POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Diffusion Models for Image Motion Blur Removal

Tesi di Laurea Magistrale in
Computer Science and Engineering

Author: **Lorenzo Innocenti**

Student ID: 977180
Advisor: Prof. Giacomo Boracchi
Co-advisors: Diego Stucchi
Academic Year: 2022-23

Abstract

Image deblurring is an image restoration technique that aims at recovering the sharp image corresponding to an observation affected by blur. Removal of blur is key in many applications, such as astronomy, microscopy, and digital photography. It has also been shown that deblurring greatly improves the performance of other computer vision systems, such as image recognition, classification, and segmentation. Deep learning models have proven effective in solving many image-to-image translation problems, meaning all those problems where images are both the input and the output of the model. Since image deblurring can be framed as such a problem, different studies in the literature have tried to approach it by deep neural networks. One of the key elements in deep learning is the dataset employed to train the neural network, as performance of the network are heavily influenced by how well the dataset represents the problem. For this purpose, we consider, for the first time in image restoration by deep learning models, a novel image-degradation pipeline to simulate the blur and noise resulting from the camera movement during the exposure process. Moreover, for the first time, we apply diffusion models to the problem of image deblurring. This new class of generative models has shown impressive results in various applications, and is becoming increasingly popular in the deep learning community. We train and test diffusion models on a state-of-the-art deblurring dataset and on our own image-degradation pipeline, to show the potential of the approach on both deblurring and noise suppression.

Keywords: image deblurring, diffusion models, image processing, image restoration, deep learning, computer vision

Abstract in lingua italiana

Il deblurring è una tecnica di restauro delle immagini che mira a recuperare l'immagine nitida da una osservazione affetta da sfocatura. La rimozione della sfocatura è fondamentale in molte applicazioni, come l'astronomia, la microscopia e la fotografia digitale. È stato inoltre dimostrato che il deblurring migliora notevolmente le prestazioni di altri sistemi di computer vision, come il riconoscimento, la classificazione e la segmentazione delle immagini. Il deep learning si è dimostrato efficace nel risolvere molti problemi di traduzione da immagine a immagine, ovvero tutti quei problemi in cui le immagini sono sia input che output del modello. Poiché il deblurring può essere inquadrato come un problema di questo tipo, diversi studi in letteratura hanno cercato di affrontarlo utilizzando reti neurali. Uno degli elementi chiave del deep learning è il dataset utilizzato per addestrare la rete neurale, poiché le prestazioni della rete sono fortemente influenzate dalla capacità del dataset di rappresentare il problema. A questo scopo, per la prima volta nel campo del deblurring di immagini mediante modelli di deep learning, consideriamo una nuova pipeline di degradazione delle immagini per simulare la sfocatura e il rumore risultanti dal movimento della fotocamera durante il processo di esposizione. Inoltre, per la prima volta, introduciamo l'applicazione dei modelli diffusivi al problema del deblurring. Questa nuova classe di modelli generativi ha dimostrato risultati notevoli in diverse applicazioni ed è sempre più diffusa nella comunità del deep learning. Alleniamo e testiamo modelli diffusivi su un dataset stato dell'arte per il deblurring, e su dataset generati dalla nostra pipeline, per mostrare il potenziale di questo approccio sulla rimozione della sfocatura e del rumore.

Parole chiave: deblurring di immagini, modelli diffusivi, elaborazione di immagini, ristorazione di immagini, deep learning, computer vision

Contents

1 | Introduction

Image restoration is the field of research that focuses on enhancing the visual quality of degraded images. Degradation can occur due to various factors such as noise, distortion, or compression artifacts during image acquisition, transmission, storage or downsampling, to name a few. Image deblurring is a technique used in image restoration that aims to recover the original sharpness of the picture from a blurry image. Many types of blur are possible, depending on the causes of such blur, for example camera shake, out of focus subject, or object motion.

The applications of deblurring are wide, such as astronomy, microscopy, and digital photography. Since the introduction of compact sensors in smartphones, and given the continuous increase in computational power of those devices, the focus has been shifting from relying on the camera hardware to implementing sophisticated algorithms of image restoration to improve the quality of pictures, and thus the interest in this field has been increasing. Image restoration is a notoriously ill-posed problem: many non degraded images can generate the same degraded image, for example, different image could result in the same downsampled half-resolution image. Moreover, in the deblurring scenario, the interaction between pixels are much more complex then in downsampling.

The relatively recent introduction of deep learning has revolutionized the field of computer vision. Neural networks have proven effective in many fields, such as classification, segmentation, and image-to-image translation, meaning all those problems where images are both the input and the output. Since image restoration can also be framed as an image-to-image translation, some studies tried to approach it with deep learning, obtaining sometimes better results than classic, non learning-based approaches.

One of the most important elements in deep learning is the dataset employed to train the neural network. The performance of a neural network are directly correlated with how well the dataset represents the problem. In other image restoration fields, dataset building is trivial. For denoising, for example, it's enough to synthesize an image by artificially adding noise to the image. For super resolution, the dataset can be generated by downsampling images. Generating a dataset for image deblurring can be an

intricate process. This is because various types of blurring effects can occur, and multiple types can overlap in a single image. For some of those, mathematical models exist and can be leveraged to generate a blurred image artificially. For other types, such a model does not exist yet. A large part of this study will be focused on developing image processing pipelines for degraded dataset generation, and another part will be instead focused on using that pipeline for neural network training. We release publicly our tool at `github.com/lorenzoinnocenti/csb-dataset-generator`.

Diffusion models are a relatively recent class of generative models. These models are based on the concept of diffusion, where an initial random signal is iteratively transformed to reach a target distribution, leveraging the power of denoising neural networks. Diffusion models have shown impressive results in various applications, including image and audio generation, and are becoming an increasingly popular research topic in the deep learning community. The impressive results obtained from the application of diffusion models in other traditional image restoration tasks, such as super-resolution, inpainting, and colorization, served as a significant inspiration for this study.

This thesis investigates the performance of diffusion models in the field of image deblurring. This study spawned from an article by Foi and Boracchi [1], which proposes an algorithm to generate realistic images affected by blur and noise. The main contributions of our work are:

- presentation of our synthetic blurring tool and its integration in neural network training;
- showing that diffusion models are indeed a promising tool in image deblur, and investigating which architecture is the most successful;
- employing our dataset generation tool to analyze the behaviour of our approach on different magnitudes of blur and noise.

1.1. Structure of the thesis

The thesis is structured as follows. In Chapter 2, we give a mathematical base for the problem, the basic rules, the assumptions, and the various subfields of image deblurring. Chapter 3 reports the related work. We describe the article [1] and the algorithm that we use for the dataset generation, and the most common deconvolution approach. Then, we explore the various approaches others have tried to solve the image deblurring problem, using deep learning. In Chapter 4, we illustrate the article that introduces diffusion models in the field of image generation, and a few modifications to the architecture that has been

proposed to improve generative performance. In Chapter 5 we present our contributions, starting with an in detail description of our blurring pipeline, and the integration in deep learning frameworks. We follow by describing the used model, the modification taken into account, and a way to deblur images with large resolutions. In Chapter 6 we illustrate our experiments, starting from the exploration of diffusion model architectures. Following, we compare the performance of our approach with other blind, deep learning based deblurring models. We then present our synthetic blurring pipeline, and we employ it analyze the behaviour of our approach on different magnitudes of blur and noise. The final chapter of this work, Chapter 7, offers concluding thoughts and insights.

2 | Problem formulation

We denote the degradation process attributable to blurring as a function of the sharp image $\mathbf{x}$:

$$\mathbf{y} = \Phi(\mathbf{x}), \tag{2.1}$$

where $\mathbf{y}$ is the blurred image, and Φ is the blurring process. If a model of the degradation process is available, we can represent it as

$$\mathbf{y} = \Phi(\mathbf{x}; \theta), \tag{2.2}$$

where θ is the vector of parameters that characterize the blur, given the degradation model. For the whole study, unless otherwise specified, both $\mathbf{x}$ and $\mathbf{y}$ are objects in $\mathbb{R}^{w \times h}$ if grayscale images, and $\mathbb{R}^{w \times h \times 3}$ if RGB.

2.1. Blur types

We hinted to the existence of multiple blur types, and thus multiple blur models. Real-life blur is usually a combination of these phenomena. In this section, we expand on this concept.

2.1.1. Out of focus blur

Out-of-focus blur is the type of blur that happens when the camera lens fails to focus the incoming light rays from a scene onto the image sensor. The magnitude of the effect is proportional to the distance between the focal point of the image and the camera sensor. Simulating out-of-focus blur can be achieved by convolving the image with a disk-shaped Point Spread Function (PSF) kernel. When an object is out of focus, the light from each point on the object creates a circular blur spot on the camera's image sensor or film plane. This circular blur spot is commonly known as the *circle of confusion*. By convolving the image with a disk PSF, each pixel is spread to neighboring pixels in a manner similar to

Figure 2.1: Example of out of focus picture. Notice how the small details, like the flowers, are spread in a circular manner.

how light gets spread by the camera aperture. The blur model equations are

$$\mathbf{y} = \mathbf{k} * \mathbf{x}, \tag{2.3}$$

$$\mathbf{k}(x, y) = \begin{cases} \frac{1}{\pi r^2} , & \text{if} (x-k)^2 + (y-l)^2 \leq r^2 \\ 0 , & \text{otherwise} \end{cases} \tag{2.4}$$

where $\mathbf{x}$ is the sharp image, $\mathbf{y}$ is the blurred image, $\mathbf{k}$ is the PSF kernel, (k, l) is the center of the PSF and r the radius of the blur. This formulation assumes that the whole scene is at the same distance to the camera, and therefore the blur is spatially invariant, meaning that the degradation is the same in every part of the image. For the whole study, unless otherwise specified, $\mathbf{k}$ is an object in $\mathbb{R}^{w \times h}$. In the case of convolutional blur, the parameters θ in 2.2 are the kernel $\mathbf{k}$.

Figure 2.2: Example of picture affected by camera shake blur. Notice how the small details are spread in an horizontal manner. Also notice how both the subject and the background are blurred in the same way.

2.1.2. Camera shake blur

Camera shake blur is a type of blur that occurs when a camera is moved during the acquisition process. The severity of camera shake blur can vary, depending on the degree and direction of camera movement, the speed of the shutter, and other factors. Given the labeling of the camera axis in Figure 2.3, a common assumption that can be made is that the camera rotation only occurs along the x and y axes, and not on the z axis. Under this assumption, the blur effect can be considered spatially invariant, and therefore modeled with a convolutional process using a PSF kernel. If we want to take into account rotation along z, the model must include spatially variant blur, so a convolutional model is no longer possible. The PSF kernel, in this scenario, represents the trajectory of the movement of the camera during the sensor exposure period. This degradation process is

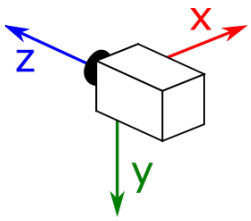


Figure 2.3: Graphical representation of the camera axes.

based on the concept that, when an image is convolved with the PSF kernel, each pixel is spread along the trajectory of the camera movement, simulating the way light is imprinted on the sensor during the motion. In this case, the PSF kernel is not as trivial to synthesize as the disk kernel of the out-of-focus blur, as it can take into account complex motion patterns, depending on the desired realism of the blur. One common simplification is the assumption of linear rotation of the camera, and therefore a kernel composed of a straight line. Others have tried creating algorithms to generate more realistic and complex kernels. Another approach is to record camera motion via external hardware, and use this data to generate PSF kernels.

In this thesis, in contrast to what other deep learning deblurring studies do, additionally to the camera shake blur degradation, we also include a Poisson and Gaussian noise component in the degradation process, as proposed in [1]. Poisson noise is a model of the photon acquisition of the sensor, and the noise associated with this kind of process, called shot noise. It is is expressed as

$$\mathbf{u} \sim \mathcal{P}(\lambda(\mathbf{k} * \mathbf{x})), \tag{2.5}$$

where λ is a parameter that quantifies the quantum efficiency of the sensor and $\mathcal{P}$ is the poisson distribution. Gaussian noise is a model of the noise caused by the electronic amplification of the signal. It is added to the result of the application of poisson noise:

$$\mathbf{y} = (\mathbf{u} + \mathbf{n})/T, \tag{2.6}$$

$$\mathbf{n} \sim \mathcal{N}(0, \sigma^2), \tag{2.7}$$

where σ is a parameter that quantifies the thermal noise of the sensor, $\mathcal{N}$ is the gaussian distribution. In this degradation model, the parameter T represents the exposure time, and controls the length of the trajectory of the blurring kernel. It is the same value as the sum of the kernel pixels. The authors use this term to simulate the tradeoff between blur and noise present in cameras: when T is small, the signal range is shrunk, increasing the noise effect, and reducing the blur magnitude. The multiplication by $1/T$ in (2.6) is an amplification factor that serves to restore the full dynamic range of the image. This

Figure 2.4: Example of picture affected by object motion blur. Notice how only the subject is blurred, while the background is sharp.

effect simulates the same tradeoff between blur and noise that we can find in cameras. A more in-detail explanation of this process is in Section 3.1.

2.1.3. Object motion blur

Object motion blur is a type of blur that occurs when an object in the scene being photographed is in motion during the exposure period of the camera. The degree and direction of the blur depend on several factors, including the speed and direction of the object's motion, the distance between the object and the camera, and the exposure time. This is the most complex type of blur, as its effect is a composition of multiple differently moving parts. It is typically spatially variant, meaning that the degradation varies in different parts of the image. A simple mathematical model is not possible, so different methods must be used to generate images with this characteristic.

2.2. Deblurring approaches

Depending on whether we have a mathematical model that describes the degradation or not, the inverse problem can be subdivided in two fields. When there is a model for the blur, and the blurring parameters are known, it is possible to use *non-blind* approaches. When the model is not known, or if it is known but the parameters are not, a *blind* approach is necessary.

Non-blind approaches assume the blur as a known model, characterizable with a vector of parameters θ, and assume those parameters to be known. It can be represented as

$$\tilde{\mathbf{x}} = \Phi^{-1}(\mathbf{y}; \theta), \tag{2.8}$$

where $\tilde{\mathbf{x}}$ is the estimated deblurred image, with same resolution as $\mathbf{x}$ and $\mathbf{y}$. This category encompasses not only deconvolution methods but also the inversion of non-convolution-based models. Moreover, the performance of these methods are directly proportional on how well the model and the parameters represents the degradation. If the accuracy is high, they generally perform better than methods without model assumptions.

Blind approaches include all other deblurring methods that do not conform to the assumptions outlined in the previous section, which can happen in two cases. In some settings, the model is known but the parameters θ are not. When the model is known, a possible approach is to use a parameter estimator to estimate θ, and then use a non-blind approach to estimate $\tilde{\mathbf{x}}$:

$$\tilde{\theta} = \Psi(\mathbf{y}), \tag{2.9}$$

$$\tilde{\mathbf{x}} = \Phi^{-1}(\mathbf{y}; \tilde{\theta}), \tag{2.10}$$

where Ψ is the parameters estimator. The performance of these methods are strictly tied to the accuracy of the considered model. In other settings, the blurring model is not known, or is not possible to build one, for example if the blurring types are multiple in the same image. In this case, the only method possible is to estimate the sharp image directly from the blurred image:

$$\tilde{\mathbf{x}} = \Phi^{-1}(\mathbf{y}). \tag{2.11}$$

2.3. Metrics

Multiple metrics are available to assess the quality of restored images. We introduce three of them in the next sections, and we use them to compare the methods in our work.

The two most commonly used are Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM). In addition to those, we also introduce a more recent metric that is more closely aligned with the field of deep learning, Learned Perceptual Image Patch Similarity (LPIPS).

2.3.1. PSNR

The PSNR is the most popular metric used in the field of image reconstruction. It is calculated as the ratio of the peak signal power to the mean squared error (MSE) between the original and reconstructed signals. Due to the wide dynamic range of images, it is usually expressed in dB. It is computed as

$$\mathrm{PSNR}(\mathbf{x},\mathbf{y}) = 10 \cdot \log_{10}\left(\frac{\mathrm{MAX}^2(\mathbf{x})}{\mathrm{MSE}(\mathbf{x},\mathbf{y})}\right), \tag{2.12}$$

where $\mathrm{MAX}(\mathbf{x})$ is the maximum pixel value of the sharp image, and $\mathrm{MSE}(\mathbf{x},\mathbf{y})$ is the mean squared error between the original image and the degraded one. In the case of RGB images, it is computed as

$$\mathrm{MSE}(\mathbf{x},\mathbf{y}) = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{k=1}^{3}(\mathbf{x}(i,j,k) - \mathbf{y}(i,j,k))^2 \tag{2.13}$$

where M and N are the image dimensions. This value is straightforward, easy to implement, and widespread in image processing, but has the downside that it only takes into account per pixel differences. If, for example, the reconstruction is slightly shifted, the PSNR can change greatly.

2.3.2. SSIM

The SSIM [2] is based on three main components: luminance, contrast, and structural similarity. The luminance component measures the similarity in brightness between the two images, the contrast component measures the similarity in contrast, and the structural similarity component measures the similarity in texture and edges. These components are computed as

$$l(\mathbf{x},\mathbf{y}) = \frac{2\mu_{\mathbf{x}}\mu_{\mathbf{y}} + c_1}{\mu_{\mathbf{x}}^2 + \mu_y^2 + c_1}, \tag{2.14}$$

$$c(\mathbf{x},\mathbf{y}) = \frac{2\sigma_{\mathbf{x}}\sigma_{\mathbf{y}} + c_2}{\sigma_{\mathbf{x}}^2 + \sigma_{\mathbf{y}}^2 + c_2}, \tag{2.15}$$

$$s(\mathbf{x},\mathbf{y}) = \frac{\sigma_{\mathbf{xy}} + c_3}{\sigma_{\mathbf{x}}\sigma_{\mathbf{y}} + c_3}, \tag{2.16}$$

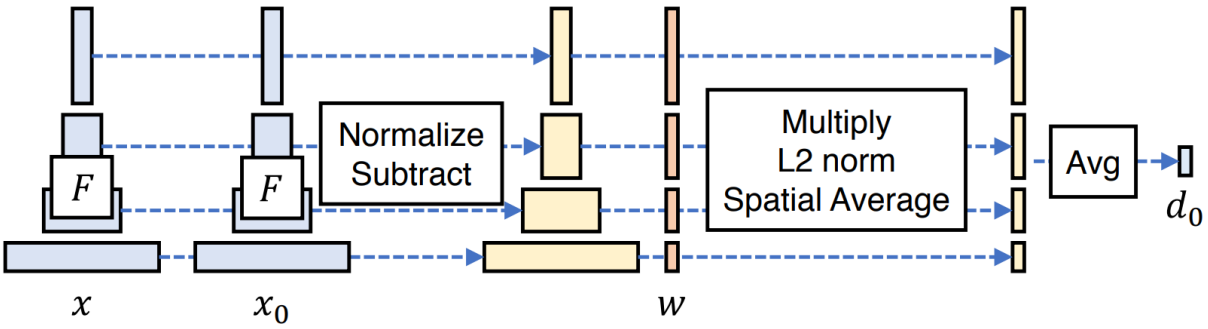


Figure 2.5: Illustration of the LPIPS computation algorithm, from [3].

where $\mathbf{x}$ and $\mathbf{y}$ are two input images, $\mu_\mathbf{x}$ and $\mu_\mathbf{y}$ are the mean values of $\mathbf{x}$ and $\mathbf{y}$, $\sigma_\mathbf{x}^2$ and $\sigma_\mathbf{y}^2$ are their respective variances, and $\sigma_\mathbf{xy}$ is the covariance between $\mathbf{x}$ and $\mathbf{y}$. Moreover, $c_1 = (k_1 L)^2$, $c_2 = (k_2 L)^2$ and $c_3 = c_2/2$ are constants added to avoid division by zero. The value of L is the dynamic range of the pixel values, typically $2^b - 1$, where b is the number of bits per pixel. Finally, $k_1 = 0.01$ and $k_2 = 0.03$ unless otherwise specified.

The overall SSIM index is the product of these three components:

$$\begin{aligned} \mathrm{SSIM}(\mathbf{x}, \mathbf{y}) &= l(\mathbf{x}, \mathbf{y}) \cdot c(\mathbf{x}, \mathbf{y}) \cdot s(\mathbf{x}, \mathbf{y}) \\ &= \frac{(2\mu_\mathbf{x}\mu_\mathbf{y} + c_1)(2\sigma_\mathbf{xy} + c_2)}{(\mu_\mathbf{x}^2 + \mu_\mathbf{y}^2 + c_1)(\sigma_\mathbf{x}^2 + \sigma_\mathbf{y}^2 + c_2)} \end{aligned} \tag{2.17}$$

The SSIM takes into account the more complex interactions between pixels, which is important for assessing the quality of the restored image. This gives a different point of view on reconstruction quality than PSNR, and therefore these two metrics are often used in tandem.

2.3.3. LPIPS

In addition to the traditional metrics, we also introduce a more modern one, which is more closely aligned with the field of deep learning, LPIPS [3]. Unlike conventional image similarity metrics, LPIPS is designed to emulate the way humans perceive differences between images by considering features or patterns within the image. The metric employs a deep neural network that has been trained on a vast image dataset to extract these features, and subsequently measures the similarity between the features of the two images. In practice, LPIPS is computed as a scaled L2 distance between the activation values of a vgg19 *conv3.3* layer, trained for classification. The complete algorithm is illustrated in Figure 2.5. It begins by preprocessing the two images, and extracting the features with the pretrained neural network. From the feature representation, the distance is computed

Figure 2.6: Samples of random image patches from the ImageNet dataset [4].

as

$$\mathrm{d}(\hat{\mathbf{x}}, \hat{\mathbf{y}}) = \sum_l \frac{1}{HW} \sum_{hw} \|w_l \odot (\hat{\mathbf{x}}(h,w,l) - \hat{\mathbf{y}}(h,w,l))\|^2, \tag{2.18}$$

where $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ are, respectively, the features extracted from $\mathbf{x}$ and $\mathbf{y}$, in the form of 3D matrices of sizes $H \times W \times L$, and w_l are a set of weights. The values of the weights are trained though gradient descent, on a dataset made of image couples and human similarity scores, which is the percentage of humans that considers the image as the same, despite the distortions.

The authors of [3] demonstrate that LPIPS is a more accurate reflection of human perception similarity when compared to non-deep learning based metrics. LPIPS is not as widely used in other image restoration studies as PSNR and SSIM, so we utilize this metric mainly for comparisons between our results. LPIPS is also used as partial loss in some training settings, for example in [7], which is explained in Section 3.3.3.

Figure 2.7: Samples of random image patches from the MS COCO dataset [5].

2.4. Datasets

2.4.1. ImageNet

The ImageNet dataset is introduced in [4]. It consists of a collection of images of objects, animals and people, organized by semantic hierarchy, and labeled as one of the 80000 classes of the set. It is a dataset of images gathered for training of image recognition neural networks. Since this dataset primarily consists of sharp images, it is suitable for use when a synthetic degradation model is present. In such a scenario, the images from ImageNet can be artificially blurred, and a neural network can be trained to reverse the degradation. Since the images are scraped from various websites, the dataset has no bias induced by the use of a single camera.

2.4.2. MS COCO

The MS COCO Dataset is introduced in [5]. It contains 328.000 images of complex everyday scenes, segmented and labeled to aid the training of object localization an similar problems. It also contains sharp images scraped from internet, and it can be used in a

Figure 2.8: Samples of random image patches from the GoPro dataset [6].

same way as the ImageNet dataset.

2.4.3. GoPro

The GoPro dataset is introduced in [6]. The authors captured multiple videos with an high framerate 240 fps camera, and averaged a number of frames between 13 and 7 to get a blurred image. The middle frame is taken as the corresponding ground truth. The blur in this dataset is a mix of camera motion and object motion. The main drawback of this dataset is the bias introduced by the use of a single camera: all the images have similar light, contrast and color balance. The dataset contains two subsets, depending on the type of image averaging. We use the *linear* subset for this thesis.

3 | Related work

3.1. Camera shake blur image generation

In this section, we explain the image degradation algorithm proposed in [1]. This article presents a first part about generating a realistic image degradation model, which is the one that we introduced in 2.1.2, and a second part on testing reconstruction algorithms.

The article proposes an algorithm for the generation of images affected by camera shake blur and noise. A trajectory is generated by simulating the motion of a particle in a 2D domain. The particle has an initial velocity and, at each iteration of the generation algorithm, is affected by a random perturbation and by a centripetal component. In addition, with small probability, a random inversion of direction can happen. This trajectory is then sampled in a 2D grid, generating the PSF kernel. The kernel is applied to the image by convolution. Then, the image is further degraded by applying Poisson and Gaussian noise, as in (2.6).

We describe the full degradation process in the three main steps: the trajectory generation, the trajectory sampling and the synthetic image degradation, that simulates both noise and blurring.

3.1.1. Trajectory generation

The trajectory generation algorithm is represented in Algorithm 3.1. It takes as inputs the parameters that influence the behavior of the trajectory and outputs the trajectory, represented as an array of complex numbers. The trajectory is later sampled in a 2D matrix by the PSF kernel generation algorithm. It has, as inputs:

- h: kernel size, size in pixels of the canvas in which the trajectory is later be drawn, i.e. the PSF matrix size.
- $a \in [0, 1]$: anxiety, scales the random vector that is added at each sample. Also influences the probability of *big shakes*, a phenomenon that randomly inverts the direction of the movement;

Algorithm 3.1 trajectory generation

1: **function** TRAJ_GENERATION($h, a, \eta_t, L, c, g, f_s$)
2: $\theta \sim U(0, 2\pi)$
3: $v = \cos(\theta) + i \times \sin(\theta)$
4: $v = v \times a$
5: **for** $t = 0$ **to** $\eta_t - 1$ **do**
6: $u \sim U(0, 1)$
7: **if** $u < a \times f_s$ **then**
8: $n_d = 2 \times v \times e^{i \times (\pi + rand - 1/2)}$
9: **else**
10: $n_d = 0$
11: **end if**
12: $a, b \sim N(0, 1)$
13: $g = g \times (a + ib)$
14: $dv = n_d + a \times (g - c \times x_t) \times \frac{L}{\eta_t - 1}$
15: $v = v + dv$
16: $v = \frac{v}{|v|} \times \frac{L}{\eta_t - 1}$
17: $\mathbf{j}[t + 1] = \mathbf{j}[t] + v$
18: **end for**
19: $c = h/2$
20: $d_x = \overline{(Re(x))} - c$
21: $d_y = \overline{(Im(x))} - c$
22: $\mathbf{j} = \mathbf{j} - d_x$
23: $\mathbf{j} = \mathbf{j} - i \times d_y$
24: **return j**
25: **end function**

- η_t: resolution of the trajectory curve, i.e. the number of samples of the curve;
- L: maximum length in pixels of the trajectory, computed as the sum of all the distances between consecutive points;
- $c \in [0, 1]$: centripetal parameter, reoresents the strength of the pull towards the previous point;
- $g \in [0, 1]$: gaussian parameter, determines the randomness of the movement in each step;
- $f_s \in [0, 1]$: frequency of big shakes.

The output is an array of complex numbers, $\mathbf{j}$, that represent the samples of the curve. The real part encodes the x coordinate, and the imaginary part encodes the y coordinate.

The algorithm is developed as follows. Firstly, on lines 2-4, a random initial direction for the vector v is chosen. Then, a for loop is started on line 5 to iterate through the

Algorithm 3.2 psf generation

1: **function** PSF_GENERATION($\mathbf{j}, T$)
2: $\mathbf{k}$ = 2D matrix of zeros
3: **for** $t = 0$ **to** $\eta_t - 1$ **do**
4: **if** $T \times \eta_t \geqslant t + 1$ **then**
5: $t_p = 1$
6: **else if** $T \times \eta_t \geqslant t$ **then**
7: $t_p = (T \times \eta_t) - t$
8: **else**
9: $t_p = 0$
10: **end if**
11: $m_1 = floor(Re(\mathbf{j}[t]))$
12: $m_2 = floor(Im(\mathbf{j}[t]))$
13: $M_1 = ceil(Re(\mathbf{j}[t]))$
14: $M_2 = ceil(Im(\mathbf{j}[t]))$
15: $\mathbf{k}[m_1, m_2] = \mathbf{k}[m_1, m_2] + t_p \times tri\_prod(Re(\mathbf{j}[t]) - m_2, Im(\mathbf{j}[t]) - m_1)$
16: $\mathbf{k}[m_1, M_2] = \mathbf{k}[m_1, M_2] + t_p \times tri\_prod(Re(\mathbf{j}[t]) - m_2, Im(\mathbf{j}[t]) - m_1)$
17: $\mathbf{k}[M_1, m_2] = \mathbf{k}[M_1, m_2] + t_p \times tri\_prod(Re(\mathbf{j}[t]) - m_2, Im(\mathbf{j}[t]) - m_1)$
18: $\mathbf{k}[M_1, M_2] = \mathbf{k}[M_1, M_2] + t_p \times tri\_prod(Re(\mathbf{j}[t]) - m_2, Im(\mathbf{j}[t]) - m_1)$
19: **end for**
20: **return k**
21: **end function**
22:
23: **function** TRI_PROD(d_1, d_2)
24: $x_1 = max(0, (1 - |d_1|))$
25: $x_2 = max(0, (1 - |d_2|))$
26: **return** $x_1 \times x_2$
27: **end function**

values of t. Throughout each iteration, the algorithm has the chance to make a random determination on whether to perform a big shake (lines 6-11). If so, it computes the term n_d that is used to invert the direction later. The new direction is picked randomly in the range $-v \pm 1$ in radians. The algorithm then proceeds to compute the g term, which contains the random component of the direction update (lines 12-13). Next, it updates the direction vector v with the gaussian component, the centripetal component, a normalization term, and the potential direction inversion (lines 14-15). The algorithm then normalizes v again and updates the vector $\mathbf{j}$ with the next value (lines 16-17). Finally, after the computation of all the terms of $\mathbf{j}$ from 0 to η_t, the trajectory is centered to $h/2$ (lines 19-23).

Algorithm 3.3 blur image and add noise

1: **function** BLUR_IMAGE($\mathbf{x}, \mathbf{k}, \lambda, \sigma$)
2: $\mathbf{x} = \mathbf{x} \times \lambda$
3: $\mathbf{z} = \mathbf{x} * \mathbf{k}$
4: $\mathbf{n} \sim \mathcal{N}(0, \sigma)$
5: $\mathbf{u} \sim \mathcal{P}(\mathbf{z})$
6: $\mathbf{y} = \mathbf{u} + \mathbf{n}$
7: **return y**
8: **end function**

3.1.2. PSF kernel generation

The PSF kernel generation algorithm is represented in Algorithm 3.2. It samples the trajectory array in a 2D matrix, also called canvas, that will be use to compute the convolution with an image to simulate camera shake blur. It takes, as inputs:

- **j**: trajectory generated by Algorithm 3.1, in a complex array form;
- T: [0,1] term that simulates the exposure time. It determines the percentage of the trajectory that is sampled.

The output is the matrix of pixels of the PSF, **k**.

The algorithm proceeds as follows. First, the kernel matrix is initialized to zeros on line 2. Then, a for loop is started on line 3 to iterate through the values of t. The purpose of this loop is to compute the value of the t_p term, which scales the intensity of the next sample. This is done in lines 4-10, in such a way that the sum of all the values of **k** amounts to T. After this step, the value of $\mathbf{j}[t]$ is used to compute the value of the four closest pixels to the sample using linear interpolation. This is achieved in lines 11-18.

3.1.3. Synthetic blurring and degradation

The kernel generated in the previous two algorithms is used in the third algorithm, 3.3. It has, as inputs:

- **x**: the image to be processed;
- **k**: the kernel generated with the previous algorithms;
- λ: parameter of the poisson distribution, that simulates one of the noise components of the image degradation process;
- σ: parameter of the gaussian distribution, that simulates the other noise component.

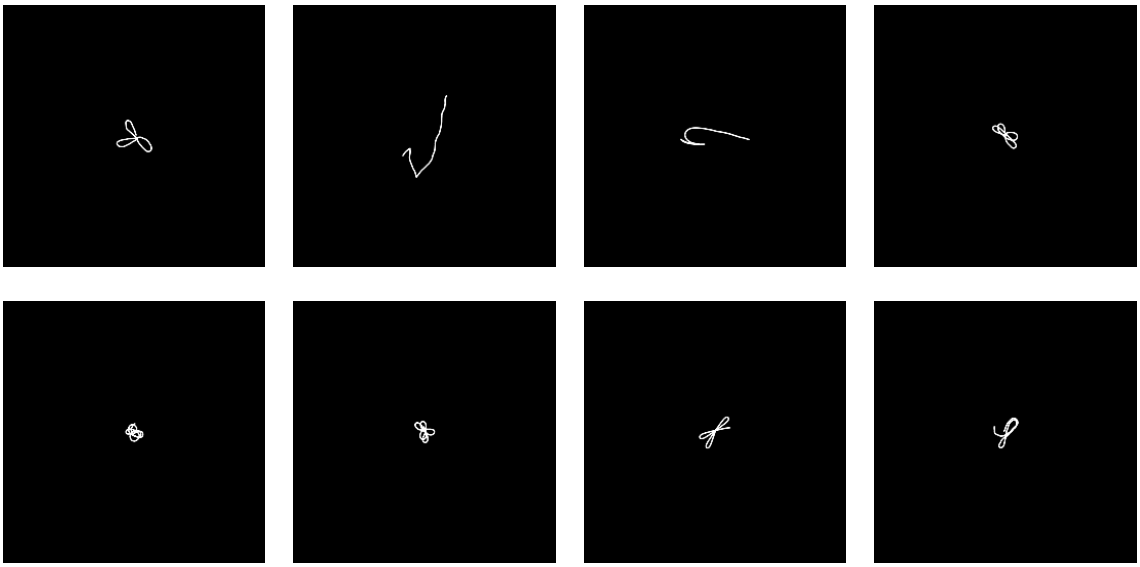


Figure 3.1: Examples of kernels generated by the official MATLAB implementation of [1].

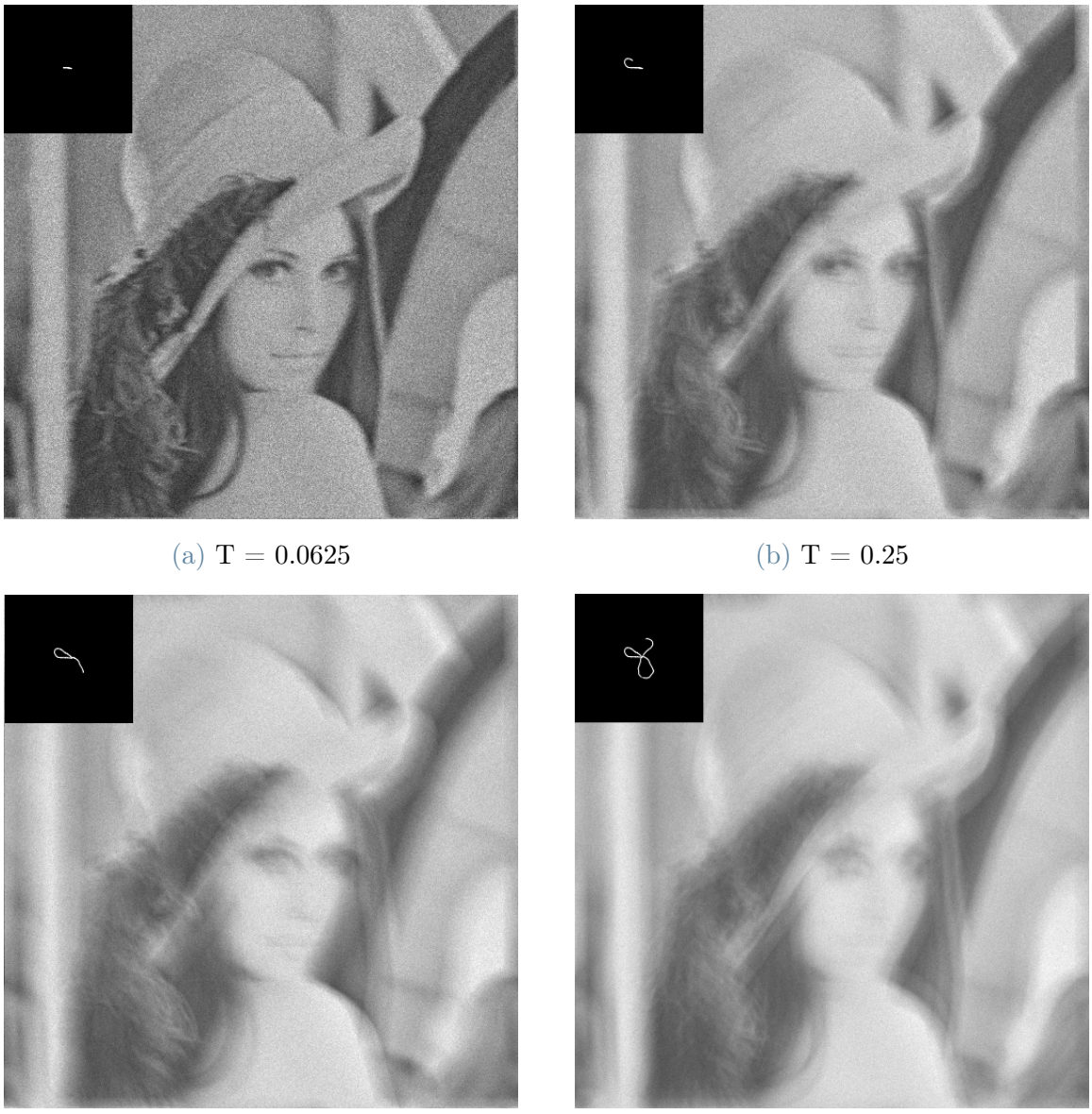
The output is the degraded image.

The algorithm proceeds as follows. First, on line 2, the image is scaled with the λ parameter. This scaling is be used for the Poisson noise, which is applied on line 6. Next, on line 3, the image is convolved with the PSF matrix. The algorithm then generates Gaussian noise on line 4. On line 5, Poisson noise is applied to the image. Finally, on line 6, Gaussian noise is added to the image.

The full algorithm is implemented by the authors in a MATLAB package. Some sample PSF kernels are depicted in Figure 3.1. An example of the application of the blur process is in Figure 3.2. In the image, the same trajectory is used to generate kernels with various values of the exposure time T, meaning that a shorter portion of the trajectory is sampled. A lower value of T means that the image gets diminished in value, so the noise contribution is greater. A bigger value of T means a longer trajectory, so the image gets blurred more. The images are depicted with the corresponding kernel, and are ordered by ascending value of T.

3.1.4. The noise-blur tradeoff

The authors analyze the performance of the reconstruction of some non-blind, non deep learning based deconvolution approaches. The aim is to study how the reconstruction error developes with varying values of T. To do so, they degrade six the test images with a set of six different trajectories, at a fixed noise level of $\lambda = 3000$ and $\sigma = 0$, with varying T values. They use one linear trajectory and five trajectories created with the camera



(a) T = 0.0625 (b) T = 0.25

(c) T = 0.5 (d) T = 1

Figure 3.2: Example of the application of the camera shake degradation implemented in the algorithm from [1] to the Lenna test image.

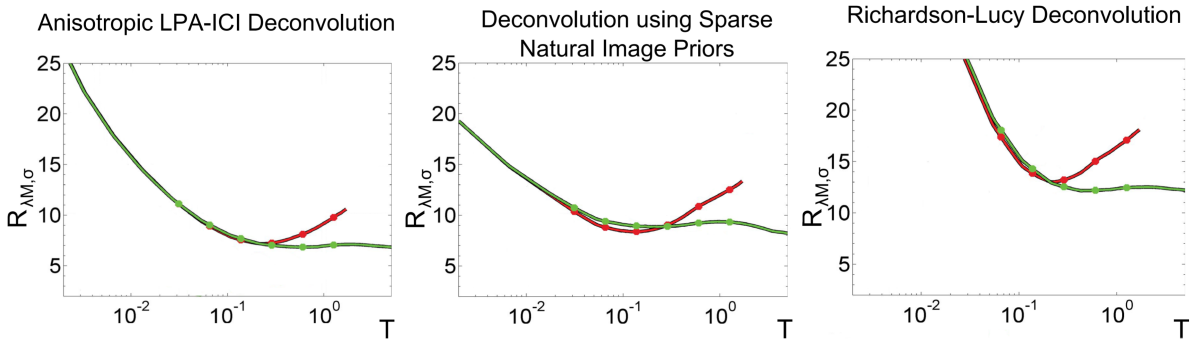


Figure 3.3: Results from [1] of the experiment reported in Section 3.1.4. In red the RMSE of the reconstruction of images degraded with a linear trajectory, in green with camera shake trajectory, restored using three different deconvolution algorithms. Lower is better.

shake blur kernel generation algorithm. They deconvolve the images using non deepl learning based deconvolution algorithms and show the variation of RMSE as T varies.

What they find is that there are two distinct behaviors, depending on the PSF kernel. If the blur trajectory is linear, the reconstruction presents an optimum value of T in witch the reconstruction is the best. Otherwise, if the PSF trajectory is more complex, the reconstruction levels off after a certain T value. Such behavior is reported in Figure 3.3.

3.2. Wiener deconvolution

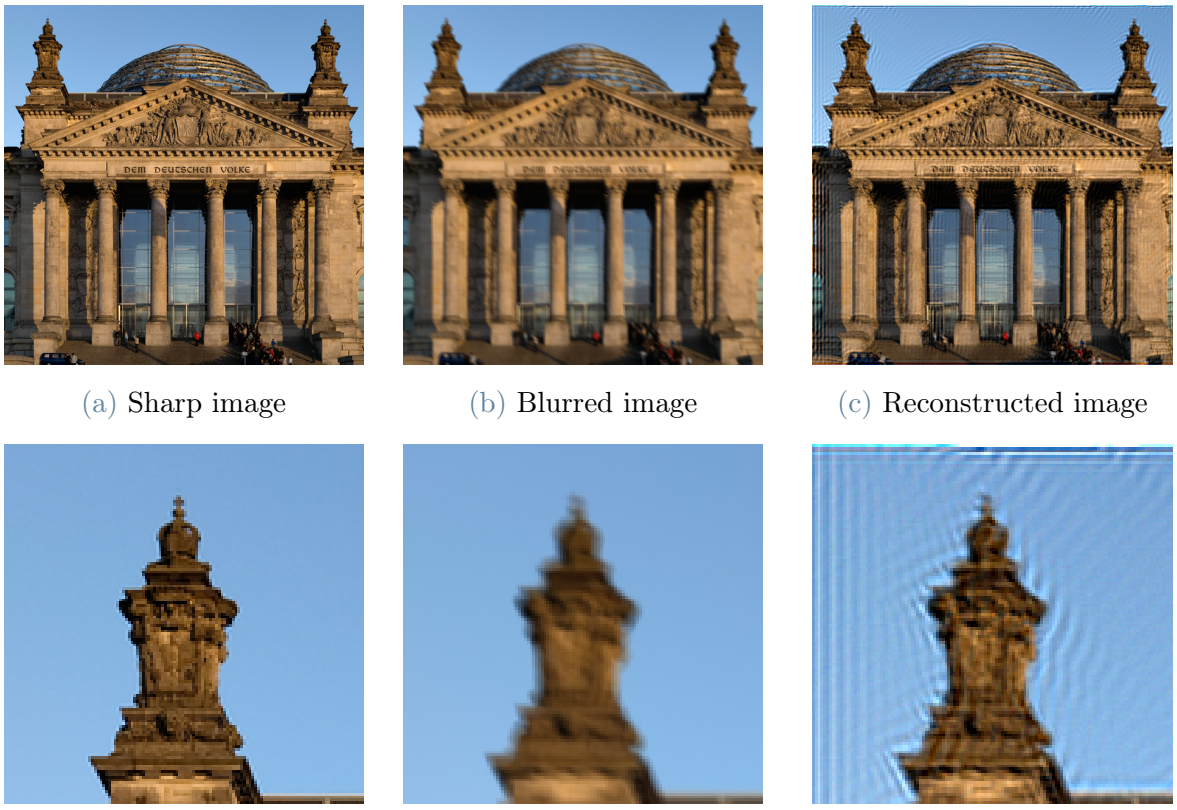
Wiener deconvolution is the most famous non-blind, non machine learning based technique used to restore a signal from a convolutional degradation. It is introduced by Norbert Wiener in [8], and has since been widely used in signal deconvolution problems. It assumes, as degradation model,

$$\mathbf{y} = \mathbf{k} * \mathbf{x} + \mathbf{n}, \tag{3.1}$$

where $\mathbf{x}$ is the sharp image, $\mathbf{y}$ is the degraded image, $\mathbf{k}$ is the PSF kernel, and $\mathbf{n}$ is additive noise. The Wiener deconvolution consists is a convolution operation with a filter that can be computed as

$$\mathbf{k}^{\dagger} = \mathcal{F}^{-1}\left(\frac{1}{K} \cdot \frac{|K|^2}{|K|^2 + r}\right), \tag{3.2}$$

where $K = \mathcal{F}(\mathbf{k})$, and $\mathcal{F}$ is the Fourier transform operator. The parameter r is a regularization parameter, to be tuned to get the optimal reconstruction. If the noise power spectral density is known, its value is $r = \frac{|N|^2}{|Y|^2}$, where $Y = \mathcal{F}(\mathbf{y})$, $N = \mathcal{F}(\mathbf{n})$. The filter is



(a) Sharp image (b) Blurred image (c) Reconstructed image

(d) Sharp image, zoom (e) Blurred image, zoom (f) Reconstructed image, zoom

Figure 3.4: Example of Wiener deconvolution. The second row is a zoom on the images to highlight the ringing effects.

usually applied in the Fourier domain:

$$\tilde{\mathbf{x}} = \mathcal{F}^{-1}\left(\frac{1}{K} \cdot \frac{|K|^2}{|K|^2 + r} \cdot Y\right). \tag{3.3}$$

If we assume $\mathbf{n} \sim N(0, \sigma^2)$, then $|N|^2 = w \times h \times \sigma^2$. Images obtained using Wiener deconvolution are usually affected by ringing artifacts, which are dark and light ripples around bright features of the image. Such artifacts are depicted in Figure 3.4.

3.3. Deblur in deep learning

3.3.1. Non-blind deblurring

The assumption behind non-blind approaches is that the degradation process can be modeled as a known mathematical operation, and that the degradation parameters are also known. Most of these approaches feature convolutional blur, but other models are

possible.

Deep Convolutional Neural Network for Image Deconvolution

In [9], Xu et al. propose the first deep convolutional neural network approach for image deconvolution. In the article, the noise component is $\mathbf{n} \sim N(0, \sigma^2)$. The authors aim to address the limitations of traditional methods such as the Wiener deconvolution, which are either computationally expensive or produce reconstruction artifacts. The main idea is to model a network that is initialized to compute the Wiener deconvolution, and fine tune it on pairs of synthetically blurred and sharp images, to train it to avoid artifacts.

To keep the computational cost low, matrix $\mathbf{k}^\dagger$ is decomposed using SVD [10]:

$$\mathbf{k}^\dagger = \mathbf{USV}^T, \tag{3.4}$$

$$\mathbf{k}^\dagger * \mathbf{y} = \sum_j s_j \cdot \mathbf{u}_j \left(\mathbf{v}_j^T * \mathbf{y}\right), \tag{3.5}$$

where $\mathbf{u}_j$ and $\mathbf{v}_j$ are the j^{th} columns of $\mathbf{U}$ and $\mathbf{U}$, and s_j is the j^{th} singular value. This shows that a big 2D convolution can be expressed with a weighted sum of separable 1D filters, and in practice some of those can be dropped due to a low value of s_j.

Figure 3.5 reports the network architecture used in [9]. It consists of a convolutional layer of size 121×1 and 38 filters, initialized with values from $\mathbf{v}_j$ followed by an activation layer. A second convolutional layer of size 1×121 and 38 filters, initialized with values from $\mathbf{u}_j$, followed by an activation layer. A last convolutional layer of size 1×1 and 38 filters, initialized with values from s_j.

The network is trained with as input synthetically blurred images, and as target the sharp version. A denoising CNN composed by two 16×16, 512 filters layers, called *Outlier Rejection Sub-Network*, is added in cascade after the deblurring network. The last layer of the Deconvolution Sub-Network can be fused with the first from the Outlier Rejection Sub-Network, obtaining the architecture shown in Figure 3.5. The two networks are trained separately and then fine tuned together.

The results obtained with this model are better than the Wiener deconvolution, and reduces greatly the Wiener ripple artifacts. The main drawback of this approach is that a new model must be trained for each different kernel, effectively reducing the general applicability.

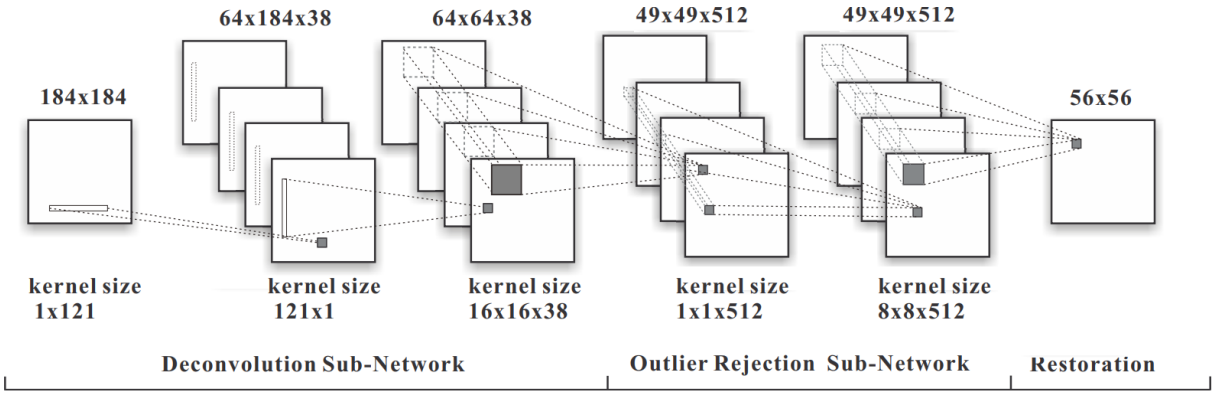


Figure 3.5: Graphical representation of the architecture of the network in [9].

Deep Image Prior

In [11], the authors introduce a new approach to image restoration and inpainting using deep convolutional neural networks. The basic idea of the Deep Image Prior approach is to use a deep convolutional neural network with randomly initialized weights as a prior for natural images. In [11], deblurring is not addressed, but others have later adapted Deep Image Prior in approaching that problem. The network is an encoder-decoder architecture with, as input, a random 32-channel tensor of the same size as the desired output. At each epoch, the network is trained via gradient descent, to produce the distorted image from the noise tensor. What the authors notice is that, by stopping the training before the complete convergence, the image produced as output has less distortions. The training is represented as the minimization of a distance term E which depends on the problem

$$\theta^* = \underset{\theta}{\operatorname{argmin}} E\left(f_\theta(\mathbf{z}); \mathbf{y}\right), \tag{3.6}$$

where f_θ is the model transformation, $\mathbf{z}$ is the random input tensor, $\mathbf{y}$ is the distorted image, θ are the model parameters, θ^* are the parameters after training.

To adapt the method to image deconvolution, using the blur model as in Equation 3.1 as an example for the formulation, the term E is defined as

$$E\left(f_\theta(\mathbf{z}); \mathbf{y}\right) = \left\| f_\theta(\mathbf{z}) * \mathbf{k} - \mathbf{y} \right\|. \tag{3.7}$$

At the end of the training, the restored image is computed as

$$\tilde{\mathbf{x}} = f_{\theta^*}(\mathbf{z}). \tag{3.8}$$

By itself, this approach proves to be effective for various problems such as denoising,

inpainting, JPEG artifacts restoration, but underperforming for deconvolution. Some further studies succeed in achieving good results by adding a regularization term $R\left(f_\theta(\mathbf{z})\right)$ to the loss function:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \left[E\left(f_\theta(\mathbf{z}); \mathbf{y}\right) + \lambda \cdot R\left(f_\theta(\mathbf{z})\right)\right]. \tag{3.9}$$

A particularly well performing technique is DIP-TV [12], which uses total variation regularization, expressed as

$$R_{TV}(\mathbf{x}) = \sum_{i,j} \sqrt{(\mathbf{x}_{i,j} - \mathbf{x}_{i-1,j})^2 + (\mathbf{x}_{i,j} - \mathbf{x}_{i,j-1})^2}, \tag{3.10}$$

where $\mathbf{x}_{i,j}$ is the pixel in the i^{th} row and j^{th} column. Another well performing technique is DeepRED [13], which uses regularization by denoising:

$$R_{DN}(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \left[\mathbf{x} - d(\mathbf{x})\right], \tag{3.11}$$

where $d(\mathbf{x})$ is a denoising function of choice. In the article, the authors use either non-local means (NLM) or block-matching and 3D filtering (BM3D).

3.3.2. Blind parameters estimation

In this section, the degradation model is known but not the degradation parameters. For example, a convolutional degradation model is assumed, but the PSF kernel is not known. What this models attempt to do is to estimate the degradation parameters, and then reverse it.

Learning to deblur

The first end to end blind deblurring model is introduced in [14]. The model is composed of three stages. The first stage is a feature extractor, composed of convolutional layer, activation, linear layer, activation, and another linear layer. The idea behind this stage is that the outputs should be composed of features tuned to make the kernel estimation easier. The outputs are two maps, $\tilde{\mathbf{x}_i}$ and $\tilde{\mathbf{x}_i}$. The second stage is the kernel estimation stage, which is performed by minimizing

$$\sum_i \|\tilde{\mathbf{k}} * \tilde{\mathbf{x}_i} - \tilde{\mathbf{y}_i}\| + \beta_k \|\tilde{\mathbf{k}}\|, \tag{3.12}$$

equivalent to solving

$$\tilde{\mathbf{k}} = F^H \frac{\sum_i \overline{F\tilde{\mathbf{x}}_i} \odot F\tilde{\mathbf{y}}_i}{\sum_i |F\tilde{\mathbf{x}}_i|^2 + \beta_k}, \tag{3.13}$$

where F is the DFT matrix, F^H is the transpose conjugate of F, $\odot$ is the Hadamard product, and $\overline{\mathbf{v}}$ is the complex conjugate of $\mathbf{v}$. The division and absolute value are performed element-wise. This operation is implemented in what the authors call *quotient layer*. The third stage is the image estimation stage, performed by minimizing

$$\sum_i \|\tilde{\mathbf{k}} * \tilde{\mathbf{x}} - \tilde{\mathbf{y}}\| + \beta_x \|\tilde{\mathbf{x}}\|, \tag{3.14}$$

which is also solved with a quotient layer.

A multiscale approach is implemented, in which the base network is stacked multiple times, each computing the deblurred image at a different resolution. The result of the smallest resolution level is upscaled by using a deconvolution layer, and given as input to the second level, concatenated with the distorted image at higher resolution, to help the reconstruction. Same with the third one. For example, if the model operates on a 256p image, the image is downscaled to 64p and fed to the lowest resolution pipeline, obtaining a 64p reconstruction. This reconstruction is upscaled to 128p, and concatenated with a 128p downscaled version of the distorted image, obtaining a 128p reconstruction. The same is done to get the full 256p reconstruction. The approach proves to be competitive with the non machine learning based approaches of blind deconvolution.

A Neural Approach to Blind Motion Deblurring

In [15] another approach to deblur based on a convolutional model is presented. The authors propose a model that is trained on image patches of 65 x 65 pixels, and outputs the Fourier coefficients of a deconvolution filter. The deconvolution filter is applied to the patch to compute the sharp version. The loss used to train the network is the L2 distance between the estimated sharp image via deconvolution, and the real sharp image. The patches are converted in the Fourier domain and separated into frequency bands. The bands are fed to a network that has the first two layers with a connectivity limited to adjacent bands, to have a low computational cost. After these two layers, follows five fully connected layers. The architecture is presented in Figure 3.6.

Multiple patches are processed, and then are merged into an initial estimation of the sharp image. To merge the deblurred patches, the value of a pixel in the full merged image is computed as the weighted average between the same pixel in the patches that contain it, scaled by a Hann window value. The computed image is used to perform a

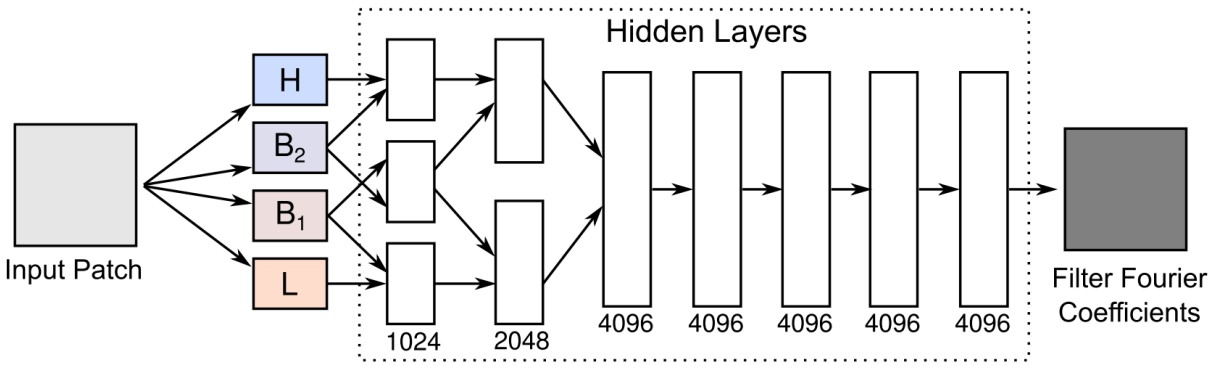


Figure 3.6: Architecture described in [15]. L, B1, B2, H represent lowpass, low bandpass, high bandpass, highpass filters.

better estimation of the kernel $\mathbf{k}_\lambda$, as

$$\mathbf{k}_\lambda = \operatorname{argmin} \sum_i \|(\mathbf{k} * (\mathbf{f}_i * \mathbf{x})) - (\mathbf{f}_i * \mathbf{y})\|^2 + \lambda \sum_n |\mathbf{k}[n]|, \tag{3.15}$$

where $\mathbf{f}_i$ are various derivative filters, $\mathbf{x}$ is the output of the averaging of the patches, $\mathbf{y}$ is the blurred image, λ is a regularization weight. The resulting kernel is used to deconvolve $\mathbf{y}$ using a state of the art non-blind algorithm, and the final estimation of the reconstruction is obtained.

Non-convolutional synthetic blur

In this work we focused mainly on convolutional blur, due to the article by Boracchi and Foi [1] being the main inspiration for the study, but other kinds of synthetic blur have been proposed. In [16] the authors propose a technique to deblur an heterogeneously blurred image, but restricting the blur only to linear kernels. What it means is that each pixel is considered as individually blurred with a kernel consistent of a straight line, and the kernel varies spatially. To do so, a CNN is trained to recognize the linear blur in a small patch. This model is applied on overlapping patches, and a dense motion field estimation is computed as average of the multiple kernels for each single pixel. From this field, a non-uniform deconvolution algorithm is used to restore the sharp image. Another article based on the estimation of a dense motion field is [17]. The authors take the idea a step further and implement a U-net trained to directly translate an image to a 2 channel representation of the motion field. This method proves to be more effective than the patch-wise version. By using this better estimation of the degradation, the reconstruction is more accurate.

3.3.3. Blind image-to-image translation

In this chapter the analysis focuses on the approaches without assumptions about parametrization of the image distortion. All these techniques are image-to-image translations, and the models have, as only input, the distorted image, and outputs the restored image. We start by introducing Generative Adversarial Networks (GANs) [18], a network architecture which has proven to be particularly effective in this problem.

GANs

GANs are generative models, meaning that their goal is to learn the probability distribution of a training set, so that we can use the model to generate new data with the desired features. The structure of a GAN is composed by two competing networks. A generator network takes as input a random seed, which usually is a random noise image of the same size as the desired output, and generates an image. A discriminator networks takes as input an image and outputs a [0,1] value that represents the confidence that the image is natural or generated by the generator network.

The training is conducted in two alternating steps. The discriminator is fed a mixed batch of real images, to be classified 1, and images generated by the generator, which are composed of noise in the beginning of the training, to be classified 0, and iterated for some epochs. Then, the generator is fed a random seed, and trained for some epochs to generate an image that fools the discriminator.

The loss is defined as

$$\mathcal{L}_{GAN} = \mathbb{E}_{\mathbf{x}\sim P_{data}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim P_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))], \tag{3.16}$$

where D is the discriminator, G is the generator, $\mathbf{x} \sim P_{data}$ means that $\mathbf{x}$ is extracted from the dataset of real images, and $\mathbf{z} \sim P_{\mathbf{z}}$ is extracted from the space of random seed values. The generator is trained to minimize the function, and the discriminator to maximize it. The two networks are trained for a fixed number of epochs each. After completing the training the discriminator is no longer required, and the generator can be used to produce images from random seeds.

Generative adversarial networks face two significant challenges, namely stability and mode collapse. Stability issues emerge when the training oscillates without finding an equilibrium point between the generator and discriminator functions. On the other hand, mode collapse occurs when the generator only learns to produce a limited set of outputs, leading to lack of variation in the generated samples. The original version of GANs lack a

conditioning mechanism, so they are not suitable to the problem of image restoration.

Pix2pix

The first paper to use generative adversarial networks for image-to-image translation is [19]. The conditioning mechanism is the concatenation of the input image to the seed. The architecture is composed by a generator network, that takes as input a noise image seed concatenated with the input image, and generates an image, and a discriminator network, that takes as input a pair of images, and outputs a [0,1] value that represents the confidence that the images are a input and a ground truth pair, or an input image and a generated output.

The training is conducted in the same way as with non conditional GANs, but using as loss:

$$\mathcal{L}_{cGAN} = \mathbb{E}_{\mathbf{x},\mathbf{y}}[\log D(\mathbf{y},\mathbf{x})] + \mathbb{E}_{\mathbf{y},\mathbf{z}}[\log(1 - D(\mathbf{y}, G(\mathbf{y},\mathbf{z}))], \tag{3.17}$$

where $\mathbf{y}$ is the input image, $\mathbf{x}$ is the target image and $\mathbf{z}$ is the noise seed image. For the generator, the loss is also mixed with an L1 loss as shown in Equation 3.19

$$\mathcal{L}_{L_1} = \mathbb{E}_{\mathbf{x},\mathbf{y},\mathbf{z}}[||\mathbf{x} - G(\mathbf{y},\mathbf{z})||_1], \tag{3.18}$$

$$\mathcal{L}_{tot} = \mathcal{L}_{cGAN} + \lambda\mathcal{L}_{L1}, \tag{3.19}$$

while the discriminator is trained on $\mathcal{L}_{cGAN}$ only. The findings suggest that the model tends to disregard the noise seed during training. However, eliminating the seed results in a model that can only match delta functions and produce deterministic outputs, which limits its ability to learn input-to-output relationships. To inject stochasticity into the model, the authors replace the seed with several dropout layers during both testing and training. Other studies, like [6], instead completely remove the seed, without dropout.

The authors demonstrate the versatility of their image-to-image model by successfully applying it to several different problems. While they did not use the model for the deblurring task in this article, it serves as a meaningful example of the potential of image-to-image GANs.

DeepDeblur

DeepDeblur is GAN-based deblurring model proposed in [6]. As generator, the authors use a multi-scale architecture, like the one used in [14] and presented in Section 3.3.2, consisting of three U-Net models with residual blocks pipelines, that compute the reconstruction of the sharp image at three different resolutions. Each of the three models is

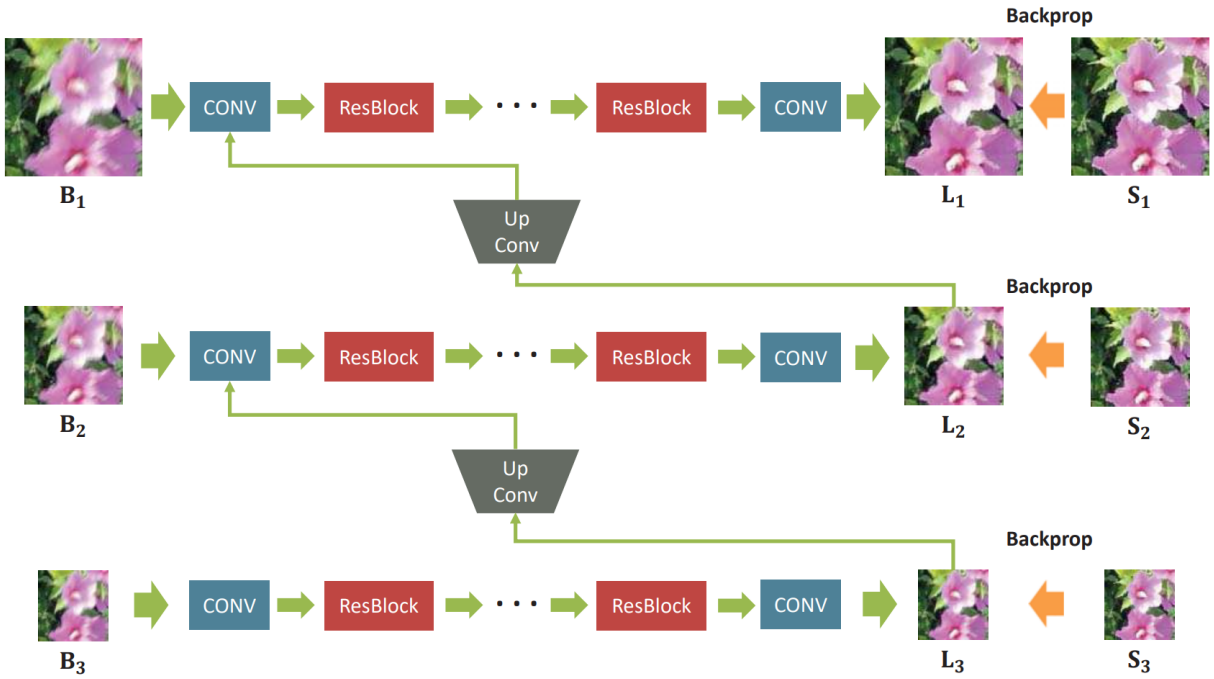


Figure 3.7: Architecture presented in the article [6].

constituted of multiple cascaded residual blocks, made of a convolution layer, an activation layer, a convolution, and a skip connection from the input to the output. The the result of the smallest resolution models is upscaled by using a deconvolution layer, and given as input to the second models, concatenated with the distorted image at higher resolution, to help the reconstruction. The same happens to the third one. The architecture is represented in Figure 3.7.

The discriminator takes as input either a sharp image or a deblurred image, and classifies it as such. It is trained on the loss in (3.17). The generator is trained, together with the adversarial loss, on an MSE component at each scale of the model. The total loss for the generator is defined as

$$\mathcal{L}_{tot} = \mathcal{L}_{MSE} + \lambda \cdot \mathcal{L}_{adv} \tag{3.20}$$

where $\lambda = 10^{-4}$ is constant. An important addition of this paper is the creation of the first non-synthetically blurred dataset, the GoPro dataset, described in Section 2.4, which is used in this article for both training and testing.

DeblurGAN

DeblurGAN is proposed by Kupyn et al. in [7]. It uses the same method as the Pix2pix model, with the exception of the loss function, which is defined by two main contributions.

The first contribution is the Wesserstein GAN loss, as in [20], expressed as

$$\mathcal{L}_{GAN} = -\mathbb{E}_{\mathbf{y}} \left(D(G(\mathbf{y})) \right), \tag{3.21}$$

where $\mathbf{y}$ is the blurred image.

The second contribution is a perceptual loss term, which follows the same reasoning as the LPIPS metric, described in Section 2.3.3, discarding the scaling of the features, defined as

$$\mathcal{L}_X = \mathbb{E}_{\mathbf{y},\mathbf{x}} \left((\phi(G(\mathbf{y})) - \phi(\mathbf{x}))^2 \right), \tag{3.22}$$

where ϕ is the output of the *conv3.3* layer of a trained VGG-19. The two losses are combined in:

$$\mathcal{L} = \lambda \cdot \mathcal{L}_X + \mathcal{L}_{GAN} \tag{3.23}$$

Where $\lambda = 100$ is a constant weight. The architecture is a U-Net with ResBlocks.

Three models are trained, one with convolutional camera shake blur, like in [1] but discarding the noise component, one trained with the Gopro dataset, and one with a mix of both. What the authors show is that the model trained with the mixed dataset obtain better performance than the others. An advanced version of this approach is developed in [21]. In this article, the authors modify the architecture of DeblurGAN to incorporate a multiscale approach, as seen in DeepDeblur [6], obtaining better results than the previous DeblurGAN.

SRN

In [22], the authors introduce an architecture based on a multiscale U-Net with skip connection, like the one in [6]. The difference is that, at the bottleneck layer, a ConvLSTM cell [23] is present, that passes the state to the next level of the multiscale pyramid. This allows grater receptive field and improves the performance of the system compared to the [6]. The loss used in this article is only composed by a MSE term, without adversarial loss. The authors show that the performance of the model is still better than [6], even without adversarial component. The GoPro dataset is used. In the article, the authors also test the performance of a single scale versus the full multiscale approach, to prove that the latter actually improves the result.

4 | Diffusion models

Diffusion models [24] are a recent alternative generative model to GANs. The fundamental concept, which is drawn from thermodynamics, is to use an iterative process to gradually corrupt information of a data distribution, until obtaining a random distribution. A neural network model is trained to reverse the corruption process, and by doing so obtaining a powerful and very flexible generative model. The original authors initially applied the method to test distributions, such as synthetic toy datasets and simple sequences. Additionally, they utilized the diffusion model to generate small images.

4.1. Denoising Diffusion Probabilistic Models

Ho et al. [25] demonstrated the successful application of diffusion models in high-resolution image generation. The article presents the diffusion process as navigation on a Markov chain, where the states represent an image with varying levels of noise, ranging from pure Gaussian noise to an image without any noise. See the Figure 4.1 for a graphical representation.

4.1.1. Training and inference algorithms

The process that starts from an image $\mathbf{x_o}$ drawn from a real image distribution and progressively adds small amounts of gaussian noise is called *forward diffusion* process, and is represented as q. This process produces a sequence of noisy samples $\mathbf{x_1}$,..., $\mathbf{x_T}$ for each value of t, called *timestep*, where T is the number of diffusion steps. The diffusion process is determined only by the noise schedule, controlled by the noise variance $\{\beta_t \in (0,1)\}_{t=1}^{T}$. For a large T, $\mathbf{x_T}$ can be approximated as an isotropic Gaussian distribution.

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{4.1}$$

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \tag{4.2}$$

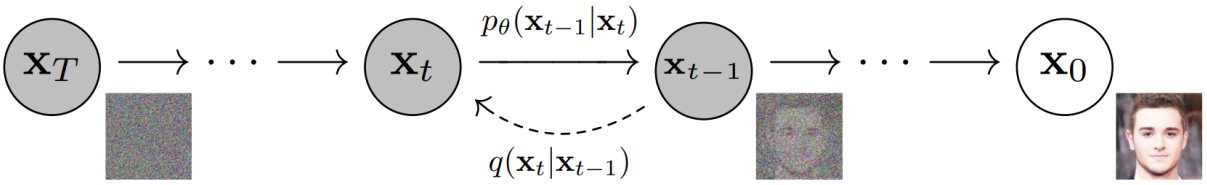


Figure 4.1: Graphical representation of the markov chain model considered in [25].

This notation is used by the authors to indicate that each sample $\mathbf{x_t}$ has normal distribution with mean $\sqrt{1-\beta_t}\mathbf{x}_{t-1}$ and variance $\beta_t\mathbf{I}$. By using the reparametrization $\alpha_t = 1-\beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$, the following property is derived:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}), \tag{4.3}$$

What the diffusion model tries to accomplish is to learn the opposite process, called *reverse diffusion* process, denoted as p_θ, where θ indicates the array of learnt parameters.

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \tag{4.4}$$

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \tag{4.5}$$

where $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T, \mathbf{0}, \mathbf{I})$.

While $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is unknown, $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ is tractable.

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t\mathbf{I}) \tag{4.6}$$

Using Bayes' rule, the mean and variance of $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ can be parameterized as follows

$$\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t \tag{4.7}$$

$$\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) = \left(\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0\right) \tag{4.8}$$

The authors use variational bound on negative log likelihood to train the model:

$$-\log p_\theta(\mathbf{x}_0) \leq \mathbb{E}_q\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] \tag{4.9}$$

The objective can be further modified to be a mixture of many KL-divergence and entropy

terms to make each term in the equation analytically calculable.

$$L_{\text{VLB}} = \mathbb{E}_q \left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})} \right] \tag{4.10}$$

$$= \mathbb{E}_q \Bigg[\underbrace{D_{\text{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p_\theta(\mathbf{x}_T))}_{L_T} + \sum_{t=2}^{T} \underbrace{D_{\text{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{- \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0} \Bigg] \tag{4.11}$$

The distribution $p_\theta(\mathbf{x}_T)$ has no learnable parameters, so L_T is a constant and is ignored during training. The last reverse step, the one to estimate $\mathbf{x}_0$, is computed using a separate decoder. At the end of sampling, $\mathbf{x}_0 = \boldsymbol{\mu}_\theta(\mathbf{x}_1, 1)$.

The authors assume that $\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t) = \sigma_t^2 \mathbf{I}$, assumption that will be lifted in some of the following articles, so the objective of the article is to train $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$ to predict $\tilde{\boldsymbol{\mu}}_t$. The authors also reparametrize the problem to use the model to predict the noise at each step, instead of the mean value, and include the reparametrization $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$ with $\epsilon \sim \mathcal{N}(\mathbf{x}_T, \mathbf{0}, \mathbf{I})$:

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}} \Big(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \Big) \tag{4.12}$$

where $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ is the noise estimation on $\mathbf{x}_t$ at the timestep t. Sampling $\mathbf{x}_{t-1} \sim p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ means to compute:

$$\mathbf{x}_{t-1} = \mathcal{N}\left(\mathbf{x}_{t-1}; \frac{1}{\sqrt{\alpha_t}} \Big(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \Big), \sigma_t^2 \mathbf{I} \right) \tag{4.13}$$

By doing this, the loss at each timestep t is simplified to:

$$L_t = \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}} \left[\frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1-\bar{\alpha}_t)} \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2 \right] \tag{4.14}$$

The authors also found that the training works better when the weighting term is ignored:

$$L_t^{\text{simple}} = \mathbb{E}_{t \sim [1,T], \mathbf{x}_0, \boldsymbol{\epsilon}} \left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2 \right] \tag{4.15}$$

$$= \mathbb{E}_{t \sim [1,T], \mathbf{x}_0, \boldsymbol{\epsilon}} \left[\left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2 \right] \tag{4.16}$$

When the model is completely trained, random gaussian noise can be fed as $\mathbf{x}_T$ and the

Algorithm 4.1 DDPM training algorithm

repeat
  extract $\mathbf{x}_0$ from the target images dataset
  $t \sim Uniform(1, ..., T)$
  $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take a gradient descent step on $\nabla_\theta \|\epsilon - \epsilon_\theta \left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t\right)\|^2$
until convergence

Algorithm 4.2 DDPM sampling algorithm

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, ..., 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
end for
return $\mathbf{x}_0$

Markov chain can be navigated backwards to generate a sample with the features learnt from the dataset. The final algorithms proposed for training and sampling are described in 4.1 and 4.2. For the complete mathematical discussion, see article [25].

The article employs a noise schedule that involves a linear interpolation between $\beta_1 = 10^{-4}$ and $\beta_T = 0.02$. The authors suggest two options for σ_t used in the sampling process, namely β_t and $\tilde{\beta}_t$, both of which produced similar results in experiments. While other interpolation methods, such as quadratic or cosine-shaped, can be used between β_1 and β_T, most studies, including the original article, tend to favor the linear interpolation for its balance between simplicity and performance. The article also employs exponential moving average regularization, with a with a decay factor of 0.9999.

4.1.2. Architecture

The backbone of the denoising neural network is a U-Net with skip connections, with a few adjustment. The model include residual blocks and attention blocks as main computation components. The parameters are shared along the Markov chain, and we effectively have only one denoising model that works for every value of t. To inject in the network knowledge about the timestep t, position embedding is used.

Position embedding

Position embedding is a technique described in the article that introduces the capabilities of the transformer architecture [26], as a mean of encoding the position of a word in a

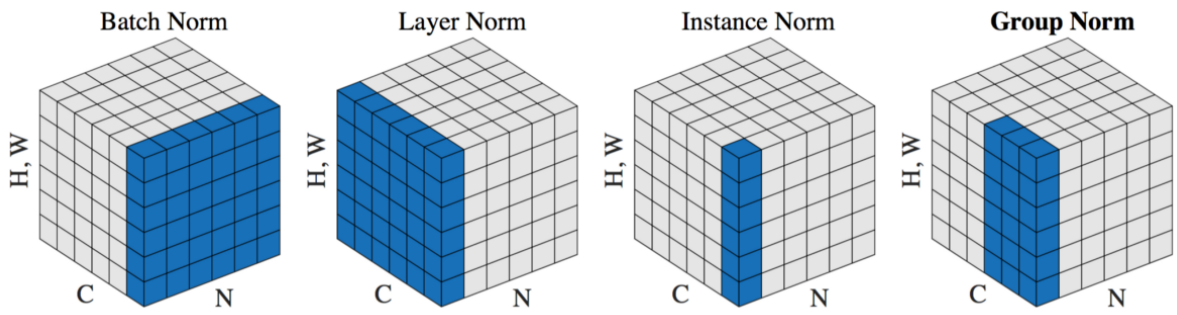


Figure 4.2: Graphic depicting in blue the values used for normalization, in various normalization layers, as described in [27].

phrase to be processed by a network. This layer takes as input the values of t for the images in the batch, in a tensor of shape $(b, 1)$, and outputs a tensor of embeddings of shape (b, c), where c is the number of channels of each image in the tensor and b is the batch size. The embeddings are computed as

$$\vec{p_t}^{(i)} := \begin{cases} \sin(\omega_k t), & \text{if } i = 2k \\ \cos(\omega_k t), & \text{if } i = 2k+1 \end{cases} \tag{4.17}$$

where:

$$\omega_k = \frac{1}{10000^{2k/d}} \tag{4.18}$$

This process has no learnable parameters. In addition, the position embeddings are fed to a *position encoding block* composed by a linear layer, an activation layer, and another linear layer.

Group normalization

Group normalization is introduced in [27] as an alternative to batch normalization. In batch normalization, as the batch size decreases, the error quickly rises due to poor batch statistics estimation. This restricts the use of this mechanism for computer vision applications, where often samples are large and batches must be reduced in size to allow them to fit in the RAM of the machine.

Group normalization divides the channels into groups and calculates the mean and variance for each group. The comparison with other normalization mechanisms is depicted in Figure 4.2. The computation is batch size independent, and its accuracy is steady over a large range of batch sizes.

Attention layer

The attention mechanism is another technique described in [26]. The idea is to compute, from an input tensor, an attention map, which can be thought of as a vector of importance weights, or as a vector that represents how strongly each value in the output correlates with values from the input. Another advantage of this layer is that, contrarily to what happens in convolutional layers, the output depends on each value in the input, and so the receptive field is increased.

For each input tensor, three matrices of shape $wh \times d$ are computed, named $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$, where w and h are the dimensions of each input channel and d is the dimension of the representation in the attention layer, an hyperparameter to be decided when designing the model. Each one of the maps is computed via multiplication by a weight matrix. In this context, each input channel is considered as a $\mathbb{R}^{wh}$ vector. The input matrix $\mathbf{X}$ is thus a $wh \times c$ matrix, where c is the number of channels.

$$\mathbf{Q} = \mathbf{X} \cdot \mathbf{W}_q \tag{4.19}$$

$$\mathbf{K} = \mathbf{X} \cdot \mathbf{W}_k \tag{4.20}$$

$$\mathbf{V} = \mathbf{X} \cdot \mathbf{W}_v \tag{4.21}$$

The weight matrices $\mathbf{W}_q$, $\mathbf{W}_k$ and $\mathbf{W}_v$ are learnt with gradient descent, and are in $\mathbb{R}^{c \times d}$. From the tensors, an attention map for each pixel is computed as:

$$\mathbf{Z} = \text{softmax}\left(\frac{\mathbf{Q} \cdot (\mathbf{K})^T}{\sqrt{d}}\right) \cdot \mathbf{V} \tag{4.22}$$

where $\mathbf{Q} \cdot (\mathbf{K})^T \in \mathbb{R}^{wh \times wh}, \mathbf{Z} \in \mathbb{R}^{wh \times d}$. The softmax operation applied to the matrix is so that the resulting matrix has all the rows that sum up to 1. In the case of [25], $d = c$ is chosen, so $\mathbf{Z}$ can be directly used as output. The tensors $\mathbf{W}_q$, $\mathbf{W}_k$ and $\mathbf{W}_v$ are learnt with gradient descent.

Residual block

The residual block used in this model differs from the classic one because in addition it has to incorporate the information about the timestep. The structure comprises *convolutional blocks*, each composed of a group normalization layer, an activation layer, and a 2D convolutional layer. It also contains a block composed by a SiLU and a linear layer that gets as input the output of the *position encoding block* and outputs a $b \times c$ tensor. A graphical representation is the second block in Figure 4.5.

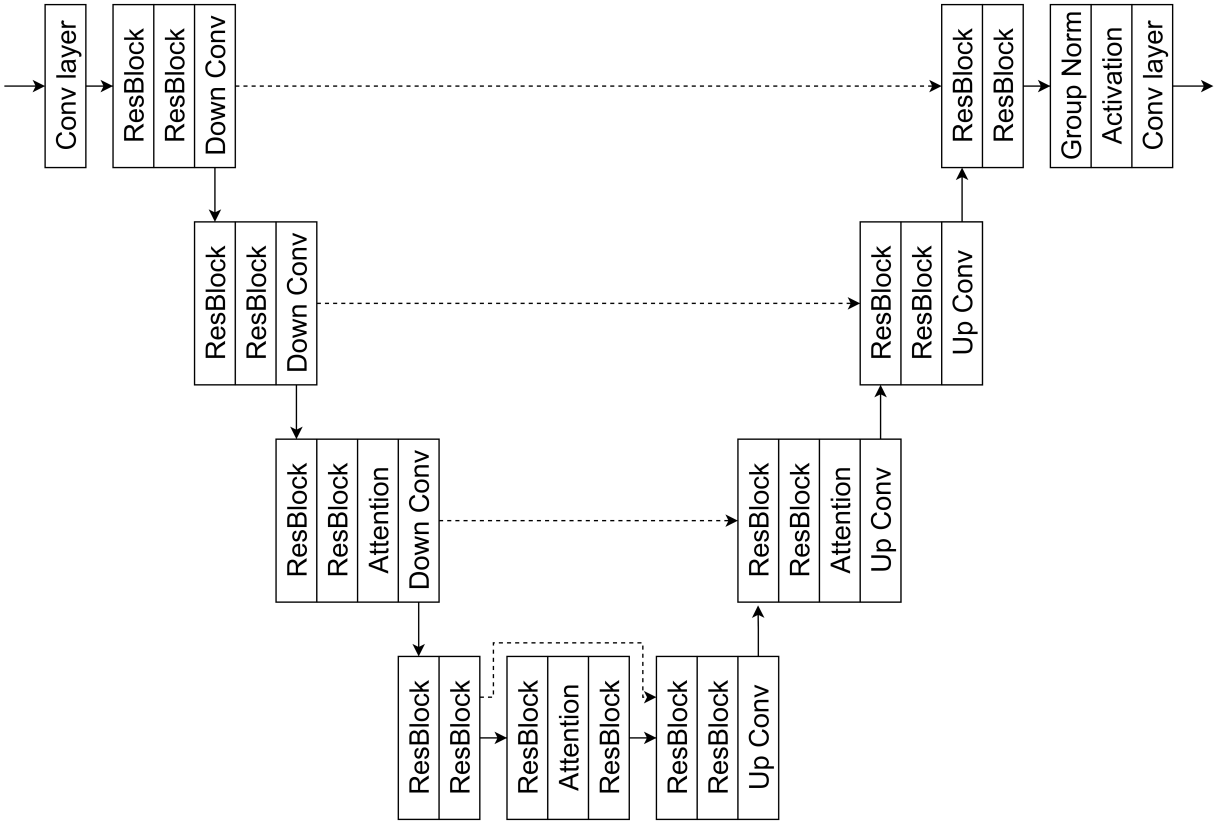


Figure 4.3: Full architecture from [25], example used for 64 pixel images. Dashed lines are the skip connections.

The full residual block is composed by a convolutional block, the result is added to the output of the timestep block, and this is fed to another convolutional block. In addition, there is a residual connection from before the first block to the output of the second, with a 1×1 convolutional layer on it.

Full architecture

The full architecture of the diffusion model in [25] is composed as follows:

- the input tensor is fed to a 2D convolutional input layer, and the position embeddings are fed to the *position encoding* block;
- cascade of downsampling stages, composed by two residual blocks and a downsampling convolution (except for the last stage, which has no downsampling);
- middle stage, composed by a residual block, an attention layer, another residual block;
- cascade of upsampling stages, that take as input the concatenation of the output of

the previous stage and the skip connections from the downsampling stage, composed by two residual blocks and an upsampling convolution (except for the last stage, which has no upsampling);

- an output stage, composed of a group normalization, an activation layer and a convolutional layer, to restore the correct number of output channels.

In addition, at the 16×16 resolution map, there is another attention block between the residual blocks and the downsampling layer of both the downsampling and upsampling stage.

4.1.3. Performance

The authors conclude stating that the model has samples quality comparable to state-of-the-art GAN models, with the important advantage that diffusion models avoid model collapse and the stability problems that GANs present. On the drawbacks, they admit that the computational cost is greatly increased, because the model needs to be applied to the sample T times, instead of the one-shot sample approach of GANs. Another big drawback is the fact that, due to the group normalization and the global attention blocks, the output resolution of the image is fixed at model creation. While sampling speed is an aspect on which many studies has focused on, a non-fixed resolution diffusion model is yet to be tested.

4.2. Further improvements

BigGAN residual blocks

In [28], the authors introduce a large GAN architecture to improve the generation power of previous GANs for high resolution images. Among the other changes to the structure of the model, an alternative to the classic convolutional upsampling and downsampling layers is proposed, represented in Figure 4.4. This block can be used as alternative to the upscale/downscale convolution in the diffusion model architecture. To be adapted to the diffusion model, a position embedding mechanism is added between the two convolution stages of the block, as shown in Figure 4.5.

SR3

In [25] no conditioning mechanism is proposed. To generate images belonging to a certain class, the authors train the network using only the images belonging to the class. This is not a very flexible implementation, as it needs the training of a different network for

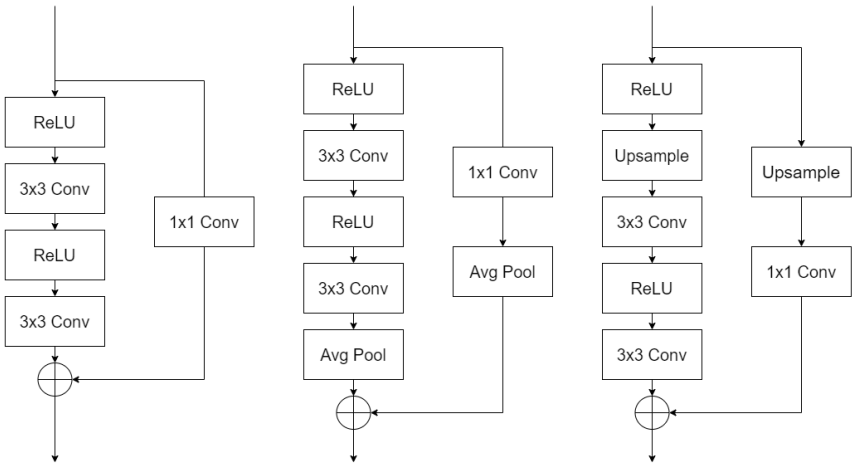


Figure 4.4: Standard residual block (left), upsampling residual block (center) and downsampling residual block (right) blocks, BigGAN style.

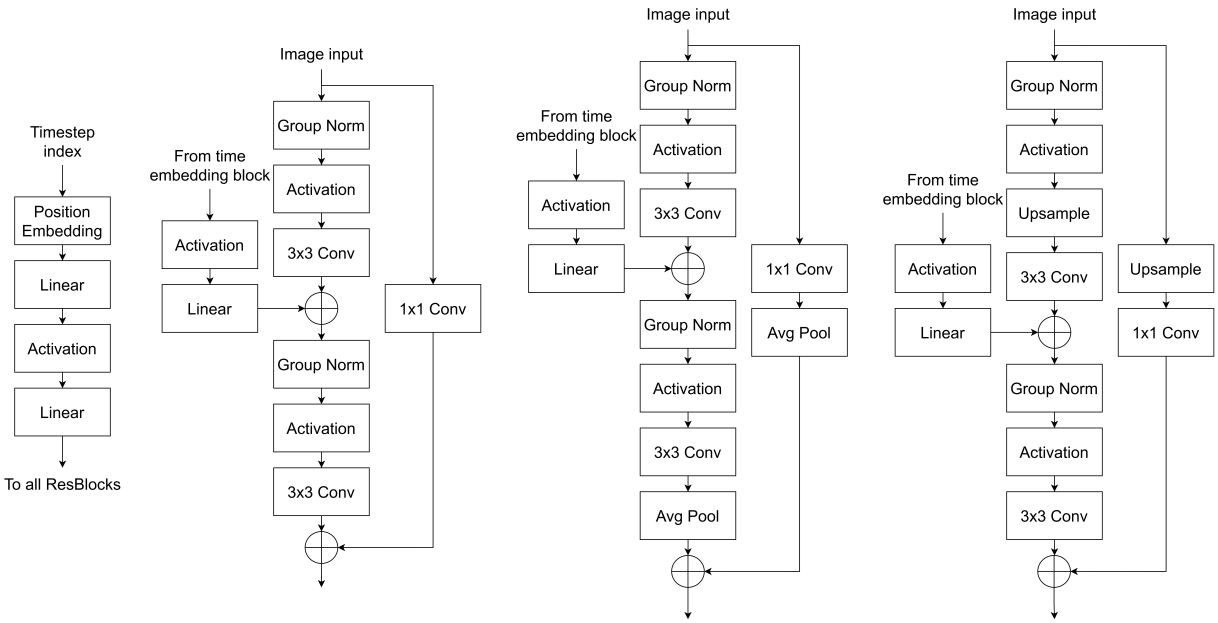


Figure 4.5: Standard residual block (left), upsampling residual block (center) and downsampling residual block (right) blocks, BigGAN style with time embedding, used in [29].

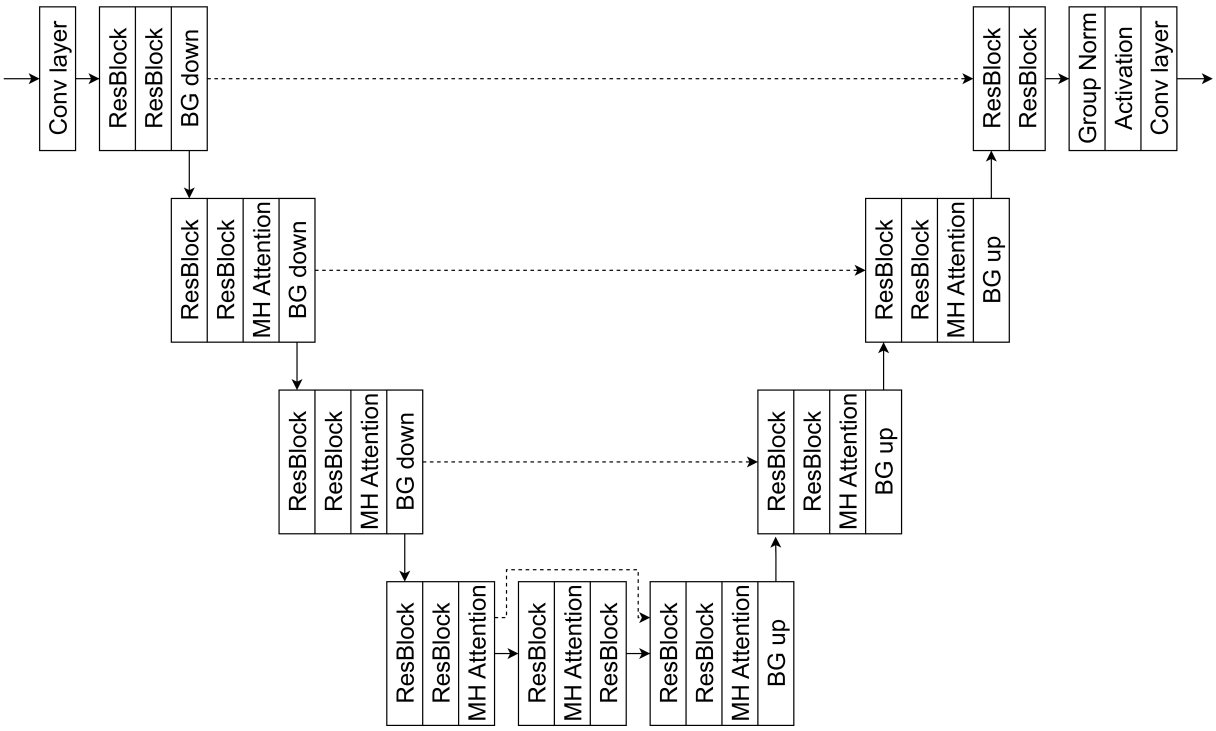


Figure 4.6: Full architecture from [29]. Example used for 64 pixels images. MH stands for multi-head, BG for BigGAN. Dashed lines are the skip connections.

Algorithm 4.3 SR3 training algorithm

1: **repeat**
2: extract $\mathbf{x}_0$ from the target images dataset
3: extract $\mathbf{y}$ from the low resolution images dataset
4: $\gamma \sim p(\gamma)$
5: $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
6: Take a gradient descent step on $\nabla_\theta \left\| \epsilon - \epsilon_\theta \left(\mathbf{y}, \sqrt{\gamma}\mathbf{x}_0 + \sqrt{1-\gamma}\epsilon, \gamma \right) \right\|^2$
7: **until** convergence

each class. A naive yet effective way of conditioning image generation on input images is proposed in [30], in witch the authors concatenate the conditioning image to the partially denoised image at each diffusion step. This model is called conditional diffusion model.

In [31], a diffusion model architecture similar to the one proposed in [25] is trained in a conditional manner to address the problem of super resolution, in a model called SR3. In SR3, following the intuition of [32], the positional encoding on t is replaced by conditioning on $\gamma = \bar{\alpha}$. This allows a more intuitive conditioning than the one on natural numbers, and also to use different β schedules on training and inference, and thus use less timesteps for inference, effectively reducing the sampling time. The model is trained to take as input, in the first step of the diffusion process, the blurred image and the noise seed,

Algorithm 4.4 SR3 sampling algorithm

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, ..., 1$ **do**
3: $\quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(\mathbf{y}, \mathbf{x}_t, \gamma_t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

concatenated in a 6 channel image, and the γ of the first step, and to output the next step of the diffusion process. This output is concatenated with the blurred image, and fed to the model together with the next γ value. The process is iterated until the end of the Markov chain. The conditional version of the algorithms proposed by Ho et al. is described in Algorithm 4.3 and 4.4. The architecture used is similar to the one found in the original DDPM model, but using residual blocks in the style of BigGAN and increasing the number of residual blocks per resolution.

The input fed to the model, as done in other super resolution models, is an upscaled version of the low resolution image to the desired resolution, obtained using bilinear interpolation. Super resolution is sometimes seen as a special case of deblurring: the aim of super resolution neural networks is to remove the smoothing introduced by the interpolation used for the upscaling. The good performance of this approach in the super resolution problem is one of the facts that inspired our use of diffusion model in image deblurring.

Multi-head attention layer

Multi-head attention is an alternative to the previously described attention layer, also proposed in [26]. The difference is that instead of computing the three matrices $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$, it computes a series of n triplets matrices of shape $wh \times d$, named $\mathbf{Q}_j$, $\mathbf{K}_j$ and $\mathbf{V}_j$, where n is the number of attention heads and d is the head dimension. Both are hyperparameters to be decided when designing the model. Each one of the maps is computed via multiplication by a weight matrix. Again, each input channel is considered as a $\mathbb{R}^{wh}$ vector. The input matrix $\mathbf{X}$ is thus a $\mathbb{R}^{wh \times c}$ matrix, where w and h are the dimensions of each input channel, and c is the number of channels.

$$\mathbf{Q}_j = \mathbf{X} \cdot \mathbf{W}_{qj}, \tag{4.23}$$

$$\mathbf{K}_j = \mathbf{X} \cdot \mathbf{W}_{kj}, \tag{4.24}$$

$$\mathbf{V}_j = \mathbf{X} \cdot \mathbf{W}_{vj}, \tag{4.25}$$

where j is the index that represents the head number, ranging from 1 to n. The weight matrices are in $\mathbb{R}^{c\times d}$. From the tensors, an attention map for each pixel and for each head is computed as:

$$\mathbf{Z}_j = softmax\left(\frac{\mathbf{Q}_j \cdot (\mathbf{K})_j^T}{\sqrt{d}}\right) \cdot \mathbf{V}_j, \tag{4.26}$$

where $\mathbf{Q}_j \cdot (\mathbf{K})_j^T \in \mathbb{R}^{wh\times wh}, \mathbf{Z}_j \in \mathbb{R}^{wh\times d}$. The $softmax$ operation is applied to the matrices so that the resulting matrix has all the rows that sum up to 1. The matrices $\mathbf{Z}_j$ are concatenated in a matrix $\mathbf{Y} \in \mathbb{R}^{wh\times nd}$. To get an output of the layer with the same shape as the input, $\mathbf{Y}$ is multiplied by another weight tensor $\mathbf{W}_o \in \mathbb{R}^{nd\times c}$. The result is a tensor in $\mathbb{R}^{wh\times c}$. The tensors $\mathbf{W}_q$, $\mathbf{W}_k$, $\mathbf{W}_v$ and $\mathbf{W}_o$ are learnt with gradient descent.

Usually, only one of the hyperparameters d and n is fixed, while the other is computed so that $n \cdot d = c$. By doing this, the number of network parameters is kept almost constant, but the sample quality is potentially increased.

Guided diffusion

In [33] and [29], Nichols and Dhariwal propose a series of architectural and algorithmic improvements to be applied to the method proposed by Ho et al., to enhance the quality of the generated samples. The authors find that using fixed values for the matrix $\mathbf{\Sigma}_\theta$ is sub-optimal for sampling. What they observed is that $\sigma_t^2 = \beta_t$ and $\sigma_t^2 = \tilde{\beta}_t$ are the upper and lower bounds for the reverse process variances. They propose to let the network learn $\mathbf{\Sigma}_\theta$, but instead of letting it do so freely, they learn the matrix as interpolation between β_t and $\tilde{\beta}_t$. The model specifically produces a vector with one component for each dimension, and this vector is converted into variances as follows:

$$\mathbf{\Sigma}_\theta(\mathbf{x}_t, t) = \exp(\mathbf{v}\log\beta_t + (1-\mathbf{v})\log\tilde{\beta}_t). \tag{4.27}$$

They also observe that v usually does not learn values outside the $[0, 1]$ range, suggesting that the bounds are indeed expressive enough. Given that the objective L_{simple} does not depend on $\mathbf{\Sigma}_\theta$, the authors propose as alternative objective the one given by:

$$L_{\text{hybrid}} = L_{\text{simple}} + \lambda \cdot L_{\text{VLB}} \tag{4.28}$$

where $\lambda = 0.001$ is a constant weight value, and stop-gradient is applied to $\boldsymbol{\mu}_\theta$ in the L_{VLB} term, so that L_{VLB} only guides the learning of $\mathbf{\Sigma}_\theta$. They also use γ conditioning as in [32].

As architectural modifications, they propose to change the diffusion model U-Net with:

- 256 base channels against 128 (i.e. the number of output channels of the first convolutional layer);
- BigGAN residual blocks for upscale and downscale;
- Attention blocks at 32×32, 16×16 and 8×8 resolutions, instead of only 16×16;
- Attention blocks with residual connections from before the $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ decomposition to the output of the block;
- Multi-head attention with either the number of head fixed to 4 or the number of channel per head fixed to 64.

The authors conclude that with this changes they are able to produce samples with quality metrics comparable to GAN approaches. In the same articles, a mechanism to condition image generation to a class, and an alternative approach to diffusion called DDIM, not based on a Markov chains, are proposed. These techniques are not relevant in the scope of this thesis, so they will not be treated.

Palette

The authors of [34] present an approach to extend the application of diffusion models to image-to-image translation. Their approach involves modifying the architecture proposed in [29] by removing class conditioning and integrating an image conditioning mechanism, similar to the one described in [31]. The loss function used is L_{simple} and there is no prediction for $\mathbf{\Sigma}_\theta$, as they found no practical benefit in performance. The modified architecture is utilized for various image processing tasks, including image inpainting, uncropping, colorization, and restoration of JPEG artifacts. The authors demonstrate that the diffusion model can be adapted to serve as a general image-to-image translation method for various classic problems in the field, where computational time is not the primary concern. The authors find that diffusion model often outperforms GANs and CNNs in these tasks.

5 | Contributions

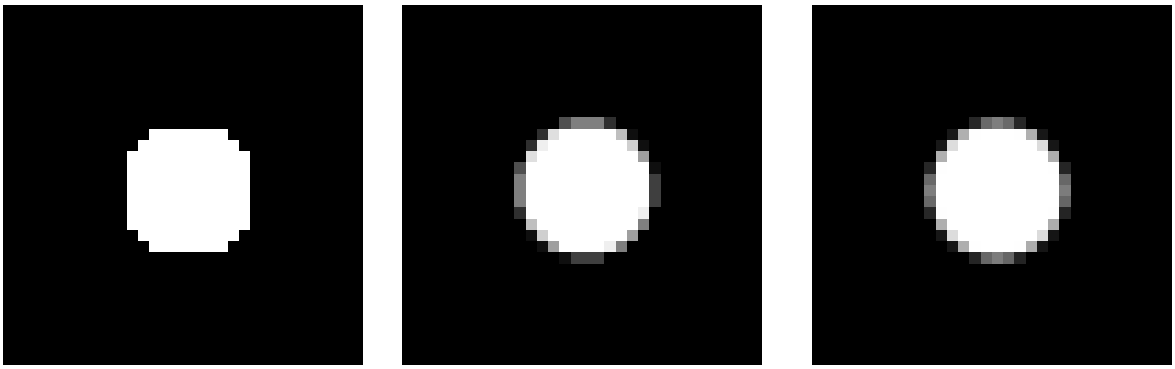
In this section we present the main contributions of this thesis. We present our novel dataset generation pipeline for neural network training. Following, we present our investigation of the application of conditional diffusion models in image deblurring.

5.1. Camera shake dataset generators

We present an implementation of the generator classes for the blurring kernels presented in Chapter 2: an *Out Of Focus (OOF) kernel generator* by following the formulation in Section 2.1.1, and a *Camera Shake Blur (CSB) kernel generator* by following the one presented in Section 2.1.2. The camera shake blur follows the kernel generation algorithm from Section 3.1, and includes an algorithm to avoid pixel displacement during the degradation.

We use the kernel generators in three other classes, used to degrade images by producing a random degradation within user specified ranges of parameters: *OOF random degradation generator*, *CSB random degradation generator*, and *Noisy Camera Shake Blur (NCSB) random degradation generator*. We also present a tool to degrade images with a constant trajectory, to be used to investicate the tradeoff between noise and blur, as shown in 3.1.4, the *Constant Trajectory (CT) degradation generator*. These functionalities are then integrated in dataset generators, to train neural networks. The implementation is used to train conditional diffusion models, and to show that they are an effective approach to reverse such degradation.

We are among the first to propose a randomized camera shake blur pipeline for neural network training, and the first to consider noise in the formulation. We think that this contribution can be helpful to future research in the image deblurring field, and we publicly release the code at `github.com/lorenzoinnocenti/csb-dataset-generator`.



(a) Aliased version (b) Precision = 4 (c) Precision = 100

Figure 5.1: Example of disk with radius of 6 pixels, generated at different precisions.

5.1.1. OOF degradation pipeline

We present an implementation the OOF kernel generator. To achieve this, we develop a class capable of generating a PSF kernel with customizable kernel size and disk radius. We create a function that evaluates, for every pixel, whether:

$$r^2 < (x - c)^2 + (y - c)^2, \tag{5.1}$$

where r is the disk radius, c is the matrix center, equal to half the matrix size. If the equation is true, the pixel is set to 1, otherwise is set to 0.

To avoid the creation of distorted kernels, due to the low pixel count, as shown in Figure 5.1a, we develop a function that can produce higher resolution kernels which are subsequently downsampled. This approach allows the PSF kernels to include values between 0 and 1 at the disk's edge, resulting in a more realistic and smooth edge. We introduce a *precision* parameter which determines the resolution of the larger kernel in terms of multiples of the desired PSF size. The impact of different precision values on the PSF kernel is illustrated in Figure 5.1. The described algorithm takes as inputs the size of the kernel matrix, disk radius, and precision, and produces the corresponding kernel.

We use the implemented functionality in the OOF random degradation generator. To achieve this, we develop a class which is initialized by providing a range of disk radius values and a kernel size to the constructor. The instantiated object offers a *process image* function, used to degrade an image. Specifically, when an image is passed to this function, a disk radius value is randomly selected within the specified range, a PSF kernel is created, and the image is blurred using the resultant kernel. This approach can be seamlessly integrated into the training of neural networks, as we will discuss later.

5.1.2. CSB degradation pipeline

We propose an implementation of the algorithm for camera shake blur as described in Section 3.1, which we call the CSB kernel generator. Our implementation comprises two classes, one for the creation of the trajectory and one for the sampling of the trajectory in the 2D grid, producing the kernel. During the creation of an instance of the trajectory creation class, the user can specify the parameters described in Section 3.1, which are kernel size, trajectory length, and parameters that characterize the motion, including anxiety, centripetal term, gaussian term, frequency of big shakes, resolution of the trajectory curve, and the maximum length in pixels. Once the instance is created, a function can be called to produce a trajectory with the specified parameters, in the form of a 1D array of complex values. To create an instance of the CSB kernel generator class, the desired trajectory and kernel size are passed as parameters. This class provides a *generate* function, that requires an exposure value as input. When called, the function creates the kernel by sampling the trajectory on a 2D grid of desired size, in a proportion according to the provided exposure value. If no exposure value is defined, the function defaults to using an exposure of 1.

We enhance the approach with a kernel recentering algorithm, to improve restoration accuracy in blind deblurring. If the trajectory is not centered in the kernel, it causes pixel displacements during the convolution process. This scenario happens frequently, as the algorithm was not originally meant to be used in deep learning training, particularly when the exposure value is different than 1. This is to be avoided, to ensure that the model does not learn to restore misaligned images. To address this, we propose an algorithm that calculates the barycenter of the kernel through a weighted pixel average, and subsequently shifts it to align the barycenter as closely as possible with the center of the matrix. To do so, we first crop the *region of interest*, which is the zone of the kernel with values different than 0. We compute the barycenter of the region, as a weighted average of the coordinates by the value of the pixel:

$$b_x = \sum_i \sum_j i \times \mathbf{k}[i,j], \tag{5.2}$$

$$b_y = \sum_i \sum_j j \times \mathbf{k}[i,j] \tag{5.3}$$

where b_x and b_y are the coordinates of the barycenter, and $\mathbf{k}$ is the kernel. Once the barycenter coordinates are found, we approximate them to the nearest integer values. We

compute the padding to be added to the region of interest, by computing:

$$p_x = \frac{w}{2} - b_x, \tag{5.4}$$

$$p_y = \frac{h}{2} - b_y, \tag{5.5}$$

where p_x and p_x are the left and top padding, w and h are the width and height of the PSF matrix. We then pad the region of interest with the computed paddings, and add right and bottom paddings to reach the desired size.

The CSB kernel generation is used in the CSB random degradation generator. This class is instantiated with ranges of motion parameters and of trajectory lengths in pixels. The object exposes a *process image* function that works analogously as in the OOF random degradation generator: each time an image is passed to the function, new motion parameters and trajectory length are randomly picked, a trajectory is created, and then it is sampled to create a kernel. This kernel is used to degrade the image by convolution, and the blurred image is returned. In this class, the exposure is fixed to 1, as we use the trajectory length parameter in the trajectory creation class for the same function.

We present an implementation the full degradation algorithm from [1], which accounts for both blur and noise, in a class called NCSB random degradation generator. The main mechanism is the same as the CSB random degradation generator, but also features the application of Poisson and Gaussian noise. In addition to the parameters that controls motion, it has as input, during instantiation a lists of values for exposure, and the two noise parameters λ and σ. Each time the *process image* function is called on an image, a random value for the exposure is extracted from the lists and used for degrading the image, in addition to the application of the blurring kernel. In this class we do not randomize the choice of the trajectory length parameter for the trajectory class, as we want to control it with the exposure value.

We also present a pipeline that resembles the NCSB random degradation generator, but uses the same trajectory and noise levels for each image, and randomizes the exposure time T. This class is called *Constant Trajectory (CT) degradation generator*, and can be used to analyze the performance of deblurring on different values of T, in a similar way to what is done in [1].

5.1.3. Dataset generators

We present a way to integrate the functionalities introduced in the previous section in neural network training. The integrations are presented as dataset generators for both

Tensorflow and Pytorch. The first dataset generator, named *constant kernel dataset generator*, is instantiated by providing a kernel created by either the CSB kernel generator or the OOF kernel generator, previously described. The kernel is then stored as an attribute of the object during instantiation. When a sample is required from the dataset, the corresponding sharp image is retrieved from its designated path and convolved with the stored kernel to produce the degraded image.

The second dataset generator, the *generic degradation generator*, is instantiated by providing a function that takes an image as input and produces another image as output. In our approach, we utilize the previously implemented *process image* functions from the random degradation classes, by passing it to this dataset generator during instantiation. The function is stored as an attribute of the object. When a sample is retrieved from this generator, the corresponding sharp image is loaded from its designated path and the stored function is applied to the image, to produce the degraded version.

5.2. Conditional diffusion models for image deblurring

In this study, we apply conditional diffusion models to the problem of image deblurring. The application of diffusion models to solve the deblurring problem was a novel approach when we started this research. A recently published work [35] uses a similar idea, but employing a fully convolutional neural network as denoiser.

5.2.1. Diffusion model architecture exploration

We use the architecture proposed in Section 4.1.2, modified along with the training tools, to work in a conditional manner, for image-to-image translation. We utilize the conditioning mechanism outlined in Section 4.2, which consists in the concatenation of the noise seed and partially denoised images with the blurred image, at each timestep of the Markov chain.

We explore various architecture modifications, among the ones proposed in Section 4.2. These modifications are:

- multi head attention mechanism, with heads of a fixed size of 64;
- additional attention blocks at resolutions of 32×32 and 8×8, in addition to the ones at 16×16, which we will refer to from now on as *multi resolution attention*;
- BigGAN-style downscale and upscale blocks;

- increase of the latent representation, going from a channel count after the input convolutional layer of 128 to 256.

We test each of these contributions individually to show their impact on performance. We avoid training for the estimation of $\mathbf{\Sigma}_\theta(\mathbf{x}_t, t)$, as other studies [34] have found no practical benefit in doing so. We also avoid the use of γ embedding, as the focus of the study is not on estimation speed.

We train and test our models with both the GoPro and our synthetic dataset, which we present in the following section. We evaluate the performance of our models using PSNR, SSIM, and LPIPS metrics. In the experiments, we show that conditional diffusion models are an effective deblurring solution, with a comparable performance with other GAN-based studies, and that the proposed modifications actually improve the performance.

5.2.2. Training and testing diffusion models on synthetic blur

We present the training of diffusion models for the removal of the camera shake blur as presented in [1], one with the sole blurring and one with the addition of noise, by using the dataset generators presented in the previous section. We show that the model presented is effective in restoring images affected by camera shake blur and with noise components of different magnitudes. We employ the CT degradation pipeline to investigate the presence of an optimal value of T for the reconstruction, and we show that the findings of [1] also apply to our method.

5.3. Arbitrary resolution image deblurring

As a mean to overcome the limitation of fixed resolution, imposed by some blocks in the architecture, we employ an algorithm to deblur arbitrary sized images. The algorithm decomposes the image in multiple patches, and builds a list with all of them. If the patch would be bigger than the remaining part of the image, a mirrored version of the image is used to fill the missing pixels. This list is transformed in multiple batches of the desired batch size, and all the batches are processed with the diffusion model. The output batches are transformed back in a list of patches, and those patches are merged back in an image, which represents the full deblurred image.

To avoid color discrepancies in the final image, we implement a color correction algorithm,

to restore the original brightness and color balance of the patches, by computing

$$\hat{\mathbf{x}}_r = \tilde{\mathbf{x}}_r - \overline{\tilde{\mathbf{x}}}_r + \overline{\mathbf{y}}_r, \tag{5.6}$$

$$\hat{\mathbf{x}}_g = \tilde{\mathbf{x}}_g - \overline{\tilde{\mathbf{x}}}_g + \overline{\mathbf{y}}_g, \tag{5.7}$$

$$\hat{\mathbf{x}}_b = \tilde{\mathbf{x}}_b - \overline{\tilde{\mathbf{x}}}_b + \overline{\mathbf{y}}_b, \tag{5.8}$$

where $\hat{\mathbf{x}}$ is the color restored version of the image, $\tilde{\mathbf{x}}$ is the output image of the network, $\mathbf{y}$ is the blurred image, $\overline{\mathbf{v}}$ is the average of the vector $\mathbf{v}$, and the subscript indicates the color channel.

We propose three ways to merge the patches together, beginning with the basic strategy of dividing the image into non overlapping patches. We call this merging algorithm *trivial merging*. We proceed to make a more sophisticated tool, that computes the estimation of the image on multiple overlapping patches. To do so, it extract patches with a shift s, computed as $s = w/N$, where w is the width of the patch, which is the same as the height, and N is the number of overlapping patches. To combine them in a full image, we set the value of each pixel to the average of its estimated values from all patches that contain it. We call this *overlapping merging*. Lastly, we present a third patch merging algorithm, which averages the patches in a way that gives a pixel more weight the closer it is to the center of the patch, similarly to what is proposed in [15] and presented in Section 3.3.2. To implement this, we concatenate to the image a fourth channel, that contains the values of the weights. The weights are computed as a 2D Hann window. The 1D Hann window is defined as

$$v(n) = \frac{1}{2}\left(1 - \cos\left(2\pi\frac{n}{N}\right)\right), \tag{5.9}$$

where N is the size of the window. The 2D version is defined as

$$w(x, y) = \sqrt{v(x)v(y)}. \tag{5.10}$$

To combine them in a full image, we set the value of each pixel to the average of its estimated values from all patches that contain it, weighted by the value of the window. We call this *Hann merging*.

6 | Experiments

In this section we implement, train and test the proposed solutions. We start by analyzing the behaviour of our model at different training checkpoints, to set the number of epochs to conduct our experiments. We follow by investigating the best architecture for image deblurring on the GoPro dataset, with a comparison with other popular deep learning based solutions. Then, we train and test on our proposed camera shake dataset, to show the effectiveness of our method on both deblurring and noise suppression. Finally, we implement a study on the noise-blur tradeoff, analogously to what is presented in Section 3.1.4.

6.1. Training and testing on the GoPro dataset

We start the experimentation phase by implementing all the tools necessary for the training and testing of conditional diffusion models. We proceed by implementing the architecture from [25], and then modify it into a conditional version. We follow by implementing some of the improvements presented in Section 4.2. Finally, we compare our results with some competing studies in the same field.

6.1.1. Training of the base model

We use Pytorch as a framework for this study. We implement a *Dataset* class, which along with the *Dataloader* class is what handles dynamic loading of the images in Pytorch. Our implementation of dataset class also handles the necessary preprocessing steps for training. Initially, both the blurred and sharp images are loaded. The images can then optionally be resized by a scaling factor. Two cropping options are available, a *random crop* for the training set and a *center crop* for the test set, which ensures the same patch is selected every time. Finally, the images undergo normalization and are converted to tensors.

We implement the training and sampling functions to take a six-channel tensor per batch sample, which is the concatenation of the noisy image $\mathbf{x}_t$ and the blurred image $\mathbf{y}$. Additionally, we incorporated exponential moving average regularization into our approach,

following [25]. To further facilitate our training and testing efforts, we create a package of helpful tools, including model saving and loading capabilities, timed training functionality, and periodic result sampling.

Architecture

We implement a model as close as possible to the original architecture outlined in [25]. We use an input resolution of 64×64, and a base channel count of 128. Since the original article only uses resolutions of 32×32 and 256×256, we have no indication of the appropriate channel multipliers, which determine how much the channel dimension is increased at each stage. To address this, we follow the recommendations from [29] and implement multipliers of 1, 2, 3, and 4. Each stage employs 2 residual blocks. For group normalization, we employ a group size of 32, and we set the exponential moving average regularization factor to 0.9999. We use 1000 diffusion steps for both training and testing. We use a β schedule consisting of linear interpolation of values ranging from $\beta_1 = 0.0001$ to $\beta_{1000} = 0.02$. We call this architecture *base model*.

Dataset

As previously mentioned, our trainings and tests are conducted on the GoPro dataset, given the common use in image deblurring. The dataset test serves as a benchmark for both the comparison of our architecture performance and for evaluating our study with the others in the field. We use randomly extracted patches of a resolution of 64×64 pixels. However, we observed that many patches are composed almost entirely of a constant value, lacking sufficient detail for accurate reconstruction, while others were smaller than the blur size, rendering them hard to restore. As a result, we opted to use a resize factor of $1/2$, for this phase. We call the datasets that we use for this phase *halved GoPro dataset*.

We evaluate the performance of our models using PSNR, SSIM, and LPIPS metrics. We always test on the central patch extracted from every image of the test set. To calculate the evaluation metrics, we developed a tool that scans through the test set, divides it into batches, and navigates the sampling Markov chain, starting from a randomly generated noise image seed, until a fully denoised sample is obtained for each batch sample. Once all the predictions are computed, we calculate the PSNR, SSIM, and LPIPS for each sample and aggregate them through averaging.

Training settings

During training, we employ an L2 type of loss, which aligns with the general trend in the field. We have also tested the use of L1 loss, but did not observe any significant difference

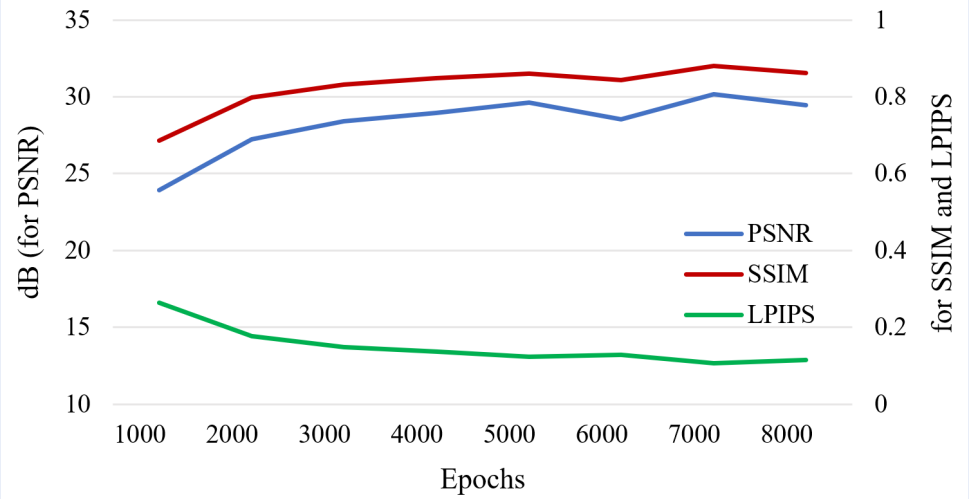


Figure 6.1: Metric measured at various checkpoints, obtained on the halved GoPro dataset, with the architecture from [25].

in performance. Therefore, we opt for L2 for the sake of simplicity and consistency with the theory presented in Chapter 4. We use the Adam optimizer throughout the whole study.

We fix the number of samples per epoch to 1024, randomly selected patches from the available training samples. Due to hardware limitations, we use the maximum possible batch sizes that fit in the RAM of our machines, which are smaller than the typical values used in this field. For instance, while [29] uses a batch size of 2048 for a resolution of 64×64, we use a batch size of 32. We experiment with various learning rates and found that, at a batch size of 32, rates larger than 1×10^{-5} had convergence issues, so we use that value.

Epochs analysis

To decide how many epochs to use to test our results, we obtain a validation set by splitting the training set samples. The GoPro dataset is composed of 21 subsets, each of them containing images from the same video. Among the 21 subsets in the Gopro training set, we use 7 for validation and 14 for training. We test on the validation set every 1000 epochs of 1024 samples, until 8000 epochs. The results are in Table 6.1. We did not observe overfitting phenomena in this training range. As a compromise between performance and training time, we decide to continue our analysis at 4000 training epochs for all our next experiments, unless otherwise specified.

	PSNR ↑	SSIM ↑	LPIPS ↓
Base model	27.10	0.814	0.158
Base model + 256 base channels	28.66	0.860	0.116
Base model + multi head attention	27.61	0.829	0.146
Base model + multi res attention	27.54	0.825	0.147
Base model + BigGAN blocks	27.81	0.838	0.146
Improved model	**29.58**	**0.880**	**0.102**

Table 6.1: Performance of the base model, the structural changes, and the improved model, at 4000 epochs. Trained on prandom patches from the halved GoPro train set, and tested on the central patches from the halved GoPro test set.

6.1.2. Improvements

We train and test the base model, and all the modifications to the architecture proposed in Section 5.2. We implement all the modification, and apply them individually to the base model. We use the same training and testing settings as explained in the previous section. The results for each modification are reported in Table 6.1.

The modification that has the most success is the increase of base channel count, with the drawback of nearly doubling the training time. Both the addition of BigGAN blocks and multi head attention brought increases in performance with non-noticeable training time increase. Multi resolution attention also improved results, with little increase in computation time. The performance of the model is further improved by combining all the modifications, as shown in Table 6.1. We call the architecture with all the modifications *improved model*, from the title of [33].

For the remainder of the study, we will use a final model that incorporates the improvements. Specifically, the model has 256 base channels and employs 64 channels per head, multi head attention. It includes attention layers at resolutions of 32, 16, and 8, as well as BigGAN style blocks for both downsampling and upsampling, unless otherwise specified.

6.1.3. Comparison with other deep learning deblurring solutions

We compare our improved model against some deep learning based blind deblurring methods. As previously stated, the 64×64 resolution model has convergence problems on the full resolution GoPro dataset, due to the small size of the receptive field. Consequently, we proceed to train a model with a resolution of 128×128, utilizing patches extracted from the GoPro training set without resizing factors.

	PSNR ↑	SSIM ↑	LPIPS ↓
Improved model, on central patches	28.30	-	-
Improved model, on full resolution images	28.00	0.870	0.183
Sun et al. [16]	24.64	0.842	-
DeblurGAN [7]	27.20	0.954	-
DeepDeblur [6]	28.30	0.917	0.182
SRN [22]	30.10	0.932	0.788
DeblurGAN v2 [21]	29.55	0.934	0.253

Table 6.2: Comparison with other blind deblurring, neural network based, baseline approaches, tested on the GoPro set. The missing values are due to the limited use of LPIPS in literature.

In the state of art, the test on the GoPro datasets are conducted by measuring the average performance on the full 1280×720 resolution. However, our method is unable to achieve this due to its inherent fixed resolution limitation. We test on a subset of the pixels, composed of the central 128×128 patch of each image from the GoPro test set. The PSNR value obtained in this test is presented in Table 6.2, labeled as *Improved model, on central patches*, along those achieved by the alternatives presented in Chapter 3. We only show PSNR because it is the only one normalized by number of pixels, while SSIM and LPIPS are content-based. The reported results are obtained in the linear subset of the GoPro dataset, which is the same that we used for training. We avoid reporting performance metrics of methods that use different datasets, as we feel that using the same dataset is the most fair way to compare the methods. The LPIPS metric is not as widespread as the other two, and have never been reported in literature for some studies. We show this value for some methods, as presented in [36].

Although the other methods are designed to process images of larger resolutions, and thus they can recognize and reconstruct larger and more complex patterns, the results achieved by our model on low-resolution patches are promising. We expect our method to outperform the alternatives, if trained on higher resolution images, for more time, and with proper data augmentation techniques.

6.1.4. Qualitative assessment

We present some image samples to showcase the potential of our method. The image samples shown in Figures 6.2 and 6.3 illustrate the ability of our model ability to produce sharp, clear images from blurry inputs.

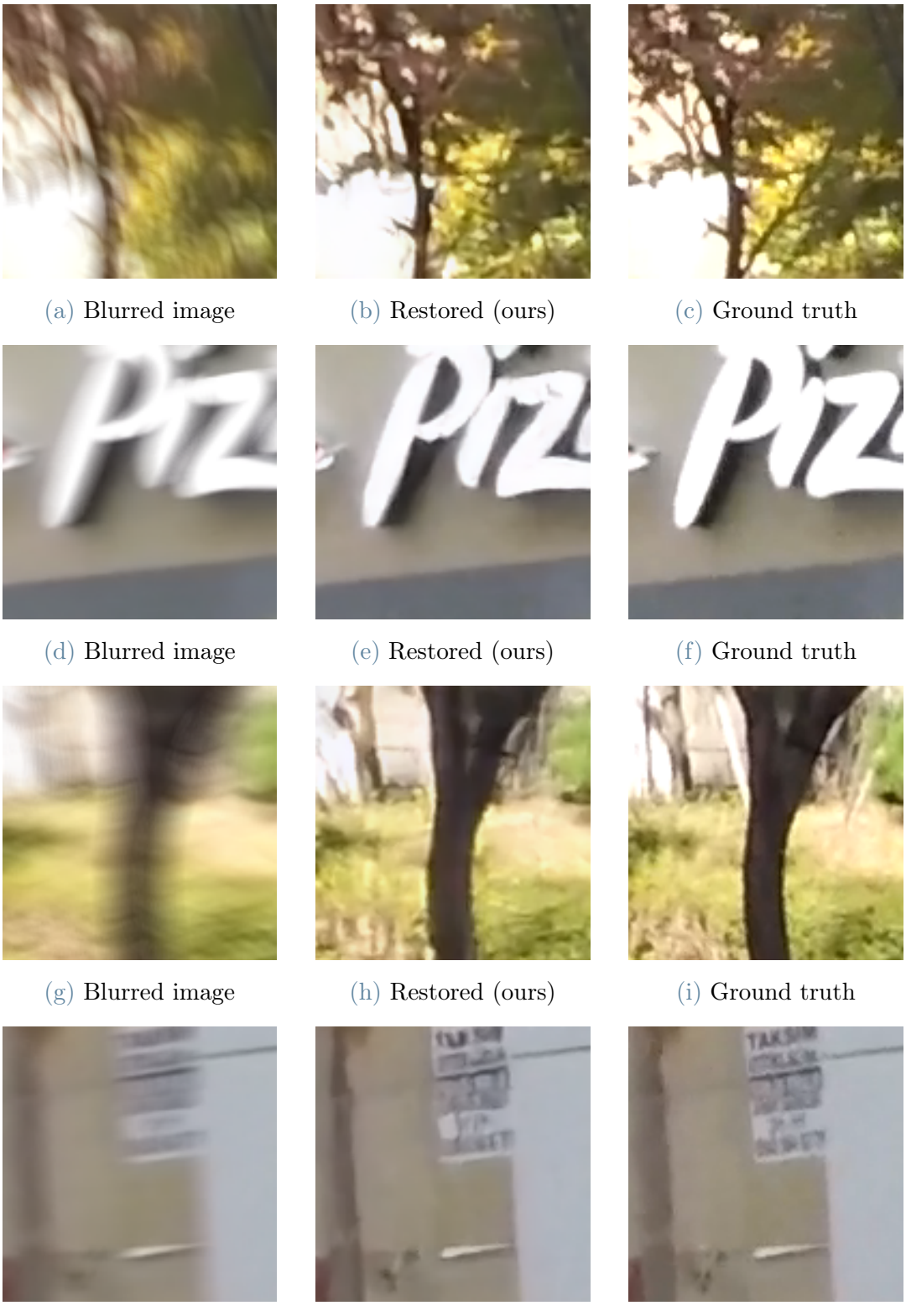
In the sample images, we can observe that the deblurred images appear realistic, despite the occasional reconstruction mistakes. Nevertheless, we believe that these errors can be deemed acceptable considering the fact that the blurring process obfuscates such intricate details, resulting in a substantial loss of information. For example, we can see the branch disappearing from Figure 6.2b and the white stick missing from Figure 6.2h. The model seems to be effective when the image has distinct borders and less detailed textures, such as in Figures 6.3h and 6.2e. When the image is composed of complex patterns, like in Figure 6.3k, the network restores a realistic result, even if it is not exactly the same pattern as in the original image.

6.2. Arbitrary resolution deblur

In this section, we evaluate the performance of the arbitrary resolution image deblurring algorithms introduced in Section 5.3. To conduct the testing, we employ the 128×128 improved model trained in Section 6.1.3 on the GoPro training set. The images utilized for testing are from the full resolution GoPro test set.

Figure 6.6 showcases an example of trivial merging output. The restored image exhibits artifacts at the edges where the patches are stitched together. A closer examination of these artifacts is displayed in Figure 6.4a. When overlapping merging is employed, the artifacts persist, albeit with reduced magnitude, and their occurrence becomes more frequent. This can be attributed to the multiplication of the number of edges that need to be stitched together, by a factor corresponding to the number of overlapping patches. Figure 6.7 serves as an illustration of such output, while a detailed view of the artifacts can be seen in Figure 6.4b. When the Hann merging version of the algorithm is employed the artifacts are completely eliminated. This is due to the fact that the edge of the patch, which is the primary cause of these artifacts, has no influence on the final result. Notable improvements in restoration quality can be observed in Figures 6.8, 6.12, 6.10, and 6.14, where the artifacts are entirely absent, resulting in noticeably superior restoration outcomes compared to other versions ov the algorithm.

The Hann merging algorithm is tested on a subset of images from the GoPro test set. The outcomes of the evaluation are presented in Table 6.2 under the label of *Improved model, on full resolution*. Due to time limitations, we utilized only a subset of the train set, as employing the entire set would have been time-consuming. We test on the first 10 images from each of the 11 subsets of the test set, for a total of 110 images. This test allowed us to also compare the SSIM and LPIPS values with those obtained from other methods. Although there is a minor decrease in PSNR when compared to the patch-based



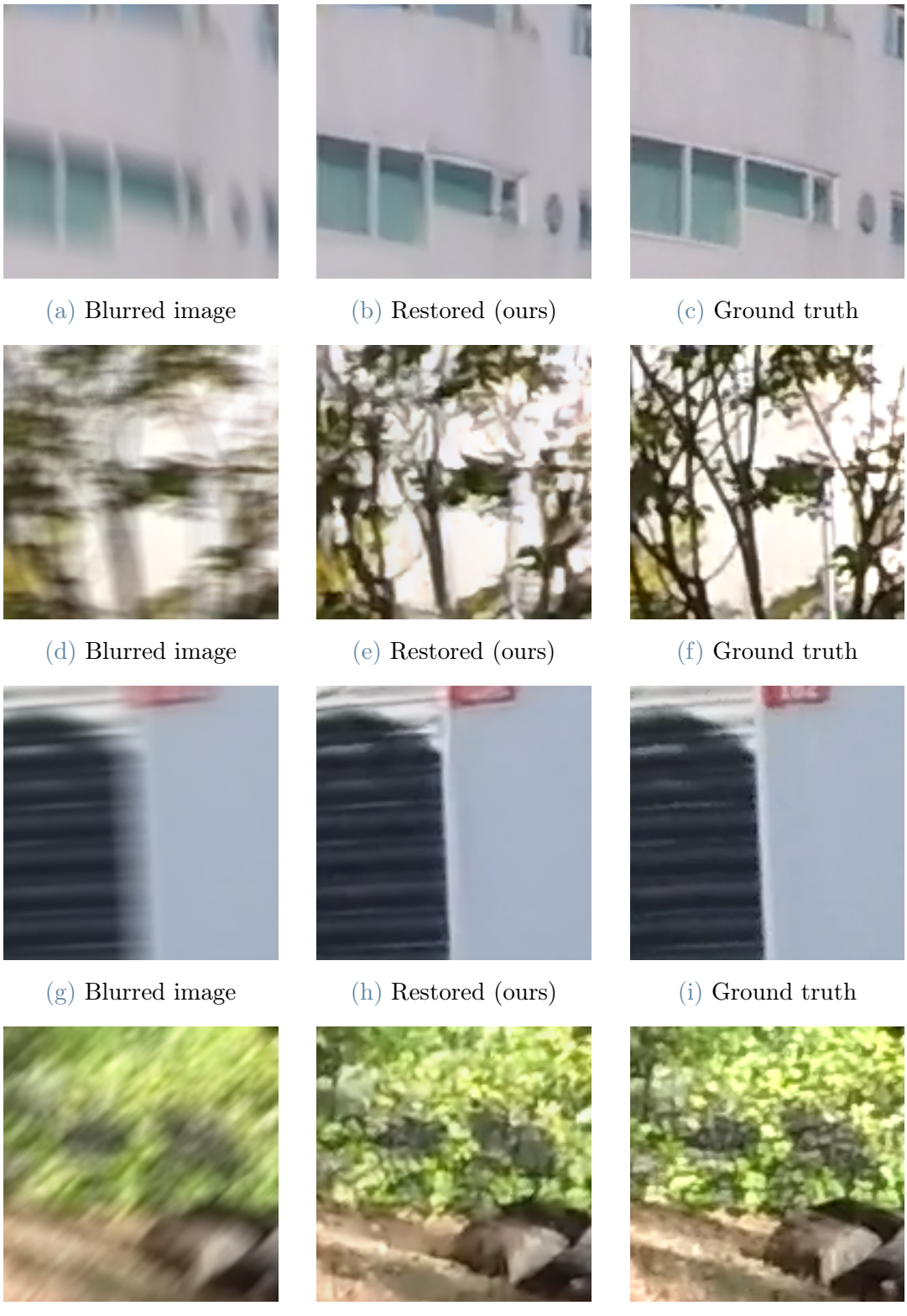
(a) Blurred image (b) Restored (ours) (c) Ground truth

(d) Blurred image (e) Restored (ours) (f) Ground truth

(g) Blurred image (h) Restored (ours) (i) Ground truth

(j) Blurred image (k) Restored (ours) (l) Ground truth

Figure 6.2: Example of patches from the GoPro dataset, processed with the improved model, 128 pixels of resolution.



(a) Blurred image (b) Restored (ours) (c) Ground truth

(d) Blurred image (e) Restored (ours) (f) Ground truth

(g) Blurred image (h) Restored (ours) (i) Ground truth

(j) Blurred image (k) Restored (ours) (l) Ground truth

Figure 6.3: Example of patches from the GoPro dataset, processed with the improved model, 128 pixels of resolution.

(a) Trivial merging (b) Overlapping merging (c) Hann merging

Figure 6.4: Zoomed in details of the full image deblur. Overlapping and Hann merging computed using 4 layers of patches.

test, our approach yields competitive results in terms of LPIPS, indicating the production of high-quality images. As we use a subset of the test set, we cannot directly compare the metrics, but they nonetheless provide the suggestion that diffusion models are a valid alternative to GANs for image deblurring.

6.3. Training and testing on the synthetic camera shake blur dataset

Among the multiple implemented pipelines, we want to test the performance of the restoration on the degradation proposed in [1]. In this section, use the functionalities presented in Section 5.1 to train diffusion models with the most effective architecture that was discovered in Section 6.1. For the whole section, use models with a resolution of 64×64 pixels, and train for 4000 epochs of 1024 samples each.

6.3.1. CSB without noise

In this section, our aim is to examine the performance of our method across various ranges of blur magnitudes. To achieve this, we utilize three different settings of the CSB dataset generator, resulting in the generation of three datasets with distinct blur magnitudes of varying intensities. We train and test on datasets without the application of noise. The following settings apply to the three datasets. We use, as maximum trajectory length, half of the resolution of the kernel, and 1 pixel as minimum. We maintain the motion parameters as close as possible to the ones presented in [1]. The centripetal parameter c is randomly picked in $[0, 0.7]$, the gaussian parameter g in $[0, 10]$, the frequency of big

Figure 6.5: Blurred image.

Figure 6.6: Image processed with trivial merging of the patches.

Figure 6.7: Image processed with the overlapping method. 4 layers of patches.

Figure 6.8: Image processed with Hann merging. 4 layers of patches.

Figure 6.9: Blurred image.

Figure 6.10: Image processed with Hann merging. 4 layers of patches.

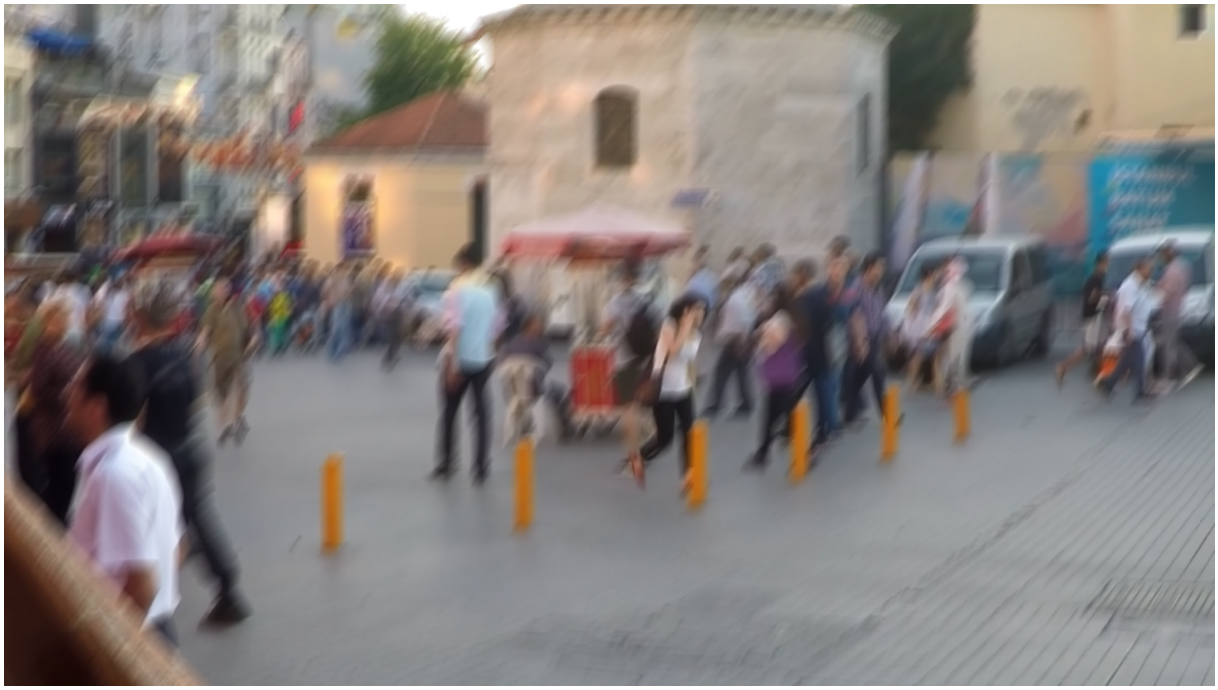


Figure 6.11: Blurred image.

Figure 6.12: Image processed with Hann merging. 4 layers of patches.

Figure 6.13: Blurred image.

Figure 6.14: Image processed with Hann merging. 4 layers of patches.

		PSNR ↑	SSIM ↑	LPIPS ↓
16p	Blurred	27.47	0.752	0.202
	Wiener	29.52	0.870	0.144
	Ours	32.83	0.917	0.058
32p	Blurred	27.47	0.752	0.202
	Wiener	25.53	0.737	0.259
	Ours	28.83	0.796	0.135
64p	Blurred	25.24	0.652	0.283
	Wiener	22.64	0.586	0.406
	Ours	25.76	0.702	0.176

Table 6.3: Results from the experiments in Section 6.3.1. PSF kernels resolutions are reported in the first column.

shakes f_s in $[0, 0.2]$, the trajectory resolution η_t is set to 2000. We found that the anxiety parameter a, which in the article is set to a maximum value of 0.1, generated very shaky kernels. We reduce this value, and set it to be randomly chosen in $[0, 0.01]$. We generate the three datasets using a PSF kernel resolution of 16×16 for the first, 32×32 for the second, and 64×64 pixels for the last.

The dataset generation pipelines are not usable in testing, as a fixed set of images without randomness is required. Therefore, we process and store the first 1024 images from the ImageNet test set using the same pipelines settings employed for the generation of the training sets. We store, along the images, the kernels used to degrade them, to be used for the Wiener deconvolution algorithm. We repeat the process to generate a test set for the 16p kernel dataset, the 32p kernel dataset, and the 64p kernel dataset.

The Wiener deconvolution algorithm, which is detailed in Section 3.2, requires the PSF kernel and a regularization parameter as inputs. To select the optimal regularization parameter, we implemented a grid search algorithm that evaluates the average PSNR on images from the training set in the range of values of $[0.001, 10]$. For each value, 3000 random images are picked from the ImageNet training set, degraded with the CSB random degradation class, and restored using the Wiener deconvolution algorithm. The optimal regularization values for the three datasets previously described are 0.0112 for the 16p kernel degraded dataset, of 0.0483 for the 32p, 0.545 for the 64p. Those regularization values are used to compute the metrics of the deblurring of the Wiener deconvolution algorithms on the test sets. The performance metrics are reported in Table 6.3, along with the same metrics computed on the blurred images. Some of the examples of the outputs are in Figure 6.15.

		PSNR ↑	SSIM ↑	LPIPS ↓
no noise	In	27.47	0.752	0.202
	Out	28.83	0.796	0.135
$\sigma = 1$,	In	24.66	0.643	0.281
$\lambda = 768000$	Out	25.67	0.710	0.184
$\sigma = 2$,	In	22.40	0.481	0.336
$\lambda = 192000$	Out	23.43	0.613	0.248
$\sigma = 4$,	In	19.32	0.312	0.438
$\lambda = 48000$	Out	22.69	0.572	0.309

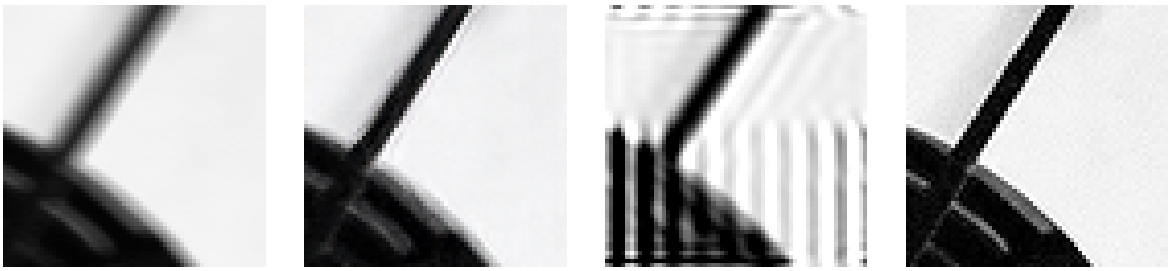
Table 6.4: Results from the experiments in Section 6.3.2. We display in the *In* row the metrics of the degraded images, and in the *Out* row the metrics of the images restored by our solution.

It can be noted, on both the metrics and on the output samples, that the Wiener deconvolution algorithm struggles when the blurring kernel is large in comparison with the size of the blurred image, often leading to images worse than the blurred ones. This is due to the ringing artifacts discussed in Section 3.2, a known problem of this method.

Three improved diffusion models are trained on the three training datasets, for 4000 epochs of 1024 samples each, and then tested on the same test sets. Their performance metrics are reported in Table 6.3, and some of the examples of the output are in Figure 6.15. Our approach reaches both better values on each of the considered metrics and a better visual quality.

6.3.2. CSB with noise

In this section, our objective is to demonstrate the behavior of our method under different noise intensities. To accomplish this, we train three models using the NCSB dataset generator at three distinct settings, each corresponding to increasing levels of noise intensity. We employ the same motion parameters presented in the previous section. We use a kernel size of 32p. We select three levels for λ among the ones presented in [37]. We select values of γ so that the Gaussian noise contribution is visually similar to the Poisson noise. We create a dataset with a low amount of noise, using $\sigma = 1$ and $\lambda = 768000$, a medium one with $\sigma = 2$ and $\lambda = 192000$, and one with a strong noise component, with $\sigma = 4$ and $\lambda = 48000$. The exposure value T is randomly picked between $(0, 1]$. For testing, we process and store the first 1024 images from the ImageNet test set using the same pipelines settings employed for the training sets.

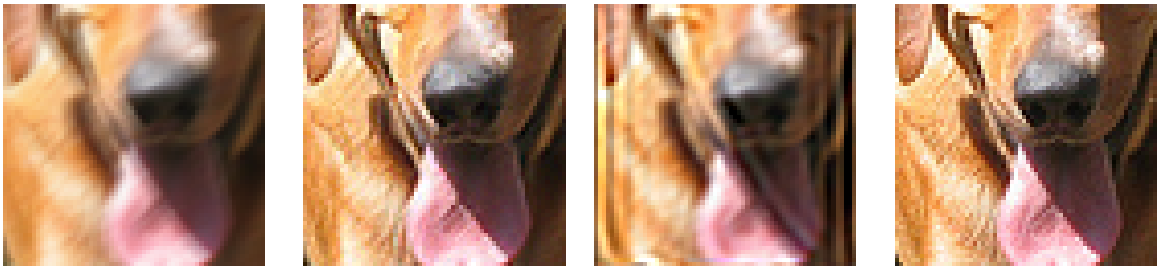


(a) Blurred image (b) Restored (ours) (c) Restored (wiener) (d) Ground truth

(e) Blurred image (f) Restored (ours) (g) Restored (wiener) (h) Ground truth

(i) Blurred image (j) Restored (ours) (k) Restored (wiener) (l) Ground truth

(m) Blurred image (n) Restored (ours) (o) Restored (wiener) (p) Ground truth



(q) Blurred image (r) Restored (ours) (s) Restored (wiener) (t) Ground truth

Figure 6.15: Samples of restored images using Wiener deconvolution and our method. 64p images, blurred with random 32p kernels.

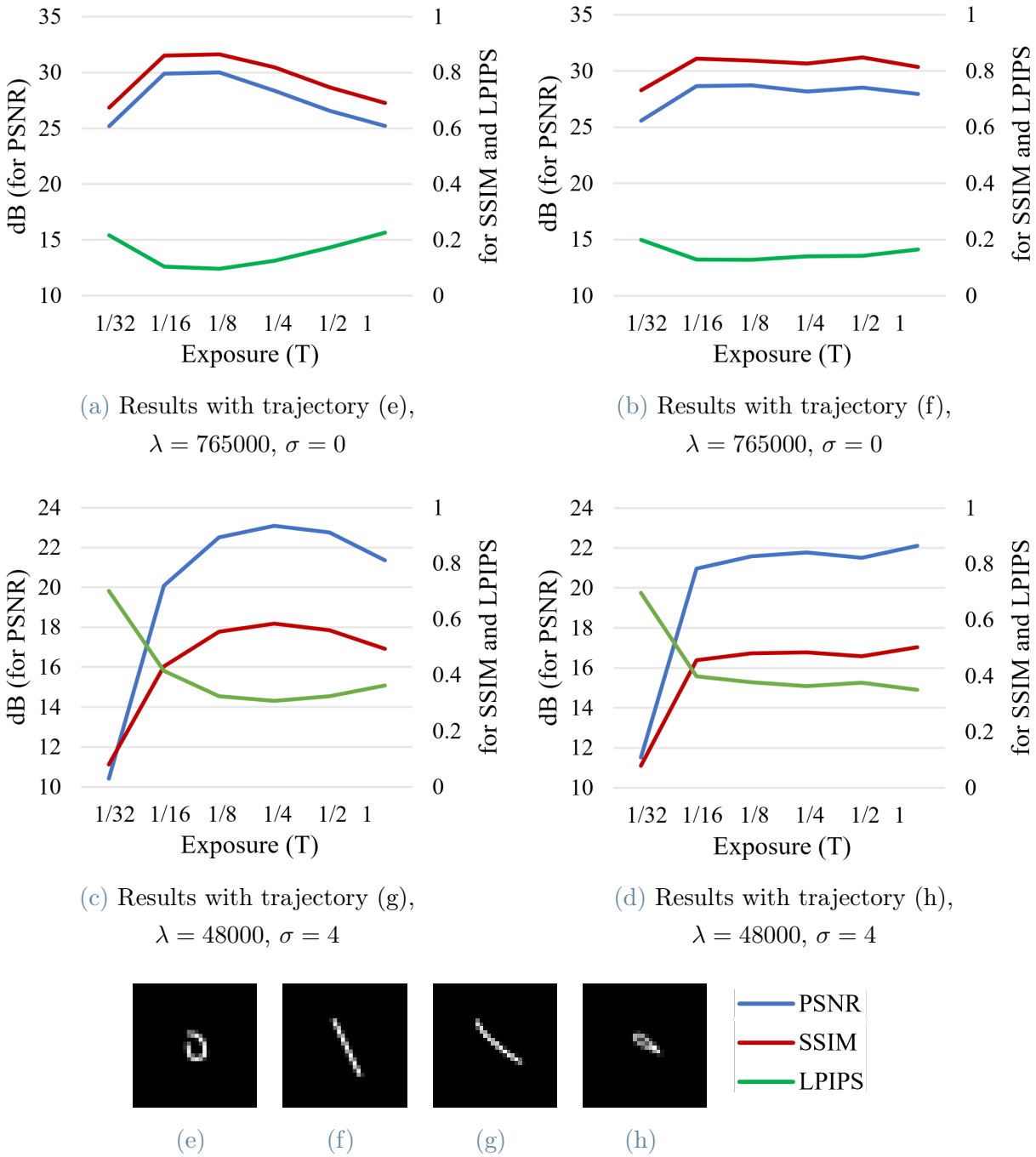
We train three models on the three datasets, for 4000 epochs of 1024 samples each, and test them on the respective test sets. The results are reported in Table 6.4, along with the results on a dataset without noise. We report both the metrics of the blurred images and the restored ones. The table shows that our method demonstrates comparable success in restoring degradation across all three noise levels, as well as in noise-free conditions.

6.3.3. Noise-blur tradeoff exploration

We replicate the experimental setup outlined in [1], aiming to explore the existence of an optimal exposure value in which the reconstruction is the best. Our experiment consists in training models on datasets composed of images extracted from the ImageNet training set, degraded with a fixed trajectory and noise level, with T ranging from 0 to 1. To assess the reconstruction performance after training, we employ six distinct test sets comprising 100 images from the ImageNet test set, subjected to the same degradation trajectory, noise levels, and a different T value for each test set. The metrics that we display at each exposure value are the average of the reconstruction metrics on che corresponding subset.

This process is repeated for four distinct trajectories, and two noise settings. One setting is the same used in [1], of $\lambda = 765000$ and $\sigma = 0$, and the second is one with stronger noise of $\lambda = 48000$ and $\sigma = 4$. The results are depicted in Figure 6.16. The findings validate the observations made on the deconvolution algorithms discussed in [1]: when the trajectory is linear, like in the case in Figure 6.16a, or semi linear, like in the case in Figure 6.16d, there is a discernible optimal T. When the trajectory is more complex, like in the case of Figures 6.16b and 6.16d, there is no clear optimum value and the performances levels off after a threshold value.

A qualitative analysis of the tradeoff is displayed in Figures 6.17 to 6.20. In Figure 6.18 the optimal reconstruction is particularly clear, with the reconstructions with shortest and longest exposure being visibly worse than the middle ones. On the other hand, in Figure 6.19, the reconstruction quality is good along multiple exposure values.



(a) Results with trajectory (e), $\lambda = 765000$, $\sigma = 0$

(b) Results with trajectory (f), $\lambda = 765000$, $\sigma = 0$

(c) Results with trajectory (g), $\lambda = 48000$, $\sigma = 4$

(d) Results with trajectory (h), $\lambda = 48000$, $\sigma = 4$

(e) (f) (g) (h)

Figure 6.16: Performance of the reconstruction on constant trajectory, as exposure time changes.

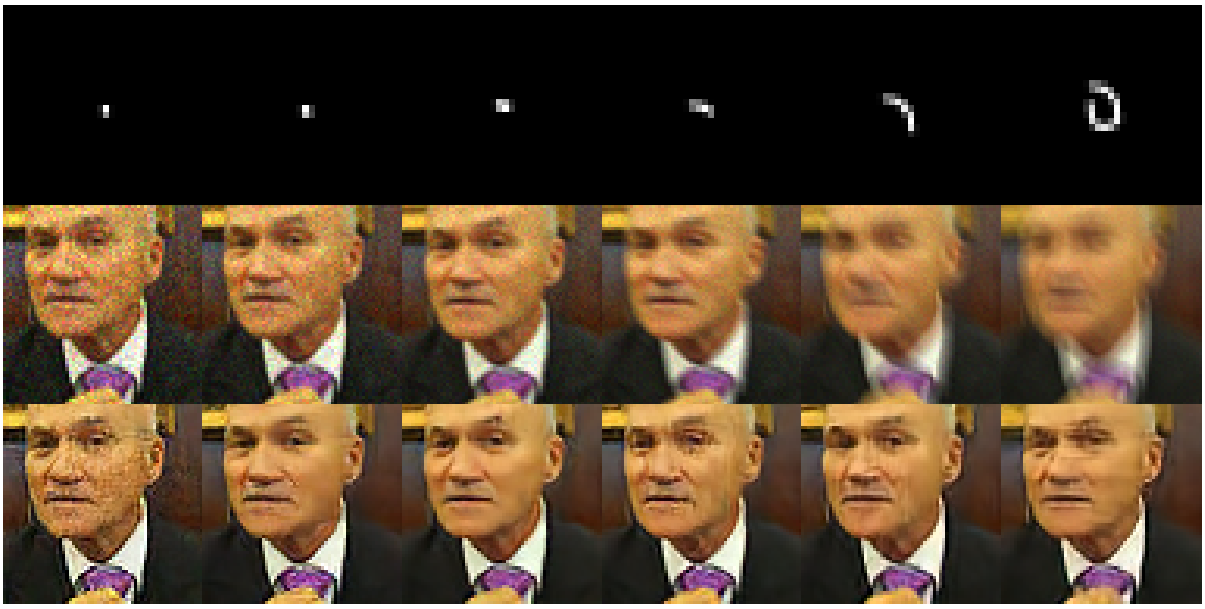


Figure 6.17: Images deblurred with a constant trajectory. Example with CSB kernel. Images degraded at exposure values of 1/32, 1/16, 1/8, 1/4, 1/2, 1. PSF kernels on the first row, degraded images on the second, reconstructions by our model on the third.

Figure 6.18: Images deblurred with a constant trajectory. Example with linear kernel. Images degraded at exposure values of 1/32, 1/16, 1/8, 1/4, 1/2, 1. PSF kernels on the first row, degraded images on the second, reconstructions by our model on the third.

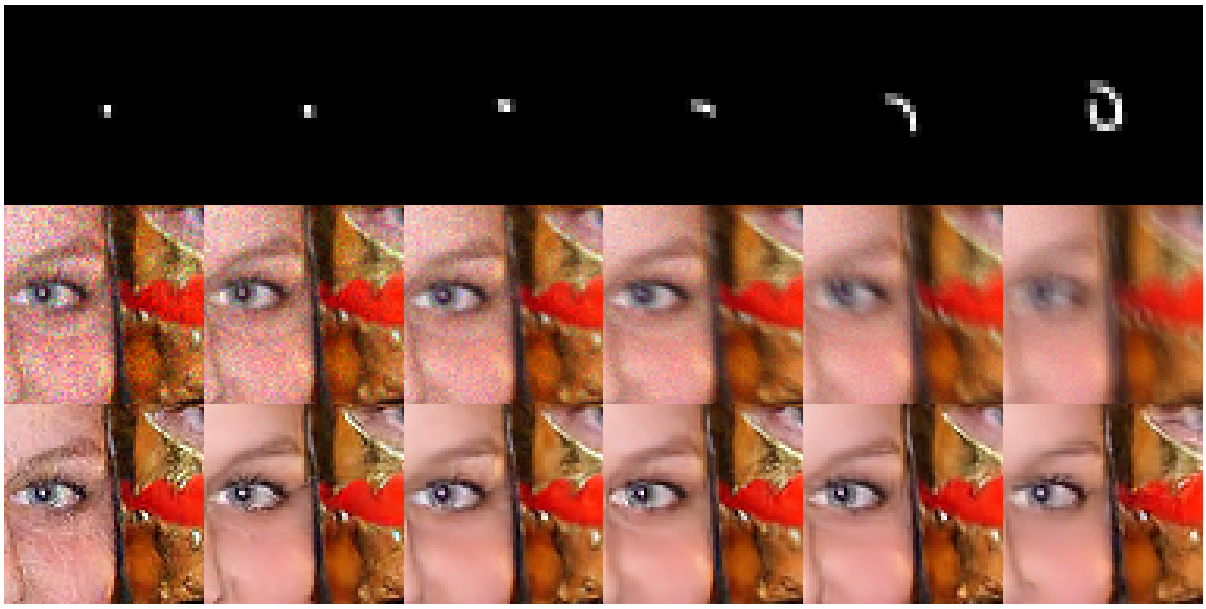


Figure 6.19: Images deblurred with a constant trajectory. Example with CSB kernel. Images degraded at exposure values of 1/32, 1/16, 1/8, 1/4, 1/2, 1. PSF kernels on the first row, degraded images on the second, reconstructions by our model on the third.

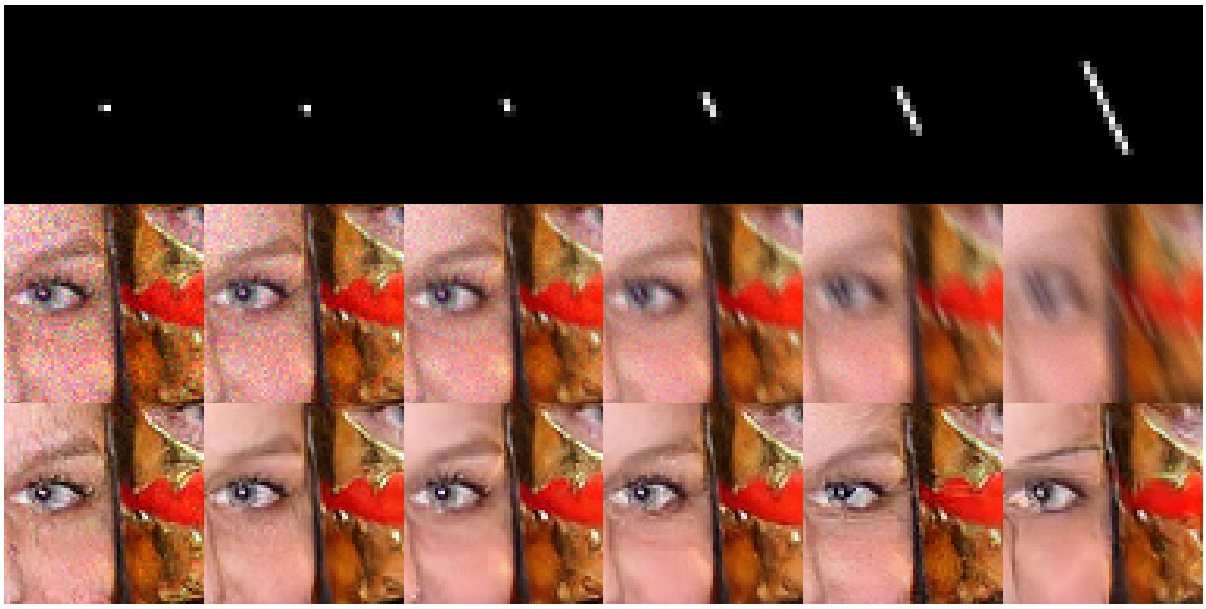


Figure 6.20: Images deblurred with a constant trajectory. Example with linear kernel. Images degraded at exposure values of 1/32, 1/16, 1/8, 1/4, 1/2, 1. PSF kernels on the first row, degraded images on the second, reconstructions by our model on the third.

7 | Conclusion

The objective of this study was to develop a set of tools for training neural networks, and to use them to investigate the performance of conditional diffusion models as a blind deblurring technique. To explore the effectiveness of the proposed method we utilized the GoPro dataset, which consists of pairs of sharp and heterogeneously blurred images. Due to hardware constraints we used low-resolution models, but we obtained promising results that were competitive with baseline blind deblurring neural networks. We expect that these findings might generalize well to higher resolutions, and to be able to surpass other studies if given more trining time and proper augmentation techniques, but further research is needed. Our dataset generators serve as effective tools for simulating blurred and degraded images, providing a valuable training pipeline for neural networks. We believe this resources to have the potential to contribute to future research in the field. The dataset generators developed in this study were employed to train diffusion models, and assess their performance in multiple different degradation conditions. We also explored the noise-blur tradeoff, as in [1], and found that our method behaves similarly to the deblurring algorithms analyzed in the paper.

Future development

We conducted extensive testing on the performance of conditional diffusion models in image deblurring by using our pipeline. It would be worth exploring how other deep learning solutions perform under similar test conditions, examining their adaptability to various blurring sizes and noise levels. Additionally, it would be insightful to investigate if these solutions demonstrate comparable noise-blur tradeoff reconstruction profiles. We leave these tasks to future research.

One major limitation of our approach is the significant computation time required. Although our focus was not on reducing it, several studies suggest strategies to address this issue, by reducing the number of timesteps required and the size of the model. It would be valuable to investigate the extent to which we can reduce the computation time while maintaining effective deblurring performance.

The field of diffusion models is relatively young, with the article that sparked general interest [25] published in 2020. As a result, numerous new studies are conducted each month proposing ways to enhance performance. While we chose to focus on what we believed were the most significant studies at the start of our research, many more have emerged since then. We are optimistic that incorporating these new enhancements could further improve image deblurring using diffusion models.

Bibliography

[1] G. Boracchi and A. Foi, "Modeling the performance of image restoration from motion blur," *IEEE Transactions on Image Processing*, vol. 21, no. 8, pp. 3502–3517, 2012.

[2] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[3] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 586–595, 2018.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[5] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pp. 740–755, Springer, 2014.

[6] S. Nah, T. H. Kim, and K. M. Lee, "Deep multi-scale convolutional neural network for dynamic scene deblurring," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[7] O. Kupyn, V. Budzan, M. Mykhailych, D. Mishkin, and J. Matas, "Deblurgan: Blind motion deblurring using conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8183–8192, 2018.

[8] N. Wiener, *Extrapolation, Interpolation, and Smoothing of Stationary Time Series, with Engineering Applications*. MIT Press, 1949.

[9] L. Xu, J. S. Ren, C. Liu, and J. Jia, "Deep convolutional neural network for image deconvolution," *Advances in neural information processing systems*, vol. 27, 2014.

[10] P. Perona, "Deformable kernels for early vision," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 17, no. 5, pp. 488–499, 1995.

[11] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Deep image prior," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 9446–9454, 2018.

[12] J. Liu, Y. Sun, X. Xu, and U. S. Kamilov, "Image restoration using total variation regularized deep image prior," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 7715–7719, Ieee, 2019.

[13] G. Mataev, P. Milanfar, and M. Elad, "Deepred: Deep image prior powered by red," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 0–0, 2019.

[14] C. J. Schuler, M. Hirsch, S. Harmeling, and B. Schölkopf, "Learning to deblur," *arXiv preprint arXiv:1406.7444*, 2014.

[15] A. Chakrabarti, "A neural approach to blind motion deblurring," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III 14*, pp. 221–235, Springer, 2016.

[16] J. Sun, W. Cao, Z. Xu, and J. Ponce, "Learning a convolutional neural network for non-uniform motion blur removal," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 769–777, 2015.

[17] D. Gong, J. Yang, L. Liu, Y. Zhang, I. Reid, C. Shen, A. Van Den Hengel, and Q. Shi, "From motion blur to motion flow: A deep learning solution for removing heterogeneous motion blur," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2319–2328, 2017.

[18] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[19] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.

[20] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," *Advances in neural information processing systems*, vol. 30, 2017.

[21] O. Kupyn, T. Martyniuk, J. Wu, and Z. Wang, "Deblurgan-v2: Deblurring (orders-

of-magnitude) faster and better," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 8878–8887, 2019.

[22] X. Tao, H. Gao, X. Shen, J. Wang, and J. Jia, "Scale-recurrent network for deep image deblurring," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8174–8182, 2018.

[23] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," *Advances in neural information processing systems*, vol. 28, 2015.

[24] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *International Conference on Machine Learning*, pp. 2256–2265, PMLR, 2015.

[25] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[27] Y. Wu and K. He, "Group normalization," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.

[28] A. Brock, J. Donahue, and K. Simonyan, "Large scale gan training for high fidelity natural image synthesis," *arXiv preprint arXiv:1809.11096*, 2018.

[29] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," *Advances in Neural Information Processing Systems*, vol. 34, pp. 8780–8794, 2021.

[30] G. Batzolis, J. Stanczuk, C.-B. Schönlieb, and C. Etmann, "Conditional image generation with score-based diffusion models," *arXiv preprint arXiv:2111.13606*, 2021.

[31] C. Saharia, J. Ho, W. Chan, T. Salimans, D. J. Fleet, and M. Norouzi, "Image super-resolution via iterative refinement," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[32] N. Chen, Y. Zhang, H. Zen, R. J. Weiss, M. Norouzi, and W. Chan, "Wavegrad: Estimating gradients for waveform generation," *arXiv preprint arXiv:2009.00713*, 2020.

[33] A. Q. Nichol and P. Dhariwal, "Improved denoising diffusion probabilistic models," in *International Conference on Machine Learning*, pp. 8162–8171, PMLR, 2021.

[34] C. Saharia, W. Chan, H. Chang, C. Lee, J. Ho, T. Salimans, D. Fleet, and M. Norouzi, "Palette: Image-to-image diffusion models," in *ACM SIGGRAPH 2022 Conference Proceedings*, pp. 1–10, 2022.

[35] J. Whang, M. Delbracio, H. Talebi, C. Saharia, A. G. Dimakis, and P. Milanfar, "Deblurring via stochastic refinement," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16293–16303, 2022.

[36] K. Zhang, W. Ren, W. Luo, W.-S. Lai, B. Stenger, M.-H. Yang, and H. Li, "Deep image deblurring: A survey," *International Journal of Computer Vision*, vol. 130, no. 9, pp. 2103–2130, 2022.

[37] G. Boracchi and A. Foi, "Uniform motion blur in poissonian noise: Blur/noise trade-off," *IEEE Transactions on Image Processing*, vol. 20, no. 2, pp. 592–598, 2010.

List of Figures

List of Tables