CNN Image Deblurring Analysis

Hanna Kienast, 12020688
Jonas Tatzberger, 12002213
Kateryna Ponomarenko, 12330660

February 2024

1 Introduction

For our image deblurring analysis we first chose our Convolutional Neural Network as suggested by the assignment task [6]. The implementation is from Seungjun Nah, it is a Deep Multi-scale Convolutional Neural Network for Dynamic Scene Deblurring. There are five trained and pre-trained models, trained on two datasets, GOPRO_Large, and REDS.
In a further approach to train our model, we also used another implementation from [1], which also uses Deep Learning for Image Deblurring.
We chose 400 random images of the suggested coco dataset [2] to test the performance of the (pre-)trained models. For training, testing and validating a new model we chose 3000 random images. We chose this limited number of images due to our limited resources. We applied four types of blur to our coco dataset: gaussian blur, box blur and simple blurring, using the python package pillow [3] and additionally the motion blur, for which the source code was taken from the tutorial [4]. We should note that for the second implementation only gaussian blur from the package opencv-python [5] was used to train the model. As a comparison metric, we chose the peak signal-to-noise ratio (PSNR) for which we implemented code to calculate the average PSNR of a whole folder of deblurred images.

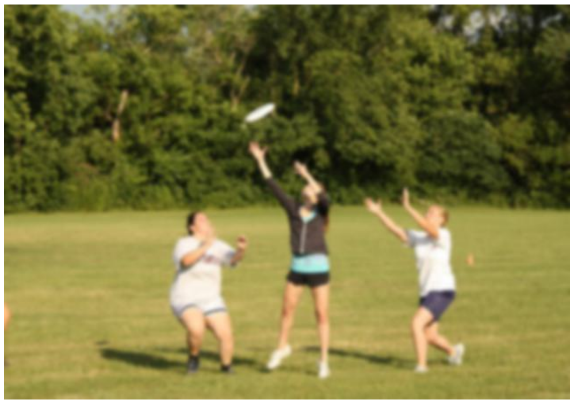
2 Results and Analysis

2.1 Applying pre-trained models on our dataset

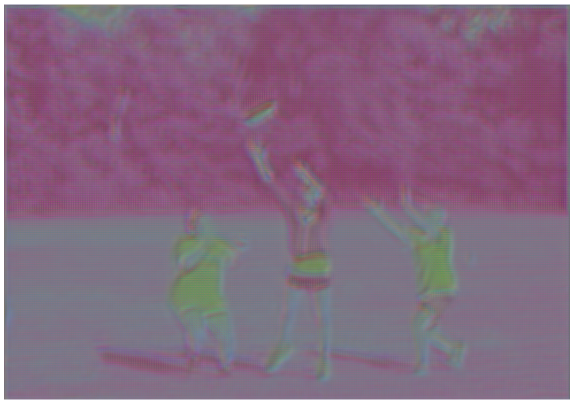
Our initial approach to run the code locally did not succeed as there was an incompatibility with the python package "readline", our windows system and the python version we were using. We found a solution by running the code on Google colab and also used its T4 GPU.
We started with testing all models on one kind of blur and chose the gaussian blur, as we would later want to choose only one model to work with. Below are our results for applying the (pre-)trained models on our blurred images dataset, so deblurring the blurred images. By default we used 1000 epochs, a learning rate of 1e-4, its decay at 500, 750 and 900 epochs with a factor 0.5.

model	Average PSNR [dB]	runtime [s]
GOPRO L1	27.91	408.568
GOPRO L1 amplified	27.87	423.241
REDS L1	27.90	417.356
REDS L1 amplified	27.93	409.782
REDS L1 amplified pre-trained	27.94	416.903

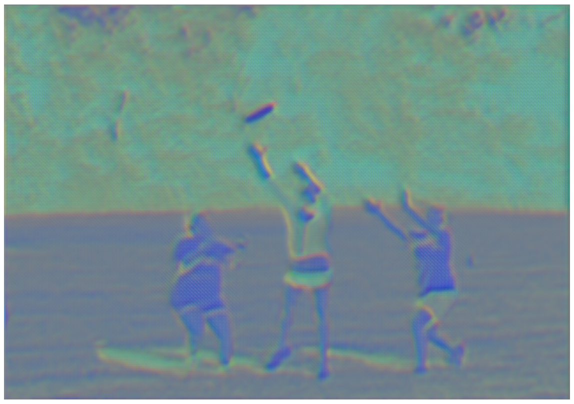
Table 1: Average PSNR and runtime for deblurring the gaussian blurred images with different pre-trained models.



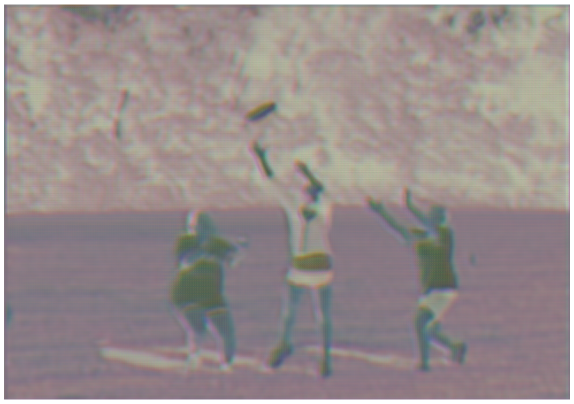
(a) The original gaussian blurred image.



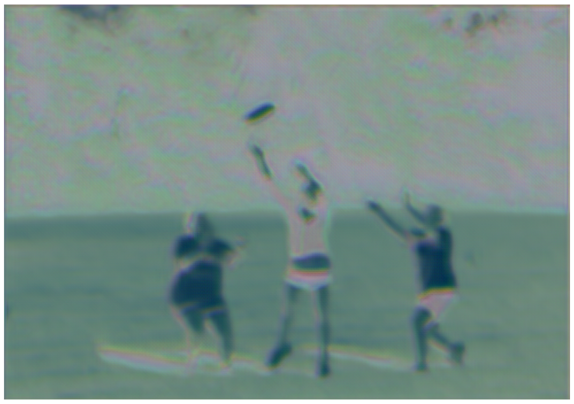
(b) Deblurred with model GOPRO_L1_amp.



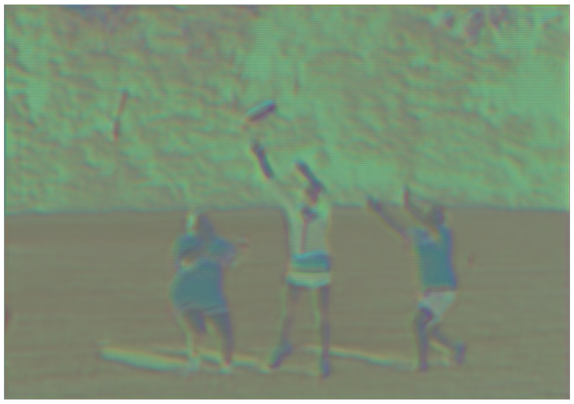
(c) Deblurred with model GOPRO_L1.



(d) Deblurred with model REDS_L1_amp.



(e) Deblurred with model REDS_L1_amp_pretrained.



(f) Deblurred with model REDS_L1.

Figure 1: Deblurring with the (pre-)trained models from [6].

A high PSNR value indicates a high similarity between the original and the reconstructed image. A PSNR value of 100 dB would mean that the input and output images are identical. In other words, a PSNR of around 27 dB suggests a rather low level of similarity between the original and reconstructed images. We can also look at the images themselves.

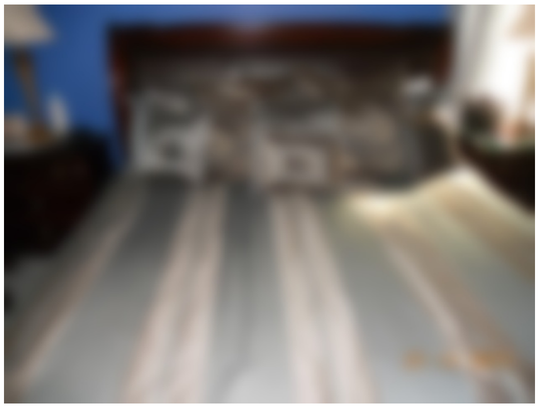
Since the pre-trained models did not perform as expected on our dataset we decided to not try them on other blur types and instead train our own model.

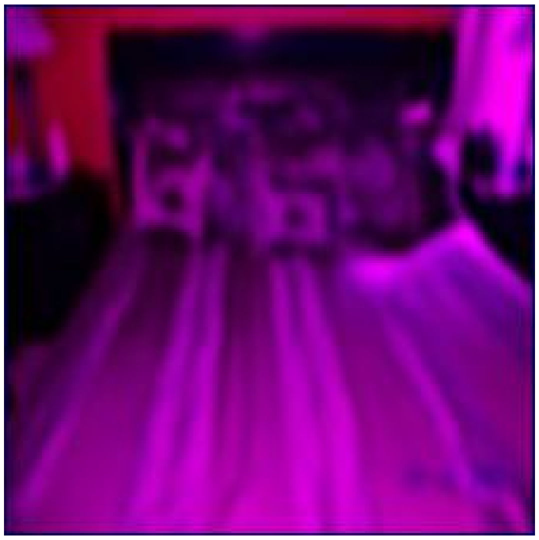
2.2 Attempt to train our own model

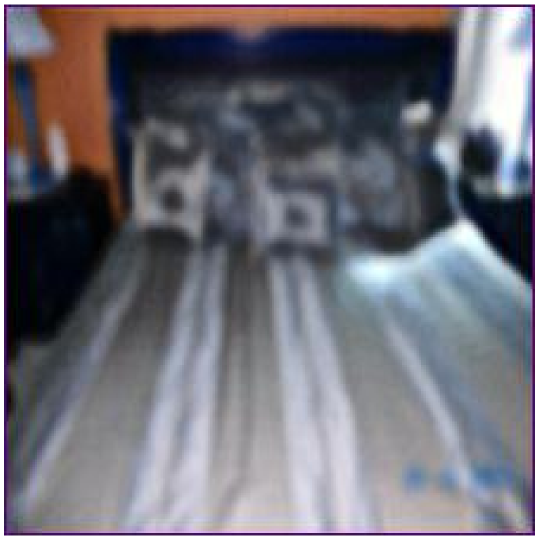
We split our dataset into training, validating, and testing (0.8, 0.1, 0.1) for sharp images and the different types of blurred images. We then tried to train our own model with the authors' instructions on their github. However, their implementation was not set out to be used on unlabeled data. As we had no experience with programming data loaders we were not able to change the code to satisfy our task. Instead, we looked for similar implementations to train a model.

2.3 Attempt to train our model using another implementation

We found another implementation from [1] using Deep Learning with Image Deblurring. We started training with 2547 images, which were split into training (1573 images), testing (449 images), and validating (525 images) set by the code itself. It took us 267.428 minutes to train and 20.13 seconds to test. By default, we ran the code with 50 epochs. The following are our results.
Comparing the sharp images and the deblurred images we established an average PSNR of 30.47 dB. While this seems like only a small increase to the previous models, we also had a look at the images. For one example image, the code would provide a deblurred image for every epoch. We will show a small collection:


(a) The original gaussian blurred image.


(b) Deblurred at epoch 1.


(c) Deblurred at epoch 20.

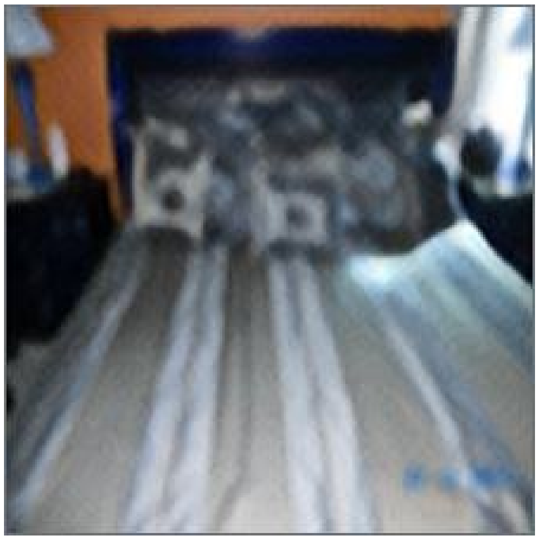

(d) Deblurred at epoch 50.

Figure 2: Different stages of deblurring.

We can see that the final product exhibits quite a deblur, contrary to the relatively bad PSNR value. We noticed that for the first 18 epochs, the image had turned pink, similar to the colorful images for the first implementation. The architecture of the deep learning model, particularly the initial layers, might introduce color transformations or adjustments as part of the feature extraction process. These results led us to believe that our attempt to run the GOPRO and REDS models was not successful due to the number of epochs or other running issues. It can also be seen that the code changes the layout of the image. In this example, we also noticed that deblurring and using the pinkish color lead to a different color of the background wall.
Another example of blurring and deblurring:

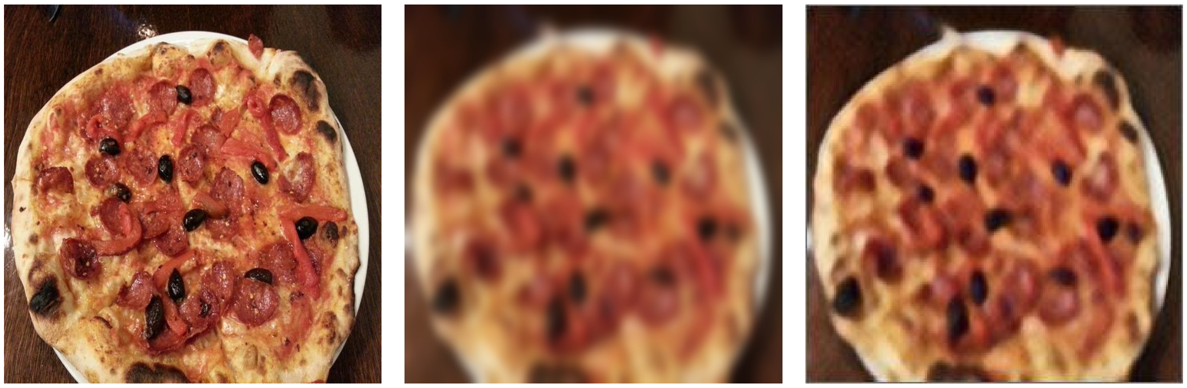

(a) The sharp image. (b) The gaussian blurred image. (c) The deblurred image.

Figure 3: A comparison of a sharp, blurred, and deblurred image.

We also had a look at the training and validation loss for every epoch. The sharp decline marks the epoch where the color changes from pink to the original color.

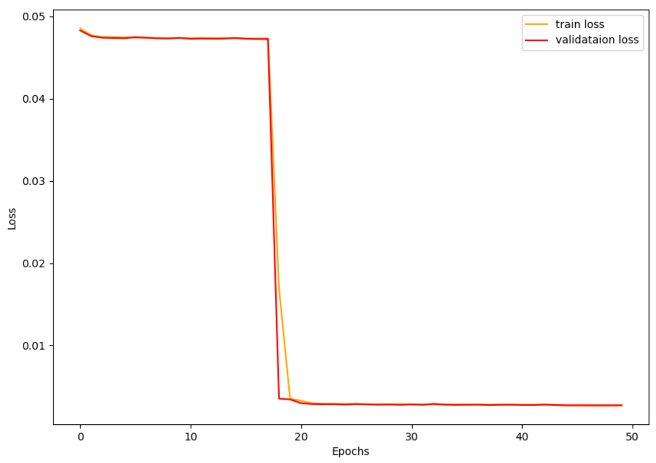
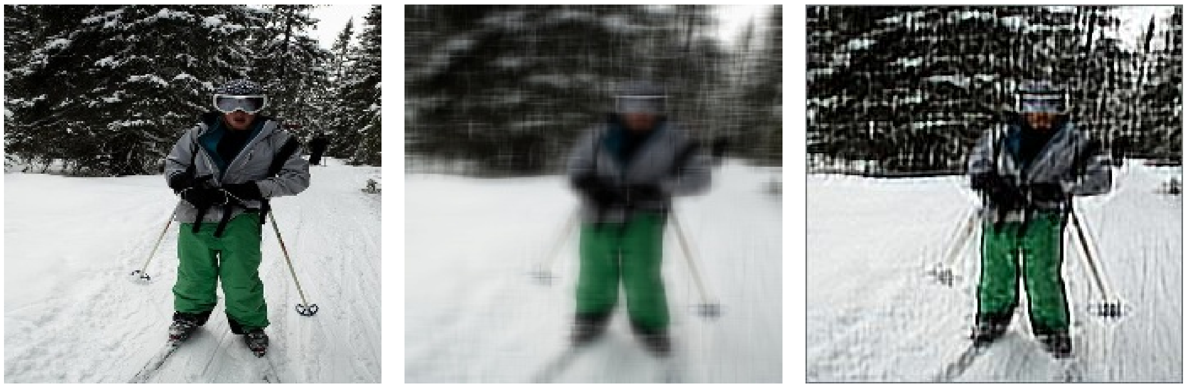


Figure 4: Train and validation loss.

We also tested our model on the motion-blurred coco dataset (451 images). This resulted in a PSNR of 29.62 dB, so quite similar to all previous examples. The testing took 20.23 seconds. We also had a look at one example image:

The model seems to work on these types of blurs as well, although not perfectly.



(a) The sharp image. (b) The motion-blurred image. (c) The deblurred image.

Figure 5: A comparison of a sharp, blurred, and deblurred image.

3 Conclusion

While the handling of the first implementation did not seem to work, we successfully deblurred images with the second implementation. The model works on gaussian and motion-blurred images and can be tested on several more types of blurs.

References

[1] URL: https://github.com/sovit-123/image-deblurring-using-deep-learning/tree/master.

[2] URL: https://cocodataset.org/#home.

[3] URL: https://pillow.readthedocs.io/en/stable/reference/ImageFilter.html.

[4] URL: https://www.geeksforgeeks.org/opencv-motion-blur-in-python/.

[5] URL: https://pypi.org/project/opencv-python/.

[6] Seungjun Nah, Tae Hyun Kim, and Kyoung Mu Lee. “Deep Multi-Scale Convolutional Neural Network for Dynamic Scene Deblurring”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.