Real-world low-light image enhancement using Variational Autoencoders

Olle Eriksson

Master's thesis
2020:E75

Lund Institute of Technology
Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Centrum Scientiarum Mathematicarum

Real-world Low-light Image Enhancement Using Variational Autoencoders

Olle Eriksson

Lund University

October 5, 2020

Thesis project carried out at Ericsson

Supervisors: Anders Berkeman (Ericsson), Magnus Oskarsson (LU)

Examiner: Niels Christian Overgaard

Abstract

Low-light image enhancement is a hard task mainly due to the amount of noise and little information stored in the dark image. In this thesis project we develop a method for low-light image enhancement based on a Conditional Variational Autoencoder (CVAE). The CVAE is a deep learning model trained with a specific objective function (ELBO). A CVAE can be implemented as a variant of a U-Net utilizing skip connections. We train models on a dataset called 'See in the dark' (SID), which contains aligned dark-bright image pairs. This thesis is mainly focusing on building models which accept an unprocessed RAW image input and predict a noise-free bright color image. Additionally we also consider alternative input data (sRGB instead of RAW) and the task of predicting a grayscale image (instead of color). Different neural-network architectures are implemented and compared, so are different strategies for merging image-patches. Different variants of the CVAE are compared, in particular a model only using a reconstruction loss is considered.

Regarding the different tasks the results turn out as expected. Models with RAW input data perform better than the model with sRGB input. Models which predict color perform worse than the one predicting only luminance. The output images for the model predicting only grayscale have a bit more detail and have less artifacts. The specific type of interpolation (between predicted patches) is of significant importance in case the inputs are extremely noisy, otherwise the faster methods are good enough. Comparison of different types of CVAE models with the model trained only using reconstruction loss shows that the very type of loss-function is not of great importance for the results, but the CVAE models are holding up well. The overall results are rather a function of the network architecture and the type of input data. Overall we conclude that the CVAE model can successfully be applied to the task of real-world low-light image enhancement, for visual improvement, in case the input is not too noisy. In case of an extremely noisy input the results are not as good.

Contents

Acronyms

AE Auotencoder. 8

CNN Convolutional Neural Network. 9

CVAE Conditional Variational Auotencoder. 16

ELBO Evidence Lower Bound. 12

MAE Mean Absolute Error. 33

MLP Multilayer Perceptron. 9

PSNR Peak Signal to Noise Ratio. 33

SSE Summed Squared Error. 15

SSIM Structural Similarity Index Measure. 33

VAE Variational Auotencoder. 11

Glossary

blocking effects Image artefact. Horizontal and vertical lines causing an unwanted 'mosaic' pattern. 6

hybrid An extension of the CVAE model. 18

ISO Setting in a camera. Refers to the ISO number (a value). 7

KL-divergence Short for Kullback Leibler-divergence. 11

ratio Or exposure ratio. Ratio between the exposure times of the bright (ground truth) image and the corresponding dark image. 7

RAW Unprocessed image with 'raw' sensor data. 5

SID Short for 'See in the dark'. 7

sRGB A color space for RGB images. A sRGB image is a type of processed image. 6

U-Net A type of neural network with skip connections. 10

1 Introduction

Low-light image enhancement is the task of enhancing an image captured in a dark environment. Processing the dark image aims to improve the visibility and quality of the image, this in terms of the exposure itself but also in terms of the noise level and colors. This is a hard task especially due to the high noise level, quantization and the low dynamic range in the dark image. An image captured in a extremely dark environment exhibits a very low Signal-to-Noise ratio, this is the reason classical methods usually fail to produce satisfying results. In recent years deep learning methods have successfully been applied to various image to image translation tasks, where the goal is to learn a mapping from an input image to an output image. Many works have been using an Autoencoder (AE), which is a neural network with a low dimensional bottleneck, or a U-Net, which is a architecture with skip connections avoiding the bottleneck. A later trend has been to use generative models based on a Variational Autoencoder (VAE) or its extension the Conditional Variational Autoencoder (CVAE). Whereas a VAE is modelling a probability distribution $p(x)$, a CVAE models a conditional distribution $p(x|y)$. Simplified, the VAE and CVAE are models defined by a objective function called the ELBO. The VAE is usually implemented as a neural network with a bottleneck, similar to an AE. The CVAE can be implemented as a variant of a U-Net.

Inspired by the work [2], which used a non-generative deep learning method based on a U-Net network [17] for low-light image enhancement, and motivated by the success of generative models on various image translation tasks this thesis is investigating whether a deep learning method based on a CVAE could successfully be applied to the task of real-world low-light image enhancement. This focus will focus on a particular low-light image dataset collected by [2] called 'See in the dark' (SID). This report refers to it as *SID-dataset*. The dataset contains paired dark and bright aligned images in RAW format (from which regular sRGB images can be extracted). The images in the dataset were captured outdoor in the night or in a dark interior environment, and there is a wide variation of scenes. An image in RAW format is unprocessed and contains the 'raw' data from the sensor of the camera. A RAW usually contains more information than a regular image (e.g. sRGB image saved as JPG), they have a higher bit-depth and provide a higher dynamic range. This thesis will focus on using RAW images as inputs to the models.

1.1 Related Work

In recent years there have been a few works using CNN-based deep learning methods for real-world image enhancement. Most have been using a non-generative approach or a Generative Adversarial Network (GAN), but not as many a VAE/CVAE. When it comes to works based on the *SID-dataset*, of course there is the original work [2]. Their ideas main were 1) using a U-Net architecture and 2) taking RAW image patches as inputs to the model. They use a network which accepts RAW patches of size 512×512 and outputs RGBs in double the resolution. Their network is constructed such that at testing time a full-scale image can be used as input to the network. Their method was shown to be superior to classical methods, mainly in terms of image-quality. Follow-up works include [16] which instead of a U-Net architecture aimed for a ResNet architecture, but this network did not use any kind of bottleneck. Another is [18], they used a deep learning model based on a Generative Adversial Network (GAN) with a U-Net architecture. They successfully managed to fit the GAN model to a variant of the *SID-dataset*.

Two more examples of applications of a GAN to real-world image enhancement are [26] and [15] which both applied GANs for de-quantization of low bit-rate real-world images. Regarding a CVAE, it was introduced by [19], they applied a CVAE to several image translation tasks, for example imputation of missing quadrants on MNIST [13] and segmentation, with great success and outperforming deterministic deep learning approaches. They argued that the model was best suited for multi-modal predictions, where the output is ambiguous given the input. Follow-up works using the CVAE include [3] which used a variant of a CVAE to the predict diverse colorizations based a grayscale input. Another is [6] which applied a CVAE based on a U-Net architecture to predict shape and appearance based on input images.

In the work [9] a CVAE model based on a U-Net architecture was used. They altered the model

to be used for arbitrary conditioning, and used the model for sampling of missing parts of images. Overall most works using a VAE/CVAE have considered images of low resolution as inputs to the model. This is in contrast to our work which focuses on patches from images of high resolution.

Based on our literature study we could not find any previous works with attempts of applying a VAE or CVAE on the task of low-light image enhancement, or for that matter on RAW image input data. Based on the presented previous works there are reasons to believe that a CVAE model with a U-Net architecture could be successfully applied to the task of low-light image enhancement of RAW images.

1.2 Aims and Objectives

Based on the success of [2] on the image to image translation problem:

> (main): predicting a bright/denoised color image in sRGB space based on a dark RAW input

This work will mainly consider the same problem. Additionally we will also consider the the following tasks:

> (1): predicting a bright/denoised grayscale image based on a dark RAW input, and
>
> (2): predicting a bright/denoised color image in sRGB space based on a dark sRGB image

Predicting a grayscale image is expected to be an easier task than predicting a color image. Using a sRGB input is of interest mainly due to the smaller memory size versus a RAW image. This thesis aims to construct/implement deep learning models mainly based on U-Net architectures, (as baseline) trained as CVAEs, for enhancing the dark images in the *SID-dataset*. Due to little previous work on the application of CVAEs on real-world images another main goal is to investigate more in detail how to implement the pipeline, and especially the network network architecture. Furthermore, we compare different ways of training the main model. The CVAE model is compared with a simpler model only using a reconstruction loss. The models are constructed to accept a small image patch as input. At testing time interpolation (between patches) is used to predict a full-size image. The goal here is to avoid so called blocking effects. The implemented models will be compared in terms of their performance, mainly in terms of measures but also by visual inspection. Compared to [2] our work considers task (1) which they did not. Furthermore we focus on CVAE models instead of regular U-Nets. To summarize, this thesis will try to answer the following main questions:

1. *Can a CVAE successfully be applied to the task of real-world low-light image enhancement?*
2. *What type of network architecture is suitable? And how should we train the model?*
3. *How can we (efficiently) combine the predicted patches to a full-scale image without getting any unwanted blocking effects?*
4. *What is the performance of the method for different types of input images?*

1.3 Organization of Report

The report is organized in the following way. We start by introducing the *SID-dataset* in section 2. Next after that, in the Background 3, a brief background knowledge on deep learning and latent models is given. This work is focusing on the VAE/CVAE models, naturally this work will therefor include an extensive theoretic background on these methods, see section 4. Before we continue with the *SID-dataset*, we are doing a demonstration of the VAE/CVAE models on the MNIST dataset to show how they can be implemented and used, see section 5. Next after this, in the Method section 6, our method for enhancing the low-light images in the *SID-dataset* is presented. Finally the Results section 7 presents the results of our method on the *SID-dataset*. The results are then later discussed and concluded in section 8 and section 9 respectively.

2 SID-dataset

'See in the dark' (SID) is a dataset of images which was released in 2018 [2]. The dataset consists of dark-bright pairs of RAW images of different scenes. Each image pair consists of a dark image, taken with a short exposure, and a bright image taken with a long exposure. The images are of real scenes, both indoor and outdoor, in a dark environment. The majority of the dark images are very noisy and exhibit a very low Signal-to-Noise ratio. They were captured with a short exposure time that is between 100 and 300 times as short as the corresponding correctly exposured bright image. The bright images are seen as ground truths. Since the dark images were captured with varying ISO and exposure time the dataset represents input images of varying level of noise and brightness. ISO is a setting in the camera, defining the camera's sensitivity to light. The value of the ISO has effect on the amplification of the analogue image signal. We consider only the half of dataset with images captured by a Sony camera which is using a Bayer sensor [21].

Each RAW image is of size $H \times W = 2848 \times 4256$ pixels, i.e. about 12 megapixels. Using a Python library called `rawpy`[1] a processed image in sRGB space (or a grayscale image) can be extracted, using the white balance of the camera. Given the exposure settings for a given pair, the ratio between the exposure time of the bright and dark images is used to amplify the dark RAW image. Thus a correct exposured (on average) image is obtained. This amplified image is although still very noisy. The RAW images are mosaicked and have all colors in a single channel. One way to arrange the data into channels, is by reducing the spatial resolution by a factor of four and packing each 2×2 Bayer block into four channels; one for blue, one for red, and two for green. The packed image can be seen as a RGBG image. Thus for an image of size $H \times W = 2848 \times 4256$ the packed image is four-channeled and of spatial size 1424×2128. This is the approach used in [2], following their success we use the same idea. Due to packing problems with Bayer-pattern showing up in images are disregarded. Our code for loading the images is based on theirs[2]. Regarding the sRGB images, each 2×2 RAW block is reduced to a single pixel, thus they are of spatial resolution 1424×2128 and have three channels.

The dataset is split into three parts: a training set consisting of 161 unique ground truth images, a validation set of 20 ground truth images and test set of 50 ground truth images. For each ground truth image there are at least one dark RAW of ratio either 100, 250 or 300. Thus several dark images have the same ground truth bright image. Regarding the ISO levels of the input images, they range between 50 and 25,600. Whereas the ratio indicates how dark the image is, the ISO level is an indication of the amount of noise. Examples of images in the dataset can be seen below in Figure 1.

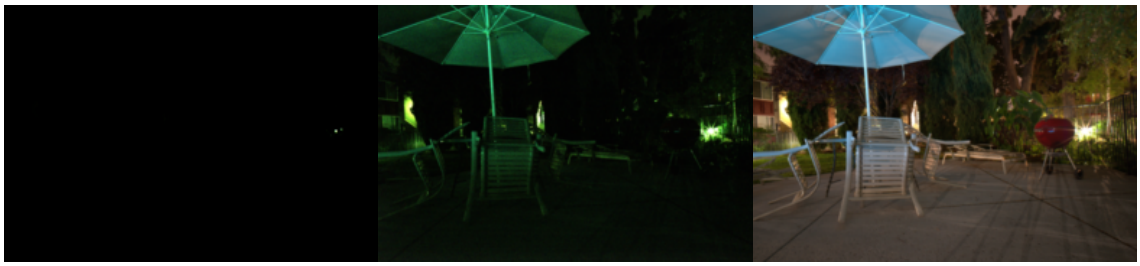


Figure 1: Example of dark-bright image pair from the dataset. We show downsampled images with order 1) non-amplified dark RAW, 2) amplified dark RAW, 3) bright sRGB (ground truth). We are ignoring one of the green channels for the RAW files when plotting. The non-amplified image (left) is the input to the pipeline. The amplified image (center) would be the input to the neural network model. The non-amplified dark image represents the 'darkness' of the original unprocessed input.

[1] `https://pypi.org/project/rawpy`
[2] Their code can be found at `https://github.com/cchen156/Learning-to-See-in-the-Dark`

3 Background

All models implemented in this project are based on Deep learning. These are all implemented in the library `PyTorch`. `PyTorch` is an open source machine learning library developed by Facebook and it is mainly intended for research purposes. The two main features which the library is based on are 1) tensor computation and 2) automatic differentiation. Automatic differentiation is a technique to evaluate the derivative of an expression, i.e. a computer program, with respect to some parameters. It is based on the fact that the output of a program is a product of a (nested) sequence of elementary operations and functions. It uses the chain rule to calculate the derivatives.

3.1 Dimension Reduction and Autoencoder

In representation learning the goal is to find informative non-redundant low dimensional features describing observed high dimensional data. The main assumption is that the observed data is located on a manifold of dimension D_1 embedded in the observation space which is of dimension D_2, where $D_1 < D_2$. Here D_1 represents the degrees of freedom, however this dimension is often unknown. An example of a manifold is the one for a curve, a curve has one degree of freedom, thus the underlying manifold has dimension $D_1 = 1$. One assumes there exists a function $f(\cdot)$ mapping a point in the manifold to the observation space and the goal is to learn this function. Notable unsupervised representation learning techniques are Principal Component Analysis (PCA), and the Autoencoder (AE).

The Autoencoder (AE) is a type of neural network F decomposed as $F = f \circ g$, the function $g(\cdot)$ is called the encoder and $f(\cdot)$ is the decoder. The encoder maps $x \in \mathbb{R}^{D_2}$ to $z \in \mathbb{R}^{D_1}$, and the decoder maps z to x-space. The AE has a bottleneck z of low dimension, it is the latent representation of the input x. The AE model is learned from data X, the goal is to learn the function $F(\cdot)$ which should approximate the identity function: $F(x) \approx x$. Due to the low dimension of the bottleneck, after training z (usually) represents global features of the input. This is the originally intended usage of the AE, but an AE can also be used for the purpose of modelling a transformation $y \to x$. The key feature of using a bottleneck is still maintained, but the goal is now to learn $F(y) \approx x$. In this case 'AE' refers to the type of network architecture and not the way it is trained.

3.2 Latent Models

Latent variable models are stochastic models which relate observed variables x with latent, or hidden, variables z. The main assumption is that the observed, usually high dimensional, variable x is generated from an underlying low dimensional random variable z. This z is mapped through a function $f(\cdot)$ taking z to x space, then in x space noise is added. This is essentially a model modelling how data is generated from the low dimensional z space. In a latent model just the x variable is observed, the z variable is not but is still a part of the model. We assume that the latent variable has a prior distribution $p(z)$, and that the observed variable x is originating from the model $p\big(x|f(z)\big)$, where $f(z)$ are the parameters of the distribution in x space. Since we only have data of x we have to consider its marginal distribution, which is in case of a continuous z given by

$$p(x) = \int_z p(x, z)dz = \int_z p(z)p\big(x|f(z)\big)dz \tag{1}$$

A latent space model is closely related to the decoder function in the AE. The latent model acts as a probabilistic decoder $f(\cdot)$. Since z is here seen as a random variable distributions are put on z and x. A probabilistic decoder is in other words a generative model. The remaining question is where does the encoder come into this framework, could we somehow also define a probabilistic encoder?

3.3 Deep Learning

Deep learning is an area within Machine learning where the models are based on Artificial neural networks (ANN). Simplified, an ANN can be seen as set of connected neurons together forming a directed weighted graph. Each neuron takes an input and maps it through an optional activation function to form the output. The goal is to learn the weights such that a chosen loss function is minimized. A simple type of an ANN is the Multilayer perceptron (MLP). It consists of several layers of stacked neurons, the intermediate layers are referred to as hidden layers. In an MLP two neighboring layers are fully connected in the sense that all the neurons in the first layer are connected to all the neurons in the second layer. Each such connection has an unique weight. Another type of neural network is the CNN which will be explained in detail soon.

The goal of training an ANN is to learn the weights θ of the neural network f_θ. Given training data, a chosen loss function L is to be minimized. In case of a image to image transfer problem $(y \to x)$, in the supervised setting the loss function compares the prediction $f_\theta(y)$ with the ground truth x. The loss function is optimized by using local continuous optimization techniques. Starting at random initialized weights, the weights of the network are updated iteratively by taking steps in the θ-space until convergence to (hopefully) a local minima.

Given a dataset X, the optimization acts on batches (y_m, x_m) of size M. Using a stochastic gradient descent scheme, the loss function is expressed as

$$L = L\big(x_m, f_\theta(y_m)\big).$$

The basic update rule (gradient descent) for the weights is

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L\big(x_m, f_{\theta_t}(y_m)\big),$$

here t is the iteration and $\eta > 0$ the learning rate.

In each forward pass one batch is passed through the network, after which the network parameters are updated. The choice of the learning rate η and the batch size M has effect on the convergence speed of the optimization. A common optimizer is the Adam optimizer [12], it differs from the scheme above in the way it has individual learning rates on per weight basis. Adam uses the first (mean) and second (variance) moments of the gradient to update the learning rates. Since a neural network is optimized by iterative optimization a global minimum cannot be expected to be learned.

3.3.1 CNN Networks

A Convolutional neural network (CNN) is a deep learning model based on convolutions (or cross correlations). It is most commonly applied to image data where there exist spatial and spectral dependencies. Designed for these purposes it utilizes parameter sharing and the concept of local receptive field efficiently. A CNN layer has its parameters stored in a filter/kernel, this kernel is ranging over all spectral dimensions but only over a local spatial field: this range is specified by the kernel size. The input to the layer is convolved with the kernel to create the output. The convolution operation can be seen as sweeping the filter over the input and computing the inner product between the filter and the covered part of the input. The convolution itself is a linear operation, but by using activation functions a non-linear mapping is obtained.

In the context of a CNN there are some important hyperparameters. The *kernel* size refers to the receptive field, i.e. a certain number of pixels in each of the spatial dimensions. The *stride* refers to the step length when the filter is moved. The number of *channels* refers to the number of spectral dimensions. *Padding* refers to how the input image is extended, usually by adding zeros, before the convolution. A convolutional layer may have several input channels, in case the input is an RGB image there are three channels. There might also be several output channels, such that the layer can learn several features maps. With a stride of two (and for a suitable kernel size / padding) the output of the CNN layer is downsampled two times compared to the input size. An alternative way to downsample is by using a stride-one CNN layer combined with max-pooling.

Whereas a regular CNN layer usually downsamples the input, a transposed CNN network is upsampling the input. A transposed convolutional layer is defined by the same hyperparameters as the regular convolutional layer, except (with a stride > 1) the output is of larger spatial dimension than the output. As mentioned earlier the convolution operation is linear, this means that operation can be expressed as a matrix multiplication. If I_1 is the input image, then the output of the convolution I_2 can expressed as $I_2 = CI_1$, where C is a sparse matrix corresponding to the convolution operation, it is defined by the weights of the filter. Thus in the forward pass we use C, and in the backward pass the transposed matrix C^T is used. If the roles of C and C^T are interchanged, such that the forward pass is using C^T, a corresponding transposed convolution is obtained. Thus the backward pass of a transposed convolution can be seen as a forward pass of a regular convolutional operation [4].

3.3.2 Skip Connections

Skip connections in a neural network, can in a nutshell be seen as connections which skips over intermediate layers. In a U-Net [17] architecture there are long range skip connections between the encoder and the decoder part of the network, whereas in a ResNet [7] architecture there are shortcuts between two layers which usually are close to each other. There are two distinguished ways to merge the skipped features with the sequential features: adding or concatenating. A U-Net uses concatenation, whereas a ResNet architecture uses addition. A simple U-Net architecture can be seen in Figure 2. U-Net architectures have been applied to many different image transfer tasks, such as segmentation and image denoising. The U-net architecture is a form of an AE, except there are shortcuts between the encoder and decoder.

Skip connections have been shown to simplify the loss space, making it less chaotic, which in turn makes optimization easier [14]. They also make the gradients back-propagate to earlier layers in a better way, due to less of a problem with vanishing/exploding gradients. Overall, skip connections can stabilize the training. Using long range skip connection between the encoder and the decoder, features from the encoder are passed over directly to the decoder, avoiding the bottleneck. In the case of image-data, these kind of long range skip connections have been shown to help the network to learn low level image features, which are lost when downsampling [1].

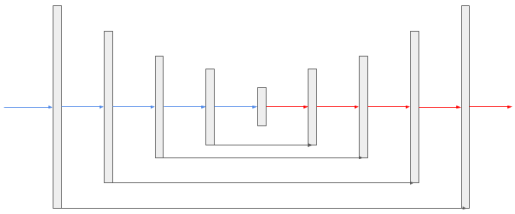


Figure 2: A U-Net architecture. The blue arrows represent the downsampling computation blocks, whereas the red ones represent the upsampling computation blocks. The black arrows are skip connections between the encoder and the decoder. Removing the skip connections we get a regular AE.

4 Theory

In this section the VAE and CVAE models are derived and described. First we introduce the information theoretic measure KL-divergence which is an important ingredient in the models.

4.1 KL-divergence

The KL-divergence between two probability densities p_1 and p_2 over a random variable X is defined as

$$D_{KL}\big(p_1||p_2\big) = E_{p_1(x)}\big[\log \frac{p_1(x)}{p_2(x)}\big]. \tag{2}$$

The KL-divergence measures the difference between p_1 and p_2. Jensen's inequality [5] can be used to prove some properties of the KL-divergence. It states that if $\gamma(\cdot)$ is a convex function, and X is a random variable, then the following is true [23]

$$\gamma(E[X]) \leq E[\gamma(X)].$$

Now if we use $\gamma(\cdot) = -\log(\cdot)$, which is a convex function we see the following

$$D_{KL}\big(p_1||p_2\big) = E_{p_1(x)}[\log \frac{p_1(x)}{p_2(x)}] = E_{p_1(x)}[-\log \frac{p_2(x)}{p_1(x)}] \geq$$
$$-\log E_{p_1(x)}[\frac{p_2(x)}{p_1(x)}] = -\log \int p_2(x)dx = -\log 1 = 0.$$

Thus it clear that the KL-divergence is positive. We also find that if $p_1(x) = p_2(x)\ \forall x$ we have $D_{KL}(p_1||p_2) = 0$. Thus it is also clear that the KL-divergence is zero if distributions are the same. In fact, it can be shown that the KL-divergence is zero if and only if the distributions are the same. If the distribution $p_2(x)$ has less support than $p_1(x)$, then we get into a problem since $\lim_{\alpha\to 0} p(x)\log(p(x)/\alpha) = \infty$. This could happen when $p_2(x)$ is a distribution with very small variance in comparison with $p_1(x)$.

4.2 Variational Autoencoder (VAE)

The Variational Autoencoder (VAE), can seen be as a mathematical construction from two different perspectives: from the perspective of the Autoencoder, and from the perspective of Variational Inference (VI). The loss function of the VAE is derived from VI, but the implementation can be interpreted as a stochastic Autoencoder. The VAE is a generative algorithm, modelling how the data x is generated from some hidden factors of variation z. We will now consider the VI perspective.

Usually when one wants to statistically model data x one would set up a probabilistic model $p_\theta(x)$ depending on parameters θ and maximize the likelihood of data: $\max_\theta p(x|\theta)$. To get good results we need a very good model of data, we need to have a very good understanding of the data distribution. Instead of modelling $p(x)$ directly VAEs include a latent variable z in the model, assuming the data x is generated from the variable z according to the distribution $p(x, z) = p(x|f(z))p(z)$. This generative model is visualized in Figure 3. The generative model implies that the marginal likelihood of x is given by equation (1). This distribution has potential to be more complex than a simple $p(x)$ due to the fact that x depends on an additional variable z.

There is however one big difficulty with this approach, even though $p(x, z)$ is tractable, the integral may be very hard to compute. It may not have an analytic solution or good estimator, e.g. when $f(\cdot)$ is a neural network, meaning that we cannot solve $\max_\theta \int p_\theta(x, z)dz$ directly. In principle we could just use a Monte Carlo approximation for $p(x) = E_{p(z)}[p(x|f(z))]$ directly by drawing samples from $p(z)$, but this is not a efficient method: a lot of samples need to be drawn to get a good estimator.

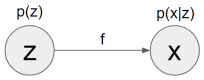


Figure 3: Graphical representation of the generative model. The latent variable z is the source random variable. Given a value of z it is mapped through the function $f(\cdot)$ to the observation space where noise is added.

The main idea behind VI is to approximate the posterior $p(z|x)$ with $q(z|x)$, where $q(z|x)$ is a tractable distribution. By introducing $q(z|x)$ it is possible to maximize a lower bound on the likelihood, the Evidence Lower Bound [22], instead of maximizing the intractable marginal likelihood. The distribution $q(z|x)$ is called the recognition model, just like the generative model the recognition model uses a function $g(\cdot)$ according to $q(z|g(x))$, where $g(x)$ is the parameters of the distribution $q(z|x)$. A graphical representation of the recognition model is shown in Figure 4. The generative and recognition model are the two main components of the VAE model.

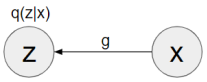


Figure 4: Graphical representation of the recognition model. Given a value of x it is mapped through the function $g(\cdot)$ to the latent space where noise is added.

The VAE uses the so called Amortized inference technique, the idea behind Amortized inference is to use a set of shared parameters to model the relation between x and z. This basically means that instead of optimizing a set of free parameters, different for each x, parameters of the functions $f(\cdot)$ and $g(\cdot)$ mapping between x and z are optimized. Neural networks are powerful function approximators [24], for this reason f and g are implemented as such functions. Thus optimization of a VAE model corresponds to finding the parameters, or weights, of them.

4.2.1 ELBO

We will now introduce the objective function of the VAE model: (ELBO). As its name suggests, it is a lower bound on the marginal likelihood of x. First we introduce some notations:

X	dataset
x	data point $\in X$
z	hidden/latent variable
$p(z)$	prior distribution over z
$p(x\|z)$	posterior distribution over x
$p(x, z)$	generative model
$q(z\|x)$	recognition model, approximative posterior distribution over z
$f(\cdot)$	function mapping a point z to parameters of $p(x\|z)$
$g(\cdot)$	function mapping a point x to parameters of $q(z\|x)$
θ	parameters/weights of $f(\cdot)$
ϕ	parameters/weights of $g(\cdot)$

We will use the notation $q_\phi(z|x)$ for the recognition model, this is another notation for $q\big(z|g_\phi(x)\big)$. It should be understood that ϕ are parameters of $g(\cdot)$. In the same way $p_\theta(x|z)$ is used instead of the notation $p\big(x|f_\theta(z)\big)$ for the posterior over x, here θ are parameters of $f(\cdot)$. $p_\theta(x, z) = p(z)p_\theta(x|z)$ is thus the generative model. We might want to use parameters for the prior $p(z)$ as well, then $p_\theta(x, z) = p_\theta(z)p_\theta(x|z)$, and θ refers to the parameters of both factors. Optimization of the ELBO aims to jointly find the parameters θ and ϕ.

The marginal log likelihood of one data point x, can be expressed as

$$\log p_\theta(x) = E_{q_\phi(z|x)}\big[\log p_\theta(x)\big].$$

Using Bayes theorem

$$p_\theta(x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(z|x)} = \frac{p_\theta(x,z)}{p_\theta(z|x)},$$

the marginal can be split into two factors

$$\log p_\theta(x) = E_{q_\phi(z|x)}\big[\log p_\theta(x)\big] = E_{q_\phi(z|x)}\big[\log \frac{p_\theta(x,z)}{p_\theta(z|x)}\big]$$
$$= E_{q_\phi(z|x)}\big[\log \frac{p_\theta(x,z)}{p_\theta(z|x)}\frac{q_\phi(z|x)}{q_\phi(z|x)}\big] = E_{q_\phi(z|x)}\big[\log \frac{p_\theta(x,z)q_\phi(z|x)}{q_\phi(z|x)p_\theta(z|x)}\big].$$

Using the log-rule $\log ab = \log a + \log b$ and the fact that the expectation is linear we get

$$\log p_\theta(x) = E_{q_\phi(z|x)}\big[\log \frac{p_\theta(x,z)q_\phi(z|x)}{q_\phi(z|x)p_\theta(z|x)}\big] = \overbrace{E_{q_\phi(z|x)}\big[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\big]}^{\text{A}} + \overbrace{E_{q_\phi(z|x)}\big[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\big]}^{\text{B}}.$$

Here term B can easily be identified as a KL-divergence according to

$$E_{q_\phi(z|x)}\big[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\big] = D_{KL}\big(q_\phi(z|x)||p_\theta(z|x)\big)$$

This term is intractable to compute due to the fact $p_\theta(z|x)$ is unknown. The term A is in fact the ELBO function, from now on we will use notation $L_{\theta,\phi}(x)$ for this term. To be explicit, the ELBO is given by

$$L_{\theta,\phi}(x) = E_{q_\phi(z|x)}\big[\log \frac{p_\theta(x,z)}{q_\phi(z|x)}\big] = E_{q_\phi(z|x)}\big[\log p_\theta(x,z) - \log q_\phi(z|x)\big]. \tag{3}$$

The ELBO can expanded even more according to

$$L_{\theta,\phi}(x) = E_{q_\phi(z|x)}\big[\log \frac{p_\theta(x|z)p_\theta(z)}{q_\phi(z|x)}\big] = E_{q_\phi(z|x)}\big[\log p_\theta(x|z)\big] + E_{q_\phi(z|x)}\big[\log \frac{p_\theta(z)}{q_\phi(z|x)}\big].$$

Using the logarithm rule $\log(a/b) = -\log(b/a)$ we see that the second term can be expressed as a KL-divergence according to

$$L_{\theta,\phi}(x) = E_{q_\phi(z|x)}\big[\log p_\theta(x|z)\big] - E_{q_\phi(z|x)}\big[\log \frac{q_\phi(z|x)}{p_\theta(z)}\big]$$
$$= E_{q_\phi(z|x)}\big[\log p_\theta(x|z)\big] - D_{KL}\big(q_\phi(z|x)||p_\theta(z)\big). \tag{4}$$

Thus we have (if we forget about the old A)

$$\overbrace{\log p_\theta(x)}^{\text{A}} - \overbrace{D_{KL}\big(q_\phi(z|x)||p_\theta(z|x)\big)}^{\text{B}} = L_{\theta,\phi}(x) = \overbrace{E_{q_\phi(z|x)}\big[\log p_\theta(x|z)\big]}^{\text{C}} - \overbrace{D_{KL}\big(q_\phi(z|x)||p_\theta(z)\big)}^{\text{D}}. \tag{5}$$

Now since the positiveness property of the KL-divergence it can be seen that the ELBO is a lower bound on the marginal likelihood

$$L_{\theta,\phi}(x) = \log p_\theta(x) - \overbrace{D_{KL}\big(q_\phi(z|x)||p_\theta(z|x)\big)}^{\geq 0} \leq \log p_\theta(x).$$

4.2.2 Intuition on the ELBO

In equation (5) term A is the marginal log likelihood, term B is the KL-divergence between the approximative and true posterior over z, term C can be seen as a reconstruction term, and term D is the KL-divergence between the recognition model and the prior. The last term can be seen as a regularizer. Since neither A or B can be computed, we cannot directly optimize the left hand

side of (5). But we can actually optimize the right hand side, since both terms C and D can be computed given suitable model choices. Thus the objective which is optimized is (4) which is the same as (3). Optimizing the right hand side the left hand side is implicitly optimized.

By optimizing the ELBO the marginal likelihood of x is approximately optimized. This implies that the generative model will be optimized. Also the the difference between the true posterior $p_\theta(z|x)$ and the approximate posterior $q_\phi(z|x)$ is minimized (in KL-divergence sense). Meaning that the recognition model will be improved. This ensures that $p_\theta(z|x) \approx q_\phi(z|x)$. Now we realize that the recognition model $q_\phi(z|x)$ acts as a stochastic encoder, taking a data point x and mapping it to a distribution in z space, and that the posterior $p_\theta(x|z)$ acts as a stochastic decoder, taking a latent variable z and mapping it to a distribution in x space.

The recognition model $q_\phi(z|x)$ tells us from what values of z, x most probably was generated. Since term C, the reconstruction term, is taking the expectation with respect to $q_\phi(z|x)$ it helps out with a few things. First it improves Monte Carlo sampling over using $p(z)$ as in the naive approach $p(x) = E_{p(z)}[p(x|z)]$, since $q_\phi(z|x)$ is dependent on x and thus only points z which are likely to produce x will be sampled meaning fewer samples are needed in the Monte Carlo estimator. Second, since it is an expectation with respect to a distribution, in a sense all latent points z which are likely under $q_\phi(z|x)$ should reconstruct x. This helps out with forming good semantics of the latent space. Term D, the regularization term, forces $q_\phi(z|x)$ to agree with the prior $p_\theta(z)$ for all x.

4.2.3 Reparameterization

The goal of optimizing the ELBO is to find the model parameters θ and ϕ. Using an iterative optimizer, gradients with respect to the model parameters θ and ϕ need to be calculated. However since the reconstruction term is a stochastic function of the encoder parameters ϕ, it is not straight forward how to compute the gradient of $L_{\theta,\phi}(x)$ with respect to ϕ. We want to use Monte Carlo estimates of the gradients, for that we need to take gradients inside the expectations.

Starting with θ, the gradient of the right hand side of equation (3) can be taken inside the expectation, assuming continuity, since the $q_\phi(z|x)$ does not depend on θ according to

$$\begin{aligned}\nabla_\theta L_{\theta,\phi}(x) =& \nabla_\theta E_{q_\phi(z|x)}\big[\log p_\theta(x,z) - \log q_\phi(z|x)\big] \\ &= E_{q_\phi(z|x)}\big[\nabla_\theta\big(\log p_\theta(x,z) - \log q_\phi(z|x)\big)\big] \\ &= E_{q_\phi(z|x)}\big[\nabla_\theta \log p_\theta(x,z)\big].\end{aligned}$$

For the parameters ϕ, the gradient ∇_ϕ cannot be taken inside the expectation due to the fact $q_\phi(z|x)$ depends itself on ϕ. To solve this problem, Kingma. et. al introduced the 'reparameterization trick' [11]. Instead of sampling from the $q_\phi(z|x)$ directly, the idea is to use an independent random variable $\epsilon \sim p(\epsilon)$, i.e. independent noise, and express $z \sim q_\phi(z|x)$ as a differentiable transformation $T(\cdot)$ of this random variable according to

$$z = T_\phi(\epsilon, x).$$

Due to this transformation the expectation with respect to $q_\phi(z|x)$ can instead be expressed in terms of ϵ, this in turn allows us to take gradients inside the expectations. Given a function $\gamma_{\theta,\phi}(x,z)$ of the parameters, a data point x and the hidden variable z, the expectation of this function with respect to the distribution $q_\phi(z|x)$ is the same as the expectation with respect to ϵ, provided $z = T_\phi(\epsilon, x)$, i.e.

$$E_{q_\phi(z|x)}[\gamma_{\theta,\phi}(x,z)] = E_{p(\epsilon)}[\gamma_{\theta,\phi}\big(x, T_\phi(\epsilon,x)\big)].$$

Thanks to the reparameterization trick we are able to compute gradients in a simple way.

For a Gaussian distribution $z \sim \mathcal{N}(\mu, \Sigma)$ we can use the reparameterization trick by sampling ϵ from $\mathcal{N}(0, I)$ and using $T(\epsilon) = \mu + L\epsilon$ where L is a lower triangular Cholesky factorization of Σ.

If it would happen that we have an analytical solution to the KL-divergence term in equation (4) we use the reparamterization trick to take gradients according to

$$\nabla_{\theta,\phi} L_{\theta,\phi}(x) = \nabla_{\theta,\phi} E_{p(\epsilon)}\big[\log p_\theta\big(x|T_\phi(\epsilon,x)\big)\big] - \nabla_{\theta,\phi} D_{KL}\big(q_\phi(z|x)||p_\theta(z)\big).$$

Using a L sample Monte Carlo estimator, with $\epsilon_1, \epsilon_2, \ldots, \epsilon_L \sim p(\epsilon)$, we have an estimation of the ELBO according to

$$L_{\theta,\phi}(x) \approx -D_{KL}\big(q_\phi(z|x)||p_\theta(z)\big) + \frac{1}{L}\sum_{i=1}^{L} \log p_\theta\big(x|z_i = T_\phi(\epsilon_i, x)\big). \tag{6}$$

4.2.4 Implementation

The ELBO derived above applies only to one data point x, but we want to maximize a function over a whole dataset X. Assuming X consists of i.i.d. observations x, the total ELBO is given by

$$L_{\theta,\phi}(X) = \sum_{x \in X} L_{\theta,\phi}(x).$$

This function should be maximized w.r.t. the model parameters θ, ϕ according to

$$\max_{\theta,\phi} L_{\theta,\phi}(X) = \min_{\theta,\phi} -L_{\theta,\phi}(X).$$

For most experiments we will use $L = 1$ Monte Carlo samples to estimate the reconstructions term. Mainly since $L > 1$ does not scale well, in terms of training times.

We have yet not specified how a VAE is implemented in practice. First of all the functional forms of the functions $f(\cdot)$ in $p\big(x|f_\theta(z)\big)$, and $g(\cdot)$ in $q\big(z|g_\phi(x)\big)$ need to be specified. They are implemented as neural networks. By fixing the architecture, θ and ϕ refer to the weights and biases of these neural networks. For example if the encoder is modelled as a Gaussian distribution the outputs y of the network would be a mean vector y_1 and a variance vector y_2. By using neural networks for the encoder and decoder functions, the reparameterization trick corresponds to moving the randomness in z; an internal node, to an input node ϵ. The distributions: $p_\theta(z)$, $p_\theta(x|z)$ and $q_\phi(z|x)$ also need to be specified. In the vanilla variant of the VAE the distributions are all Gaussian distributions:

- $p_\theta(z) = \mathcal{N}(0, I)$
- $q_\phi(z|x) = \mathcal{N}\big(\mu_\phi(x), \sigma^2_\phi(x)\big)$
- $p_\theta(x|z) = \mathcal{N}\big(\mu_\theta(z), I\big)$

Here all distributions have diagonal covariances, i.e. no cross correlation between dimensions of the variable in question. Although all these distributions are Gaussian, there is nothing saying that the marginal $p_\theta(x)$ is Gaussian. By using a Gaussian distribution with unit variance for the decoder distribution $p_\theta(x|z)$ according to above, the reconstruction term becomes a summed squared error (SSE) (plus additional constant), or L_2 error, between the input x and the decoder output $\mu_\theta(z)$, according to

$$-\log p_\theta(x|z) = \frac{1}{2}||x - \mu_\theta(z)||_2^2. \tag{7}$$

Using Gaussian distributions for the prior $p_\theta(z)$ and encoder distribution $q_\phi(z|x)$ the KL-divergence term is given by a closed form expression (25). Due to the diagonal covariances the dimensions of z can be seen as independent continuous factors of variations. The prior $p_\theta(z) = \mathcal{N}(0, I)$ is pulling $q_\phi(z|x)$ towards the origin of the latent space.

The VAE can seen as a stochastic AE. Just like with an AE the (random) variable z is chosen to be low dimensional, in comparison with the observation space, thus forcing the encoder to learn global structures of the input that the decoder can use to approximately reconstruct it. In comparison with an AE which is trained with only a reconstruction loss, what are the main differences? It comes down to a few things. First of all there is no regularization of the latent space in the AE, thus there is nothing that prevents similar x's to map to z's far away from each other through the encoder mapping. The VAE is encoding points x to distributions, and decoding z in expectation. This helps with forming good latent semantics. A trained VAE can also be used to generate new data.

4.2.5 Alternative Objectives

Tthere is nothing saying that the output of the decoder (given z) has to be modelled as a Gaussian. It could for example be modelled as a Laplace distribution instead, and we would get a L_1 loss function. The summed absolute error (SAE), or L_1 error, between two variables (e.g. images) I_1 and I_2 is given by

$$L_1(I_1, I_2) = ||I_1 - I_2||_1. \tag{8}$$

In practical applications one may want to instead of optimizing the regular ELBO, optimize a variant with a weight $\beta > 0$ on the KL-divergence according to

$$L_{\theta,\phi}(x) = E_{q_\phi(z|x)}\big[\log p_\theta(x|z)\big] - \beta \cdot D_{KL}\big(q_\phi(z|x)||p_\theta(z)\big) \tag{9}$$

There are several reasons to why one would like to do this. First of it encourages the optimization to focus more or less on the KL-divergence term. A low KL-divergence often implies worse reconstructions, due to the regularizing effect of the prior encouraging overlap between the empirical encoder distributions. We can thus control the trade off between reconstructions and KL-divergence by using a weight like this.

A VAE is really not intended to model a conditional distribution $p(x|y)$, although the encoder could be modified to accept another variable y instead, i.e. $q_\phi(z|y)$. On the other hand the CVAE is theoretically more justified for this purpose.

4.3 Conditional Variational Autoencoder (CVAE)

Imagine a scenario where we instead of learning a distribution $p(x)$ would like to learn a conditional distribution $p(x|y)$, i.e. a distribution that depends on the value of another variable y. The variable y could be anything we have data of, e.g. a label for x. In case of an image transfer problem we could model the transformation $y \to x$ in a probabilistic manner using $p(x|y)$. For these purposes we can use the Conditional VAE (CVAE) model [19], which is an extension of the original VAE. The ELBO of the CVAE is given by [19]

$$L_{\theta,\phi}(x|y) = L_{CVAE} = E_{q_\phi(z|x,y)}[\log p_\theta(x|y,z)] - D_{KL}\big(q_\phi(z|x,y)||p_\theta(z|y)\big). \tag{10}$$

The first term is approximated by decoding Monte Carlo samples (obtained by reparamterization) from $q_\phi(z|x,y)$, just like for the VAE, according to

$$E_{q_\phi(z|x)}[\log p_\theta(x|y,z)] \approx \frac{1}{L}\sum_{i=1}^{L} \log p_\theta\big(x|y,z_i\big),\ z_i \sim q_\phi(z|x,y). \tag{11}$$

If not else stated $L = 1$ sample Monte Carlo estimations are used. When implementing a CVAE using neural networks, we let the $q_\phi(z|x,y)$ network encode x and y into z. The decoder $p_\theta(x|y,z)$ decodes the latent variable z and y, into x. One can imagine that using e.g. the label of a sample as y may help improving the model and make it more expressive. The conditional distribution $p_\theta(z|y)$ is just like the encoder and decoder implemented by a neural network and we

call it *prior network* or just *prior*. Both the encoder and prior network are downsampling networks leading to the bottleneck z. The decoder accepts two inputs: z and y. One way to implement the y dependence is by using skip connections from the prior network into the decoder network. The KL-divergence for a CVAE is between two conditional/modelled (factorized Gaussian) distributions, given by (24). Figure 5 shows a visualization of this KL-divergence in the case of two one-dimensional normal distributions.

During training, even though the goal is to predict a distribution over x given y, x is used as an input to the model (to the encoder). Thus the task of training can be seen as learning to reconstruct x given y, which should be an easier task than directly predicting x. One thing to note here is that we are using the opposite notations compared to article where the model was introduced [19]: we have switched $x \leftrightarrow y$.

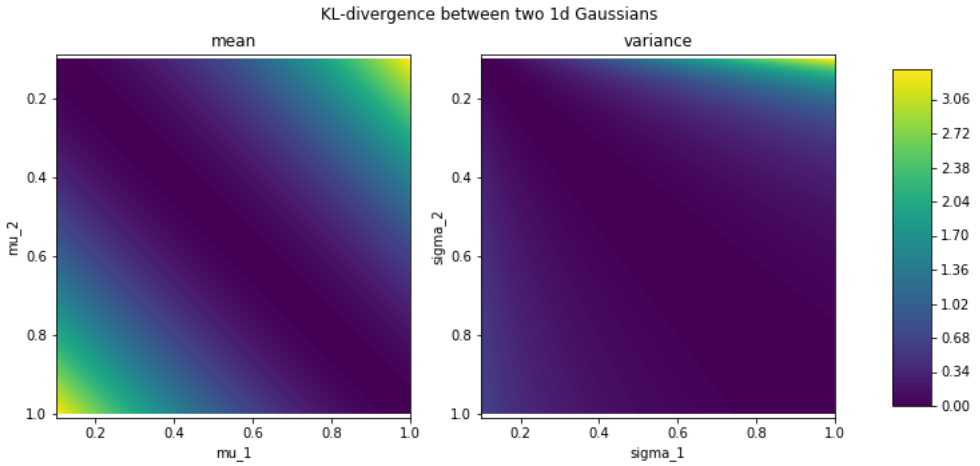


Figure 5: KL-divergence between two one-dimensional normal distributions q and p. In the left plot the KL-divergence $D_{KL}\big(q||p\big)$ between $q(z) = \mathcal{N}(\mu_1, 1)$ and $p(z) = \mathcal{N}(\mu_2, 1)$ is plotted. In the right plot the KL-divergence is between $q(z) = \mathcal{N}(0, \sigma_1)$ and $p(z) = \mathcal{N}(0, \sigma_2)$. Remark that as the variance of p: $\sigma_2 \to 0$ the KL-divergence blows up. With fixed variances the KL-divergence is a squared error between the two means.

4.3.1 Alternative Variants

That was the original variant of the CVAE, there are some alternatives though. For example the encoder distribution $q_\phi(z|x, y)$ does not have to depend on y, i.e. $q_\phi(z|x)$ instead. The y dependence in the decoder could be ignored and instead the decoder would be $p_\theta(x|z)$. By changing the distributions like this, i.e. by conditioning on less variables we can just interchange the terms in equation (10). For example when modelling the encoder as $q_\phi(z|x)$ we get the following ELBO

$$L_{\theta,\phi}(x|y) = L_{CVAE} = E_{q_\phi(z|x)}[\log p_\theta(x|y, z)] - D_{KL}\big(q_\phi(z|x)||p_\theta(z|y)\big). \tag{12}$$

This is the variant we mainly focus on. Regarding the distributions we stick to Gaussians on the bottleneck according to:

- $p_\theta(z|y) = \mathcal{N}\big(\mu_\theta(y), \sigma^2_\theta(y)\big)$
- $q_\phi(z|x) = \mathcal{N}\big(\mu_\phi(x), \sigma^2_\phi(x)\big)$

For purpose of numerical stability all CVAE models in this thesis use an unit variance according to $\sigma^2_\theta(y) = I$. The distribution of the decoder is implicitly set by the reconstruction loss function we consider.

A CVAE is trained in a similar manner as a VAE, and the reparameterization trick is used to sample from the encoder distribution. During training z is sampled from $q_\phi(z|x)$, and this z is passed to the decoder $p_\theta(x|y,z)$. At testing time when the goal is to predict x based on y the following algorithm is used:

$$p_\theta(x|y,z),\ z \leftarrow E[z|y],$$

where the expectation is with respect to $p_\theta(z|y)$. Thus the CVAE model is vastly different at testing stage compared to training stage. The main difference is that the encoder network is not used anymore, it is only used during training.

Another alternative variant of the CVAE is obtained by also ignoring the y dependence in the decoder, such that decoder is $p_\theta(x|z)$ instead. In this case the ELBO is

$$L_{\theta,\phi}(x|y) = L_{CVAE} = E_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}\big(q_\phi(z|x)||p_\theta(z|y)\big). \tag{13}$$

4.3.2 Hybrid Model

When training a CVAE, the goal of the KL-divergence objective is to match the distributions $q_\phi(z|x)$ and $p_\theta(z|y)$. Since the prior of a CVAE is modelled as $p_\theta(z|y)$, i.e. it is not unconditional anymore, the KL-divergence is not acting as much as a regularizer as for a VAE. The marginal $p_\theta(z)$ could be very expressive due to this. This allows more freedom than an unconditional $p_\theta(z)$. At evaluation time when we want to predict x based on y, the encoder network $q_\phi(z|x)$ can of course not be used. Instead we use the prior network to 'encode' into the latent variable z, and decode the mean from $p_\theta(z|y)$. This is in great contrast with how the model is trained, during training z is sampled from $q_\phi(z|x)$. For this reason the prior and encoder distributions should match as well as possible.

One way to force the distributions to match better is by putting a large weight β on the KL-divergence term. Another way, suggested by [19], is to use the so called GSNN objective function. The GSNN objective is given by

$$L_{GSNN} = E_{p_\theta(z|y)}[\log p_\theta(x|y,z)]. \tag{14}$$

This objective can be obtained from setting $q_\phi(z|x,y) = p_\theta(z|y)$ in the CVAE objective L_{CVAE} (10) above. Together the two objectives are combined as a weighted average with a weight $\alpha > 0$ into a hybrid objective according to

$$L_{hybrid} = \alpha L_{CVAE} + (1-\alpha) L_{GSNN}. \tag{15}$$

The hybrid model can be trained in the same way as a CVAE, e.g. by using the reparameterization trick to sample from $p_\theta(z|y)$.

5 Experiments on MNIST

To get a better understanding of the VAE and CVAE methods we implement a few models and train them on the MNIST dataset [13]. MNIST is a dataset of images of handwritten digits, each image is grayscale of size 28×28 and consists of one digit. There are 60,000 training samples and 10,000 validation samples. The pixel values of the images are normalized to the range $[0, 1]$, and thus each image is seen as a continuous signal. Most pixels are either black or white, this combined with the fact it is a very structured dataset (all digits centered in the frame) makes the dataset easy to work with. For these experiments two different architectures are used: a pure MLP as given by Table 12 and a combine CNN-MLP as given by Table 13 in the appendix. For optimization we use the Adam optimizer with the standard settings, and a batch size of 1000 samples and 200 epochs of training.

5.1 Reconstruction

The goal for these experiments is to model an unconditional distribution $p(x)$ over the MNIST dataset. Starting with the MLP architecture. We compare a vanilla VAE with an AE, both using two-dimensional bottlenecks. The AE is using the same network architectures as the VAE, except the z_var layer (for the variance in the VAE) which is not used. The AE is trained using the regular L_2 reconstruction loss (7), and no reparameterization or KL-divergence. VAEs with larger bottlenecks are also trained, as well as a VAE with the CNN-MLP architecture. Lastly we train a CVAE model given by (13) with a conditional prior $p_\theta(z|y)$ depending on the label (number between 0 and 9) of the input. The only difference between this CVAE and a VAE is that the conditional prior has replaced $\mathcal{N}(0, I)$. The prior network of the CVAE is modelled as an one hidden layer MLP with ten hidden nodes accepting an one-hot vector corresponding to the class label.

Regarding the reconstructions we use ten samples from the encoder distribution $q_\phi(z|x)$ and then average (for each input image), this to reduce the variance. The reconstruction errors are reported as SSE errors (ignoring the 0.5 factor in (7)). More details about each of the considered models, and the resulting errors over the training set and validation set can be found in Table 1.

Starting with the comparison of the MLP-VAEs with different latent dimensions. Increasing the dimensions of the latent space better reconstructions can be obtained. It is clear that up to 10 dimensions, the more dimensions we use the better reconstructions we get. Using 50 dimensions results are not better compared to 10. We observed that many of the components of the latent vector z were not used. The reconstructions from the 10-dimensional VAE with the CNN-MLP architecture (VAE-CNN) are significantly better than those of the MLP architecture (VAE-10). Comparing the AE with the two-dimensional VAE (VAE-2), we notice the AE is overfitting a bit more than the VAE but the reconstructions are better. The KL-divergence is punishing the reconstructions. Comparing the CVAE with the 10-dimensional VAE (VAE-10) we observe improved reconstructions, which can be explained with the much more flexible prior.

Evaluating the VAE model with a two-dimensional bottleneck (VAE-2), the empirical encoder distribution (of the encoded means) i.e. the latent space can be seen in Figure 6. Ignoring the colors, i.e. the labels, the latent space is tightly clustered around the origin. In fact it looks much like a $\mathcal{N}(0, I)$ distribution which is to be expected due to the KL-divergence with such a prior. Taking into account the colors, we can see that those of same label are clustered together in the latent space. The learned image manifold is shown in the right figure. Observe the (close to) independence between the two dimensions, this is due to the prior being diagonal. It seems like semantics in observation space are in general kept through the encoder mapping. The first dimension z_0 seems to represent roundness, whereas z_1 seems to represent the tilt of the digits.

The latent spaces of the 10-dimensional VAE (VAE-10) and CVAE are compared. To visualize the difference between the latent spaces we calculate and project the empirical encoder distribution of the validation data onto the two first PCA components in each case, the results are shown in Figure 7. We notice better clustering of the latent space for the CVAE. There are several modes, each class has its own spherical Gaussian component.

We conclude that the type of architecture is very important. The bottleneck size is important (up to a certain size), so is the combination of layers (CNN, MLP). Based on these experiments we can see that using a more flexible prior (using a CVAE) can improve upon the restrictive vanilla unconditional prior in the VAE, in terms of reconstructions.

Model	Objective	Architecture	$\|z\|$	Train. SSE	Valid. SSE
AE	AE	MLP	2	27.00	31.68
VAE-2	VAE	MLP	2	31.47	32.83
VAE-5	VAE	MLP	5	25.32	25.63
VAE-10	VAE	MLP	10	24.32	24.32
VAE-CNN	VAE	CNN-MLP	10	19.24	19.55
CVAE	CVAE	MLP	10	22.82	22.96
VAE-50	VAE	MLP	50	24.37	24.41

Table 1: Details about models and their corresponding training and validation SSE errors. The fourth column $|z|$ is the latent dimension.

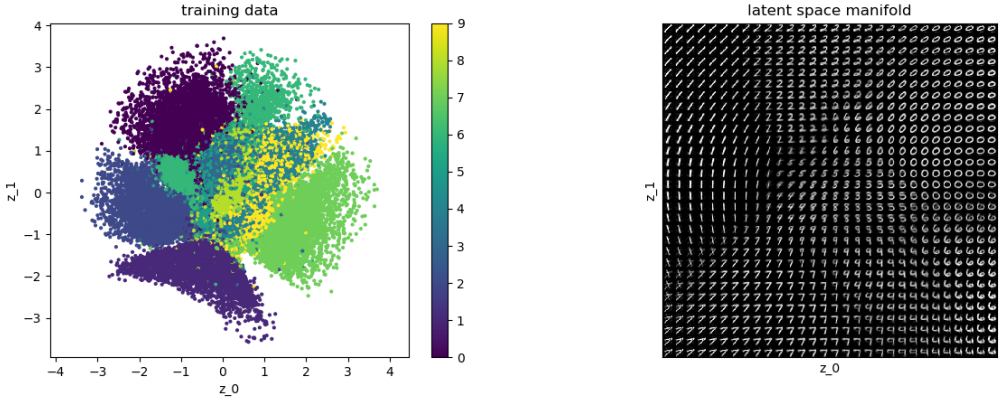


Figure 6: Two-dimensional VAE. The left figure shows the latent space, more specifically the mean vectors $\mu(x)$ of the encoder. The color of a point indicates its class, it can be seen that those observations from same class are clustered together. The right figure shows the learned z manifold, i.e. the decoder $p_\theta(x|z)$ as a function of z.

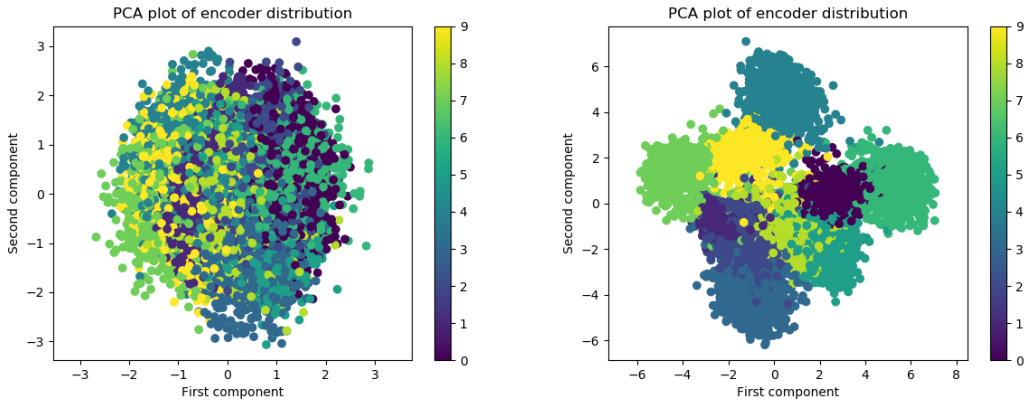


Figure 7: First two PCA components of the 10-dimensional latent spaces of the VAE (left) and CVAE (right). The color indicates class belongings. The CVAE has better class separation.

5.2 Image Transfer

Now we move on to consider an image transfer problem. We are interested in how a CVAE model performs on the task of restoring corrupted images. For this investigation we consider pairs of images (x, y) where x is the non-corrupted image we want to predict based on the corrupted image y. For each image in the MNIST dataset, we corrupt the image randomly by inverting each pixel black $\leftrightarrow$ white with probability 0.5. Each time an image is loaded during training, we alter it randomly this way. Thus we assure the model cannot overfit too much to the data. The goal of training is to learn the model to restore the clean image given the corrupted one. Both a CVAE model implemented with skip connections using ELBO (12) and a model without (using (13)) are considered. Regarding architectures we use the CNN network (see Table 13) for the encoder, and a mirrored one for the decoder. The prior is implemented in the same way as the encoder network. For the first alternative, we use three skip connections (from each output of the convolutional layers) from the prior network into the decoder.

As reconstruction loss function the L_1 error (8) is used. Both models are trained with $\beta = 1.0$ for the KL-divergence term. After 200 epochs of training the results are evaluated. For each image in the validation set we corrupt it 10 times and make a prediction. The average L_1 losses are 17.78 for the model with skip connections and 36.56 for the model without. This is a large difference, the skip connections seem to help out. In Figure 8 a resulting prediction based on the best model is shown.

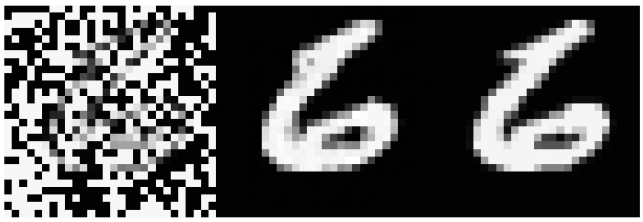


Figure 8: Results of CVAE model using skip connections, on corruption task. Order is 1) input, 2) prediction, 3) ground truth.

6 Method

The main problem we consider is to learn a model to:

> (main): predict a bright/denoised color image in sRGB space x based on a dark RAW input y

To solve this problem we propose to use a CVAE as presented in the theory section. This CVAE will be implemented as a U-Net architecture with a bottleneck and where the main computational blocks being of CNN type (i.e. convolutional). Two different base network architectures are considered: *SID-net* and *Pix2Pix* (soon to be described). As alternative problems we also consider problems where:

> (1): the output x is a grayscale image instead of a color image
>
> (2): the dark input y is a sRGB image instead of a RAW image

The first task should be easier than the main task due to the network does not have to learn color (only 1/3 times as much information to learn). The second task might be harder than the main task, due to less information stored in the dark sRGB compared with the dark RAW. Although, the input image is already in the correct color space in this case. [2] found that using a RAW over sRGB input improved the predictions a lot. In implementation the only difference between these three problems is the number input- and/or output-channels.

For the sake of comparison with a CVAE we will also implement a regular U-Net model, trained with only reconstruction loss, with a corresponding architecture to the CVAE. Models implemented as a CVAE will have a Gaussian factorized prior distribution on the latent space z. To model the y dependence in the decoder $p_\theta(x|y,z)$ the skip connections in the U-Net architecture are used. During training the output of the decoder $p_\theta(x|y,z)$, i.e. its mean parameter $\mu_\theta(y,z)$, is compared with the ground truth x in terms of a loss function, thus we have a supervised model.

Inputs to the models are (amplified) image patches of size $S \times S$ where $S = 256$. Small enough to encourage the network to focus on local image features and big enough to reveal a bit of the context of the scene (surrounding lighting conditions). The original work by [2] let the prediction have double spatial size (in each dimension) of the input[3]. But we are instead letting the output be of the same size as the input. The reasons we choose to use the same size is two-fold: mainly all of the images in the dataset need to fit memory (loading of images is a bottleneck in terms of CPU-time), but also to be able to implement a CVAE with the same architecture for the encoder and the prior network (except the number of input channels). So, both the x and y patches have spatial resolution 256×256.

Following the same approach as [2] the pipeline just to be described is used during training. Given an image from the dataset, the image is first randomly cropped to the patch size S. Then for the purpose of data augmentation, the patch is randomly flipped and transposed, with probability 0.5 to flip/transpose along each of the horizontal and vertical axes. This patch is then the input to the network. The output from the network is compared with the corresponding ground truth bright sRGB patch, which has been cropped/flipped/transposed the same way as the dark RAW. The input RAW image is amplified with the ratio between the exposure time of the corresponding ground truth and that of the dark RAW just before entering the network.

[3] They considered a version of *SID-dataset* where the ground truth RGB images are double the size of the inputs. Whereas we use ground truths of the same spatial resolution as the packed RAW input: 1424×2128.

Given a packed 4-channeled dark RAW image in its original format it is pre-processed in in the following way before being fed as an input to the network:

1. subtract the black level and normalize to range $[0, 1]$
2. amplify the image by the ratio and clip the signal at pixel-value 1
3. randomly crop to the patch size 256×256
4. flip/transpose the patch with probability 0.5

One thing to note is that the output of the network in question is not necessarily in the correct range ($[0, 1]$), depending on the activation function at the output layer. Thus the pixel values of the output are clamped to this range. Regarding the dataset, all models are trained on all the different ratios (100, 250, 300) simultaneously. The neural networks are all implemented in `PyTorch`, and trained on a computer with an Nvidia Tesla-V100 GPU. In the coming section 6.1 about the network architectures we assume the input is a 4-channeled RAW, and the output a 3-channeled RGB image.

After training, the models are evaluated on the full-scale images (using interpolation) of the validation set, this to find out the best hyper-parameters. The performance is evaluated based on the measures PSNR, SSIM and MAE. To get an unbiased estimate of the true performance, the best model is in the end evaluated on the test set.

6.1 Network Architectures

Given a dark input image (patch) y the goal of training a model F is to learn it to predict the corresponding ground truth bright image (patch) x based on y as input. If we let $F(y) = \hat{x}$ be the prediction (output of decoder), the purpose of training is to minimize a loss $L(x, \hat{x})$ (possibly together with additional loss terms on the bottleneck). As baseline we implement F as a variant of a U-Net neural network, which as baseline is trained as a CVAE with the ELBO objective (12). Additionally, alternative ways of training this model are considered: as a hybrid using objective function (15), and as a model with only reconstruction loss. As reconstruction loss (for all models) the L_1 loss function (8) is used, i.e. the summed (over pixels) absolute difference between the ground truth and the prediction. We choose this loss function due to [2] found this it to give rise to better results than the standard L_2 loss.

In [2] a U-Net architecture taking patches of spatial size 512×512 was used. The encoder part of the network is downsampling the 512×512 input to a bottleneck of spatial size 32×32 with 512 channels. This network is due to the size of the bottleneck not really suited for a VAE/CVAE model, we want a low dimensional embedding of the input. Also the factorized Gaussian assumption is not going in hand with a variable in form of an image (with remaining correlations). There are two common approaches to deal with this: either to 1) downsample the input all the way to an 1×1 (or some other very small size) bottleneck, or to 2) use fully connected layers. In the works [3], [9] the second option is used, whereas in the work [6] the first option is used.

We consider two different base architectures. Both of them are U-Net architectures with a bottleneck with 512 dimensions and mainly have CNN computational blocks. Both of them are based on existing architectures from other works, but neither of them was designed to be used in a CVAE setup. For this reason the original networks are modified: one more downsampling network is added and the encoder is modified to output two variables: a mean and variance. We use the same architecture for both downsampling networks. The first architecture is based on the architecture used in [2], we call this network *SID-net* from now on. Our version uses fully connected layers leading from and to the bottleneck. The second considered architecture is called *Pix2Pix* and was introduced by [8]. It was designed to be used as a generative network in a GAN setup and it is uses convolutions all the way down to 1×1 spatial resolution.

6.1.1 CVAE Model

Regarding the CVAE model, we consider a model with Gaussian factorized distributions on the bottleneck as described in the Theory section. For the decoder output we use

$$p_\theta(x|y,z) = \prod_i \text{Laplace}\big(\mu_i(z,y), 1\big).$$

This is a Laplace distribution (with scale parameter 1) independent over pixels. It is implicitly set by the L_1 loss function. We use an encoder given by $q_\phi(z|x)$, not depending on y. For our main problem with x and y being a sRGB and RAW image respectively it does not make sense to use an encoder like that due to the images being in different color spaces and having different noise levels.

Using a prior network $p_\theta(z|y)$ with skip connections to the decoder $p_\theta(x|y,z)$ together these two networks form a U-Net architecture, i.e. the function $F(y)$. To clarify, the skip connections forward signals from the features in the prior network $p_\theta(z|y)$ to the decoder network $p_\theta(x|y,z)$. Thus they are modelling the y dependence of the decoder. The idea of using a U-Net architecture between the prior and the decoder is borrowed from the works of [9], [3] and [6]. At training time the CVAE model has an encoder network $q_\phi(z|x)$ additional to the U-Net architecture. At testing time the encoder is not longer used, only the U-Net is left. In implementation the CVAE network differs from that of a model (U-Net) trained with only reconstruction loss, mainly due to a CVAE has two downsampling networks during training.

To not limit the ranges of possible values we avoid using activations on the bottleneck, this since a Gaussian distribution has unlimited support. The prior network and the encoder are implemented exactly the same, except the prior is taking a four-channeled input whereas the encoder is taking a three-channeled input. Another difference is that the prior only outputs a mean vector, whereas the encoder outputs a variance (or rather a log-variance in implementation) as well. As an alternative, a CVAE model without skip connections is also considered. Except the fact this model has no shortcuts avoiding the bottleneck, it is the exactly the same as the baseline CVAE model. This model is trained using the ELBO given by (13).

6.1.2 Hybrid Model

The baseline CVAE model is trained with an encoder network $q_\phi(z|x)$ together with the prior network $p_\theta(z|y)$. During training z is sampled from $q_\phi(z|x)$ and passed through the decoder, but at evaluation we are instead using the $p_\theta(z|y)$ network to obtain the bottleneck z. Thus, it is important that the model has been trained such that:

- $q_\phi(z|x)$ is producing samples which decode to low reconstructions (errors), and at the same time
- the gap (e.g. the KL-divergence) between the distributions $q_\phi(z|x)$ and $p_\theta(z|y)$ is small

To investigate the efficiency of during training focusing most on samples from $q_\phi(z|x)$, instead of most from $p_\theta(z|y)$, and the balance between the two distributions we will train hybrid models. When the weight of the L_{GSSN} term in (15) $\alpha = 1.0$ we get the regular CVAE back which during training only samples from $q_\phi(z|x)$, and when $\alpha = 0$ the samples are all from $p_\theta(z|y)$ instead and the model is acting more like a deterministic model (no KL-divergence). In-between these two values the optimization is focusing more or less on one of the distributions. The hybrid models have the exact same distributions/networks as the baseline CVAE, the difference is only in the weight between the L_{CVAE} and L_{GSNN} terms in (15). And for the hybrid model two passes through the decoder ($L = 1$ Monte Carlo samples from each of the distributions) are demanded during training. The L_1 loss is used for both outputs of the decoder.

6.1.3 Reconstruction-loss Model

If we set $\alpha = 0$ in the hybrid loss function we get simply the L_{GSNN} loss. This loss function implies there is no encoder network $q_\phi(z|x)$ anymore, and thus no KL-divergence either. Also if $p_\theta(z|y)$ is set to have zero-covariance according to $p_\theta(z|y) = \mathcal{N}(\mu_\theta(y), 0)$ we get a deterministic model. This model is a regular U-Net trained with only reconstruction loss (L_1), and without any reparameterization. It has the same architecture at training and testing stages, i.e. only the U-Net part. Since this model does not have an encoder network it is the fastest to train.

6.1.4 SID-net

The network used in [2] is a regular U-Net network, i.e. an AE network with skip connections between the downsampling (encoder[4]) and the upsampling (decoder) networks. They also considered the SID-dataset, but another spatial resolution than we do. They take input patches of spatial resolution 512×512, and let the decoder predict an sRGB image of double that resolution, i.e. 1024×1024. This work is on the other hand focusing on inputs and predictions of the spatial size 256×256.

The downsampling part of the network uses repeating blocks consisting of:

1. a double convolution block
2. a max-pooling block with stride two to downsample

Their encoder is using five of the double convolutions and four max-pooling layers. This architecture is modified to better suite a CVAE model. We are doing two main changes:

1. one more max-pooling layer is added
2. after this pooling a fully connected layer is used

The main reason we add a pooling is to keep the network size reasonably low. The decoder part of the network is using the same type of convolutional blocks, followed by an upsampling block based on stride 2 transposed convolutions. We modify their decoder network to output a variable with same spatial size as the input (to the downsampling network). For an RGB image the output has three channels. Since we modify their original encoder network we have to modify the original decoder network accordingly, by adding symmetrical fully connected layers / convolutions to the first part of the decoder. Furthermore, one more skip connection is added.

All convolutional layers are followed by a LeakyReLU activation (of all neurons) with negative slope 0.2. The convolutions are all using kernel size 3 and stride 1, whereas for the upsampling transposed convolutions are using kernel size 2 and stride 2. Zero-padding is applied to all convolutions such that the spatial size of the output is the same as the input (or times 2 in case of transposed convolutions).

Now we are introducing some notations for the operations used in the network. Starting with a convolutional layer, we let $C_{K,S}(I, O)$ be a convolution with kernel size K and stride S accepting I input channels and outputting O output channels. A downsampling operation in form of a max-pooling with stride s is expressed as $d(s)$. Together these operations define the encoder (downsampling) network. A double convolution block $B_{K,S}(I, O)$ consists of two convolutions each followed by an activation according to:

$$
\begin{aligned}
& B_{K,S}(I, O) = \\
1) \quad & C_{K,S}(I, O) \\
2) \quad & LeakyReLU(0.2) \\
3) \quad & C_{K,S}(O, O) \\
4) \quad & LeakyReLU(0.2)
\end{aligned}
$$

[4]The word 'encoder' does not necessarily refer to the encoder $q_\phi(z|x)$ in the CVAE, it instead refers to the downsampling part of the network.

We use a fully connected layer as the very last part of the downsampling network. The input is first flattened before passing through this part. The notation $fc(D_1, D_2)$ is used for a fully connected layer which takes a D_1 dimensional input and outputs a D_2 dimensional variable. The dimension D_1 depends on the spatial resolution of the input of course. Using these notations our downsampling network can be expressed as:

$$
\begin{aligned}
&B_{3,1}(4, 32) \rightarrow d(2) \rightarrow \\
&B_{3,1}(32, 64) \rightarrow d(2) \rightarrow \\
&B_{3,1}(64, 128) \rightarrow d(2) \rightarrow \\
&B_{3,1}(128, 256) \rightarrow d(2) \rightarrow \\
&B_{3,1}(256, 512) \rightarrow d(2) \rightarrow \\
&fc(32768, 512)
\end{aligned}
$$

The outputs of each of the five double blocks are skipped into the decoder. The decoder is using upsampling in form of transposed convolutions. We let such a layer be expressed as $CT_{K,S}(I, O)$ in same way as the a convolutional layer. The decoder network has skip connections as inputs, these are concatenated with the downstream decoder features at the corresponding level. A concatenation of two feature vectors with same spatial dimension, and C_1 and C_2 channels respectively, along the channel axis is expressed as $Con(C_1, C_2)$. For the very first part of the decoder, i.e. the layer taking z as input, we use a fully connected layer taking 512 input features and outputting 32768 features. Our decoder[5] network is given by:

$$
\begin{aligned}
&fc(512, 32768) \rightarrow \\
&CT_{2,2}(512, 512) \rightarrow Con(512, 512) \rightarrow B_{3,1}(1024, 512) \rightarrow \\
&CT_{2,2}(512, 256) \rightarrow Con(256, 256) \rightarrow B_{3,1}(512, 256) \rightarrow \\
&CT_{2,2}(256, 128) \rightarrow Con(128, 128) \rightarrow B_{3,1}(256, 128) \rightarrow \\
&CT_{2,2}(128, 64) \rightarrow Con(64, 64) \rightarrow B_{3,1}(128, 64) \rightarrow \\
&CT_{2,2}(64, 32) \rightarrow Con(32, 32) \rightarrow B_{3,1}(64, 32) \rightarrow \\
&C_{1,1}(32, 3)
\end{aligned}
$$

To use this network architecture in a CVAE setup with the factors $p_\theta(z|y)$, $q_\phi(z|x)$ and $p_\theta(x|y, z)$ we let the prior network $p_\theta(z|y)$ have the downsampling architecture described above and the decoder network $p_\theta(x|y, z)$ the upsampling architecture. Thus the prior and decoder networks together form a U-Net architecture. The encoder network $q_\phi(z|x)$ is implemented the same as the prior network, except it does not output any skip connections and the number of input channels is three instead of four. The prior network does only output a mean vector, whereas the encoder network outputs two vectors: a mean and a variance according to:

$$
\cdots \rightarrow \begin{cases} fc(32768, 512) \\ fc(32768, 512) \end{cases}
$$

A visualization of the *SID-net*, implemented as an CVAE is shown in Figure 9. In Figure 10 a visualization of the architecture at the testing stage is shown, this is also how the model trained with only reconstruction loss looks like.

Two alternative variants of the *SID-net* are also considered, explanations of the differences between them and the baseline are listed below in Table 2. *SID-net no-skip* is as the name suggests a variant with no skip connections at all, thus there are no concatenation in the decoder, and there are fewer input channels to the double convolution blocks in the decoder. *SID-net small* is implemented to be a faster and smaller version of the baseline *SID-net*, it is first of all using a larger bottleneck of size 768, but the main difference is in the number of filters of the

[5]The last layer of our decoder is different from the one in [2]. It is modified to suite the spatial size we consider.

convolutions. All convolutions (except the inputs and output) have half the amount of filters compared to the baseline. Due to another latent dimension and different amount of channels the fc blocks are changed, 512 is replaced with the alternative latent dimension: 768, and 32768 is replaced with 16384.

Alternative	Explanation
SID-net no-skip	No skip connections at all.
SID-net small	Half the amount of filters and larger bottleneck of size 768.

Table 2: Alternative variants of the *SID-net*. Explanations of differences compared to the baseline.

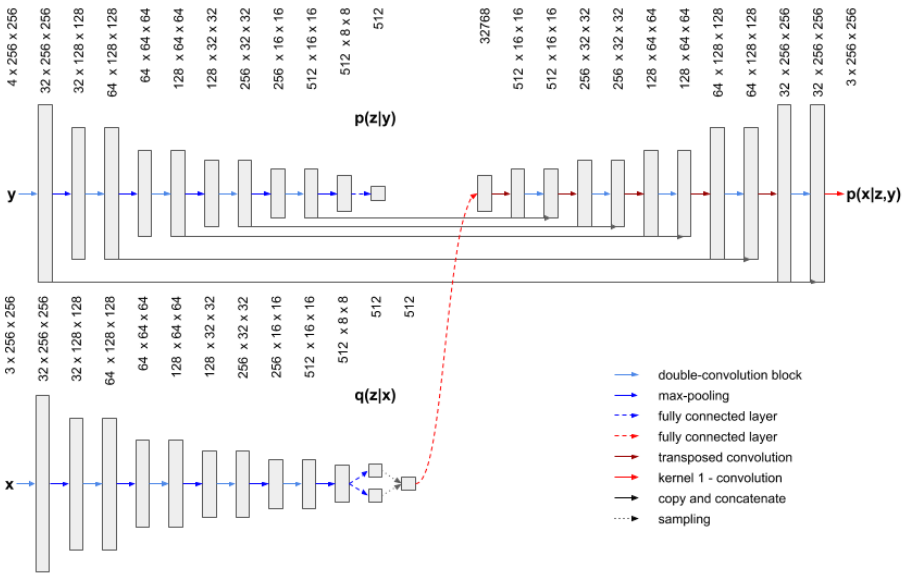


Figure 9: The *SID-net* as a CVAE, at training time. The downsampling network in top-left is the prior network, the one in the bottom is the encoder, in top-right the decoder can be found.

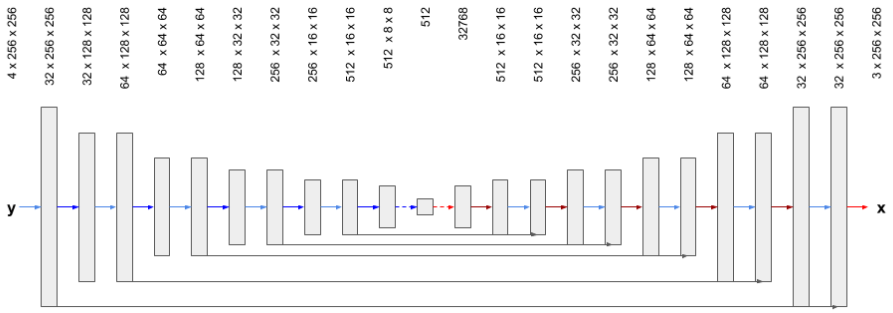


Figure 10: Visualization of *SID-net* as a regular U-Net (the model trained with only reconstruction loss). The testing time architecture of the CVAE variant looks like this too.

6.1.5 Pix2Pix

The *Pix2Pix* network architecture was introduced by [8]. It was originally used as the generator network of a GAN architecture. This network was successfully applied to several image translation problems based on RGB images. Even though it was used in a GAN setup this network perfectly fits our needs, it is a U-Net architecture with a downsampling encoder and upsampling decoder. Given an input of size 256×256 the bottleneck is of spatial dimension 1×1, and has 512 channels. There is not a lot of things we have to change to turn the architecture into a CVAE. All the convolutional operations in the network use kernel size 4. The encoder network uses convolutions with stride 2 to downsample the input. The decoder uses transposed convolutions to upsample, also with stride 2. Dropout with probability 0.5 is applied in the decoder, this dropout is turned off at evaluation. The encoder uses LeakyReLU activations with negative slope 0.2 whereas the decoder uses regular ReLU activations.

One difference between this network architecture and the *SID-net* is the number of skip connections, this network uses seven whereas the other is uses five. We modify the original *Pix2Pix* network a bit to suite our needs, but the main building blocks are the same. In the original architecture most weight blocks are followed by a batch normalization layer, however we choose to not use any batch normalization, due to we found it to decrease the performance. We keep calling this network *Pix2Pix* even though we modify it.

The encoder building block $E_{K,S}(I, O)$ is given by:

$$
\begin{aligned}
& E_{K,S}(I,O) = \\
1)\quad & C_{K,S}(I,O) \\
2)\quad & LeakyReLU(0.2)
\end{aligned}
$$

The decoder is using a similar block, but here activations are not leaky, and an optional dropout is applied before the activation. We use the notation $D^d_{K,S}(I, O)$ for a block of this type which is using dropout, and ignore the d if dropout is not used. It is explicitly given by:

$$
\begin{aligned}
& D^d_{K,S}(I,O) = \\
1)\quad & CT_{K,S}(I,O) \\
2)\quad & Dropout(0.5) \\
3)\quad & ReLU
\end{aligned}
$$

Using this convention and assuming the input has four channels the encoder network can be expressed as:

$$
\begin{aligned}
& E_{4,2}(4, 64) \rightarrow \\
& E_{4,2}(64, 128) \rightarrow \\
& E_{4,2}(128, 256) \rightarrow \\
& E_{4,2}(256, 512) \rightarrow \\
& E_{4,2}(512, 512) \rightarrow \\
& E_{4,2}(512, 512) \rightarrow \\
& E_{4,2}(512, 512) \rightarrow \\
& E_{4,2}(512, 512)
\end{aligned}
$$

The decoder network takes skip inputs from the encoder which are merged with downstream decoder features along the filter axis. The last layer is mapping the input to the correct number of channels, e.g. 3 for an RGB image. This layer uses Tanh activation instead of ReLU.

Using the convention described above, and replacing the last activation with Tanh, the decoder architecture can be expressed as:

$$
\begin{aligned}
&D^d_{4,2}(512, 512) \rightarrow Con(512, 512) \rightarrow \\
&D^d_{4,2}(1024, 512) \rightarrow Con(512, 512) \rightarrow \\
&D^d_{4,2}(1024, 512) \rightarrow Con(512, 512) \rightarrow \\
&D_{4,2}(1024, 512) \rightarrow Con(512, 512) \rightarrow \\
&D_{4,2}(1024, 256) \rightarrow Con(256, 256) \rightarrow \\
&D_{4,2}(512, 128) \rightarrow Con(128, 128) \rightarrow \\
&D_{4,2}(256, 64) \rightarrow Con(64, 64) \rightarrow \\
&D_{4,2}(128, 3)
\end{aligned}
$$

The prior and encoder networks (in a CVAE) has the same base architectures. The prior network uses the 'encoder' architecture just described. To model the variance in the encoder in a CVAE model we add one more output of the downsampling network according to:

$$
\cdots \rightarrow \begin{cases} E_{4,2}(512, 512) \\ E_{4,2}(512, 512) \end{cases}
$$

The *Pix2Pix* architecture as CVAE model is visualized in Figure 11 below.

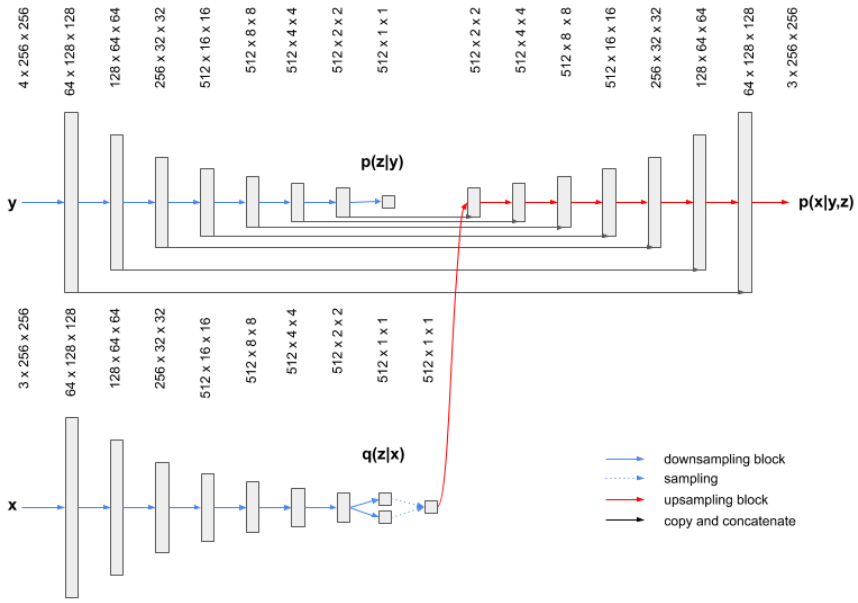


Figure 11: The *Pix2Pix* network architecture as a CVAE, during training. The downsampling network in top-left is the prior network, the one in the bottom is the encoder, in top-right the decoder can be found.

6.1.6 Training Details

For all models we use batch optimization of the model weights using the Adam optimizer with batch size $M = 5$. This optimizer has two momentum parameters: β_1 and β_2, the standard values are used. The weights of the models are initialized randomly from a Gaussian distribution with zero mean and standard deviation 0.02, and the biases are initialized to zeros. The learning rate η is tweaked a bit depending on the model and the data, this for training stability and speed together with performance on the validation set. We are using a learning rate schedule which starts at an initial value which is reduced with a factor γ after some epochs, we call this epoch 'milestone'. The model in question is trained during N epochs until training has converged.

As standard we use $\eta = 10^{-4}$ which is decreased with factor $\gamma = 0.1$ at epoch 'milestone' (learning rate 10^{-5} is used for the remaining epochs). Most models are trained during 4000 epochs, and use a milestone of 2000. If not else stated, we search over the different weights β of the KL-divergence term in the loss function in case of a CVAE model. The values $\beta = 0.1$, 0.5, 1.0, 2.0, 10.0 are considered. In case of a model trained with the hybrid loss (15) there is also a weight α hyperparameter. A summary of the training parameters is found in Table 3.

During training batches of five randomly picked patches (from the whole training set) are used. These are sampled uniformly from the given image. In practice a pixel is sampled, this pixel is then defining the top left corner of the patch. Ignoring pixels along the edges, i.e. the outermost 256 pixels along the frame, the probability that a random patch is covering a certain pixel is $p = 1424 \times 2128/256^2 = 0.021627$. Using 4000 epochs and patch size 256 each pixel is expected to be used as input 86 times. This is of course true for all images in the training set, since in each epoch one patch is randomly sampled from each image.

To compensate for the randomness in training, or rather the fluctuations from the mean weights, at the very end of training, we save the weights from the four last epochs. Thus we get four versions of the same model. Also, if not else mentioned, to compensate for randomness due to the random data, we train each model twice and report the average results over them (in total 8 versions).

Parameter	Value	Explanation
β_1	0.9	Momentum parameter
β_2	0.999	Momentum parameter
η	10^{-4}	Initial learning rate
γ	0.1	Multiplicative factor for learning rate decrease
M	5	Batch size
N	4000	Epochs
milestone	adjusted	What epoch lowering the learning rate
α	adjusted	Weight of L_{GSSN} term in hybrid loss
β	adjusted	Weight of KL-divergence term in ELBO

Table 3: Hyperparameters and their standard values.

6.2 Interpolation

At testing time the goal is to predict a full-size bright image based on a full-size dark image as input. Given a full-size dark image, the following pipeline is used:

1. (extraction) extract patches from the dark input image using a grid
2. (prediction) for each dark patch: predict a bright patch using trained network
3. (interpolation) merge the predicted patches together into a full-size image using interpolation

For extraction a grid of patches, each of size 256×256, spanning the input image is used. Two alternative grids are considered, see Figure 12 and 13. They are constructed in the following way:

1. Dividing the frame into regions based on strides $(s_x, s_y) = (146, 144)$, or
2. Dividing the frame into regions defined by an overlap $f = 22$

The first alternative implies quite some overlap between patches since the strides are about half of the patch size $S = 256$. For pixels in the interior of the image there will be nine patches which cover them, thus a lot of predictions will contribute to these pixels. The strides are chosen such that the last patch (in lower right corner) is neither extending or cutting the frame short, i.e. is exactly aligned with the corner. Due to the different number of pixels per axis we have to use different strides for the x- and y-axis. The second option basically implies there will be much fewer predictions contributing to each pixel, on average 1.16 patches are covering each pixel. Thus computation times will be faster for this option. The possibly negative side with the second approach is that the transitions between the merged patches may not turn out as well even though interpolation algorithm is used. As a baseline we will consider option 1.

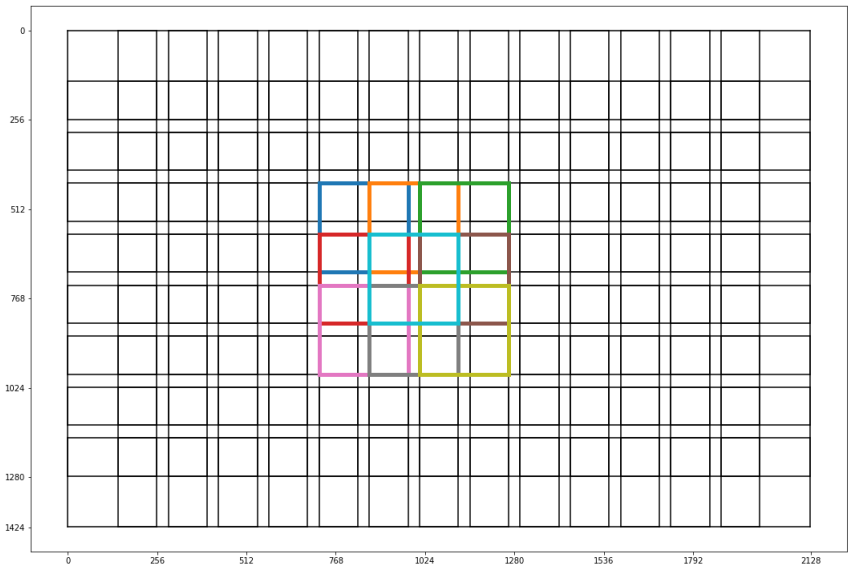


Figure 12: Grid for option 1. Here nine of the patches are highlighted. Remember each patch is of size 256×256.

The extracted patches are fed to the trained model which outputs bright predictions. To merge all these (predicted) patches together an algorithm called *inverse distance interpolation* is used, which now is introduced. Given a region of the image which N predictions/patches $P_1, P_2, \ldots, P_N$ cover. The inverse distance interpolation weighs the contribution from a patch to a pixel $\tilde{x} = (x, y)$ (in the global image coordinates) according the inverse distance between the pixel and the center of the patch.

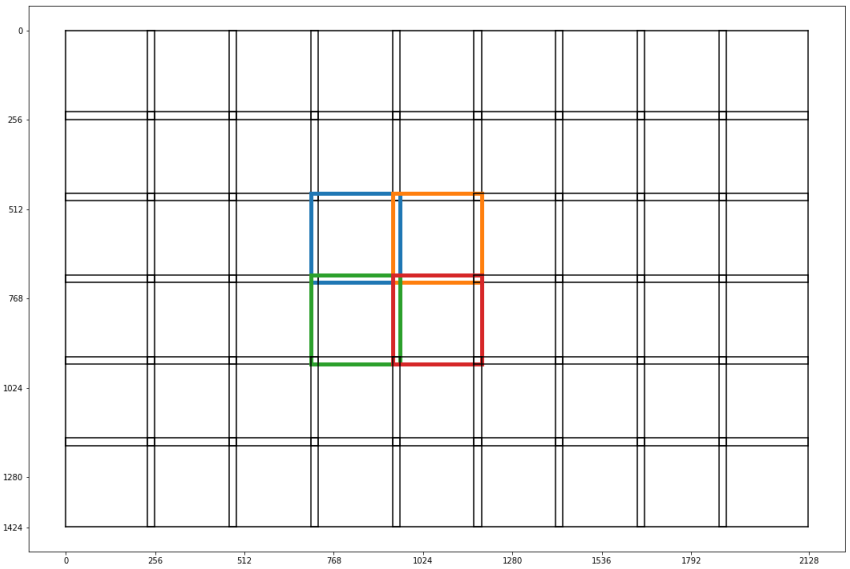


Figure 13: Grid for option 2. Here four of the patches are highlighted.

If the output image is denoted as $I(x, y)$, and the centers are given by $c_1, c_2, \ldots, c_N$ where $c_i = \big(c_i(x), c_i(y)\big)$ then the output is given by

$$I(x,y) = \frac{1}{\sum_{j=1}^{N} d(\tilde{x}, c_j)^{-p}} \sum_{i=1}^{N} P_i(x,y) d(\tilde{x}, c_i)^{-p}, \tag{16}$$

where $d(\cdot, \cdot)$ is the Euclidean distance function. Note that the weights together sum to 1. The factor p determines the importance of short distances. With $p = 0$ we get a simple averaging interpolation. The main difference between $p \geq 1$ and $p = 0$ is that the first takes into account distances whereas the second does not. On the other hand $p = 0$ is faster to run due to no need for using any multiplications. Figure 14 shows a visualization of the interpolation setting between three patches.

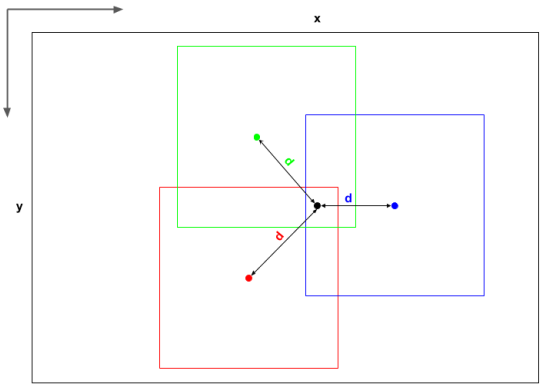


Figure 14: Interpolation on pixel (x, y) between three overlapping patches. Note that the distances are from the centers of the patches. The contribution of the blue patch is the largest since it has its center closest to the pixel.

6.3 Evaluation Measures

When it comes to evaluating the performance of an image enhancing algorithm there are several common measures. Pixelwise measures include MAE, MSE, and PSNR. PSNR is commonly used to measure the performance of a denoising algorithm. SSIM is a measure which takes into account correlations. Here we suppose I_1 and I_2 are two images that are to be compared. They are assumed to be of same shape $C \times H \times W$, where C is the number of channels, H the number of vertical pixels and W the number of horizontal pixels.

6.3.1 MAE - Mean Absolute Error

The mean absolute error (MAE) between two images: I_1 and I_2 is given by

$$MAE(I_1, I_2) = \frac{1}{CHW} \sum_{i=0}^{C-1} \sum_{j=0}^{H-1} \sum_{k=0}^{W-1} \big| I_1(i,j,k) - I_2(i,j,k) \big|.$$

6.3.2 PSNR - Peak Signal to Noise Ratio

The mean squared error (MSE) between I_1 and I_2 is given by

$$MSE(I_1, I_2) = \frac{1}{CHW} \sum_{i=0}^{C-1} \sum_{j=0}^{H-1} \sum_{k=0}^{W-1} \big(I_1(i,j,k) - I_2(i,j,k) \big)^2. \tag{17}$$

Peak Signal to Noise Ratio (PSNR) is a measure of difference between two signals, for example between two images. Given a noise-free ground truth image, and a corresponding estimate of the ground truth, PSNR is the ratio between the maximum possible power of the noise-free image and the power of the corrupting noise, assuming the noise is white, additive and Gaussian distributed. Given the mean squared error (MSE), the PSNR is given by the formula

$$PSNR(I_1, I_2) = 10 \cdot \log_{10} \big(\frac{MAX_I^2}{MSE(I_1, I_2)} \big). \tag{18}$$

In case of a normalized image $MAX_I = 1.0$.

6.3.3 SSIM - Structural Similarity Index Measure

SSIM takes is a measure which takes into account the dependence between pixels. The measure is composed of three components: l the luminance, c the contrast and s the structure. Given the images x and y, the SSIM is given by

$$\begin{aligned} SSIM(x,y) &= [l(x,y]^\alpha \cdot [c(x,y]^\beta \cdot [s(x,y]^\gamma, \\ l(x,y) &= \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1}, \\ c(x,y) &= \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2}, \\ s(x,y) &= \frac{\sigma_{xy} + c_3}{\sigma_x\sigma_y + c_3}. \end{aligned} \tag{19}$$

The terms in this formula are:

μ_x	the average of x
μ_y	the average of y
σ_x	the standard deviation of x
σ_y	the standard deviation of y
σ_{xy}	the covariance between x and y

The terms c_1, c_2 and c_3 are regularizing constants, used to stabilize the numerical behaviour. The standard choice of c_3 is $c_3 = c_2/2$. The power constants in the weighted combination are usually set to $\alpha = \beta = \gamma = 1$, with these values the SSIM simplifies to

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_x\sigma_y + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}. \tag{20}$$

The remaining two constants are set to

$$c_1 = (0.01L)^2$$

and

$$c_2 = (0.03L)^2,$$

where L is the dynamic range of the image.

The SSIM can take values between -1 and 1, where a value of 1 means perfect similarity. SSIM can be calculated using a sliding window, e.g. by convolving the images by a Gaussian kernel or an uniform kernel. If the images have several channels, the SSIM is calculated independently per channel and then averaged. For images with pixels represented as floats with values in range $[0, 1]$ the dynamic range is given by $L = 1$. As in the original article introducing SSIM [20], for the sliding window we use a kernel of size 11×11 with Gaussian weights (standard deviation 1.5). The SSIM implementation in the Python library `scikit-image`[6] is used for calculations.

6.3.4 Jensen-Shannon divergence

The Jensen-Shannon divergence between two distributions p_1 and p_2 is given by

$$JSD(p_1||p_2) = \frac{1}{2}D_{KL}\big(p_1||M\big) + \frac{1}{2}D_{KL}\big(p_2||M\big), \tag{21}$$

where

$$M = \frac{1}{2}(p_1 + p_2).$$

We use this symmetric measure to compare two normalized image histograms (p_1 and p_2), where each image is discreatized to the range $[0, 255]$, thus in this case each histogram is a discrete distribution. The KL-divergence between two discrete distributions is given by

$$D_{KL}\big(p_1||p_2\big) = \sum_i p_{1,i} \log \frac{p_{1,i}}{p_{2,i}}. \tag{22}$$

In log-base 2, we have $0 \leq JSD(p_1||p_2) \leq 1$.

[6] `https://scikit-image.org/docs/dev/api/skimage.metrics.html#skimage.metrics.structural_similarity`

7 Results

The Results section is divided in the following way. We start with considering the main problem of predicting a bright sRGB based on a dark input in RAW format, and compare different network architectures and training losses. Next after this we are doing a detailed comparison of different interpolation strategies. After which the results based on the alternative tasks are presented.

As baseline a CVAE model based on the *SID-net* architecture as explained in the Method is used. Regarding extraction and interpolation we use as baseline

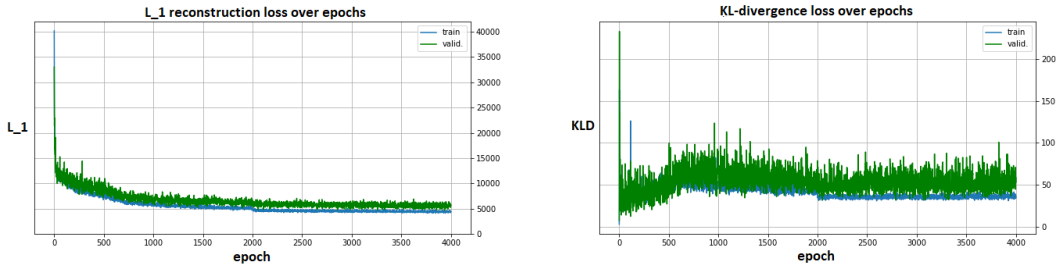
- strategy 1 for extracting and merging patches
- $p = 2$ in equation (16) for spatial interpolation

All the dark input images which are shown in this section are amplified with the exposure ratio. Originally the input images (to the pipeline) are actually much darker, but to be able to see details amplified versions are shown instead.

7.1 Main Problem

Based on evaluation of the full-scale images in validation set and using our interpolation algorithm we found that $\beta = 2.0$ worked the best for *SID-net*. The model is trained with milestone at epoch 2000. Loss curves over the training set and validation set can be seen in Figure 15. Table 4 shows the evaluation results on the test set. We present the results based on versions of the weights from the epochs: 1000, 2000 and the last: 4000, all of these are based on the results after merging the patches with the baseline interpolation strategy. In this table we are also listing the results for 'Patches', this corresponds to the results on the extracted patches alone (without merging the results) based on the epoch 4000-version. Obviously the results are getting better with increased epoch, however the difference in the results between the last epoch and epoch 2000 is not very large, which also can be observed in the loss-curve. There is not a big difference between the results based on interpolation versus not using interpolation, except in PSNR where there is quite a difference.

Breaking the results into the three exposure ratios: 100, 250 and 300 we get the results listed in Table 5. Not to our surprise the model performs best on ratio 100 and worst on ratio 300. Although, in MAE the model performs better on ratio 300 than on ratio 250. In Figure 16 it can be seen that the ISO of the input, i.e. the noise level, is a stronger indicator of the prediction quality than the exposure time (in terms of the ratio) of the input.



Reconstruction loss. KL-divergence loss.

Figure 15: Loss curves for the baseline *SID-net* CVAE architecture, the reconstruction loss is L_1. The losses are averaged over batches of size 5. The blue curves are training losses whereas the green ones are the validation losses.

Model	Epoch/setting	PSNR	SSIM	MAE
SID-net	1000	26.8296	0.6750	0.03655
	2000	27.9252	0.7021	0.03247
	4000	28.3697 ± 0.0502	0.7093 ± 0.0010	0.03080 ± 0.00016
	Patches	28.9384	0.7068	0.03136

Table 4: Results on the test set for the baseline CVAE model based on the *SID-net* architecture. The higher PSNR the better, the higher SSIM the better, and the lower the MAE the better. The standard deviation for the last epoch (4000) is calculated over 8 versions (of weights) of the same model (4 last epochs $\times$ 2 training sessions). The 'Patches' row corresponds to evaluation over the extracted patches without merging.

Ratio	PSNR	SSIM	MAE
100	29.3662	0.7354	0.02835
250	28.0806	0.7132	0.03248
300	27.7195	0.6824	0.03159

Table 5: Test results for the baseline CVAE architecture. Here the results are shown separately per exposure ratio whereas in Table 4 the results are over the full dataset. In general the darker input images (ratio 250 and 300) are predicted a bit worse than the brighter ones.

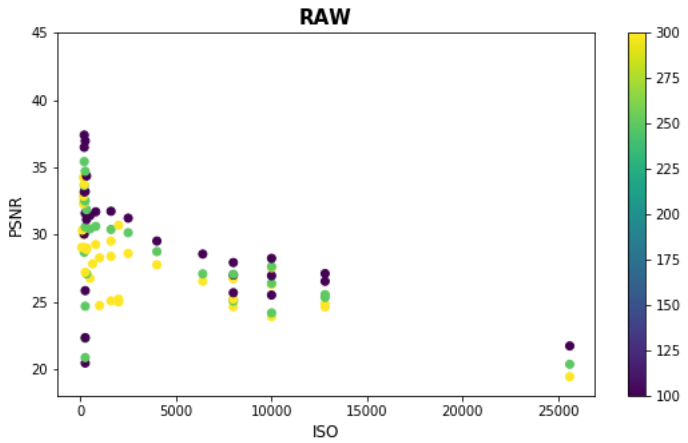


Figure 16: ISO versus PSNR per prediction of *SID-net* (versus ground truth) on the test set for the main task of predicting a color image based on RAW input. The color indicates what ratio each sample has (one of 100, 250, 300).

Now we are moving on to inspect the outputs of the model. Regarding the full-scale predictions, we are showing downsampled images thus it can be hard to note all the details. In Figure 17 we show four examples of predictions and the corresponding input and ground truth, this is the result after merging the patches together. The first three inputs are 'easy' inputs, due to them being relatively noise-free, the last input is harder due to the high noise level in the foreground. At this scale the outputs do not look too different from the ground truths, exposures and contrasts are good, and overall the colors are about right (except a bit low saturation). We cannot notice any type of blocking effects: the interpolation is working, and the noise is all gone. Overall, for inputs which are not too noisy the outputs look good with little artifacts. What the network is struggling with is mainly the extremely noisy inputs, e.g. the dark foreground of example four. In Figure 18 we have zoomed in on details in each prediction.

Figure 17: Full-sized predictions of the baseline CVAE *SID-net*, together with corresponding ground truth. The order is 1) input RAW (amplified), 2) *SID-net* prediction, 3) ground truth. The first input has ISO 2000, so has the second one. The third one has ISO 320, and the last one 1600.

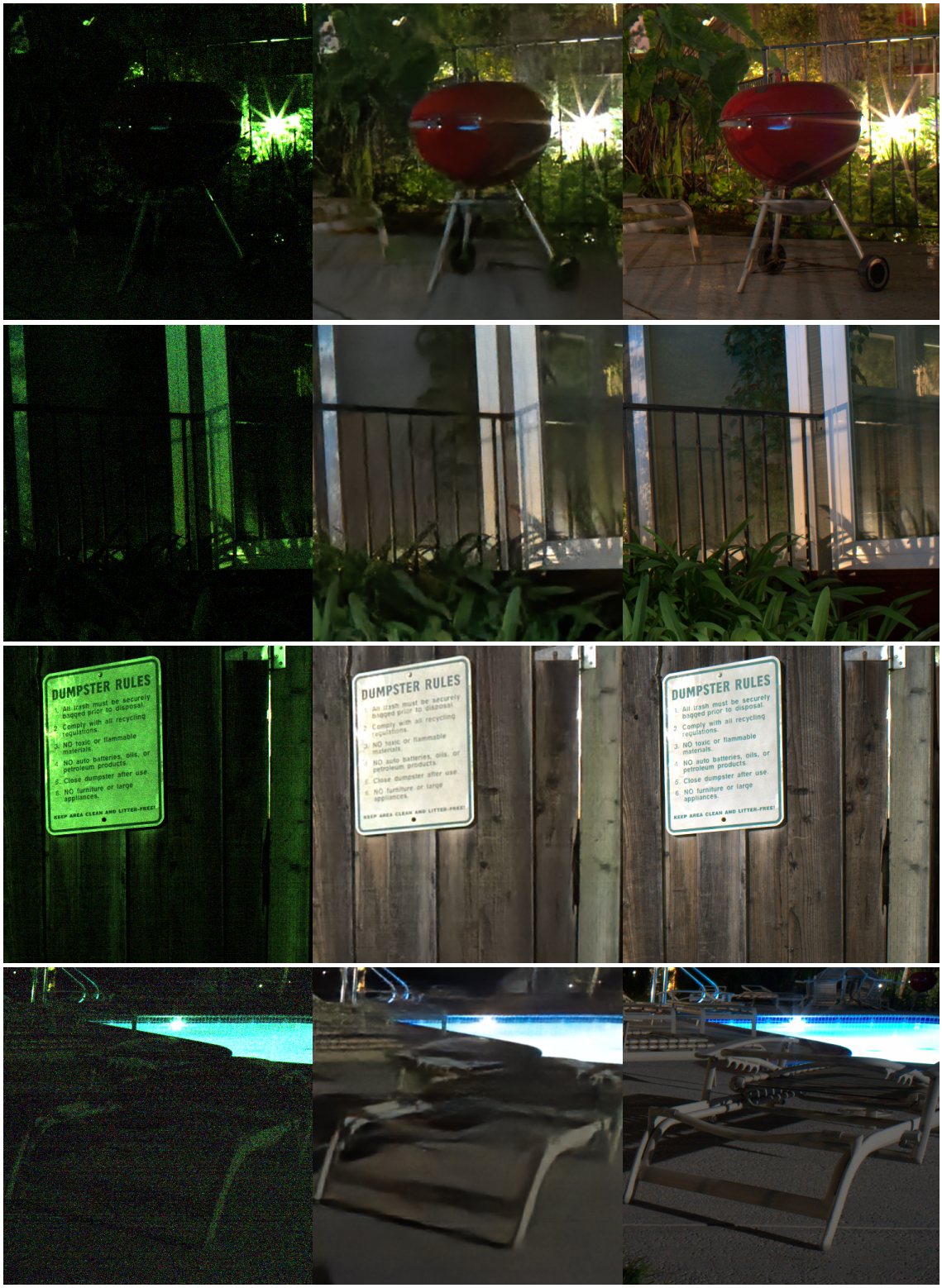


Figure 18: Details. Order is: 1) input RAW (amplified), 2) *SID-net* prediction, 3) ground truth. These are crops of size 512×512 from the merged full-sized predictions.

7.1.1 Alternative Training

We are now going to investigate how training (the U-Net part of) the *SID-net* architecture with different loss functions affects the results. The baseline models was trained with the regular CVAE ELBO. As explained in the Method, a hybrid model with $\alpha = 0$ (together with zero-covariance) corresponds to a U-Net with only reconstruction loss. This architecture has only one downsampling network during training, see Figure 10. We call this model 'L_1'. Furthermore we consider hybrids with the weights $\alpha = 0.5$, 0.9. Also a CVAE with $L = 5$ instead of only $L = 1$ Monte Carlo samples in equation (11) is trained. These four models are each trained during 4000 epochs using the same learning rate schedule as the regular CVAE, i.e. reducing it from 10^{-4} to 10^{-5} at epoch 2000. The loss curve for the L_1 model can be seen in Figure 19. Note that the level at convergence is higher than that of the baseline CVAE model, see figure 15, the relatively large difference is mainly due to the training loss for the CVAE is based on samples from $q_\phi(z|x)$ (which is producing much better samples than $p_\theta(z|y)$).

Evaluating the results on full-scale images using interpolation we get the results listed in Table 6. In this table the results for the baseline CVAE are listed again. We ignore the hybrid models for now and compare the CVAE with the L_1 model. First of all the mean values are very close to each other, although the CVAE performs the best in all the measures. But we have to consider the standard deviations too. In PSNR and SSIM the CVAE seems to be significantly better than the L_1 model, but in MAE there is quite some overlap between the two. Regarding the hybrid models, in mean values the hybrid $\alpha = 0.9$ is the better, it is performing about the same as the CVAE, whereas the hybrid $\alpha = 0.5$ is closer to the L_1 model. Overall there is barely any difference in the results between different loss functions, but it seems like as α decreases the reconstructions are getting a little bit worse. Regarding the CVAE model trained with $L = 5$ samples, it is in average performing a bit better than the regular CVAE in PSNR and SSIM, but a bit worse in MAE. However there is overlap between the results in all three measures.

Loss function	PSNR	SSIM	MAE
CVAE	28.3697 ± 0.050	0.7093 ± 0.0010	0.03080 ± 0.00016
hybrid $\alpha = 0.9$	28.2769 ± 0.042	0.7086 ± 0.0007	0.03103 ± 0.00015
hybrid $\alpha = 0.5$	28.1836 ± 0.055	0.7042 ± 0.0009	0.03127 ± 0.00027
L_1	28.1469 ± 0.029	0.6999 ± 0.0007	0.03124 ± 0.00013
CVAE $L = 5$	28.3732 ± 0.039	0.7099 ± 0.0005	0.03083 ± 0.00017

Table 6: Results on the test set for *SID-net* trained with different loss/objective-functions. The averages and standard deviations are over two training sessions. The L_1 row corresponds to a model (*SID-net*) with only reconstruction loss. The first row 'CVAE' is the baseline model, these are the same numbers as those in Table 4 row '4000'.

A comparison of a few predictions of each model, except the CVAE $L = 5$ model, can be seen in Figure 20. We notice that the CVAE and $\alpha = 0.9$ model predicts similarly, the same goes for $\alpha = 0.5$ and the L_1 model. Overall the differences between the four are minor. There are some minor differences in colors / tones between the models, in the first example it can be seen in the water, and in the second example in the leaves. Regarding the third example the input is very noisy, all the models struggle but the CVAE and the hybrid $\alpha = 0.9$ are doing a bit better job than the other two. There are also differences in the amount of structure in the uniform gray/white areas. In Figure 21 we are showing a further comparison of full-scale predictions from the CVAE and L_1 model respectively, together with the ground truths. In the first example we can see differences in the bottom corners, the L_1 model predicts too green, the same goes for the second example. In the third example we can see the L_1 model actually finds some of the yellow color of the wall, which the CVAE does not. In the last example both struggle, the CVAE predicts the color of the box a bit better, but on the other hand it is producing green stains on the wall. Overall both the CVAE and L_1 model have some color problems, but the CVAE has a bit less of them.

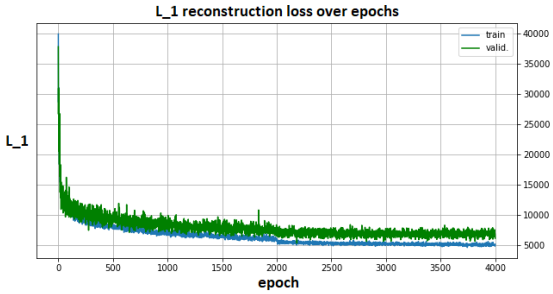


Figure 19: Loss over epochs for the L_1 *SID-net* (i.e. the model trained with only reconstruction loss). The blue curve is the training loss whereas the green one is the validation loss.

Figure 20: Comparison of *SID-net*s trained with the following objectives: 1) CVAE (baseline), 2) hybrid $\alpha = 0.9$, 3) hybrid $\alpha = 0.5$, 4) L_1. These are 512×512 crops from the full-sized predictions.

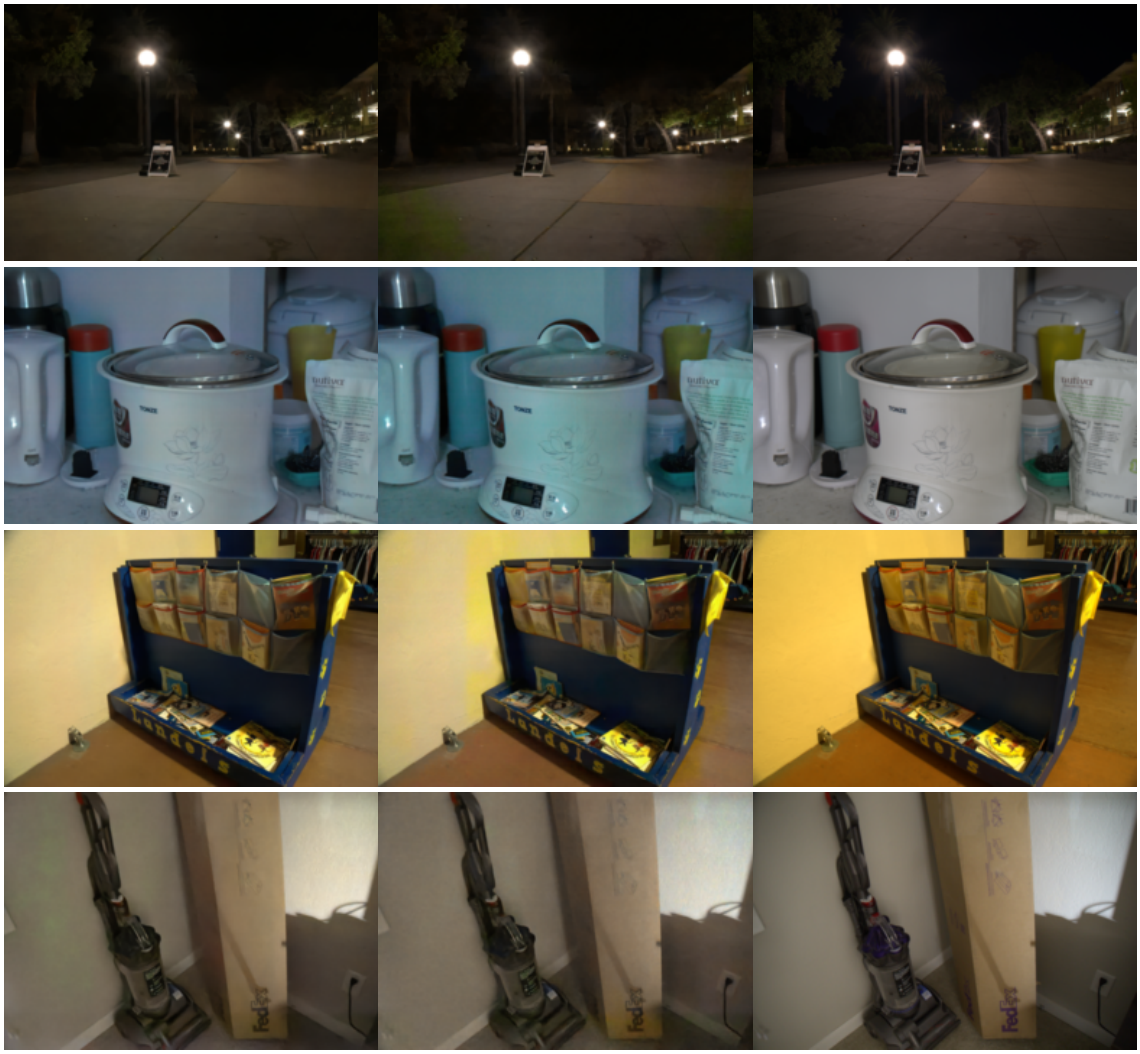


Figure 21: Comparison of *SID-net*s trained with following objectives: 1) CVAE (baseline) and 2) L_1, third column is ground truth. In the first example there are color differences in bottom corner. In the second example we notice color differences of the boiler, and in the third example the amount of yellow of the wall. In the last example we notice minor differences in color contrast of the box, the CVAE has some color artifacts on the wall.

7.1.2 Alternative Network Architectures

Now we are considering alternative networks architectures to the baseline *SID-net* architecture. All models here are trained as regular CVAE models. Regarding the *Pix2Pix* architecture, it is trained with a milestone at 1000 instead of 2000, for the KL-divergence the weight $\beta = 10.0$ is used. Furthermore, we also consider the alternative variants of the *SID-net* architecture, as listed in Table 2. We keep β at 2.0 for bot of these models. The *SID-net small* is faster to predict than the baseline architecture, in practice we found it to be a bit more than 40% faster at predicting. It is also an about 40% smaller model in terms of parameters. This network is trained during 4000 epochs with a milestone at epoch 3000. The model without skip connections: *SID-net no-skip* is trained during 4000 epochs with a milestone at epoch 2000. We list the number of parameters and amount of multiplications (MACs), in a forward pass, per model in Table 7.

Architecture	MACs (G)	Parameters (M)
Pix2Pix	18.16	54.41
SID-net	15.86	49.48
SID-net no-skip	12.84	46.33
SID-net small	4.01	29.16

Table 7: Number of MACs and weights/biases for different architectures. MACs are listed in $G = 10^9$ per forward pass of a 256×256 RAW patch at testing time, and parameters in $M = 10^6$. Ideally the number of MACs and parameters should be as small as possible.

The results are listed in Table 8. Regarding the model without skip connections, we notice the results are significantly worse than all other models so far. The *Pix2Pix* model is under-performing in all measures, compared to the baseline. Regarding the fast model it is a quite a bit worse than the baseline, but the difference is not that big considering the large difference in number of parameters. It is actually beating the largest model *Pix2Pix* in PSNR and MAE and also the no-skip version comfortably.

In Figure 22 a few predictions from the no-skip models are shown. Obviously they are on the soft side and little detail is restored, on the other hand the colors look quite good. In Figure 23 and 24 we are doing a comparison between the predictions from the baseline *SID-net* and the *Pix2Pix* networks. It can be seen that the *Pix2Pix* outputs images which often are a bit too green. The amount of detail is about the same between the two. But *Pix2Pix* has more problems with colors and a bit more artifacts. In Figure 25 a comparison of outputs from the baseline *SID-net* with those from *SID-net small* can be seen. We notice differences in clarity, with the baseline being a bit sharper and is doing a bit better job at restoring details. There is also a slight difference in saturation and dynamic range, to the favor of the baseline. Overall the *SID-net small* is holding up well considering the much smaller size.

Architecture	PSNR	SSIM	MAE
SID-net	28.3697 ± 0.050	0.7093 ± 0.0010	0.03080 ± 0.00016
Pix2Pix	27.8833 ± 0.083	0.7026 ± 0.0007	0.03226 ± 0.00035
SID-net no-skip	25.5379 ± 0.045	0.6497 ± 0.0007	0.03777 ± 0.00026
SID-net small	28.0005 ± 0.007	0.7021 ± 0.0002	0.03175 ± 0.00009

Table 8: Results on the test set for CVAE models based on alternative network architectures. Averages and standard deviations are over two training sessions each, except for *SID-net small* which was trained once.

Figure 22: Outputs of the *SID-net no-skip* CVAE model. These are crops of size 512×512 from the merged full-sized predictions.

Figure 23: Comparison of predictions from *SID-net* and *Pix2Pix*. Order is 1) *SID-net* prediction, 2) *Pix2Pix* prediction, 3) ground truth. These are crops of size 512×512 from the merged full-sized predictions.

Figure 24: Comparison of full-scale predictions from *SID-net* and *Pix2Pix*. Order is 1) *SID-net* prediction, 2) *Pix2Pix* prediction.

Figure 25: Comparison of the baseline *SID-net* with *SID-net small*. Order is: 1) *SID-net*, 2) *SID-net small*. These are crops of size 512×512 from the merged full-sized predictions.

7.1.3 Interpolation

All results so far have been based on using the baseline strategy for interpolation, i.e. strategy 1 for extracting patches and $p = 2$ as the power in equation (16) for interpolation. We are now comparing this baseline strategy with some alternatives. The values $p = 0$ (averaging) and $p = 2$ (spatial interpolation) are compared. We use the trained baseline *SID-net* model from before to predict patches. For extraction strategy 1 this means 126 patches are predicted, and for strategy 2, 54 patches are predicted. The following alternatives are considered:

- *baseline*: extraction strategy 1, $p = 2$ for interpolation
- *faster1*: extraction strategy 1, $p = 0$ for interpolation
- *faster2*: extraction strategy 2, $p = 2$ for interpolation
- *fastest*: extraction strategy 2, $p = 0$ for interpolation

The prediction times, and the merging times (interpolation) for the different methods are listed in Table 9. They were all measured on the Nvidia GPU we are using, and averaged over all samples in the test set. The evaluation scores on the test set, for each alternative are given in Table 10. The baseline option performs the best, but overall there are only small differences between the four. In Figure 26 we are showing a comparison between predictions based on the four alternatives for two differently noisy inputs. For the noisy input (left one) there are large differences between the predictions, only *baseline* manages to avoid most blocking effects. The difference between *faster1* and *fastest* is small, the difference is in the number of visible blocks. The effect of $p = 2$ versus averaging is noticeable, especially for extraction strategy 1, for extraction strategy 2 there is only a slight improvement. Regarding the second input, the less noisy one, there are only small differences between the four alternatives, *fastest* is not much worse than *baseline* on this example.

To get the best results we should use the baseline *SID-net* and *baseline* strategy above, enhancing a full-sized dark input RAW then takes about 0.6 seconds in total (prediction + interpolation). The fastest alternative would be to use *SID-net small* together with strategy *fastest* from above, then we should expect a bit over 0.2 seconds in total per image.

Strategy	Prediction time [s]	Interpolation time [s]
baseline	0.30	0.28
faster1	–	0.18
faster2	0.16	0.16
fastest	–	0.12

Table 9: Computation time per full-sized image for the different alternatives. As measured over all images in the test set. 'Prediction' refers to prediction (using *SID-net*) of bright patches based on the extracted dark patches as inputs. 'Interpolation' refers to merging of these patches into a full-size image.

Strategy	PSNR	SSIM	MAE
baseline	28.3560	0.7083	0.03082
faster1	28.3480	0.7075	0.03087
faster2	28.2721	0.7072	0.03117
fastest	28.2686	0.7069	0.03113

Table 10: Evaluation results on test-data for the considered strategies. Small differences between the strategies although the baseline performs best on each measure, but it is not very significant.

Figure 26: Comparison of (extraction and) interpolation strategies. Order is from top-down (excluding the input): 1) *baseline*, 2) *faster1*, 3) *faster2*, 4) *fastest*. These are crops of size 1024×1024 from the merged full-sized (RAW inputs and) predictions. ISO of the left input is 25600, and the right input has ISO 200. In the left example the input is extremely noisy, and the difference between the four strategies can be seen clearly. Going from *faster1* $\rightarrow$ *baseline*, and from *fastest* $\rightarrow$ *faster2* we can see the effect of using spatial interpolation versus averaging. Going from *faster2* $\rightarrow$ *baseline*, and from *fastest* $\rightarrow$ *faster1*, we can see the effect of larger overlap between patches. Look for visible 'blocks' / grid pattern.

7.2 Alternative Tasks

Earlier we have considered the main task of predicting a color image based on a RAW input. But we are also interested in tasks (1) and (2). *SID-net*s are trained for these alternative tasks, both trained as CVAEs with $\beta = 1.0$ using the same learning rate schedules as for the main task. Table 11 shows the results for *SID-net*s trained on the alternative tasks. Just like expected the scores for task (1) are improved versus the main, and the other way around for task (2).

Figure 27 and 28 show a comparison of predictions for the three tasks. Overall there are quite noticeable differences between the main model and task (2) model, in many aspects of image quality. The main difference is the tones of the images, and the RAW-based model is producing a bit sharper results with a bit better dynamic range. Differences in sharpness can be seen in the fourth (in Figure 28) example clearly, difference in amount of restored details can be seen on the wall in the third example. In the last example we can see color artifacts, it especially clear for the sRGB-based model (task (2)). In this example the model predicting only grayscale produces an image which looks more clean. Overall the outputs from the RAW-based model are better, which is also what the numbers in Table 11 say. The model which predicts a grayscale image is producing the sharpest results with best contrast and least artifacts. The difference between the color (RAW input) and grayscale outputs is not major in terms of amount of restored details. What is often bringing down the results for the color model (main) is wrongly predicted colors/tones (in comparison with the ground truths).

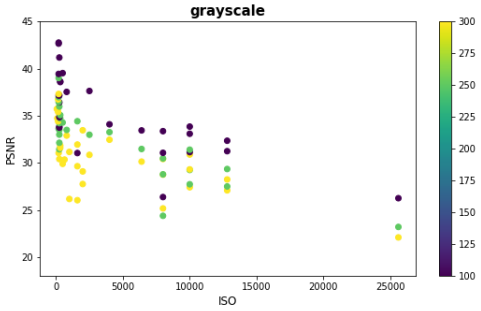
In Figure 29 the performances for the two alternative (tasks) *SID-net*s are plotted as functions of the ISO-level of the input image (and the exposure ratio). Compare with the corresponding Figure 16 for the baseline *SID-net* on the main task. Again we note a quite clear trend that as the ISO goes up the performance is reduced. In Figure 30 a comparison of the luminance distribution between the sRGB-based (task (2)) and RAW-based model (main) with the ground truth distribution is shown. It is clear that the RAW-based model is better in predicting the right exposure, the sRBG model mostly struggles with the darkest parts of the images.

Task	PSNR	SSIM	MAE
(1)	32.4972 ± 0.059	0.8629 ± 0.00081	0.02016 ± 0.00018
(2)	27.0572 ± 0.089	0.6928 ± 0.00090	0.03515 ± 0.00037

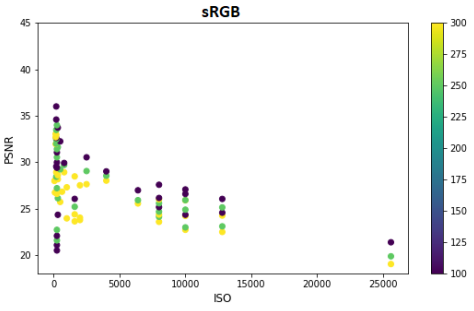
Table 11: Results of the *SID-net* on the alternative tasks. Both models were trained twice.

Figure 27: Comparison of full-size predictions of *SID-net*s for the three tasks. Order is: 1) main task (RAW input), 2) task (1) (grayscale prediction), 3) task (2) (sRGB input). Take note especially to the color difference between the first and third image.

Figure 28: Comparison of *SID-net* for the three tasks. Order is: 1) main task (RAW input), 2) task (1) (grayscale prediction), 3) task (2) (sRGB input). These are crops of size 512×512 from the merged full-sized predictions.

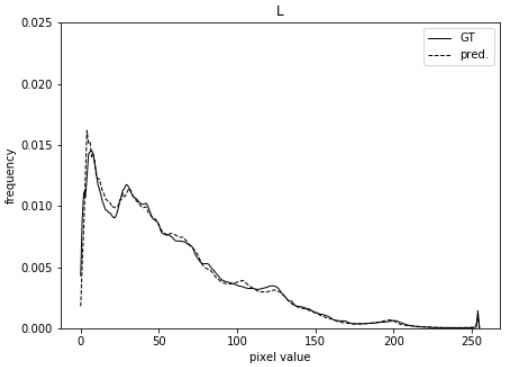


Task (1): RAW input, grayscale output.

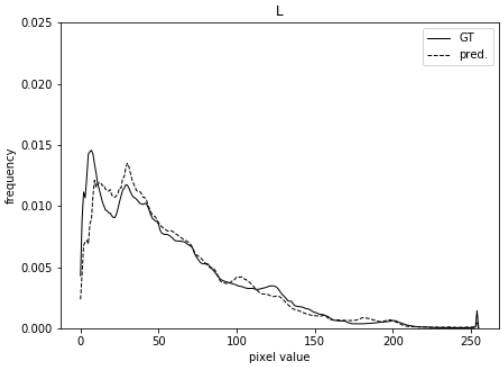


Task (2): sRGB input, color output.

Figure 29: ISO versus PSNR per image on the test set for the two alternative tasks. The color indicates what ratio each sample has (one of 100, 250, 300). Compare this figure to Figure 16. Note how the point clouds are lifted up when going from sRBG $\rightarrow$ RAW $\rightarrow$ grayscale. The ISO of the input seem to be an indicator of the prediction quality, this is especially clear for the grayscale model. The exposure ratio (i.e. how dark the input is) of the input is a weaker indicator as the input is amplified before being fed into the networks. Fitting a linear regressor on ISO (x) versus PSNR (y) for the grayscale predictions we get $R^2 = 0.42$, whereas a regressor based on the ratio returns $R^2 = 0.19$.



RAW input. Jensen-Shannon divergence 0.0018.



sRGB input. Jensen-Shannon divergence 0.0072.

Figure 30: Histograms over predicted luminance as over all predictions based on the input-data (either RAW or sRBG) in the test set. And comparison with ground truth sRGB images. Dashed lines correspond to predictions, the stroked ones to the ground truths. It can be seen that the sRBG model is predicting the light distribution worse than the RAW model. The sRBG model predicts what should be dark too bright.

8 Discussion

The main focus of this work has been on the VAE/CVAE models as methods. The aim was to build a pipeline based on a CVAE which accepts a dark input image and outputs a bright / denoised image. When constructing the CVAE models focus has mainly been on performance, but also on the speed of the pipeline. Our work was heavy influenced by that of [2], different from them we set for an approach based on a CVAE whereas they used a regular U-Net trained with only reconstruction loss. Our main architecture *SID-net* is based on their network architecture, but adapted to a CVAE model. This work mostly focused on the task of predicting a color image based on a RAW input, but we also considered the alternative task of predicting a grayscale image. We also investigated the effect of replacing the RAW input with an sRGB image. Due to different versions (spatial resolutions) of the *SID-dataset* our results should not be directly compared to the results of [2]. Ignoring this fact, a quick comparison of our best model with theirs shows that the predictions look very similar.

8.1 Architectures

Due to little related work on using a CVAE architecture for real-world image enhancement there was not a clear choice for what exact architecture to use. Although, based on the success of a U-Net on the same dataset before [2], it was quite an obvious choice to use a similar architecture with skip connections. Inspired by the works of [6] and [9] we set our focus on a CVAE with a U-Net architecture between the prior and decoder networks. However our research could not find an existing CVAE U-Net exactly suiting our needs. For this reason we chose to implement our own (*SID-net*) network architecture based on the regular U-Net architecture used in [2]. Through experiments of different variants of the architecture the baseline *SID-net* architecture turned out to be good. This with respect to balance between size of the model and performance. We also found the *Pix2Pix* architecture, which fitted our needs well. This network was not modified as much. There is still room for improvement when in it comes to both of these architectures.

We found that the *SID-net* outperformed *Pix2Pix*, see Table 8. The baseline *SID-net* is the smaller architecture, see Table 7, making it the better model of the two. One major difference between the two architectures (*SID-net* and *Pix2Pix*) is that the *Pix2Pix* uses dropout during training. The original version of the *Pix2Pix* architecture was designed to work well as generator of GAN network, whereas the base architecture of *SID-net* actually was tuned to well on the SID-dataset by [2]. These factors may be explanations to the difference in the results between the networks. In terms of visual results, in our eyes the predictions from *SID-net* were looking better and more alike the ground truths. The main differences between them are the tones. The *Pix2Pix* often predicts too green, see Figure 23, and the *Pix2Pix* has more artifacts. Though, it is hard to objectively decide which of two images is looking the best without being affected by your subjective taste. For this reason it might be better to trust the numbers. However, just because the *Pix2Pix* is performing worse on the baseline task, it does not mean it would perform worse on the alternative data/tasks.

As an alternative architecture we also considered a 40% smaller and faster network (than the baseline *SID-net*): *SID-net small*. It stacked up well to the baseline *SID-net*, see Table 8. Furthermore we also considered a variant without skip connections. This model under-performed versus the other networks and produced soft results. On the main task, most models produce similar results, but the results from the model without skip connections are significantly worse than the results from the others. Using skip connections results with better predictions are obtained with much more detail in the images. This is an indication of that most of the performance comes from the usage of the skip connections. Based on the results it seems like a U-Net architecture with skip connections is working well on the main task. Overall, the network architecture that performs the best is the baseline *SID-net*. Additional to U-Net architectures using skip connections with concatenation we also experimented with models using residual connections (addition), but we found those models to perform worse.

8.2 Training

We found that our learning rate schedule, with a drop of the initial learning rate at some epoch 'milestone', to work out well. This is the same approach as was used by [2]. As standard this milestone was set to 2000. Mainly due to limiting time we did not tune the learning rate schedule for each individual model very much. The milestone was really the only hyperparameters which was tuned, it was mainly just changed if we observed training was too unstable or slow. Due to training not being tuned 'optimally' for all the models this might cause a bias towards one of the models. To make comparisons as fair as possible we chose to train all models during the same number of epochs (4000). Even though some models could have been stopped training earlier, around epoch 3000. In the end of training most models had weights (of the neural networks) oscillating quite a lot. Thus between two versions of the weights from two consecutive epochs, there could be a difference in the evaluation performance which was not insignificant. To reduce the uncertainty in the results we chose to save the weights from the four last epochs and evaluate the results based all of them. This number could have been increased even more to bring down the uncertainty.

Additional to the regular CVAE ELBO we considered alternative ways of training the *SID-net* architecture: as hybrid models and as regular U-Net with only reconstruction loss (L_1). Out of the CVAE and L_1 model, on the main task the results show that the regular CVAE is the better performer in PSNR, SSIM and MAE (in average). The differences in the measures are very subtle, especially in MAE there is quite some overlap between the two. Visual results, see Figure 20, do not show large differences between the four models (CVAE, two hybrid models and L_1 model). We saw that both the CVAE and L_1 model had some color problems, but those of the CVAE are less severe. But it should be mentioned this is not a feature of the CVAE loss function, we saw the *Pix2Pix* network had similar problems, see Figure 24. We also tried training the baseline *SID-net* CVAE model with $L = 5$ samples, but there was not a significant difference in the results, see Table 6. For this reason, and to keep the training times low, we chose to stick with $L = 1$ samples for the remaining models.

What is the best way to train the *SID-net* with respect to loss function then? Due to small differences in the results it is hard to answer. There is a weak indication that for the baseline *SID-net* (on the main task) with the fixed learning rate schedule, that a high $\alpha \approx 1$ of the hybrid loss (15) is a better choice than a low $\alpha \approx 0$ [7]. Ignoring the training times, then the regular CVAE is the most promising model. If training time is a factor, then non of the hybrid models with $0 < \alpha < 1$ are a good choice due to longer training times than the CVAE (and L_1 model). The L_1 model is fastest to train due to it only has one downsampling network (at training time). Taking into account training time differences, we cannot really say the CVAE loss is bringing much value over the L_1 model. The main thing the CVAE *SID-net* brings over the L_1 version is a bit less of color problems.

8.3 Latent space

One of main features of a CVAE is that it has two downsampling networks $q_\phi(z|x)$ and $p_\theta(z|y)$ during training. And the z which is passed through the decoder during training is a sample from $q_\phi(z|x)$ and not from $p_\theta(z|y)$. Even though training may turn out to work well (in terms of reconstructions based on $z \sim q_\phi(z|x)$), this does not necessarily mean samples from $p_\theta(z|y)$ will be good. We tried to force the distributions together by usage of a high β for the KL-divergence, and we tried to push the pipelines together by using the hybrid loss function. As we could see in Table 6 the results were actually worsened a bit (in average) when using $\alpha < 1$. We could not see any significant difference in the results between models trained with different values of β, when decoding using $p_\theta(z|y)$. For this reason we fixed β on the alternative tasks and for the alternative SID-nets.

[7] Hybrid with $\alpha = 1$ is a regular CVAE. Hybrid with $\alpha = 0$ is the reconstruction-loss model.

Regarding the baseline *SID-net* CVAE model, we found that there was actually a lot of information stored in the latent space, mainly information about color. Even though it is not the way the model should be (or can be) used in real applications, we compared using samples from $q_\phi(z|x)$ and $p_\theta(z|y)$, and found that reconstructions based on the mean of $q_\phi(z|x)$ in comparison with the mean of $p_\theta(z|y)$ were much better. There was usually about 25% difference in MAE in favor of $q_\phi(z|x)$ (obviously the results in this report were based on samples from $p_\theta(z|y)$). The CVAE model has learned to store useful information in the latent space, the problem is just that $p_\theta(z|y)$ has a hard time navigating it. Even though the samples from $p_\theta(z|y)$ do not provide as much information, it is still a significant advantage to use a conditional prior over an unconditional prior due to it allows more freedom of the latent space. And also since no sampling is needed at testing time, which could have caused overlapping patches to look totally different.

8.4 Interpolation

Our main goal regarding interpolation was to avoid unwanted blocking effects when merging predicted patches. But we also wanted to keep the time of merging as low as possible. As can be seen Figure 26 our *baseline* algorithm produces results which leaves little blocking effects. For the extremely noisy inputs we found that only the *baseline* strategy produced acceptable results. For the less noisy inputs, all of the considered alternatives are doing a good enough job. Even though there are still some visible blocking effects left if you really look for them (especially for the three remaining strategies). See figure the right example in 26. The effect of using spatial interpolation ($p = 2$) over averaging is quite noticeable especially for the noisy inputs and for extraction strategy 1. For extraction strategy 2 there is only a small difference between averaging and inverse distance interpolation. Ignoring the baseline strategy there is really not a large visual difference between the remaining, except the number of visible blocks and where edges are located. Thus since *fastest* is the fastest of them it must be considered the best option. If the input is not so noisy and speed is of concern then *fastest* strategy is the best choice. Otherwise the *baseline* is the best choice.

8.5 Alternative Tasks and Image Quality

Regarding the baseline *SID-net* on the main task. In general the outputs look visually fine for inputs which are not too noisy. The pipeline manages to successfully brighten up the image and remove the noise. Colors are mostly right, although have a bit low saturation compared to the ground truths (which is to be expected). The amount of detail is fine, and there are usually only little artifacts. For inputs which are very noisy (and especially if they are very dark too) turned often out to look quite bad, see for example the left example in Figure 26, and the green tones of the wall in the image at fourth row, first column in Figure 21. In our eyes color predictions based on inputs with very high ISO are barely usable. Of course enhancing a noisy image is a harder task than enhancing a quite noise-free image. We saw that many models, even the best architecture *SID-net* with RAW input, had problems with colors being on the green side (for some outputs) and this sometimes causing the images to have what looks like green stains. This was especially prominent for models with sub-average reconstructions no matter if RAW or sRGB input.

Regarding the alternative tasks, we could see a significant difference in the results between the three models, see Table 11. What all models have in common is that they produce the worst results for the inputs which are extremely noisy, see figures 16 and 29. Starting with the comparison between the two models trained to predict color, i.e. the main model taking a RAW input, and model trained with sRGB input. We could see significant differences in the three measures. Also the visual results show large differences. Even though there are differences in the amount of restored details / sharpness and dynamic range, the largest differences are in the colors, see Figure 28. The *SID-net* taking sRGB input often produces unsatisfactory results, even for the not so noisy inputs, see for example the prediction at row four, column three in Figure 28. Overall it can be concluded that using a RAW input over a sRGB input is improving the results significantly. This is not to our surprise given the higher level of information in the RAW image (compared to the corresponding sRGB) and the fact [2] got the same results.

Between the two models taking RAW input, i.e. the main model which predicts color and the model predicting luminance only, the results are not extremely different. The grayscale model restores more details, and the results are looking cleaner. The main difference in the results is due to the model predicting color has some color artifacts which is bringing down the results, and colors which are predicted wrong (e.g. the yellow wall in Figure 21). In general, if the goal is to get a clean image output, with as much detail as possible, then the *SID-net* predicting only grayscale is the best option.

8.6 Future Work

A drawback of our pipeline is that at testing time, the input is still amplified by the exposure ratio. This means that we are assuming the correct exposure time is known beforehand. One way to come around this is to estimate what the true shutter time should be, and then try different values of the amplification ratio. However it would be convenient with a module in the pipeline which automatically estimates the amplification. Another disadvantage is that the model is learned from a dataset with images from a specific camera. Thus we cannot expect the model to work out well for input images from another camera. If we would have several cameras, we probably would have to train one model per camera. These are the same disadvantages as [2] has, since our pipeline is based on theirs.

There is actually a way one could come around using interpolation at testing time, and instead input full-sized images directly to the trained *SID-net* just like [2] do. As it was constructed and the way we have used it throughout this thesis, the network is not compatible straight away with a full-sized 1424×2128 input. But by converting the network to a fully-convolutional network (e.g. by converting the fully connected layers to CNN layers), and possibly some other changes, it could be. This would of course mean that the size of the bottleneck would change. We did not try to implement this, mainly due to shortage of time. There is no guarantee it would work out well, but if it would this way the pipeline can be simplified and possibly improved in terms of speed.

Regarding the CVAE models. There are several things that could be improved, and especially the network architectures. To simplify things we were using the same architectures for both downsampling networks. There is however nothing saying that the encoder and prior networks must have the same base architectures. Regarding training of the CVAEs, we found the samples from the prior $p_\theta(z|y)$ to be much worse than those from the encoder $q_\phi(z|x)$, thus attempts to improve the samples from $p_\theta(z|y)$ would be of interest. Another possible improvement could be to use a more sophisticated reconstruction loss function, such as loss functions based on features from deep learning networks, for example the perceptual similarity metric [25]. Any effort to try to improve predictions based on dark input sRGBs would be useful. This due to their wider availability and smaller size compared to RAWs.

9 Conclusion

This thesis shows it is possible to apply a Conditional Variational Autoencoder (CVAE) to a real-world image transfer problem and get good results for input images in RAW format which have a reasonably low noise-level. Models taking inputs which are in RAW format perform significantly better than the model accepting regular sRGB images. The model predicting grayscale images produces cleaner and sharper images than the others. Regarding interpolation we found that the simpler averaging algorithm to work well enough on relatively noise-free inputs, whereas only the slowest algorithm with spatial interpolation performed well enough for extremely noisy inputs. We found that a U-Net was a suitable type of network architecture. Without the skip connections very little detail is restored. Overall the performance is due to the architecture of the neural network itself and type of input data, rather than the type of loss function.

References

[1] Nikolas Adaloglou. "Intuitive Explanation of Skip Connections in Deep Learning". In: *https://theaisummer.com/* (2020). URL: https://theaisummer.com/skip-connections/.

[2] Chen Chen et al. "Learning to See in the Dark". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (June 2018). DOI: 10.1109/cvpr.2018.00347. URL: http://dx.doi.org/10.1109/CVPR.2018.00347.

[3] Aditya Deshpande et al. "Learning Diverse Image Colorization". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (July 2017). DOI: 10.1109/cvpr.2017.307. URL: http://dx.doi.org/10.1109/CVPR.2017.307.

[4] Vincent Dumoulin and Francesco Visin. *A guide to convolution arithmetic for deep learning*. 2016. arXiv: 1603.07285 [stat.ML].

[5] Encyclopedia of Mathematics. *Jensen inequality*. [Online; accessed 30-September-2020]. URL: http://encyclopediaofmath.org/index.php?title=Jensen_inequality&oldid=47465.

[6] Patrick Esser and Ekaterina Sutter. "A Variational U-Net for Conditional Appearance and Shape Generation". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (June 2018). DOI: 10.1109/cvpr.2018.00923. URL: http://dx.doi.org/10.1109/CVPR.2018.00923.

[7] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[8] Phillip Isola et al. *Image-to-Image Translation with Conditional Adversarial Networks*. 2016. arXiv: 1611.07004 [cs.CV].

[9] Oleg Ivanov, Michael Figurnov, and Dmitry Vetrov. *Variational Autoencoder with Arbitrary Conditioning*. 2018. arXiv: 1806.02382 [stat.ML].

[10] John Duchi. "Derivations for Linear Algebra and Optimization". In: (2014). [Online; accessed 8-March-2020]. URL: http://web.stanford.edu/~jduchi/projects/general_notes.pdf.

[11] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2013. arXiv: 1312.6114 [stat.ML].

[12] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[13] Yann LeCun, Corinna Cortes, and CJ Burges. "MNIST handwritten digit database. 2010". In: *http://yann.lecun.com/exdb/mnist* (2010).

[14] Hao Li et al. "Visualizing the Loss Landscape of Neural Nets". In: (2017). arXiv: 1712.09913 [cs.LG].

[15] Chang Liu, Xiaolin Wu, and Xiao Shu. "Learning-Based Dequantization For Image Restoration Against Extremely Poor Illumination". In: (2018). arXiv: 1803.01532 [cs.CV].

[16] Paras Maharjan et al. "Improving extreme low-light image denoising via residual learning". In: *2019 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE. 2019, pp. 916–921.

[17] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: (2015). arXiv: 1505.04597 [cs.CV].

[18] Yangming Shi, Xiaopo Wu, and Ming Zhu. "Low-light Image Enhancement Algorithm Based on Retinex and Generative Adversarial Network". In: (2019). arXiv: 1906.06027 [cs.CV].

[19] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. "Learning structured output representation using deep conditional generative models". In: *Advances in neural information processing systems*. 2015, pp. 3483–3491.

[20] Zhou Wang et al. "Image quality assessment: from error visibility to structural similarity". In: *IEEE transactions on image processing* 13.4 (2004), pp. 600–612.

[21] Wikipedia contributors. *Bayer filter — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-September-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Bayer_filter&oldid=977384299.

[22] Wikipedia contributors. *Evidence lower bound — Wikipedia, The Free Encyclopedia*. [Online; accessed 28-March-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Evidence_lower_bound&oldid=934842406.

[23] Wikipedia contributors. *Jensen's inequality — Wikipedia, The Free Encyclopedia*. [Online; accessed 8-March-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Jensen%27s_inequality&oldid=938883278.

[24] Wikipedia contributors. *Universal approximation theorem — Wikipedia, The Free Encyclopedia*. [Online; accessed 12-March-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Universal_approximation_theorem&oldid=941288803.

[25] Richard Zhang et al. "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric". In: (2018). arXiv: 1801.03924 [cs.CV].

[26] Yang Zhang, Changhui Hu, and Xiaobo Lu. "Deep Attentive Generative Adversarial Network for Photo-Realistic Image De-Quantization". In: (2020). arXiv: 2004.03150 [eess.IV].

Appendix

KL-divergence between two Gaussians

In the case of two Gaussians, of the same dimensionality k, i.e. $q(z) = \mathcal{N}(z|\mu_q, \Sigma_q)$ and $p(z) = \mathcal{N}(z|\mu_p, \Sigma_p)$ the KL-divergence [10] is given by

$$D_{KL}\big(q||p\big) = \frac{1}{2}\big(\log\frac{\det\Sigma_P}{\det\Sigma_Q} - k + \mathrm{tr}(\Sigma_P^{-1}\Sigma_Q) + (\mu_P - \mu_Q)^T\Sigma_P^{-1}(\mu_P - \mu_Q)\big), \tag{23}$$

here $\mathrm{tr}(\cdot)$ is the trace-function.

In the case of both q and p having diagonal covariances, and where the parameters are given by:

$$\mu_q = \big[\mu_{q,1}, \mu_{q,2}, \ldots, \mu_{q,k}\big]$$

,

$$\Sigma_q = \mathrm{diag}\big([\sigma_{q,1}^2, \sigma_{q,2}^2, \ldots, \sigma_{q,k}^2]\big)$$

and

$$\mu_p = \big[\mu_{p,1}, \mu_{p,2}, \ldots, \mu_{p,k}\big]$$

,

$$\Sigma_p = \mathrm{diag}\big([\sigma_{p,1}^2, \sigma_{p,2}^2, \ldots, \sigma_{p,k}^2]\big)$$

we have

$$\begin{aligned} D_{KL}\big(q||p\big) =& \frac{1}{2}\big(\sum_i \log\sigma_{p,i}^2 - \sum_i \log\sigma_{q,i}^2 - k + \sum_i \frac{\sigma_{q,i}^2}{\sigma_{p,i}^2} + \sum_i \frac{(\mu_{q_i} - \mu_{p,i})^2}{\sigma_{p,i}^2}\big) \\ =& \frac{1}{2}\sum_i \log\frac{\sigma_{p,i}^2}{\sigma_{q,i}^2} - 1 + \frac{\sigma_{q,i}^2}{\sigma_{p,i}^2} + \frac{(\mu_{q_i} - \mu_{p,i})^2}{\sigma_{p,i}^2}. \end{aligned} \tag{24}$$

In the case of $p(z) = \mathcal{N}(0, I)$, this formula simplifies to

$$D_{KL}\big(q||p\big) = \frac{1}{2}\sum_i -\log\sigma_{q,i}^2 - 1 + \sigma_{q,i}^2 + \mu_{q,i}^2. \tag{25}$$

MNIST network architectures

The first architecture is a MLP which accepts an image stacked into an array as input. For the hidden layers we use LeakyReLU activations, except for the the layer leading down to the bottleneck. The architecture of the MLP is listed in table 12 in the case of a two-dimensional bottleneck. CNN networks are usually more efficient than pure MLPs when it comes to image-data, based on this we consider a combined CNN-MLP. The encoder network is given in Table 13 below. The decoder is inverted and uses transposed convolution layers instead. We use kernels of size 3, and strides of size 2 and zero-padding to get output sizes listed. This model actually has fewer parameters than the standard MLP model.

Layer name	**Layer type**	**Details**	**Input size**	**Output size**
fc1	fc	512 hidden nodes	784	512
z_mean	fc	2 mean components	512	2
z_var	fc	2 variance components	512	2
fc2	fc	input is sampled z	2	512
x_mean	fc	mean	512	784

Table 12: MLP architecture for a VAE with two-dimensional bottleneck, *fc* is short for fully connected.

Layer name	**Layer type**	**Details**	**Input size**	**Output size**
conv1	CNN	32 channels	(1,28,28)	(32,14,14)
conv2	CNN	64 channels	(32,14,14)	(64,7,7)
conv3	CNN	64 channels	(64,7,7)	(64,4,4)
fc1	fc	256 nodes, input is flattened first	(64,4,4)	256
z_mean	fc	input is fc1	256	10
z_var	fc	input is fc1	256	10

Table 13: Encoder CNN architecture, *fc* is short for fully connected.

Master's Theses in Mathematical Sciences 2020:E75
ISSN 1404-6342

LUTFMA-3432-2020

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/