Strategies for Designing Deep Denoising Networks for Plug-and-Play Image Recovery

(Report)

Kang Lin
03676489

Tobias Sukianto
03670221

Wisam Waad Noori
03674798

March 1, 2021

Abstract

Plug-and-Play (PnP) methods are a fairly new framework that has gained more popularity in recent years. It deals with inverse problems for which the forward model is known by utilizing convex solvers and PnP denoisers. In this report we present our methods and findings for evaluating certain strategies for designing deep denoising networks for Plug-and-Play image recovery. Specifically, we focus on CNN, U-net, and GAN based denoisers. We demonstrate that using different loss-functions other than the conveniently used MSE-loss for training PnP denoisers may exhibit performance advantages. Furthermore, we illustrate that better denoising performance of the PnP denoiser does not necessarily imply better performance for PnP image recovery.

Contents

1 Introduction

Solving inverse problems is a challenging but a required task in image processing, e.g. deblurring, inpainting, etc. If the forward model is known, it can be effective to incorporate this information into the problem formulation. Then, solving this inverse problem may be reformulated as a maximum a posteriori (MAP) estimation problem, for which under the assumption of convexity, exact solving algorithms exist, e.g. [BPC11, BT09]. However, defining an explicit prior distribution over the image-type of consideration is often times difficult to realize; yet having a prior assumption may be essential, especially if the problem is ill-conditioned or even ill-posed [ABB+20, BT09].

To this end, a fairly new framework has been proposed which is known under the name of *Plug-and-Play* (PnP) algorithms [VBW13]. In the Plug-and-Play framework, one does not express an explicit prior. Instead it exploits solving-algorithms that inherently have a denoising-like step. That is, PnP methods simply substitute the denoising-like step in the original solving-algorithm with some sophisticated denoiser and run the algorithm from there on. The plug-in denoiser is implicitly associated with some image-prior.

Since deep neural network based denoisers, or in short deep denoising networks (DDNs), have become powerful tools for image denoising, a natural step is to consider DDNs as denoisers within the PnP framework. This has led to promising results in image recovery tasks such as deblurring or super-resolution [ZLZ+20, BHSD19, MME19], or image reconstruction in compressive MRI [ABB+20].

However, due to the novelty of this framework, fundamental strategies concerning the design or training of DDNs specifically for benefiting PnP algorithms remain to be open topics. For example, one could ask if performance gains can be achieved by trading-off denoising performance for more confident priors or vice versa. Or, which denoiser architecture should be used for a specific inverse problem. Hence, this work hopes to provide some directions and ideas to address these topics.

1.1 Problem Formulation

Consider the following inverse problem,

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{w}, \quad \mathbf{w} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I}), \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the unknown vectorized image to be recovered, $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m \leq n$ is the known linear transformation operating on the image $\mathbf{x}$, $\mathbf{y} \in \mathbb{R}^m$ is the measurement vector, and $\mathbf{w} \in \mathbb{R}^m$ is zero-mean Gaussian noise with covariance $\sigma^2\mathbf{I}$. The goal is now to recover $\mathbf{x}$ from the known measurements $\mathbf{y}$. Note, however, that this problem is often ill-conditioned, or even ill-posed when $\mathbf{A}$ is rank deficient or $m < n$. Hence, in such cases, estimating $\mathbf{x}$ by means of maximum-likelihood or least-squares becomes impractical.

Therefore, instead of performing maximum-likelihood estimation, we consider MAP estimation by defining a prior density $p(\mathbf{x})$ over the image-space of the form $p(\mathbf{x}) \propto \exp(-\phi(\mathbf{x}))$, where $\phi : \mathbb{R}^n \to \mathbb{R}$ is convex. This results in an optimization problem that is equivalent to the MAP estimation of $\mathbf{x}$, i.e.,

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} \{ \frac{1}{2\sigma^2} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 + \phi(\mathbf{x}) \}, \tag{2}$$

where $\hat{\mathbf{x}}$ is the estimated image, and $\|\cdot\|_2$ denotes the ℓ_2-norm.

Since both terms in (2) are convex, we attempt to solve the problem by first relaxing it through variable-splitting and then apply a suitable convex solver such as *alternating direction method of multipliers* (ADMM) [BPC11]. By means of variable splitting we introduce the variable $\mathbf{v}$ and rewrite problem (2) into

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} \{ \frac{1}{2\sigma^2} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 + \phi(\mathbf{v}) \}, \quad \text{s.t.} \quad \mathbf{x} = \mathbf{v}. \tag{3}$$

Then, by applying ADMM to (3), we equivalently solve the following unconstrained min-max problem,

$$\min_{\mathbf{x},\mathbf{v}} \max_{\mathbf{u}} \{ \frac{1}{2\sigma^2} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 + \phi(\mathbf{v}) + \frac{1}{2\eta} \|\mathbf{x} - \mathbf{v} + \mathbf{u}\|_2^2 - \frac{1}{2\eta} \|\mathbf{u}\|_2^2 \}, \tag{4}$$

where $\eta \in \mathbb{R}$ is a penalty parameter affecting the convergence of ADMM and $\mathbf{u} \in \mathbb{R}^n$ is referred to as *scaled* dual variables. The ADMM-based algorithm for solving (4) is then provided by the iterative computation of

$$\mathbf{x}_k = \arg\min_{\mathbf{x}} \{ \frac{1}{2\sigma^2} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2^2 + \frac{1}{2\eta} \|\mathbf{x} - (\mathbf{v}_k - \mathbf{u}_k))\|_2^2 \} \tag{5}$$

$$\mathbf{v}_k = \arg\min_{\mathbf{x}} \{ \phi(\mathbf{x}) + \frac{1}{2\eta} \|\mathbf{x} - (\mathbf{x}_k + \mathbf{u}_k)\|_2^2 \} \tag{6}$$

$$\mathbf{u}_k = \mathbf{u}_{k-1} + (\mathbf{x}_k - \mathbf{v}_k). \tag{7}$$

This algorithm converges to $\hat{\mathbf{x}}$, i.e., the solution of (2) [ABB$^+$20].

Note that (6) resembles a typical denoising problem. The idea behind Plug-and-Play methods is to simply replace (6) with some sophisticated denoiser (*plug-in*) and let the algorithm run (*play*); thus, without defining $\phi(\mathbf{x})$ explicitly. The remarkable aspect is that PnP algorithms can work well even when unsure what prior the denoiser is associated with, or even when the denoiser is not associated with any prior [ABB$^+$20]. In other words, PnP methods can work well even when they violate the solver-dependent convergence assumptions.

Hence, general performance questions regarding DDNs applied to PnP methods remain. This motivates us to investigate possible design strategies for DDNs in an attempt to provide some answers to those question based on empirical evidence.

1.2 Related Work

The Plug-and-Play framework was introduced by Venkatakrishnan et al. [VBW13]. Here, they suggest that the denoising step (6) may be replaced by some arbitrary denoiser. In their experiment they use the BM3D as denoiser [DFKE07]; Although the BM3D is not associated with any prior, it worked well. However, deep denoising networks are not mentioned.

Ahmad et al. [ABB$^+$20] have worked out a comprehensive overview of methods in the Plug-and-Play framework for compressive MRI. The authors discuss several solving-algorithms that admit a denoising-like step which can be used in the PnP framework. Furthermore, they also discuss existing work on convergence properties of denoiser usage in PnP methods which do not admit an image prior, or when the prior is unknown. In their experiments they compare several PnP methods together with more classical compressive sensing based methods within the MRI setting, in which the PnP methods have shown superior performance. Although, we are not addressing the MRI setting in our work, Ahmad et al.'s paper can be used to gain general insights into the PnP framework, since their discussions about the PnP methods themselves are not limited to MRI.

Zhang et al. [ZZGZ17, ZLZ$^+$20] have introduced two DDN architectures for the Plug-and-Play method that uses Half Quadratic Splitting as the solving-algorithm. The first work uses a convolutional neural network (CNN) denoiser based on residual training [ZZC$^+$17]. That is, instead of learning the denoised image directly, the noise map is learned and then subtracted from the noisy image. This has worked well on image restoration tasks such as deblurring or super-resolution, but was beaten by their latter contribution. There, they use a network based on the U-Net [RFB15] which slightly improved the PSNR on similar tasks.

To this end, we would like to emphasize that we are not mainly interested in outperforming existing work, but would rather like to provide general DDN design suggestions for PnP methods depending on some standardized setting or available resources. This could then lead to building reasonable model architectures which may beat the state of the art; however, that would go beyond the scope of this work but is potentially considered in future work.

1.3 Approach Outline

The report is structured as follows: In Sec. 2, we provide an overview of the PnP denoiser architectures used for this project. Section 3 explains the experiments that are performed to evaluate certain design strategies for PnP denoiser design, and illustrates our results. In Sec. 4, we provide a more thorough discussion of the results that may suggest the one or other consideration when designing PnP denoisers. Lastly, we conclude our findings in Sec. 5.

2 Methods

In this section we present the PnP denoising networks that we use in the PnP framework to substitute (6). In total, we try three different but rather general approaches: A more classical convolutional neural network (CNN) approach described in Sec. 2.1, an U-net model in Sec. 2.2, and a Generative Adversarial Network (GAN) approach in Sec. 2.3. We mention our motivation for using each model and explain their specific architecture.

2.1 PnP CNN Denoiser

CNNs are usually the preferred architectures when the problem at hand is image related since they have shown impressive performance, e.g. in image-denoising. The main reason for their success is their ability to use and learn correlation filters, which can capture correlations in the inputs more efficiently. This is the case when dealing with images, since neighboring pixels are typically highly correlated. It is suspected that these filters encode certain prior information of the images that the network sees during training. This turns out to be useful, since our PnP denoisers should not only be effective at denoising itself, but rather it should also be able to learn a certain prior about the images in the dataset, as (6) suggests. We presume that having a strong prior will be especially useful when we consider the reconstruction of highly subsampled images.

The PnP CNN denoisers we use for PnP are motivated by the PnP denoiser from [ZZGZ17]. There, the authors suggest a Gaussian CNN denoiser whose filter's dilation parameter increases and then decreases in a symmetrical fashion. Furthermore, their CNN is based on residual learning [ZZC$^+$17] - i.e., learning the noise-map of the input-image explicitly - and utilizes batch-normalization as another key-component. Lastly, they also employ 3D and 4D convolutions. Except for the symmetrical dilation approach, we abandon the other ideas since we did not observe any benefits of using batch-normalization or explicit residual learning. The increasing and decreasing dilation shall allow the network to have a higher receptive field while maintaining the number of parameters. This seems to be beneficial for image-denoising.

Specifically, our architecture consists of 7 convolutional layers with 64 output channels each, for which the filter dilation parameters are set to follow the sequence (1, 2, 3, 4, 3, 2, 1). The filter size is 3x3, the activation functions in the first 6 layers are ReLU. A flow diagram of the model is given in Fig. 8.

2.2 PnP U-net Denoiser

U-net is a neural network architecture that is used for image segmentation tasks mainly in medical/biological imaging, but is also known to perform well in denoising tasks as shown in [HSB18]. The U-net architecture was first proposed in 2015 [RFB15]. It is shown in [KG20] that U-nets tend to retain the colors of images more effectively when compared to other denoising network architectures in classical denoising tasks. This may suggest superior capabilities for learning image priors which is a motivation to use the U-net as PnP denoiser.

The main idea is to supply a successive convolutional layers in an autoencoder structure, where in the encoding part pooling layers are utilized, and in the decoding part upsampling operators. An important feature of the U-net is that there are a large number of feature channels in the decoding part, which allows the network to propagate more information to higher layers. The encoder is also called the contracting path and the decoder is also called the expanding path.

In the contracting path, the input image is fed into the network. The data is propagated through along all possible paths. The network architecture consists of repeated applications of two 3x3 convolutions, each followed by a rectified linear unit. We use zero padding to retain the same image size. Then, a 2x2 max-pooling operation with stride 2 is for down-sampling performed. At each down-sampling step the number of feature channels is doubled.

Each step in the expansive path consists of an upsampling of the feature map followed by a 2x2 transpose convolution that halves the number of feature channels and the concatenation with the correspondingly padded feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU activation function. The feature map of the expansive map is zero padded until it reaches the contracting path's feature dimension size. The latter is also done to ensure that the output image is the same size as the input image. The final layer uses a 1x1 convolution to map the feature channels to the output channel dimension. A flow diagram of the model is given in Fig. 9.

2.3 PnP GAN Denoiser

Generative adversarial network is a deep learning framework for generative models. This architecture consists of a generative network and a discriminative network. The generative network is trained to generate indistinguishable samples from real data. The discriminative network is trained to distinguish whether the data comes from the generative network or from the real dataset. According to [CCCY18], GANs leverage the capability of CNNs to learn the noise model implicitly, without having explicit analytical knowledge of the prior. When we have data of the ground truth images and data of the noisy images, an intuitive way is to train the generative network as the denoising network. This motivates us to use the GAN as PnP denoiser. We build a GAN denoiser based on the ResNet architecture. The goal is to train a generator network to denoise images by generating a noise free image while competing with the discriminator.

In each residual block, three 3x3 convolutions are stacked along with batch normalization and leaky ReLU activation. Three of these residual blocks are stacked to increase the depth of the network. We use skip connections between the input and output of each residual block. Skip connections make the network more efficient in training and have better convergence performance and are less prone to vanishing gradients. The skip connections feed the input to the deep layers so each residual block tune the output with reference to input. Zero padding is used in each convolutional layer to retain the image's original size. The architecture of the output block is similar to the one of the residual block, with the difference that we use sigmoid as an activation function instead the leaky ReLU and do not propagate forward the its input to the output. The respective channel dimensions are (128, 128, 128 , 3) after each consecutive residual block.

The goal of denoising tasks is not only to make the denoising result visually appealing, but also indistinguishable to the ground truth. Therefore, a discriminator to classify if each input image is real or fake is trained. Five convolutional layers with batch normalization and leaky ReLU activations are used in the discriminator network. Then these learned features are stacked at the end to map the output to a probability score with the sigmoid activation,

We are using the minmax objective function to train our GAN as proposed in [GPAM$^+$14].

$$\min_{\theta_g} \max_{\theta_d} \mathbb{E}_{x \sim P_{\mathrm{data}}}[\log D_{\theta_d}(x)] + \mathbb{E}_{z \sim P_z}[\log(1 - D_{\theta_d}(G_{\theta_g}(z)))] \tag{8}$$

where P_{data} is the distribution over the real images fed into the discriminator, and P_z is the distribution of the images generated by the generator. G_{θ_g} and D_{θ_d} are the generator and the discriminator with their respective parameters θ_g and θ_d.

3 Evaluation

At this point, we would like to mention that the general results and discussion in Sec. 4 are mainly concerned with the PnP U-net denoiser and PnP CNN denoiser. The results and discussion for the PnP GAN denoiser are provided in separate sections as we did not manage to make it work effectively. Nevertheless, we want to provide the GAN results and discussions for completeness reasons.

3.1 Experimental Set-up

3.1.1 Dataset

We focus on recovering natural images. For training and evaluating our denoising networks, we use the Berkeley image segmentation and denoising dataset BSDS500 [AMFM11], as well as the Waterloo exploration dataset [MDW$^+$17]. The Berkeley image segmentation and denoising dataset contains roughly 500 images and is a commonly used dataset for training and bench-marking deep denoising networks. However, as this dataset itself is rather small, we add more images from the Waterloo exploration dataset. Furthermore, we incorporate random flipping as data augmentation. By having a larger set of natural images, we hope that our PnP denoiser networks learn a stronger and more confident prior.

Our training set for the PnP denoiser training contains 2000 images and their pixel values are normalized to the range $[0, 1]$. However, as it is typically done for denoiser training, we train on smaller image-patches that are cut out randomly from the images. These image-patches have size 40x40.

To assess the performance of our PnP denoisers regarding denoising itself (not in PnP), we use a denoising test-set that contains 1000 40x40 image-patches taken from unseen images. Note, that our objective is not directly related to having good denoiser performance for the denoising-task itself; but rather to analyze the relation between denoising performance and PnP performance.

To evaluate our PnP denoising networks for PnP image recovery we use the popular test images "Lena", "Parrot" and "Monarch". These images are directly taken from `https://homepages.cae.wisc.edu/~ece533/images/`. The ground-truth images are depicted in Fig. 1.

3.1.2 Loss Functions for PnP Denoiser Training

We will differentiate between MSE-loss and other loss-functions for the denoiser training. The MSE-loss is intuitive as it seems to be suggested by the proximal map (6). However, we consider also other loss-functions to reveal possible preferences for PnP. These loss-functions are a custom loss defined as the multiplication between the L1-loss and the SSIM measure, denoted by $\mathbb{L}_{\mathrm{L1*SSIM}}$, or simply the L1-loss, denoted by $\mathbb{L}_{\ell_1}$.

3.1.3 PnP Denoiser Training

We train each PnP denoiser once on the MSE-loss and twice on custom loss functions. The mini-batch size is 32. We present three CNNs and three U-net denoisers denoted by $\mathrm{C_{mse}}, \mathrm{C_{good}}, \mathrm{C_{bad}}$ and $\mathrm{U_{mse}}, \mathrm{U_{good}}, \mathrm{U_{bad}}$, respectively. The subscript "bad" or "good" refers to the relative denoising performance of the denoiser itself. This is motivated by our question on how denoising performance is related to PnP performance. This is discussed in further detail in Sec. 4.

For the CNN denoiser we injected additive i.i.d. zero-mean Gaussian noise with standard-deviation 0.05 to the training images. The CNN denoisers were trained using a total of 920-1840 gradient descent steps with Adam-optimizer and the custom-loss was $\mathbb{L}_{\mathrm{L1*SSIM}}$ for training $\mathrm{C_{good}}$ and $\mathrm{C_{bad}}$.

The U-net denoisers training use images with uniform noise between -0.1 and 0.1. The U-net denoisers $\mathrm{U_{good}}$ and $\mathrm{U_{mse}}$ are trained with a total of 9400 gradient steps. We used the Adam-optimizer with learning rate 10^{-4} for the first 8460 gradient descent steps, followed by 940 gradient descent steps of SGD with a momentum of 0.9 and a learning rate of 10^{-5}. $\mathrm{U_{bad}}$ was trained for a total of 940 gradient descent steps with the Adam optimizer using a learning rate of 10^{-3}. Both, $\mathrm{U_{good}}$ and $\mathrm{U_{bad}}$, are trained using the L1-loss.

3.1.4 Inverse Problems for PnP Image recovery

For the set of inverse problems (1) to test our PnP denoisers, we consider the randomly subsampled noisy deblurring problem, i.e., when $m < n$. Specifically, we aim for $m \approx 0.2n$, since we consider this to be reasonable data reduction. As a special scenario to highlight an important finding, we will also show one result for $m \approx 0.1n$. The blur is a Gaussian-blur with window size thrice its standard deviation. We use this window size as it incorporates almost the entire Gaussian bell yielding maximal blur. The blur's standard deviation is 1.6 as we wish to be consistent with previous works [ZLZ$^+$20, MME19]. The noise is additive i.i.d. Gaussian with standard deviation $\sigma = 0.01$.

3.1.5 PnP Solver

The solving-algorithm in our PnP methods is ADMM as described in (5)$-$(7). We solve (5) using the conjugate gradients method. The hyperparameters are $\eta = 10\sigma^2$, 4 conjugate gradient iterations, and overall 100 ADMM iterations.

3.1.6 Evaluation Metrics

To quantify the denoising performance, we take the average PSNR and SSIM value of the 1000 images patches from the denoising test-set. Also, to quantify the PnP performance of our denoisers, we utilize the PSNR and the SSIM metrics. These two are arguably the most popular reference metrics used in image processing tasks. While PSNR is a good indicator for image quality, it is slightly biased towards over-smoothed results. SSIM, however, takes into account structural similarity or textures - for example edges - between two images [HZ10].

3.2 Results

3.2.1 Denoising Performance

In Fig. 2, we see the performances of the individual denoising networks on the denoising task itself. Illustrated are the average PSNR in dB and SSIM values with $\pm$ one standard-deviation. The left half of the table contains the results of the CNN denoisers, while the right half shows the results of the U-net denoisers. Although, we trained the MSE-loss based denoiser to the best of our ability, it is interesting to see that the+ CNN denoisers trained on the custom loss yield better or at least similar results for Gaussian denoising. Even the “bad" denoiser $\mathrm{C}_{\mathrm{bad}}$ that we purposely trained to be worse than $\mathrm{C}_{\mathrm{good}}$, seems to have better denoising capabilities. This can be possibly explained by the results from [ZGFK17], where the authors demonstrate that other non MSE-loss functions, such as L1-loss, can be more beneficial than MSE-loss for Gaussian denoising.

3.2.2 PnP: MSE-Loss vs. Custom Loss

Figure 3 depicts the quantitative image recovery performances on the three test images, Lena, Monarch, Parrots, using our PnP denoisers. The left half of the table contains the results of the CNN denoisers, while the right half those of the U-net denoisers. The table entries contain the obtained PSNR in dB and SSIM value of the recovered image. The results between the CNN denoisers and U-net seem comparable. However, there is possibly a tendency that the U-net denoisers obtain slightly higher values on average. Within the CNN denoiser results, there is a consistent observation that the CNN trained on MSE-loss performs worse than the CNN denoisers trained on the custom-loss.

Figure 4 and Fig. 5 illustrate the performance of the PnP recovery task on Parrots using the PnP CNN denoisers, and on Monarch using the PnP U-net denoisers, respectively. The top rows show the noisy, blurred and subsampled input images, while the rows beneath show the result using our PnP denoisers. The second row illustrate the results for the MSE-loss trained denoisers, the third rows for the “good" denoisers, and the fourth rows for the “bad" denoisers. While the results for all U-Net denoisers seem a bit over-smoothed but rather similar overall, we notice that in the case of the CNN denoiser the MSE-loss trained denoiser reveal small black artifacts in the recovered image. Furthermore, the “good" CNN denoiser yields a smoother reconstruncted image than the “bad" denoiser, whose reconstruction appears to be a bit grainy.

3.2.3 PnP: Good vs. Bad Denoiser

The results for the special scenario where we use a subsampling of $m \approx 0.1n$ are illustrated in Fig. 6. Here, we do not compare with the MSE-loss trained denoiser, but only between the “good" and “bad" denoiser. The top image shows the noisy subsampled and blurred image of Lena. In the middle row we show the reconstruction using $\mathrm{C}_{\mathrm{bad}}$ (left) and $\mathrm{C}_{\mathrm{good}}$ (right). In the bottom row we show the reconstruction using $\mathrm{U}_{\mathrm{bad}}$ (left) and $\mathrm{U}_{\mathrm{good}}$ (right). We notice immediately that the reconstruction using the “good" denoisers yields various artifacts, many more than in the reconstructions obtained by using the “bad" denoisers. Comparing the CNN versus the U-net denoiser, we see that the U-net generates fewer artifacts but the result is more over-smoothed.

3.3 Evaluating GANs

The GAN is trained on the objective function (8). We feed the generator noisy training images with zero-mean Gaussian noise with standard deviation 0.05. The GAN is trained using a batch size of 16. The discriminator and the generator are trained equally. For both the generator and the discriminator, we use the ADAM optimizer with a learning rate of 10^{-4}. The discriminator and the generator are both trained using a total of 18000 gradient steps.

The top image of Fig. 7 shows the individual performance of the denoising network of GAN on the monarch image. The result shows a yellow tint on the generator output. The bottom image of Fig. 7 shows the result where we use a subsampling of $m \approx 0.2n$. We can see that the reconstruction using the GAN denoiser yields various miniscule artifacts along with the yellow tint.

4 Discussion

4.1 Consideration of Loss Functions

Note that (6) is implied by Gaussian denoising, thus, suggesting the use of MSE-loss for denoiser training. However, we demonstrate that using a custom loss for PnP denoiser training may lead to similar or even better performances in PnP image recovery, as shown in Figs. 3, 4, 5. This is interesting to note since many existing works use the MSE-loss to train their PnP denoisers, e.g. [ZZGZ17]. Hence, judging by our results we may hypothesize that it is worth considering other loss-functions for PnP denoiser training.

Analogously, similar statements can be made about the artificial noise distribution on which the PnP denoisers are trained on. The U-net denoisers are trained on uniformly distributed noise, while the CNN denoisers are trained on Gaussian noise. Nevertheless, the models that were trained on uniform noise do not seem to be inferior to the models which used Gaussian noise.

4.2 Better Denoiser $\not\Rightarrow$ Better PnP

Contrary to our initial beliefs, our results suggest that a better denoising performance does not necessarily imply better PnP results. This is especially noticeable when recovering highly subsampled images, e.g., $m \approx 0.1n$, Fig. 6. While the results in Fig. 3 do not illustrate significant differences between using a “bad" or “good" PnP denoiser, Fig. 6 demonstrates that both bad PnP denoisers outperform the good PnP denoisers. Note that this is interesting since learning a “bad" denoiser generally requires significantly less training. We hypothesize that the “bad" denoisers learned a better image prior since recovering from fewer measurements need better priors. However, we are not able to provide more explicit reasons to explain this phenomenon.

4.3 PnP CNN denoiser vs PnP U-net denoiser

Although the quantitative results suggest that the PnP U-net and PnP CNN denoisers perform similarly well, from qualitative assessments we noted that the PnP U-net denoisers tend to provide more over-smooth results than the PnP CNN denoisers. It is not clear to us if that is due to the inherent autoencoder-like architecture of the U-net, or due to training on uniformly distributed noise, or due to possibly other reasons. We notice, however, that both PnP denoisers exhibit grainier PnP image recovery results when the denoising performance itself is worse.

Furthermore, the U-net denoisers require more training time than the CNN denoisers for similar denoising results. Also, due to the larger network structure of the U-net, the PnP ADMM (5) $-$ (7) iterations with the U-net take longer time. Nevertheless, we may still believe that the U-net denoisers have bigger potentials over conventional CNN denoiser for the PnP framework, due to their ability to possibly learn better image priors, as for example demonstrated in Fig. 6 where the PnP U-net denoiser yields fewer artifacts, or in [ZLZ$^+$20] where the authors use a U-net based PnP denoiser.

4.4 Performance of GAN

As the results show, the overall denoising performance of our PnP GAN denoiser with our settings yield insufficient results. In fact, we observe that training the GAN proves to be a difficult task. During the training process we encountered various problems, such as vanishing gradients resulting in black output images. Since, GANs are known to be unstable and have a long training time, we were restricted on the number of times we could evaluate its performance. We observed that each independent training run produced a different tinted image.

In order to counteract such instabilities, we also tried to incorporate regularized approaches such as Wasserstein GAN or Wasserstein GAN-GP loss functions. However, the latter proved to be rather ineffective, due to the even longer training time. Both loss functions require 5 discriminator steps before performing one generator step. Compared to using the standard GAN loss function, we achieved worse results for similar training time. Nevertheless, we may believe that training the GAN with the latter loss functions may yield better results in the long run. However, confirming the latter hypothesis would require us to train the GAN even longer.

Because of aforementioned reasons, we decided to stop focusing on the GAN and instead focus on the other denoising architectures which offered better immediate results along with less training time.

5 Conclusions

For this project, we evaluated various deep denoising networks as plug-in denoiser for the Plug-and-Play (PnP) framework, as an attempt to conclude possible design strategies for PnP image recovery. We considered the use of different loss functions, noise distributions, and denoiser architectures for training the PnP denoisers. The main PnP denoising architectures for our analysis were a U-net based denoiser and a classical CNN based approach. We also attempted to use a GAN based denoiser. However, this was ineffective for our purposes.

While the PnP U-net denoisers and PnP CNN denoisers exhibited similar performance, we illustrated that using a custom-loss other than the conveniently used MSE-loss may lead to superior results for PnP image recovery. Furthermore, we also demonstrated that a PnP denoiser with better denoising performance does not necessarily lead to a better a plug-in denoiser for PnP image recovery.

Possible directions for future work may consist of a more rigorous analysis to better understand the presented phenomena, or building PnP denoisers to outperform existing work by leveraging the findings in this report.

References

[ABB+20] Rizwan Ahmad, Charles A Bouman, Gregery T Buzzard, Stanley Chan, Sizhuo Liu, Edward T Reehorst, and Philip Schniter. Plug-and-play methods for magnetic resonance imaging: Using denoisers for image recovery. *IEEE Signal Processing Magazine*, 37(1):105–116, 2020.

[AMFM11] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011.

[BHSD19] Siavash Bigdeli, David Honzátko, Sabine Süsstrunk, and L Andrea Dunbar. Image restoration using plug-and-play cnn map denoisers. *arXiv preprint arXiv:1912.09299*, 2019.

[BPC11] Stephen Boyd, Neal Parikh, and Eric Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[BT09] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[CCCY18] Jingwen Chen, Jiawei Chen, Hongyang Chao, and Ming Yang. Image blind denoising with generative adversarial network based noise modeling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[DFKE07] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising by sparse 3-d transform-domain collaborative filtering. *IEEE Transactions on Image Processing*, 16(8):2080–2095, 2007.

[GPAM+14] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

[HSB18] Mattias P Heinrich, Maik Stille, and Thorsten M Buzug. Residual u-net convolutional neural network architecture for low-dose ct denoising. *Current Directions in Biomedical Engineering*, 4(1):297–300, 2018.

[HZ10] A. Horé and D. Ziou. Image quality metrics: Psnr vs. ssim. In *2010 20th International Conference on Pattern Recognition*, pages 2366–2369, 2010.

[KG20] Rina Komatsu and Tad Gonsalves. Comparing u-net based models for denoising color images. *AI*, 1(4):465–487, 2020.

[MDW+17] Kede Ma, Zhengfang Duanmu, Qingbo Wu, Zhou Wang, Hongwei Yong, Hongliang Li, and Lei Zhang. Waterloo Exploration Database: New challenges for image quality assessment models. *IEEE Transactions on Image Processing*, 26(2):1004–1016, Feb. 2017.

[MME19] Gary Mataev, Peyman Milanfar, and Michael Elad. Deepred: Deep image prior powered by red. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[VBW13] Singanallur V Venkatakrishnan, Charles A Bouman, and Brendt Wohlberg. Plug-and-play priors for model based reconstruction. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 945–948. IEEE, 2013.

[ZGFK17] H. Zhao, O. Gallo, I. Frosio, and J. Kautz. Loss functions for image restoration with neural networks. *IEEE Transactions on Computational Imaging*, 3(1):47–57, 2017.

[ZLZ$^+$20] Kai Zhang, Yawei Li, Wangmeng Zuo, Lei Zhang, Luc Van Gool, and Radu Timofte. Plug-and-play image restoration with deep denoiser prior. *arXiv preprint arXiv:2008.13751*, 2020.

[ZZC$^+$17] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE Transactions on Image Processing*, 26(7):3142–3155, Jul 2017.

[ZZGZ17] Kai Zhang, Wangmeng Zuo, Shuhang Gu, and Lei Zhang. Learning deep cnn denoiser prior for image restoration. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3929–3938, 2017.

6 Appendix

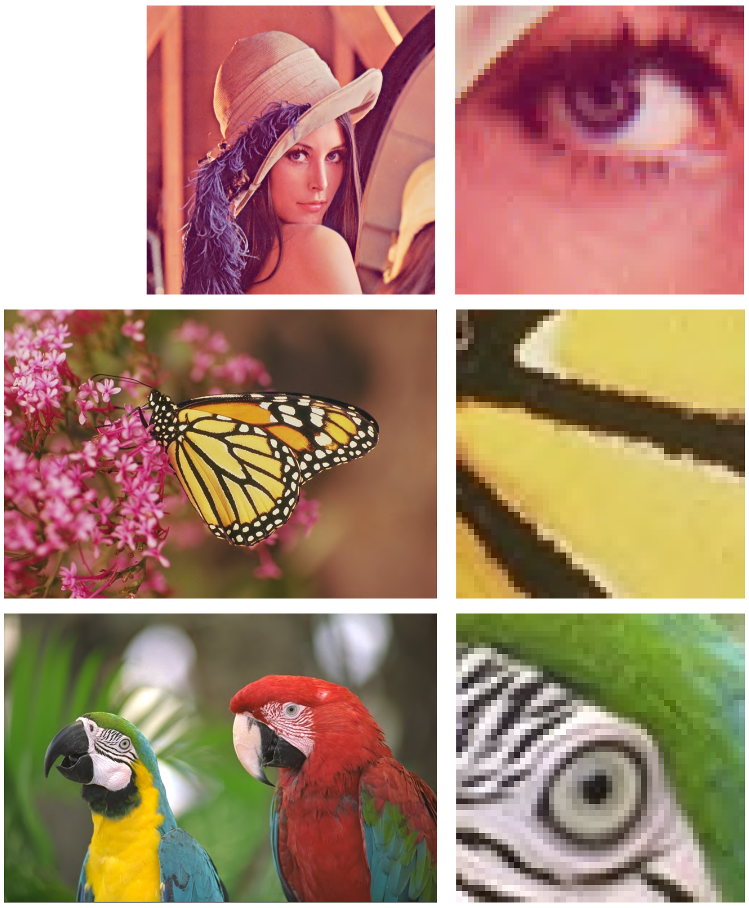

Figure 1: Ground truth test images. Top is “Lena". Middle is “Monarch". Bottom is “Parrots". *Return to Dataset*.

	C_{mse}	C_{good}	C_{bad}	U_{mse}	U_{good}	U_{bad}
PSNR	29.304 ± 1.568	33.81 ± 2.474	31.11 ± 2.049	32.16 ± 2.94	32.564 ± 3.386	30.45 ± 2.782
SSIM	0.812 ± 0.119	0.938 ± 0.033	0.918 ± 0.037	0.914 ± 0.048	0.899 ± 0.039	0.919 ± 0.045

Figure 2: Denoising Results. Mean PSNR [dB] and SSIM $\pm$ one standard deviation. *Return to Denoising Performance.*

	C_{mse}	C_{good}	C_{bad}	U_{mse}	U_{good}	U_{bad}
Lena	28.92 0.71	29.17 0.776	29.51 0.778	29.12 0.77	29.29 0.78	28.87 0.77
Monarch	28.15 0.79	28.37 0.899	29.09 0.901	29.16 0.9	28.97 0.9	28.49 0.89
Parrots	29.55 0.75	29.58 0.879	30.44 0.882	29.7 0.88	29.87 0.88	29.55 0.87

Figure 3: PnP Results on Lena, Monarch, Parrots. PSNR [dB] on top and SSIM on bottom. *Return to PnP: MSE-Loss vs. Custom Loss or Consideration of Loss Functions or Better Denoiser $\not\Rightarrow$ Better PnP.*

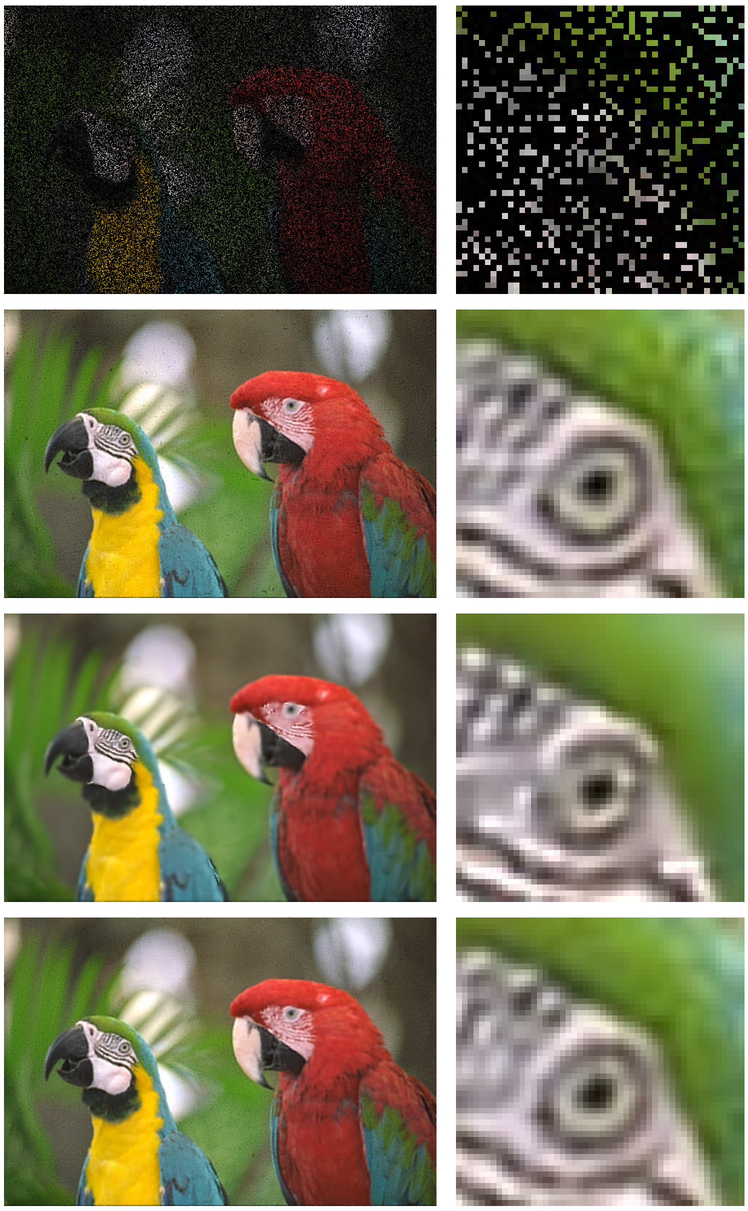

Figure 4: PnP recovery of noisy Parrots with Gaussian blur and random subsampling of $m \approx 0.2n$ using PnP CNN denoiser. First row: Input image that needs to be recovered. Second row: Recovery using $\mathrm{C}_{\mathrm{mse}}$. Third row: Recovery using $\mathrm{C}_{\mathrm{good}}$, Fourth row: Recovery using $\mathrm{C}_{\mathrm{bad}}$. *Return to PnP: MSE-Loss vs. Custom Loss or Consideration of Loss Functions.*

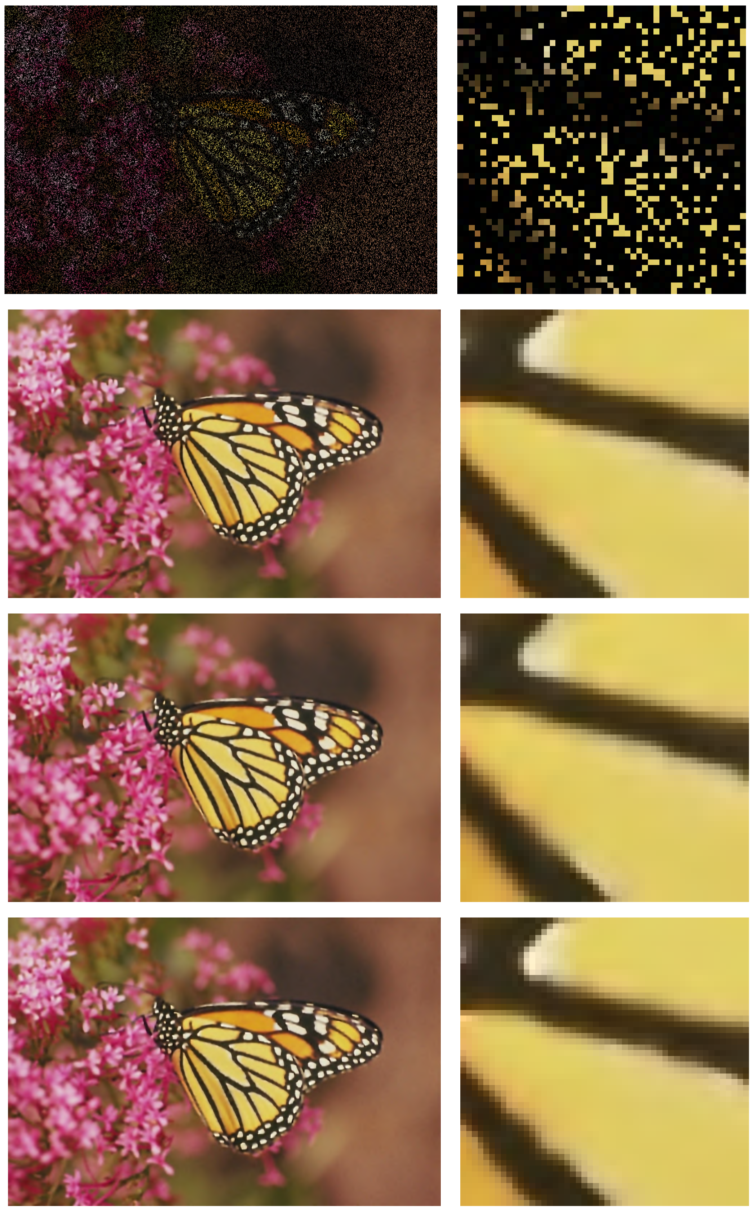

Figure 5: PnP recovery of noisy Monarch with Gaussian blur and random subsampling of $m \approx 0.2n$ using PnP U-net denoiser. First row: Input image that needs to be recovered. Second row: Recovery using $\mathrm{U}_{\mathrm{mse}}$. Third row: Recovery using $\mathrm{U}_{\mathrm{good}}$, Fourth row: Recovery using $\mathrm{U}_{\mathrm{bad}}$. *Return to PnP: MSE-Loss vs. Custom Loss or Consideration of Loss Functions*.

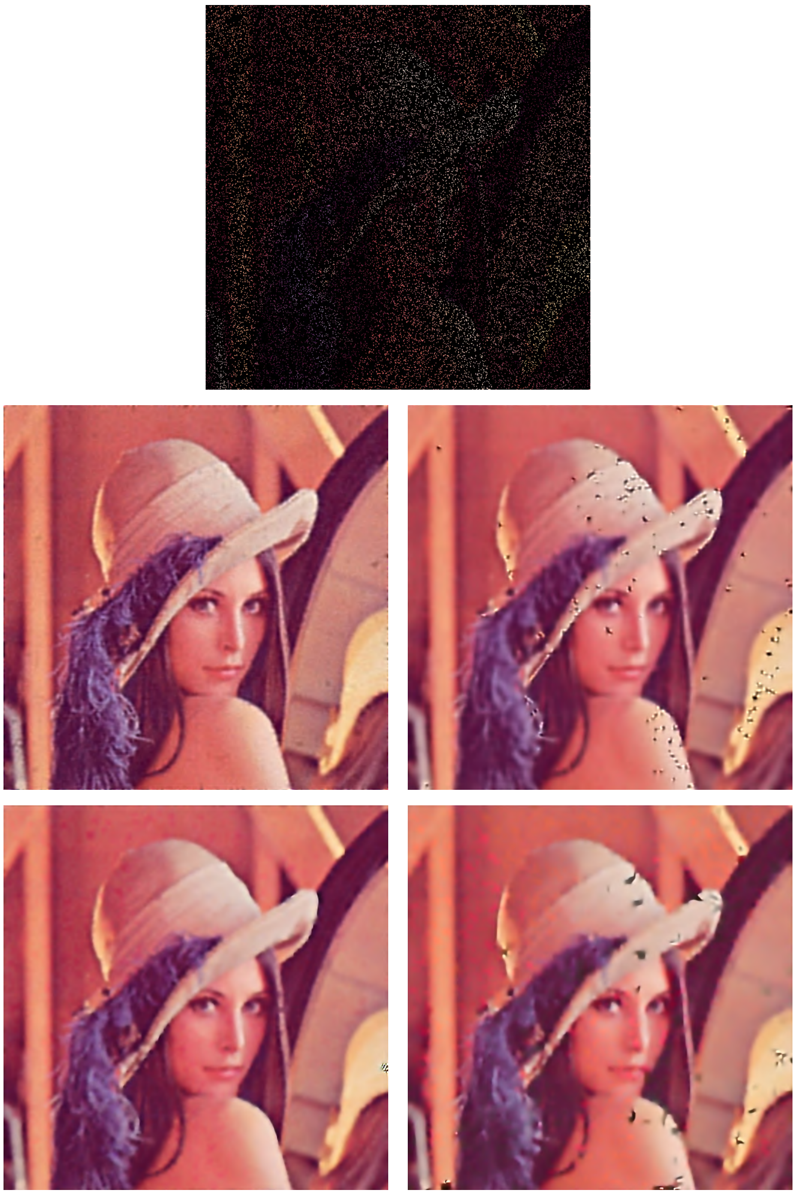

Figure 6: PnP recovery of noisy Lena with Gaussian blur and random subsampling $m \approx 0.1n$. First row: Subsampled Lena image that needs to be recovered. Second row: left: Recovery with $\mathrm{C}_{\mathrm{bad}}$, right: Recovery with $\mathrm{C}_{\mathrm{good}}$. Third row: left: Recovery with $\mathrm{U}_{\mathrm{bad}}$, right: Recovery with $\mathrm{U}_{\mathrm{good}}$. *Return to PnP: Good vs. Bad Denoiser or Better Denoiser $\not\Rightarrow$ Better PnP or PnP CNN denoiser vs PnP U-net denoiser.*

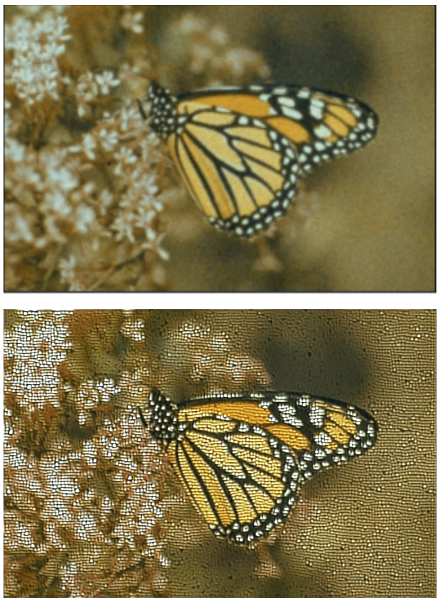

Figure 7: Denoiser performance and PnP recovery of noisy Monarch with Gaussian blur and random sub-sampling of $m \approx 0.2n$ using GAN denoiser. First row: Denosing performance of PnP GAN denoiser. Second row: Recovery using PnP GAN denoiser. *Return to Evaluating GANs.*

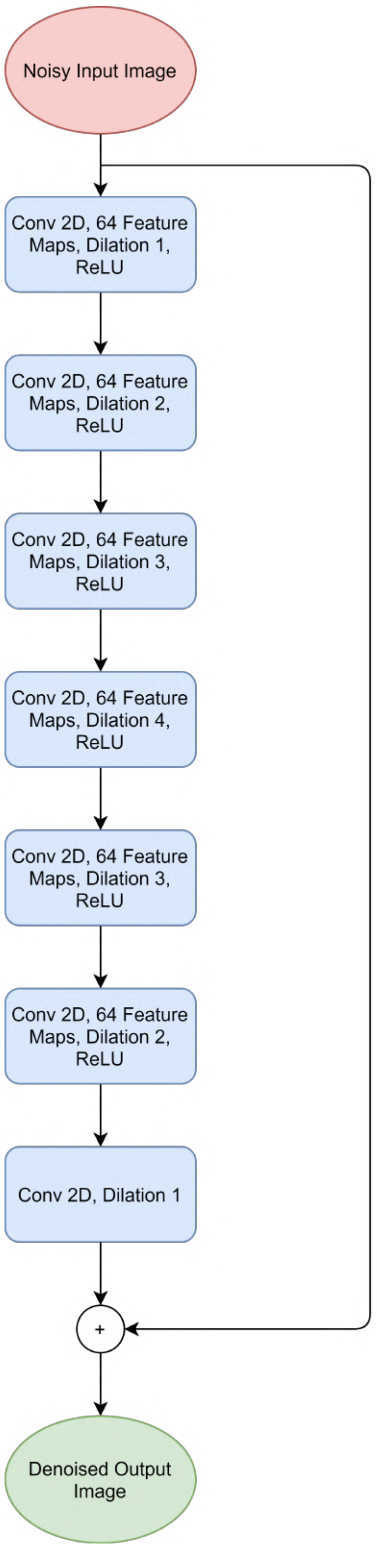
Noisy Input Image
Conv 2D, 64 Feature Maps, Dilation 1, ReLU
Conv 2D, 64 Feature Maps, Dilation 2, ReLU
Conv 2D, 64 Feature Maps, Dilation 3, ReLU
Conv 2D, 64 Feature Maps, Dilation 4, ReLU
Conv 2D, 64 Feature Maps, Dilation 3, ReLU
Conv 2D, 64 Feature Maps, Dilation 2, ReLU
Conv 2D, Dilation 1
+
Denoised Output Image

Figure 8: Model architecture of the CNN based PnP denoiser. *Return to PnP CNN Denoiser*.

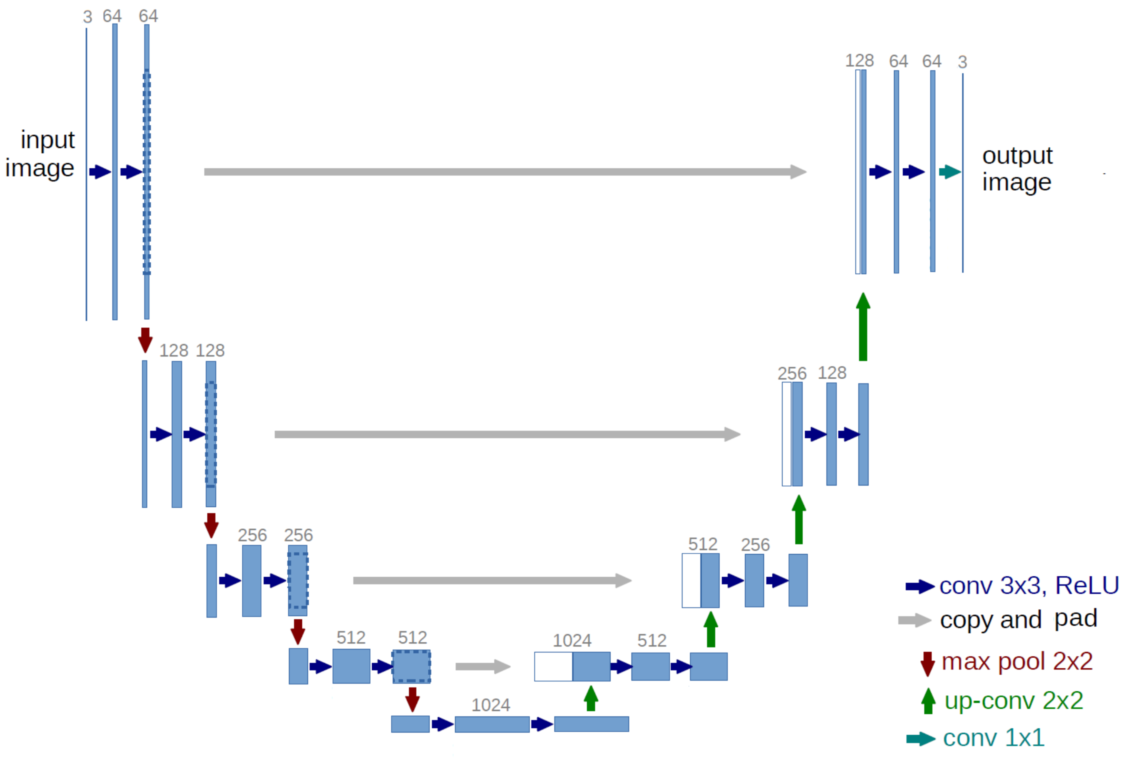

Figure 9: Model architecture of the U-net based PnP denoiser. *Return to PnP U-net Denoiser.*