EE-610 Assignment 2

ML Approach for reducing blur generated by PSFs

by Lars Wolf, 18V051003

Table of Contents

I. Abstract
II. Introduction
III. Background
IV. Approach
V. Test Images & Results
VI. Discussion
VII. References

I. Abstract

In this second assignment of EE-610, I focus on task two of the given options: Training a ML-model to deconvolve images that have been blurred by convolution with a point-spread-function *PSF*. To achive this goal, a deep CNN model with 6 hidden layers was used. Since the task of convolving an image with a PSF is well know and easy to compute, I did not create a seperate dataset with already blurred images, but the data is manipulated in-training to have as many different input images / patches as possible.

Results?

II. Background

Debluring images that have been convolved with a point-spread-function *PSF* is a very important task to improve the quality of photos with motion blur of the photographed objects or lense-shakes. The phenomenon can be easily replicated by creating a random kernel (our PSF) and convolving unblured images. The resulting images can then be used as input for a Neural Network for recreating the original images.
For this kernel creation, I originally tried to use an adjusted version of pyblur – originally for creating a new dataset. After finding a good example of tensorflow CNN of [1], I decided to use their approach to generate small subimages first to reduce training time. After longer periods of training, I came to find out that np.random is only randomized once after a function gets enabled to be used in tensorflow with tensorflows map_fn, wherefore I had to renew my approach and now use a uniformly random distribution as blur kernel.

III. Approach

For our model, I altered the code of [1], which resembles a tensorflow implementation of the model used by Zhang et al. for their paper on image denoising. The model is a Convolutional Neural Network with 6 hidden layers. As there was a problem with our code, dropout of 0.2 was only applied after epoch 24. Due to time constraints, we were not able to completely retrain our model, as training would have taken more then 15 hours.
Because of these limited resources, we were further only able to use patches of size 32 x 32 pixels as input. We still applied PSF-kernels of 10 x 10 pixels to them. To cover as many different PSFs as possible, the blur kernel was created using a random uniform distribution. The sum of each kernel is 1. We further could only compare to deeper CNNs that were trained on a lot less epochs due to the vast increase in training time for deeper networks.

IV. Training

Since the data was altered in the training process, we will first take a look at the used images in this section.
We generally used grayscale images to reduce training time. From 60 different images covering different nature scenes, 300 patches of 32 x 32 pixels were created. During training, the pictures are convolved with a 10 x 10 pixel kernel that follows a uniformly random distribution. This distribution was chosen as it best describes all possibilities of kernels (compared to normal or Poisson distributions which are symmetrical). For convolution, the original image is zero-padded which results in darker borders of the images. Additionally, Gaussian noise is added with a random normal distribution.
As we found a bug in our model, training was performed for 24 epochs without dropout, and then another 5 epochs with dropout. Even though our input data is altered in each epoch, we chose this step as our model is fairly deep and has a lot of parameters. Overfitting parts of our input might therefore still be a possibility.

Due to the late introduction of our model to dropout the loss curve shows a huge incline at 7200 steps.

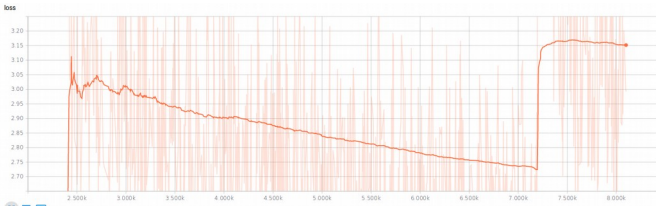

Illustration 1: Loss before and after introduction to dropout

We evaluated our model on the PSNR scores of 12 development images that were not contained in the training set. The model with the best PSNR of the dev set was then chosen for evaluation.
The highest average PSNR was 23.7 and was achived at 7200 steps, just before dropout was applied. This suggests, that the results would have still improved given a longer training time.
Dropout on the other hand was not able to improve the result immediately. This suggests both that overfitting might not be a big issue (especially since we are altering our training data every epoch) and that 3 epochs with dropout do not have a sufficient effect on the weights to change the learned relationships.

V. Results

Illustration 2: Test images from model trained for 7200 steps

The results of our network definitely show the potential of the used approach. In Illustration 2 compares the original image on the left, an altered version in the middle and the prediction of the model on the right. As clearly visible, the network learns to fill the border of the image (that was influenced by zero-padding) very well. Furthermore, the resulting images appear a lot more sharp. Nevertheless, many slight variations go missing – this is especially visible for the small circles on the starfish or the textures of the cloths of the woman. Still, some details are predicted even when it is very hard to tell from the blurred image – e.g. the letters on the Air Force Jet in Illustration 3

Illustration 3: Results on test image 6

VI. Discussion & Conclusions

As already described in *Background*, I had some bigger issues in creating a proper machine learning model to begin with. Due to the high training times, I was only able to catch some errors after longer periods of time, wherefore I ran into some time issues. Nevertheless, the achived results look promising in the sense that the network is definitely able to learn very different Blur-Kernels. The resulting images sometimes contain ringing artifacts, but generally look more sharp and clear than the blurred images. If there would have been more time, training the network with dropout originally, with larger patches and for a longer period of time would have probably improved the results even further.

The used code can be found here:
https://github.com/D0nPiano/EE-610-Assignment-2

VII. References

[1] https://github.com/crisb-DUT/DnCNN-tensorflow

[2] *Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising* (2017), Zhang et al, https://github.com/cszn/DnCNN

VIII. Illustration Index

Illustration 1: Loss before and after introduction to dropout.....

Illustration 2: Test images from model trained for 7200 steps.....

Illustration 3: Results on test image 6.....