Spatio-Temporal Noise Filtering using Convolutional Neural Networks with a Realistic Noise Model under Low-Light Conditions

Tim Borglund
dic13tbo@student.lu.se

Tor Nilsson
bas12tni@student.lu.se

February 5, 2019

Master's thesis work carried out at
the Department of Mathematics and the Department of
Computer Science, Lund University.

Supervisor: Niels Christian Overgaard, nco@maths.lth.se

Examiner: Pierre Nugues, pierre.nugues@cs.lth.se

Abstract

Convolutional neural networks have in recent years been successfully employed for various image processing tasks, such as filtering noise. There are however relatively few published attempts for processing video in this way. Image processing methods on single images can be applied frame by frame, but often fail to consider continuity and flow between frames. In this master's thesis we constructed several fully convolutional neural network models, trained to filter noise spatially as well as temporally. We present the differences between these models and compare the performance of each of them with a noise filter from a state-of-the-art camera, as well as with a solely spatial filter. Our data was created by adding noise to clean videos according to a noise model which realistically simulates noise from camera sensors under low-light conditions. On a frame by frame basis, our best model outperforms the state-of-the-art camera in most situations. Despite still having minor struggles with continuity in video, clear improvement can also be seen in comparison with only spatial noise filtering.

Keywords: Deep Learning, CNN, Image Processing, Video Denoising, Noise Filtering

Preface

This master's thesis was produced from the 1st of September until the 31th of January at the facilities of Axis Communications in Lund. It was carried out in cooperation with the Department of Mathematics at LTH, the Department of Computer Science at LTH, and Axis Communications AB.

We would like to give special mention to our supervisors Niels Christian Overgaard, Gunnar Dahlgren, Karin Dammer, and Niclas Svensson for their guidance and engagement in the project. We would also like to thank other employees at Axis for taking their time to help us whenever asked.

Both authors have spent roughly a similar amount of time working with this project, and we value each other's contribution equally.

Contents

Chapter 1
Introduction

(a) Our result PSNR: 29.92 | **(b)** State-of-the-art camera* PSNR: 26.38 | **(c)** Input | **(d)** Ground truth

Figure 1.1: Our denoising result compared to a state-of-the-art camera*.

In recent times, convolutional neural networks (CNNs) have been demonstrated to outperform most other methods for processing images in tasks such as deblurring, demosaicing, and denoising. A good example are the results produced by Chen et al. [1]. They successfully denoised low-light images using a fully convolutional neural network (FCNN), with better performance than current standard algorithms. Nevertheless, Plotz and Roth [2] showed that many recent papers on image processing with neural networks simply use synthetic (artificial) noise added to clean images. The results from these papers may therefore be inaccurate in practical settings, as commonly used noise models such as the Gaussian distribution may be different from noise generated by real cameras.

*Our method for producing the results for the state-of-the-art camera is explained in Section 3.2.1.

Our primary goal was to investigate the potential of using CNNs for spatio-temporal noise filtering of videos, and how it would fare in comparison to other methods. There are multiple proposed network architectures for single image denoising [1] [3] [4], but not nearly as many for video. A reason why could be the difficulty of finding a metric for video comparison. Prevalent metrics in image processing may be appropriate for frame by frame comparisons, but lack some relevance when comparing entire videos due to not measuring flow and temporal coherence. We propose a hypothesis that a CNN can learn how to incorporate the temporal similarities between nearby frames in a video, in order to achieve better denoising results than what currently exists.

We have experimented with multiple configurations in several architectures of CNNs. We studied these configurations and evaluated them based on performance. A noise filter in a state-of-the-art camera was used as comparison. This noise filter is part of a modern, high performing image pipeline, contained in the camera. An image showing the difference in denoising results between this camera and one of our models can be seen in Figure 1.1. Our training and test data were put together by adding realistic synthetic noise to videos according to our noise model, to simulate images captured by a real camera sensor in low light.

1.1 Purpose and Problem Statement

The purpose of this master's thesis was to investigate whether CNNs can be trained to utilize temporal similarities in order to filter noise in videos, and if there is benefit in doing so rather than only filtering spatially. A primary objective was to ensure that our results are comparable to a state-of-the-art noise filter. Our aim was to design and train the noise filter to perform well under low-light conditions. We present the following problem formulations for this thesis:

- Is it possible to perform spatio-temporal noise filtering on video using a CNN, in such a way that it outperforms a high performing, solely spatial, noise filter applied frame by frame under low-light conditions?
- Is it possible to perform spatio-temporal noise filtering on video using a CNN, such that it outperforms a state-of-the-art denoising algorithm in a modern image pipeline under low-light conditions?

We were interested in concluding the best model given a number of configurations used for training and testing. The performance of a noise filter was measured frame by frame using peak signal-to-noise ratio (PSNR). In addition, we analyzed temporal coherence through manual inspection of videos.

1.2 Limitations

Training the networks on images with synthetic noise and then testing them on real noise, would be an excellent way of verifying that the noise model accurately mimics real noise. However, because of time constraints this was never attempted. Focus was instead put on improving the performance of our models on synthetic noise.

There are few public datasets available for video processing. To clarify, we refer to datasets that have both the clean and noisy versions of the videos. This is particularly true for datasets where the noise was added to videos in raw format. We therefore had to compile and pre-process our own data. By having an even higher quality dataset, better results than what we achieved may be possible. Furthermore, different types of neural network architectures such as recurrent neural networks (RNN) have been successful in image processing tasks [4]. To limit the scope of this thesis, we chose to stay within the realm of CNNs.

Finding the most computationally efficient denoising method, or one that is limited by camera hardware, was not part of this thesis. Instead, focus was put on achieving high performance with regards to suitable measurements presented in Chapter 3. Moreover, the hardware at disposal (i.e. the GPU and the CPU) for training the networks put up its own limitations. Having many frames in memory at the same time (as opposed to one frame at a time, as in the spatial case) proved to be expensive with regards to both memory and computational cost. This hindrance restricted network architecture and design.

Chapter 2
Theoretical Background

The necessary theory for this master's thesis includes how images are processed in camera image pipelines, how images are affected by noise and how it can be reduced, how neural networks are used in image processing, and how our thesis relates to previous work. These areas are covered in this chapter.

2.1 Digital Images and Color Models

There are multiple ways to represent images digitally. Generally, light that reaches an image sensor is encoded to information through a color filter array (CFA), which is placed right in front of the sensor [5]. As the name indicates, the light is filtered into components of different colors, namely green, red, and blue. An image represented in this way is said to be in raw format. 12 bits per channel are often used, so that the color components for each pixel accepts a value between 0 and 4095. The most common type of CFA is a Bayer pattern arrangement of color filters, which is depicted in Figure 2.1. Half the pixels filter green light as the human eye is most sensitive to this color, and one fourth of the elements filter red and blue light respectively. An image in raw format has a threshold where some pixel values greater than zero will correspond to a pixel value of zero. This threshold is known as the *black level*.

The RGB format is most commonly used for representing color images. This is an additive color model where the image is divided into three channels, red, green, and blue, so that each pixel has three color values. Another frequently used color model is YCbCr which separates the luminance, meaning brightness, and chrominance, which is the color information. Here Y denotes the luminance whereas Cb and Cr are the chrominance-blue (blue-difference) and chrominance-red (red-difference) components. Black and white images are usually referred to as grayscale images. Such images have only one channel, in which pixel values represent the intensity of light.

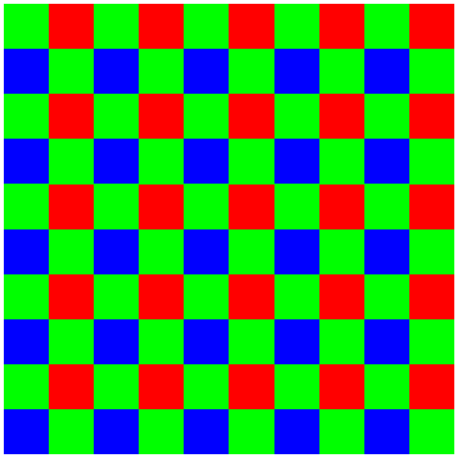


Figure 2.1: Bayer filter mosaic, a type of color filter array.

2.2 Modelling Image Noise

Image noise can be described as random variations of pixel values and is essentially loss of information. Noise typically makes images look visually grainy. As an image sensor is struck by photons, the energy of the photons is converted into electrons. The electrons produce an electric charge that is measured [6]. This measured value represents the intensity of light. The sensor is divided into pixels and has a certain full well capacity, meaning the number of photons that may be recorded. Low-light situations lead to low photon counts, resulting in noisy images. A higher full well therefore leads to less noise and easier denoising.

The cause of noise in an image originates from the nature of light as well as from the sensor and electronic components of the camera. Hence, noise as modelled in this thesis has two components: shot noise μ_s and read noise μ_r. A noisy image I_n can be expressed as $I_n = I_c + \mu_s + \mu_r$, where I_c is a noise-free image. Noise-free images will in this thesis be referred to as clean images.

2.2.1 Shot Noise

Despite illuminating a sensor evenly for some amount of time, the number of photons X that hit a certain pixel is not constant [7]. This variance is called shot noise and affects all sensors. Shot noise is signal dependent, meaning that it depends on how many photons that currently hit the sensor. It follows a Poisson distribution, $X \sim \text{Poisson}(\lambda)$, as is illustrated in Equation (2.1).

$$f(x;\lambda) = \frac{\lambda^x e^{-\lambda}}{x!} \tag{2.1}$$

The mean value λ of the Poisson distribution is the average number of photons arriving at the sensor during exposure and varies from element to element. Incidentally, by using the central limit theorem, X can be approximated to a Gaussian distribution when λ is sufficiently large [8]. The mean and variance of this distribution are then both equal to λ.

2.2.2 Read Noise

The process of measuring the generated electric charge in a pixel includes several sources of error, in the pixel itself as well as in the following camera components. The combination of errors originating from these sources is called read noise [7]. Unlike shot noise, read noise is not signal dependent. It follows a Gaussian distribution, $X \sim \text{Gaussian}(\lambda)$, i.e.,

$$f(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.2}$$

with a mean value of $\mu = 0$ and a standard deviation σ, which depends on the above mentioned sources of error.

2.3 Image Processing

Image processing encompass all usage of computer algorithms to carry out processing of images [9]. Some of the most common procedures are image quality improvement tasks such as removing blur (deblurring) or demosaicing [10]. Demosaicing is the process of reconstructing a color image from the information loss in the CFA [11].

During recent years, there have been a huge surge in the use of neural networks, especially for classification tasks [12], but also for image processing. A famous example is the AlexNet which won the 2012 edition of ImageNet large scale visual recognition challenge, a competition that has been a driving force for the development of CNNs since 2010. AlexNet introduced and popularized multiple ideas in deep learning that are frequently used today, such as the rectified linear unit (ReLU) activation function.

2.3.1 Image Processing Pipeline

An image processing pipeline consists of several steps to transform an image from raw format into a format that is more suitable to be observed by humans. Which steps to include and how to perform them vary from camera to camera. The general steps, however, are: capturing the raw image from the sensor, white balancing, tone mapping, demosaicing, denoising, black level removal, gamma correction, and further post-processing. Other than making images look appealing to humans, considerations and trade-offs can be made for storage space and computational cost.

2.3.2 Convolutions and Kernels

Kernels, or filters, are the cornerstones of image processing. These are matrices, often with sizes 3×3 or 5×5, used in convolutions. Convolution is an operation where the kernel slides over another matrix. For each position, every pixel is added together with its neighbors, weighted by the kernel. The number of neighbors to consider is determined by the kernel size. Equation (2.3) is an example of the calculations for a single position in a convolution, where the element y_{11} in a resulting matrix Y is calculated through addition of element-wise products.

$$
\begin{aligned}
y_{11} &= \begin{bmatrix} x_{00} & x_{01} & x_{02} \\ x_{10} & x_{11} & x_{12} \\ x_{20} & x_{21} & x_{22} \end{bmatrix} \odot \begin{bmatrix} k_{00} & k_{01} & k_{02} \\ k_{10} & k_{11} & k_{12} \\ k_{20} & k_{21} & k_{22} \end{bmatrix} \\
&= (x_{00} \cdot k_{00}) + (x_{01} \cdot k_{01}) + (x_{02} \cdot k_{02}) + (x_{10} \cdot k_{10}) + (x_{11} \cdot k_{11}) + (x_{12} \cdot k_{12}) + (x_{20} \cdot k_{20}) \\
&\quad + (x_{21} \cdot k_{21}) + (x_{22} \cdot k_{22})
\end{aligned} \tag{2.3}
$$

The entire resulting matrix can be expressed as $Y = X * K$, where the symbol $*$ denotes convolution and K is a filter kernel. It is possible to achieve different visual effects and distortions through modification of K. Commonly desired effects are blurring and sharpening, but also edge detection which may be used in feature extraction for a classification task.

2.4 Denoising

The image processing technique of removing noise is called denoising. An example of an image denoising algorithm that is currently used in research and industry is block-matching and 3D filtering (BM3D) [13]. A simpler denoising can be achieved through convolution with, for example, a Gaussian filter. This way, information from neighboring pixels is utilized in order to recover information for a certain pixel. This is in image processing called spatial filtering.

PSNR

The peak signal-to-noise ratio (PSNR) is commonly used to measure image quality of a denoised image given a clean target. Since PSNR is the ratio of power between signal and noise, a higher PSNR value means that the denoised image is more similar to the target. The formula for PSNR is given in the following equation:

$$
\mathrm{PSNR}(y, \hat{y}) = 10 \cdot \log_{10}\left(\frac{\mathrm{MAX}^2}{L_{\mathrm{MSE}}(y, \hat{y})}\right) \tag{2.4}
$$

where L_{MSE} is given by Equation (2.9), y is the target image, $\hat{y}$ is the output image, and MAX is the largest possible pixel value. As an example, for an 8-bit image this value is $2^8 - 1 = 255$.

2.4.1 Video

In video, a third dimension called depth is introduced in addition to the spatial dimensions of height and width. The depth corresponds to the number of frames in a series, hence it is a temporal dimension. The majority of adjacent frames in a video consist in a practical setting of similar information. In other words, for a large proportion of pixel values $f_{0_{ij}}$ in position i, j of a video frame F_0, $f_{0_{ij}} \approx f_{1_{ij}}$ for an adjacent frame F_1. Similarities in the temporal dimensions could therefore be used to recover information lost due to noise. The noise in one frame does not depend on noise in previous frames, meaning that the noise is temporally uncorrelated. Suppose, for instance, that some noise μ_i is applied to every frame F_i, $i = 0, 1, \ldots, t$ in a video $\boldsymbol{F}$ with t frames. Then the observed image becomes $I_i = F_i + \mu_i$. If $\mu_{1_{ij}} \approx 0$ so that $f_{1_{ij}} + \mu_{1_{ij}} = i_{1_{ij}} \approx f_{1_{ij}}$ for a pixel in position i, j, then in cases where $|\mu_{0_{ij}}| \gg 0$, $f_{0_{ij}}$ may be approximated with the value of $i_{1_{ij}}$. We say information is recovered temporally.

2.5 Neural Networks

The functionality of neurons in the human brain may be modelled by artificial neurons. A neuron is said to be fired when provided with some stimulation. They may in turn be connected in artificial neural networks (ANN), inspired by how the human brain is structured. Neurons are arranged in layers in such networks: Every ANN has an input layer and an output layer, with connections between. A network may be extended depth-wise with hidden intermediate layers that essentially increase the complexity. The discipline of working with such networks goes by the name of Deep Learning. The learning part refers to the methodology of updating the trainable variables of a network to achieve desired results.

A basic neural network can be described as a function $f(x, \theta)$, where x is an input and θ is a vector containing the trainable variables. By evaluating f for some input x, an output $\hat{y}$ is obtained as $\hat{y} = f(x, \theta)$. This operation is referred to as a forward pass [14]. A common application in which neural networks are employed is image classification, where there exists a set of classes C with size n. Every input image x (i.e. a two- or three-dimensional array) has a certain classification $c_i \in C$. The output $\hat{y}$ from such a network may be a vector with n predicted probabilities, such that x has classification c_i with probability $\hat{y}_i$, $i = 1, 2, \ldots, n$.

An activation function determines the output of a neuron based on the input [14]. A neuron is fired if the value of the activation function passes a certain threshold. The value for determining the threshold is known as a bias, and every neuron's activation function has one associated with it. In addition, every connection in a neural network has a weight which determines its strength. The weights and biases of a network are commonly denoted as W and B respectively. These are included in θ.

The output $\hat{y}$ of a neural network is compared to a ground truth y through a loss function. A loss function can be defined as:

$$L(y, \hat{y}) = \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{2.5}$$

where $\hat{y} = f(x, \theta)$. Since $\hat{y}$ depends on the trainable variables θ, one can attempt to minimize the loss function $L(y, \hat{y})$ in order to train the network. This is done by using an optimization algorithm, such as gradient descent. Optimization algorithms are further expanded on in Section 2.5.2. $L(y, \hat{y})$ is differentiated with the values of θ, which are then updated accordingly. This update can occur after every training example or after a so-called batch of training examples have been processed. The network is then updated based on the mean of the loss function for all examples in a batch. With a greater batch size, more examples are considered simultaneously during backpropagation. The magnitude with which the θ values are adjusted is based on a *learning rate*. The whole process described in this paragraph is known as a backward pass, or backpropagation, and it is during this time that a neural network is learning [15].

2.5.1 Loss Functions

A loss function, sometimes called a cost function, quantifies the disparity between the output of a network and its ground truth, also known as the target [16]. In the following sections, we present commonly utilized loss functions in deep learning, as well as one that was built for this master's thesis.

Mean Absolute Error

The mean absolute error (MAE) loss function calculates the absolute difference between the output values and the target values. It is given by Equation (2.6), where y is the target and $\hat{y}$ is the output.

$$L_{\text{MAE}}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{2.6}$$

Specifically, with the case of having *images* as inputs and outputs in a (fully convolutional) neural network, the MAE loss function can be expressed as:

$$L_{\text{MAE}}(y, \hat{y}) = \frac{1}{mn} \sum_{i=1}^{n} \sum_{j=1}^{m} |y_{ij} - \hat{y}_{ij}| \tag{2.7}$$

where n and m correspond to the width and height of the images.

This loss function is famously referred to as the L_1*-loss*, which is what we will call it for the remainder of the thesis.

Mean Squared Error

The mean squared error (MSE) loss function calculates the squared difference between the output values and the target values, and is given by Equation (2.8).

$$L_{\text{MSE}}(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.8}$$

Similarly to the L_1-*loss*, for the case of having images as inputs and outputs in a (fully convolutional) neural network, the MSE loss function can be expressed as:

$$L_{\text{MSE}}(y, \hat{y}) = \frac{1}{mn}\sum_{i=1}^{n}\sum_{j=1}^{m}(y_{ij} - \hat{y}_{ij})^2 \tag{2.9}$$

Also similarly to the L_1-*loss*, the MSE loss is commonly known under another name, namely the L_2-*loss*. We will use this name to refer to the function going forward. In terms of image processing, the L_2-loss can correlate poorly with perceived image quality according to Zhao et al. [17], and is often outperformed by the L_1-loss function.

Huber Loss

The Huber loss, L_δ, got its name from Peter Huber [18], the person who defined the function. It is calculated as shown in Equation (2.10). For arguments below a certain threshold δ, L_δ behaves as a squared error loss function, and above the threshold as an absolute error loss function.

$$\begin{aligned} f_{cond}(a, b) &= \begin{cases} \frac{1}{2}(a-b)^2 & \text{if } |a-b| \leq \delta \\ \delta(|a-b| - \frac{1}{2}\delta^2) & \text{if } |a-b| > \delta \end{cases} \\ L_\delta(y, \hat{y}) &= \sum_{i=1}^{n} f_{cond}(y_i, \hat{y}_i) \end{aligned} \tag{2.10}$$

L_1 Pyramid Loss

The loss function L_P defined in Equation (2.11) was specifically designed for this thesis. The idea was to downsample the images and incorporate the loss of the downsampled images with the standard L_1-*loss*. By downsampling, noise is reduced while most objects should remain but appear smaller. Based on this assumption, this loss function was designed as an attempt to punish networks for removing objects rather than noise. We refer to it as the L_1-*pyramid loss*.

$$L_p(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| + \frac{1}{n}\sum_{i=1}^{n}|y_{i_2} - \hat{y_{i_2}}| + \frac{1}{n}\sum_{i=1}^{n}|y_{i_4} - \hat{y_{i_4}}| \tag{2.11}$$

Here y_{i_2} and $\hat{y}_{i_2}$ are downsampled by a factor 2 from y_i and $\hat{y}_i$ respectively, whereas y_{i_4} and $\hat{y}_{i_4}$ are downsampled by a factor 4.

2.5.2 Optimization

In machine learning, an optimization algorithm is often used to find the minimum of a loss function in order to optimize a model. This section covers a number of different optimization algorithms.

Gradient Descent

Gradient decent is an iterative algorithm for finding a (local) minimum [16]. Given some initial values for $x = x_1, x_2, ..., x_n$ in the function $F(x)$, the gradient vector is defined as:

$$\nabla F = \left(\frac{\partial F}{\partial x_1}, \frac{\partial F}{\partial x_2}, ..., \frac{\partial F}{\partial x_n} \right)^T \tag{2.12}$$

By defining the vector of changes for the input parameters x, as $\Delta x = (\Delta x_1, \Delta x_2, ..., \Delta x_n)^T$, the changes in F can be expressed as:

$$\Delta F \approx \nabla F \cdot \Delta x. \tag{2.13}$$

By choosing $\Delta x = -\alpha \nabla F$ and substituting Δx in Equation (2.13), we get that:

$$\Delta F \approx \nabla F \cdot -\alpha \nabla F = -\alpha ||\nabla F||^2 \tag{2.14}$$

where it is guaranteed that $\Delta F \leq 0$ as long as the learning rate α is positive. So, for every step of gradient descent, a new parameter vector x' is computed according to:

$$x \rightarrow x' = x - \alpha \nabla F. \tag{2.15}$$

This way, F will decrease until it descends close to a local minimum. How close depends on the learning rate, a smaller learning rate will most likely make the algorithm descend slower, but eventually reach a value closer to the minimum.

Momentum

Gradient descent can be regarded as a ball slowly rolling down a hill until it reaches the bottom. Something that exists in the real world but is lacking in gradient descent is momentum. By rolling a ball down a hill it would not simply stop at the bottom, because of its momentum the ball would continue rolling until halted by friction. The same idea may be applied in optimization algorithms and is exactly the case in *momentum-based gradient descent* [16]. By introducing a velocity variable v_i for each parameter as well as a friction term β, a new parameter vector is calculated after each iteration as can be seen in Equation (2.16).

$$\begin{aligned} v &\rightarrow v' = \beta v - \alpha \nabla F \\ x &\rightarrow x' = x + v' \end{aligned} \tag{2.16}$$

Adam Optimizer

A more modern optimization algorithm is Adam, conceived in 2014 by D. Kingma and J. Ba [19]. A description of the algorithm can be seen in Equation (2.17), where f is the objective function to minimize, α is the learning rate, β_1 and β_2 are the exponential decay rates of first- and second-moment, and ϵ is a small constant.

$$\begin{aligned} t &\leftarrow t + 1 \\ g_t &\leftarrow \nabla_\theta f_t(\theta_{t-1}) \\ m_t &\leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \\ v_t &\leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \\ \hat{m}_t &\leftarrow \frac{m_t}{(1 - \beta_1^t)} \\ \hat{v}_t &\leftarrow \frac{v_t}{(1 - \beta_2^t)} \\ \theta_t &\leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{(\sqrt{\hat{v}_t} + \epsilon)} \end{aligned} \tag{2.17}$$

The gradient vector of the function f for time step t is assigned to g_t. m_t and v_t are the moving averages of the gradients and squared gradients respectively, which approximates the mean and variance of the gradients. In the final step, the trainable variables θ are updated.

2.5.3 Convolutional Neural Networks

In the same way that an ANN is an attempt to model how the human brain learns, the idea with convolutional neural networks (CNNs) is to mimic current belief of how our eyes and brain interpret images. A typical CNN consists predominantly of convolutional and pooling layers [20]. These allow for feature extraction, which is the process of breaking down an input into smaller elements. A segment of fully connected layers which maps features to classes, is put on top of the feature extraction mechanism.

CNNs are said to be of different dimensions, referring to how the convolutional operations are used. To give a few examples, typically one-dimensional networks are used for classification of sound whereas two-dimensional networks are better suited for image classification. A two-dimensional convolutional layer takes as input X a two- or three-dimensional matrix. An output Y_i, or feature map, is obtained through:

$$Y_i = \phi(X * K_i + B_i), \; i = 0, 1, \ldots, n \tag{2.18}$$

where ϕ is some activation function, K_i is a kernel, and B_i its respective bias. The number of outputs n is commonly referred to as the output channels. The kernel contains what is referred to as the weights of the layer. The output is added together with a bias and passes as input to an activation function, often a ReLU, see Figure 2.2. This function outputs zero for negative inputs, but increases linearly for positive values. There are variations

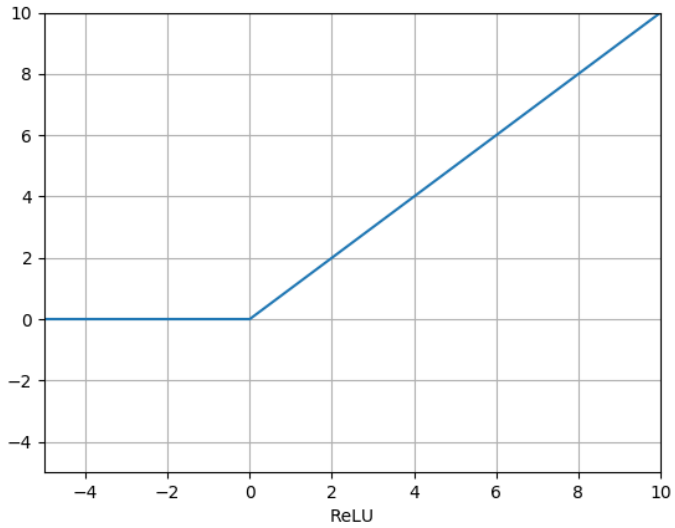


Figure 2.2: The ReLU function.

such as the leaky ReLU, which slowly decreases linearly for inputs less than zero based on a parameter α. Using ReLUs is a way to introduce nonlinearity in the network; a necessity for carrying out complex tasks such as classification.

The way a layer convolves with the input depends not only on the weights and biases, but also the following parameters:

1. **Padding:** Determines whether the edges of the matrix should be padded with zeros, and to what extent. Having zero-padding set to one means that a single border of zeroes will be placed around the matrix. Thus, the feature maps may be forced into the same dimensions as the input [21].

2. **Stride:** The distance a kernel skips over the input between each sum of element-wise multiplications. By increasing the stride in a convolutional layer, its output's dimensions are reduced.

Pooling layers downsample the input based on the stride and kernel size. An element in the resulting matrix R of a max-pooling layer using input matrix X with dimensions $m \times n$, kernel size 2×2, and stride s, is defined as:

$$R_{i,j} = \max\left(x_{si,sj}, x_{si+1,sj}, x_{si,sj+1}, x_{si+1,sj+1}\right), \; i = 1, 2, \ldots, \left\lceil \frac{m}{s} \right\rceil \text{ and } \; j = 1, 2, \ldots, \left\lceil \frac{n}{s} \right\rceil. \quad (2.19)$$

Convolutional layers may use particular kernels for extracting features in images, such as edges. Pooling layers ensure invariant feature extraction with regards to scale, to some degree. This is especially useful for classification problems, since the scale of objects should not affect model accuracy.

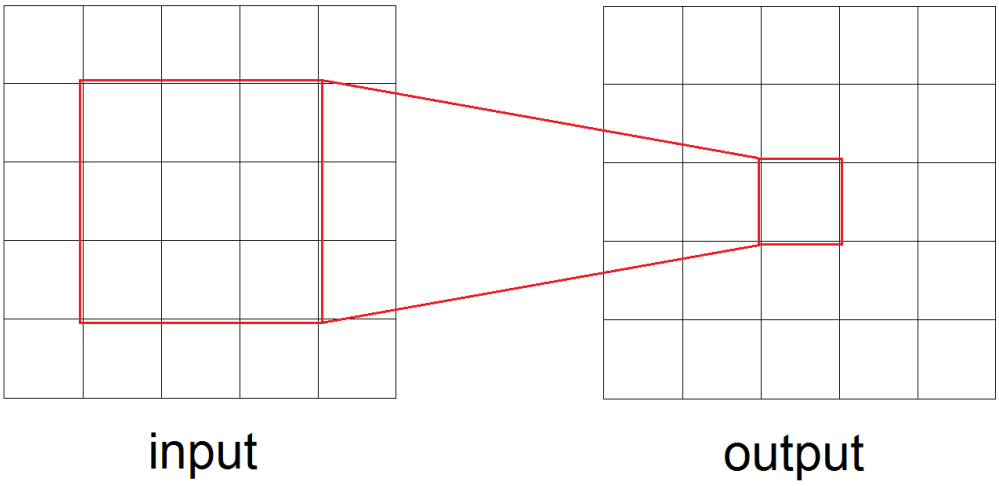


Figure 2.3: Receptive field in a CNN. The left matrix represents an input image, and the right matrix an output image. The area of the red box in the input indicates what input pixels an output pixel could depend on, as an example.

2.5.4 Receptive Field

All neurons in a layer are each connected to every neuron of the following layer in a standard ANN. This means that all output neurons are individually dependent on the entire input, unless specific weights are set to zero.

In a CNN, the output elements only depend on a certain area of the input. This area is defined as the *receptive field*, see Figure 2.3. The size of the receptive field is determined by the number of layers in addition to the sizes of the different kernels.

2.5.5 Fully Convolutional Neural Networks

A fully convolutional neural network (FCNN) doesn't contain the classifier segment typically found in CNNs. Instead, an upsampling part is introduced, which allows for pixel-wise predictions [22]. This architectural alteration leads to a positive side-effect. It allows for FCNNs to take input of variable size, as long as the number of dimensions remain constant.

So-called *transposed convolutions* are commonly used for upsampling in FCNNs, in favor of classical interpolation methods such as nearest neighbor or bilinear interpolation. By using transposed convolutions, the weights and biases for the upsampling becomes part of the learning mechanism. This means that the network updates the way upsampling is done, based on the loss function and the optimization algorithm.

A standard convolution is expressed as $Y = X * K$. As an example, with a stride equal to one, the result Y_C from a convolution with a 4×4 matrix X_C and 3×3 kernel K_C may be expressed as:

$$Y_C = X_C * K_C = \begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix} * \begin{bmatrix} k_{00} & k_{01} & k_{02} \\ k_{10} & k_{11} & k_{12} \\ k_{20} & k_{21} & k_{22} \end{bmatrix} \tag{2.20}$$

The square matrix X with dimensions $m_X \times n_X$ can be flattened into a column vector X_F with dimensions $m_X n_X \times 1$. By transforming K into a kernel matrix M with dimensions $m_Y n_Y \times m_X n_X$, where $m_Y \times n_Y$ are the dimensions of Y, the expression $Y = M \cdot X_F$ is obtained. In a practical example this may look like:

$$Y_C = M_C \cdot X_{F_C} = \begin{bmatrix} k_{00} & k_{01} & k_{02} & 0 & k_{10} & k_{11} & k_{12} & 0 & k_{20} & k_{21} & k_{22} & 0 & 0 & 0 & 0 & 0 \\ 0 & k_{00} & k_{01} & k_{02} & 0 & k_{10} & k_{11} & k_{12} & 0 & k_{20} & k_{21} & k_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & k_{00} & k_{01} & k_{02} & 0 & k_{10} & k_{11} & k_{12} & 0 & k_{20} & k_{21} & k_{22} & 0 \\ 0 & 0 & 0 & 0 & 0 & k_{00} & k_{01} & k_{02} & 0 & k_{10} & k_{11} & k_{12} & 0 & k_{20} & k_{21} & k_{22} \end{bmatrix} \cdot \begin{bmatrix} x_{00} \\ x_{01} \\ x_{02} \\ x_{03} \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{20} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{30} \\ x_{31} \\ x_{32} \\ x_{33} \end{bmatrix} \tag{2.21}$$

Given a square matrix Z that is flattened into a column vector Z_F, a transposed convolution Y_{TC} can be expressed as $Y_{TC} = M^T \cdot Z_F$, where M^T is a transposed kernel matrix [23]. An example of this is given in Equation (2.22).

$$Y_{TC} = M_C^T \cdot Z_{F_C} = \begin{bmatrix} k_{00} & 0 & 0 & 0 \\ k_{01} & k_{00} & 0 & 0 \\ k_{02} & k_{01} & 0 & 0 \\ 0 & k_{02} & 0 & 0 \\ k_{10} & 0 & k_{00} & 0 \\ k_{11} & k_{10} & k_{01} & k_{00} \\ k_{12} & k_{11} & k_{02} & k_{01} \\ 0 & k_{12} & 0 & k_{02} \\ k_{20} & 0 & k_{10} & 0 \\ k_{21} & k_{20} & k_{11} & k_{10} \\ k_{22} & k_{21} & k_{12} & k_{11} \\ 0 & k_{22} & 0 & k_{12} \\ 0 & 0 & k_{20} & 0 \\ 0 & 0 & k_{21} & k_{20} \\ 0 & 0 & k_{22} & k_{21} \\ 0 & 0 & 0 & k_{22} \end{bmatrix} \cdot \begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} \tag{2.22}$$

Let m_T and n_T denote the number of rows and columns of M^T. If Z is interpreted as an image with one channel, by having $m_T = c^2 n_T$, Z is upsampled with a factor c.

FCNNs are remarkably useful for image processing tasks since such networks can output an entire processed image. Hence an FCNN can be viewed as a single, incredibly complex, image filter.

2.6 Related Work

This section presents previous work in noise filtering with neural networks, related to this master's thesis.

U-Net

In 2015 an article by Ronneberger et al. [24] was published, describing a novel FCNN architecture known as a U-net. It had shown excellent results in image segmentation tasks. In a U-net, information is passed through connections symmetrically from earlier to later layers, so that information from earlier contexts is propagated further in the network. This leads to higher performance in for example image segmentation. An example of a U-net structure is illustrated in Figure 2.4. It has also been adopted in multiple other research projects [1] [25].

Learning to See in the Dark

In mid-2018, Chen et al. [1] built a model which produced excellent results for image processing of low-light images. They assembled a new dataset containing short-exposure images with corresponding long-exposure ones, of the exact same physical scene. The short-exposure versions were used to successfully simulate low-light conditions and served as input for their network. The long-exposure images being bright and clear served as ground

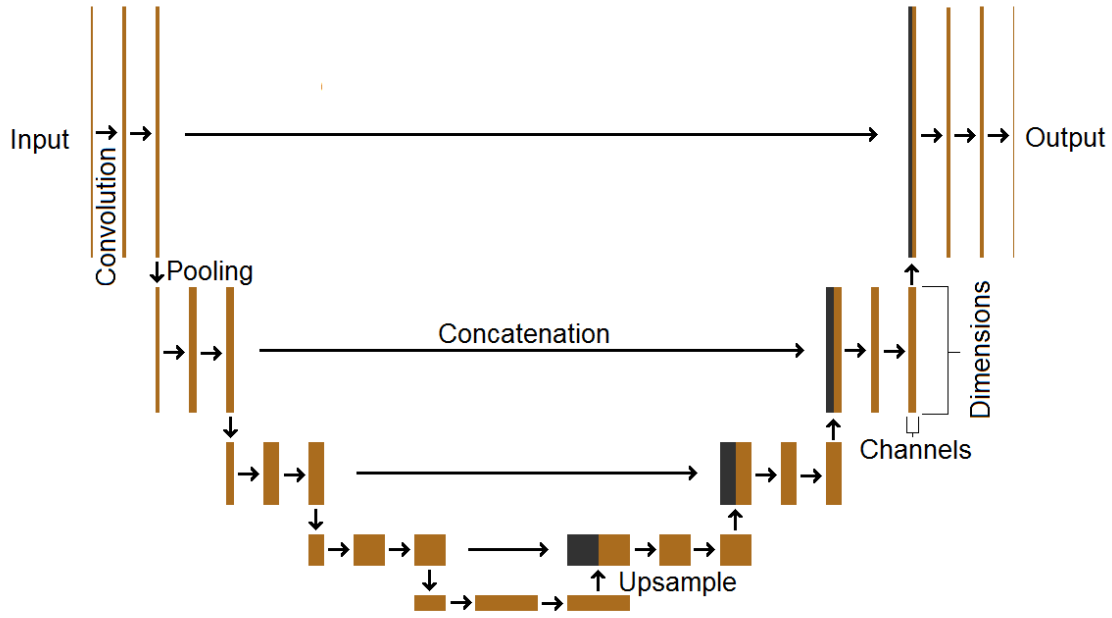


Figure 2.4: U-net architecture.

truths. After testing several networks, a conclusion was reached that an architecture like the FCNN U-net structure [24] produced the best results. When comparing this model with a traditional pipeline as well as with BM3D denoising, the results were indeed favorable in regard to the U-net.

Deep Burst Denoising

A recent type of recurrent FCNN-architecture was presented in late 2017 by Godard et al. [4]. Their approach to obtain high-quality low-light images involved capturing bursts of short-exposure photos, as increasing a camera's shutter speed may lead to undesirable artifacts. These short-exposure photos would then be intelligently combined to obtain satisfactory quality. However, since short exposure leads to extensive noise, the image bursts had to be denoised first. The recurrent FCNN model was used for exactly this. The achieved results were better than state-of-the-art methods.

Deep Joint Demosaicing and Denoising

In 2016, Gharbi et al. [10] developed a model for both demosaicing and denoising images. Perhaps the most exciting point this article makes is the importance of having good data in quantity. A network was first trained on around 2.3 million images. Poorly demosaiced patches were then extracted from the training results, and around 2.5 million were selected for a second training. The model's performance reached some of the best results recorded at that time, while doing inference immensely faster compared to its counterparts.

Chapter 3
Method

We have built three different FCNN architectures that were trained and evaluated with different configurations. They were all implemented in Python using the TensorFlow framework [26]. This chapter describes the designs of our models, how our dataset was compiled and processed, and what hardware was used during training. First, however, we present the central idea with the thesis.

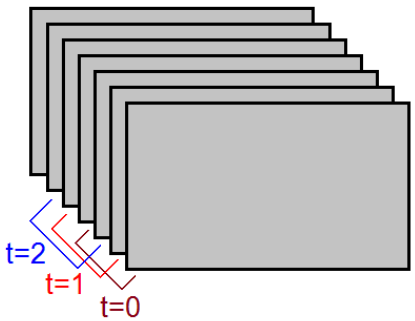


Figure 3.1: Sliding windows over a sequence of frames at three different time steps t, where $T = 3$.

Noise Filtering Video

We define a *scene* as multiple adjacent frames in a video. A *sliding window* with a certain length T may be applied over a scene, grouping T frames together, see Figure 3.1. T is referred to as the *sequence length*. As the sliding window moves by one frame for every new example, having S images in a scene would result in $S - (T - 1)$ examples. Our models take a video sequence as input and outputs a single frame according to the following formalization.

We denote a noisy input sequence with T adjacent frames as $\boldsymbol{x_T} = (x_1, \ldots, x_T)$ and a corresponding clean, ground truth sequence as $\boldsymbol{y_T} = (y_1, \ldots, y_T)$. Here, x_i and y_i denotes single frames. A frame is assumed to share information with a variable number of adjacent frames. In other words, pixel areas in adjacent frames have luminous, chromatic, and structural similarities. Let y_m be the middle frame of $\boldsymbol{y_T}$, where $m = \lceil \frac{T}{2} \rceil$. For clarity, if there is an even number of frames, the frame right before the middle cut is chosen as the middle frame. The goal for our models is to learn a mapping $f : \boldsymbol{x_T} \mapsto \hat{y}$ where $\hat{y}$ is a single output frame that corresponds to the denoised version of x_m. As the model is learning, $\hat{y}$ approaches y_m . The clean middle frame y_m is referred to as the ground truth.

Suppose that we have two clean neighboring frames, y_1 and y_2, such that $y_1 \approx y_2$. If there is some loss of information in y_1 such that we observe $x_1 = y_1 + \mu_1$ (y_1 and μ_1 are unknown), where x_1 is said to be noisy, μ_1 can be expressed as $\mu_1 \approx x_1 - y_2$. Thereby, y_1 can be recovered. Consider a sequence $\boldsymbol{y_T}$ of clean frames respectively affected by T independent noise terms $\mu_1, \mu_2, \ldots, \mu_T$, of some unknown distribution so that we can observe the sequence $\boldsymbol{x_T}$. We express a recovered, clean frame as $\hat{y} = f(\boldsymbol{x_T}; \theta)$, where $\hat{y}$ approaches y_m, and θ is the parameter vector of trainable variables that are updated during backpropagation.

3.1 Architectures

The layers and filters used for the different architectures are shown in Table 3.1. As can be seen, there are four levels of this U-net, meaning that the networks will contract and expand four times each. Every convolutional layer is followed by a leaky ReLU with $\alpha = 0.2$. The input to each convolutional layer is zero-padded so that the height and width (and depth for 3D) stay the same. The pooling layers downsample images by a factor two, and analogously the upsample layers upsample the images by the same factor. The loss L as in Equation (2.5), takes for all our networks the single frames y_m and $\hat{y}$ as arguments.

3.1.1 Two-Dimensional Sequential Input FCNN

The first idea for an architecture was to craft an extension of U-Net [24]. The U-net design have shown great success, Chen et al. [1] built a model with a U-net-like architecture that displayed excellent performance in low-light denoising. Being heavily inspired by this paper, a slightly modified version of their network was chosen as a starting point. Our modified network takes a sequence of variable length T as input instead of a single image. Additional differences are that the original network takes input images in raw format, and has 12 output channels in the final layer, whereas our network uses RGB images and have 3 final output channels. Our version with $T = 1$ will therefore serve as the state-of-the-art, completely spatial noise filter mentioned in our first goal. This architecture will in this thesis be referred to as *2D FCNN*.

The dimensions of the input take the shape $B \times H \times W \times (C \cdot T)$, where B denotes the batch size, H the height, W the width, and C the color channels, for an input image with dimensions $H \times W \times C$. Since RGB images are used, $C = 3$. The channels of the input images in the sequence are stacked together in chronological order. Note that the model is

Table 3.1: The layers and the number of filters for the architectures. The layers follow the U-net shape shown in Figure 2.4.

Layer	Input	Output channels	Function
input	-	$3 \cdot T$	-
2×conv_1	input	32	Convolution 3×3 (×3 for 3D)
pool_1	conv_1	32	Max Pooling 2×2 (×2 for 3D)
2×conv_2	pool_1	64	Convolution 3×3 (×3 for 3D)
pool_2	conv_2	64	Max Pooling 2×2 (×2 for 3D)
2×conv_3	pool_2	128	Convolution 3×3 (×3 for 3D)
pool_3	conv_3	128	Max Pooling 2×2 (×2 for 3D)
2×conv_4	pool_3	256	Convolution 3×3 (×3 for 3D)
pool_4	conv_4	256	Max Pooling 2×2 (×2 for 3D)
2×conv_5	pool_4	512	Convolution 3×3 (×3 for 3D)
upsample_1	conv_5 ⊕ conv_4	256	Transposed convolution and concatenate
2×conv_6	upsample_1	256	Convolution 3×3 (×3 for 3D)
upsample_2	conv_6 ⊕ conv_3	128	Transposed convolution and concatenate
2×conv_7	upsample_2	128	Convolution 3×3 (×3 for 3D)
upsample_3	conv_7 ⊕ conv_2	64	Transposed convolution and concatenate
2×conv_8	upsample_3	64	Convolution 3×3 (×3 for 3D)
upsample_4	conv_8 ⊕ conv_1	32	Transposed convolution and concatenate
2×conv_9	upsample_4	32	Convolution 3×3 (×3 for 3D)
conv_10	conv_9	3	Convolution 1×1 (×1 for 3D)

not explicitly told which dimensions belong to which image - it is simply given a number of channels in the final dimension $C \cdot T$.

The number of weights that this network has for each layer is given by the following equation:

$$w = K_h \cdot K_w \cdot C \cdot F \tag{3.1}$$

where K_h and K_w is the height and the width of the kernel, C is the number of input channels, and F is the number of output filters.

3.1.2 Three-dimensional Sequential Input FCNN

We wanted to build an architecture that more explicitly filters temporally, as this was the focus of the thesis. Our ambition materialized in a switch to a 3D-convolutional network [27], here referred to as *3D FCNN*. In this type of network, a new *depth* dimension is included in the convolutions. Hence the kernel dimensions must be extended from 3×3 to $3 \times 3 \times 3$. Thus, for every step the kernel takes, input from 3 frames are taken into account. This network therefore convolves both the spatial and the temporal dimensions explicitly. The max-pooling layers consequently not only decrease the spatial dimensions, but also the depth dimension. As an example, if the depth dimension of the input initially was 16,

it would after the first layer be decreased to 8, then 4, and so on. By upsampling through transposed convolutions, the depth dimension increases again so that the penultimate layer has a depth of 16. The initial depth dimension is the same as the sequence length T. Instead of having the channel dimension be $(T \cdot C)$ as was the case with the 2D-convolution networks, it is now simply C. With an input image of dimensions $H \times W \times C$, the input would have the shape $B \times T \times H \times W \times C$.

The number of weights for each layer is given by the following equation:

$$w = K_h \cdot K_w \cdot K_d \cdot C \cdot F \tag{3.2}$$

where K_d is the depth of the kernel and the rest of the parameters are the same as in Equation 3.1.

Three-Level Variant

The architecture of the 3D-convolution network is, aside from the type of convolution, equivalent to the 2D FCNN as they both contain the same number of layers and filters. We did however introduce a variant where the bottom convolution and pooling layers, i.e. the lowest level in the U-net, were removed, but the number of filters were doubled. Consequently, the same number of weights were kept, despite the spatial dimensions being downsampled one less time. We refer to this variant as the 3D FCNN having *3 levels*.

Our idea was to combine the aforementioned architecture with halved values for H and W while keeping T constant. As a result, the spatial dimensions at the bottom of the U-net stay the same as in the standard four level architecture, while the temporal dimension is doubled. The network would thereby use a larger proportion of temporal information relative to other configurations.

3.1.3 Combined Three- and Two-dimensional FCNN

The final architecture that we wanted to try was to keep the U-net structure, similarly to the 2D FCNNs and 3D FCNNs, but use different strategies when contracting and expanding the U-net. 3D convolutions were used when contracting and 2D convolutions for expanding. This net required one additional convolution for each level in the U-net to transform the downsampled 3D feature maps into the 2D domain, in order to perform the concatenation with the upsampled feature maps correctly. The idea was to use as much of the temporal information as possible when contracting, but to expand with mainly the spatial information in focus. We hypothesized that utilizing the spatial information when upsampling could increase the image quality compared to doing this temporally. It is referred to as *Combined 3D and 2D FCNN*.

3.2 Data

Our goal when creating a dataset was to assemble a collection of relatively short video sequences. We sought after data that would be adequately difficult to denoise with regard to temporal change. In other words, there should be a reasonable difference between frames so that the network is challenged but not overwhelmed. Constant smooth movement was sought after, since we hypothesized that having a moderate amount of difference between frames would allow our models to learn how to process temporal information. This would be in contrast to only considering the spatial dimensions. Video sequences should therefore only consist of frames from the same cut, and not have large objects travelling with unnatural speed, such as a hand moving directly in front of the sensor.

The videos used for the dataset were provided by Axis Communications. They contain scenes in raw format filmed in different settings by Axis personnel. As movement was occasionally sparse in these videos, interesting crops of 100 frames-scenes were selected based on magnitude of difference between temporally adjacent areas. In total, 153 such scenes were selected which resulted in 15300 frames. Other versions of these images were generated with noise applied. Both versions were pre-processed by a practical image pipeline and extracted right after the demosaicing step, but before any denoising transpired. After the extraction, the clean versions of the images served as ground truth and the images with noise served as input.

We placed approximately 80% of the data into a training set, 10% into a validation set, and 10% into a test set. The scenes were kept together as units but distributed randomly. The selection was manually corrected afterwards to create the most diverse and challenging validation and test set as possible. The validation set was not switched or changed throughout training. We deemed this to be the best validation method all things considered, at least to our knowledge. Our models were evaluated between each epoch on the validation set, to make sure that they did not overfit to the training data. An indication of overfitting is if the training loss continues to decrease while the validation loss is increasing. The test set was used for the final evaluation of the trained model.

3.2.1 Data from the State-of-the-art Camera

The images in this thesis that show the produced output from the state-of-the-art camera were extracted directly after the denoising step in the image pipeline, and then minimally post-processed. The same post-processing was applied to the denoised results from the networks. The entire image pipeline is however designed with the idea that images go through more extensive post-processing. Excluding these steps may therefore produce output which diverges from what the final output of the camera would look like. It was however the best method we could provide in order to exclusively compare denoising results.

3.3 Noise Model

The goal of our noise model was to simulate noise from a real camera sensor, as described in Section 2.2. To do this correctly, the synthetic noise was added to the images in raw format. This is essential since real noise occurs when images are captured by the sensor. Before the generation of noise, the images were scaled to have values between 0 and 1. Because of the black level, the images actually only had pixel values between the scaled black level and 1.

To correctly generate the noise, the pixel values were first multiplied by the given full well so that the maximum possible pixel value was set to the full well. This is crucial since it simulates the range of values the images would obtain after being captured by the sensor, as described in Section 2.2.1, and since the noise distribution depends on what the pixel values are. For each pixel, the shot noise was then simulated by collecting a sample from a Poisson distribution with the pixel value as the expected value, and replacing the pixel value with this sample. The read noise was simulated by adding a sample drawn from a Gaussian distribution. Finally, the pixel values were scaled back to the same range they were in before the noise was added, except that the values could now go below the black level as well. The images were then put into the pipeline.

With this model we could easily control the extent of shot noise and read noise in our data. To simulate real low-light conditions, the full well capacity was set to 25 and the standard deviation for the Gaussian distribution of the read noise was set to 1. The mean value for the read noise was set to 0, as stated in Section 2.2.2.

3.4 Hardware

Axis Communications provided hardware for training and testing our models. Two different setups were used. The first had a single Nvidia GeForce GTX 1080 Ti GPU with 10 GBs of memory as well as an Intel Core i7-8700K CPU with 3.70GHz clock rate. The second included four GPUs: one Nvidia GeForce GTX Titan V and three Nvidia GeForce GTX Titan X, as well as an Intel Core i7-6850K CPU with 3.60GHz clock rate.

3.5 Metrics for Measuring Performance

Comparing the performance of different image processing algorithms is a difficult task. Appropriate metrics have to be defined, which accurately measure the desired effect. When comparing a single image to another, PSNR is a good measurement and will be used for this purpose. Videos are more complex, where the flow and continuity of the video is important. This will be measured as something we call *temporal coherence*. Our evaluation of a network will take both PSNR and temporal coherence into account.

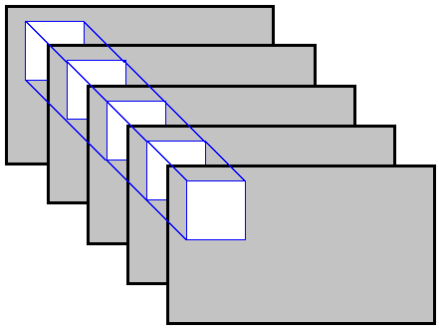


Figure 3.2: A patch cropped from a sequence of images with length $T = 5$.

3.5.1 Temporal Coherence

We define temporal coherence as continuity and flow when watching a video. We noticed that a high PSNR does not necessarily equate to satisfying temporal coherence. A video could for example look fluttery even though the frame by frame comparison showed exceptional results. For this reason, we chose to measure performance using temporal coherence in addition to PSNR. Objectively measuring temporal coherence in videos is however difficult, and there does not seem to be a good enough metric for doing this. When stating how good a video's temporal coherence is, we refer to how little flutter can be seen when visually analyzing the videos. This measurement will therefore be subjective.

3.6 Standard Training Setup

For every sequence, a random area of $P \times P$ pixels in height and width was selected and cropped to serve as input, see Figure 3.2. We refer to P as the patch size. The data was shuffled and new patches were cropped for every epoch. The data was augmented so that the crop had a 50% chance to be flipped horizontally and a 50% chance to be flipped vertically. This augmentation of the input data yielded more diverse training examples, which can often help with counteracting overfitting.

When deciding how many epochs to run the training sessions for, we had to make a trade-off between time and performance. We trained a network for 300 epochs (which when multiplied by $15300 \cdot 0.8$ becomes 3672000 training examples when $T = 1$) to see if the performance would eventually start to decrease, see Figure 3.3. The increase in performance remained stable, despite some temporary decline, and the network did not overfit to the training data. The time required to train for 300 epochs was however substantial for some of the networks, especially the 3D FCNNs. With $T = 16$ this took about two weeks. Though it is likely that the performance of the models would increase by further training, 300 epochs was chosen as the standard training time due to time constraints of this thesis.

(a) Input

(b) Ground truth

(c) Epoch 1
PSNR: 28.53

(d) Epoch 5
PSNR: 29.76

(e) Epoch 10
PSNR: 29.78

(f) Epoch 20
PSNR: 31.06

(g) Epoch 50
PSNR: 32.32

(h) Epoch 100
PSNR: 31.51

(i) Epoch 150
PSNR: 32.76

(j) Epoch 200
PSNR: 32.73

(k) Epoch 250
PSNR: 32.86

(l) Epoch 300
PSNR: 32.98

Figure 3.3: How the performance changed during 300 epochs of training the 3D FCNN with $T = 16$, $P = 128$, and 3 levels.

Chapter 4
Results

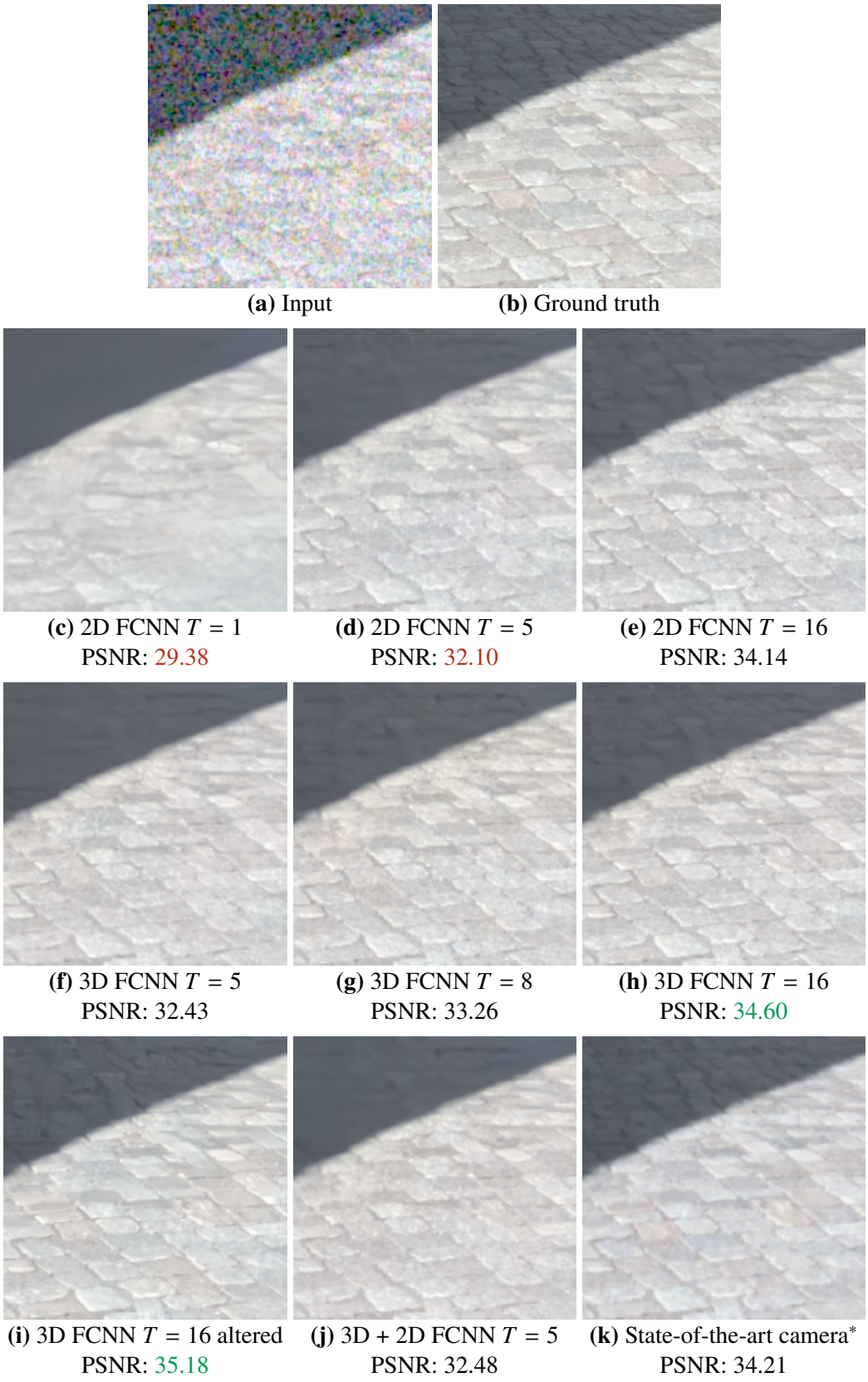
Our experiments were conducted using the standard training setup explained in Section 3.6. Unless stated otherwise in the sections below, the parameter selection followed Table 4.1. The results presented here were produced through inference of our models on the test set, which contained 1530 frames.

Table 4.1: Default parameter selection. Used for all experiments presented if nothing else is stated.

Parameter	Value
Number of epochs	300
Batch size	4
Loss function	L_1
Optimizer	Adam, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$
Learning rate	10^{-4} initially, reduced by a factor 10 after 150 epochs
Patch size	256×256 pixels

The following four pages show our results from denoising four frames in video sequences with different characteristics. The first image on every page is the noisy input, followed by its corresponding ground truth. We show performance for the different configurations on our three architectures as well as the denoising segment in the image pipeline of the state-of-the-art camera*. The first frame was produced by denoising an area in a video sequence with no movement, meaning very little difference between adjacent frames. The second frame features a bush affected by wind, which results in small movement. The third and fourth frames show denoising of a frame in a sequence that depicts a person moving with different speeds.

*For how the results of the state-of-the-art camera were produced, see Section 3.2.1.



(a) Input **(b)** Ground truth

(c) 2D FCNN $T = 1$ PSNR: 29.38 **(d)** 2D FCNN $T = 5$ PSNR: 32.10 **(e)** 2D FCNN $T = 16$ PSNR: 34.14

(f) 3D FCNN $T = 5$ PSNR: 32.43 **(g)** 3D FCNN $T = 8$ PSNR: 33.26 **(h)** 3D FCNN $T = 16$ PSNR: 34.60

(i) 3D FCNN $T = 16$ altered PSNR: 35.18 **(j)** 3D + 2D FCNN $T = 5$ PSNR: 32.48 **(k)** State-of-the-art camera* PSNR: 34.21

Figure 4.1: The performances when denoising a frame in a still video sequence. Figure 4.1i refers to the network with $P = 128$ and 3 levels.

(a) Input **(b)** Ground truth

(c) 2D FCNN $T = 1$
PSNR: 26.16

(d) 2D FCNN $T = 5$
PSNR: 28.85

(e) 2D FCNN $T = 16$
PSNR: 29.20

(f) 3D FCNN $T = 5$
PSNR: 29.02

(g) 3D FCNN $T = 8$
PSNR: 29.41

(h) 3D FCNN $T = 16$
PSNR: 29.89

(i) 3D FCNN $T = 16$ altered
PSNR: 29.92

(j) 3D + 2D FCNN $T = 5$
PSNR: 28.93

(k) State-of-the-art camera*
PSNR: 26.38

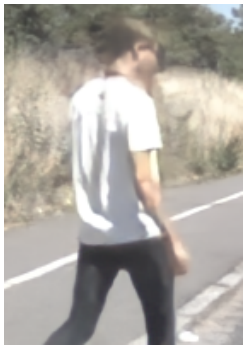
Figure 4.2: The performances when denoising a frame in a sequence with some movement. Figure 4.2i refers to the network with $P = 128$ and 3 levels.

(a) Input

(b) Ground truth

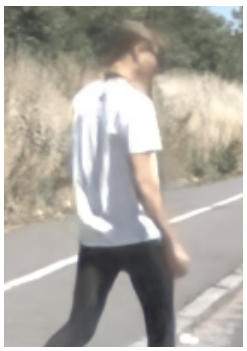
(c) 2D FCNN $T = 1$
PSNR: 29.14

(d) 2D FCNN $T = 5$
PSNR: 30.10



(e) 2D FCNN $T = 16$
PSNR: 30.06

(f) 3D FCNN $T = 5$
PSNR: 30.42



(g) 3D FCNN $T = 8$
PSNR: 30.68

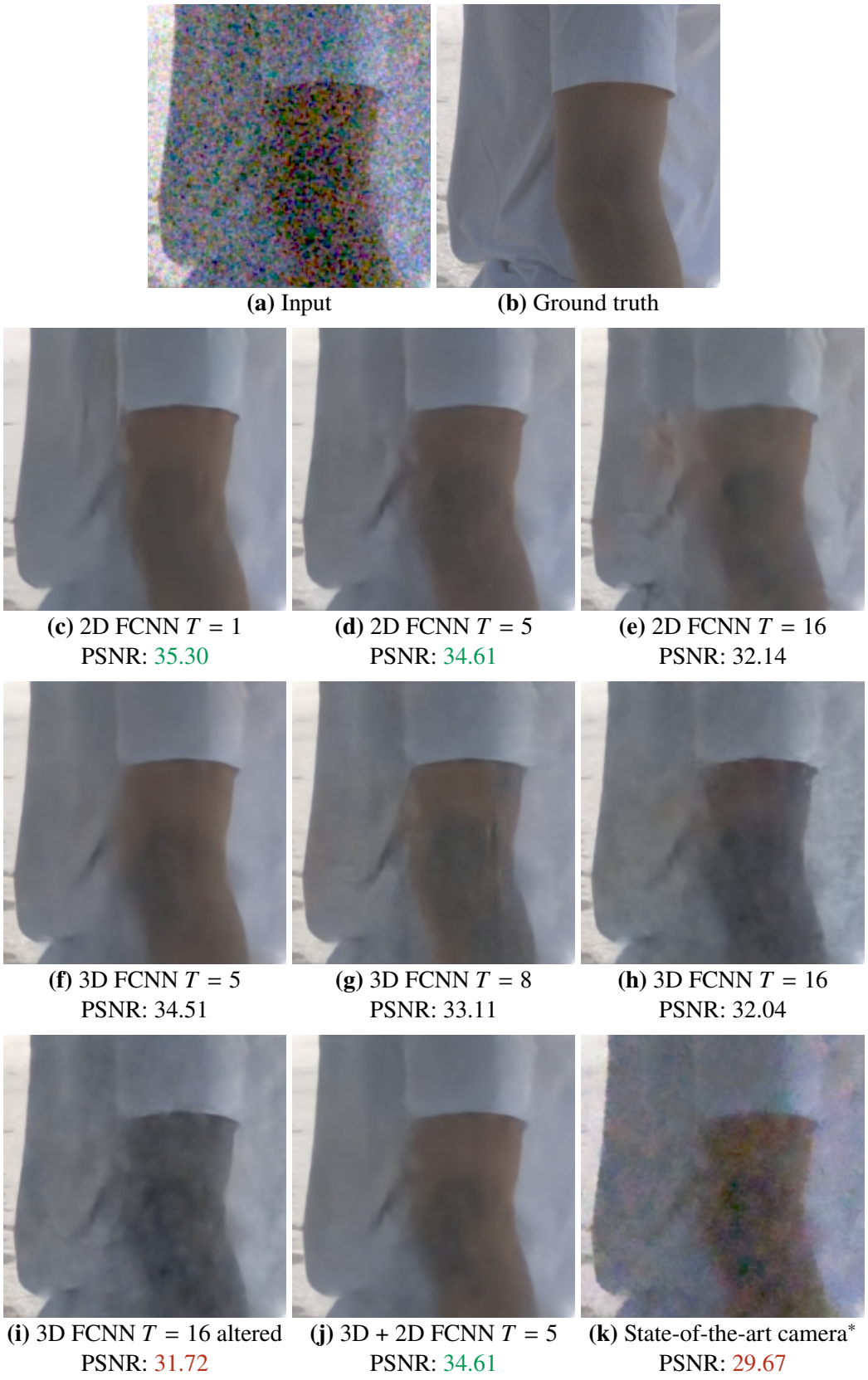
(h) 3D FCNN $T = 16$
PSNR: 30.87

(i) 3D FCNN $T = 16$ altered
PSNR: 30.94

(j) 3D + 2D FCNN $T = 5$
PSNR: 30.37

(k) State-of-the-art camera*
PSNR: 27.80

Figure 4.3: The performances when denoising a frame in a sequence with a person that is moving relatively slowly. Figure 4.3i refers to the network with $P = 128$ and 3 levels.



(a) Input **(b)** Ground truth

(c) 2D FCNN $T = 1$
PSNR: 35.30

(d) 2D FCNN $T = 5$
PSNR: 34.61

(e) 2D FCNN $T = 16$
PSNR: 32.14

(f) 3D FCNN $T = 5$
PSNR: 34.51

(g) 3D FCNN $T = 8$
PSNR: 33.11

(h) 3D FCNN $T = 16$
PSNR: 32.04

(i) 3D FCNN $T = 16$ altered
PSNR: 31.72

(j) 3D + 2D FCNN $T = 5$
PSNR: 34.61

(k) State-of-the-art camera*
PSNR: 29.67

Figure 4.4: The performances when denoising a frame in a sequence with a person that is moving quickly. Figure 4.4i refers to the network with $P = 128$ and 3 levels.

Measuring Performance

As stated in Section 3.5, we used PSNR and temporal coherence for measuring the performance of the different networks and configurations. Unfortunately, since the evaluation of temporal coherence can only be carried out by observing the resulting videos, we are unable to display examples of this in a good way in the report. Before evaluating the performance, the output images had their black level removed and underwent a gamma correction. This is a simplification of the remaining steps of the image pipeline after the denoising step where the input videos were extracted from, and will make the videos look more like they were processed by a complete pipeline.

4.1 Experiments

The major experiments of this thesis are shown in Table 4.2 and Table 4.3. The tables show several configurations for the 2D FCNN, the 3D FCNN, and the combined 3D and 2D FCNN with their respective performances. We carried out additional experiments, but the ones listed in the tables showed the most promise and were thus prioritized to train for a longer time. The results of the other experiments can therefore not be compared fairly to the ones listed in these tables, and are instead described and evaluated in Section 4.2.

Table 4.2: Performance of the networks. The average PSNR is calculated over every frame in the test set. The best performances are written in green and the worst ones in red.

Network configuration \ Performance	Average PSNR	Temporal coherence
2D FCNN $T = 1$	29.69	Bad, very fluttery
2D FCNN $T = 5$	31.37	Mediocre
2D FCNN $T = 16$	32.37	Very stable
3D FCNN $T = 5$	31.57	Mediocre
3D FCNN $T = 8$	32.15	Stable
3D FCNN $T = 16$	32.72	Very stable
3D FCNN $T = 16$, $P = 128$, 3 levels	32.93	Very stable
Combined 3D and 2D FCNN $T = 5$	31.44	Mediocre
State-of-the-art camera*	30.57	Exceptionally stable

Table 4.3: Performance of the networks on single frames with varying degrees of movement. The frames can be seen in Figure 4.1, 4.2, 4.3, and 4.4, respectively. The best performances are written in green and the worst ones in red for each column.

Network configuration \ Frame	No movement	Tree	Slow person	Fast person
2D FCNN $T = 1$	29.38	26.16	29.14	35.30
2D FCNN $T = 5$	32.10	28.85	30.10	34.61
2D FCNN $T = 16$	34.14	29.20	30.06	32.14
3D FCNN $T = 5$	32.43	29.02	30.42	34.51
3D FCNN $T = 8$	33.26	29.41	30.68	33.11
3D FCNN $T = 16$	34.60	29.89	30.87	32.04
3D FCNN $T = 16$, $P = 128$, 3 levels	35.18	29.92	30.94	31.72
Combined 3D and 2D FCNN $T = 5$	32.48	28.93	30.37	34.61
State-of-the-art camera*	34.21	26.38	27.80	29.67

4.1.1 Increasing the Sequence Length

For the 2D FCNNs, increasing T resulted in a higher average PSNR as well as a significant reduction of flutter, as can be seen in Table 4.2. Table 4.3 shows that a greater value for T gave better performance on frames with no movement or some movement, such as a frame with a person walking slowly. However, increasing T led to lower PSNR on frames depicting rapid movement, such as a person moving quickly as seen in Figure 4.4. Nevertheless, the network with $T = 16$ gave the best performance in general.

Similar behavior was seen for the 3D FCNNs. As T increased, the average PSNR got better, along with the PSNR for images with some or no movement. Additionally, there was a significant improvement in overall temporal coherence. The exceptions once again were swiftly moving objects, which appeared grainier for higher values of T. The network with 3 levels and a lower value for P resulted in the highest PSNR in most of the categories.

Now, suppose $\boldsymbol{x_T}$ depicts slow movement, meaning the difference between each adjacent frame is relatively small, or that there is no movement at all. Further assume that $\boldsymbol{x_1}$ denotes the input sequence with $T = 1$ which only contains the middle frame x_m of $\boldsymbol{x_T}$. Given $\hat{y}_1 = f(\boldsymbol{x_1}; \theta_1)$ and $\hat{y}_2 = f(\boldsymbol{x_T}; \theta_2)$, we have showed that

$$\text{PSNR}(\hat{y}_2, y_m) > \text{PSNR}(\hat{y}_1, y_m) \tag{4.1}$$

when $T \geq 5$. For clarity, note that $f(\boldsymbol{x_1}; \theta_1)$ describes a completely spatial noise filtering.

4.1.2 Comparing the Architectures

By comparing the 2D FCNNs with the 3D FCNNs in Table 4.2 and Table 4.3, we can see that the difference in performance is relatively minor. A general trend is that the 3D FCNNs produced higher average PSNR and PSNR for images with some or no movement, but were more affected by graininess of objects that were moving quickly. The combined 3D and 2D FCNN performed similarly well to these architectures. Overall it had slightly lower PSNR than the 3D FCNN and slightly higher PSNR than the 2D FCNN.

4.2 Additional Experiments

In addition to the experiments in Table 4.2, other attempts were made that resulted in no significant difference in performance. These experiments were all conducted by using the 2D FCNN with $T = 5$ and are given in the following list:

- Experimenting with different loss functions. This resulted in three attempts where the L_1-loss was replaced with the L_2, L_δ with $\delta = 0.5$, and L_1-pyramid loss functions.
- Using a variation of dropout. In this technique, one of the 5 images was turned black for two thirds of the training, and then no dropout was used for the remainder of the training. Note that this is not the same as using regular dropout, as that implies removing some of the weights for each training example, whereas this technique changed the actual input images.
- Using forced temporal as a pre-training method. By this we mean that the middle frame in every input sequence was turned black while the rest of the frames were noise free. The intent with this technique was to train the network to recover information temporally in a more explicit way. We experimented with pre-trainings from between a few epochs to two thirds of the total training time. Thereafter, the training would proceed normally.

Several experiments were conducted in comparable ways to the above list, but showed significantly worse performance. For example, these include using only one level in the U-net (i.e., removing three of the levels) or having $P = 64$ or less. These experiments produced output images which turned out considerably grainy. Additionally, similar results were produced when having the network output T frames, as opposed to only a single frame, and picking the middle frame as the output.

4.2.1 Dynamic Noise Level

Further experiments were made on data with a dynamic noise level rather than a constant one. While generating noise, the full well ranged from 12 to 25. This resulted in worse performance on a fixed noise level, compared to a network only trained on that particular level of noise. The performance was however drastically improved for inference on images with the same dynamic noise range.

Chapter 5
Discussion

With Equation (4.1) in our results we showed that when $T \geq 5$, a higher PSNR was achieved compared to when $T = 1$. It should be noted that we did not experiment with values for T in the range $1 < T < 5$, so it is possible that this equation still holds for an input sequence $\boldsymbol{x_T}$ with a smaller value for T as well. In addition, we did not experiment with values of T that was larger than 16, so we cannot be certain if the performance continues to increase proportionally with T.

We believe there is an interesting discussion to be had regarding the number of frames to consider in a sequence, and the related benefits when filtering videos. Our belief before this project was that by increasing the sequence size, a sizable benefit could be expected, while paying the price of a higher computational cost. This turned out to be mostly true, but from our results, we have seen that adding temporal information by increasing T does not only bring benefits. On one hand, a larger T implies better performance on most of our example images, while also providing better stability and flow in the video. On the other hand, the measured PSNR is worse on frames with quickly moving objects. It is therefore not completely obvious which the best configuration is. It depends on what setting the filter should be used in and what type of footage that is most important for the filter to denoise correctly. For filtering very fast movement, the networks with $T = 1$ or $T = 5$ would be the most performant. In other cases, the 3D network with $T = 16$, $P = 128$, and 3 levels would be the best one.

We believe there is a rather simple reason why the networks trained with the $T = 16$ configuration performed so well. These networks were fed with a comparatively large temporal dimension and therefore learned to use a larger amount of temporal information. Specifically, the reason why the 3D FCNN with $T = 16$, $P = 128$, and 3 levels achieved the best result was most likely because the network could focus even more on the temporal information. As speculated in Section 3.1.2, with the temporal depth being halved one time less and the number of filters doubled, the network could more accurately map information from all frames in the input sequence to the output.

5.1 Comparing Computational Cost and Number of Weights

Another aspect other than performance that should be taken into consideration is the computational cost. Increasing T results in higher computational cost for both training and testing. Training 300 epochs for the 2D FCNN with $T = 1$ took one day, whereas with $T = 16$, two days were required. This difference is rather insignificant when considering the performance increase in most of the images. The time difference is however quite significant when comparing 3D FCNNs to 2D FCNNs. It took the 3D FCNN with $T = 16$ about two weeks to complete the 300 epochs. Although showing slightly higher performance for most images, it is roughly 7 times slower to train and evaluate as the 2D FCNN with the same value for T.

When comparing the 2D FCNN and 3D FCNN one also has to take into account that the 3D FCNNs have more weights. A kernel that has a depth of 3, as was the case for the 3D FCNNs, results in 3 times as many weights in accordance to Equation 3.1 and Equation 3.2. An argument could therefore be made that the comparison between these networks may not be entirely fair. At the same time, they are still utilizing an equivalent U-net structure when just considering the spatial information. It is not exactly clear how to fairly compensate the 2D FCNNs to attain the same number of weights as the 3D FCNNs.

5.2 Comparison with the State-of-the-art Camera

Although the state-of-the-art camera* showcased the most stable output video, it performed poorly relative to the neural networks. Nevertheless, one should take into consideration that the computational cost of the state-of-the-art camera is multitudes cheaper than the neural networks. By considering our performance, however, we believe that neural networks could have a promising future in practical image pipelines, either by reducing network sizes or by using more powerful hardware in cameras.

5.3 Possible Improvements

We believe that a reason for why the overall best performing network, the 3D FCNN with $T = 16$, $P = 128$, and 3 levels, couldn't handle swift movement was because of inadequate size of the receptive field. For the aforementioned network, the receptive field measured 140×140 pixels. If an object has such rapid movement that it exits out of the receptive field during a sequence, certain frames become useless in the task of denoising this object. We believe the network even tries to incorrectly incorporate temporal information from frames where the object is outside of the receptive field, and thereby worsen the result. One improvement could therefore be to increase the receptive field.

*For how the results of the state-of-the-art camera were produced, see Section 3.2.1.

Still sequences might dilute the dataset, as moving objects are harder to denoise compared to motionless areas. A high-quality dataset would in the context of this thesis involve constant motion of varying speed. By further improving our dataset in this manner, higher performance could possibly be obtained. It is also possible that a completely different method, featuring a sophisticated motion-tracking technique to temporally denoise movement over multiple frames, could prove to be effective.

The temporal coherence is, according to Table 4.2, improved as T is increased. We believe that a loss function that correctly incorporates loss in the temporal dimension, as opposed to working on a frame by frame basis, could lead to better results. This is however difficult when there is movement, since it is not always trivial to make the distinction between noise and movement. This type of loss function would require some kind of motion detection.

5.3.1 Metrics for Video Comparison

Comparing videos posed a tough challenge, when something so seemingly simple as comparing single images is not entirely solved. PSNR is indeed a useful metric, but may fail to account for errors that humans perceive as obvious. Naturally, there is more room for errors with a third temporal dimension to consider as is the case for video. We failed to find an accurate and objective way of measuring the flutter between frames and therefore decided to compare the results through visual inspection. It is entirely possible that suitable metrics exists that we are not aware of, which could have been used to further validate our conclusions.

5.3.2 Training Data

We never noticed any tendency for overfitting in any of our models during the training sessions. To introduce a regularization term in the loss function was therefore deemed unnecessary. A simple explanation for the behavior could be that the random selection of patches and augmentation of our input data led to large enough variation, which kept the model from overfitting during at least 300 epochs. Therefore, it is possible that a bigger dataset would not be beneficial. The data could perhaps be improved in other ways. Although our noise model appears to be accurate, a dataset featuring video sequences with real noise would be optimal.

5.4 Simulating Low-Light Videos

Even though the goal of our thesis was to denoise videos under low-light conditions, most of the videos in our dataset were captured in brightly lit situations. This may seem counterintuitive, but was decided to avoid inherent noise in the clean images and to control the noise fully with the noise model. Since the pixel values were scaled to an interval that simulates a low-light situation, brightly lit images could therefore be used for this purpose.

5.5 Benefits with a Dynamic Noise Level

Training a noise filter on dynamic noise levels would most likely be more useful in a real-life scenario. The noise level naturally varies significantly during the duration of a day, since the amount of light that a camera sensor receives changes with time. A filter that is capable of handling varying levels of noise would therefore be more flexible and overall produce better images compared to a network that is trained to perform well on a single noise level.

Chapter 6
Conclusion

We have built three different architectures with better denoising performance than a solely spatial filter as well as a noise filter in a state-of-the-art camera*, as was our goal. We have showed that by increasing the sequence length for the 2D FCNNs, higher PSNR is achieved for denoising most of our example sequences. These include sequences with no movement, slight movement (e.g. wind blowing against trees), and slowly moving objects. In addition, a larger sequence length resulted in a higher overall PSNR and a drastically better temporal coherence when inspecting the videos. However, increasing the sequence length meant worse performance when an object moved quickly across the scene.

We can also conclude that the 3D FCNNs scored slightly higher PSNR than the 2D FCNNs in almost every category, except for the objects that were moving rapidly. The 3D FCNNs took roughly 7 times longer to train and test for the higher values of sequence length, though, so a trade-off between time and performance should be made between these networks. The combined 3D and 2D FCNN scored somewhere in between the 3D FCNNs and 2D FCNNs when it came to performance as well as time spent training. A variant that further increased the performance of the 3D FCNNs used a smaller patch size and one less level in the U-net.

Several different experiments were conducted for the 2D FCNNs that displayed similar results with one another, both in terms of PSNR and visual performance. These include replacing the loss function with either the L_2, Huber, or L_1 pyramid loss, using so-called forced temporal, and using our variation of dropout.

In addition, some experiments for the 2D FCNN resulted in poor performances. These include using only one level in the U-net, having the network output the same number of frames as in the input sequence, and using a patch size of 64 or less.

*For how the results of the state-of-the-art camera were produced, see Section 3.2.1.

Chapter 7
Future Work

There is almost an endless number of configurations to train a neural network with. We have found some configurations that work better than others, but there are many left unexplored. One way to continue from here could therefore be to optimize the parameters or the structure of the network. One thing in particular that we mentioned in our discussion, that could possibly increase the performance on swiftly moving objects, is to make the receptive field larger and train the network specifically on data with more movement.

The loss function could be further developed in order to improve temporal noise filtering. It was attempted to some degree, but unsuccessfully so. Optical flow may be a useful metric for comparing temporal change [28], that could be worth looking into.

There are promising network architectures for capturing the temporal aspect not included in this thesis. One of them is a RNN-based design, where the network keeps information about previous frames in a state. This could certainly be something to experiment with in order to enhance temporal denoising. A successful attempt was made by Godard et al. [4], using combinations of recursive and convolutional layers. Another approach discussed for this thesis was to use a generative adversarial network (GAN), resembling Wang et al. [29]. This type of network could be used to dynamically learn the loss function and incorporate the temporal dimension that way.

Bibliography

[1] C. Chen, Q. Chen, J. Xu, and V. Koltun, "Learning to see in the dark," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[2] T. Plötz and S. Roth, "Benchmarking denoising algorithms with real photographs," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 2750–2759.

[3] P. Chandramouli, S. Burri, C. Bruschini, E. Charbon, and A. Kolb, "A 'little bit' too much? high speed imaging from sparse photon counts," *CoRR*, vol. abs/1811.02396, 2018. [Online]. Available: http://arxiv.org/abs/1811.02396

[4] C. Godard, K. Matzen, and M. Uyttendaele, "Deep burst denoising," *arXiv preprint arXiv:1712.05790*, 2017.

[5] Wikipedia, "Color Filter Array," https://en.wikipedia.org/wiki/Color_filter_array, accessed: 2018-11-28.

[6] Q. Imaging, "Understanding CCD Read Noise," http://www.qsimaging.com/ccd_noise.html, accessed: 2018-11-29.

[7] J. R. Janesick, *Scientific Charge-Coupled Devices.* Bellingham, WA, USA: SPIE – The international Society for Optical Engineering, 2001, isbn: 0-8194-3698-4.

[8] J. Peacock, A. Taylor, and L. A, "Astronomical statistics," https://www.roe.ac.uk/japwww/teaching/astrostats/astrostats2012.pdf, September 2012, accessed: 2019-01-28.

[9] Wikipedia, "Digital Image Processing," https://en.wikipedia.org/wiki/Digital_image_processing, accessed: 2018-11-29.

[10] M. Gharbi, G. Chaurasia, S. Paris, and F. Durand, "Deep joint demosaicking and denoising," *ACM Trans. Graph.*, vol. 35, no. 6, pp. 191:1–191:12, Nov. 2016. [Online]. Available: http://doi.acm.org/10.1145/2980179.2982399

[11] M. Guarnera, G. Messina, and V. Tomaselli, *Image Processing for Embedded Devices*. Bentham Science Publishers, 01 2010, ch. Demosaicing and Antialiasing Correction, pp. 149–190, isbn: 9781608051700.

[12] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, M. Hasan, B. C. V. Esesn, A. A. S. Awwal, and V. K. Asari, "The history began from alexnet: A comprehensive survey on deep learning approaches," *CoRR*, vol. abs/1803.01164, 2018. [Online]. Available: http://arxiv.org/abs/1803.01164

[13] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising with block-matching and 3d filtering," in *Image Processing: Algorithms and Systems, Neural Networks, and Machine Learning*, vol. 6064, San Jose, California, United States, Feb 2006.

[14] F. Rosenblatt, *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*, ser. Report (Cornell Aeronautical Laboratory). Spartan Books, 1962, no. 62012882. [Online]. Available: https://books.google.ca/books?id=7FhRAAAAMAAJ

[15] P. J. Werbos, *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*. New York, NY, USA: Wiley-Interscience, 1994, isbn: 0-471-59897-6.

[16] M. A. Nielsen, *Neural Networks and Deep Learning, Chapter 1*. Determination Press, 2015.

[17] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, "Is l2 a good loss function for neural networks for image processing?" *IEEE Transactions on Computational Imaging*, 11 2015.

[18] P. J. Huber, "Robust estimation of a location parameter," *The Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73–101, 1964. [Online]. Available: http://www.jstor.org/stable/2238020

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[21] S. U. Andrej Karpathy, "CS231n Convolutional Neural Networks for Visual Recognition," http://cs231n.github.io/convolutional-networks/, accessed: 2018-12-06.

[22] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 3431–3440.

[23] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *CoRR*, vol. abs/1603.07285, 2016.

[24] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds., vol. 9351. Cham: Springer International Publishing, 2015, pp. 234–241, isbn: 978-3-319-24574-4.

[25] R. Bermúdez-Chacón, P. Márquez-Neila, M. Salzmann, and P. Fua, "A domain-adaptive two-stream u-net for electron microscopy image segmentation," in *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, April 2018, pp. 400–404.

[26] "Tensorflow," https://www.tensorflow.org/, accessed: 2018-11-30.

[27] Ö. Çiçek, A. Abdulkadir, S. Lienkamp, T. Brox, and O. Ronneberger, "3d u-net: Learning dense volumetric segmentation from sparse annotation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, ser. LNCS, M. S. G. U. S. Ourselin, W.S. Wells and L. Joskowicz, Eds., vol. 9901. Springer, Oct 2016, pp. 424–432, (available on arXiv:1606.06650 [cs.CV]). [Online]. Available: http://lmb.informatik.uni-freiburg.de/Publications/2016/CABR16

[28] D. Fleet and Y. Weiss, "Optical flow estimation," in *Handbook of Mathematical Models in Computer Vision*, N. Paragios and O. Chen, Yunmeiand Faugeras, Eds. Boston, MA: Springer US, 2006, pp. 237–257, isbn: 978-0-387-28831-4. [Online]. Available: https://doi.org/10.1007/0-387-28831-7_15

[29] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, G. Liu, A. Tao, J. Kautz, and B. Catanzaro, "Video-to-video synthesis," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.