Computational Science and Engineering
Semester Project
Master Semester 3 - Fall 2022

Light Field Denoising Using Conventional And Learning-Based Methods

Student: Tomás Soares de Carvalho Feith

Supervised by: Michela Testolina
Prof. Dr. Touradj Ebrahimi

January 11, 2023

Multimedia Signal Processing Group
EPFL

Abstract

In this project, we conducted a thorough investigation of existing image and light field denoising methods, and proposed a novel learning-based method for light field denoising. We also acquired a large dataset of light fields and developed a pipeline for generating synthetic noise. Our comparison of BM3D [1], a classical image denoising method, with LFBM5D [2], its light field denoising extension, showed that while LFBM5D offers a significant performance improvement, it is slower due to its increased dimensionality. Additionally, in scenes with large angular spacings, none of the light field denoising methods outperformed their image denoising counterparts. Our novel method is based on a patch-matching algorithm that builds stitch-patched versions of each subaperture image (SAI) using neighboring SAIs and a learning-based denoising model. While it performs slightly worse than the original DnCNN, we show, via image quality assessment metrics and qualitative tests, that it is capable at denoising light fields. In future work, we plan to improve the model architecture and optimize the patch-matching code.

All code produced under this project is available at https://www.github.com/tsfeith-epfl/LightField_Denoising.

Contents

1 Introduction

A light field is a function that describes the amount of light passing through every direction and all the points in the 3-D space, which leads to a five-dimensional function $f(x, y, z, \theta, \phi)$ which expresses the light intensity at point (x, y, z) and along the angular direction (θ, ϕ). This function is frequently called the plenoptic function, a term coined by Adelson and Bergen (1991) [3]. One way of considering Light Fields is as a matrix of different views, where each view is called a Sub-Aperture Image (SAI).

While the concept of light fields is not new, only recently has the technology to capture them become available, either with multi-camera arrays or with plenoptic cameras such as Lytro and Raytrix. The main applications for light fields are image refocusing [4], novel view synthesis [5], and depth estimation [6]. However, both due to the novelty of the technology and to the limited light throughput in some light field cameras, light field denoising is a task of vital importance. Even though standard image denoising is a field that has been extensively studied, with proven success, there is still a lot to study with light field denoising. Namely, the multiple views composing a light field contain redundant information, which allows for directions of study that were not previously possible.

In this project, we propose a learning-based light field denoising method, which exploits the redundant informant present in neighboring views by building "patch-stitched" versions of each SAI from similar patches in other SAIs. The patch-stitched images are then passed through a convolutional neural network (CNN) inspired by the DnCNN [7] architecture. We compare our proposed method with state-of-the-art image and light field denoising methods, and measure their generalization capabilities for realistic noise. Finally, we perform a visual perception analysis on selected key scenes, allowing us to see how each method performs in different scenarios.

2 State-of-the-Art Review

2.1 Noise Generator

Capturing data necessarily leads to noise. In image capture, this noise can be attributed to two main factors. The first factor comes from the randomness in the number of photons arriving at the sensor per unit time. The second factor stems from sensor noise.

All the denoising models developed have the ultimate goal of being used in real-world tasks. To correctly quantify their performance, it is necessary to test them in a controlled environment. The way to do that is by using noise generators, that turn a clean image into a noisy one. In this project, we'll consider two kinds of noise.

2.1.1 Additive White Gaussian Noise

One of the most studied forms of noise is additive white gaussian noise. Under this model, the noisy image is assumed to take the form

$$z(x) = y(x) + n(x) \tag{1}$$

where $x \in X$ is the pixel position in the domain X, $z : X \longrightarrow \mathbb{R}$ is the noisy image, $y : X \longrightarrow \mathbb{R}$ is the clean image, and $n : X \longrightarrow \mathbb{R}$ is the noise. The noise is simply characterized as

$$n(x) \sim \mathcal{N}(0, \sigma) \tag{2}$$

where σ acts as a strength parameter for the noise generator.

2.1.2 Realistic Noise

While white gaussian noise is the most studied and the easiest to implement, it is not the most realistic. A more accurate noise generator allows us to get a better estimate of the performance of the denoiser. Alvar and Bajić (2022) [8] developed a noise generator that simulates realistic noise found in smartphone cameras. This noise is sampled from a Poissonian-Gaussian model [9].

The noisy images take the same general form as in Equation 1. However, now the noise is defined as

$$n(x) = n_p(y(x)) + n_g(x) \tag{3}$$

where n_p is a Poissonian signal-dependent component and n_g is the Gaussian signal-independent component. Each one of these components is characterized by

$$\frac{1}{a}\left(y(x) + n_p(y(x))\right) \sim \mathcal{P}\left(\frac{y(x)}{a}\right) \tag{4}$$

$$n_g(x) \sim \mathcal{N}(0, b) \tag{5}$$

With the distributions presented above, the noisy image $z(x)$ will have a variance of the form

$$\mathbb{V}[z(x)] = a \cdot y(x) + b \tag{6}$$

where $a > 0$ and $b > 0$ are the parameters of the Poisson distribution ($\mathcal{P}$) and the Gaussian distribution ($\mathcal{N}$), respectively. The fact that the variance has this formula, means that it is not possible to get one value of variance valid for all images, as is in the additive white gaussian model.

In their original work, the parameters were estimated from the Smartphone Image Denoising Dataset [10].

2.2 Image Denoising

Image denoising is a key step of many practical applications, which makes it a very active area of research. The typical approach to image denoising is to consider the noisy image as a noise map added to the original image, and the task is to restore it to its original state.

Over the years, several classical methods have been proposed, like non-local self-similarity models [11] [12], sparse models [13] [14], or gradient models [15] [16]. However, most of these classical methods suffer from the same drawbacks. Firstly, they tend to be time-consuming and computationally expensive [17]. Secondly, these methods often rely on several hard-coded parameters, which makes it harder to achieve optimal denoising.

More recently, with the rise of machine learning, several learning-based approaches surfaced. Early work focused on discriminative models that attempted to learn image priors like Cascade of Shrinkage Fields [18] or Trainable Nonlinear Reaction Diffusion [19] [20]. More recent work mainly focused on Convolutional Neural Networks (CNNs) [21] [22] [23] and Generative Adversarial Networks (GANs) [24] [25]. The usage of CNNs stemmed from its ability to exploit local information efficiently, while GANs surfaced as an alternative for handling real noisy images, for which deep CNNs can be insufficient [26].

In this project, we'll consider two classical methods and one learning-based method. The learning-based method is a CNN since this is still the predominant architecture used for image denoising. Wavelet denoising was chosen as the baseline, as it's the simplest and fastest method. BM3D [1] was picked because of its high performance but also because there is a generalization of this method to Light Fields called LFBM5D [2]. Finally, the DnCNN [7] method was selected because the novel method introduced in this work is a generalization of it to Light Fields. By looking at the image denoising method, and its light field denoising variant, it is possible to compare the two and see the benefits and costs of each one.

In the following sections, we'll introduce the methods chosen.

2.2.1 Wavelets

The main technique behind wavelet denoising is the wavelet transform, which produces a series of coefficients that are an image representation in the transformed domain. In the transformed domain, typically "large" coefficients correspond to the signal, and "small" coefficients correspond to the noise signal [27]. So, by applying a thresholding operator on the transformed domain, and then applying the inverse transform to the thresholded coefficients, the resulting image should have less noise.

In this project, we'll use the implementation from SciPy [28], which implements wavelet denoising via adaptive thresholding by computing separate thresholds for each wavelet sub-band, instead of one universal threshold, as described in [29].

2.2.2 BM3D

Block-matching and 3D Filtering, introduced by Dabov *et al.* (2007) [1], is a non-local self-similarity model used for image denoising, and even though it is over a decade old, it is still used as a baseline for most new methods proposed.

This method is composed of three stages: grouping, collaborative filtering, and aggregation. In the grouping stage, the image is first divided into patches. Then, for each patch, it tries to find similar (potentially overlapping) patches and stacks them together in a 3D array. In the collaborative filtering stage, a 3D transformation is applied to this array, and, within the transform domain, the coefficients are thresholded to remove the noise. The remaining coefficients are used for an inverse transformation, and the patches are returned to their original positions. Finally, in the aggregation stage, since the similar patches were chosen in a way that allowed overlapping, the final result is obtained by averaging the overlapping patches. An additional second stage can be added, where the process is repeated using now Wiener filtering, instead of thresholding, for the collaborative filtering stage. According to the authors, this second stage significantly improves the results over the thresholding stage.

Several heuristics are used to speed up the process, like only searching within a neighboring region of the reference patch, using strides bigger than 1 to search for the similar patches, or using only separable transforms. These choices increase the efficiency of the method, and thus its applicability.

In this project, we chose not to use the implementation from the original authors [30], but the rather the one submitted alongside the LFBM5D implementation [31]. This was because the original BM3D was provided as Matlab code, and we felt that a C++ code made it more accessible. Also because using the codes for BM3D and LFBM5D with exactly the same compilation settings and consistent mode of usage makes it fairer to compare them in terms of run-time complexity.

2.2.3 DnCNN

The DnCNN method, by Zhang *et al.* (2017) [7], is a CNN-based model for image denoising. However, instead of trying to infer the clean image from the noisy image, it tries to infer the noise map and then subtracts it from the noisy image to get the denoised one.

It is a deep CNN, with 20 layers for RGB images and 17 for Grayscale, and the authors trained it for blind denoising. The latter is important especially for realistic noise, as in that case the noise variance is harder to correctly estimate, as it depends on the image values (see Equation 6).

In our work, we'll use the original implementation [32].

2.3 Light Field Denoising

Light Field denoising is a much newer field, still in its start. While all the image denoising methods can be trivially applied to light field denoising, by looking at a single SAI at the time, the ultimate goal of light field denoising would be to use the redundant information present in multiple views to improve the quality of the captured light fields.

Previous classical attempts at light field denoising split the light field into 2D slices and process those [33], or try to take into account the 4D structure of light fields by using Gaussian Mixture Model light field patch priors [34]. A particularly interesting approach is to stack the SAIs and use video denoising, where each SAI is considered to be one frame of the "video" [35].

Learning-based methods for light field denoising are much more scarce. One of the main methods available [36] uses anisotropic parallax analysis to guide the denoising process, which is done via two CNNs trained jointly. However, there don't seem to be other alternative approaches, which suggests that this is a challenging problem, with potentially many directions of study.

2.3.1 LFBM5D

In this project, we'll consider a method called LFBM5D, presented by Alain and Smolic (2017) [2], which is a generalization of the BM3D method introduced in Section 2.2.2. Similarly to BM3D [1], this method stacks the SAIs into 5D patches, and these patches pass through a two-stage process: first they pass through a 5D transform and their coefficients are hard-thresholded, then the resulting coefficients are filtered in the transform domain and that leads to the denoised estimate.

More specifically, the patches are constructed in the following way. Taking a specific SAI, a 2D patch P of shape $[k \times k]$ is selected. Then, an angular search window of shape $[n_a \times n_a]$ is defined around the SAI (i.e. a sub-grid of shape $[n_a \times n_a]$ centered in the current SAI). Finally, the N most similar patches to the target patch P are found in the angular search window, thus producing a 5D stack of shape $[n_a \times n_a \times k \times k \times N]$. Then these patches suffer a process of collaborative filtering similar to the one described for BM3D, and the final result stems from an averaging of the estimates for each overlapping block.

In our work, we'll use the original implementation [31].

2.3.2 Related Tasks

Image denoising falls under the broader category of image restoration and enhancement. As such, there are closely related fields that should also be considered, as results from them can be translated into denoising. In this project, we considered image superresolution, the task of producing a high-resolution image from a low-resolution one. Just like denoising, it attempts to produce the original image from a degraded version of it (in this case, a down-sampled version of it)

A successful attempt at light field superresolution, by Fan *et al.* (2017) [37], used a 2-stage CNN, and the way they exploited the 4D structure of light fields was by applying a patch match algorithm to produce "stitch-patched" versions of a SAI from other views of the scene. This idea led to the method proposed in this project, which is described in more detail in Section 3.2.

2.4 Light Field Datasets

A large dataset of Sub-Aperture Images was constructed from several other datasets. They, and any post-processing needed on these images, are described in Table 1. Decoding from ESLF was done using the Light Field Toolbox for MATLAB [38] and decoding from LFR was done using PlenoptiCam [39].

The full dataset has a total of 492 scenes, with and 28'604 images, and it includes natural images, geometrical shapes, synthetic images, various light conditions,... This dataset was divided

Table 1: Description of datasets used.

Dataset	Scenes	Post-Processing Applied
Stanford Lytro Light Field Archive [40]	294	Decoding ESLF format to SAI grid
EPFL Light Field Dataset [41]	79	Decoding LFR format to SAI grid
INRIA Synthetic Light Fields Dataset [42]	48	None
4D Light Field Benchmark [43], [44]	27	None
INRIA Lytro Dataset [45]	27	Decoding LFR format to SAI grid
Stanford Light Field Archive [46]	10	None
MIT Synthetic Light Fields Dataset [47]–[50]	7	None

into a training set with 468 scenes, a validation set with 18 scenes and a testing set with 6 scenes. It was necessary to gather such a large dataset because we needed the train the novel to generalize, and so it was important to have a large number of diverse scenes.

3 Proposed Method

For the novel method, we consider the DnCNN model as our baseline and attempt to extend it for Light Fields, not just as a trivial single-image usage, but in a way that exploits the redundant information present in neighboring images.

3.1 3DDnCNN: A not-so-trivial generalization

Taking inspiration from the original DnCNN, the most trivial way of generalizing it to light fields is to swap the 2D Convolutional layers with 3D Convolutional ones. That was the first attempt at developing a novel method.

This modification of the original led the convolutional kernels to take shape $3 \times 3 \times 3$, instead of 3×3 (with the extra dimension being depth). What this means is that each SAI would ideally be able to extract added information from its two neighboring SAIs. For this to work, the 2D image grid was reduced to 1D via "snail sort", which flattens the grid by traveling in a spiral from the top-left image to the center. This method did not work, and there are two main reasons for that.

First, by flattening the SAI grid, we lose a lot of spatial information, which is only partly recovered by the sorting heuristic. Ideally, we would use 4D Convolutional layers, which would allow us to keep the full spatial information from the grid. However, there is no native implementation for 4D Convolutions, mostly because there aren't many applications for which they would be useful[1]. While it is possible to construct a 4D Convolutional layer by grouping 3D layers, as implemented by [51], it would slow down both training and inference[2].

However, the most important explanation for the failure of this method stems from the very nature of CNNs, and how they are not suited to handle this particular problem with this approach. CNNs search in a local neighborhood to find small structures to exploit. However, the images in the SAI have non-negligible disparities, as is demonstrated in Figure 1. These are three neighbor images in the grid, and the enhanced detail is in the same pixel position for all of them. Nevertheless, even with a large 150×150 window, the change of perspective from one image to the next would make it extremely hard to find local patterns. Considering now that the CNN would only see 3×3 patches for each image, it is unrealistic to assume it would be able to adequately use the extra information from the neighboring images.

[1] That was the opinion of the PyTorch development team, as of writing this report. See https://discuss.pytorch.org/t/is-there-a-way-to-realize-4d-convolution-using-the-convnd-function/5999/4

[2] This is due to its implementation, as it would rely on a `for` loop to navigate through the 3D layers, which is notoriously slow in Python, and can't be vectorized with GPU.

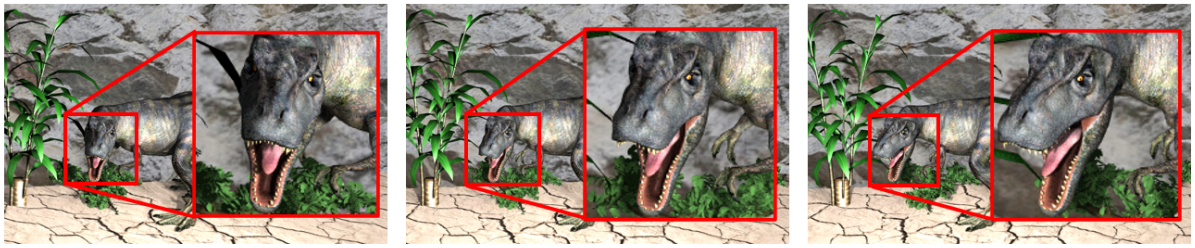


Figure 1: 3 consecutive figures from SAI grid, with magnified detail of size 150×150 in same relative position.

So, it is necessary to consider a new approach to apply CNNs for Light Field denoising.

3.2 LFDnPatch: Method Description

To be able to apply a CNN similar to DnCNN to denoise light fields, it is necessary to fix the disparity problem. The simplest solution would be to shift the neighboring images of every SAI so they fit better. But, in practice, this would not work, as the shift would not be constant throughout the image and the implementation of such a method would be most likely very inaccurate and computationally expensive.

However, it is possible to construct images similar to a given SAI using the remaining images in the scene. For that, and following the work from [37] on super-resolution, a set of "stitch-patched" versions of the image is produced, and those are used to give extra information to the CNN. A detailed explanation of how these stitch-patched versions are built is in Section 3.2.1.

Two examples of such stitch-patched versions are displayed in Figure 2. The top images were taken from a scene with very large angular spacing between SAIs, which leads to the very visible block artifacts on the stitch-patched version. This happens because the large angular spacing means that there are a lot of occlusions and disparity effects, and so while the found patches are similar to the reference ones, when stitched together they don't fit.

On the other hand, on the bottom images, it is almost impossible to spot such artifacts, which suggests that the patch matching method gives adequate results under reasonable levels of angular spacing.

3.2.1 Method Overview

This method takes an entire grid of SAIs as input. For each image, it builds N stitch-patched versions ($N = 6$ in out implementation). These images are built by decomposing each SAI into non-overlapping patches, and then finding similar patches in the other SAIs, where the similarity is computed via mean absolute error. The first stitch-patched image is built by using the most similar found patches in place of each of the reference patches, the second image using the second most similar, and so on. This way we get several images which are locally similar to the original one, and result from the stitching together of all these patches, hence the name. The images will never be identical, because of disparities and occlusions introduced by the different viewpoints, but by making them locally similar we make it easier to exploit redundancies for denoising.

The built images are stacked along the channels dimension, producing an array of shape $[H, W, (N + 1) \cdot C]$, where C is 3 for RGB images and 1 for grayscale. The last C channels correspond to the reference image, i.e. the image used to build all the stitch-patched versions. This augmented image is fed to a network identical to the DnCNN except for the input shape for the first convolutional layer. The resulting output of the model is the noise map which, subtracted from the original noisy image, will give us the denoised estimate.

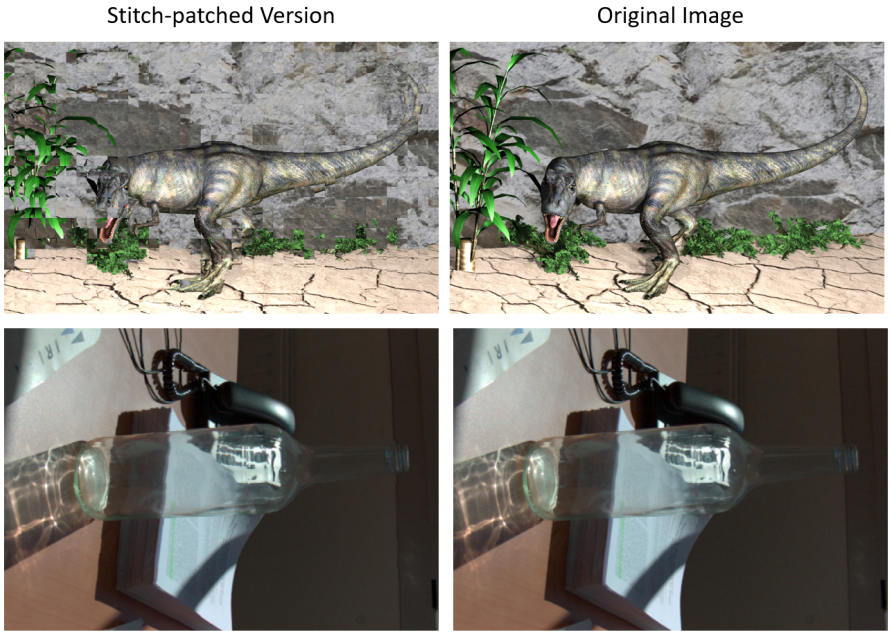


Figure 2: Examples of original image and stitch-patched version for two scenes. The top scene has wide angular spacing (so the local similarity isn't enough to build a coherent image) and the bottom has a short angular spacing (locally similar patches fit well together to form a globally similar image).

3.2.2 Patch Match Algorithm Description

It is worthwhile explaining the patch-matching algorithm in more detail because several heuristics had to be used to decrease the run time. So, in this section, we'll present a detailed explanation of the algorithm implemented.

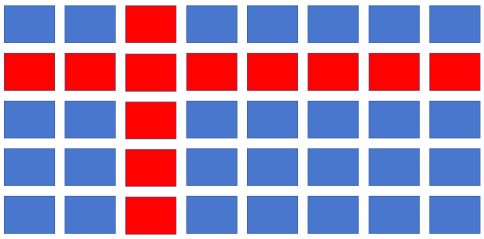
Considering an image grid of shape $[G_H, G_W]$, composed of SAIs of shape $[H, W]$, the goal is to compose N "patched-stitched" versions of the SAI in position (i, j) in the grid, taking patches of dimensions $[k, k]$.

An optimal implementation of the patch-matching algorithm would go through every other SAI in the scene, and within every SAI it would look at every possible patch in the scene in search of the best candidates. While this method would yield the best results, it is also prohibitively expensive. The number of operations to compare one patch with another is k^2, and each SAI has $\lceil H/k \rceil \cdot \lceil W/k \rceil$ non-overlapping patches. Additionally considering that each reference patch from each SAI needs to be compared with every one of the $(H-k+1)\cdot(W-K+1)$ overlapping patches in all the other SAIs, and that there are $G_H \cdot G_W$ SAIs, this would give a total complexity of

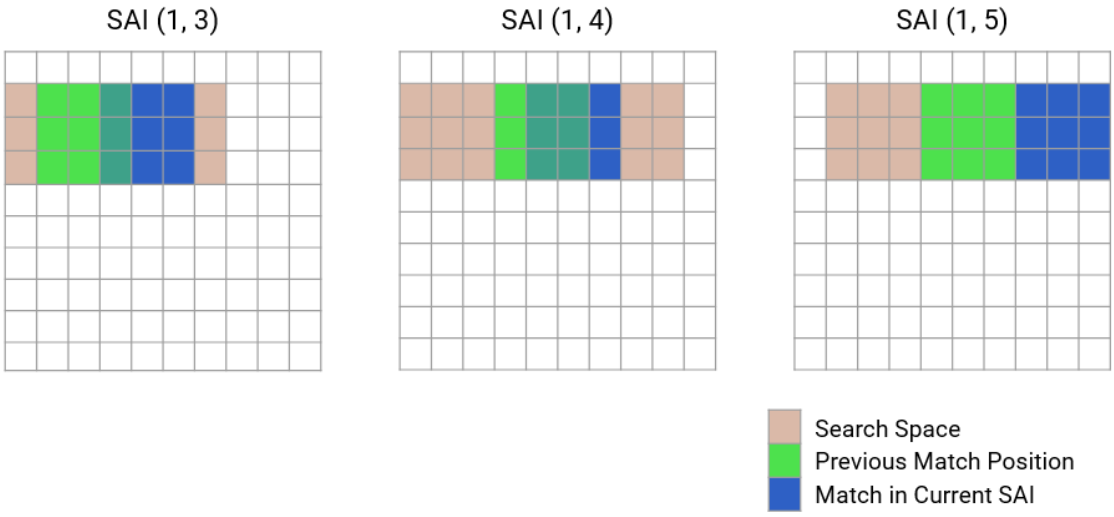
$$\mathcal{O}(H^2 \cdot W^2 \cdot G_H^2 \cdot G_W^2)$$

Considering that it is still necessary to sort the candidates (which takes roughly $\mathcal{O}(H^2 \cdot W^2 \cdot G_H^2 \cdot G_W^2 \cdot k^{-2} \cdot \log N)$ if done with binary insertion), it is evident that this does not scale. So, several heuristics were used to speed up the process, which are visually represented in Figure 3 and explained in detail below.

The first heuristic to consider stems from the fact that there are strong geometrical assumptions that can be taken for the SAI grid. Namely, we can assume that for SAIs in the same row of the grid, there is only horizontal disparity. Likewise, for images in the same column of the grid, we can assume there is only vertical disparity. This means that in these SAIs we only need to search in one direction of the image relative to the current patch, and not in the whole image, as is shown by the horizontal search in Figure 3b, which significantly reduces the computational cost. This is not to say that there would be no viable, or even better, candidates in the remaining



(a) In red are the only SAIs considered when applying patch-matching to the one at the intersection. All the blue ones are ignored.



(b) Representation of the 1D search for the closest match over a row of SAIs.

Figure 3: Visual representation of the heuristics applied to speed up the patch-matching algorithm

images of the grid, but they would be much more costly to find. Figure 3a represents a SAI grid, and in such a grid, when looking for patch matches for the SAI in the center of the cross, we would only look at the red images. This heuristic already simplifies the complexity to

$$\mathcal{O}(G_H \cdot G_W \cdot H \cdot W \cdot (H \cdot G_H + W \cdot G_W))$$

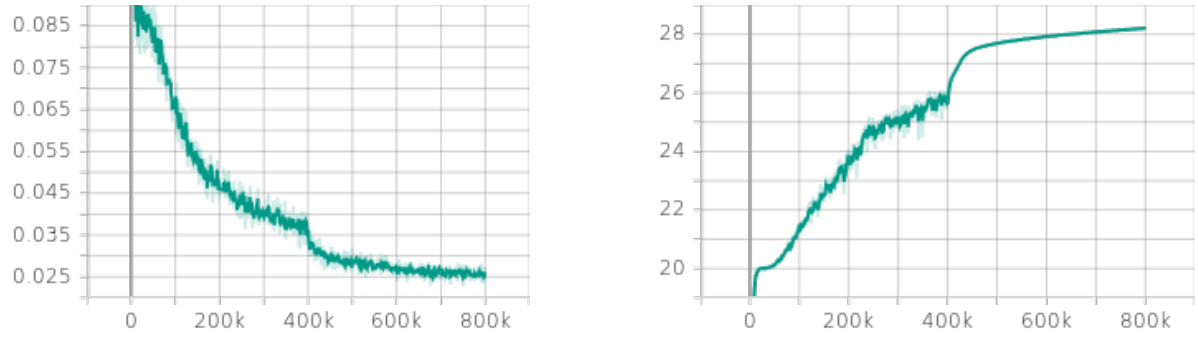
Having narrowed the search to a 1D problem instead of a 2D problem, it is also possible to simplify it further, this time by assuming that the disparity present between neighboring SAIs is not large. This means that it is only necessary to search in the vicinity of the position of the closest match in the previous SAI. We call this the search space, and it is essentially a sliding window, which is represented in Figure 3b by the region in brown. If there are $S \ll H, W$ positions within the search space, then the final complexity for the simplified algorithm is

$$\mathcal{O}(G_H \cdot G_W \cdot H \cdot W \cdot S \cdot (G_H + G_W)) \tag{7}$$

In comparison with the naive implementation's complexity, there is a tremendous gain which becomes more significant as the images/scenes get larger.

3.3 Training Routine

The process of computing the stitch-patched versions is too slow to be done at every step of training. This is mainly because, unlike for images, we can't just take a small patch and run



(a) Training loss (l_1).

(b) Validation PSNR.

Figure 4: Evolution of training and testing metrics over the training routine.

it through the model. Instead, to compute the stitch-patched images we need to consider the entire scene, which becomes very expensive. So, the stitch-patched versions were computed for the entire training dataset beforehand, and then those were used as the input to train the model. Even still, the process was very time-consuming, so each image was reduced to a width of 256px and the height was adjusted so that the height-width ratio was kept constant. These choices allowed us to train the model in a reasonable time frame, but they came with two main consequences.

First, the patches were computed on clean images, and then different levels of additive white gaussian noise were applied to the stitch-patched versions at each step of training. In reality, the patch-matching would be performed on noisy images so, while the choice taken allows faster training, it introduces a difference between the training pipeline and the inference pipeline.

Secondly, by using only small images for training, there is no guarantee that the learned kernels will generalize well for large images. However, this choice was once again necessary to train the model, as even with the small images it still took 1 day to compute all the stitch-patched images for the 492 scenes.

To train the model, we used l_1 loss, as well as a step-based decaying learning rate. We used a batch size of 128 and a training patch of shape $[21, 192, 192]$, 21 channels from the stitch-patched images, and square patches with side 192px. Using 2 GPUs, it took 4 days to train the model (added to the 1 day it took to compute the stitch-patched images).

The model at the start of training contained batch-norm layers after the convolutional layers, but following the original training procedure from DnCNN, these were merged after 400'000 steps had been done. While the original paper doesn't justify that, looking at Figure 4 we can see a big improvement both in training loss and validation PSNR exactly after the layers were merged. Our main hypothesis for this is that while the batch norm layers help speed up training and even boost performance at early stages, later on they should be removed and turn the model into a purely convolutional one.

To train the model for blind denoising, the noise strength was not fixed during training. Instead, for each of the extracted training patches, we selected a random noise strength in the range [0-55], and applied additive white gaussian noise to the patch. This way the model doesn't just learn how to denoise one specific noise strength, but is able to adapt to different noise levels.

4 Benchmark Settings

4.1 Testing Dataset

From the retrieved dataset, six scenes were selected for testing. A SAI from each scene is shown in Figure 5, as well as the names which will be used to refer to them from here on.

These scenes were selected to allow a comparison between the different methods. However,

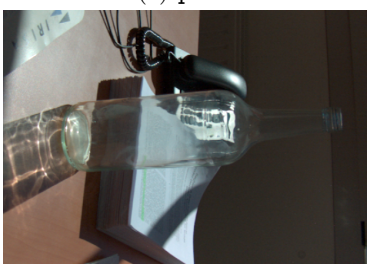
(a) `backlight_1`

(b) `flower`

(c) `pens`

(d) `reflecting_structure_1`

(e) `t_rex`



(f) `translucent`

Figure 5: SAIs taken from each of the testing scenes. The names of each scene are in the subcaptions.

Table 2: Description of the key characteristics for each of the testing scenes. Note that SAI size is given in format $W \times H$ to obey standard image conventions.

Scene Name	Key Characteristics	Scene Size (imgs)	SAI Size (px)
`backlight_1`	Complex backlighting effects	7×7	620×432
`flower`	Large scene	17×17	960×1152
`pens`	Very challenging background	9×9	512×512
`reflecting_structure_1`	Semi-reflective structures	7×7	620×432
`t_rex`	Wide angular spacing	5×5	840×525
`translucent`	Inner reflections and transparency	7×7	619×430

each scene also allows for a subjective assessment of different characteristics, as described in Table 2.

4.2 IQA Metrics

To measure the performance of the denoising algorithm, it is necessary to pick an image quality assessment (IQA) metric. Such a metric would be a function of the original image X and the restored image Y, and it would condense all the similarity information to a single value. Ideally, this metric should mimic the human visual perception of quality, but there is no universally effective metric to satisfy this [52].

So, we decided to use two of the most used IQA metrics: peak signal-to-noise-ration (PSNR) and structural similarity index measure (SSIM) [53]. By using two very well-known metrics, it becomes easier to compare the methods and replicate results.

4.2.1 PSNR

PSNR can be classified as a "mathematically-based metric", following the terminology from [52], as it operates only on the intensity of the distortions. This metric starts by computing the mean squared error (MSE) between the original image X and the restored image Y, given by

$$\text{MSE} = \frac{1}{HW} \sum_{i=1}^{H} \sum_{j=1}^{W} (X[i,j] - Y[i,j])^2 \tag{8}$$

where the images have shape $[H, W]$. Then PSNR metric is

$$\text{PSNR} = 10 \cdot \log \left(\frac{\text{MAX}^2}{\text{MSE}} \right) \tag{9}$$

where MAX is the maximum possible pixel value of the image.

This metric attempts to estimate the absolute difference between the original and the reconstructed images, and so is unbounded.

Note that color images (RGB), first should be converted to grayscale, and then the PSNR should be applied to them. In our experiments, we followed the perception-based conversion to grayscale, using the luminance for grayscale following ITU-R Rec. BT.709-6 [54], computed as

$$Y = 0.2126R + 0.7152G + 0.0722B \tag{10}$$

where R, G, B stand, respectively, for the value of the red, green, and blue channels, and Y is the luminance value used for grayscale.

Given that this metric is easy to compute, we used our own implementation for it.

4.2.2 SSIM

SSIM [53], following the same grouping as before, is a high-level metric, as it measures quality based on the idea that our human visual system is adapted to extract information or structures from the image.

This metric attempts to quantify the difference between the original and the distorted images via a combination of luminance, contrast, and structure comparisons. The comparison is done using local windows, and the final result is the mean of all the window values. For each window, SSIM is computed using

$$\text{SSIM} = \frac{(2\mu_X\mu_Y + C_1)\,(2\sigma_{XY} + C_2)}{\left(\mu_X^2 + \mu_Y^2 + C_1\right)\left(\sigma_X^2 + \sigma_Y^2 + C_2\right)} \tag{11}$$

where μ is the mean intensity for signal X or Y, and σ is the standard deviation for the signals. The constants C_1 and C_2 are defined taking into account the dynamic range of the image.

Unlike PSNR, this metric gives a relative measure of the difference between the two images. So it takes values in the range $[-1, 1]$, where 1 means identical images, 0 means no correlation, and -1 means perfect anti-correlation.

In our experiments, we used the implementation from [55].

4.3 Hardware Specifications

For the experiments, two kinds of hardware were used. For the methods Wavelets, BM3D and LFBM5D, there was no trivial way to switch them to GPU, so they ran on an Intel Xeon CPU E5-2630 with 20 cores. For the DnCNN method, it ran on a Nvidia GeForce RTX 2080 GPU. Finally, the LFDnPatch runs on a hybrid set-up, where the construction of the stitch-patched versions is performed on CPU, and the processing of the enhanced image by the learned model is done on GPU.

(a) Soft additive white gaussian noise ($\sigma = 6$).

(b) Medium additive white gaussian noise ($\sigma = 15$).

(c) Hard additive white gaussian noise ($\sigma = 23$).

(d) Soft realistic noise ($\sigma_{est} = 8$).

(e) Medium realistic noise ($\sigma_{est} = 13$).

(f) Hard realistic noise ($\sigma_{est} = 24$).

Figure 6: Examples of noisy images, for each of the strengths and noise types considered.

Table 3: Aggregated results for additive white gaussian noise and realistic noise. Aggregation done via averaging over Tables 4 and 5 for each method.

(a) Aggregated method results for additive white gaussian noise.

Method	Soft Noise		Medium Noise		Hard Noise	
	$\overline{\text{PSNR}}$	$\overline{\text{SSIM}}$	$\overline{\text{PSNR}}$	$\overline{\text{SSIM}}$	$\overline{\text{PSNR}}$	$\overline{\text{SSIM}}$
Wavelets	35.4	0.904	31.6	0.790	29.8	0.727
BM3D	40.4	0.953	35.6	0.889	33.4	0.845
LFBM5D	**41.0**	**0.957**	**36.6**	**0.904**	**34.2**	**0.862**
DnCNN	39.2	0.929	35.4	0.883	33.4	0.846
LFDnPatch	37.9	0.919	34.3	0.857	32.5	0.818

(b) Aggregated method results for realistic noise.

Method	Soft Noise		Medium Noise		Hard Noise	
	$\overline{\text{PSNR}}$	$\overline{\text{SSIM}}$	$\overline{\text{PSNR}}$	$\overline{\text{SSIM}}$	$\overline{\text{PSNR}}$	$\overline{\text{SSIM}}$
Wavelets	34.1	0.870	32.3	0.815	30.6	0.750
BM3D	38.5	0.933	34.1	0.800	33.4	0.793
LFBM5D	**39.7**	**0.941**	**36.9**	**0.901**	**34.0**	**0.850**
DnCNN	38.0	0.920	35.4	0.880	33.3	0.833
LFDnPatch	36.7	0.913	34.3	0.855	32.5	0.811

5 Experiments

5.1 Quantitative Method Performance Comparison

To perform a quantitative comparison of all the methods, we used scenes `backlight_1`, `pens`, `reflecting_structure_1`, `t_rex`, and `translucent`. For each scene, we applied either additive

white gaussian or realistic noise, with three strength levels, as can be seen in Figure 6. Then we applied all the denoising methods on these scenes and measured the reconstruction quality with PSNR and SSIM. The full results can be found in the appendix, in Tables 4 and 5. It should be noted that BM3D and LFBM5D require an estimate of the noise variance for parameter selection. We also use the noise variance as a strength indicator in our analysis. To estimate the noise variance we used scikit-image's [56] function `estimate_sigma`, which implements the method proposed by [57]. However, this method is designed for additive white gaussian noise, not realistic noise. So we'll make the distinction between using σ for gaussian noise and σ_{est} for realistic noise.

Since the tables in the Appendix A are quite large, Table 3 shows the aggregated results for easier understanding of the key points. It is clear that the best method was the LFBM5D. It outscored all the other methods for all noise strengths and types, both in PSNR and SSIM. This is a strong indication that light field-specific methods are capable of exploiting scene redundancies in different SAIs, and achieving better performance than standard image denoising methods. However, one interesting takeaway is that, looking at the full results in the appendix, LFBM5D was consistently beaten by BM3D or DnCNN in the `t_rex` scene. This matches our expectations (as there is also a large drop in performance for LFDnPatch in this scene) because this scene was selected for testing due to its large angular spacing between neighboring SAIs. So, there are fewer redundancies and more occlusions and disparity effects that make light field-specific methods fail. Light Fields with large angular spacing will probably need their specific methods, as they pose a very extreme problem.

Another observation is that the BM3D method slightly outperforms DnCNN for soft noise, but as the noise intensity increases the DnCNN becomes more competitive and surpasses BM3D. It seems that learning-based methods are capable of adapting better than classical methods to stronger noise levels. Adding this to the fact that these methods tend to be faster (see the following sections), applications with strong noise and strong time constraints may benefit from learning-based methods.

Comparing the results from additive white gaussian noise and realistic, there doesn't seem to be any significant drop in performance. This means that, even though all the methods tested were designed/trained for gaussian noise, they generalize well for other kinds of noise, namely realistic noise.

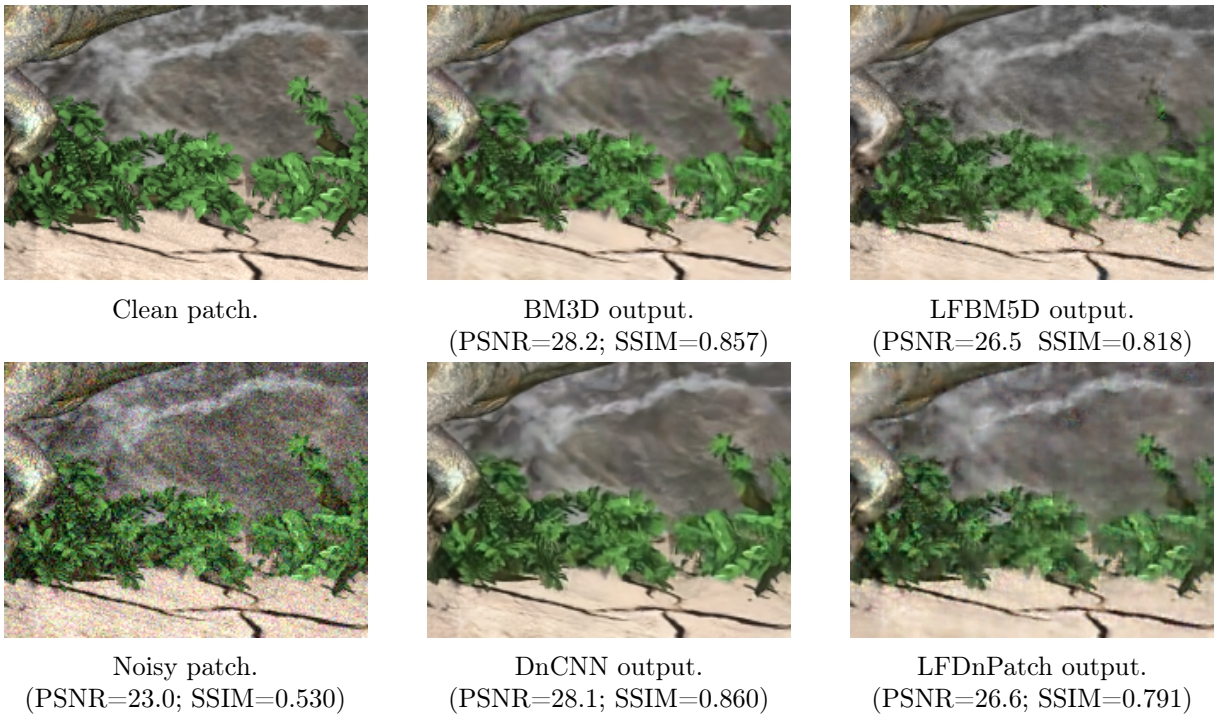
Looking at LFDnPatch, we see that even though it performed slightly worse than DnCNN, it always did better than wavelets, and even surpassed BM3D in some circumstances. The results are promising, and with the possible directions of study explained in Section 6, it seems feasible that this method will reach comparable or even superior performance than DnCNN in the future.

Finally, we can't forget that IQA metrics aren't perfectly correlated with human perception, and are not sufficient for a complete analysis. This is clear, for example, when looking at the aggregated scene results, where the `t_rex` scene performed much worse than the others in terms of PSNR, but this is not observed in terms of SSIM. So, it is necessary to also perform a visual analysis, which can be found in the next section.

5.2 Visual Examples

To provide a visual assessment of the methods, we give several details from the testing scenes. For each of them, we show the clean detail, the noisy detail, and the outputs from the methods BM3D, LFBM5D, DnCNN, and LFDnPatch. The wavelet denoising method was ignored in this part, because it cluttered the images, and it was only used as a rough baseline. These results are better seen in color and zoomed-in.

In Figure 7, we see a detail from the `t_rex` scene. This is a scene with large angular spacing, and it is clear that the light field-specific denoising methods performed much worse than the image-based denoisers. Looking at their behaviors in the fine structures of the foliage, none of them were perfect but the output from LFBM5D and LFDnPatch is much blurrier, even erasing



Clean patch.

BM3D output.
(PSNR=28.2; SSIM=0.857)

LFBM5D output.
(PSNR=26.5 SSIM=0.818)

Noisy patch.
(PSNR=23.0; SSIM=0.530)

DnCNN output.
(PSNR=28.1; SSIM=0.860)

LFDnPatch output.
(PSNR=26.6; SSIM=0.791)

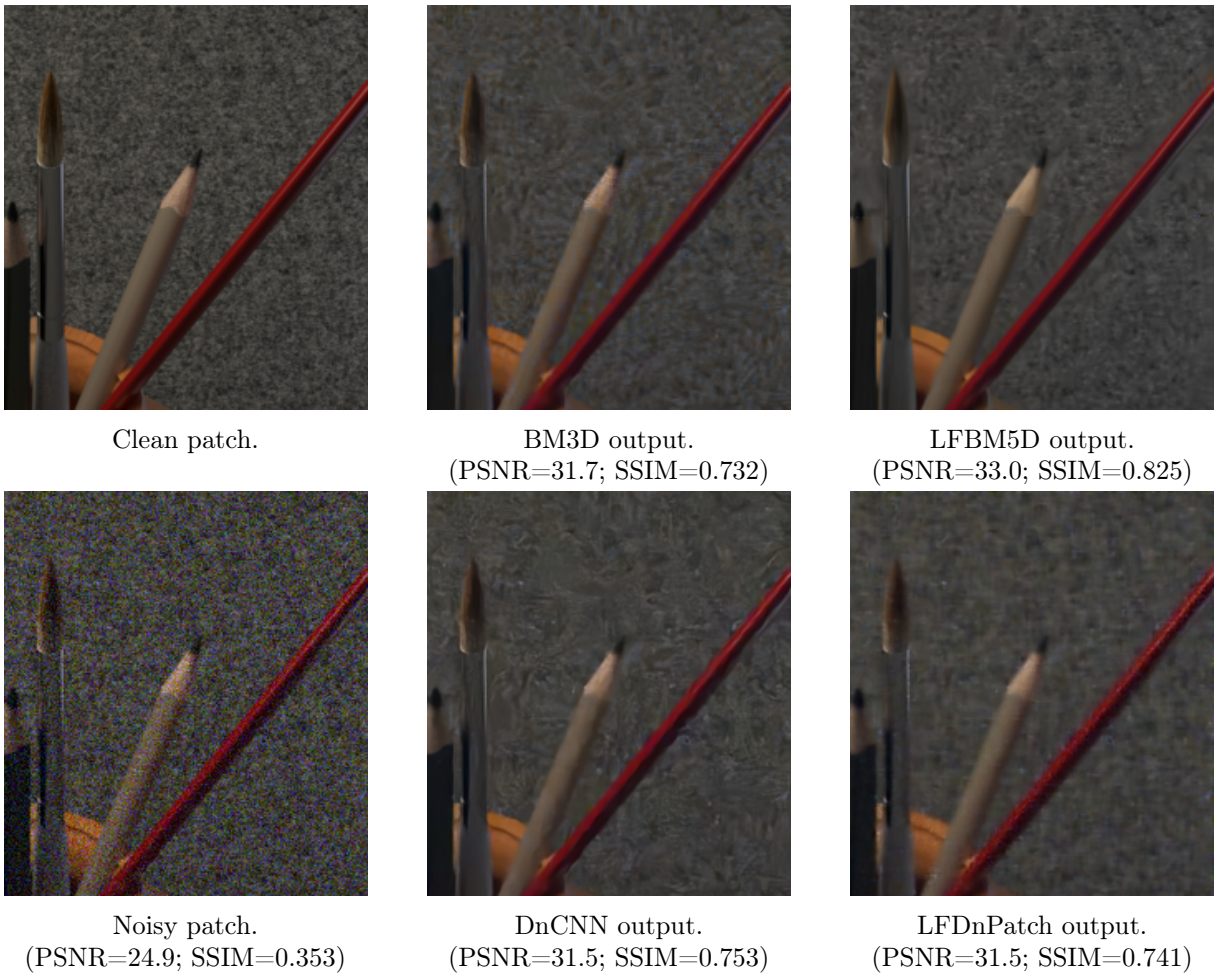
Figure 7: Detail from `t_rex` scene, clean, noisy (gaussian hard noise) and as output of denoising methods.

certain fine details.

In Figure 8, we look at a particularly challenging scene, as the background has a texture very similar to noise. And indeed, all the methods gave a very blurred output for the background, as they were not capable of distinguishing it from the noise. However, the LFBM5D method is the only one that kept the sharpness on the foreground objects, which is most clearly seen in the red pencil. In this scene, it seems that the LFDnPatch method was the worst-performing method because even though it managed to remove the noise, the output is also very blurred. The BM3D and DnCNN, on the other hand, are very similar and its hard to declare a clear winner.

Figure 9 shows how the methods generalize to realistic noise. From the noisy patch, it is clear this image had much more red noise than green or blue. Looking at the BM3D output, it seems to not have removed the red noise. This stems from the fact that this method was designed for additive white gaussian noise, and so expects similar noise strengths along the channels. Comparing it with DnCNN, it seems that learning-based methods may generalize better to different noise statistics, as they were not designed with specific priors in mind, and even though DnCNN was trained with white gaussian noise, it is probably less strict than BM3D. The LFBM5D method, on the other hand, seems to exploit the light field redundancies in a way that allows it to overcome its design priors, giving an image almost identical to the original, from a visual perception sense.

Finally, Figure 10 shows something that is always necessary to keep in mind with image restoration tasks. The original image, often thought of as the "clean image", is simply the version of the image without the added noise. However, there are no guarantees that it is actually without noise. In this case, the output from LFBM5D is much cleaner than the original image, and the other methods seem to give results that are also at least as clean. This raises a problem when comparing the denoised images to the original with quantitative metrics. A very clean image may get a lower score because it is different from the original, even though it is performing better. Additionally, the LFDnPatch model was trained on these images, an noise in the "clean" samples may impact its performance. As such, it may be worthwhile to revise the



Clean patch.

BM3D output.
(PSNR=31.7; SSIM=0.732)

LFBM5D output.
(PSNR=33.0; SSIM=0.825)

Noisy patch.
(PSNR=24.9; SSIM=0.353)

DnCNN output.
(PSNR=31.5; SSIM=0.753)

LFDnPatch output.
(PSNR=31.5; SSIM=0.741)

Figure 8: Detail from `translucent` scene, clean, noisy (realistic hard) and as output of denoising methods.

dataset to remove images that may be too noisy to be disregarded[3].

These results visually confirm that the best method is LFBM5D, except for images with large angular spacing. Nevertheless, all the methods give reasonable results, and the difference in performance between BM3D and DnCNN is, in some cases, quite subjective, and there is no clear-cut winner. As far as LFDnPatch, while it seems to perform slightly worse than the other methods, we believe it is a reasonable first try, and it shows potential.

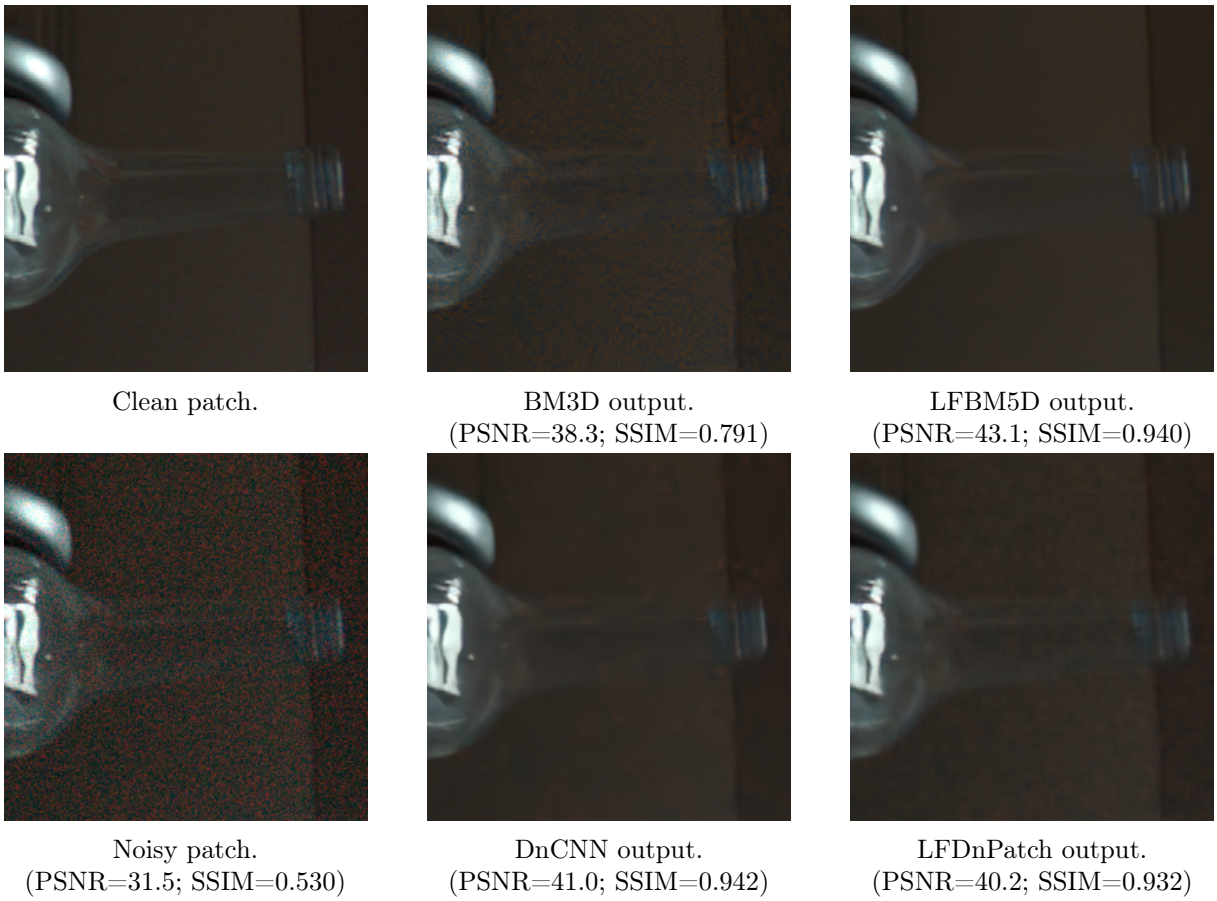
5.3 Scene Size Effect on Run Time

In this section, we study how the scene size (i.e. number of images per scene), affects each methods' run time. For these tests, we used the `flower` scene, with a downscale factor of 0.3. We chose to down-sample it, instead of using the original one, because here we are only interested in studying how the scene size affects the execution time of each method. Using the same, smaller image on all the tests, allows us to run them faster without impacting our study.

The results obtained are in Figure 11. The original `flower` scene had a scene size of (17×17), leading to a total of 289 SAIs. The different-sized scenes were taken from the original by using sub-grids of size $(i \times i), i \in [1, 17]$ starting from the top-left corner of the scene.

Considering that the methods BM3D, Wavelets, and DnCNN are image denoising methods,

[3]We believe that these should be the ones from the Stanford Lytro Light Field Archive [40], as the decoding from ESLF to SAI grid seems to be the most noise inducing conversion.



Clean patch.

BM3D output.
(PSNR=38.3; SSIM=0.791)

LFBM5D output.
(PSNR=43.1; SSIM=0.940)

Noisy patch.
(PSNR=31.5; SSIM=0.530)

DnCNN output.
(PSNR=41.0; SSIM=0.942)

LFDnPatch output.
(PSNR=40.2; SSIM=0.932)

Figure 9: Detail from `translucent` scene, clean, noisy (realistic medium) and as output of denoising methods.

which were extended to light fields by simply looking at one image at a time, we expected to see their run time grow linearly with the scene size. Indeed that is observed, with BM3D being the slowest and Wavelets the fastest. However, it should be pointed out that the DnCNN method is running on GPU, not CPU, which is what gives it its boost over BM3D. The LFBM5D method, on the other hand, is light field-specific, and so its behavior is not linear with scene size. However, it is hard to estimate how it is exactly behaving, probably due to internal heuristics to optimize it. Nevertheless, it seems to be growing in a superlinear way, but a bigger scene would be needed to observe it.

For the proposed method LFDnPatch, we should do a more careful analysis. We can immediately see that it is growing superlinearly as well, but that was to be expected. Considering the complexity expression in Equation 7, and noting that in this experiment only G_H and G_W change, we should see a dependency of the form $\mathcal{O}(G_H \cdot G_W \cdot (G_H + G_W))$. For simplicity, let us take $G \coloneqq G_H \approx G_W$. So, if there are $N \coloneqq G^2$ images in the scene, the dependency should be of the form $\mathcal{O}(N^{3/2})$. To test this, we fit a function of the form

$$aN^b + c \tag{12}$$

with free parameters a, b, c. The result of the fit is shown in Figure 11 as the black dashed line. The a and c parameters are not particularly relevant, as they are only a scale factor and a small vertical shift, respectively. However, the exponential parameter obtained was $b = 1.68 \pm 0.01$. This result is close to the expected 1.5, which suggests the implementation is (quasi) optimal with regard to scene size scaling.

Clean patch.

BM3D output.
(PSNR=34.1; SSIM=0.790)

LFBM5D output.
(PSNR=35.0; SSIM=0.798)

Noisy patch.
(PSNR=23.0; SSIM=0.297)

DnCNN output.
(PSNR=34.0; SSIM=0.777)

LFDnPatch output.
(PSNR=33.5; SSIM=0.774)

Figure 10: Detail from `reflecting_structure_1` scene, clean, noisy (gaussian hard) and as output of denoising methods.

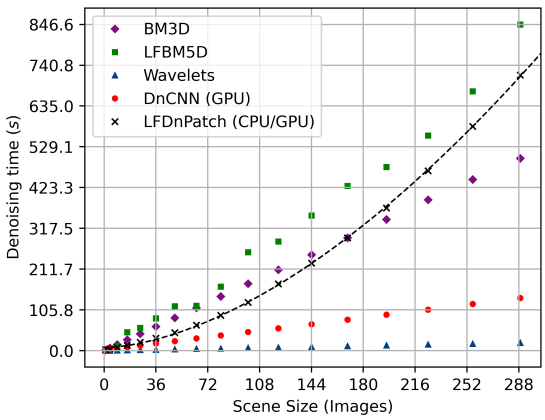


Figure 11: Execution times for the methods as a function of scene size. The dashed line corresponds to the fit of Equation 12 to the LFDnPatch data points.

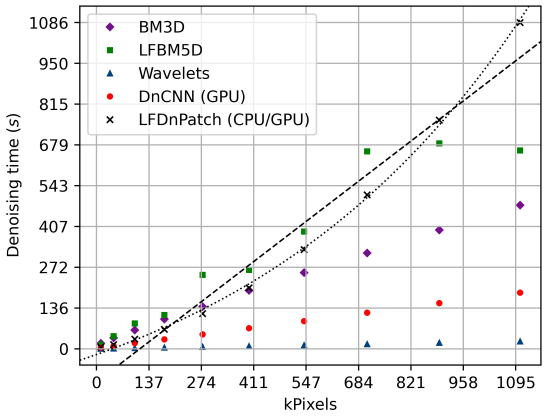


Figure 12: Execution times for the methods as a function of SAI size. The dashed and dotted lines correspond, respectively, to a linear and exponential fit to the LFDnPatch data points.

5.4 SAI Size Effect on Run Time

Similarly to the previous section, we also study how the SAI size (i.e. number of pixels) affects the run time of each method. We used again the `flower` scene, but to speed up the tests we only used a sub-grid of size (6×6). The results are in Figure 12.

Like before, we see that the methods Wavelets, DnCNN and BM3D grow linearly with the

number of pixels. For the Wavelets and DnCNN, this happens by design, while for the BM3D it is due to internal heuristics that speed up the program, making it linear with image size.

The LFBM5D is, like in the previous section, more erratic. So much so that in the tests performed it did not even grow monotonically with image size. However, this is most likely due to this method having a higher time variance because just like BM3D its internal heuristics are designed to keep it linear in image size.

However, the results from LFDnPatch are a lot more interesting. Looking once again at the complexity expression in Equation 7, and considering that now only H and W are variables, the method should behave like $\mathcal{O}(H \cdot W)$. In other words, it should be linear with image size. However, the data in the figure does not match this. Observing a linear fit (dashed line) and an exponential fit (dotted line), the latter fits the data much better. The most likely candidate for this unexpected behavior is inefficient code, like memory allocations or data movements, that would cause this exponential behavior. This warrants a more in-depth analysis of the code, using a profiler, to see what are the bottlenecks slowing it down and fix those.

6 Next Steps

There are several potential directions of study to improve the LFDnPatch method.

Firstly, an architectural change might be needed. Instead of simply increasing the number of channels on the input layer of the DnCNN architecture, we can try to create a 2-stage model. In the first stage, there would be a shallow architecture that would turn the stitch-patched version of the image back into a single RGB image, which would then be launched through the DnCNN. These two could be trained jointly, but the main advantage is that we already have access to the pre-trained weights for the standard DnCNN. This might speed up training and boost performance.

Secondly, we should note that on the original DnCNN work, when switching from grayscale to RGB images, they made the network deeper (from 17 to 20 layers). Given that on the LFDnPatch we are adding channels, the network might benefit from added depth. This could be studied either with the network as it is or with the 2-stage model.

To improve performance, it is also necessary to do a clean-up of the training dataset, and remove any images that may be too noisy. While this most likely will not have a drastic impact, and it is definitely less relevant than the previous points, it might help.

Thirdly, as mentioned in Section 5.4, there is most likely inefficient code in the patch-matching algorithm. This needs to be studied because the method is currently extremely expensive for larger images.

Finally, in this project, we used only the IQA metrics PSNR and SSIM. However, these metrics only measure intra-image consistency between the original and the denoised images. It would be interesting to get a light field quality assessment metric. The PSNR metric only looks at pixel values, which makes it harder to generalize, but SSIM considers 2D windows within the image. Generalizing these to 4D windows might make it possible to get a quality assessment metric specific to light fields, that takes into account the inherent 4D structure of light fields.

7 Conclusion

In this project, we studied several existing image and light field denoising methods and proposed a novel learning-based light field denoising method. We have also acquired a large dataset of light fields, as well as implemented a pipeline for synthetic noise generation. Comparing BM3D, a classical image denoising method, with LFBM5D, its light field denoising generalization, we showed that while there is a clear performance boost by using the latter, it is also slower, as the added dimensions increase its run-time complexity. However, for scenes with large angular

spacings, none of the light field denoising methods were able to outperform their image denoising counterparts.

The novel method we propose is based on a patch-matching algorithm to build stitch-patched versions of every SAI from its neighboring SAIs. This allows us to exploit redundancies in similar patches, and feed this enhanced image to a denoising learning-based model. We show that even though our method performed slightly worse than the original DnCNN, it is capable of exploiting the stitch-patched images and perform light field denoising. There are still several directions of study, and in the future we hope to improve the model architecture by adding depth and studying a 2-stage model and optimize the patch-matching code to bring it closer to its theoretical optimum.

References

[1] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-d transform-domain collaborative filtering," *IEEE Transactions on Image Processing*, vol. 16, no. 8, pp. 2080–2095, 2007. DOI: 10.1109/TIP.2007.901238.

[2] M. Alain and A. Smolic, "Light field denoising by sparse 5d transform domain collaborative filtering," in *2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP)*, 2017, pp. 1–6. DOI: 10.1109/MMSP.2017.8122232.

[3] E. H. Adelson and J. R. Bergen, "The plenoptic function and the elements of early vision," in *Computational Models of Visual Processing*, The MIT Press, 1991.

[4] Y. Wang, J. Yang, Y. Guo, C. Xiao, and W. An, "Selective light field refocusing for camera arrays using bokeh rendering and superresolution," *IEEE Signal Processing Letters*, vol. 26, no. 1, pp. 204–208, Jan. 2019. DOI: 10.1109/lsp.2018.2885213. [Online]. Available: https://doi.org/10.1109%2Flsp.2018.2885213.

[5] Z. Li, L. Song, C. Liu, J. Yuan, and Y. Xu, "Neulf: Efficient novel view synthesis with neural 4d light field," in *33rd Eurographics Symposium on Rendering, EGSR 2022 - Symposium Track, Prague, Czech Republic, 4-6 July 2022*, A. Ghosh and L. Wei, Eds., Eurographics Association, 2022, pp. 59–69. DOI: 10.2312/sr.20221156. [Online]. Available: https://doi.org/10.2312/sr.20221156.

[6] N. Khan, M. H. Kim, and J. Tompkin, "Edge-aware bidirectional diffusion for dense depth estimation from light fields," in *32nd British Machine Vision Conference 2021, BMVC 2021, Online, November 22-25, 2021*, BMVA Press, 2021, p. 47. [Online]. Available: https://www.bmvc2021-virtualconference.com/assets/papers/0637.pdf.

[7] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising," *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142–3155, 2017. DOI: 10.1109/TIP.2017.2662206.

[8] S. R. Alvar and I. V. Bajić, *Practical Noise Simulation for RGB Images*, 2022. DOI: 10.48550/ARXIV.2201.12773. [Online]. Available: https://arxiv.org/abs/2201.12773.

[9] A. Foi, M. Trimeche, V. Katkovnik, and K. Egiazarian, "Practical Poissonian-Gaussian noise modeling and fitting for single-image raw-data," en, *IEEE Trans. Image Process.*, vol. 17, no. 10, pp. 1737–1754, 2008.

[10] A. Abdelhamed, S. Lin, and M. S. Brown, "A High-Quality Denoising Dataset for Smartphone Cameras," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1692–1700. DOI: 10.1109/CVPR.2018.00182.

[11] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, 2005, 60–65 vol. 2. DOI: 10.1109/CVPR.2005.38.

[12] J. Xu, L. Zhang, W. Zuo, D. Zhang, and X. Feng, "Patch group based nonlocal self-similarity prior learning for image denoising," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 244–252. DOI: 10.1109/ICCV.2015.36.

[13] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *IEEE Transactions on Image Processing*, vol. 15, no. 12, pp. 3736–3745, 2006. DOI: 10.1109/TIP.2006.881969.

[14] Z. Zha, X. Liu, X. Huang, *et al.*, "Analyzing the group sparsity based on the rank minimization methods," in *2017 IEEE International Conference on Multimedia and Expo, ICME 2017, Hong Kong, China, July 10-14, 2017*, IEEE Computer Society, 2017, pp. 883–888. DOI: 10.1109/ICME.2017.8019334. [Online]. Available: https://doi.org/10.1109/ICME.2017.8019334.

[15] L. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D: Nonlinear Phenomena*, vol. 60, pp. 259–268, Nov. 1992. DOI: 10.1016/0167-2789(92)90242-F.

[16] Y. Weiss and W. T. Freeman, "What makes a good model of natural images?" In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–8. DOI: 10.1109/CVPR.2007.383092.

[17] S. Gu, Q. Xie, D. Meng, W. Zuo, X. Feng, and L. Zhang, "Weighted nuclear norm minimization and its applications to low level vision," *International Journal of Computer Vision*, vol. 121, Jan. 2017. DOI: 10.1007/s11263-016-0930-5.

[18] U. Schmidt and S. Roth, "Shrinkage fields for effective image restoration," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 2774–2781. DOI: 10.1109/CVPR.2014.349.

[19] Y. Chen, W. Yu, and T. Pock, "On learning optimized reaction diffusion processes for effective image restoration," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, IEEE Computer Society, 2015, pp. 5261–5269. DOI: 10.1109/CVPR.2015.7299163. [Online]. Available: https://doi.org/10.1109/CVPR.2015.7299163.

[20] Y. Chen and T. Pock, "Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1256–1272, 2017. DOI: 10.1109/TPAMI.2016.2596743.

[21] R. Pardasani and U. Shreemali, *Image denoising and super-resolution using residual learning of deep convolutional network*, 2018. DOI: 10.48550/ARXIV.1809.08229. [Online]. Available: https://arxiv.org/abs/1809.08229.

[22] C. Tian, Y. Xu, W. Zuo, B. Du, C.-W. Lin, and D. Zhang, "Designing and training of a dual cnn for image denoising," *Knowledge-Based Systems*, vol. 226, p. 106 949, 2021, ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2021.106949. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705121002124.

[23] P. Xiao, Y. Guo, and P. Zhuang, "Removing stripe noise from infrared cloud images via deep convolutional networks," *IEEE Photonics Journal*, vol. 10, no. 4, pp. 1–14, 2018. DOI: 10.1109/JPHOT.2018.2854303.

[24] S. Tripathi, Z. C. Lipton, and T. Q. Nguyen, *Correction by projection: Denoising images with generative adversarial networks*, 2018. DOI: 10.48550/ARXIV.1803.04477. [Online]. Available: https://arxiv.org/abs/1803.04477.

[25] A. Yu, X. Liu, X. Wei, T. Fu, and D. Liu, "Generative adversarial networks with dense connection for optical coherence tomography images denoising," in *2018 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, 2018, pp. 1–5. DOI: 10.1109/CISP-BMEI.2018.8633086.

[26] C. Tian, L. Fei, W. Zheng, Y. Xu, W. Zuo, and C.-W. Lin, *Deep learning on image denoising: An overview*, 2019. DOI: 10.48550/ARXIV.1912.13171. [Online]. Available: https://arxiv.org/abs/1912.13171.

[27] I. Fodor and C. Kamath, "Denoising through wavelet shrinkage: An empirical study," *J. Electronic Imaging*, vol. 12, pp. 151–160, Jan. 2003. DOI: 10.1117/1.1525793.

[28] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: 10.1038/s41592-019-0686-2.

[29] S. Chang, B. Yu, and M. Vetterli, "Adaptive wavelet thresholding for image denoising and compression," *IEEE Transactions on Image Processing*, vol. 9, no. 9, pp. 1532–1546, 2000. DOI: 10.1109/83.862633.

[30] A. F. Ymir Mäkinen Lucio Azzari, *Wrapper for BM3D denoising*, version 3.0.9, Oct. 2021. [Online]. Available: https://webpages.tuni.fi/foi/GCF-BM3D/.

[31] M. Alain, *C/C++ implementation of the LFBM5D filter for light field denoising and super-resolution*, Apr. 2020. [Online]. Available: https://github.com/V-Sense/LFBM5D.

[32] K. Zhang, *Image Restoration Toolbox (PyTorch). Training and testing codes for DPIR, USRNet, DnCNN, FFDNet, SRMD, DPSR, BSRGAN, SwinIR*, version 1.1, Sep. 2021. [Online]. Available: https://github.com/cszn/KAIR/.

[33] Z. Li, H. Baker, and R. Bajcsy, "Joint image denoising using light-field data," in *2013 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, 2013, pp. 1–6. DOI: 10.1109/ICMEW.2013.6618326.

[34] K. Mitra and A. Veeraraghavan, "Light field denoising, light field superresolution and stereo camera based refocussing using a gmm light field patch prior," in *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2012, pp. 22–28. DOI: 10.1109/CVPRW.2012.6239346.

[35] A. Sepas-Moghaddam, P. L. Correia, and F. Pereira, "Light field denoising: Exploiting the redundancy of an epipolar sequence representation," in *2016 3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON)*, 2016, pp. 1–4. DOI: 10.1109/3DTV.2016.7548963.

[36] J. Chen, J. Hou, and L. Chau, "Light Field Denoising via Anisotropic Parallax Analysis in a CNN Framework," *IEEE Signal Process. Lett.*, vol. 25, no. 9, pp. 1403–1407, 2018. DOI: 10.1109/LSP.2018.2861212.

[37] H. Fan, D. Liu, Z. Xiong, and F. Wu, "Two-stage convolutional neural network for light field super-resolution," in *2017 IEEE International Conference on Image Processing (ICIP)*, 2017, pp. 1167–1171. DOI: 10.1109/ICIP.2017.8296465.

[38] D. G. Dansereau, O. Pizarro, and S. B. Williams, "Decoding, Calibration and Rectification for lenselet-Based Plenoptic Cameras," in *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, IEEE, Jun. 2013.

[39] C. Hahne and A. Aggoun, "PlenoptiCam v1.0: A Light-Field Imaging Framework," *IEEE Transactions on Image Processing*, vol. 30, pp. 6757–6771, 2021. DOI: 10.1109/TIP.2021.3095671.

[40] A. S. Raj, M. Lowney, R. Shah, and G. Wetzstein, *Stanford Lytro Light Field Archive*, 2016. [Online]. Available: http://lightfields.stanford.edu/LF2016.html.

[41] M. Rerábek and T. Ebrahimi, "New Light Field Image Dataset," in *International Workshop on Quality of Multimedia Experience*, 2016. [Online]. Available: https://www.epfl.ch/labs/mmspg/downloads/epfl-light-field-image-dataset/.

[42] J. Shi, X. Jiang, and C. Guillemot, "A Framework for Learning Depth From a Flexible Subset of Dense and Sparse Light Field Views," *IEEE Transactions on Image Processing*, vol. 28, no. 12, pp. 5867–5880, 2019. DOI: 10.1109/TIP.2019.2923323. [Online]. Available: http://clim.inria.fr/Datasets/InriaSynLF/index.html.

[43] K. Honauer, O. Johannsen, D. Kondermann, and B. Goldluecke, "A Dataset and Evaluation Methodology for Depth Estimation on 4D Light Fields," in *Computer Vision – ACCV 2016*, Cham: Springer International Publishing, 2017, pp. 19–34, ISBN: 978-3-319-54187-7. [Online]. Available: https://lightfield-analysis.uni-konstanz.de.

[44] O. Johannsen, K. Honauer, B. Goldluecke, *et al.*, "A taxonomy and evaluation of dense light field depth estimation algorithms," in *Conference on Computer Vision and Pattern Recognition - LF4CV Workshop*, 2017.

[45] X. Jiang, M. Le Pendu, R. A. Farrugia, and C. Guillemot, "Light Field Compression With Homography-Based Low-Rank Approximation," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 7, pp. 1132–1145, 2017. DOI: 10.1109/JSTSP.2017.2747078. [Online]. Available: http://clim.inria.fr/IllumDatasetLF/index.html.

[46] Stanford Computer Graphics Laboratory, *Stanford Light Field Archive*, 2008. [Online]. Available: http://lightfield.stanford.edu/lfs.html.

[47] G. Wetzstein, *Synthetic Light Field Archive*, 2013. [Online]. Available: https://web.media.mit.edu/~gordonw/SyntheticLightFields.

[48] K. Marwah, G. Wetzstein, A. Veeraraghavan, and R. Raskar, "Compressive light field photography," Aug. 2012. DOI: 10.1145/2342896.2342959.

[49] G. Wetzstein, D. Lanman, M. Hirsch, and R. Raskar, "Tensor displays: Compressive light field synthesis using multilayer displays with directional backlighting," *ACM Transactions on Graphics*, vol. 31, pp. 1–11, Jul. 2012. DOI: 10.1145/2185520.2335431.

[50] G. Wetzstein, D. Lanman, W. Heidrich, and R. Raskar, "Layered 3d: Tomographic image synthesis for attenuation-based light field and high dynamic range displays," *ACM Trans. Graph.*, vol. 30, p. 95, Jul. 2011. DOI: 10.1145/1964921.1964990.

[51] T. Gebhard, *Rudimentary Conv4D Layer Implementation for PyTorch*, Jan. 2019. [Online]. Available: https://github.com/timothygebhard/pytorch-conv4d.

[52] M. Pedersen and J. Hardeberg, "Full-reference image quality metrics: Classification and evaluation," *Foundations and Trends in Computer Graphics and Vision*, vol. 7, p. 51, Jan. 2012. DOI: 10.1561/0600000037.

[53] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. DOI: 10.1109/TIP.2003.819861.

[54] Recommendation ITU-R BT.709-6, "Parameter values for the hdtv standards for production and international programme exchange," *International Telecommunication Union*, 2006.

[55] G. Fang, *pytorch-msssim: Fast and differentiable MS-SSIM and SSIM for pytorch.* Version 0.2.1, Aug. 2020. [Online]. Available: https://pypi.org/project/pytorch-msssim/.

[56] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, *et al.*, "Scikit-image: Image processing in Python," *PeerJ*, vol. 2, e453, Jun. 2014, ISSN: 2167-8359. DOI: 10.7717/peerj.453. [Online]. Available: https://doi.org/10.7717/peerj.453.

[57] D. L. Donoho and I. M. Johnstone, "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol. 81, no. 3, pp. 425–455, Sep. 1994, ISSN: 0006-3444. DOI: 10.1093/biomet/81.3.425. eprint: https://academic.oup.com/biomet/article-pdf/81/3/425/26079146/81.3.425.pdf. [Online]. Available: https://doi.org/10.1093/biomet/81.3.425.

Table 4: Results for all methods applied on the selected testing scenes, with additive white gaussian noise.

Method	backlight_1		pens		reflecting_structure_1		t_rex		translucent	
	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
$\sigma = 6$										
Noisy	35.0	0.805	35.0	0.814	35.0	0.725	35.1	0.907	35.0	0.725
Wavelets	36.2	0.898	36.1	0.909	36.8	0.879	29.8	0.916	38.1	0.918
BM3D	40.7	0.951	39.6	0.960	40.1	**0.919**	**38.1**	**0.974**	43.5	0.963
LFBM5D	**42.0**	**0.958**	**41.0**	**0.974**	**40.2**	0.917	37.2	0.970	**44.6**	**0.966**
DnCNN	39.7	0.939	39.2	0.958	38.4	0.856	36.6	0.969	42.2	0.960
LFDnPatch	38.8	0.938	38.2	0.944	37.8	0.868	33.3	0.929	41.6	0.958
$\sigma = 13$										
Noisy	27.0	0.450	27.0	0.433	26.9	0.447	27.0	0.660	26.9	0.340
Wavelets	31.8	0.785	31.8	0.766	32.4	0.735	28.0	0.823	34.0	0.842
BM3D	36.0	0.892	34.3	0.860	36.3	**0.833**	**32.2**	**0.920**	39.4	0.942
LFBM5D	**37.3**	**0.914**	**36.4**	**0.920**	**37.1**	0.830	31.2	0.904	**41.1**	**0.951**
DnCNN	35.8	0.889	34.4	0.871	35.7	0.806	31.9	0.919	39.1	0.942
LFDnPatch	35.2	0.886	33.7	0.834	34.9	0.805	29.6	0.850	38.2	0.933
$\sigma = 20$										
Noisy	23.2	0.288	23.2	0.255	23.0	0.275	23.2	0.492	23.2	0.207
Wavelets	29.8	0.724	30.2	0.683	30.4	0.671	26.8	0.758	31.9	0.801
BM3D	33.7	0.852	32.3	0.786	34.6	0.795	**29.5**	0.870	37.1	0.924
LFBM5D	**34.7**	**0.879**	**33.7**	**0.845**	**35.5**	**0.803**	28.3	0.842	**38.9**	**0.940**
DnCNN	33.7	0.854	32.5	0.800	34.2	0.777	**29.5**	**0.873**	37.2	0.929
LFDnPatch	33.2	0.850	32.4	0.784	33.6	0.772	27.5	0.791	35.9	0.905

A Complete Scene Scores

In this section we include the full results obtained for comparing the methods. Table 4 shows the results from additive gaussian noise, with soft ($\sigma = 6$), medium ($\sigma = 15$) and hard ($\sigma = 23$) noise. Likewise, in Table 5 are the results from realistic noise, with soft ($\sigma_{\text{est}} = 8$), medium ($\sigma_{\text{est}} = 13$) and hard ($\sigma_{\text{est}} = 24$) noise.

In Table 6 we also have the results aggregated by scene. While these are not particularly relevant for the analysis performed, they can be used to see which scenes are harder and easier. We leave this as potential reference for future work.

Table 5: Results for all methods applied on the selected testing scenes, with realistic noise.

Method	backlight_1		pens		reflecting_structure_1		t_rex		translucent	
	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
$\sigma_{\text{est}} = 7$										
Noisy	32.4	0.710	32.6	0.724	32.4	0.724	32.5	0.626	32.4	0.842
Wavelets	34.6	0.862	34.5	0.867	35.4	0.833	29.6	0.894	36.5	0.894
BM3D	38.7	0.929	37.6	0.936	38.6	**0.891**	**35.7**	**0.958**	41.9	0.952
LFBM5D	**40.7**	**0.947**	**39.6**	**0.962**	**39.3**	0.885	34.8	0.949	**44.1**	**0.962**
DnCNN	38.4	0.921	37.4	0.934	37.6	0.837	34.8	0.954	41.7	0.954
LFDnPatch	37.7	0.919	36.3	0.906	36.9	0.840	31.9	0.951	40.9	0.951
$\sigma_{\text{est}} = 11$										
Noisy	31.3	0.597	31.3	0.622	31.2	0.602	31.2	0.727	31.3	0.540
Wavelets	32.9	0.809	32.4	0.797	33.4	0.757	28.5	0.855	34.2	0.859
BM3D	34.4	0.793	33.3	0.792	34.9	0.757	**31.5**	0.877	36.4	0.779
LFBM5D	**37.7**	**0.913**	**37.0**	**0.926**	**37.4**	**0.837**	31.1	0.892	**41.4**	0.938
DnCNN	35.8	0.885	34.3	0.862	35.8	0.803	31.7	**0.912**	39.4	**0.940**
LFDnPatch	35.3	0.882	33.7	0.827	34.9	0.800	29.4	0.836	38.4	0.931
$\sigma_{\text{est}} = 20$										
Noisy	24.9	0.308	25.0	0.310	24.8	0.313	24.9	0.495	25.0	0.243
Wavelets	30.7	0.744	30.9	0.714	31.3	0.692	27.2	0.784	32.9	0.818
BM3D	33.7	0.783	32.5	0.752	34.7	0.746	**29.4**	0.831	36.9	0.852
LFBM5D	**34.3**	**0.866**	**33.6**	**0.832**	**35.5**	**0.799**	27.8	0.819	**38.8**	**0.936**
DnCNN	33.5	0.840	32.4	0.779	34.1	0.768	29.2	**0.854**	37.2	0.923
LFDnPatch	33.2	0.838	32.5	0.778	33.6	0.766	27.4	0.778	35.9	0.894

Table 6: Aggregated results for additive white gaussian noise and realistic noise. Aggregation done via averaging over Tables 4 and 5 for each scene.

(a) Aggregated scene results for additive white gaussian noise.

Scene	Soft Noise		Medium Noise		Hard Noise	
	$\overline{\text{PSNR}}$	$\overline{\text{SSIM}}$	$\overline{\text{PSNR}}$	$\overline{\text{SSIM}}$	$\overline{\text{PSNR}}$	$\overline{\text{SSIM}}$
backlight_1	39.5	0.937	35.2	0.873	33.0	0.832
pens	38.8	0.949	34.1	0.850	32.2	0.800
reflecting_structure_1	38.7	0.888	35.3	0.802	33.7	0.764
t_rex	35.0	0.952	30.6	0.883	28.3	0.827
translucent	42.0	0.953	38.4	0.922	36.2	0.900

(b) Aggregated scene results for realistic noise.

Scene	Soft Noise		Medium Noise		Hard Noise	
	$\overline{\text{PSNR}}$	$\overline{\text{SSIM}}$	$\overline{\text{PSNR}}$	$\overline{\text{SSIM}}$	$\overline{\text{PSNR}}$	$\overline{\text{SSIM}}$
backlight_1	38.0	0.916	35.2	0.856	33.1	0.814
pens	37.1	0.921	34.1	0.841	32.4	0.771
reflecting_structure_1	37.6	0.857	35.3	0.791	33.8	0.754
t_rex	33.4	0.941	30.4	0.874	28.2	0.813
translucent	41.0	0.943	38.0	0.889	36.3	0.885