May 21, 2025

Project 2 – PnP with CNN denoisers, DIP, and SURE

General information:

In this work we consider **the same 8 test images from Project1**: cameraman, house, peppers, Lena, Barbara, boat, hill, and couple. They can be downloaded from moodle.
Each ground truth image should be considered in its original size (i.e., do not use resize) and after loading, its intensity values should be divided by 255, so the range is within [0,1].

Methods will be quantitatively evaluated via the PSNR, given in this case by $PSNR = 10\,log_{10}\left(\frac{r^2}{MSE}\right) = 10\,log_{10}\left(\frac{1}{MSE}\right)$, where $MSE = \frac{1}{n}\sum_{i,j}\left(\hat{x}(i,j) - x_{gt}(i,j)\right)^2$.
Image processing methods typically require hyper-parameters tuning, e.g., aiming for maximizing PSNR. Note that it is not valid to tune hyper-parameters differently per image, because in "real life" the ground truth image is unknown. On the other hand, it is considered valid to optimize via the PSNR a **single** tuning configuration that is shared by a (sufficiently large) test set. Typically, good tuning for 5-10 diverse images will perform well also on larger test sets.

- The project report must be printed in Word or LaTeX, and submitted electronically as a PDF file.
- This project needs to be implemented in **Python**. Include all your code within the document as an appendix, with syntax coloring and indents, as it appears in the Python editor.
- Wrap up the report and the code files inside a single ZIP file with **name that includes your ID**.

Q1. ADMM-PnP with CNN denoisers

Consider the same dublurring setting from Project1:
$y = x_{gt} * k + e$ where x_{gt} is the original image (after division by 255), $e \sim \mathcal{N}(0, \sigma_e^2 I)$, and k is a blur kernel. We assume that the 2D convolution is **cyclic**. Specifically, we consider a setting where $\sigma_e = 0.01$ (use random seed of 0) and $k(i,j) = \frac{1}{1+i^2+j^2}$, $i,j = -7,\ldots,7$, followed by scaling such that $\sum_{i,j} k(i,j) = 1$.
In Project1, you have written a script `runme_deblurring` where you apply the ADMM-PnP on the test set. Here you are requested to build on this code to study the performance gaps between using the BM3D denoiser and CNN denoisers (omit your previous code on bilateral filter and introduce a flag to select between BM3D and CNN denoiser).

First, reach the best results that you can obtain for ADMM-PnP with the BM3D denoiser. You are allowed (and expected) to fix any issue that may occured in yours Project1's

implementation and also improve the hyper-parameter tuning, e.g., by adding flexibility to the hyper-parameters (denoted ρ, β in the slides).

Now, the goal is to see if these results can be outperformed when using pretrained CNN denoisers. Specifically, we consider the IRCNN variant of DnCNN, termed 'ircnn_color' or 'ircnn_gray' provided in:
https://drive.google.com/drive/folders/13kfr3qny7S2xwG9h7v95F5mkWs0OmU0D
(despite its name, note that 'ircnn_color' can be applied on grayscale images).
Use the repo: https://github.com/tomtirer/IDBP-python to get an idea how these models were incorporated into a different PnP scheme.

Denote by $\hat{x}_{PnP-BM3D}$ and $\hat{x}_{PnP-CNN}$ the estimates of ADMM-PnP with BM3D denoiser and IRCNN denoiser, respectivly. Report in a table the PSNR of y (input PSNR) as well as the PSNR of $\hat{x}_{PnP-BM3D}$ and $\hat{x}_{PnP-CNN}$ per image and their average PSNR across images.
State clearly what hyper-parameters values gave the best results for each of the methods $\hat{x}_{PnP-BM3D}$ and $\hat{x}_{PnP-CNN}$. Recall that per method you need to use a single tuning configuration (shared by all the test images).

Provide several visual examples ($x_{gt}, y, \hat{x}_{PnP-BM3D}$, $\hat{x}_{PnP-CNN}$). You can save the results as jpeg instead of png to reduce memory storage. **State the PSNR below each image of an estimate**.

Q2. Deep Image Prior for denoising

Here, the goal is to output $\hat{x}$, an estimate of x_{gt}, given $y = x_{gt} + e$ where x_{gt} is the original image (after division by 255) and $e \sim \mathcal{N}(0, \sigma_e^2 I)$ with noise level of $\sigma_e = 0.1$. Fix the random seed to 0 before generating the noise (this will make the results reproducible).

Q2a.

Write a script `runme_dip_denoising` that addresses this problem by the Deep Image Prior (DIP) approach. Use their repo (build directly on their denoising.ipynb):
https://github.com/DmitryUlyanov/deep-image-prior
Present a figure of the average PSNR (averaged over the 8 test images) vs optimizer iteration.

Let the number of iterations be large enough to demonstrate an increase in the average PSNR and then a decrease.

Q2b.

Add a flag to `runme_dip_denoising` that allows replacing the plain least squares loss used in DIP with the SURE loss. Implement this loss. Recall that for applying SURE you also need to modify the input to the network that is used by plain DIP.

May 21, 2025

Present a figure of the average PSNR (averaged over the 8 test images) vs optimizer iteration with a large number of iterations.

Provide several visual examples ($x_{gt}, y, \hat{x}_{DIP}, \hat{x}_{DIP+SURE}$) along the iterations, before and after DIP's overfit occur. **State the PSNR below each image of an estimate**.