Assignment 3: superresolution with CNNs

You have seen examples of convolutional neural networks (CNN) in the lectures. In this assignment you will implement a convolutional layer from scratch. Then you build a fully convolutional network (no pooling or fully-connected layers) to perform a super-resolution task. In this task an image is gradually transformed into a higher resolution version of itself. The idea[1] is to downscale image $I^0_{n\times n}$ to $I^1_{n-1\times n-1}$ .. $I^k_{n-k\times n-k}$ using for example bicubic interpolation. Then a CNN is trained to upscale the image. Specifically, the network learns to reconstruct each image $I^{k-1}_{n-k+1\times n-k+1}$ from $I^k_{n-k\times n-k}$. At this point we apply I^0 to this CNN to generate $I^{-1}_{n+1\times n+1}$. Next we train *a new CNN* on $I^{-1}, I^0..I^{k-1}$ as a basis to get to image I^{-2}. We can repeat this procedure till we get to the image with the desired resolution.
The CNN consists of 8 hidden convolutional layers each with 64 channels. The kernels are all 3×3 with padding of 1 to preserve the size of the original input. In other words, the input and output images of the CNN have the same size. To account for the upscaling that the network is supposed to learn, the input image is first upsampled by bicubic interpolation to get to the desired size. The network is then trained to *adjust the difference* between the input image and the target image. In particular, if H and J are the input and output images respectively, the network learns to output $J - H$[2] using mean squared error between the pixel values of the input image and the difference image.
Note that because the CNN is fully convolutional, the architecture is the same for any input dimensionality (though the number of channels of the input cannot change).
The whole process is as follows:

(1) Implement a convolutional layer as a Python class. You can use numpy and scipy libraries in your code and may look into online resources as a reference but overall the code should be your own work. This class should have at least one property for kernels (can be a 4-dimensional numpy array for example) and one for biases (there is one bias value for each output channel). The number of kernels will be $input\_channels \times output\_channels$ so for example if the layer gets an RGB image and generates 7 output channels, the number of kernels will be 21 (for our problem each kernel will always be 3×3). Furthermore, the class should have at least 3 methods, one constructor, one $forward$ method (calculates and returns the output from the input), and a $backward$ method (calculates and returns the gradients for any proceeding layers).

(2) Implement a ReLU layer as a Python class. This class should have at least one $forward$ and a $backward$ method.

(3) Build a CNN with 8 hidden convolutional layers, each proceeded by a ReLU layer. Add a final output convolutional layer (no ReLU afterwards). The first and final channels are going to be 3 (to account for RGB images). Apply He[3] initialization method to initialize the kernel values in your network.

(4) Implement the super-resolution algorithm discussed earlier. The algorithm will get one square image I^0, and the preferred output resolution and output an image with the desired resolution. This upscaling happens in several steps, where at each step one CNN is trained and applied to add one pixel to the original image. The training set of each CNN consists of 7 image pairs, for the original image this set will be $\{(\hat{I}^0, I^0-\hat{I}^0), (\hat{I}^1, I^1-\hat{I}^1), (\hat{I}^2, I^2-\hat{I}^2), (\hat{I}^3, I^3-\hat{I}^3), (\hat{I}^4, I^4-\hat{I}^4), (\hat{I}^5, I^5-\hat{I}^5), (\hat{I}^6, I^6-\hat{I}^6)\}$, where $\hat{I}^k$ is the upsampled image from I^{k-1}. This down/up sampling is done with rescale method from

[1]This is based on the method discussed in http://openaccess.thecvf.com/content_cvpr_2018/papers/Shocher_Zero-Shot_Super-Resolution_Using_CVPR_2018_paper.pdf, though the details are different.

[2]It does not learn to generate J itself.

[3]https://www.cv-foundation.org/openaccess/content_iccv_2015/papers/He_Delving_Deep_into_ICCV_2015_paper.pdf

OpenCV2. After we train a CNN on this set, we will apply it to $\hat{I^{-1}}$ to get I^{-1}. Then in the next step I^{-1} is used to generate I^{-2} in a similar manner. We continue this procedure till we reach the desired image size.

This assignment can be completed as a group of no more than three students. The code can be written in python but C/C++ is also acceptable. The files you submit on Webcourses should include:

- An extensive pdf report explaining details of what you did in each step. This report should be self-sufficient (do not refer to the code in your report) and include all the details of your algorithm as implemented in your code (a brief explanation is not enough). If you made any decisions in your implementation (for example your code accepts grayscale image in addition to RGB images) it should be mentioned in the report. Finally, the report should include some sample images and the produced high-resolution images.
- The source code for the convolutional layer class. This code should be written independent of the other components of the program. A small test case that shows how a convolutional layer is defined and a sample forwards, backward function calls should also be included as part of the documentation for this code.
- The source code for the ReLU class. This should also follow the same guidelines as for the convolutional layer class.
- The source code for the super-resolution algorithm discussed. Your code should be able to read at least two image formats, png and jpg, and save the resulting image in the same format. At least one sample image and the necessary code to produce the output image should be included in your submission.