MASTER THESIS

Bc. David Honzátko

Generative Neural Networks in Image Reconstruction

Department of Software and Computer Science Education

Supervisor of the master thesis: RNDr. Michal Šorel, Ph.D.
Study programme: Computer Science
Study branch: Artificial Intelligence

Prague 2017

Title: Generative Neural Networks in Image Reconstruction

Author: Bc. David Honzátko

Department: Department of Software and Computer Science Education

Supervisor: RNDr. Michal Šorel, Ph.D., Institute of Information Theory and Automation

Abstract: Recent research in generative models came up with a promising approach to modelling the prior probability of natural images. The architecture of these prior models is based on deep neural networks. Although these priors were primarily designed for generating new natural-like images, its potential use is much broader. One of the possible applications is to use these models for solving the inverse problems in low-level vision (i.e., image reconstruction). This usage is mainly possible because the architecture of these models allows computing the derivative of the prior probability with respect to the input image. The main objective of this thesis is to evaluate the usage of these prior models in image reconstruction. This thesis proposes a novel model-based optimization method to two image reconstruction problems – image denoising and single-image super-resolution (SISR). The proposed method uses optimization algorithms for finding the maximum-a-posteriori probability, which is defined using the above mentioned prior models. The experimental results demonstrate that the proposed approach achieves reconstruction performance competitive with the current state-of-the-art methods, especially regarding SISR.

Keywords: neural network generative image reconstruction MAP image prior

I would like to dedicate this thesis to all people that supported me throughout the research and writing process. I would like to especially thank my supervisor Michal Šorel for his expertise, friendly attitude, patient explanations and hardware support.

Contents

Introduction

Image reconstruction is an image processing problem of estimating uncorrupted images from the degraded ones. The degradation can have a form of noise, low resolution, missing part, blur, distortion, etc.; thus, the corresponding reconstruction tasks are denoising, super-resolution, inpainting, deblurring, inverse distortion, etc. Since many of these degradations are inevitable during the natural-image acquisition process, the image reconstruction is nowadays an essential operation.

There has been an extensive research addressing this topic for last several decades. Many specialized methods for each reconstruction tasks were introduced as well as some generic ones. These methods are usually based on filter theory, spectral analysis, wavelets, partial differential equations, or stochastic modelling. In this work, we focus on the latter one. We model images as random variables that follow some prior distribution, which prefers images capturing real-world scenes (i.e., natural images). This prior is then used to reconstruct the degraded image using a maximum-a-posteriori (MAP) principle, where the objective is to maximize the conditional probability of the reconstructed image given the degraded one. For such approach, it is necessary to have a very good prior model of the natural images. We obtain such model using machine learning.

In recent years, there has been huge progress in the field of machine learning. In the beginning, in the image related problems, machine learning was applied to image analysis; however, with the major break-through in deep feed-forward convolutional neural networks, it is starting to be extensively used also in image processing including image reconstruction. The feed-forward networks now provide the state-of-the-art performance for many reconstruction tasks. However, the drawback of these methods is the fact that they are usually trained for a certain task or a small set of tasks only. Conversely, in this work, we do not use the neural networks for direct reconstruction, but we employ them for a much more generic task, image prior modelling.

The long-term goal of machine learning is to build a model that understands the real world. For such task, there is an almost infinite number of easily accessible training data. However, even if we abstract from the detail and consider only the visual information about the world, learning of such model is not straightforward since it is necessary to specify how to evaluate the model. Following the famous quote of Richard Feynman: "What I cannot create, I do not understand", recent research came with a promising approach of training models that are able to create data like the ones on which they are trained. Such models are called *generative models*. These models can be trained in various manners; however, the core idea is the same. The capacity of the model (i.e., the number of parameters) is much smaller than the amount of training data; hence, the model is forced to learn key features of the data. The ideal generative models should be therefore able to capture the essential features of the real-world, in our case natural images. There are many types of generative models, but for the MAP approach to image reconstruction the most useful models are those that can predict the prior probability of an image.

The main objective of this work is to evaluate the employment of the latest image prior models that are based on generative neural networks in MAP based image reconstruction. We focus on two image reconstruction tasks – image denoising and super-resolution. We also develop a prototype implementation and evaluate it against the existing state-of-the-art methods.

The image reconstruction problems with the main focus on denoising and super-resolution are presented in Chapter 1. We mathematically describe the problems, briefly introduce the existing methods and propose the MAP approach to these reconstruction tasks. Chapter 2 discusses the natural image prior modelling with emphasis on models based on generative neural networks. Particularly, models based on PixelCNN architecture are presented in detail. Chapter 3 briefly introduces the programming framework and offers the implementation details of the proposed image reconstruction methods. Finally, Chapter 4 discusses some design choices and empirically evaluates the methods against the current state-of-the-art.

1. Image Reconstruction

Image reconstruction is a typical example of an *inverse problem.* Given a set observations resulting from a known *forward model* the task is to reconstruct its cause. Regarding the image reconstruction, the set of observations is formed by images of some scene, and the forward model represents the imperfect capturing process. We simplify this task by the assumption that there is some *original image*, and that the captured (degraded) images arose from the following linear forward model:

$$y = Ax + \eta, \tag{1.1}$$

where y denotes the degraded image, x stands for the original image, η is noise, and A denotes a degradation matrix that determines the reconstruction problem. For example, if A is an identity, the reconstruction problem is denoising. A can also represent a down-sampling operator or a convolution with some kernel (e.g., Gaussian blur kernel).

We usually assume η to be an additive white Gaussian noise (AWGN) with a standard deviation σ

$$\eta \sim N(0, I\sigma^2).$$

Such noise model well approximates the noise produced by conventional cameras.

What makes the image reconstruction complicated is the fact that it is often ill-posed (i.e., its solution is ambiguous or unstable). There are numerous approaches how to tackle this ill-posedness. In this work we focus on solving the image reconstruction through the maximum-a-posteriori (MAP) principle:

$$\hat{x} = \arg\max_z p(z|y) = \arg\max_z \frac{p(y|z)p(z)}{p(y)} = \arg\max_z p(y|z)p(z).$$

In this formula, $\hat{x}$ denotes the most probable estimate of the original image x given the observed image y, $p(y|z)$ is the likelihood determined by the forward model, and $p(z)$ denotes the image prior. When solving MAP problems on computers, it is more convenient to work with negative log probabilities rather than with probabilities themselves:

$$\hat{x} = \arg\min_z - \ln(p(z|y)) = \arg\min_z \left[- \ln(p(y|z)) + - \ln(p(z))\right]. \tag{1.2}$$

The density of the likelihood term $p(y|z)$ can be directly inferred from the forward model. For the linear model with AWGN defined in equation (1.1) it is computed in the following manner:

$$\begin{aligned} p(y|z) &= (\det(2\pi I\sigma^2))^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\eta - 0)' I\sigma^{-2}(\eta - 0)\right) \\ &= \left(2\pi\sigma^2\right)^{-\frac{N}{2}} \exp\left(-\frac{1}{2\sigma^2}(y - Az)'(y - Az)\right). \end{aligned}$$

Negative log-likelihood is then:

$$- \ln p(y|z) = \frac{N}{2} \ln\left(2\pi\sigma^2\right) + \frac{1}{2\sigma^2}(y - Az)'(y - Az). \tag{1.3}$$

Unfortunately, the exact inference of the density of the true image prior $p(z)$ is infeasible if not impossible. Therefore, we approximate the prior from known or learned properties of the image. A traditional approach was to approximate it from the first and second order derivatives of the image. Newer methods usually learn the prior distribution from real-world examples; however, until recently, it was infeasible to learn a good prior model for generic natural images. That changed with the recent breakthrough in the generative models, and we will take a closer look at it in Chapter 2.

In this work, we present MAP approach to two reconstruction problems – image denoising and single-image super-resolution. We describe these problems in the next two sections. We formulate their forward models, define the corresponding likelihood terms and outline complete solutions following the equations (1.2) and (1.3).

1.1 Denoising

Images acquired by optical sensors of cameras are always corrupted by noise. Especially if the target scene is insufficiently illuminated, and the camera is forced to increase the sensitivity of the sensor (ISO). Since the presence of the noise in the acquired images is inevitable, many methods for noise attenuation were developed.

1.1.1 Related Work

Basic image denoising methods suppress the sharp differences between the adjacent pixels of the degraded image. Although this approach efficiently attenuates the noise, it also suppresses the detail and leads to blurry images that are often less desirable than the noisy images themselves. Hence, the main objective of image denoising is to attenuate the noise while preserving as much detail as possible.

Some current state-of-the-art methods are based on non-local self-similarities. It refers to the fact that natural images often exhibit repetitive patterns such as geometric shapes and textures. Similar patches in the degraded image are identified, grouped, collaboratively denoised and aggregated to the final image estimate. The collaborative denoising can have different forms. For instance, Block Matching and 3D Filtering (BM3D) method (Dabov et al. [2007a]) employs the fact that these groups of patches have much sparser representation after a three-dimensional decorrelating transform. Groups are transformed, filtered and transformed back. Another method exploiting non-local similarities is Weighted Nuclear Norm Minimization (WNNM) (Gu et al. [2014]). This method uses the MAP approach on image patches and approximates the negative log prior using weighted nuclear norm of a matrix formed by the group of similar patches. The minimization of the nuclear norm is a surrogate for minimizing the rank, which is computationally demanding as it is an NP-hard task.

Another state-of-the-art approach to denoising has risen with the latest development in deep learning. We list at least two successful methods: DnCNN (Zhang et al. [2017a]) and IRCNN (Zhang et al. [2017b]). For denoising, these methods share the overall architecture. They consist of several convolutional layers connected with rectified linear units (ReLUs), and they use residual learning and batch normalization techniques. They differ in the number of layers and type of convolutions. DnCNN uses 17 to 20 classic convolutional layers. IRCNN, on the other hand, employs more expensive dilated convolutions, but it is redeemed by using roughly two times smaller number of layers. The main problem of feed-forward denoising, which also applies to IRCNN, is that the best denoising performance is achieved when the network is trained for a certain noise variance; however, DnCNN overcomes this issue and gives competitive results for blind denoising as well.

Our primary interest in this work lies in methods based on the MAP principle. Many of such methods achieve state-of-the-art results. They can be divided into two main categories – model-based methods and discriminative learning methods. The model-based methods learn a generic prior model and use optimization algorithms. Into this category we can classify also the methods proposed in this work. Conversely, the discriminative learning methods encapsulate the entire optimization process into a non-linear function, which is learned on pairs of original and degraded images. So far, the most promising results have provided discriminative learning methods that are based on deep learning.

One type of the discriminative learning methods are those that simulate the gradient descent optimization with a pre-defined number of steps. We can list at least Recurrent Inference Machines (RIM) (Putzky and Welling [2017]), whose architecture is based on convolutional recurrent neural networks, or Unrolled Optimization with Deep Priors (ODP) (Diamond et al. [2017]). The latter uses a deep convolutional neural network (CNN) to model the prior; however, there is a separate CNN for each gradient step that allows learning a specialized prior in each step.

The second mentioned category of MAP methods, model-based methods, optimize the image using a generic prior model of natural images. Learning such model is a severe task; therefore, some methods learn the prior model of small image patches of fixed size only. One of these methods is Expected Patch Log Likelihood (EPLL) (Zoran and Weiss [2011]). The optimization is made on an entire image, but the image prior is approximated by a product of all patch priors. Authors achieved best results with Gaussian Mixture Model for the patch prior; however, this method works with any patch prior model.

1.1.2 MAP Approach

The objective of image denoising is to obtain an estimate of the original image x given the noisy image y and noise variance σ. We consider the AWGN noise $\eta \sim N(0, I\sigma^2)$ to be the only source of degradation in the image; therefore we assume the simplified forward model (1.1) where A is an identity:

$$y = x + \eta.$$

By substituting A with the identity, the likelihood term defined in equation (1.3) representing the forward model will change to the following:

$$
\begin{aligned}
-\ln p(y|z) &= \frac{N}{2}\ln\left(2\pi\sigma^2\right) + \frac{1}{2\sigma^2}(y - Iz)'(y - Iz) \\
-\ln p(y|z) &= \frac{N}{2}\ln\left(2\pi\sigma^2\right) + \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - z_i)^2.
\end{aligned}
$$

If we incorporate this term into the MAP equation (1.2), we can omit all members that do not depend on z as it does not affect the minimization. Putting all pieces together, we get the complete MAP solution to image denoising:

$$\hat{x} = \arg\min_z \frac{1}{2\sigma^2}\sum_{i}^{N}(y_i - z_i)^2 - \ln(p(z)). \tag{1.4}$$

1.2 Single-Image Super-Resolution

The level of detail captured in an image is directly affected by the image resolution, which is naturally limited by the camera hardware. It depends mostly on the resolution of the image sensor and on the optical system of the camera (e.g., lenses). To increase the resolution beyond the hardware limitations, one can split the scene into multiple images that are captured separately and join these images into one image of higher resolution. However, this approach is possible only if we have suitable hardware equipment and enough time. Consequently, several software techniques for increasing the image resolution, collectively called *super-resolution*, have been developed.

Basic super-resolution is a task of increasing resolution of an image given multiple slightly different images of the same scene. The images are aligned with sub-pixel accuracy and combined into a single image of higher resolution. Given a sufficient number of images of one static scene, the only limiting factor is usually the optical system of the camera. However, sometimes the scene is so dynamic that it is not possible to capture multiple images of it, or we are just provided with a single image of the scene. Such problem is called *single-image super-resolution* (SISR), and the task is to increase the resolution of an image given only that image. In this case, traditional aligning methods that combine images and do not use any prior information about them cannot be used. Nevertheless, the recent research on SISR produced plenty of new methods specialized for this problem.

1.2.1 Related Work

As baseline approach to SISR we consider simple interpolation-based methods like bilinear or bicubic. Unfortunately, these methods tend to create overly smooth images with chessboard effect at pixel boundaries. Any method that employs some prior knowledge about natural images should, therefore, provide better results than these simple interpolations. Methods are usually evaluated in terms of reconstruction performance which is mostly measured using PSNR and Structural Similarity Index (SSIM) (Wang et al. [2004]) computed between the original high-resolution (HR) image and degraded low-resolution (LR) image.

In the general case, the reconstruction of the subpixel details of a natural scene from a single image is by definition ill-posed as there may be multiple natural scenes that would result in the same image. However, having the prior knowledge about the structure of natural images, we know that some details are very probable given the LR image.

First methods for SISR used hard-coded prior knowledge. The most studied priors for super-resolution are *generic smoothness prior* and *edge smoothness prior* (Dai et al. [2007]). Other hard-coded priors include for example *gradient profile prior* (Sun et al. [2008]) and priors based on total variation (Rudin et al. [1992]). Methods usually incorporate these priors into an optimization term that they solve. These methods are particularly good in preserving the edges in the image with increased resolution; however, they usually fail to reconstruct fine textures.

In order to reproduce the textures more precisely, some methods (Glasner et al. [2009], Tsurusaki et al. [2016]) exploit the fact that the textures often consist of repetitive patterns. They identify these patterns and use them to improve the reconstruction. Such methods are denoted as self-similarity based super-resolution.

All the aforementioned methods use only the input image and some hard-coded prior knowledge. Although they well preserve the edges and sometimes even the textures, they are unable to model arbitrary subpixel details of the natural scene as there is no information about them in the LR image.

Another approach to SISR that deals with the previously stated problems, is machine learning, where the prior (explicit or implicit) is learned from the examples; hence also called *example-based* super-resolution. Until recently, it was infeasible to learn a good prior on generic natural images; therefore, methods that learn the prior knowledge for image patches only were introduced (e.g., Chang et al. [2004], or ScSR by Yang et al. [2010]). During the training, the mentioned methods learn some representative pairs of LR and HR patches. Later, during the reconstruction, they try to find a mapping between each patch of the LR image and representative LR patches and then use this mapping with the corresponding HR patches to reconstruct the image.

The newest and currently the most successful SISR methods use deep neural networks and end-to-end learning. Many methods have been developed in recent few years with continuously increasing performance (e.g., SRCNN by Dong et al. [2016], VDSR by Kim et al. [2016], SRGAN by Ledig et al. [2016]). The competition on SISR (Timofte et al. [2017]) that took place in 2017 proved that this approach is the current state-of-the-art as the best scoring methods used it. The winner of this competition was method called Enhanced Deep Residual Networks for SISR (EDSR) (Lim et al. [2017]) that is using a network consisting of convolution layers with ReLUs organized in ResNet blocks (He et al. [2016]).

Aside from excelling in generic natural image super-resolution, the example-based methods are the preferable choice also for restricted problems as super-resolution of face images since they can be trained specifically for the restricted problem. Although these methods produce the best reconstruction performance, they can hallucinate the details that were not part of the original scene and yet seem believable to the human, which may not be desirable in some applications.

1.2.2 MAP Approach

The objective of SSIR is to increase the resolution of a single image. In this work, we focus on the most basic yet demanding task – doubling both the horizontal and vertical resolution of an image. In this case, the forward model defined in equation (1.1) uses a degradation matrix A that down-samples the image by a factor of 2. This degradation can be imagined so that the HR image is covered by non-overlapping patches of size 2×2 where each patch determines a pixel of the LR image and the value of this pixel is defined as an arithmetic average of the patch pixels:

$$y_i = \frac{x_{i_{0,0}} + x_{i_{1,0}} + x_{i_{0,1}} + x_{i_{1,1}}}{4} + \eta_i, \tag{1.5}$$

where $\eta_i \sim N(0, \sigma^2)$ is AWGN noise, y_i denotes the i-th pixel of LR image, and $x_{i_{0,0}}$ is the top-left pixel of the i-th patch of HR image (similarly defined for other pixels of the patch). It is also possible to rewrite this forward model using the matrix A, where each row would contain exactly four times value the of 1/4 on the appropriate places and zeros elsewhere; however, the above-mentioned model for single pixel should be more comprehensible.

The likelihood term for this forward model inferred from the equation (1.3) is the following:

$$-\ln\left(p(y|z)\right) = \frac{N}{2}\ln(2\pi\sigma^2) + \frac{1}{2\sigma^2}\sum_{i=1}^{N}\left(y_i - \frac{z_{i_{0,0}} + z_{i_{1,0}} + z_{i_{0,1}} + z_{i_{1,1}}}{4}\right). \tag{1.6}$$

We substitute this term into the MAP equation (1.2), and like in denoising, we omit all the members that do not depend on z. Putting all pieces together, we get the complete MAP solution to twofold single-image super-resolution:

$$\hat{x} = \arg\min_z \frac{1}{2\sigma^2}\sum_{i}^{N}\left(y_i - \frac{z_{i_{0,0}} + z_{i_{1,0}} + z_{i_{0,1}} + z_{i_{1,1}}}{4}\right)^2 - \ln(p(z)). \tag{1.7}$$

We have defined the forward model with the noise η of variance σ^2; however, we usually consider the super-resolution problem to be noise free. One way how to get around this issue is to assume sufficiently low noise variance and ignore the inaccuracy of such model. An analytically correct solution, however, does not assume any noise at all. This can be achieved by setting the variance infinitely close to zero. The limit of the MAP equation for SISR (1.7) as the noise variance approaches zero produces the following constrained minimization problem:

$$\begin{gathered}\hat{x} = \arg\min_z -\ln(p(z)) \\ \text{subject to:} \\ y_i = \frac{z_{i_{0,0}} + z_{i_{1,0}} + z_{i_{0,1}} + z_{i_{1,1}}}{4}.\end{gathered} \tag{1.8}$$

1.3 Optimization

The minimization problems for denoising and SISR with noise that are defined in equations (1.4) and (1.7) can be solved by a simple gradient descent algorithm or one of its variants (e.g., Momentum (Rumelhart et al. [1986]), Adam (Kingma and Ba [2014]), etc.):

$$z = z - \alpha \frac{d}{dz}\left(-\ln p(y|z) - \ln p(z)\right),$$

where α represents the gradient descent step size. However, this approach cannot be applied to the constraint minimization problem for SISR defined in equation (1.8), and some constraint minimization technique has to be applied. For the simplicity, let us rewrite this equation (1.8) into the following form:

$$\hat{x} = \arg\min_z E(z) \quad \text{subject to} \quad Az - y = 0,$$

where $E(z)$ here represents the negative log prior probability and A the downscaling operator.

The traditional approach to solve constraint optimization is to find stationary points of the Lagrange function (Lagrangian):

$$\mathcal{L}_A(z, \lambda) = E(z) - \lambda^T(Az - y),$$

where λ represents a vector of Lagrange multipliers. The gradient descent algorithm cannot be used for finding the stationary points since they occur at saddle points rather than at local extremes. Also, we cannot solve the equation analytically due to the complexity of the prior model. Consequently, we use a gradient descent based method to find the stationary points called Augmented Lagrangian Method (ALM) or Method of Multipliers (Hestenes [1969], Afonso et al. [2011]).

The Augmented Lagrangian (AL) for this problem is defined as:

$$\mathcal{L}_A(z, \lambda, u) = E(z) - \lambda^T(Az - y) + \frac{\mu}{2}||Az - y||_2^2,$$

where the first two terms represent the standard Lagrangian, and the last term serves as a quadratic penalty function with weight μ. ALM is searching for a stationary point z of the Lagrangian by alternating two steps – minimizing the $\mathcal{L}_A(z, \lambda, u)$ with respect to z and updating the Lagrange multipliers λ.

Algorithm 1 Augmented Lagrangian Method

```
1: choose $\mu > 0$ and $\lambda$
2: repeat
3:     $z = \arg\min_z \mathcal{L}_A(z, \lambda, \mu)$
4:     $\lambda = \lambda - \mu(Az - y)$
5: until some stopping criterion is met
```

For the imaging inverse problems as SISR, this algorithm can be simplified using the complete-the-squares procedure:

$$\begin{aligned}
&- \lambda^T(Az - y) + \frac{\mu}{2}||Az - y||_2^2 = \\
&= - \frac{1}{2\mu}||\lambda||_2^2 + \frac{1}{2\mu}||\lambda||_2^2 - \lambda^T(Az - y) + \frac{\mu}{2}||Az - y||_2^2 = \\
&= - \frac{1}{2\mu}||\lambda||_2^2 + \frac{\mu}{2}\left\|Az - y - \frac{1}{\mu}\lambda\right\|_2^2.
\end{aligned}$$

The first term of this equation is a constant independent of z, and therefore it can be omitted in the minimization. Moreover, by substituting

$$d = y + \frac{1}{\mu}\lambda, \tag{1.9}$$

we can simplify the minimization in the following way:

$$z = \arg\min_z E(z) + \frac{\mu}{2} \|Az - d\|_2^2 .$$

Finally, we get rid of λ in the update term by using the substitution (1.9):

$$\begin{aligned} \lambda' &= \lambda - \mu(Az - y) \\ \mu(d' - y) &= \mu(d - y) - \mu(Az - y) \\ d' &= d - (Az - y). \end{aligned}$$

The prime symbol denotes the updated values that will be used in the next step. The complete simplified ALM then looks as follows:

Algorithm 2 Augmented Lagrangian Method II

```
1: choose $\mu > 0$ and $d$
2: repeat
3:     $z = \arg\min_z E(z) + \frac{\mu}{2}||Az - d||_2^2$
4:     $d = d - (Az - y)$
5: until some stopping criterion is met
```

The presented optimization methods for the MAP approach to denoising and two-fold single-image super-resolution that we defined in equations (1.4), (1.7), and (1.8) serve as a theoretical base to the implementation, which is presented in Chapter 3, and provided with this work.

So far we have not talked much about the natural image prior, we only assumed that it is complex enough for any direct solution. In order to use the presented techniques, we have to be able to compute a derivative of the prior with respect to the image. In the context of priors that are based on neural networks and that we use in this work, we need to back-propagate the gradients through the prior model back to the image. Depending on the model complexity, this can be a severe task since the back-propagation has to be made in every minimization step.

Since the a-posteriori term is not convex, the gradient descent may fail to find the global solution if the initial image z is arbitrary. Consequently, we initialize the image z to be close to the desired result. For denoising, we use the noisy image as the initial value, and for SISR we use a simple bicubic or nearest-neighbour interpolation of the LR image. However, we could also apply some existing reconstruction method and then use this MAP approach to improve on it. For constraint optimization it is also necessary to set the initial values of Lagrange multipliers. We initialize them to zero as the first step of ALM algorithm then corresponds to the super-resolution with noise. The initial values are elaborated more in Chapter 4.

2. Image Prior Modelling

Traditionally, image reconstruction methods have used some specific hard-coded prior knowledge about the images to tackle the ill-posed inverse problem (e.g., assumption of smoothness). However, with the growing interest of researchers in machine learning, the hard-coded prior knowledge has started to be replaced by learned prior knowledge. The drawback of this move to learned priors is the loss of interpretability. While the hard-coded priors can have a form of a sentence "Image often consist of repetitive patterns", with learned priors there is no straightforward way to present the learned knowledge in human-readable format. Conversely, they have an opportunity to learn very complex knowledge about the presented images.

The space of all possible images is very large, and the natural images (i.e., images capturing the real-world scenes) cover only a small fraction of this space. For determination whether the image is natural or not serves the image prior model, which assigns each image a probability of being natural.

The first priors used in image reconstruction modelled the presumption that surface of objects is smooth, and that adjacent pixels have similar values. This smoothness assumption was introduced by Tikhonov and Arsenin [1977] in the form of a regularization term that we can rewrite in our notation as:

$$-\ln p(x) = \lambda \int_\Omega |\nabla x|^2,$$

where Ω denotes an image domain, and ∇x is a gradient usually implemented using convolution with a gradient filter. This prior penalizes all gradients, and thus prefers not-noisy images. However, it also penalizes edges that are the essential part of natural images. One of the first attempts to tackle the over-smoothness was proposed by Rudin et al. [1992]. They have substituted the L^2 with L^1 norm. Hence, the resulting prior is called total variation (TV) prior. The best results with the TV prior were achieved if the L^1 norm was encapsulated into some potential function ϕ. In our notation such prior looks as follows:

$$-\ln p(x) = \lambda \int_\Omega \phi(|\nabla x|).$$

Using a proper ϕ function, this prior prefers images formed by smooth regions separated by sharp edges. For more information about the properties of the ϕ function and all the so-far presented priors, we refer to a book by Aubert and Kornprobst [2006].

The idea of using the TV with potential functions was extended by Zhu and Mumford [1997]. They proposed a framework that incorporates multiple different linear filters instead of one gradient filter. These filters are chosen from a bank of linear filters such as Gabor Filters. The particular set of filters from the bank and their corresponding potential functions are learned from data by maximizing the likelihood of the training set.

The next logical step was the usage of arbitrary non-parametric filters which was done in the Field of Experts (FOE) model (Roth and Black [2005]). It assumes each potential to be a parametric t-distribution. The parameters together with the filters themselves are again estimated by maximizing likelihood the of the training data.

Although the so-far presented methods were able to capture some low-level characteristics of the images such as edges and patterns, they were unable to capture the high-level context of the image. That changed with deep architectures (Bengio et al. [2009]) that decompose the problem of image modelling into multiple layers. While the first layer is able to extract some low-level features of the image, the second layer is built on top of those features, and it could capture a slightly higher structure of the image. In this manner, the layers are stacked one by one in order to include higher and higher abstraction of the image. The most promising results regarding natural image modelling offer deep generative models that are subject of the next section.

2.1 Generative Models

In the case of images, standard feed-forward neural networks are usually used for image analysis or for direct image reconstruction. Generative models, on the other hand, are trained for the inverse task to image analysis, image generation. It can be either conditioned on some input (e.g., text description) or unconditioned. Although the concept of generative models is very general, we stick to its application to images in this work.

To generate a natural image, the model has to know what are all the features of these images. In the language of probabilistic theory, it has to know the prior information. Such features are learned from the existing natural images, and they are stored in the parameters of the neural network to be used as a generative model. Since the number of parameters is much smaller than the total number of existing natural images, the network is forced to learn their defining features.

The tricky part of generative models is their learning. Since it is an unsupervised environment, we cannot use the simple end-to-end learning as in the supervised case.

One of the successful approaches is to use Generative Adversarial Network (GAN) (Radford et al. [2015]). It consists of two separate networks: generator and discriminator. While the generator is responsible for generating new images, the discriminator has to recognize if the image was generated, or if it is a real-world image. These two networks are connected so that the gradient can backpropagate from discriminator to generator. The training of GAN consists of two alternating phases. First, the parameters of the generator are fixed and the discriminator, which is a standard binary classifier, is trained to distinguish a natural image from a generated image. Second, the parameters of the discriminator are fixed, and the generator is trained to produce images that are more confusing for the discriminator. These two networks are competing with the ultimate goal of generating images indistinguishable from the real ones. The main problem with GANs is that it is not possible to directly extract prior probability model $p(x)$ and use it in other applications.

Another approach to generative models are Variational Auto-Encoders (VAEs) (Kingma and Welling [2013], Kingma et al. [2016]). Generic auto-encoders also consist of two networks. First, encoder transforms the image to a vector (also called code) in a latent space that is much smaller than the original data space, and second, decoder, which transforms the code from the latent space back to the original data space. Due to the limited capacity of the latent space, the auto-encoder is forced to learn an efficient representation of the data and the corresponding transformations. In the case of Variational Auto-Encoders, the encoder given an image yields parameters of the posterior approximation $q_\phi(z|x)$ (e.g., mean and variance of a multivariate Gaussian), where z represents the latent code. Then z' is sampled from this approximation $z' \sim q_\phi(z|x)$ and passed to the decoder which yields parameters of the conditional distribution $p_\theta(x|z)$. Conversely, the unconditional $p_\theta(x)$ follows the standard normal distribution. The optimization criterion is KL-divergence between $q_\phi(z|x)$ and $p_\theta(z)$ combined with expected conditional probability $p_\theta(x|z)$. Such criterion forces the latent representation z to follow the normal distribution and forces the encoded image to be close to the decoded one. Image generation is here possible by sampling from the distribution $p_\theta(z) \cdot p_\theta(x|z)$. The described learning process is the basic idea behind the simple VAE. Better performance was achieved by Kingma et al. [2016], who proposed a slightly more complicated version of this approach.

The last approach to generative models that we mention and that provides the best results in terms of image likelihood models the prior as a product of parametric conditional distributions over the pixels, where each pixel is conditioned on all the preceding pixels:

$$p_\theta(x) = \prod_{i=1}^{m \cdot n} p_\theta(x_i|x_1, ..., x_{i-1}). \tag{2.1}$$

The term $m \cdot n$ denotes the size of an image and θ stands for the distribution parameters that are represented by a neural network. Learning is then achieved using maximum likelihood principle on true natural images, and generation of a new image is here done using sampling from the prior pixel by pixel, row by row. The conditioning is illustrated in Figure 2.1.

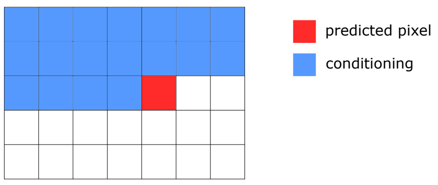

Figure 2.1: Illustration of a factorized conditional distribution for a pixel.

The parameters θ of the conditional distributions $p_\theta(x_i|x_{\leq i})$ are shared for all pixels of an image. Therefore, it is relevant to think of recurrent neural networks (RNNs), which are designed for processing of data streams. In our case, the data stream consists of image pixels and the weights in RNN cells represent the distribution parameters. The idea of employing RNNs for this task was adopted by Theis and Bethge [2015] in Recurrent Image Density Estimator (RIDE) for grey-scale images. It uses spatial (two-dimensional) Long Short-Term Memory (LSTM) (Graves et al. [2007]) to create a latent vector for each pixel. This vector then serves as a set of parameters to a certain Gaussian-based mixture distribution. The image is processed by 2D LSTM from left to right from top to bottom, and by definition, it captures the long-range dependencies in both horizontal and vertical direction. RIDE network is trained by a simple stochastic gradient descent algorithm on the negative log-likelihood (NLL).

This work was later extended by van den Oord et al. [2016b] in many ways. While RIDE models the pixels as continuous values, they have proposed to model them as discrete values using a multinomial distribution, without any assumption on its shape. Moreover, they have extended the method to colour images, and they maintain the dependencies between individual RGB colour channels:

$$p(\mathbf{x}_i|\mathbf{x}_{<i}) = p(x_{i,R}|\mathbf{x}_{<i})p(x_{i,G}|\mathbf{x}_{<i}, x_{i,R})p(x_{i,B}|\mathbf{x}_{<i}, x_{i,R}, x_{i,G}). \tag{2.2}$$

In this equation $\mathbf{x}_i$ represents the whole pixel and $x_{i,R}$, $x_{i,G}$, and $x_{i,B}$ stands for the corresponding colour channels of this pixel.

Regarding RNN, van den Oord et al. [2016b] propose network architecture called PixelRNN. It consists of multiple residual blocks each containing one LSTM layer. The LSTM cells of these layers use masked convolutions in their state-to-state and input-to-state operations. The masking ensures that only the valid context for each pixel is taken into account. This design beat all preceding approaches in terms of likelihood on the CIFAR-10 dataset (Krizhevsky and Hinton [2009]).

Besides PixelRNN, van den Oord et al. [2016b] also proposed solely convolutional network called PixelCNN. The complex LSTM layers are substituted by masked convolutional layers that offer much faster learning process since the image can be processed in parallel. The drawback of this approach is the fact that potentially unbounded receptive field (context of each conditional distribution) in the case of LSTM is here limited to large but bounded receptive field. Consequently, the PixelRNN architecture performed better in terms of likelihood; however, the PixelCNN architecture offers much faster learning, and the networks derived from it now offers the best likelihood scores. This is the reason why we have chosen this type of architecture for our prior model in this work. In the next section, we present this type of network and its derivations in detail.

2.2 PixelCNN

A generative neural network architecture that is based solely on convolutional layers and that models the image prior as a product of conditional probabilities over the pixels as it is described in Equation (2.1) was first introduced by van den Oord et al. [2016b]. The main idea is to capture the context of each conditional distribution using two-dimensional convolutions. To ensure that only the valid context is taken into account, the convolutions are masked so that for prediction of a pixel only the preceding pixels can be read. An example of such mask is depicted in Figure 2.2. The size of the context that the network is able to capture (i.e., the receptive field) depends on the sizes of convolutional kernels and the number of convolutional layers. The growth of the receptive field with the depth of the network is illustrated in Figure 2.3.

The original PixelCNN method is composed of multiple layers where each layer, except the first and the last one, consists of one masked 3×3 convolution with Rectifier Linear Unit (ReLU). It uses two types of masks: *mask A* and *mask*

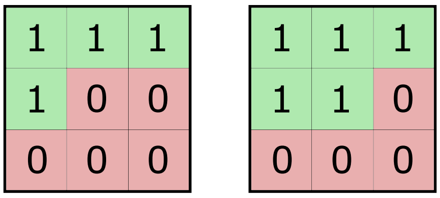

Figure 2.2: Example masks for 2D convolution with kernel size 3×3. **Left:** *Mask A* for the first convolutional layer. **Right:** *Mask B* for all subsequent convolutional layers.

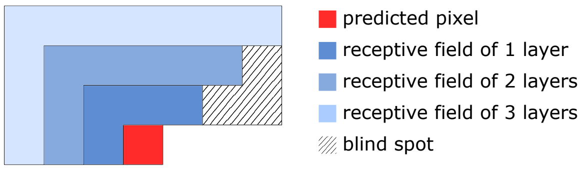
predicted pixel
receptive field of 1 layer
receptive field of 2 layers
receptive field of 3 layers
blind spot

Figure 2.3: The Growth of receptive field with the depth of PixelCNN assuming masked convolutional layers with kernel size 3×3

B. The *Mask A* is applied to the first layer to ensure that the convolution reads only pixels preceding the predicted pixel. *Mask B* is a relaxed version of *mask A* that, in addition, allows connection from the predicted pixel to itself, and it is used in all subsequent layers. In order to improve the learning process, the layers are connected using residual connections (He et al. [2016]). The last layer is a simple 256-way softmax.

Just like for PixelRNN, the conditioning is not done on whole pixels but rather on their individual colour channels as it was presented in Equation (2.2). Therefore, the *mask A* is slightly more complicated than it was depicted as it also allows reading of all preceding channels of the same pixel. In the end, each colour channel is predicted separately using the softmax layer. For more detailed description, we refer to the original work of van den Oord et al. [2016b].

2.2.1 Gated PixelCNN

One of the problems of the original PixelCNN architecture was the blind spot in the receptive field illustrated in Figure 2.3. This problem was addressed by van den Oord et al. [2016a] who introduced an architecture called Gated Pixel-CNN. Besides the blind spot removal, they have proposed gated convolutional layers allowing more complex interactions between layers.

Gated PixelCNN resolved the problem with the blind spot by combining two stacks of convolutional layers: vertical and horizontal. While the vertical stack has a receptive field consisting of rows above the current pixel, the receptive field of the horizontal stack consists of pixels left of the current one that lie on the same row. This division is illustrated in Figure 2.4. Each stack contains several convolutional layers with appropriate masking and the outputs of the stacks after each layer are combined so that a layer in the horizontal stack takes as an input the output of the previous layer in this stack as well as the output of the corresponding layer from the vertical stack. Making the connection between the stacks one-way ensures that only the pixels left or above the current pixel are used as a context, which is required by the conditional distribution.

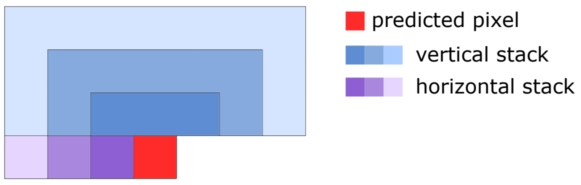

Figure 2.4: The growth of receptive field with the depth of Gated PixelCNN assuming masked convolutions of size 3×3 for vertical stack and 1×3 for horizontal stack assuming that the stacks are not connected (which is not true in the network).

Another contribution of Gated PixelCNN is the usage of gated activation units that replace the traditional ReLUs:

$$y = \tanh(W_{k,f} * x) \odot \sigma(W_{k,g} * x),$$

where $W_{k,f}$ and $W_{k,g}$ represent the convolutional kernels of the k-th layer, σ is a sigmoid non-linearity, $*$ denotes a convolution operator, and $\odot$ is an element-wise product. Although there are two convolutions, they can be implemented using a single convolution where the output features are split into σ and tanh non-linearities.

Similar to original PixelCNN, residual connections between layers have been added to the network but only to the horizontal stack, as adding it also to the vertical stack did not appear to be beneficial in any way. A single layer of the described architecture is illustrated in Figure 2.5.

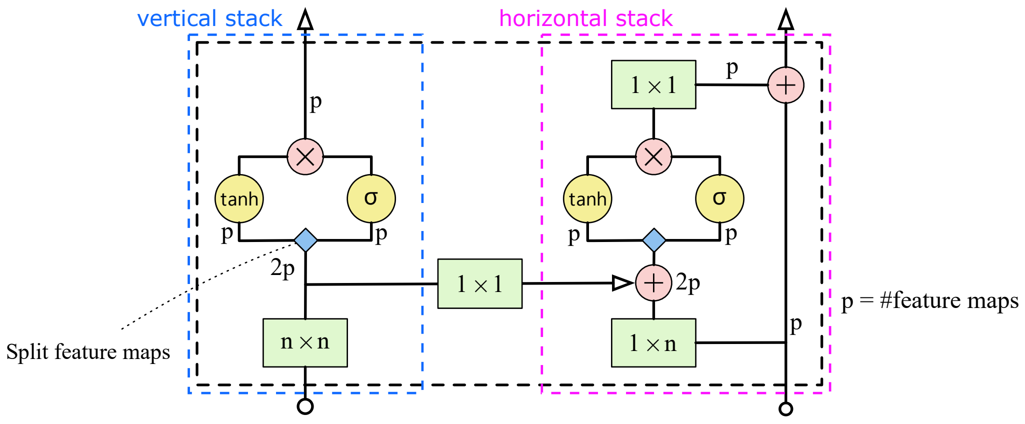

Figure 2.5: Single layer in the Gated PixelCNN architecture. Convolutions are shown in green, element-wise operations in red, and non-linearities in yellow. Based on a diagram by van den Oord et al. [2016a]

.

Gated version of PixelCNN has achieved the likelihood scores of PixelRNN while preserving the relatively fast learning time of PixelCNN. Moreover, this network has served as a basis to an even better model called PixelCNN++ (Salimans et al. [2017]) that seems to be the most advanced architecture for image prior modelling (see Table 2.1). Therefore, we use it in our experiments, and we devote it the next subsection.

2.2.2 PixelCNN++

The already outstanding performance of Gated PixelCNN was improved by Salimans et al. [2017] in an architecture called PixelCNN++. It came up with four principal modifications to the former architecture. First, it replaces the 256-way softmax with discretized logistic mixture likelihood. Second, it conditions on the whole pixels rather than on individual colour channels. Third, to capture the structure at different resolutions, it uses downsampling and upsampling with short-cut connections. Finally, it employs dropout (Srivastava et al. [2014]) during learning in order to avoid overfitting on the training data.

The problem with modelling the pixel values using 256-way softmax is, among others, its inability to capture that the two adjacent values (e.g., 127 and 128) are similar and such relationship has to be learned in the preceding layers. Therefore, like in VAEs, the PixelCNN++ network has replaced the softmax with a simple parametric continuous distribution of the latent colour intensity ν where the parameters of the distribution are produced by the network. The authors have chosen the mixture of logistic distributions:

$$\nu \sim \sum_{i=1}^{K} \pi_i \text{logistic}(\mu_i, s_i).$$

Such distribution allows to easily compute the probability of the discrete pixel value x given the distribution parameters π, μ and s:

$$p(x|\pi, \mu, s) = \sum_{i=1}^{K} \pi_i \left[\sigma\left(\frac{x + \frac{1}{2} - \mu_i}{s_i} \right) - \sigma\left(\frac{x - \frac{1}{2} - \mu_i}{s_i} \right) \right], \tag{2.3}$$

where σ is the cumulative distribution function (CDF) of the logistic distribution (i.e., sigmoid function). This equation is valid except the edge case of 0 and 255. The probability for these cases is naturally defined using CDF as:

$$\begin{aligned} p(x = 0|\pi, \mu, s) &= \sum_{i=1}^{K} \pi_i \left[\sigma\left(\frac{x + \frac{1}{2} - \mu_i}{s_i} \right) \right], \\ p(x = 255|\pi, \mu, s) &= \sum_{i=1}^{K} \pi_i \left[1 - \sigma\left(\frac{x - \frac{1}{2} - \mu_i}{s_i} \right) \right]. \end{aligned} \tag{2.4}$$

The original PixelCNN factorized the prior distribution over the colour channels as it is shown in Equation (2.2). Such factorization allows very complex relations between the individual channels but it also unnecessarily complicates the model since the relations between the colour channels of a single pixel are likely to be simple, and they do not need a deep network for modelling them. Therefore, PixelCNN++ conditions on whole pixels and outputs the parameters of the joint distribution of all colour channels. The joint distribution is a simple factorized model of logistic distributions over the colour channels. Interactions between the channels are possible in a linear way so that the mean of each logistic distribution linearly depends on the values of the preceding colour channels. This is clarified in the following equation:

$$p(x|\pi,\mu,s,\alpha,\beta,\gamma) = \sum_{i=1}^{K} \pi_i \cdot p(x_r|\mu_{i,r}, s_{i,r}) \cdot p(x_g|\mu'_{i,g}, s_{i,g}) \cdot p(x_b|\mu'_{i,b}, s_{i,b}),$$
$$\mu'_{i,g} = \mu_{i,g} + \alpha x_r, \qquad \mu'_{i,b} = \mu_{i,b} + \beta x_r + \gamma x_g, \tag{2.5}$$

where μ, s, α, β, and γ are parameters of the joint distribution that are predicted by the network and x_r is the red colour channel of pixel x (green and blue channels are denoted in the same manner).

Another problem of the original PixelCNN is a relatively small receptive field, which is large enough to capture the local dependencies, but it is not able to capture the high-level context of the image. To model the image structure on different resolutions, PixelCNN++ uses several downsampling layers implemented using convolutions of stride 2 in both horizontal and vertical direction, and the corresponding upsampling layers implemented using transposed convolutions of stride 2. Such architecture resembles an auto-encoder, where the layers between the input and the smallest resolution form an encoder part, and layers between the smallest resolution and the output form a decoder part. However, such architecture would lead to a significant loss of information; therefore, between layers of the same resolution short-cut connections are added. Each layer of the encoder is connected using a short-cut to the corresponding layer in the decoder. The short-cuts have a form of 1×1 convolutions.

Like Gated PixelCNN, PixelCNN++ uses residual blocks with gated convolutions with two connected stacks: horizontal and vertical (illustrated in Figure 2.5). The complete architecture is visualized in Figure 2.6.

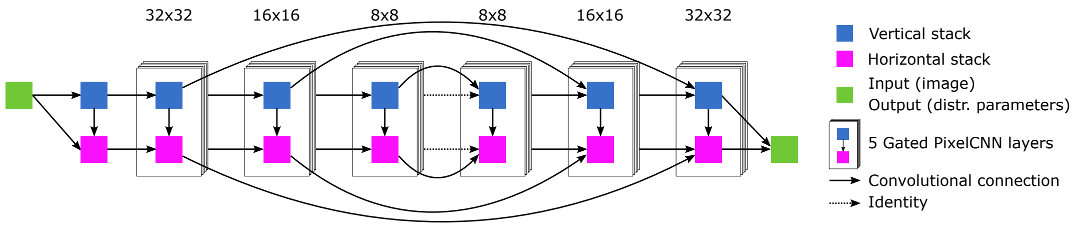

Figure 2.6: Architecture of PixelCNN++ for the CIFAR-10 dataset.

The comparison of the presented generative models that can be used for estimating the natural image prior is shown in Table 2.1. The values represent the likelihood, which is expressed in bits per dimension (bpd), of the CIFAR-10 test set. The bpd for an image measures the minimal amount of information, additional to the prior, that would be necessary for lossless compression of the image. Therefore, the lower the number, the better the prior represents the image. For CIFAR-10 test images, it is clear that PixelCNN++ architecture performs the best, and therefore, we use it for the proposed image reconstruction method.

Model	Bits per dimension
RIDE Theis and Bethge [2015]	3.47
PixelCNN van den Oord et al. [2016b]	3.14
VAE with IAF Kingma et al. [2016]	3.11
Gated PixelCNN van den Oord et al. [2016a]	3.03
PixelRNN van den Oord et al. [2016b]	3.00
PixelCNN++ Salimans et al. [2017]	2.92

Table 2.1: Comparison of the presented generative image prior models in terms of likelihood (expressed in bits per dimension) on the CIFAR-10 dataset. Source: Salimans et al. [2017] and van den Oord et al. [2016b].

3. Implementation

Implementing a simple neural network trained using the back-propagation algorithm is a straightforward task; however, for deep networks, it is more convenient to use an existing neural network framework such as TensorFlow (Abadi et al. [2016]), Caffe (Jia et al. [2014]), Microsoft Cognitive Toolkit (Microsoft Corporation [2017]), Theano (Theano Development Team [2016]), etc.

Since the implementation of the prior model that we use in our work, PixelCNN++, is maintained in TensorFlow and since there exists a pre-trained model for it, we have chosen TensorFlow also for the implementation of the methods proposed in this work.

3.1 TensorFlow

TensorFlow is an open source framework for numerical dataflow computation developed by Google Brain team. The computation is described using directed graphs, where the nodes define mathematical operations, and the edges represent multidimensional arrays called *tensors*, through which the operations exchange data. The framework allows deploying the computation not only to CPU but also to GPU, where the execution time of operations on large tensors is significantly reduced. It is most commonly used with neural networks; however, it is general enough to be used for wide variety of other cases.

The TensorFlow framework is exposed to programmer through API in several languages, such as C++, Java, or Python. We use the latter one since it is comfortable to use, the PixelCNN++ was developed in it, and it is at present the most complete. The API range from very low-level functions as a simple element-wise addition of two input tensors to high-level functions as multidimensional convolutional neural network layers.

The program in this framework is divided into two main parts. First, building the computational graph and second, running the computational graph. In the building part, the programmer usually defines the shape and type of the input tensors and the set of dependent low or high-level operations. To run the graph, one has to create a *Session* that encapsulates the control and state of the TensorFlow runtime. The primary operation of the session is *Run*, which given a set of output tensor names evaluates in the correct order all operations on which they depend and returns the values of the output tensors. A frequently used but optional parameter for the *Run* method is the data that should be fed into the particular places in the graph. This option is used primarily to provide values to some input tensors.

This framework was designed for the situation where the graph is created once and then executed many times. Sometimes it is therefore convenient (e.g., in machine learning) to save the values of some tensors in the graph for the next run without explicitly returning them. For this purpose, there exists a special operation called *Variable* that returns a tensor that survives the execution of the graph. These tensors can be updated during the execution, which is extensively used in training of neural networks or generally in any optimization process.

TensorFlow was mainly designed for deep neural networks; therefore, it contains many functions that make the building of the network (i.e., graph) easier. It contains functions for creating individual layers, such as fully connected, or convolutional. These layers usually include weight or kernel tensors that are automatically created as trainable *Variables*. The graph for training neural networks must also include all operations necessary for computing the gradients and updating the *Variables*. For that purpose, the API contains *optimizers* that given an optimization criterion, a set of trainable *Variables*, and some other arguments create the necessary nodes and edges of the graph and an operation that when evaluated by *Run* method updates the *Variables* in the desired manner. A typical graph for neural network consists of a single input tensor, single output tensor, and several layers in between. Also, it usually contains target tensor used for supervised training. Such graph is then executed in two modes: Either training, where the evaluation of an optimizer operation is required and input, and target tensors are fed with the desired data, or inference, where the value of the output tensor is required, and only the input tensor is fed with the data.

TensorFlow can also save and restore the state of the network (e.g. values of *Variables*). This is necessary not only for training deep architectures, which can take hours or days, but also for the deployment of the trained network.

3.2 Implementation

We have developed a prototype implementation for testing of the proposed approach to image reconstruction tasks. Our code can be partitioned into several segments – prior model definition, data loading, building a computational graph, running of the graph, and evaluation. In this section, we show several simplified snippets of the code. For more detailed description, we refer to the actual commented code provided with this work.

3.2.1 Prior Model Definition

For incorporating of the prior model to our code we have developed an interface containing a constructor, a method to get prior in the form of negative log probability of the provided image and a method to load the pre-trained prior model parameters from the specified file. This architecture allows us to experiment with different image priors on different datasets. In the case of PixelCNN++ model we have encapsulated the code provided by the authors of the model into this interface; however, our code should work with any prior model.

The PixelCNN++ code itself consists of two parts – a model that given an image predicts the parameters of discretized logistic distribution for each pixel, and a logistic distribution that given the parameters and the image computes the joint probability of such image.

The model follows the architecture depicted in Figure 2.6. It uses 192 feature maps for all residual connections. The masked 3×3 convolutions in the vertical stream are implemented as 3×2 convolutions on an image that are shifted down by one row. The masked 1×3 convolutions in the horizontal stream could be also implemented as 1×2 convolutions on an image shifted to the right by one row; however, the authors of the code found beneficial to use 2×2 convolutions on an image that is shifted in right-down direction by one row and one column.

It should be reminded that the horizontal stack conditions also on pixels above since the vertical stack is connected to it. This shifting is used for all layers in ResNet blocks and downsampling and upsampling layers in between. The first layer of the vertical stream moreover shifts the output down by one row, so that the subsequent layers read only the valid context. The first layer of the horizontal stream is implemented similarly. It combines a convolution of size 3×1 that is shifted down and a convolution of size 1×2 output of which is shifted to the right.

The model predicts the parameters for the discretized mixture of logistics as well as the parameters of the linear dependency between the colour channels. In the default setting, the output tensor contains 100 values for each pixel. The first 10 values represent the log probability of each of the 10 mixture components. The next 3×10 values predict the mean for each colour channel and each mixture component. Another 3×10 values represent the scale parameter of the distributions corresponding to each mean value. The remaining 3×10 values describe the linear dependency between the means of the logistic distributions of individual colour channels as it was presented in equation (2.2).

The code provided by the authors of PixelCNN++ has a very complex data dependent initialization of variables; however, since we load the values of these variables from a file, we slightly changed the code to omit this unnecessary initialization.

3.2.2 Data Loading

The source code of PixelCNN++ contained data loaders for the CIFAR-10 dataset (Krizhevsky and Hinton [2009]) and for the ImageNet dataset (Russakovsky et al. [2015]) downsampled to size 32×32. Based on these data loaders, we have developed loaders also for the ImageNet downsampled to size 64×64 and the BSDS-300 (Martin et al. [2001]) datasets, and a loader for a single image. The more detailed description of these datasets is provided in Section 4.1.

The architecture is designed to reconstruct multiple images of the same size at once; therefore, the tensors containing images are four-dimensional (batch size $\times$ height $\times$ width $\times$ number of channels). Because of this design, our code can process only images of the same size on a single run. Consequently, we split the BSDS dataset that contains images of size 481×321 or 321×481 into two datasets, horizontal and vertical. Moreover, since the PixelCNN++ model as it was written by its authors is unable to process images that are not divisible by 4, we crop the images of the BSDS dataset by one pixel both horizontally and vertically.

3.2.3 Building the Computational Graph

Each of the reconstruction tasks we deal with requires slightly different computational graph; however, the overall architecture remains very similar. First, we define the placeholders for degraded and initial images and optimization parameters (e.g., noise variance, initial values for Lagrange multipliers, etc.). In order to avoid feeding these placeholders in each optimization step, we store them as *Variables*, and we feed them only once at the beginning. Except for the optimized image, the variables do not change during the optimization; therefore, they are created as non-trainable.

```
#Placeholders
t_sigma2_ph = tf.placeholder(tf.float32, name="sigma2_ph", shape=[])
t_y_ph = tf.placeholder(tf.float32, name="degraded_ph", shape=...)
t_z_ph = tf.placeholder(tf.float32, name="initial_ph", shape=...)
#Variables
t_sigma2 = tf.get_variable("sigma2", initializer=t_sigma2_ph, trainable=False)
t_y = tf.get_variable("degraded", initializer=t_y_ph, trainable=False, ...)
t_z = tf.get_variable("estimate", initializer=t_z_ph, trainable=True, ...)
```

For the noiseless super-resolution, we do need neither `t_sigma_ph` nor `t_sigma`, but we have to define the placeholder and *Variable* for the μ parameter and d variable of the ALM algorithm presented in Section 1.3.

```
t_d_ph = tf.placeholder(tf.float32, name="lagrange_ph", shape=...)
t_mu_ph = tf.placeholder(tf.float32, name="mu_ph", shape=...)
t_d = tf.get_variable("lagrange", initializer=t_d_ph, trainable=True, ...)
t_mu = tf.get_variable("mu", initializer=t_mu_ph, trainable=False, ...)
```

Second, we construct the sub-graph for the prior model using the aforementioned interface. The output of this sub-graph is the negative logarithm of the prior probability of the optimized image:

```
#Prior model
model = PixelCNNpp(nr_resnet=5, nr_filters=160, nr_logistics=10, ...)
t_prior = model.get_prior(t_z)
```

The essential part of the graph regarding the image reconstruction is MAP minimization term. For denoising task, this term is derived from equation (1.4):

```
#Loss for denoising
t_sse_loss = tf.reduce_sum((t_y - t_z)**2, reduction_indices=(1,2,3))
t_loss = t_sse_loss + 2 * (t_sigma2**2) * t_prior
```

The resulting graph is visualized in Figure 3.1. For super-resolution with noise, we derive the graph from equation (1.7):

```
#Loss for two fold SISR with noise
t_avg=(t_z[:,::2,::2,:] +t_z[:,1::2,1::2,:] + \
       t_z[:,1::2,::2,:]+t_z[:,::2,1::2,:])/4
t_see_loss = tf.reduce_sum((t_y - t_avg)**2, reduction_indices=(1,2,3))
t_loss = t_sse_loss + 2 * (t_sigma2**2) * t_prior
```

Finally, for constraint minimization that is used for super-resolution without noise, we infer the graph from equation (1.8) and ALM algorithm:

```
#Graph for SISR without noise
t_avg=(t_z[:,::2,::2,:] +t_z[:,1::2,1::2,:] + \
       t_z[:,1::2,::2,:]+t_z[:,::2,1::2,:])/4
t_see_loss = tf.reduce_sum((t_d - t_avg)**2, reduction_indices=(1,2,3))
t_loss = (t_mu/2) * t_sse_loss + tf_prior
```

Last but not least, we define an optimizer that will be used for minimization. We have experimented with various optimizers, and it seems that the best results provide simple gradient descent with exponential learning rate decay.

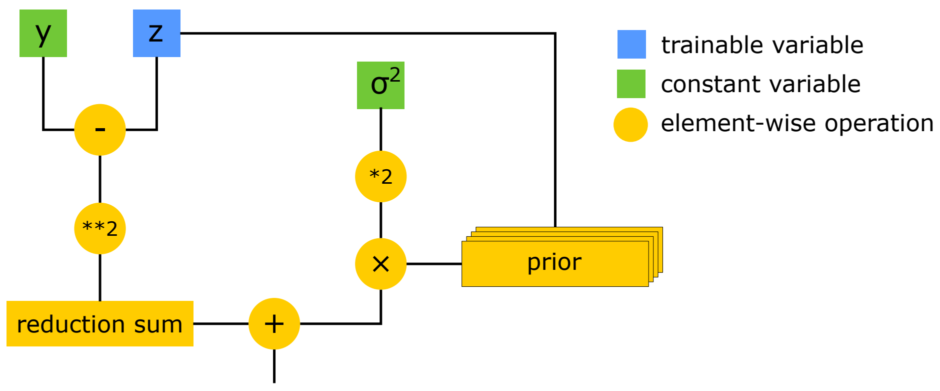

Figure 3.1: Illustration of computational graph for denoising

```
#Gradient descent on the image
t_lr = tf.train.exponential_decay(learning_rate=..., decay_rate=..., ... )
optimizer = tf.train.GradientDescentOptimizer(t_lr)
t_minimize = optimizer.minimize(t_loss, var_list=[t_z], ...)
```

The evaluation of the `t_minimize` tensor by *Run* method makes one step of the optimizer algorithm. For the constraint optimization using the ALM algorithm, we also need an operation to update the d variable.

```
#Update the "Lagrange multipliers"
t_update_d = t_d.assign(t_d - (t_avg - t_y))
```

3.2.4 Running the Computational Graph

For the evaluation purposes, we use the described graph in the following way. First, a *Session* is created, and the parameters of the prior model are loaded. Then, each batch of images from a dataset is degraded using the forward model of the reconstruction task. From the degraded images, depending on the particular task, the initial estimates of the reconstructed images are created. For denoising, the initial value is the degraded image itself, for super-resolution, the bicubic or zero order interpolation is used. All these values together with the noise variance in the case of noisy reconstruction or with the μ parameter and initial values for d variable in the case of constraint optimization are fed into the graph.

After the initialization, the optimization process is started. For the standard gradient descent, it looks as follows:

```
with tf.Session(...) as sess: #Create session
    model.restore(sess, ...)  #Load pre-trained prior
    for x in test_data:  #Loap over images of dataset
        y = forward_model(x, sigma) #Corrupt the images
        z = get_initial_value(y) #Get initial image to be optimzied
        #Initialize the TensorFlow Variables with data
        initializer = tf.variables_initializer([t_y,t_z,...])
        sess.run(initializer, feed_dict={t_y_ph:y,t_z_ph:z, ...})
        #Optimize using pre-defined number of steps
        for epoch in range(nr_steps):
            sess.run(t_minimize)
        reconstructed = sess.run(t_z)  #Return reconstructed images
```

For the constraint minimization using the Augmented Lagrangian Method, where the minimization alternates with the update of d variable. In this case, the last three lines of the above code snippet are substituted by the following lines:

```
for i in range(nr_iterations):
    for epoch in range(nr_steps[i]): #Minimization
        sess.run(t_minimize)
    sess.run(t_update_d) #Update of "Lagrange multipliers"
reconstructed = sess.run(t_z)  #Return reconstructed images
```

The best performance is achieved when the number of steps of the optimization and eventually the number of iterations is pre-defined. The stopping criterion is elaborated in Chapter 4. At the end of the optimization process, the reconstructed images are gathered from the graph.

The PixelCNN++ prior model was defined for images where the pixel values are scaled from the range $[0; 255]$ into the range $[-1; 1]$. To avoid unnecessary conversions in the graph, and to be consistent throughout the computational graph, we use this range in our method as well. Consequently, it is necessary to rescale also the noise variance appropriately: $\sigma' = \sigma \cdot 127^{-2}$

These were very simplified code snippets. Our implementation is also able to log the optimization process and evaluate the reconstruction performance in every step, which is necessary for tuning the hyper-parameters such as the number of optimization steps, learning rate, decay rate, or used optimizer. The optimal values of these parameters along with their dependencies on datasets are elaborated in the next chapter.

4. Experiments

We have proposed methods for image reconstruction based on MAP approach that exploits the currently most advanced natural image prior models. We have developed a prototype implementation, the details of which are described in Chapter 3. In this chapter, we explain some design choices we made and evaluate our solution in terms of reconstruction performance and run-time. Also, we compare the results with the current state-of-the-art methods for the corresponding reconstruction tasks.

4.1 Data Sets

For proper training and evaluation of the proposed methods, it is necessary to get a dataset of images that are close to what we would consider the ideal natural images. They should not be distorted, blurry, corrupted by any noise, they should not contain any artefacts, and so forth. Although there is a tremendous amount of images available for usage, plenty of them do not satisfy the constraints of an ideal image. Fortunately, most datasets that were initially created for image analysis fulfil those conditions, and we can employ them in the reconstructions tasks; although, no exact measure of ideality exists.

One of the widely used datasets for image analysis that can be employed in image reconstruction is the CIFAR-10 dataset (Krizhevsky and Hinton [2009]). It consists of 60000 colour images of size 32×32. These images were captured in much higher resolution; therefore, the presence of noise and other inevitable degradations in the captured images is negligible after downsampling. Each image is labelled by one of 10 classes (e.g., bird, ship, or deer) according to the main object of the image. For reconstruction, these labels are not interesting, but if this dataset is used for training of the natural image prior, it might be slightly biased as it could learn that on each image, there has to an object belonging to one of these classes. The images are divided into a training set containing 50,000 images and a test set containing 10,000 images. We use the train set for training the prior model and test set for the evaluation the reconstruction capabilities.

More suitable for the natural image prior modelling seems to be the ImageNet dataset (Russakovsky et al. [2015]). The images of this dataset are also labelled but there are more than 80,000 classes, and therefore it should not be a limitation. We use this dataset in two downsampled versions that were created for training of PixelRNN (van den Oord et al. [2016b]) prior. One contains images of size 32×32, and we denote it as ImgNet-32. The other one consists of images of size 64×64, and we denote it as ImgNet-64. Again, the downsampling makes the degradations that occurred during capturing process negligible. The dataset is also divided into a training and validation set. The first of them contains over million of images, and we use it for image prior training. The latter one consists of almost 50,000 images, but due to the high computational demands of the proposed method, we use only a subset of it to evaluate the reconstruction.

Although these datasets contain a huge number of images, their resolution is small. Therefore, we also employ Berkeley Segmentation Data Set 300 (BSDS300) (Martin et al. [2001]) that consist of slightly larger colour images. There are

200 training images and 100 test images of size 481×321 or 321×481 in the dataset. Since the PixelCNN++ implementation we use is not able to handle image dimensions that are not divisible by 4, we crop them by one row and one column. This dataset contains too few images for learning a good prior model; therefore, we use it only for the reconstruction assessment. The images in this dataset are not that ideal as in the previous ones, but we consider them to be good enough.

We had also considered using DIV2K dataset, which was created for the NTIRE2017 (Agustsson and Timofte [2017]) competition on single-image super-resolution. It consists of 1000 images that were manually chosen with special attention to image quality and dataset diversity. Each image has 2K resolution, meaning that it is over 2000 pixels high or wide. The dataset is divided into a training, validation, and test set so that the diversity of each set remains high enough. Unfortunately, the size of the images is not only too large for training of the neural network that we use as a prior model but also too large for the reconstruction using the proposed methods as well. We list this dataset here mainly because it might be interesting for the future work.

4.2 Prior Model Training

One of the advantages of the proposed reconstruction methods is the fact that the image prior model can be learned separately regardless the actual reconstruction task. In this work, we use the PixelCNN++ model that was briefly described in Section 2.2.2. It is based on a neural network consisting of masked convolutional layers, and by design, it is scalable to images of arbitrary sizes. However, for performance reasons, the TensorFlow implementation provided by the authors of the model requires being trained on images of the same size as the size has to be known during the construction of the computational graph.

Although the PixelCNN++ model can be potentially used on images of arbitrary size, the architecture was optimized for the CIFAR-10 dataset that contains images of size 32×32 only. The encoding part of the model uses 3 ResNet blocks containing 5 residual layers with 192 feature maps that are connected by 2×2 downsampling convolutions. The same architecture uses the decoding part, except it uses upsampling convolutions between the blocks. This architecture is illustrated in Figure 2.6. For the training, the authors use Adam optimizer (Kingma and Ba [2014]), learning rate decay, and a dropout rate of 0.5 for each residual layer.

In our experiments, we use the PixelCNN++ prior trained on the CIFAR-10, ImgNet-32, and ImgNet-64 datasets. The first trained model is provided directly by the authors of PixelCNN++. It was trained for 5 days on 8 NVIDIA TITAN X GPUs and reached 2.92 bits per dimension (bpd) on the CIFAR-10 test set. For training of the prior on the ImgNet-32 dataset, we use exactly the same architecture. Unfortunately, we do not have such computational power available as the best hardware we could use was a single NVIDIA P100 GPU. We had trained the model for 8 epochs, which took about five days, and reached 4.02 bpd on the test set. The last prior model that we use was trained on the ImgNet-64 dataset. It had been trained for 5 epochs, which also took about 5 days, and it reached 3.69 bpd on the test set. The comparison to other trained prior models

Dataset	PixelCNN++	GatedPixelCNN	PixelRNN
CIFAR-10	2.92	3.00	3.03
ImgNet-32	4.02*	3.83	3.86
ImgNet-64	3.69*	3.57	3.63

Table 4.1: Test performance of different models on different datasets. Values marked with * are suboptimal, because of the lack of computation power. Other values are provided by the authors of the respective models.

is shown in Table 4.1. Compared to GatedPixelCNN, for example, our models reach much lower bpd. This is, however, caused by the fact that the authors had been training the model for 2.5 days on 32 GPUs, while we had to fit into 5 days on one GPU only.

Because we use the model for MAP optimization, naturally, we were interested in how the model behaves and what images it prefers. To address this question, we have tested the model, which was trained on the CIFAR-10 dataset, on the 5 images of the test set. We used the same framework as we use for reconstruction, but we optimized only the prior term. The results after 4000 steps of gradient descent, with an initial step size of $5 \cdot 10^{-6}$ and step size decay of 0.9999, are shown in Figure 4.1. The size of a gradient descent step was chosen so that it would not add any noteworthy noise and thus change the image significantly. The optimization process is depicted in Figure 4.2.

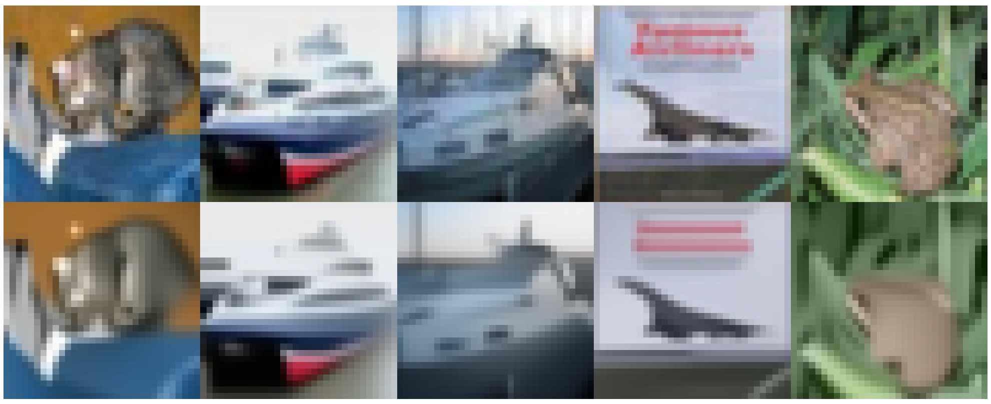

Figure 4.1: **Upper**: 5 original images from the CIFAR-10 test set. **Bottom**: More probable versions of the images according to the PixelCNN++ model.

The images show that the model prefers smoother images, but it well preserves smooth edges. During the reconstruction, this over-smoothing will be reduced by the likelihood term. Other options include decay of the gradient step size or early stopping.

Since we were not able to learn the optimal parameters of the prior model on all datasets, we have evaluated the trained prior models across the different datasets. The results presented in Table 4.2 show that the lowest score is achieved if the training set and the test set come from the same dataset. However, later we will demonstrate that the model with the lowest score on some dataset need not lead to the best MAP reconstruction on that dataset.

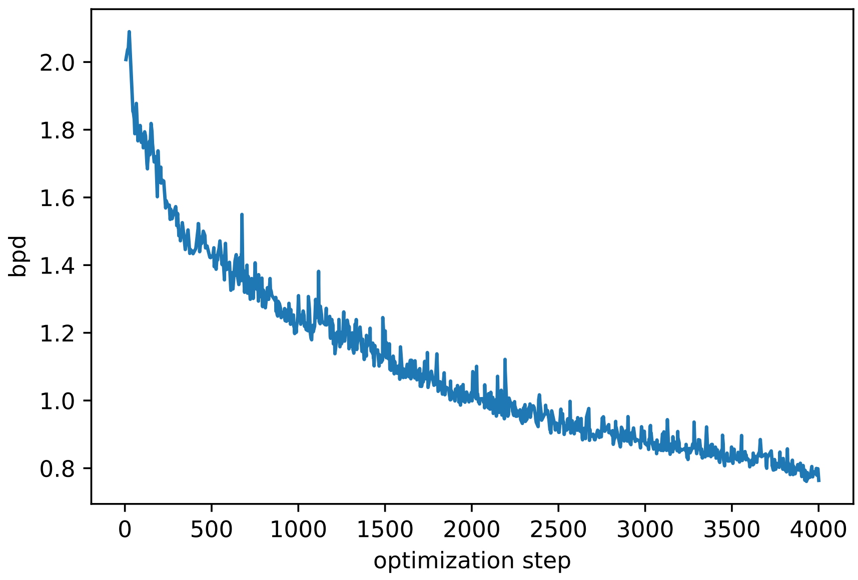

Figure 4.2: Minimalization of negative log prior probability (expressed as bpd) of the 5 CIFAR-10 images using PixelCNN++ model. Value at step 0 represents the average bpd over the original images.

Test \ Training set	CIFAR-10	ImgNet-32	ImgNet-64
CIFAR-10	**2.92**	3.25	3.19
ImgNet-32	4.27	**4.02**	4.03
ImgNet-64	3.93	3.71	**3.69**
BSDS-300	2.79	2.52	**2.43**

Table 4.2: Test performance (bpd) of PixelCNN++ model trained and tested on different datasets. Each row represent one test set and each column one training set.

4.3 Methodology

In order to evaluate the proposed against the selected state-of-the-art methods in terms of reconstruction performance, we first degrade the original images using the assumed forward model. Then we measure how well each reconstruction method restores the image. Although the individual methods may work with various pixel scales, all degradations and measurements are made on images with pixels in standard 8-bit format. We use two measures – peak signal-to-noise ratio (PSNR) and structural similarity index (SSIM) (Wang et al. [2004]).

The first one, PSNR, measures the ratio between the maximal possible power of the signal (i.e., original image) and the power of the noise (i.e., the difference between original and reconstructed image) and it is expressed in decibels. The formula for computing PSNR on images where the value of the pixel is in the range $[0;255]$ is following:

$$\mathrm{PSNR}(x,\hat{x}) = 10\log_{10}\left(\frac{255^2}{\frac{1}{N}\sum\limits_{i=1}^{N}(x_i-\hat{x}_i)^2}\right),$$

where x represents the original and $\hat{x}$ the reconstructed image. Generally, the higher the value, the more similar the images are.

The drawback of measuring the reconstruction error using PSNR is that it is not consistent with the impact of the error on human perception. It may happen that the change in two degraded images with the same PSNR would be perceived entirely differently. This imbalance is partially addressed by the second measure we use, SSIM (Wang et al. [2004]). It estimates the structural similarity of two images as a correlation between two normalized images. This measure also takes into account that perception of a luminance change is lower with growing luminance and the similar relation is also applied to contrast. The formula we use is following:

$$\mathrm{SSIM}(x,\hat{x}) = \frac{(2\mu_x\mu_{\hat{x}} + 2.55^2)(2\sigma_{x\hat{x}} + 7.65^2)}{(\mu_x^2 + \mu_{\hat{x}}^2 + 2.55^2)(\sigma_x^2 + \sigma_{\hat{x}}^2 + 7.65^2)},$$

where μ_x, σ_x^2, $\mu_{\hat{x}}$, and $\sigma_{\hat{x}}^2$ represents the local mean and variance of the original image x and the reconstructed image $\hat{x}$. Similarly, $\sigma_{x\hat{x}}$ denotes the local covariance of x and $\hat{x}$. The overall image SSIM is measured as a mean of SSIM of all local Gaussian windows with a standard deviation of 1.5. The SSIM ranges from -1 to 1, and the higher is the value, the more similar the images are perceived. Conversely to the PSNR, we measure the SSIM on the luminance channel only. We obtain the luminance channel using the algorithm presented in the following subsection.

4.3.1 Comparison to Greyscale Methods

To compare the colour reconstruction methods with single-channel methods, we convert either the degraded or the reconstructed image to grey-scale using the linear combination of colour channels:

$$x_y = \alpha x_r + \beta x_g + \gamma x_b,$$

where x_r denotes the red colour channel of pixel x (similarly defined for green and blue channels). We use the following standard constants: $\alpha = 0.2989$, $\beta = 0.5870$, and $\gamma = 0.1140$. In the case of colour methods, we take the colour degraded image, reconstruct it and then convert it to grey-scale. In the case of single-channel methods, we first convert the degraded image into grey-scale, and then we reconstruct it. Afterwards, we measure PSNR and SSIM on the reconstructed grey-scale images. It must be acknowledged, however, that the comparison favours the colour reconstruction methods as they have more information about the image available for reconstruction.

The conversion to grey-scale also affects the parameters of the grey-scale forward model of the reconstruction task. One of these parameters is the noise variance $var(n)$. Since we assume uncorrelated noise in each channel, the formula for computing the noise variance for the grey-scaled image is following:

$$var(\alpha n_r + \beta n_g + \gamma n_b) = \alpha^2 \cdot var(n_r) + \beta^2 \cdot var(n_g) + \gamma^2 \cdot var(n_b),$$

where n_r represents the noise added to the red channel (green and blue channels are defined similarly). Since we assume the colour forward model where all channels are corrupted with the noise of the same variance σ^2, the variance of the grey-scale image is:

$$\sigma_y^2 = (\alpha^2 + \beta^2 + \gamma^2)\sigma^2.$$

With the linear combination we use, the standard deviation of the greyscaled image is: $\sigma_y = 0.6685\sigma$.

4.3.2 Execution Time Comparison

Not only the reconstruction performance is an important measure but also the execution time of the methods. Some applications may prefer fast processing at the cost of lower reconstruction performance and vice versa. The proposed methods are mainly focused on the quality of reconstruction; therefore, we have not elaborated any detailed statistics regarding the execution. The provided results should serve for a rough comparison only.

We have measured the execution time for reconstruction of a single image including any initialization the method needs (e.g., building the computational graph or loading the pre-trained model). In the case of methods that are implemented in MATLAB, the measured time does not include the time necessary to start the MATLAB software. All methods were measured in the same environment – a server computer with 2 Intel Xeon E5-2643 v4 CPUs with a base frequency of 3.40GHz, 377GB RAM, single NVIDIA Tesla K40m graphics card, Windows 10 operating system, and MATLAB 2017b software (if needed). During the testing, we have tried to minimize the impact of other user and system processes.

4.4 Denoising

One of the reconstruction problems on which we have evaluated the proposed methods is denoising. Based on the experiments with prior models (presented in Section 4.2), we assume that the highest reconstruction performance should be achieved on the CIFAR-10 test set using the PixelCNN++ prior trained on the CIFAR-10 training set. First, we have searched for the best optimization parameters for a small subset of the test set. Then, we used these parameters to evaluate the reconstruction performance against the state-of-the-art methods on the entire test set. Finally, we evaluate the performance on other datasets as well.

In our case, the parameters that are affecting the optimization process (also called hyper-parameters) are – the stopping criterion, the employed optimizer algorithm, and the parameters of such algorithm as gradient step size, decay rates, and so forth. It should be stated, that these hyper-parameters are often bound to the implementation we presented in Section 3.2 since they depend on the scale of the optimized term. We have tried to find the best hyper-parameters on a subset of 20 images from the CIFAR-10 test set. We consider this subset size sufficient since the differences in the optimization process of individual images are not that large. We have tested several different optimizer algorithms – simple gradient descent, momentum, and Adam. Out of this set, the most promising results gave simple gradient descent with a step size of 0.0018.

Although the likelihood term of the MAP equation reduces the over-smoothing tendency of the used prior model, it does not avoid it completely. This leads to highly peaked reconstruction performance with respect to the number of gradient steps and poor performance if the reconstruction is not stopped after a certain number of iterations. To tackle this issue, we found beneficial to use exponential decay of the gradient descent step size with a decay rate of 0.9965 in each step. Not only it slows down the drop in the performance after the peak is reached, but also it leads to actually lower reconstruction error. The difference in optimization process with and without the decay is visualized in Figure 4.3. From there it follows that the decay positively affects the optimization process, but the problem is not solved completely. The performance still slowly deteriorates after reaching the maximum. Another option how to attenuate this property is to lower the weight of the prior model by altering the σ in the equation (1.4). Nevertheless, the experiments have shown that this altering reduces the deterioration but at the cost of high computational demands and with no improvement in the maximum reconstruction performance. Consequently, we use early stopping that halts the optimization after a certain number of steps that we found to be optimal.

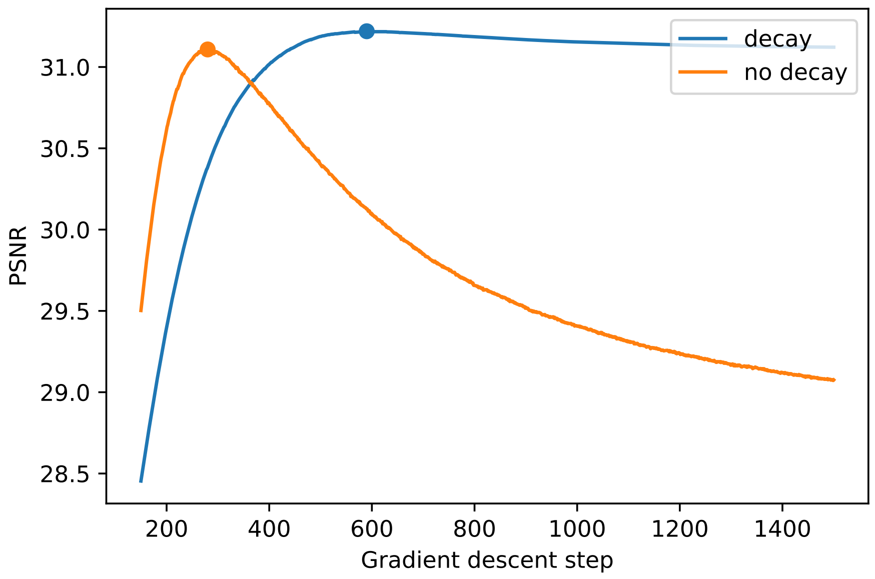

Figure 4.3: Influence of step size decay on optimization process for denoising of images corrupted by noise with standard deviation $\sigma = 15$.

We have discovered that the optimal hyper-parameters vary with the noise variance. If we use the same setting for noise with $\sigma = 50$ as we used for noise where $\sigma = 15$, we reach the best value much sooner; therefore, the early stopping with the same number of steps would not lead to the optimal solution. To solve this issue, we have tried to alter the step size, decay rate and weight of the prior term; however, the best results we obtained when we stopped the optimization early enough. Therefore, we use the same hyper-parameters for arbitrary noise variance except the number of gradient steps, which we alter accordingly. To find a dependency between the noise variance and the optimal number of gradient steps (also called epochs in this context), we have concluded several experiments on a batch of 40 images from the CIFAR-10 test set. The results are visualized in Figure 4.4. It shows that there is a difference between the average best epoch over

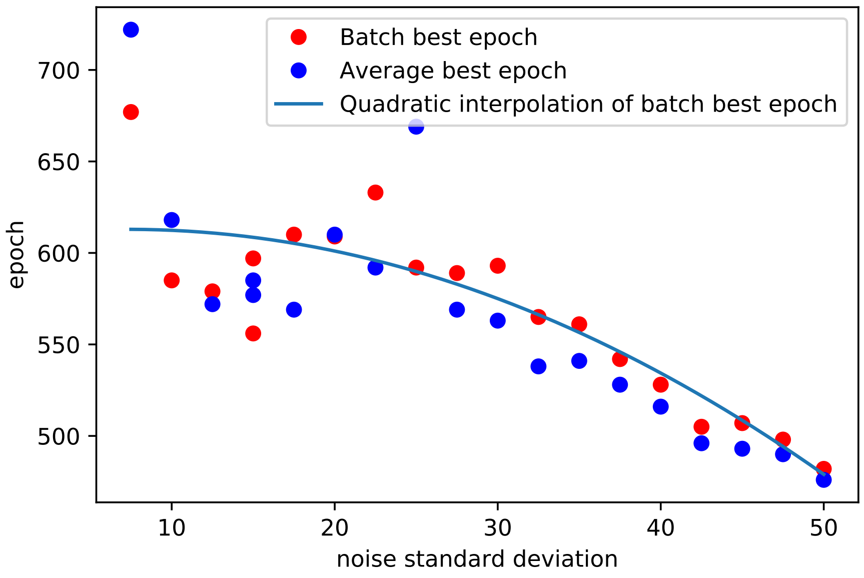

Figure 4.4: Influence of the noise variance on the optimization process of a batch of 40 images from the CIFAR-10 test set.

the individual images and the epoch that leads to the best average reconstruction performance. As a stopping criterion, we naturally use the latter one. The measured values suggest the following quadratic dependency:

$$\text{opt. number of steps} = -0.0734 * \sigma^2 + 1.07 * \sigma + 609$$

While this curve, which was inferred from the measured data using the method of least squares, explains the dependency for the variance in range $\sigma \in [10, 50]$, it does not work for extreme cases. However, for extremely high noise, the reconstruction is by principle impossible, and for the extremely low noise, it is pointless since the degradation is negligible.

Finally, we have measured the denoising performance using the above mentioned hyper-parameters on the entire CIFAR-10 test dataset. The results compared to the selected state-of-the-art colour denoising methods, which were mentioned in Section 1.1.1, are shown in Table 4.3. The proposed method is denoted as *VIRGNN*, which stands for Variational Image Restoration using Generative Neural Networks. Other compared methods are CBM3D by Dabov et al. [2007b], IRCNN by Zhang et al. [2017b], and DnCNN Zhang et al. [2017a]. For the last two methods, we use the already trained model provided by the authors of these

σ	Noisy	VIRGNN	CBM3D	IRCNN	DnCNN
15	24.79	31.53	30.96	**31.82**	31.13
25	20.49	28.25	27.92	**28.79**	28.47
50	14.94	23.83	23.66	**24.53**	24.51

Table 4.3: Comparison of PSNR of colour denoising methods on the CIFAR-10 test dataset

methods. However, since these models were trained on larger images than we use, the comparison may be slightly biased. Unfortunately, we do not have hardware capacities to neither train nor test the proposed method on larger images. The results show that the proposed approach has its place among the state-of-the-art denoising methods, although it has slightly lower reconstruction performance compared to the IRCNN and DnCNN. Due to the small resolution of images of the CIFAR-10 datasets, the methods based on non-local similarities as CBM3D performs worse as they use relatively large patches and do not have enough context available. There is a possibility that if the compared methods were optimized for this dataset, they would provide even better results. Examples of restoration capabilities of VIRGNN are depicted in Figure 4.5.

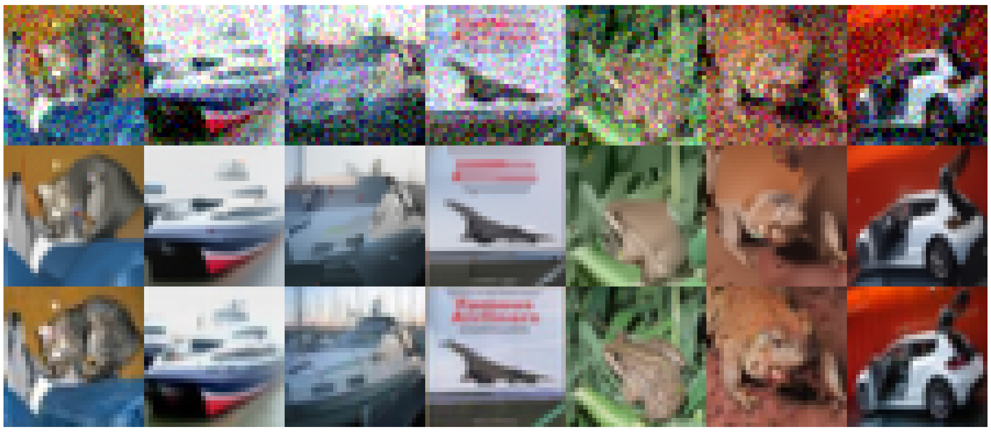

Figure 4.5: Denoising of the CIFAR-10 images degraded by noise with standard deviation $\sigma = 25$ using the proposed method.

The extended comparison that also includes the greyscale methods and that evaluates the reconstruction of the luminance channel is presented in Table 4.4 and 4.5. We compare the proposed VIRGNN method to BM3D by Dabov et al. [2007a], EPLL by Zoran and Weiss [2011], and WNNM by Gu et al. [2014]. The greyscale methods are, however, disadvantaged since they can use only a fraction of information about the degraded images.

Regarding the execution time, out of the compared methods, the presented solution performs the worst. The disadvantage of the proposed method is the need of backpropagation through the prior model in each step of the optimization. This puts high demands not only on the processing time but also on the memory as the output of each layer has to be stored during the forward pass in order to speed up the otherwise even slower backpropagation. This property together with the hardware and time limitations made it impossible to reconstruct

σ	Noisy	VIRGNN*	CBM3D*	IRCNN*	DnCNN*	BM3D	EPLL	WNNM
15	28.23	32.67	32.15	**32.92**	32.69	31.51	31.67	31.78
25	23.92	29.21	28.94	**29.72**	29.64	28.34	28.63	28.53
50	18.32	24.64	24.38	25.28	**25.34**	24.11	24.48	23.96

Table 4.4: Comparison of PSNR of greyscale and colour (marked with *) denoising methods on the CIFAR-10 test dataset. The displayed noise variance σ is related to the colour images. PSNR values are calculated on luminance channel of the images.

σ	Noisy	VIRGNN*	CBM3D*	IRCNN*	DnCNN*	BM3D	EPLL	WNNM
15	0.846	0.950	0.945	**0.954**	0.952	0.937	0.942	0.941
25	0.717	0.901	0.898	**0.916**	0.913	0.886	0.897	0.890
50	0.489	0.777	0.773	0.811	**0.815**	0.760	0.780	0.754

Table 4.5: Comparison of SSIM of greyscale and colour (marked with *) denoising methods on the CIFAR-10 test dataset. The displayed noise variance σ is related to the colour images.

large images (e.g., DIV2K dataset), or large datasets (e.g., the entire ImageNet dataset). Conversely, the reconstruction methods that are based on feed-forward neural networks as IRCNN or DnCNN use only single pass through the network, and therefore, their computational and memory demands are negligible compared to the proposed solution. To put this into perspective, Table 4.6 shows the comparison of the execution time of the all tested methods.

We have also evaluated the proposed approach with different priors on CIFAR-10, ImgNet-32, and ImgNet-64 datasets. The BSDS-300 dataset was excluded from this experiment as the model for a single image from this set could not fit the GPU memory, and optimization on CPU is too slow for conducting any meaningful experiments. We have measured the best possible performance on a subset of 20 images with the presented hyper-parameters ignoring the stopping criterion. The results are presented in Table 4.7. Even though the best likelihood score on prior was always achieved using the pair of test and training set of the same dataset, in the case of denoising much better results, especially for higher noise variance, are obtained using the prior that was trained on the CIFAR-10 dataset. Hence, we use this prior for all tested datasets. Here we have to remain, that this could have been different if we would have trained the other two prior models until the convergence. We have also tried to alter the hyper-parameters for each dataset and each prior; nevertheless, no significant increase in performance was achieved compared to the presented hyper-parameters. We have found out,

VIRGNN*	CBM3D*	IRCNN*	DnCNN*	BM3D	EPLL	WNNM
167s	0.03s	3.42s	3.26s	0.03s	0.88s	1.08s

Table 4.6: Execution time of denoising methods on a single image from CIFAR-10 test dataset with noise variance $\sigma = 25$. (Colour denoising methods are marked with *)

test \ train	σ	Noisy	CIFAR-10	ImgNet-32	ImgNet-64
CIFAR-10	15	24.77	**31.18**	30.60	30.83
CIFAR-10	25	20.49	**27.84**	27.25	27.46
CIFAR-10	50	14.97	**23.32**	22.81	22.92
ImgNet-32	15	24.84	29.68	29.74	**29.83**
ImgNet-32	25	20.57	**26.47**	26.4	26.46
ImgNet-32	50	15.05	**22.18**	21.94	22.02
ImgNet-64	15	24.87	30.64	30.64	**30.76**
ImgNet-64	25	20.61	**27.51**	27.40	27.50
ImgNet-64	50	15.09	**23.24**	22.93	23.05

Table 4.7: Comparison of maximal denoising performance of the proposed approach with different priors on several datasets. The values represent average PSNR values over the highest PSNR values. Each row stands for a subset of 20 images from a test set that were corrupted by the noise of the corresponding variance σ^2. Last three columns represent the individual prior models.

that the optimal stopping criterion (i.e., number of gradient descent steps) vary for the individual datasets. Nevertheless, the biggest difference between using the optimal stopping criterion for the dataset and the presented stopping criterion we found was in the magnitude of $1 \cdot 10^{-2}$ PSNR. Consequently, we use the same hyper-parameters and same stopping criterion for all tested datasets.

With the CIFAR-10 prior and the presented hyper-parameters, we have evaluated the proposed method on the ImageNet and BSDS datasets. Unfortunately, due to the limited hardware capabilities, we were not able to evaluate the method on entire datasets, but only on a fraction of each of them. For ImageNet it was 1000 images and for BSDS only 9 images. For BSDS the model was forced to work on CPU only as it could not fit the GPU memory. Just to give a perspective, the optimization of 9 BSDS images on CPU took about three days using the aforementioned hardware. The results may be seen in Table 4.8 and 4.9. The differences between the methods on the ImageNet datasets roughly correspond to the CIFAR-10 results; although, the PSNR scores of all methods are slightly lower due to the more complex images.

The results on a fraction of the BSDS-300 dataset suggest that the prior we use does not scale well with the increasing image size. While the edges are preserved well in the reconstructed images, on smooth surfaces, we can observe a vast number of significant artefacts. This is especially visible for higher noise variance, which is illustrated in Figure 4.6. This property could be probably solved by using a better prior trained on larger images that contain much larger flat areas and that uses a larger receptive field.

4.5 Super-Resolution

The second reconstruction problem we are dealing with in this work is two-fold Single-Image Super-Resolution (SISR). The forward model we assume is a simple two-fold downsampling as in equation (1.5) but without any noise. We evaluate both the proposed approaches to this problem – reconstruction using uncon-

dataset	σ	Noisy	VIRGNN	CBM3D	IRCNN	DnCNN
ImgNet-32	15	24.91	29.82	29.31	**29.86**	28.33
ImgNet-32	25	20.62	26.59	26.10	**26.70**	26.04
ImgNet-32	50	15.08	22.27	21.67	22.51	**22.53**
ImgNet-64	15	24.92	30.69	30.42	**30.85**	29.50
ImgNet-64	25	20.65	27.52	27.29	**27.77**	27.19
ImgNet-64	50	15.12	23.17	22.84	23.55	**23.59**
BSDS-300	15	24.72	32.37	32.67	**32.98**	32.96
BSDS-300	25	20.42	29.37	29.82	30.22	**30.29**
BSDS-300	50	14.93	25.20	25.65	26.13	**26.36**

Table 4.8: Comparison of PSNR of colour denoising methods on first 1000 images from the ImageNet test sets and first 9 vertical images of BSDS-300 test set.

dataset	σ	Noisy	VIRGNN	CBM3D	IRCNN	DnCNN
ImgNet-32	15	0.882	0.948	0.942	**0.949**	0.941
ImgNet-32	25	0.775	0.895	0.886	**0.901**	0.895
ImgNet-32	50	0.564	0.753	0.724	0.771	**0.780**
ImgNet-64	15	0.828	0.936	0.934	**0.940**	0.933
ImgNet-64	25	0.699	0.878	0.878	**0.890**	0.885
ImgNet-64	50	0.477	0.737	0.725	0.763	**0.775**
BSDS-300	15	0.740	0.916	0.920	**0.928**	**0.928**
BSDS-300	25	0.573	0.847	0.861	0.877	**0.878**
BSDS-300	50	0.342	0.706	0.728	0.755	**0.772**

Table 4.9: Comparison of SSIM of colour denoising methods on first 1000 images from the ImageNet test sets and first 9 vertical images of BSDS-300 test set.

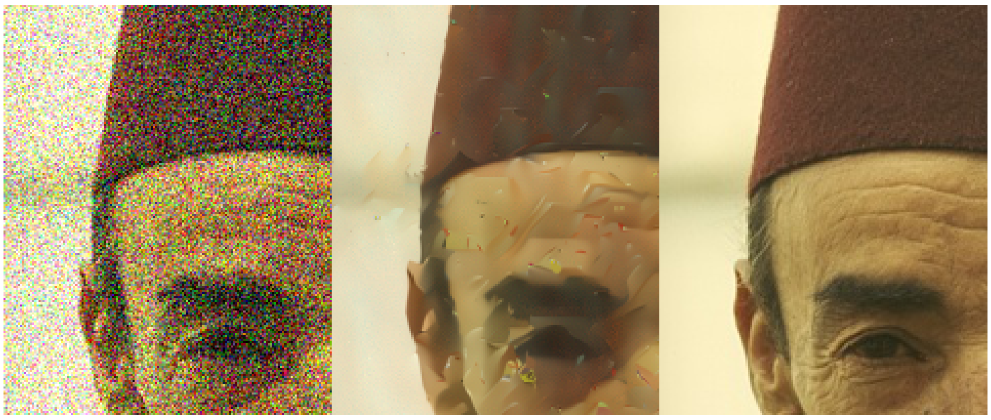
Figure 4.6: A cutout of an image from a BSDS-300 dataset degraded by noise with standard deviation $\sigma = 50$ and reconstructed using the proposed method. **Left:** degraded, **middle:** reconstructed, **right:** original.

strained optimization assuming that there is a noise of very low variance, and reconstruction using constraint optimization that correctly does not assume any noise at all.

The first problem regarding the optimization process of SISR is the initialization. While for denoising, the choice of the initial value was apparent, the degraded image, for SISR the choice is not that clear. We cannot use the same initialization as for the denoising problem since the degraded image has a different resolution than the optimized image. One of the straightforward approaches we have tried is a bicubic interpolation. Using this baseline SR method for initialization led to the best reconstruction results at the end regardless of using constrained or unconstrained optimization. Another approach is to use the same principle as in denoising and choose initial value so that the likelihood term (defined in equation (1.6)) is zero. This can be achieved for example by interpolation of zero order, which in our case splits each pixel of the degraded image into four pixels of the same value. Experiments we concluded showed that this initialization leads to smoother optimization, but slower convergence. Both of the aforementioned approaches produce differently over-smoothed initial images. This smoothness can be reduced by introducing noise to the initial image and thus encouraging the optimizer to generate some details in the reconstructed image. Nevertheless, we have found out that this noise rather decreases the overall performance. Last initialization we have tried uses an entirely random image (where each pixel is sampled independently from the uniform distribution). However, as we stated before, the prior term is not convex, and thus any gradient descent fails in this case completely.

Like in denoising, on a subset of 20 images from the CIFAR-10 dataset, we have tried to find the best hyper-parameters for both constrained and unconstrained approaches. We have experimented with various optimizers – gradient descent, momentum, and Adam, their parameters (e.g., step size, step size decay, momentum, etc.), stopping criteria, and noise variance in case of unconstrained optimization. Again, we have to remind, that the hyper-parameters are bound to the implementation proposed in Chapter 3 since they depend on the scaling of the minimized term.

Regarding the unconstrained optimization, the best performance was achieved using assumed noise variance $\sigma^2 = 1$, simple gradient descent optimizer with step size 0.1, and no step size decay. The step size has to be set relatively high in order to significantly change the pixel values and thus discover new and more probable images. Setting the step size low as in the case of denoising would stuck the optimization in local optima close to the initial image. The reconstruction performance does not deteriorate with increasing number of gradient descent steps. Nevertheless, we stop the optimization after $9,000$ steps since on average after this number it stops improving.

For the constrained optimization, we use the Augmented Lagrangian Method (ALM) presented in Section 1.3, where the goal is to find the minimum of the Augmented Lagrangian (AL) in every iteration of the method. Therefore, not only we need a stopping criterion for the minimization of AL in each iteration but also for the whole ALM. We have tried to create robust stopping criterion based on the loss; however, like in denoising the best performance we achieved with fixed optimization process, where the number of epochs in each iteration is pre-defined. For ALM it is also necessary to set the initial values of Lagrange multipliers. We found optimal to initiate them with zero. Speaking in the language of the ALM algorithm 2, we initialize d variable with the degraded image. The first iteration, therefore, corresponds to the unconstrained minimization where $\sigma^2 = \mu^{-1}$. The subsequent iterations then improve on the first one.

We have experimented with various settings of hyper-parameters, many of them led to the similar performance; nevertheless, the best results were achieved having the first iteration same as in the unconstrained case. After proper rescaling, which is necessary since we work with pixels in the range $[-1; 1]$, the gradient step size is set to $0.1/127^2$ and parameter μ to 127^2. In each subsequent iteration, we lower the step size by a decay factor of 0.7. With this setting, the number of steps was empirically chosen as $9,000$ for the first iteration and 200 for every subsequent iteration. The number of iterations we use is 6 since after that the optimization process stagnates. One may also alter the μ value in each iteration, but we have not found it beneficial in any way for this problem.

The progression of the reconstruction performance with increasing number of steps for both constrained and unconstrained optimization is depicted in Figure 4.7. From there it follows that the constrained optimization is more powerful than the unconstrained one; hence all the following experiments are made using this method. The example of the performance of the proposed super-resolution method is depicted in Figure 4.8.

We have tried to find the best settings of the hyper-parameters systematically; however, since the parameters are not independent on each other and the space of all possible values is relatively large, there may exist some better setting than the presented one.

We have evaluated the proposed constrained minimization approach denoted as *VIRGNN* and compared it against the selected state-of-the-art methods – ScSR by Yang et al. [2010], VDSR by Kim et al. [2016], EDSR+ by Lim et al. [2017], and SRCNN by Dong et al. [2016]. The results are presented in Table 4.10 and 4.11. Unfortunately, the provided implementation of the SRCNN method works with grey-scale images only; therefore, we alter it so that the luminance channel is reconstructed using the method and chromatic channels are reconstructed using

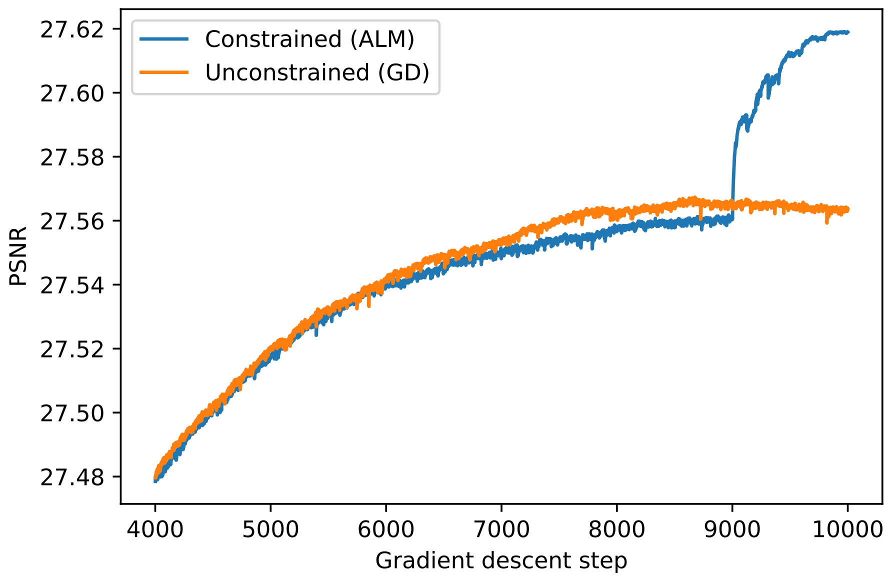

Figure 4.7: Optimization process of two-fold super-resolution using constrained and unconstrained optimization

bicubic interpolation. The same principle is employed by ScSR or VDSR. Like in denoising, we use the already trained models provided by the authors of the methods. These models were trained on larger images, and therefore, the comparison might be slightly biased. However, since the training is usually performed on patches of size around 40 pixels, the bias should not be large. We also tried to train our own VDSR model on the CIFAR-10 dataset. It achieved slightly better PSNR scores on all three tested datasets, but the increase did not exceed a value of 0.4. In terms of SSIM, the results remained the same.

The results show that the presented approach significantly exceeds all the tested methods and can be considered the current state-of-the-art in terms of reconstruction performance. However, it should be stated that the results might have been different if the methods had been trained on the same dataset as the

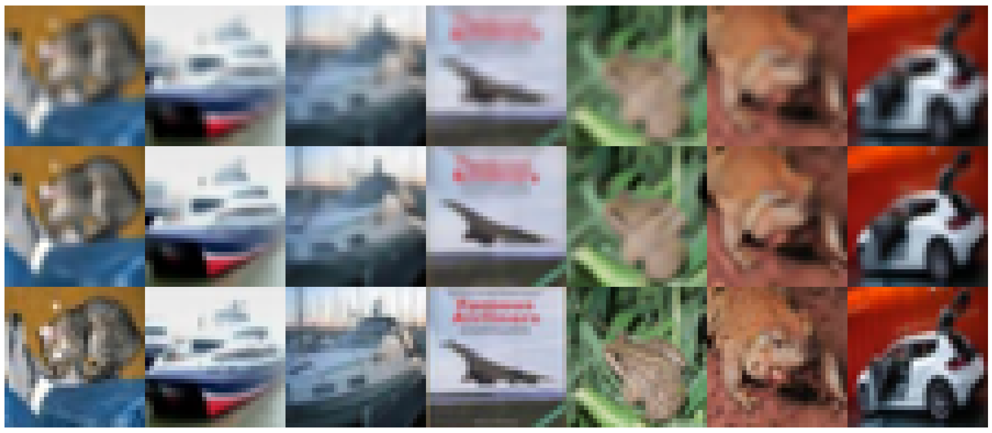

Figure 4.8: 2x super-resolution of images from CIFAR-10 dataset using VIRGNN. **Upper**: bicubic interpolation. **Middle**: VIRGNN. **Bottom**: original.

dataset	Bicubic	VIRGNN	ScSR	VDSR	SRCNN	EDSR+
CIFAR-10	26.36	**28.41**	26.94	27.27	27.09	27.52
ImgNet-32	22.16	**23.27**	22.39	22.49	22.44	22.56
ImgNet-64	23.68	**24.95**	24.06	24.19	24.13	24.15

Table 4.10: Comparison of colour single-image super-resolution methods in terms of PSNR on different datasets. Only first 500 images from CIFAR-10 and ImgNet-32 and 100 images from ImgNet-64 are evaluated.

dataset	Bicubic	VIRGNN	ScSR	VDSR	SRCNN	EDSR+
CIFAR-10	0.871	**0.913**	0.892	0.902	0.894	0.895
ImgNet-32	0.758	**0.813**	0.790	0.800	0.792	0.809
ImgNet-64	0.783	**0.833**	0.814	0.826	0.817	0.829

Table 4.11: Comparison of colour single-image super-resolution methods in terms of SSIM on different dataset. Only first 500 images from CIFAR-10 and ImgNet-32 and 100 images from ImgNet-64 are evaluated.

proposed method and tuned for processing of small images. Since the number of gradient steps is more than 15 times higher compared to the denoising, the execution time corresponds to this fact. While the VDSR method process one image in about 3 seconds, our method needs about 40 minutes for the same task. Consequently, we were not even able to measure its reconstruction abilities on the entire CIFAR-10 and ImageNet datasets. Moreover, we removed the evaluation on BSDS-300 dataset as the processing time for a single image on the available hardware would take about one week.

Conclusion

We have proposed a maximum-a-posteriori probability (MAP) approach to image reconstruction that employs the currently most successful natural image prior models based on generative neural networks. We focused on two reconstruction tasks – image denoising and two-fold single-image super-resolution; however, the proposed approach can be easily adapted to a vast variety of other inverse problems.

Throughout this work, we used PixelCNN++ generative architecture that models the image prior as a product of conditional distributions over pixels. The conditioning is modelled using convolutions that are masked, stacked and connected so that the receptive field for the single pixel is large and valid. The design of the proposed method, however, makes it possible to replace this prior model with any other model that allows backpropagation. The prototype implementation for these two tasks is provided along with this work.

Due to our limited computational capabilities, we used the PixelCNN++ model trained on the CIFAR-10 dataset. Using this model, we have achieved denoising performance approaching the current state-of-the-art on the CIFAR-10, ImageNet-32, and ImageNet-64 datasets. Regarding super-resolution, the proposed solution beat all the tested state-of-the-art methods. The already excellent results on ImageNet datasets might be even better if we would use a good model trained on these datasets.

From the practical point of view, the usability of the proposed denoising method remains very limited. The existing solutions providing similar or slightly better results runs about 50 times faster than the proposed one. For super-resolution, the difference in execution time is even bigger as it runs more than 15 times longer than denoising. However, since for super-resolution the proposed method gives the best results, it might be worth to wait for it.

The main drawback of the proposed MAP approach to image reconstruction is the need for backpropagation through the complex prior model in each gradient descent step. This puts high demands not only to the computational time but also to the memory. Moreover, we use rather a simple TensorFlow implementation of the prior model that process the entire image at once. Such implementation limits the usage of the method to small images only. More robust implementation should consider the fact that receptive field is limited, and therefore, the image areas can be processed serially with lower memory demands.

Future Work

The primary objective of this work was to evaluate the usage of the generative models of natural images. The reconstruction abilities of the proposed approach show that it is worth studying. The drawbacks of the methods then point out specific problems that should be solved.

It remains unclear how the PixelCNN++ prior behaves on larger images. For evaluation of that, it will be necessary to develop an implementation that can handle arbitrarily large images, which is possible due to the restricted receptive field of the PixelCNN networks.

Due to the complexity of the reconstruction problem, it might also be interesting to evaluate the approach proposed by Zoran and Weiss [2011] where the prior is not computed on the entire image but rather on the individual image patches. In this way, it would be possible to process much larger images even with the existing simple implementation of the PixelCNN++ model.

While the reconstruction process alone is computationally demanding, especially for super-resolution, learning a good prior model is even more demanding. Throughout this work, we mostly used the prior trained by the authors of PixelCNN++. We tried to train the prior on ImageNet datasets; however, it was infeasible with the hardware capacities we had. Moreover, for ImageNet, van den Oord et al. [2016a] propose to use a slightly more complex model which would make the training even harder. In the future, some effort should be made to evaluate the reconstruction possibilities using priors trained on more natural-looking dataset than CIFAR-10.

Recently, Kolesnikov and Lampert [2017] proposed two novel PixelCNN architectures called *Pyramid PixelCNN* and *Grayscale PixelCNN*. Both of them address the tendency of PixelCNN++ model to focus on low-level details. The first of them tackle it by factorizing the prior probability over different resolutions. The second one uses only one level of this factorization, but it conditions on grayscaled low-resolution image quantized into 4-bit pixels. Although for reconstruction problems we were dealing with, the focus on low-level detail might not be a big issue, it would be interesting to investigate the usage of these two new models in the proposed approach.

Last but not least, since the proposed approach was rather successful on the denoising and two-fold single-image super-resolution problems, it should be investigated how it behaves on other inverse low-level vision problems such as deblurring, inpainting, or other super-resolution problems.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Manya V Afonso, José M Bioucas-Dias, and Mário AT Figueiredo. An augmented lagrangian approach to the constrained optimization formulation of imaging inverse problems. *IEEE Transactions on Image Processing*, 20(3):681–695, 2011.

Eirikur Agustsson and Radu Timofte. NTIRE 2017 challenge on single image super-resolution: Dataset and study. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2017. CVPRW 2017.*, pages 1122–1131, 2017.

Gilles Aubert and Pierre Kornprobst. *Mathematical problems in image processing: partial differential equations and the calculus of variations*, volume 147. Springer Science & Business Media, 2006. ISBN 0-387-95326-4.

Yoshua Bengio et al. Learning deep architectures for AI. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2009.

Hong Chang, Dit-Yan Yeung, and Yimin Xiong. Super-resolution through neighbor embedding. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–I. IEEE, 2004.

Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. Image denoising by sparse 3-D transform-domain collaborative filtering. *IEEE Transactions on Image Processing*, 16(8):2080–2095, 2007a.

Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. Color image denoising via sparse 3D collaborative filtering with grouping constraint in luminance-chrominance space. In *IEEE Conference on Image Processing, 2007. ICIP 2007.*, volume 1, pages I–313. IEEE, 2007b.

Shengyang Dai, Mei Han, Wei Xu, Ying Wu, and Yihong Gong. Soft edge smoothness prior for alpha channel super resolution. In *IEEE Conference on Computer Vision and Pattern Recognition, 2007. CVPR 2007.*, pages 1–8. IEEE, 2007.

Steven Diamond, Vincent Sitzmann, Felix Heide, and Gordon Wetzstein. Unrolled optimization with deep priors. *arXiv preprint arXiv:1705.08041*, 2017.

Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):295–307, 2016.

Daniel Glasner, Shai Bagon, and Michal Irani. Super-resolution from a single image. In *IEEE Conference on Computer Vision, 2009. ICCV 2009.*, pages 349–356. IEEE, 2009.

Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. Multi-dimensional recurrent neural networks. In Joaquim Marques de Sá, Luís A. Alexandre, Włodzisław Duch, and Danilo Mandic, editors, *Artificial Neural Networks – ICANN 2007: 17th International Conference, Porto, Portugal, September 9-13, 2007, Proceedings, Part I*, pages 549–558. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

Shuhang Gu, Lei Zhang, Wangmeng Zuo, and Xiangchu Feng. Weighted nuclear norm minimization with application to image denoising. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014. CVPR 2014*, pages 2862–2869, 2014.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016. CVPR 2016.*, pages 770–778, 2016.

Magnus R Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4(5):303–320, 1969.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016. CVPR 2016.*, pages 1646–1654, 2016.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Diederik P Kingma, Tim Salimans, and Max Welling. Improving variational inference with inverse autoregressive flow. *arXiv preprint arXiv:1606.04934*, 2016.

Alexander Kolesnikov and Christoph H. Lampert. PixelCNN models with auxiliary variables for natural image modeling. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1905–1914, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. *arXiv preprint arXiv:1609.04802*, 2016.

Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops 2017, CVPRW 2017.*, July 2017.

David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings of the IEEE Conference on Computer Vision, 2001. ICCV 2001.*, volume 2, pages 416–423. IEEE, 2001.

Microsoft Corporation. Microsoft cognitive toolkit (cntk), 2017. URL `https://github.com/Microsoft/CNTK/`.

Patrick Putzky and Max Welling. Recurrent inference machines for solving inverse problems. *arXiv preprint arXiv:1706.04008*, 2017.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Stefan Roth and Michael J Black. Fields of experts: A framework for learning image priors. In *IEEE Conference on Computer Vision and Pattern Recognition, 2005. CVPR 2005.*, volume 2, pages 860–867. IEEE, 2005.

Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1-4): 259–268, 1992.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533, 1986.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Tim Salimans, Andrej Karpathy, Xi Chen, Diederik P. Kingma, and Yaroslav Bulatov. PixelCNN++: A PixelCNN implementation with discretized logistic mixture likelihood and other modifications. In *International Conference on Learning Representations (ICLR)*, 2017.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Jian Sun, Zongben Xu, and Heung-Yeung Shum. Image super-resolution using gradient profile prior. In *IEEE Conference on Computer Vision and Pattern Recognition, 2008. CVPR 2008.*, pages 1–8. IEEE, 2008.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL `http://arxiv.org/abs/1605.02688`.

Lucas Theis and Matthias Bethge. Generative image modeling using spatial lstms. In *Advances in Neural Information Processing Systems*, pages 1927–1935, 2015.

A.N. Tikhonov and V.I.A. Arsenin. *Solutions of Ill-Posed Problems*. Scripta series in mathematics. Winston, 1977. ISBN 9780470991244.

Radu Timofte, Eirikur Agustsson, Luc Van Gool, Ming-Hsuan Yang, Lei Zhang, Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, Kyoung Mu Lee, et al. NTIRE 2017 challenge on single image super-resolution: Methods and results. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2017. CVPRW 2017.*, pages 1110–1121. IEEE, 2017.

Hiroki Tsurusaki, Masashi Kameda, and Prima Oky Dicky Ardiansyah. Single image super-resolution based on total variation regularization with gaussian noise. In *Picture Coding Symposium (PCS), 2016*, pages 1–5. IEEE, 2016.

Aäron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. pages 4790–4798, 2016a.

Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016b.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

Jianchao Yang, John Wright, Thomas S Huang, and Yi Ma. Image super-resolution via sparse representation. *IEEE Transactions on Image Processing*, 19(11):2861–2873, 2010.

Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE Transactions on Image Processing*, 2017a.

Kai Zhang, Wangmeng Zuo, Shuhang Gu, and Lei Zhang. Learning deep CNN denoiser prior for image restoration. *arXiv preprint arXiv:1704.03264*, 2017b.

Song Chun Zhu and David Mumford. Prior learning and gibbs reaction-diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(11): 1236–1250, 1997.

Daniel Zoran and Yair Weiss. From learning models of natural image patches to whole image restoration. In *IEEE Conference on Computer Vision, 2011. ICCV 2011.*, pages 479–486. IEEE, 2011.

List of Abbreviations

AL Augmented Lagrangian
ALM Augmented Lagrangian Method
AWGN Additive White Gaussian Noise
bpd bits per dimension
CDF Cumulative Distribution Function
CNN Convolutional Neural Network
HR High-Resolution
LR Low-Resolution
LSTM Long Short Term Memory
MAP Maximum a-posteriori
NLL Negative Log Likelihood
PSNR Peak signal-to-noise ratio
ReLU Rectifier Linear Unit
RNN Recurrent Neural Network
SISR Single-Image Super-Resolution
SSIM Structural similarity index
TV Total Variation

BM3D Block-Matching and 3D Filtering
DnCNN Denoising Convolutional Neural Network
EDSR Enhanced Deep Residual Networks for SISR
EPLL Expected Patch Log Likelihood
FOE Field of Experts
GAN Generative Adversial Network
ODP Unrolled Optimization with Deep Priors
RIDE Recurrent Image Density Estimator
RIM Recurrent Inference Machines
SRCNN Super-Resolution Convolutional Neural Network
VAE Variational Auto-Encoder
VIRGNN Variational Image Reconstruction using Generative Neural Networks
WNNM Weighted Nuclear Norm Minimization