École Polytechnique Fédérale de Lausanne

Master Thesis (Spring 2018)

Larger, Better, and Faster Denoising with Deep Image Prior

August 16th, 2018

Author
Semion Sidorenko
semion.sidorenko@epfl.ch

Supervisors
Christoph Böcklin, Ph.D.
89Grad GmbH
christoph.boecklin@89grad.ch

Prof. Pascal Frossard
LTS4 — EPFL
pascal.frossard@epfl.ch

Abstract

Focusing on the task of color image denoising, this work improves Deep Image Prior (DIP) in terms of denoising performance, speed, and scalability. DIP is an image-restoration method based on Convolutional Neural Networks with the particularity that it does not use training data. First, we analyze the limitations of DIP in the denoising task and we improve its denoising ability by developing a task-specific regularization method, substantially improving on the baseline and reducing the gap towards benchmarks. Then, we propose an adaptation of the Progressively Grown Generative Adversarial Networks to DIP, yielding a 1.6x speed-up over the original implementation and improving denoising performance as well. Last, we propose a tiling method so that the algorithm can be scaled to arbitrarily large images.

Contents

1 Introduction

Real-world measurements are often corrupted with noise, and images are no exception. While it is often possible to decrease the amount of noise by influencing some factors at acquisition time such as sensor size, sensor temperature, exposition duration, ISO level, etc., it is not always practical, and sometimes the need to remove noise from an image might appear after acquisition is done. To solve this problem, it is often desired to denoise the image as a post-processing step.

Although the Additive White Gaussian Noise (AWGN) model is far from accurately representing all the noise components from real-world digital camera sensors, we will only focus on this model in this report as it serves as a sufficiently simple theoretical model to explore the problem. It has also been widely studied, therefore a number of baselines are available to compare results.

Denoising consists in estimating a clean image from a noisy image, using either assumptions about the image itself such as smoothness and self-silimarity, or using external information, such as information learned (or accessed at run-time) from an external database of images. We will refer to these two approaches as resp. "learning-free" and "learning-based" denoising methods.

While the state of the art in image denoising has been mostly held by learning-free patch-based methods such as block-matching and 3D filtering (BM3D) [1], some recent advances have been made with learning-based approaches using multi-layer perceptrons (MLPs) [2] or convolutional neural networks (CNNs) [3]. In this report, we explore and improve the Deep Image Prior (DIP) architecture developed in [4]. DIP is a learning-free method which uses a convolutional neural network, typically used in learning-based methods. Instead of training the network on external data, the denoising process only uses the noisy image itself and the prior imposed by the structure of the network.

In this report, we first review different state-of-the-art approaches to image denoising and the Deep Image Prior architecture. Then, we introduce our three improvements to Deep Image Prior:

1. We increase the denoising performance by developing a regularization technique which adaptively stops the algorithm when optimal denoising is achieved.

2. We decrease the computation time by around 60% by implementing a progressively grown network, a method inspired from [5].

3. We develop and test a implementation which is able to work with arbitrarily large images with a fixed memory budget, by tiling smaller denoised patches and using the regularization technique mentioned above to ensure consistency across patches.

As we base our work on the original DIP implementation, we develop our implementation with the same language and framework, namely Python 3.6 and PyTorch v0.4. All computations are run on a computer with an 8-core 3GHz CPU, 32G of RAM, and a Nvidia 1080 GTX Ti GPU. The code is available at `https://github.com/ssidorenko/pg_dip`.

2 Better: Improving the Denoising Performance of Deep Image Prior

In this chapter, we introduce the natural image denoising problem and review various approaches to image denoising. Then, we explain in details the Deep Image Prior method, an approach introduced in [4], which we propose to improve using the concept of method noise, an approach created by the authors of the popular denoising algorithm Non-Local Means [6]. Using this technique, we develop two iterations of a regularization algorithm for Deep Image Prior, substantially improving its denoising results.

2.1 Background

2.1.1 The natural image denoising problem

The conventional formulation of the denoising problem is defined as follows: given a clean image X and 0-mean Gaussian noise with known variance $N = \mathcal{N}(0, \sigma)$, we get a corrupted image Y such that

$$Y = X + N \tag{2.1}$$

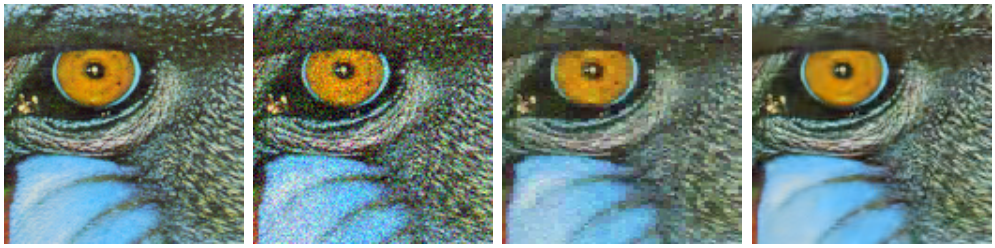
The goal is to estimate X, given only Y and σ. This is known as an ill-posed inverse problem, as there can be infinitely many combination of X and N giving the same Y. This task would not be solvable if each pixel of X were independent and sampled from a completely random distribution. But as X is sampled from the distribution of natural images, this restricts the number of possibilities, making the problem tractable in theory. There are different approaches to denoising, each one making different assumptions about the distribution of X. In this chapter, we will review some common approaches and the Deep Image Prior method, before introducing our improvements to Deep Image Prior.

2.1.2 Review of existing denoising methods

Block-Matching and 3D Filtering

BM3D [1], also known as CBM3D when applied to color images, has been the state of the art for a while, and is still used as a widely accepted benchmark for color and greyscale image denoising. BM3D works by stacking similar patches within a given neighborhood window into a 3D block, applying a 3D transform to the block, applying collaborative filtering, and transforming the block back to 2D. Blocks are then aggregated to form a basic estimate image, and those steps are repeated again but using the basic estimate and the noisy image simultaenously to finally obtain the denoised image.

The main assumption made by BM3D is that natural images have a sparse representation in transform domain. Instead of enforcing sparsity in 2D transforms such as in wavelet thresholding methods [7] which are prone to either lose texture information or let noise through as seen in figure 2.1, BM3D instead enforce sparsity by jointly filtering similar 2D fragments, which allows an easier separation of the noise coefficients and signal coefficients.



(a) Ground truth (b) Noisy image (c) Wavelet-based (d) CBM3D

Figure 2.1

As shown in a study on the theoretical limits of denoising for natural images [8], BM3D is already close to optimality. In this study, they derive a lower bound on the maximum PSNR achievable by a denoising algorithm with a fixed support size and a generic image prior. While this bound is only valid for a given support size, their results show that increasing the support size brings diminishing returns. Therefore, we can consider BM3D as a good benchmark, even when working with larger window sizes. However, this only applies to algorithms which use a generic image prior, consequently it does not apply to learning-based algorithms.

Sparse and Redundant Representations Over Learned Dictionaries

The method introduced in [9] use the K-Singular Value Decomposition (K-SVD) algorithm [10] to build an overcomplete dictionary of patches, which is then used in a sparse linear combination. Their research evaluate the performance of their method using several types of dictionary: an overcomplete Discrete Cosine Transform (DCT) basis, a global dictionary trained on a corpus of image patches, and an adaptive dictionary trained on the noisy image itself.

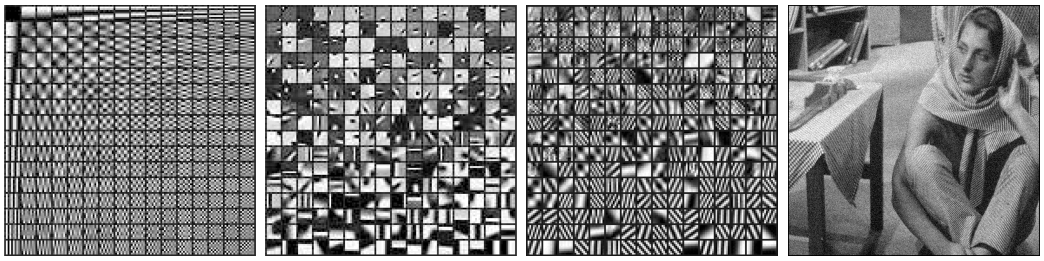


Figure 2.2: From left to right: overcomplete DCT, global dictionary, adaptive dictionary, image used for training the adaptive dictionary.

While the three types of dictionaries work reasonably well, the best results were obtained with the adaptive dictionary even though it was only trained on a single noisy image. As we can see in figure 2.2, the adaptive dictionary is able to learn clean patterns from repetitive noisy samples. This finding is relevant to our work in the sense that it shows that, in their model at least, the information contained in the noisy image is largely enough to denoise it, and that this works better than using information from a large external database. Whether this applies to other models such as CNNs-based models is yet to prove.

Neural networks-based methods

Recent denoising methods based on neural networks such as the MLP-based Stacked Sparse Denoising Auto-encoders (SSDA) [2] and the CNN Denoiser [3] have been able to leverage the representational power of neural networks coupled with large image databases to beat the denoising benchmarks, improving on (C)BM3D by a couple tenth of decibels. Although those methods seem to yield slightly better results than the benchmarks, they are computationally-intensive, requiring for example one month of training for the MLP-based model and three days for the CNN-based model. Additionally, those models must be trained separately for each different noise level.

While neural networks based models gave only slightly higher results for the denoising task than current benchmarks, they have been much more successful for other image restoration tasks such as single image super resolution. For this task, SRResNet and SRGAN [11] propose two approaches, one deep residual network which use an Mean Square Error (MSE) loss, and one generative adversarial network which use a loss function based on VGG16 features as in [12]. Although the MSE-based implementation beat the benchmark in terms of PSNR, the VGG-based gets a higher mean opinion score. This suggests that although the MSE is a good loss function for minimizing pixel-to-pixel differences, using a perceptual loss function can be more efficient for increasing the percepted image quality.

2.1.3 Deep Image Prior

Introduced in [4], Deep Image Prior (DIP) is a CNN-based model which can be applied to various image-restoration tasks such as denoising, super-resolution, and inpainting. In this section, we will present and explain the theoretical concepts introduced in their research.

In this model, the denoising problem exposed with equation 2.1 is reformulated as the problem of finding a noise-free estimate $\hat{Y}$

$$\hat{Y} = \underset{X}{\operatorname{argmin}}\, MSE(Y; X), \quad X \in \mathfrak{N} \tag{2.2}$$

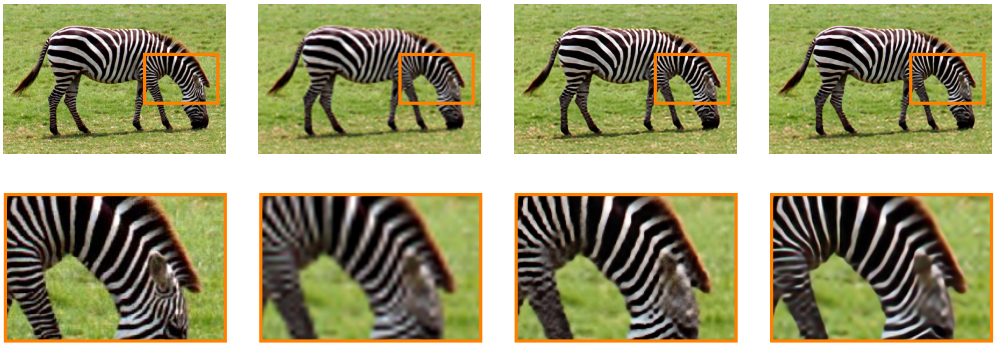
where $\mathfrak{N}$ is defined as the set of natural images. This means that the estimate $\hat{Y}$ should be as close as possible to the noisy image Y, with the restriction that $\hat{Y}$ must be in the set of natural images. However, as the explicit distribution of natural images is unkown, $\hat{Y}$ is not directly computable. The challenge is to find an appropriate prior to approximately model $\mathfrak{N}$.

While CNNs usually rely on training with large external databases for learning this prior, the DIP architecture assumes that the network structure itself, without any particular initialization, serves as a good prior for image restoration tasks such as denoising, inpainting or super-resolution. As seen in figure 2.3, the DIP method works well for the super-resolution task, doing better than learning-free methods but not as well as the SRResNet algorithm.

In the DIP method, the prior is defined as such: X is assumed to be generated by a convolutional network f_θ given a fixed code vector z. Equation 2.2 can be reformulated as follows:

$$\hat{Y} = f_{\hat{\theta}}(z), \quad \hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, MSE(Y; f_\theta(z)) \tag{2.3}$$

The objective is to estimate $\hat{\theta}$ so that $\hat{Y} = f_{\hat{\theta}}(z)$ approaches Y while avoiding reproduction of the noise part of the image. The goal is then to



(a) Ground truth (b) Bicubic (c) Deep Prior (d) SRResNet

Figure 2.3: Super-resolution of an image. The SRResNet method is trained on a large database of images, while Deep Image Prior and Bicubic algorithms are learning-free methods.

define the architecture of the network f and an optimization method so that $\hat{Y}$ easily converge toward natural images but resists noise. The main findings from the DIP research are that defining f as an hourglass shaped "U-Net" network, and defining$z \in \mathbb{R}^{C \times H \times W}$ as a fixed code vector sampled from an uniform distribution, yields this noise-resistant property, while easily converging towards natural images. While this has yielded impressive results for tasks such as super-resolution or inpainting, the results for the denoising tasks are still lagging behind the benchmark. According to our diagnosis, the main reason for this seems to be that the optimization algorithm can overfit the noise when it runs for too many iterations, as seen in figure 2.4.

(a) Ground truth (b) Noisy image (c) Denoised with visible overfit

Figure 2.4

It is important to note that although DIP is a kind of generative network, and although in chapter 3 we apply a technique originally meant for Genera-

tive Adversarial Network (GANs), DIP is not a GAN. As shown in figure 2.5, GANs for image generation usually have 2 networks, one generator and one discriminator competing against each other. The generator G is fed a random code vector z, generally 1-dimensional, and outputs a generated image $\tilde{Y}$. The discriminator D is then fed either an actual item X from the training data, or a generated one $\tilde{Y}$ and tries to guess if the item has been generated by G or not. The result is then backpropagated through D, and also through G if the item was generated. This way, the generator and the discriminator jointly improve until a Nash equilibrium is reached. While this architecture does not readily lend itself to image restoration tasks, it has been used for super-resolution in [11]. In this implementation, the low-resolution image is used as the input to the generator instead of z, the generator super-resolves the image and the discriminator must then guess if its input is an original high-resolution image or if it was super-resolved.

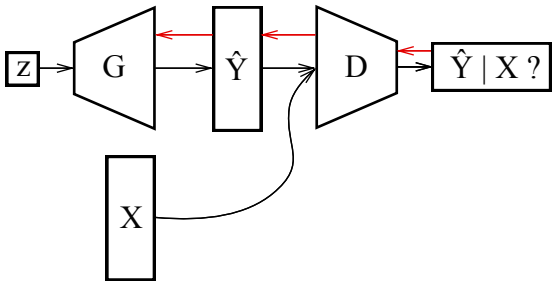


Figure 2.5: Architecture of a GAN. G is the generator, D is the discriminator. Backpropagation is shown in red.

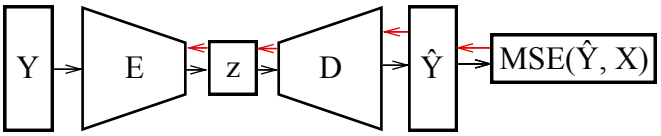


Figure 2.6: Architecture of an autoencoder. E is the encoder, D is the decoder.

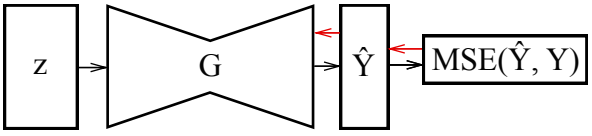


Figure 2.7: Architecture of Deep Image Prior.

DIP can also be mistaken as a denoising autoencoder. A denoising autoencoder works by first reducing the dimensionality of its noisy input Y to

a code vector z using an encoder network E, and then decoding z using a decoder network D to get a reconstructed version $\tilde{Y}$ of the input. During training, the MSE between the reconstructed input $\tilde{Y}$ and the clean version of the input X is computed and backpropagated through the network.

In the case of DIP, there is no discriminator or decoder network, and the vector z is not 1-dimensional but is actually a tensor of the same spatial size as the noisy image, and with an arbitrary depth, fixed to 32 in the case of DIP. This can be misleading in the sense that the actual "input" to the denoising network is not the noisy image but a fixed random vector. The noisy image Y is only used for backpropagation, by backpropagating through the MSE between the network prediction $\hat{Y}$ and Y. Additionally, the structure of the network itself can be misleading. CNNs for visual applications such as image recognition are usually arranged into a funnel shape, transforming spatially large but shallow data, starting from the image itself, into successively smaller but deeper tensors. While this is the structure used in the "input" part of Deep Image Prior, it shouldn't be thought as some kind of "classifier" network, the whole network is actually just a generator.

Additionally, the training procedure of DIP is different from most neural networks. Instead of feeding a large set of training data to the network, only one image is used: the image to denoise. This means that the "training" process is the denoising process. The training/denoising process is started by instantiating a new network with randomly initialized weights and settings z to an uniform random tensor. At each iteration, the network is fed z and outputs a denoised estimate $\hat{Y}$, and the MSE between this denoised estimate and the noisy image is computed. The MSE is then used for updating the weights by backpropagation. After each iteration, $\hat{Y}$ gets closer to Y and at the end of the training procedure, $\hat{Y}$ is kept as the denoised image.

2.1.4 Method noise

An essential building block for the technique developed later in this chapter, the method noise is an approach developed by the authors of the Non-Local Means denoising algorithm [6]. Originally, the method noise was developed as a tool for diagnosing denoising techniques, allowing to visualize which part of the noise was actually removed from an image, and which part of the signal was mistakenly denoised. We recall the definition of the method noise and one of its property:

Definition 2.1 (Method Noise)**.** Let Y be a (not necessarily noisy) image and D_h a denoising operator depending on h. Then we define the method

noise of Y as the image difference

$$N_m(D_h, Y) = Y - D_h(Y)$$

From this definition, we can derive the following theorem:

Theorem 2.1 (Method Noise of a perfect denoising operator)**.** When D_h is a perfect denoising operator and X is the original clean image

$$D_h(Y) = X$$

Plugging in equation 2.1 we get

$$N_m(D_h, Y) = N$$

where N is the Gaussian Noise originally used to corrupt X.

This means that ideally, when a image is corrupted with AWGN, its method noise with a perfect denoising operator should be as close as possible to a tensor sampled from a 0-centered Gaussian distribution with the same standard deviation σ as used during corruption process. See figure 2.10 and 2.11 for visualizations of the method noise of a bad and a good denoising operator.

2.2 Method

In this section, we explain the overfitting issue of the original Deep Image Prior algorithm, and we propose a new regularization method using the method noise.

2.2.1 Deep Image Prior and Regularization

The overfitting issue seen in the last section comes from the fact that if f has enough capacity, it can in theory perfectly minimize $MSE(Y; f_\theta(z))$, giving $\hat{Y} = Y$ which means that the denoiser has perfectly reproduced the noisy image, including the noise itself. This is indeed not the desired behavior, what we want instead is to approach Y while resisting noise. As in practice, f has more than enough capacity to overfit, we must find a way to avoid this overfitting.

Classical regularization techniques do not translate well to the DIP architecture. The reason is that while most of neural-networks based models train on a large set of training data, with the aim of performing well on a

large set of test data, DIP uses only one image, used for training and test as well. This means that most of the common regularization techniques, which aim at improving the generalization ability of a model so that it can perform well on new, unseen images, do not apply to DIP. In DIP, the training process is the actual denoising process, therefore for each different image to denoise, the network must be retrained from scratch on it. For this reason, techniques such as regularization through data augmentation which feeds rotated, flipped, cropped versions of an image to the model during training do not work here.

2.2.2 Using the method noise for early stopping

A widely used regularization approach is the early stopping technique [13], which works by continually checking the validation score during training, and stopping training when the validation score meets certain conditions so that the network can not overfit the training set by training further. However, this can not be used as-is as there is no validation set in DIP. Instead, we propose to use the method noise as an indicator for early stopping. As shown in figures 2.9 and 2.10, when a denoising operator underfits or overfits, correlated patterns appear in the method noise which implies that either the denoising operation has ignored some components of the clean signal, or that the operation has reproduced components from the corrupting noise.

We observed that during optimization, the method noise was monotonically decreasing as seen in figure 2.8. Additionnally, we know by theorem 2.1 that the method noise of a perfect denoising operator should be equal to the corrupting noise $N = \mathcal{N}(0, \sigma)$. While we do not know N, we know its standard deviation σ. This means we can expect the standard deviation of the method noise of a perfect denoising operator to be exactly equal to the standard deviation of N. Therefore, we can use $\sigma_{N_m} \leq \sigma$ as a criterion to stop optimization.

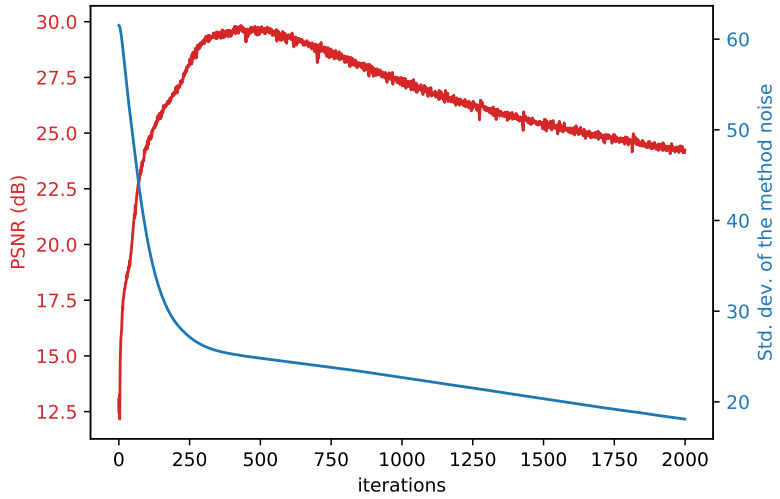


Figure 2.8: Evolution of the PSNR and the method noise during optimization, for the image house.png. The peak of the PSNR curve correspond to the maximum PSNR achievable by the model. The decrease following the peak shows the overfitting.

(a) Noisy image (b) Denoised image (c) Method noise

Figure 2.9: Visualization of the method noise of an overfitting denoising operator.

(a) Noisy image (b) Denoised image (c) Method noise

Figure 2.10: Visualization of the method noise of an underfitting denoising operator. We can see that finer details of the noisy image are missing in the denoised image, and are therefore visible in the method noise.

(a) Noisy image (b) Denoised image (c) Method noise

Figure 2.11: Visualization of the method noise of a good denoising operator. The method noise look almost like Gaussian noise with very slight artifacts.

2.2.3 Static Method Noise Stopping

The criterion defined above assumes that the method noise of a perfect denoising operator is perfectly equal to the corrupting noise. However, in practice the method noise is a truncated version of the corrupting noise as the values of the corrupted image are limited to the $[0.0; 1.0]$ range after corruption because digital images have a fixed dynamic range. Taking this into account, equation 2.1 can be rewritten as follows

$$Y = min(1.0, max(0.0, X + N)) \tag{2.4}$$

For this reason, we cannot expect σ_{N_m} to be equal to σ anymore. The expected value of σ_{N_m} will depend not only on σ but also on the image being denoised. For example, if the image is very bright or very black, it means its pixels are close to the bounds 0.0 and 1.0, which means that after corruption with AWGN, their value are more likely to be truncated, thus reducing σ_{N_m}.

Therefore, we first try to estimate the average value of σ_{N_m} as a function of σ, over a large set of natural images. This is done by iterating over images in the BSDS300 dataset [14], and for each image X_i in the dataset, we compute a corrupted image $Y_i = min(1.0, max(X_i + \mathcal{N}(0, \sigma))$ and calculate the standard deviation $\sigma_{N_{mi}}$ of the computed method noise $N_{mi} = Y_i - X_i$. We then compute $\bar{\sigma}_{N_m}$, by averaging over all $\sigma_{N_{mi}}$. Those steps are repeated for each for each $\sigma \in \{5, 10, ..., 100\}$. As seen in figure 2.12, as σ grows, the computed standard deviation of the method noise strays further. This is explained by the fact that in equation 2.4, when N is very small the equation is close to $Y = X + N$, but as N grows bigger, more of it is truncated.

Using the averages computed above, we now define a simple early stopping method called Static Method Noise Stopping (SMNS). This method works as such: For a noisy image Y corrupted with noise $N = \mathcal{N}(0, \sigma)$ where σ is known, define the threshold T to $\bar{\sigma}_{N_m}$ which is the average method noise for a given σ. For example, for an image corrupted with $N = \mathcal{N}(0, \sigma = 25)$, we get $T = 24.45$ as computed above. Then, at each iteration, compute the standard deviation of the current method noise σ_{N_m}. When it reaches T, stop iteration.

Predicting the method noise

While using the per-σ average expected method noise computed above works reasonably well as a threshold for most images, it sometimes under- or overestimate T for very bright or very dark images. For this reason, we develop a function $P(Y, \sigma)$ which can predict the standard deviation of the method

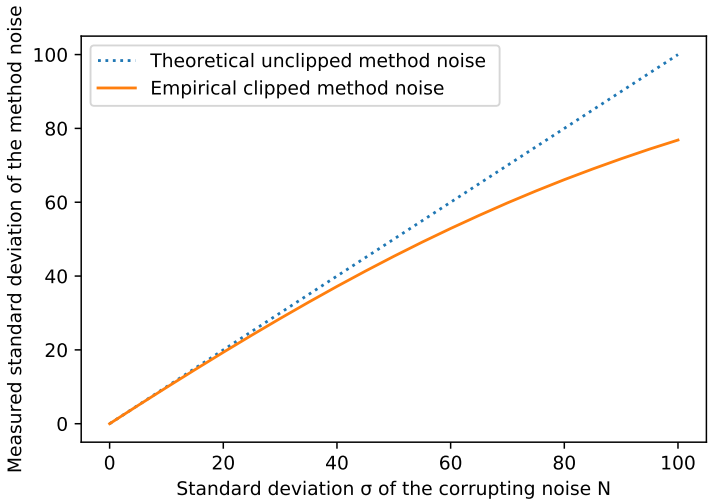


Figure 2.12: $\bar{\sigma}_{N_m}$ VS. σ

noise, given a noisy image Y and the standard deviation σ of the AWGN, prior to truncation.

We initially tried to use as-is a method defined in some research [15] by the same authors as BM3D, which covers the more complex problem of estimating the variance of the noise and denoising images corrupted with Gaussian-Poissonian noise of unknown variance. In this paper, the authors derive multiple relations between the variance and mean of a clipped Gaussian variable, among which equation (46) seems to propose a relation between the standard deviation of a clipped noisy variable and its unclipped standard deviation. This equation can be formulated as such:

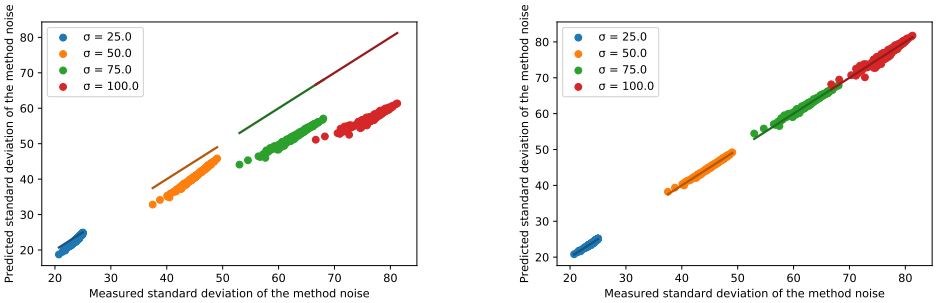
$$P_A(Y,\sigma) = \sigma \mathcal{S}_e(\frac{Y}{\sigma})\mathcal{S}_e(\frac{1-Y}{\sigma}) \tag{2.5}$$

The function $\mathcal{S}_e$, defined in the appendix, gives the standard deviation of a clipped normal variable as a function of its expectation. As seen in figure 2.13, this leads to approximately good results for $\sigma = 25$ but it becomes more biased as σ increases. However, while P_a fails to correctly predict σ_{N_m} for $\sigma > 25$, it still outputs a value seemingly linearly or polynomially correlated to σ_{N_m} as seen in figure 2.13 (a), with the coefficient of this correlation depending on σ as well. As we were unable to find the cause of this error and unable to analytically derive a correct version of P_A, we instead fitted a linear regression with two features, $P_A(Y,\sigma)$ and σ, each augmented with some degree 3 polynomial features and kept only the significant features, to obtain $P(Y,\sigma)$. As seen in figure 2.13 (b), P gives a very good predictor of

the expected method noise standard deviation.

$$\begin{aligned} .P(Y,\sigma) &= aP_A(Y,\sigma) + b\sigma + c\sigma^2 \\ +dP_A(Y,\sigma)\sigma &+ eP_A(Y,\sigma)^2\sigma + fP_A(Y,\sigma)\sigma^2 \\ a = 0.56702133\ b &= 0.42986288\ c = -1.65585067 \\ d = 1.67353735\ e &= -4.56733532\ f = 5.84740478 \end{aligned} \tag{2.6}$$

Using this equation, we can now define the Adaptive Method Noise Stopping (AMNS) technique: For a noisy image Y corrupted with noise $N = \mathcal{N}(0,\sigma)$ where σ is known, define the threshold T to $P(Y,\sigma)$. For example, for the image kodim02.png corrupted with $N = \mathcal{N}(0,\sigma = 25)$, we get $T = P(\text{kodim02.png}, 25) = 23.71$, which is lower than the average expected method noise computed above, as kodim02.png is mostly very bright for red pixels and mostly dark for green and blue ones. For a more even image such as kodim03.png, we get $T = 24.50$, much closer to the average method noise. Once T is computed, start optimization and at each iteration, compute the standard deviation of the current method noise σ_{N_m}. When it reaches T, stop iteration.



(a) P_A (b) P

Figure 2.13: Predicting σ_{N_m} with P_A and P. The diagonal lines represent the ground truth. Fitting

2.2.4 Implementation

To implement SMNS and AMNS, we take the original DIP implementation and add the following modifications:

- We compute the target method noise standard deviation T using $P(Y,\sigma)$ implemented as per equation 2.6 for AMNS, or set $T = 24.45$ for SMNS.
- We remove the fixed iteration limit of the original implementation

- At each iteration we compute the current method noise and stop iteration if T is reached.

This implementation can be found in the file `denoise_mn.py` in the source code repository.

2.3 Results

As in the original DIP method, the results are obtained by running the algorithm on the same noisy image twice, and averaging the results. Additionally, the output of each run is computed as an exponential average of the output of the previous iterations, with $\gamma = 0.99$. We first compute the average PSNR over the CBM3D dataset, with $\sigma = 25$. This leads to an average PSNR of 31.02dB, a 0.38dB improvement over the original results. To understand how much room there is for improving this score, we run the algorithm for a fixed, large number of iterations, and keep the image with the highest PSNR by measuring at each iteration the output of the algorithm against the ground truth, obtaining an average PSNR of 31.13dB. While a real denoising algorithm can obviously not access the ground truth at runtime, this gives us an upper bound on the achievable PSNR with this model. We then compute the average PSNR of DIP+AMNS, which yields an average PSNR of 31.09dB, almost reaching the upper bound.

	CBM3D	DIP	DIP+SMNS	DIP+AMNS	Estimated method noise
house.png	33.03	30.90	32.59	32.56	24.67
F16.png	32.78	32.93	32.96	32.88	24.18
lena.png	32.27	32.01	32.08	32.09	24.42
baboon.png	25.95	24.18	25.42	25.43	24.56
kodim03.png	34.54	33.13	33.43	33.40	24.51
kodim01.png	29.13	27.80	28.23	28.20	24.79
peppers.png	31.20	30.71	30.63	30.68	24.03
kodim02.png	32.44	31.95	31.20	31.96	23.71
kodim12.png	33.76	32.16	32.66	32.64	24.61
Average	31.68	30.64[a]	31.02	31.09	

[a]Per-image PSNR was not available in the Deep Image Prior paper, so here are the results we computed ourselves. Although the authors of Deep Image Prior report an average PSNR of 31.00, we were not able to reproduce this number.

Table 2.1: Denoising results with $\sigma = 25$

As we can see in table 2.1, AMNS does not improve on SMNS by more than a fews cents and seems to actually decrease the PSNRs most of the time, except for the image kodim02.png where the adaptive early stoppping

yields a 0.7dB gain. The reason for this is that static early stopping works well for the other pictures as the estimated method noise for them is close to the global threshold $T = 24.45$. However for the image kodim02.png estimated method noise is much lower, because the image RGB values are mostly close to 1 (for red pixels) or 0 (for blue and green pixels), as the image is mostly red. This means that in this image, the corrupting noise is clipped more than on other images, which means the method noise gets lower. The only other image getting improvements in denoising performance with the adaptive early stopping method is the peppers.png, which also has an estimated method noise lower than the average as it is a mostly red and green image.

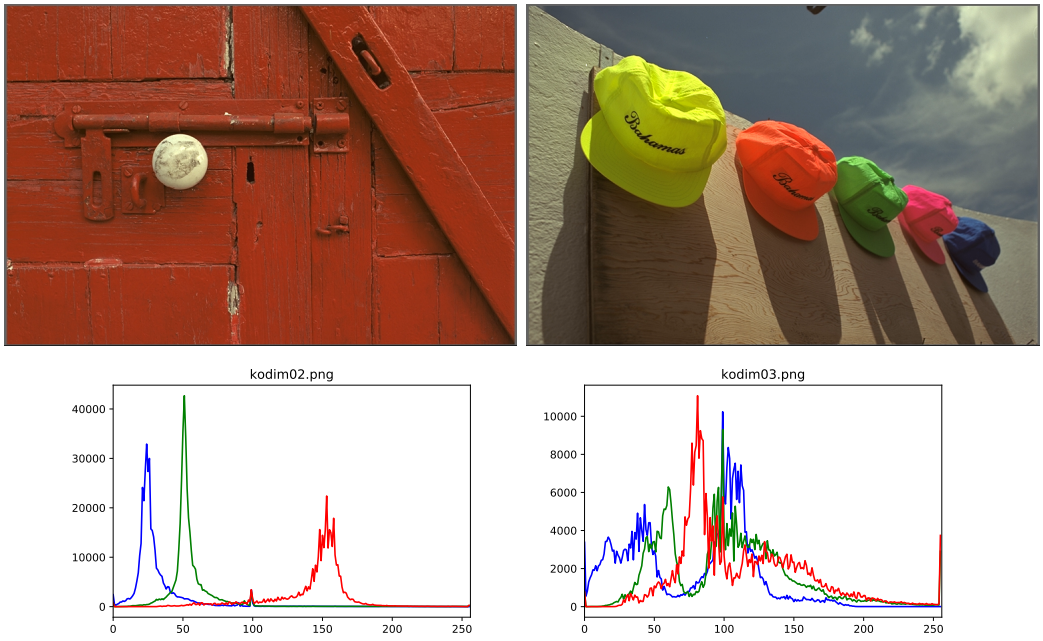
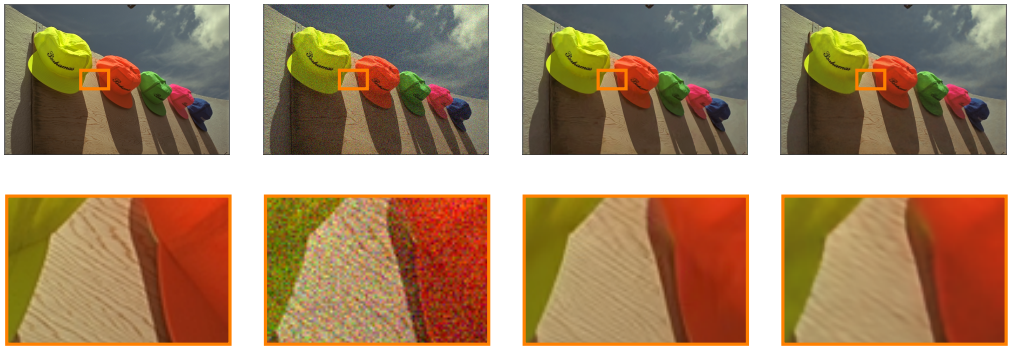


Figure 2.14: Left: kodim02.png, right: kodim03.png. As shown in the histogram, the RGB pixel values for kodim02.png are distributed closer to the clipping limit than kodim03.png, leading to more clipping when AWGN is added.

While our improvements yields a substantial PSNR increase over the original implementation, it still doesn't beat CBM3D, for this reason we analyze the per-image results, and we focus on the cases where the difference is the larger. For the image kodim03.png CBM3D outpeforms DIP+AMNS by 1.14dB. Looking at the details in figure 2.15, we see that CBM3D is much more efficient at denoising repetitive, high-frequency details by matching similar neighboring patches together, while DIP simply "hallucinates" a texture

coherent with its neighborhood. However, this aptitude comes at the cost of ringing artifacts, to which DIP is immune as shown in figure 2.16.



(a) Ground truth (b) Noisy Image (c) CBM3D (d) DIP+AMNS

Figure 2.15: Comparing denoising results for the image kodim03.png.

(a) Ground truth (b) Noisy Image (c) CBM3D (d) DIP+AMNS

Figure 2.16: Comparing denoising results for the image F16.png.

3 Faster: Progressively Growing the Network

In this chapter, we first review a technique [5] developed to speed-up and stabilize the training of GANs, for generating high-resolution pictures. Although DIP is not a GAN, we introduce an adaptation of this approach to DIP, speeding up the computing time by 1.6x and improving the PSNR by 0.15dB when combined with early stopping approaches developed in the previous chapter. Additionnally, we show that this technique also yield a substantial speed-up when applied to the super-resolution task.

3.1 Background

GANs [16] have been notoriously hard to train, due to the unstability of the minimax game played between the generator and discriminator networks. Recently, a few techniques [17] [18] have been developed which make training easier, but in the field of image generation, generating pictures with a resolution higher than 256^2 was still considered difficult. In [5], the authors propose an approach where the generator and discriminator networks are first trained with few layers and at a low resolution, before progressively adding new layers and increasing the resolution as shown in figure 3.1. This method, coupled with other stabilization techniques that are not relevant to our work, allow the authors to train GANs to generate images as large as 1024^2.

By first training a shallow network on low-resolution input, the authors can train with a much larger mini-batch, substantially speeding-up convergence. Although they were not able to compare the speed-up between their progressively grown GAN against a standard GAN, as it would take too long to train the latter, they extrapolate that their approach is about 5.4x faster.

Additionally to computational aspects, the training stability and speed-up improvements are understandable in the sense that it is easier for the network to first learn coarse spatial structure, before learning progressively finer details, instead of learning all filters at once. This behavior is likely related to the idea of Curriculum Learning [19], which proposes that, as it is the case with human learning, it is easier for a neural network to learn first on easy examples before moving on to progressively harder examples.

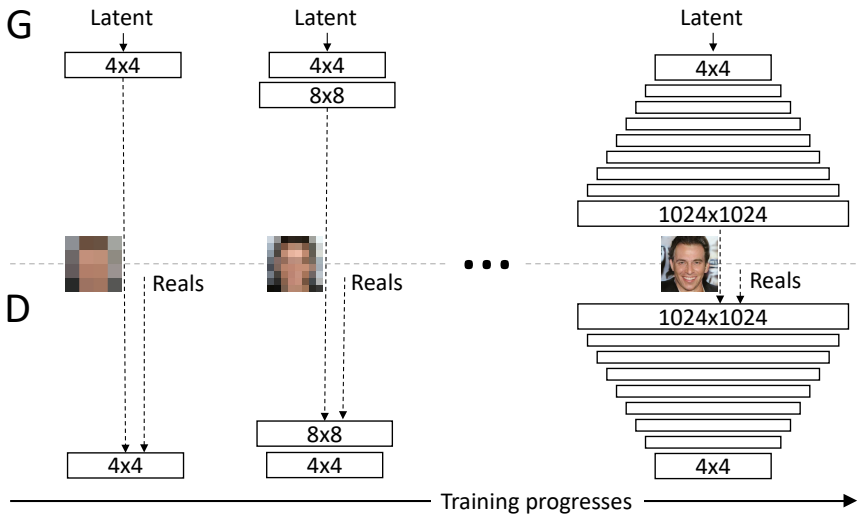


Figure 3.1: Progressively adding new layers to a GAN and increasing the resolution. Figure taken from [5].

3.2 Method

While Deep Image Prior is not a GAN, it is still possible to progressively grow the network in the same fashion as [5]. We start with a two layer network which takes inputs/outputs of spatial dimension $\frac{W}{2^{k_{max}-k}} \times \frac{H}{2^{k_{max}-k}}$ with $W \times H$ the dimension of the image to denoise, $k = 0$, and k_{max} the max number of levels the network will have. A level is defined as the part of a network which will be added for each resolution increase. We train the network for a fixed number of iterations, and then we go to the next level, incrementing k and adding additional layers at the start and the end of the network, as shown in figure 3.2.2.

Starting from equation 2.3, we can define the optimization equation for each level $k \in [0, k_{max}]$

$$\hat{Y_k} = f_{k_{\hat{\theta}_k}}(z_k), \quad \hat{\theta}_k = \underset{\theta_k}{\operatorname{argmin}}\, MSE(Y_k; f_{k_\theta}(z_k)) \tag{3.1}$$
$$Y_k = Downsample(Y, 2^{k_{max}-k})$$
$$z_k = Downsample(z, 2^{k_{max}-k})$$

where f_k is the network with k levels, further described below. When new layers are added, they do not immediately replace the previous input and output layers but are instead progressively faded in, as shown in figure 3.2.2. During this transition phase, instead of directly replacing z_{k-1} and Y_{k-1} by their 2x resolution counterpart z_k and Y_k, we use $\widetilde{z_k}$ and $\widetilde{Y_k}$ which crossfades

between them until the end of the phase. During this phase, we use a lower learning rate which is progressively ramped up back to the previous learning rate. This is necessary to avoid gradient explosion when new layers are introduced. Once the new layers are fully faded in, the previous input and output blocks are discarded.

$$\widetilde{z_k} = \alpha(z_k) + (1-\alpha)(Upsample(z_{k-1})) \tag{3.2}$$
$$\widetilde{Y_k} = \alpha(Y_k) + (1-\alpha)(Upsample(Y_{k-1}))$$

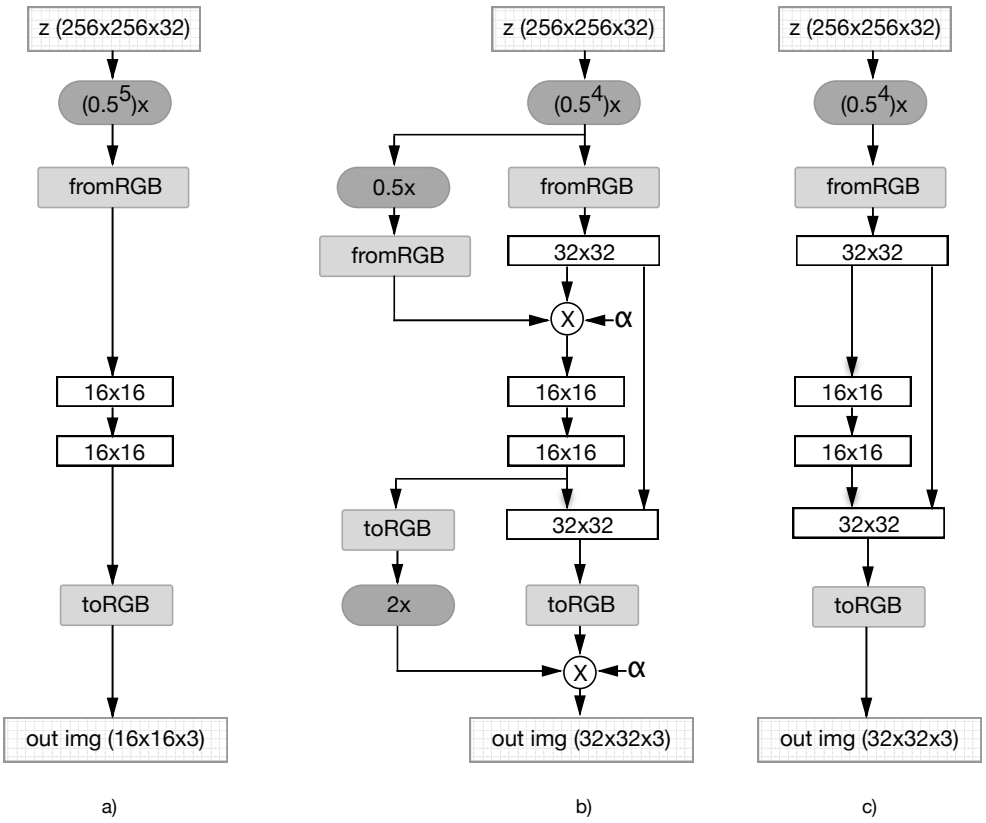


Figure 3.2: (draft figure) We start from a 16×16 network in a), then add 32×32 convolutional blocks, and fade them in progressively in b). When the fading in is done, we discard the previous *fromRGB* and *toRGB* blocks. Dark grey blocks are downsample and upsample blocks, circles marked with an X are crossfade blocks.

3.2.1 Progressively Grown Deep Image Prior for super-resolution

Although this work focus on the image denoising task, the original DIP implementation also propose implementations for other image-restoration tasks such as super-resolution and image inpainting. For super-resolution, the approach is almost the same as denoising, but instead of directly comparing the network output to the input image, the network outputs a higher resolution image $\hat{Y}_{HR}$, which is then downsampled to $\hat{Y}_{LR}$ and compared to the input image. The result is then backpropagated to the network, through the downsampling operator as well. Once training is done, $\hat{Y}_{HR}$ is kept as the super-resolved image.

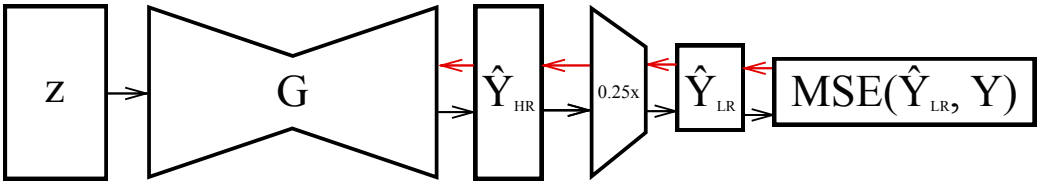


Figure 3.3: Deep Image Prior for super-resolution.

This means that in theory, PG-DIP can be applied almost as-is to the super-resolution task, we just need to adjust its output size to the desired super-resolution factor, and add a downsampling operator between its output and the MSE.

3.2.2 Implementation

The PGNetwork class

The original DIP implementation instantiates the network by constructing an `nn.Sequential` instance which includes all the required layers in it. To implement the PG-DIP method, we opt instead to define our progressively grown network `PGNetwork` by subclassing `nn.Module` so that we can use a more object-oriented approach when interacting with the network. We define three methods for this class:

PGNetwork.grow() This method is called at the beginning of each new level, to add new layers to the network. It creates new input/output layers with a larger resolution, and put them together with the I/O layers of the previous layers in front of crossfading layers. This brings the network from state a) to b) as shown in figure , effectively bringing in the transition phase.

PGNetwork.set_alpha(α) The method is called at each iteration during the transition phase, to update the α parameter of the crossfading layer. When α reaches 1, the output of layers to the left of the `CF` operator in is not used anymore.

PGNetwork.flush() This method is called once the transition phase is done, to remove layers that are not used anymore. It removes the input/output blocks of the previous level, and remove th crossfading layer. This brings the network from state b) to c) as shown in figure , effectively bringing in it back to the stabilization phase.

The training procedure

In Python-like pseudo-code, the training procedure is defined as such:

```
img_noisy[MAX_LEVEL] = load_img()
net = PGNetwork()

# Create a tensor of the same shape as the noisy image
# with an additional dimension Z_DEPTH.
z = get_uniform_noise(size=img_noisy.shape, depth=Z_DEPTH)

for k in range(1, MAX_LEVEL + 1):
    net.grow()

    # Get a downsampled version of z and of the noisy image
    z_k = Downsample(z, (2**(MAX_LEVEL - k)))
    img_noisy[k] = Downsample(img_noisy[MAX_LEVEL],
                              (2**(MAX_LEVEL - k)))

    # Create a new optimizer for each phase
    optimizer = torch.optim.Adam(net.parameters(), lr=LR)

    for phase in ["trans", "stab"] if k != 1 else ["stab"]:
        if phase == "stab":
            # Flush the network when we begin a stab. phase
            net.flush()

        for i in range(get_iterations(k)):
            # Perturbate z with normal noise for regularization
            z_ki = z_k + normal_noise()
```

```
            img_target = img_noisy[k]
            if phase == "trans":
                alpha = min(i / RAMPUP_DURATION, 1.0)
                net.update_alpha(alpha)

                # Set the learning rate to a fraction of the
                # desired learning rate LR so that it grows
                # smoothly from 0 to LR during the transition
                # phase.
                set_LR(optimizer, LR*alpha)

                # Interpolate the image with the lower res.
                # image from the previous level.
                img_target = interpolate(img_noisy[k],
                                         img_noisy[k-1],
                                         alpha)

                # Compute the current denoised estimate
                out = net(z_ki)
                mse = MSE(out, img_target)

                # Backpropagate the error
                mse.backward()

        denoised_img = out
```

This actual implementation can be found in the file `denoise_pgdip.py` in the source code repository.

3.3 Results

Implementing the above modifications, we compute the running time of PG-DIP, compared to the original implementation. For a fair comparison, we run our implementation for a fixed number of iterations such that the average PSNR is approximately equal to the one obtained in the original Deep Image Prior implementation. We were able to combine PG-DIP with AMNS and SMNS, yielding better PSNR gains as the non-progressively grown version of those methods and with a faster running time.

As seen in table 3.1, additionally to systematically reducing the running time by a factor around 1.6, the PG-DIP also increase the PSNR when coupled with AMNS or SMNS. While no certain explanation has been found for

Method	Running time [s]	PSNR [dB]
DIP	4437	30.64
PG-DIP	2742	30.52
DIP + SMNS	5629	31.02
PG-DIP + SMNS	3433	31.17
DIP + AMNS	5978	31.09
PG-DIP + AMNS	3291	31.24

Table 3.1: Running times of Progressively Grown Deep Image Prior against the original implementation.

this PSNR increase, our hypothesis is that the kind of Curriculum Learning implemented by progressively growing the network leads to a better regularized solution, as working at lower resolutions decrease the impact of noise. This means that while the model in the original implementation could start overfitting the noise as soon as the first iteration, this is delayed in in the progressively grown implementation, leading the model to let less noise pass through.

	CBM3D	PG-DIP+AMNS	Difference
house.png	33.03	32.48	-0.55
F16.png	32.78	32.96	0.18
lena.png	32.27	32.07	-0.20
baboon.png	25.95	25.82	-0.13
kodim03.png	34.54	33.65	-0.89
kodim01.png	29.13	28.60	-0.43
peppers.png	31.20	30.63	-0.57
kodim02.png	32.44	32.18	-0.26
kodim12.png	33.76	32.81	-0.95
Average PSNR	31.68	31.24	-0.44

Table 3.2: Per-image PSNR comparison between CBM3D and our best implementation.

3.3.1 Super-resolution

As shown in table 3.3, PG-DIP reduce the computation time compared to the baseline, but by a smaller factor of 1.3x compared to the speed-up obtained for the denoising task. This can be explained by the fact that for the denoising task, more work is done at coarser spatial resolutions, as the

network's input is noisy even when downsampled when training the lowest resolution levels. For the super-resolution task, the network is fed a clean image for all levels before the two last levels, therefore convergence is easy. As soon as the network starts to generate an image of higher resolution than the input, convergence is slower. This means that unlike in the denoising task, most of the work is done during training of the last layer, so we can't save much time by training the inner layers first by progressively growing the network.

Method	Running time [s]	PSNR [dB]
DIP	867	28.84
PG-DIP	1111	28.84

Table 3.3: Running time of Progressively Grown Deep Image Prior for the super-resolution task.

4 Larger: Tiling Denoised Patches for Processing Large Images

4.1 Background

Images from the CBM3D test are not bigger than 0.4 megapixels, which requires about 6GB of GPU memory to be denoised by the Deep Image Prior algorithm. As the memory usage linearly depends on the number of pixels, the algorithm can not be applied for real-life pictures which nowadays can reach in the tens of megapixels. Although CNN implementations for classification tasks can easily be distributed across multiple GPUs or machines with data parallelization [20], this is not possible with the Deep Image Prior architecture as there is no minibatch to split. Instead, we propose to split the image to denoise in smaller regions of size $k \times k$, denoise them separately, and tile them afterwards. While this could potentially lower the achievable PSNR by reducing the support size of the algorithm, research shows that using a large support size brings diminishing returns [8], and that BM3D works fine with a support size of $k = 12$ [21]. In this chapter, we describe our implementation of a tiled denoising algorithm, based on the techniques developed in the previous chapters.

4.2 Method

Assuming the image width and height are multiples of 128, we split the image by 128×128 regions. We chose this region size as choosing a smaller region size would make the GPU underused, unless we were able to process multiple regions in parallel. While this parallelization is possible and would be desirable in a practical implementation, we did not implement it. To avoid artifacts at the borders between regions, we actually make the regions larger by 16px on each side, so that they overlap. To ensure that regions at the border of the image are of the same size as the other regions, we add reflection padding of width $p = 16$ to the original image.

In one of the first implementation of the tiling algorithm, we simply fed the regions to the original Deep Image Prior implementation. We encoun-

tered an issue where the algorithm would yield different performance on different regions, underfitting for some regions, overfitting for others, as seen in figure 4.1. While in practice it is possible to hand-tune the number of iterations when working on a single image until obtaining the desired results, it is not reasonable to do this n times for an image made of n tiles. Researching a solution to this issue led to the Early Adaptive Stopping method which accidentally led to higher PSNR as shown in a previous chapter. By computing the expected method noise for each region separately, and using this threshold to stop the algorithm when reached, the regions are then homogeneous in texture and no artifacts are visible anymore.

Another issue was that the network would overfit much more than the original Deep Image Prior implementation. Indeed, when working a single region, the input is much smaller than when working on the full image which means its information is reduced. Less capacity is then needed to represent it, so it is easier for the network to overfit. For this reason, the number of filters per layer is reduced proportionally to the input size.

Figure 4.1: Left: pathologic tiled denoising, the upper right region is visibly underfitting, making the right side of the plane blurrier than the left. Right: successful tiled denoising, the texture is homogeneous.

The implementation is available in the file `denoise_tiled.py`.

4.3 Results

While the goal of the tiled denoising algorithm is ultimately to work on large pictures, it is not possible to compare its performance against the baseline Deep Image Prior using such large pictures as the baseline cannot fit pictures of more than 1Mpx on the largest available GPU. Instead, we test the tiled

algorithm on the same dataset as before, so that we can verify that PSNR is not too much impacted by working on smaller regions. As we can see in table 4.1, the average PSNR is only slightly impacted. It is moderately lower than the non-tiled algorithm but still higher than the original, non-tiled Deep Image Prior implementation.

	DIP	PG-DIP+EAS	Tiled PG-DIP+EAS
house.png	31.04	32.48	32.56
F16.png	32.89	32.96	32.65
lena.png	32.03	32.07	31.96
baboon.png	23.82	25.82	25.73
kodim03.png	32.95	33.65	32.29
kodim01.png	27.57	28.60	28.35
peppers.png	30.64	30.63	30.65
kodim02.png	31.92	32.18	31.75
kodim12.png	32.04	32.81	32.50
Average	30.54	31.24	30.94

Table 4.1: Accuracy of the tiled denoising algorithm on the color test set.

5 Conclusion

5.1 Summary

In this report, we successfully improved the Deep Image Prior algorithm on three axes: we increased its denoising performance, we made it 60% faster, and we made it able to work on arbitrarily large images. We have experimented with various network architectures and optimization techniques, as we originally thought that this would be the best way to improve performance. However, none of this yielded significant results, as more complex models overfitted the noise too easily and simpler models would not be able to reproduce accurately the signal. For this reason, we focused instead on developing a task-specific regularization technique. This yielded substantial PSNR improvements, 0.38dB more than the original implementation, greatly reducing the gap towards CBM3D.

Inspired from recent works in the field of GANs, we also implemented a progressively grown network, making the algorithm 60% faster and incidentally improving the PSNR as well, up to 0.60dB over DIP. While we have not extensively experimented with super-resolution, the progressively grown network we implemented works almost as-is for this task, but yielding a smaller speed-up than for the denoising task. Lastly, we made the algorithm able to work on arbitrarily large images, making it much more practical for real-life applications.

5.2 Limitations and Future Work

As we used the same experimental settings as in the original DIP algorithm, our model does not accurately represent the task of denoising real-life natural images. Indeed, we assumed a Gaussian-distributed noise with a fixed standard deviation, whereas in real photographs the noise is actually a combination of Gaussian and Poissonian noise, and its intensity can highly vary depending on scene illumination, sensor sensibility etc. To make our approach more practical, it would be needed to couple it with a noise-estimation technique such as [15]. An interesting avenue would be to evaluate denoising performance using real noisy photographs, where the ground truth is gener-

ated by taking the exact same picture but with lower sensibility and higher exposition time as in the Darmstadt Noise dataset [22].

Additionally, we only used the PSNR metric for assessing the algorithm performance. While it is the most commonly used metric for this task, which allows us to easily compare our scores against other results, it doesn't perfectly match the human perception of image quality. It would be interesting to measure the performance of denoising algorithms using other metrics, closer to human perception, such as Deep Features [23].

Bibliography

[1] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. Image denoising with block-matching and 3d filtering. In *IN ELECTRONIC IMAGING'06, PROC. SPIE 6064, NO. 6064A-30*, 2006.

[2] H. C. Burger, C. J. Schuler, and S. Harmeling. Image denoising: Can plain neural networks compete with bm3d? In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2392–2399, June 2012.

[3] Kai Zhang, Wangmeng Zuo, Shuhang Gu, and Lei Zhang. Learning deep CNN denoiser prior for image restoration. *CoRR*, abs/1704.03264, 2017.

[4] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Deep image prior. *CoRR*, abs/1711.10925, 2017.

[5] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017.

[6] A. Buades, B. Coll, and J. Morel. A review of image denoising algorithms, with a new one. *Multiscale Modeling & Simulation*, 4(2):490–530, 2005.

[7] S. G. Chang, Bin Yu, and M. Vetterli. Adaptive wavelet thresholding for image denoising and compression. *IEEE Transactions on Image Processing*, 9(9):1532–1546, Sep 2000.

[8] A. Levin and B. Nadler. Natural image denoising: Optimality and inherent bounds. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 2833–2840, Washington, DC, USA, 2011. IEEE Computer Society.

[9] M. Elad and M. Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image Processing*, 15(12):3736–3745, Dec 2006.

[10] M. Aharon, M. Elad, and A. Bruckstein. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, 54(11):4311–4322, Nov 2006.

[11] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew P. Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802, 2016.

[12] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, 2016.

[13] Lutz Prechelt. *Early Stopping — But When?*, pages 53–67. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[14] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.

[15] A. Foi, M. Trimeche, V. Katkovnik, and K. Egiazarian. Practical poissonian-gaussian noise modeling and fitting for single-image raw-data. *IEEE Transactions on Image Processing*, 17(10):1737–1754, Oct 2008.

[16] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. June 2014.

[17] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016.

[18] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. *ArXiv e-prints*, January 2017.

[19] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning.

[20] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017.

[21] Marc Lebrun. An Analysis and Implementation of the BM3D Image Denoising Method. *Image Processing On Line*, 2:175–213, 2012.

[22] Tobias Plotz and Stefan Roth. Benchmarking denoising algorithms with real photographs. *CoRR*, abs/1707.01313, 2017.

[23] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. *CoRR*, abs/1801.03924, 2018.

[24] Aapo Hyvrinen, Jarmo Hurri, and Patrick O. Hoyer. *Natural Image Statistics: A Probabilistic Approach to Early Computational Vision.* Springer Publishing Company, Incorporated, 1st edition, 2009.

[25] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew P. Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. *CoRR*, abs/1609.04802, 2016.

[26] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *CoRR*, abs/1501.00092, 2015.