reconstructed image for $\alpha = 0$ is blurrier than for $\alpha = 0.1$.



(a) Output at $\alpha = 0.1$ (b) Output at $\alpha = 0$

Figure 2: Reconstructed image data during the pruning process. The original image is depicted in fig. 1a.

3. Plan

Prune RGB autoencoder with real bottleneck and test its performance on CPU and GPU.

4. Summary of new scripts

- `autoencoder.py`: Architecture of the pre-trained RGB autoencoder. Subclass of `abstract_model.py`.
- `autoencoder_dataset.py`: RGB and Depth dataset used to train the autoencoder.