

ABSTRACT

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

- NIND Natural Image Noise Dataset [3][[nind-ntire](#)]
BCE Binary Cross-Entropy
BM₃D Block-Matching and 3D filtering [7]
BN Batch Normalization [14]
cGAN Conditional Generative Adversarial Network [20]
CNN Convolutional Neural Network [29]
DCGAN Deep Convolutional Generative Adversarial Network[23]
GAN Generative Adversarial Network [9]
LSGAN Least Squares Generative Adversarial Network [19]
MSE Mean Squared Error
RED-Net very deep Residual Encoder-Decoder Network [18]
ReLU Rectified linear unit [21]
PReLU Parametric Rectified Linear Unit [10]

SIDD Smartphone Image Denoising Dataset [1]

SSIM Structural Similarity [27]

C500D Canon EOS 500D DSLR camera

X-T1 Fujifilm X-T1 mirrorless camera

k kernel size

p padding size

INTRODUCTION

Photographic image noise occurs as a camera sensor's ISO sensitivity increases to capture an image faster than it would in ideal conditions ("base ISO" sensitivity).¹ A fast shutter speed is often necessary even though there is insufficient light, for instance with handheld photography where a slow shutter speed results in blur caused by the camera shake, or when a dynamic subject results in motion blur. Increasing the ISO setting is akin to linearly amplifying the value measured on each sensor cell. A small initial value that is amplified is less accurate and more prone to errors; this amplified value in turn makes up photographic noise.

Denoising is typically seen as the inverse problem of recovering the latent clean image from its noisy observation [18].

Photographic development usually incorporates a conventional denoising method such as non-local means [4] or wavelets-based methods [wavelets-denoising], or a combination thereof, sometimes making use of prior information about a specific sensor's response to different ISO noise values. [34] Block-Matching and 3D filtering [7] (BM3D) is a more powerful denoising method which has been used to benchmark newer techniques [all-bm3d-benchmarks], though it is seldom used in off-the-shelf software (presumably because of its high computational cost and the need to specify a γ noise value parameter).

The use of deep learning to solve the denoising problem by directly generating the latent clean image, or in some cases recreating the noise and subtracting it from the observed image [28], has been an active research area. However, while the synthetic results show state-of-the-art performance, testing on real data indicates that neural network-based solutions do not exceed the performance offered by BM3D [22]. It appears that neural networks simply learn the applied noise distribution and that ISO noise may involve additional transformations such as color distortions and loss of detail.

Some specialized work has shown that neural networks obtain state-of-the-art performance when trained with real data [5][30]. We sought to assess the potential of deep learning applied to the denoising problem by expanding on this previous work through a dataset of images produced with various levels of ISO noise. This dataset can be used to train neural network models for general purpose denoising of high quality images.

¹ We often make references to ISO noise because increased ISO sensitivity is the main cause of noise, but it should be noted that there are other factors affecting the magnitude of noise acquired by the image sensor.

Our work introduces an open dataset of DSLR-like² image sets with various levels of real noise caused by the digital sensor's increased ISO sensitivity. The dataset is large enough to be used for training and varies in content in order to model a great variety of scenes. Each scene was captured at multiple noise levels, with an average of 6 images per set, such that a model may be trained for blind denoising on the base ISO as well as beyond the highest ISO value of the camera by feeding it crops that have a random noise value.

The images in a set are all pixel-aligned. Some of the scenes include multiple ground-truth images which may be sampled at random during training; these would prevent the model from learning to reconstruct the random noise it has seen on one ground-truth, thus making it more difficult to overfit the noise. Overexposed areas are avoided in the ground-truth images because details are lost when the sensor is saturated and this could potentially give an advantage to higher ISO images in which the sensor is not necessarily saturated (notably in ISO-invariant cameras and high ISO pictures that are brightened using software).

The dataset is published in sRGB format on Wikimedia Commons, which is an open-platform that promotes continuous discussion and contribution.

We trained several convolutional neural network architectures presented in the literature, namely DnCNN [28], very deep Residual Encoder-Decoder Network [18] (RED-Net), and U-Net [24], and analysed different proposed methods such as DnCNN [28]'s residual learning.

In most tests a U-Net model was trained with the Natural Image Noise Dataset [3][nind-ntire] (NIND) and validated over increasing ISO values; test sets were taken with the same Fujifilm X-T1, as well as a Canon EOS 500D DSLR camera to assess the generalization. We also attempted to combine other datasets with our training data to assess the potential loss in performance caused by generalization. Finally we investigated the use of Generative Adversarial Networks[15] to solve issues such as unnatural face smoothing.

² We define a DSLR-like camera as one produced with an APS-C (25.1x16.7 mm) or larger sensor such as those present in most DSLR and mirrorless cameras

2

DATASET

The Dataset chapter goes over some existing image noise datasets before delving into the Natural Image Noise Dataset.

2.1 RELATED WORK

The following works feature datasets made of ground-truth / noisy image sets. The static scenes approach is necessary to directly compare the level of degradation using a loss function such as the Structural Similarity [27] ([SSIM](#)) index or the Mean Squared Error ([MSE](#)).

2.1.1 *Darmstadt Noise Dataset*

Synthetic noise is typically used to train and test models, but it had been unclear whether the reported synthetic results translated to real improvements. The Darmstadt Noise Dataset (DND) [22], containing 50 pairs of noisy-clean images from four cameras, was developed for the purpose of validating denoising algorithms using real data. Plötz and Roth showed that many modern denoising methods do not perform well on real data and that [BM3D](#) [7], which was published in 2006, remains one of the best performing methods [22]. RENOIR [2] is a similar dataset that was published prior to the DND; however, Plötz and Roth noted spatial misalignments that reduced its effectiveness. We have additionally found that the light sometimes differs between images in the same scene and that some photographs exhibit significant raw overexposure.

2.1.2 *Learning to See in the Dark*

See-in-the-Dark (SID) [5] is an image noise dataset that is large enough for training and, to our knowledge, was used in the first successful attempt at denoising images using real image noise. This dataset focused on very low-light photography where the camera-generated JPEG appears black. The authors used a U-Net network architecture to create an end-to-end RAW-to-JPEG pipeline that produces realistic colors, improving on standard processing and BM3D denoised images which still suffer from color bias at high ISO. Our work differs from SID in that we aimed to train a general purpose ("blind") denoiser rather than one that handles a specific condition, such as extremely low light images. We chose to work in sRGB space because handling the whole RAW-to-sRGB pipeline removes some information which

may otherwise be useful to the author during development. Moreover, one dataset can then be used with different types of color filter arrays.

2.1.3 Smartphone Image Denoising Dataset

The Smartphone Image Denoising Dataset (SIDD) [1] is comprised of 10 scenes * 5 cameras * 4 conditions * 150 images, totalling 30000 images. This dataset aims to address the problem of smartphone image denoising, where the small sensor and aperture size causes noticeable noise even in pictures taken at base ISO. Further processing is thus applied to create ground-truth images out of many images. This method of creating ground-truth images is not entirely relevant for denoising images captured with larger sensors because a single image taken at base ISO on a DSLR-like camera is clean enough to work as ground-truth for training purposes.

2.2 NATURAL IMAGE NOISE DATASET

DRAFTy-as-heck sentence

FIXME long run-on sentence

The Natural Image Noise Dataset fills the gap for an image noise dataset of ground-truth - noisy image sets that targets cameras equipped with larger sensors² and which is large and varied enough (in both noise values and content types) to train a denoising model for ISO-blind denoising.

Here we outline the physical setup required to capture image sets for the NIND, summarize its content, explain the software processing and validation requirements, and describe its publication aspects such that others that wish to do so may also contribute.

2.2.1 Capture

We captured several images per static scene; at least one ground-truth taken with the camera's lowest ISO setting and several images taken with increasing ISO settings and consequent decreasing shutter speed in order to match the original exposure value. Scenes were captured using a camera affixed to a tripod and controlled with a wireless remote control to avoid shifting the setup position. We ensured that the ground was stable, wind would not cause any change in the scene, lighting did not vary between shots, and no area was overexposed in the ground-truth images. Overexposure occurs when the sensor is saturated; on a ground-truth image this would potentially benefit the high ISO images because less light is captured with a faster shutter speed, therefore, the higher ISO images may not be overexposed and thus may contain detail that is not present in the overexposed ground-truth. The aperture remained the same on all shots and the focus was set manually so that it would not automatically adjusted for each frame.



Figure 2.1: Sample ground-truth images from the Natural Image Noise Dataset.

ISO value	100	200	250	320	400	500	640	800	1000	1250	1600	2000	2500	3200	4000	5000	6400	High	Scenes	Images
Fujifilm X-T1	100	105	11	8	18	13	16	25	16	9	12	9	19	25	17	19	91	123	90	536
Canon C500D	14	10			10		10			9			11		17	19	10	11	74	
Total	15	116	11	8	29	13	16	36	16	9	22	9	19	37	17	19	91	133	101	616

Table 2.1: Dataset content

A base ISO image (ISO200 on the Fujifilm X-T1) was always taken at least once, along with the camera’s highest ISO setting (ISO6400 on the X-T1). Several images were taken with different intermediate ISO values such that the ISO settings varied across each scene. We often also took images that we categorized as “High ISO,” which consisted of the highest ISO value and increased shutter speed. “High ISO” images result in dark frames which are then correctly exposed using software. We often tried to match shutter speeds that would be useful to denoise, such as 1/60s for handheld photography, 1/15s for devices equipped with optical image stabilization, and 1/1000s or faster for high-speed photography. We ensured that every ISO value was well represented in order to train models effective at blind denoising. On average, six images were produced per scene and some scenes featured multiple ground truth images which could be used in training to help prevent over-fitting.

2.2.2 Content

Many objects were captured in museums where subjects are plentiful (albeit we had to be mindful of copyright restricted material) and for which a denoising application would be highly relevant because indoor handheld pictures require a high ISO sensitivity. However, initial tests found that using images taken only indoors did not provide the variety needed to create a model that generalized well across all conditions, as natural colors were sometimes off in outdoor and brightly lit applications. We thus captured natural objects with vibrant colors (such as food items and plant-life) as well as outdoor scenes where the shutter speed could be taken as fast as 1/13000s using a digital shutter. We made an effort to include some text because it is prevalent, yet we expect a model would not be able to guess how to reconstruct it (Figure 5.3 shows the resulting denoised text), and we tried to make the images pleasant to look at in order to enhance the

time users would spend looking at them. Most of the NIND images were captured on a Fujifilm X-T1 mirrorless camera, which uses a 23.6×15.6 mm X-Trans sensor. A Canon EOS 500D DSLR camera, featuring a 22.3×14.9 mm standard Bayer sensor, was used to capture images that could be used to validate the generalization. The content of the dataset is summarized in Table 2.1 and a subset of the X-T1 pictures are shown in Figure 2.1.

2.2.3 Software processing

We processed the dataset images using darktable [33] (an open source development software) for raw-to-sRGB development. Our development steps are similar to those we would apply to a standard picture but no sharpening was applied, as this greatly amplifies noise and is typically applied last in the pixel pipeline (we can expect users to apply sharpening to the generated clean image without any perceptible loss). We used darktable’s automatic exposure mode to match a fixed percentile target on the histogram and calculate the required exposure compensation for all images in a set. Likewise, we ensured that the white balance was identical and all development steps were copied over to the entire scene. (The exposure percentile and white balance settings are fixed within a scene but vary across different scenes.) The raw overexposed indicator was used to verify that no overexposed areas were present in the base ISO image and if any were detected then it was cropped out or the scene was discarded. The images were visually inspected to detect slight variations in light, the introduction of foreign objects such as insects, or any movement, which also resulted in cropping or discarding images. The ground-truth images must be at least as sharp as their noisy counterparts; this is sometimes not the case due to slight movements on longer exposures. The remaining images were saved in either high-quality (98 to 100) 8-bit JPEG or lossless 16-bit PNG.

The last step of development is to use Hugin’s align_image_stack tool [13] to ensure that all images in a set are perfectly pixel-aligned. The tool will usually return the same image size as the input, in which case the whole image set can safely be used. When a difference is detected then the tool will automatically align the set and we visually analyzed whether the result was acceptable or the movement caused a change in perspective, in which case the outlier images were discarded. Some noisy images cannot easily be matched to the scene; possible solutions are to denoise these images in order to check the alignment or to take a cleaner image afterward and assume that the middle images are consistent with the previous and next ones.

2.2.4 Publishing

The Natural Image Noise Dataset is published on Wikimedia Commons (https://commons.wikimedia.org/wiki/Natural_Image_Noise_Dataset), an online repository of free-use images and other digital media. Wikimedia Commons hosts media content for all Wikimedia projects and its scope is limited only by the content having some educational value (broadly meaning “providing knowledge; instructional or informative”). As such, we believe it is a fitting platform for the publication of a research dataset.

One key advantage of using Wikimedia Commons is its collaborative aspect. Anyone is allowed to add images to the dataset, modify existing images (for example to fix a spatial misalignment), and discuss the content (through the discussion page provided for each file, category, and the dataset itself). The collaborative aspect also includes a “Quality images candidates” page [6] where users assess the technical quality of a submitted image and may promote it to a “Quality image” standing. Many of the ground-truth images have gone through this process and were promoted through human assessment. The same process was also used to validate the trained model, which ended up in a positive assessment since even the denoised dynamic ISO6400 picture presented in Figure 5.4 was among the promoted images.

On the technical side, Wikimedia Commons preserves images as they are uploaded; JPEG images are not recompressed, 16-bit lossless TIFF and PNG images are allowed, and the metadata is kept. Thumbnails are generated and images may be visualized before being downloaded in full resolution, and the download may include a select subset instead of the whole dataset. A customizable download script is provided on the dataset’s page for convenient retrieval. Even though files can be overwritten, every file uploaded on Wikimedia Commons is kept forever therefore specific snapshots of the dataset can be made and the download script can fetch the dataset as it was on any specified date.

It is also important to note the usefulness of the data outside of a denoising dataset context. Many images, such as those depicting artifacts displayed in churches and museums, have encyclopedic value and the ground-truth images present in our dataset are of higher quality than most previously available images depicting such artifacts. By publishing the dataset on Wikimedia Commons, these ground-truth images may be used in Wikipedia articles directly¹. This may be a motivating factor to those wishing to contribute.

¹ E.g. Bobo people, Bombardment of Brussels, Dengese people

3

GENERATIVE NEURAL NETWORKS

This chapter introduces the notion of a convolutional neural network and its components, then goes over the specific network architectures we have worked with to solve the denoising problem. All networks presented in this chapter are generative, that is they take an image of $W \times H$ pixels with three channels and generate a new image of the same dimensions. Generative networks differ from other network types of convolutional neural networks which have different output shapes, such as classification networks which output one to a few nodes each indicating the probability that an object is present, and segmentation networks which usually have the same shape as the input but one output channel per segmented feature.

3.1 CONVOLUTIONAL NEURAL NETWORKS

We use deep convolutional neural networks which are neural networks based on the multilayer perceptron; that is, input data from one layer is sent to the neurons of the next layer and a dot product is performed between that input data and the next layer's learned weights. This process goes on for each layer. The main difference between a multi-layer perceptrons and a Convolutional Neural Network [29] ([CNN](#)) is that a [CNN](#) is not fully connected (that is neurons in one layer only receive a subset of the previous layer's neurons), instead, a [CNN](#) takes advantage of the local spatial coherence of the input images to share weights and thus reduce the number of shared parameters.

The first and last layer have the same shape as the input image, that is its height, width, and 3-channels representing the RGB color-space. Most layers are hidden layers between the first and last one and contain many more channels (also called feature maps). Hidden layers are usually convolution (or transposed convolution) layers, activation layers, or pooling layers.

A loss function is applied to the output of the last layer, its gradient with respect to the network's weights is calculated with the back-propagation algorithm, and the Adam gradient-descent algorithm[16] adjusts the weights at the end of each batch training iteration. The learning rate needs to be fine-tuned to optimize fast learning without over-fitting.

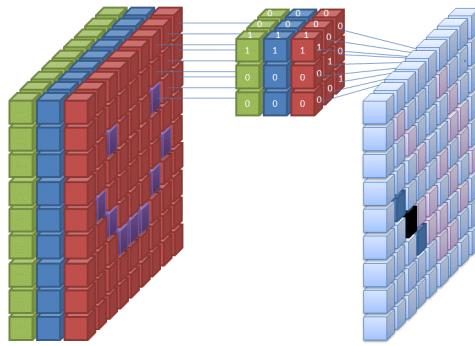


Figure 3.1: A filter is applied to the first 3-channels (RGB) input of an image in a convolutional neural network. The same filter is applied across all channels of the whole image as shown, and many of these filters are applied to form the following layer's feature map (I.e. its channels). Illustration by Wikimedia Commons user Cecbur (CC-BY-SA-4.0 license)

3.1.1 *Layers*

Convolution layers apply convolutions with C filters (also called features) to every $k * k$ area in the previous layer's width and height¹, taking in every channel and returning one value per filter. Convolutions have a receptive field that is determined by kernel size k (typical values for k are 3 and 5). The next layer's width and height is thus reduced unless padding is applied. Given an input layer's width or height dimension d and padding size (p), a convolution layer's output size can be calculated as $d - \text{kernel size } (k) + 2p + 1$. Each filter is weighted and weights are learned parameters. The same stack of filters is applied on every $k * k$ area of a given layer, therefore **CNN** have largely reduced number of weight parameters.

Activation layers introduce non-linearity, they apply an activation function to every value in the previous layer and the resulting layer has the same shape as its input layer. We use the Sigmoid ($\phi(z) = \frac{1}{1+e^{-z}}$) and Rectified linear unit [21] (**ReLU**) ($R(z) = \max(0, z)$) activation functions.

3.1.2 *Loss function*

The loss function assigns a score the the generated images with respect to the provided ground-truth. Loss functions need not to simply compare the difference between generated and ground-truth pixels

¹ assuming a stride of 1

because there is no one-to-one mapping between clean and noisy images, therefore a loss function which only focuses on the difference between pixels favors denoised images whose pixels average the many acceptable values; i.e. the network is trained to generate blurry images. [15]

The [MSE](#) (or ℓ_2) measures the average squared difference between the generated and ground-truth pixels, it is a simple yet effective loss function. Similarly the ℓ_1 loss measures the sum of the absolute differences between ground-truth and generated pixels, this typically favors blurrier results. The [SSIM](#) index is another metric which puts weights on structural information that is more likely to be perceived in images, namely local changes in luminance, contrast, and structure. A loss may also be learned such as the discriminator in a Generative Adversarial Network [9] ([GAN](#)).

Mixing different losses often appears to yield better results [31], although it is sometimes challenging to combine them because they are often scaled differently (and training may suffer a performance penalty when having to compute multiple losses). Another valid approach is to switch loss function once convergence has been attained with one. [31] We typically use the [SSIM](#) index in our standard models and a combination of loss functions in the more challenging [GAN](#).

TODO: batch normalization

3.2 NETWORK ARCHITECTURES

3.2.1 DnCNN

DnCNN is a simple flat architecture [CNN](#) architecture introduced in [28] to denoise images with residual learning. The network performs end-to-end denoising using only a convolution layer followed by Batch Normalization [14] ([BN](#)) and a [ReLU](#) activation layer at each depth level. The dimensions of the layers remain constant because the convolutions are padded appropriately ($p = \frac{k-1}{2}$).

The main contribution introduced in [28] appears to be the use of residual learning to speed up learning by modeling and subtracting the residual noise rather than generating a clean image. The use of a simple DnCNN architecture may have been introduced to emphasize the effectiveness of residual learning. DnCNN is the first network we experimented with, and although most of our experiment focus on the subsequent networks, we tested the author's residual learning theory with our dataset of ISO noise and all three networks.

3.2.2 RED-Net

The [RED-Net](#) architecture is a residual encoder-decoder network. The first half of the network encodes the image using convolutions without padding such that the layers' dimensions continuously decrease, then

the second half of the network (decoder) is made of transposed convolution layers that increase the layers' dimensions up to the original image's height and width. A [ReLU](#) activation layer is placed between every (de)convolution layer. What is referred to as a residual network differs from the residual learning of [28], a residual network has skip connections which in the case of [18] are placed every two convolution layers and connect them to transposed convolution layers of matching dimensions. These skip connections add details that may have been lost in the encoding process to the output of the transposed convolutions.

3.2.3 *U-Net*

The U-Net architecture is an encoder-decoder with skip connections originally designed for image segmentation. Its use in an image generator is possible by using the number of input color channels as output features and upsampling the resulting image. Each U-Net block applies two 3×3 convolutions followed by a parameter-free pooling operation that downsamples the image by a factor of two. The first block has 64 filters and the number of filters doubles after every pooling operation until there are 1024 filters. From this point the downsampling pooling operations are replaced with 2×2 upsampling transposed convolutions and the number of filters get halved every time, but the convolutions remain (as opposed to using transposed convolutions) so the layers' size remain smaller on the upsampling stage than it is originally. (This design results in corrupted borders which need to be discarded.) The U-Net architecture uses significantly less resources than conventional encoder-decoder networks and it is able to capture details at multiple frequencies.

3.2.4 *HulbNet*

We designed a multi-scale architecture somewhat similar to U-Net whose aim is to be more adapted to the denoising problem at hand. HulbNet uses the following concepts:

- **Transposed convolution:** The blocks present in the upsampling stage use transposed convolutions rather than standard convolution in order to restore the original layers' size. Transposed convolutions are used in RED-Net as well.
- **Dilated convolution** adds space between each element of the convolution filter as shown on Figure 3.2. This allows the receptive field to grow and the output layer's dimensions to shrink without increasing the number of parameters and computation steps.

Figure 3.2: Dilated convolution (dilation=2). Illustration by GitHub user vdu-moulin (MIT license)

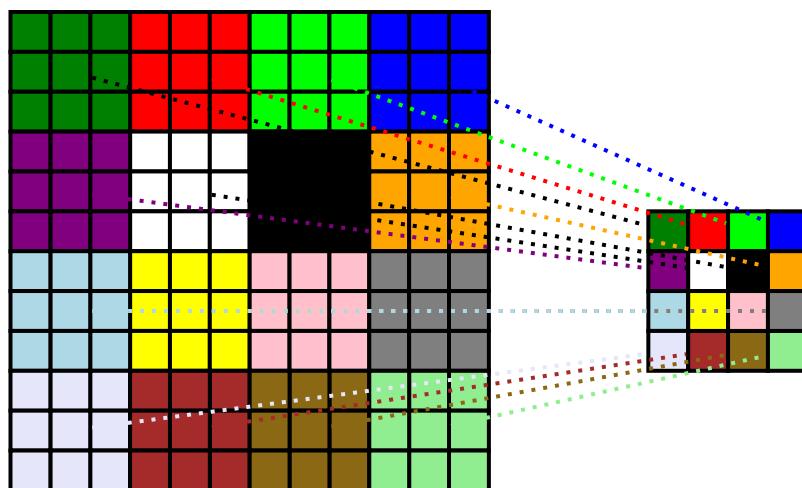


Figure 3.3: 3×3 convolution with stride=3

- **Stride** is the xy space the convolution filter moves between each step in a layer. The default convolution has a stride of one such that each element is seen three times in a 3×3 convolution. We apply convolution with a stride of three instead of using a 2×2 pooling operation to downsample the image between blocks. The resulting feature maps are downsampled by a factor of three with learnable parameters, as shown on Figure 3.3.
- **Dense connections:** HulbNet uses concatenated filters similar to the concept introduced in DenseNet [12]. Residual connections such as those used in [18][24] are element-wise sum of the feature maps, as such the feature maps are combined and the network can no longer chose individual elements to learn from. Dense connections on the other hand form concatenated features, that is the receiving filter can learn from any of the features separately and features can bring new information that does not necessarily need to be strongly correlated or have matching depths; features may therefore come from filters with different receptive fields and in different parts of the network.

HulbNet has two main types of convolution blocks; standard convolutions are made up of two series of standard 3×3 convolutions while dilated convolution blocks have one convolution with dilation set to two. These blocks have the same effective receptive field with the height and width of the feature maps being reduced by two after each block, and so they are both applied then concatenated as the input to the next layer. The motivation behind this combination is to capture features from multiple resolutions as the dilated convolution is expected to focus more on low-frequencies and repeated standard convolutions would capture the finer details. The first layer has an additional dilated convolution with dilation of five to capture an especially wide window of 10×10 .

After two blocks of each type, the feature map is downscaled by a factor of three with the aforementioned strided convolution. The strided convolution is expected to result in the most appropriate type of pooling (or combination thereof), and using a stride of 3 (which reduces the dimensions by the same factor) results in each pixel being processed exactly once therefore avoiding checkerboard patterns which can occur when some lines are processed more than others.

The blocks and downscale operations are repeated three times (with minor adjustments based on the intended input size) resulting in a single feature stack. The network is then mirrored using transposed convolutions that replace the convolutions to grow back to the original image size. All concatenated features in the downscaling part of the network are concatenated into the input of each upscaling block. The last block features a bottleneck 1×1 convolution which reduces the number of features down to three channels. Each convolution is followed by a Parametric Rectified Linear Unit [10] (PReLU) activation.

Batch Normalization [14] was not applied because experimental results showed worse performance using it in denoised image generation.

HulDisc is a matching discriminator network which follows the same structure as the aforementioned HulbNet but uses Batch Normalization and end in the middle with a single probabilistic feature.

4

GENERATIVE ADVERSARIAL NETWORKS

Generative Adversarial Networks are pairs of competing networks; a generator and a discriminator contest each other in a zero-sum game. The goal of the generator is to fool the discriminator by generating content which cannot reliably be differentiated from real ground-truth content. The discriminator attempts to tell generated content apart from real data. Each network is trained as the other's loss function (or one thereof).

The main advantage of a Generative Adversarial Network [9] is that the loss function is learned; it does not need to be defined and it can exceed the performance obtained with a conventional loss that is a limiting factor. The generator can learn to restore high-frequency details which cannot be determined from the noisy observation alone (and that are not necessarily visible in the ground-truth) instead of averaging the many possible solutions.

Moreover, data may be added without a matching pair in some scenarios. This does not work with Conditional Generative Adversarial Network [20] nor can it be combined with other loss functions (combining losses is often necessary to ensure that the result matches and that the details are not overdone), but it can be of great usefulness to further train an existing network to perform better on specialized tasks.

The discriminating network differs from the generative networks presented in chapter 3 in that they classify the output image by outputting a single $1 \times 1 \times 1$ probability.

4.1 TYPES OF GANS

- A [GAN](#) is the simplest approach introduced in [9]. A generator is called with a given input, then the discriminator trains on a mini-batch made of the generated data and another mini-batch containing ground-truth data.

A [GAN](#) does not necessarily need clean-noisy pairs of images because it only receives one image at a time. However, this feature may lead the generator to learn a mapping which has the features required to trick the discriminator without being related to the input image. For this reason we often combine the discriminator loss with a conventional loss function such as SSIM or L1 (and these losses do require a pair of clean-noisy images). The GAN could still be used for semi-supervised learning by training with conventional losses and a paired dataset until

satisfying performance is met, then training further with a non-paired dataset using the GAN only. This could be beneficial for specialized images which would be tricky to include in a paired dataset, such as those depicting moving subjects and living faces.

While it may seem counter-intuitive to train the discriminator with mini-batches of only one label at a time, this practice may be justified with the use of Batch Normalization which keeps running statistics and normalizes the activations on the given batch, it could therefore perform better given low-variance input data. [25][14]

- Conditional Generative Adversarial Network [20] (**cGAN**) are **GANs** which take two inputs, the generated data and typically a label that describes it. An approach often used in image generation is to concatenate the corrupted image with the generated input and feed these image pairs as the input data rather than using a label. [15][32][26]. The benefit of potentially semi-supervised learning is no longer present but the discriminator is built-in with a correlative loss and the network is less likely to require a conventional loss function. While the conditional discriminator needs paired data to learn, a generator can still learn from unmatched data with a pretrained conditional discriminator.

CycleGAN[32] is a popular approach which enforces cyclic consistency, that is, the network must be able to regenerate the corrupted observation back from the generated one. We believe this approach would not function well for the problem of image denoising because of the random nature of noise, whereas CycleGAN applications tend to have simple outlines as the corrupted observation.

The original **GAN** and commonly used Deep Convolutional Generative Adversarial Network[23] (**DCGAN**) use the Binary Cross-Entropy (**BCE**) loss which is defined as $\text{BCE}(\text{prediction}, \text{target}) = -(\text{prediction} \cdot \log \text{target} + (1 - \text{prediction}) \cdot \log (1 - \text{target}))$. **BCE** assumes a Sigmoid final output activation and a target probability between 0 and 1, it is used in current image-to-image translation tasks such as [15][32].

The **GAN**'s log-based **BCE** loss suffers from vanishing gradient where the discriminator's gradient reaches zero thus the generator no longer receives any information to learn. Least Squares Generative Adversarial Network [19] (**LSGAN**) mitigates this issue using the Mean Squared Error loss function which penalizes samples based on their distance from the decision boundary (ie a 0.5 uncertainty gives out a loss of $0.5^2 = 0.25$ while a wrong 0.9 probability has a loss of $0.9^2 = 0.81$) and provides more stable training. [19]

4.2 CHOSEN GAN

We explored the aforementioned methods, namely [cGAN](#) and [GAN](#), [DCGAN](#), and [LSGAN](#). Our generator architectures are the U-Net [24] and Hulbnet mentioned in chapter 3. For the discriminator classifier we have experienced with PatchGAN [15] as well as our HulDisc architectures.

PatchGAN is a convolutional classifier intended to capture high-frequency details on small patches and leave the low-frequencies up to the ℓ_1 loss. The network architecture is made up of five convolutions with 4×4 kernel and 1 to 2 stride. The last layer returns one feature per patch instead of a single feature for the whole image. The overlapping patches have a 70×70 pixels receptive field, and it is up to the chosen loss [MSE](#) function to reconcile these losses into a single value.

HulDisc uses much the same architecture as the HulbNet generator but without the expanding half and with the addition of Batch Normalization. Our main discriminator focuses on a 112×112 crops, which allows for an undiscriminated-against 8-pixels per side border off the 128^2 -pixels generated images. The network is fully convolutional and encodes the whole image down to a single pixel/feature, with stride convolutions (stride=3) downscaling the crops from 102 to 34 and from 18 to 6 pixels, the rest being dense combinations of standard 3×3 convolutions and dilated convolutions (dilation=2) followed with [PReLU](#) and Batch Normalization [14]. We experimented with different combinations before getting to this configuration, namely the use of batch normalization, different (or no) activation functions, the use of a final pooling layer, the effect of mixed losses, the use of a bottleneck 1×1 convolution layer, and different numbers of filters.

4.3 METHOD

Our (c)GAN training procedures went through multiple experimental iterations. We first summarize the initial method in section 4.3.1 because is used in some of the experimental results, then section 4.3.2 goes over our adopted method which appears to provide more stable training with fewer parameters.

4.3.1 Initial training method

Our initial (c)GAN training method is similar to that of [15] but with an unequal learning ratio for the discriminator and generator (typically 1:3), because the generator does not learn well when the discriminator is too strong. The method consists of training the discriminator with both a real and a generated minibatch of images, then training the generator using a loss function made of the updated discriminator for

high-frequencies and the ℓ_1 loss for smooth textures. We enhanced the network further by adding the [SSIM](#) loss.

We found that a generative network trained with the discriminator from the start would never learn to perform well, but a pre-trained generative network could often perform better over time (although a balance must be reached because the discriminator cannot catch up to a too well-trained generator). We added a minimum SSIM score over a specified number of consecutive iterations before adding the discriminator as part of the generator's loss. Although the discriminator is being trained all along, the generator does not receive its feedback until the set threshold is met. From this point the loss function is a weighted combination of the discriminator's [MSE](#) loss (typically 0.75), the [SSIM](#) index (0.20) which ensures stability and image quality, and the ℓ_1 loss (0.05) which provides some smoothness to prevent excessive high-frequency details caused by the discriminator.

Balanced initialization is crucial as experimental work has shown that using the discriminator too soon results in a generator whose only focus is to trick the discriminator and the image quality suffers to the point where the generator loses and its gradient vanishes. Starting off with a well-trained generator on the other hand gives no chance to the discriminator which remains forever indecisive.

The network is still highly susceptible to failure after initialization. Common issues are overconfidence (with eventual mode collapse where the network outputs the same result), vanishing gradients, and indecision, therefore a good balance must be maintained all along the training process. We provide the discriminator's [MSE](#) loss with noisy labels as recommended in [25][8] (minus 0 to 0.05 probabilities on real images) to prevent overconfidence. There is typically a 0.33 to 0.5 training ratio between the discriminator and the generator, the imbalance ensures that the two networks are not constantly trying to overcome each other's latest update. This ratio is lowered to 0.1 when the discriminator gets too accurate (95%), and it is increased to 0.9 when the discriminator has approximately 50% or below accuracy. (50% is the ideal target but it often indicates an indecisive discriminator.)

4.3.2 Adopted (c)GAN training method

We needed a more robust method which can handle varying strength between the discriminator and the generator, that is, the generator must not always learn when the discriminator has to catch up, a constant ratio may end up promoting imbalance, and it may be beneficial for the two networks to learn on different batches of data rather than immediately adjusting for the other's network last changes.

Our current method uses the discriminator's normalized loss to determine whether the discriminator and the generator get to learn. A

denoised batch is generated, then the discriminator learns if its previous normalized loss is smaller than a uniformly distributed random number between 0 and 1. The generator then learns if the discriminator did not learn or if the discriminator's previous performance is greater than another uniformly distributed random number.

The discriminator's normalized loss is between 0 and 1 where 0.5 represents indecision or being right half of the time. For this we simply invert the MSE with $\sqrt{\text{MSE}(\text{predictions}, \text{labels})}$ for each set of labels and take the average of the two. The generator learns from the discriminator's current (updated) state rather than its previous state as in the previous method, because this saves significant memory and complexity. All thresholding parameters are removed, but we introduced a variable which can be used to give the discriminator an (dis)advantage; that is, whether a network learns is determined by the discriminator's previous loss plus that parameter. For example the intended use of the discriminator advantage variable would be that if it is set to 0.1 and the discriminator has a loss of 0.5, then the discriminator will have a $0.5 + 0.1 = 0.6$ probability of learning instead of 0.5 and the generator will learn with a probability of $0.5 - 0.1 = 0.4$.)

The (c)GAN training procedure is summarized as algorithm 1. The variables (losses and predictions) are tensors which may contain any number of scalars as well as the whole computation graph (history) and gradients needed to perform the backpropagation.

Algorithm 1 (c)GAN training procedure

```

1: for all clean_batch, noisy_batch  $\in$  Dataset do
2:   generated_batch  $\leftarrow$  generator(noisy_batch) {Denoised}
3:   {Train the discriminator}
4:   if discriminator learns then
5:     {discriminator learns = discriminator's previous normalized
6:      loss > random[0,1]}
7:     ADAM_optimizer_D.clear_gradient()
8:     if discriminator.is_conditional then
9:       fake_minibatch  $\leftarrow$  concatenate(
10:        remove_borders(noisy_batch),
11:        remove_borders(generated_batch))
12:       real_minibatch  $\leftarrow$  concatenate(
13:        remove_borders(noisy_batch),
14:        remove_borders(clean_batch))
15:     else
16:       fake_minibatch  $\leftarrow$  remove_borders(generated_batch)
17:       real_minibatch  $\leftarrow$  remove_borders(clean_batch)
18:     end if
19:     predictions_on_real_data  $\leftarrow$  discriminator(real_minibatch)
20:     loss_real_D  $\leftarrow$  MSE(
21:       predictions_on_real_data,
22:       generate_noisy_probabilities(true))
23:     loss_real_D.backpropagate()
24:     predictions_on_fake_data  $\leftarrow$ 
25:       discriminator(fake_minibatch.detach_from_graph())
26:     loss_fake_D  $\leftarrow$  MSE(predictions_on_fake_data, false)
27:     loss_fake_D.backpropagate()
28:     ADAM_optimizer_D.update_weights()
29:   end if
30:   {Train the generator}
31:   if generator learns then
32:     {generator learns = discriminator didn't learn or discrimina-
33:      tor's last normalized loss < random[0,1]}
34:     predictions_on_fake_data  $\leftarrow$  discriminator(fake_minibatch)
35:     loss_G_GAN  $\leftarrow$  MSE(predictions_on_fake_data, true)
36:     loss_G_SSIM  $\leftarrow$  SSIM(
37:       remove_borders(generated_batch),
38:       remove_borders(clean_batch))
39:     loss_G_l1  $\leftarrow$  l1(
40:       remove_borders(generated_batch),
41:       remove_borders(clean_batch))
42:     loss_G_weighted  $\leftarrow$ 
43:       loss_G_GAN  $\times$  (1 - weight_loss_SSIM - weight_loss_l1)
44:       + loss_G_SSIM  $\times$  weight_loss_SSIM
45:       + loss_G_l1  $\times$  weight_loss_l1
46:     loss_G_weighted.backpropagate()
47:     ADAM_optimizer_G.update_weights()
48:     ADAM_optimizer_G.clear_gradient()
49:   end if
50: end for

```

5

RESULTS

This chapter describes our use of [CNN](#) to denoise images from the [NIND](#). The first section describes our suggested methods to handle the [NIND](#) dataset. The [CNN](#) Denoiser section first describes the various conventional network configurations we have worked with along with the Experimental results. The [GAN](#) Denoiser section then describes our work and the results associated with (c)[GAN](#) networks which require more experimental work and fine-tuning.

5.1 DATASET USAGE

The [NIND](#) is cropped in advance to speed up loading times. A crop size of 128x128 pixels was found to work well for training and larger crops did not significantly affect performance. We found that the border is most often corrupted in a U-Net model. Some network architectures perform better when required to learn to model entire crops down to their border, but it typically takes a lot of resources to get to that point and the result often still shows a grid pattern when the crops are stitched back together. We use a "useful crop size" that is 0.75 the size of the actual crop size for U-Net models and approximately 0.875 with other architectures so that only the central part of a crop is used for stitching as well as for computing the loss. A script that crops the dataset in such overlapping blocks is provided for this purpose.

An epoch consists of training the model on every crop of any ISO value for every scene, that is, a random ISO value is fed every time a crop is loaded, therefore it takes several epochs for the model to train on all of the available data. The ground-truth is also selected randomly when multiple ones are available and basic data augmentation (rotation and/or translation) is performed.

5.2 CNN DENOISER

5.2.0.1 *Models*

We initially trained a DnCNN [28] model on our data. This model attained satisfying performance when trained to model the latent clean image instead of modeling the noise. It was further improved by using a convolution filter size of 5x5 instead of 3x3. The second architecture tested was a [RED-Net](#) [18] with 22-layers and a filter size of 5x5. This model obtained very good performance, albeit with an impractical runtime and memory use. We settled on a U-Net [24] architecture

which provides slightly better performance with significantly lower runtime and memory use.

5.2.1 Experimental results

We compared the performance obtained with the following methods:

1. U-Net trained on [NIND](#) ([X-T1](#) subset):
This model encompasses the main part of our dataset.
2. U-Net trained on Smartphone Image Denoising Dataset [1] ([SIDD](#)) (320 provided image pairs):
Compare the performance obtained using our dataset with 320 images from the [SIDD](#) (Smartphone Image Denoising Dataset) [1] which were made available for the 2019 NTIRE denoising challenge.
3. [BM3D](#) [7] ([11] implementation) with $\sigma = \{5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 93, 95, 97, 99\}$ ¹:
[BM3D](#) has been ubiquitously used as a reference in non-learning based image denoising.

In addition to the aforementioned reference methods, we consider the following experiments:

4. U-Net trained on [NIND](#) (dataset composed of the union of Fujifilm X-T1 mirrorless camera ([X-T1](#)) and Canon EOS 500D DSLR camera ([C500D](#)) training scenes; 89.5 % and 10.5 %, respectively):
When tested on a [C500D](#) image, this method can be compared with the first reference method to determine whether training on images acquired with the test image sensor helps or not, thereby assessing the generalization capabilities of our reference model to different sensors. In addition, it shows whether adding data from a different sensor negatively affects performance.
5. U-Net trained on the union of [NIND:X-T1](#), [NIND:C500D](#) scenes) and [SIDD](#) (320 pairs); 20.5 %, 2.4 %, and 77.1 % respectively:
This model shows the performance impact of adding a wildly different type of noise to the training data.
6. U-Net trained on [NIND](#) ([X-T1](#) subset, ISO6400 noise only instead of random ISO sample):
This experiment tests whether a model trained for blind denoising performs significantly worse than one trained for a specific ISO value.
7. U-Net trained on [NIND](#) ([X-T1](#) subset) with artificial gaussian noise added to the ground-truth. $\sigma = \{[1, 55], [1, 60], [1, 80], [1, 95]\}$ ¹:

¹ We test every σ value mentioned in Methods 3 and 7 and report the value which yields the highest [SSIM](#) for each test image.

This experiment compares the performance obtained by a model trained on our real data to the widely applied approach of applying synthetic gaussian noise to clean images.

8. U-Net trained to reconstruct the noise on [NIND \(X-T₁ subset\)](#):
We applied the residual learning strategy proposed in [28] by training a model that reconstructs the noise and subtracts it from the image.
9. [RED-Net](#) [18] trained on [NIND \(X-T₁ subset\)](#):
This uses the same data as Method 1 with a different network architecture

ISO value	ISO200	ISO250	ISO500	ISO2500	ISO4000	ISO6400	High ISO
Number of images	5	3	2	2	2	5	9
Noisy	1.000	0.907	0.853	0.784	0.687	0.578	0.311
NIND:X-T ₁ (U-Net)	0.949	0.929	0.920	0.912	0.900	0.893	0.851
SIDD (U-Net)	0.906	0.907	0.904	0.864	0.882	0.860	0.814
BM ₃ D	0.941	0.925	0.913	0.875	0.870	0.852	0.785
NIND :X-T ₁ +C500D (U-Net)	0.949	0.929	0.920	0.912	0.899	0.893	0.851
NIND :X-T ₁ +C500D + SIDD (U-Net)	0.947	0.928	0.919	0.910	0.898	0.892	0.850
NIND:X-T ₁ ISO6400 only (U-Net)	0.919	0.915	0.911	0.907	0.901	0.894	0.821
Reconstruct noise on NIND:X-T ₁ (U-Net)	0.950	0.926	0.914	0.901	0.876	0.840	0.664
Artificial noise on NIND:X-T ₁ (U-Net)	0.963	0.920	0.899	0.880	0.810	0.769	0.531
NIND:X-T ₁ (RED-Net)	0.940	0.923	0.915	0.907	0.892	0.886	0.842

Table 5.1: Average [SSIM](#) index on 5 [NIND:X-T₁](#) denoised scenes (ursulines-building, stefantiek, CourtineDeVillersDebris, MuseeL-Bobo, ursulines-red). The best performing models (to within two significant digits) are marked in bold.

ISO value	ISO100	ISO200	ISO400	ISO800	ISO1600	ISO3200	High ISO
Noisy	1.000	0.814	0.754	0.660	0.550	0.401	0.172
NIND:X-T ₁ (U-Net)	0.911	0.901	0.898	0.896	0.893	0.887	0.868
SIDD (U-Net)	0.894	0.892	0.890	0.889	0.885	0.876	0.850
BM ₃ D	0.921	0.890	0.884	0.877	0.871	0.860	0.813
NIND :X-T ₁ +C500D (U-Net)	0.911	0.901	0.899	0.896	0.894	0.888	0.872
NIND :X-T ₁ +C500D + SIDD (U-Net)	0.912	0.901	0.899	0.896	0.893	0.888	0.871
NIND :X-T ₁ ISO6400 only (U-Net)	0.899	0.897	0.896	0.894	0.892	0.886	0.865
Reconstruct noise on NIND :X-T ₁ (U-Net)	0.908	0.895	0.887	0.875	0.855	0.807	0.617
Artificial noise on NIND :X-T ₁ (U-Net)	0.946	0.879	0.864	0.836	0.802	0.716	0.430

Table 5.2: [SSIM](#) index on [NIND:C500D](#) denoised set MuseeL-Bobo-C500D

ISO value	ISO100	ISO200	ISO400	ISO800	ISO1600	ISO3200	High ISO
# images	13	9	9	9	8	10	9
Noisy	0.954	0.766	0.707	0.619	0.501	0.380	0.220
NIND:X-T ₁ (U-Net)	0.878	0.856	0.854	0.848	0.845	0.834	0.805
SIDD (U-Net)	0.851	0.838	0.837	0.834	0.830	0.813	0.774
BM ₃ D	0.899	0.836	0.825	0.816	0.811	0.789	0.749
NIND:X-T ₁ ISO6400 only (U-Net)	0.857	0.847	0.847	0.845	0.843	0.834	0.802
Reconstruct noise on NIND:X-T ₁ (U-Net)	0.878	0.845	0.835	0.814	0.785	0.725	0.604

Table 5.3: Average [SSIM](#) index on 10 [NIND:C500D](#) scenes denoised with models trained on [NIND](#):X-T₁ or with BM₃D

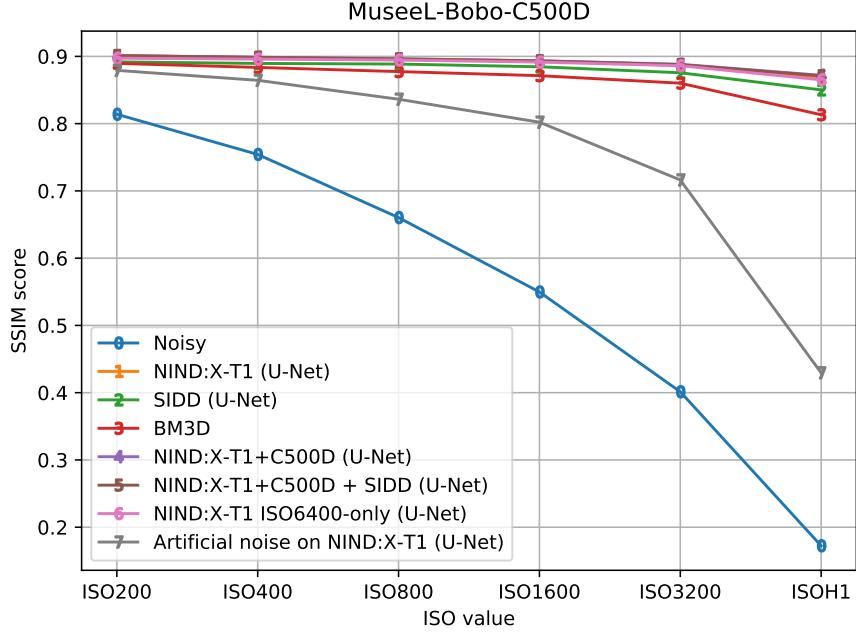


Figure 5.1: Denoising performance of the MuseeL-Bobo-C500D set over increased ISO value (SSIM values shown in Table 5.2)



Figure 5.2: Denoising stefantiek. 1: ISO200 ground-truth (1/20s), 2: high ISO (1/1900s), 3: 2 denoised using U-Net model trained with NIND, 4: 2 denoised using U-Net model trained with SIDD (320-sets), 5: 2 denoised using BM3D ($\sigma = 90^{1}$)

Sépik Masque	ISO100 GT	Sépik Masque	High ISO Sépik Masque	NIND:X-T1 Sépik Masque	SIDD:320 Sépik Masque	BM3D: $\sigma=99$
Nouvelle-Guinée 20° s. Bois polychromé N° inv. NE74 Legs Dr Ch. Delsenne	Nouvelle-Guinée 20° s. Bois polychromé N° inv. NE74 Legs Dr Ch. Delsenne	Nouvelle-Guinée 20° s. Bois polychromé N° inv. NE74 Legs Dr Ch. Delsenne	Nouvelle-Guinée 20° s. Bois polychromé N° inv. NE74 Legs Dr Ch. Delsenne	Nouvelle-Guinée 20° s. Bois polychromé N° inv. NE74 Legs Dr Ch. Delsenne	Nouvelle-Guinée 20° s. Bois polychromé N° inv. NE74 Legs Dr Ch. Delsenne	Nouvelle-Guinée 20° s. Bois polychromé N° inv. NE74 Legs Dr Ch. Delsenne

Figure 5.3: Denoising text present on MuseeL-Sepik-C500D. 1: ISO200 ground-truth (2s), 2: high ISO (1/30s), 3: 2 denoised using U-Net model trained with NIND:X-T1, 4: 2 denoised using U-Net model trained with SIDD (320-sets), 5: 2 denoised using BM3D ($\sigma = 99^{1}$)



Figure 5.4: Comparison between a noisy ISO6400 crop (top), one denoised with a model trained on [NIND](#) (middle), and one on which [BM₃D](#) ($\sigma = 30$) has been applied (bottom). Our model appears to perform well on dynamic scenes despite having been trained on static scenes.

Each network is trained for 48-hours on a GeForce GTX 1080 (11GB). Table 5.1 shows denoising performance on the Fujifilm X-T1 test pictures. We observe that the model trained on the NIND significantly outperforms BM3D and that adding training data from the Canon EOS 500D sensor, as well as part of the SIDD dataset, does not appear to negatively impact performance. Table 5.2 and Figure 5.1 show denoising performance on the scene "MuseeL-Bobo-C500D", where a model trained only with NIND:X-T1 data performs nearly as well as a model that was also trained with NIND:C500D data (and so does a model trained with both NIND and SIDD). Table 5.3 summarizes the average performance of X-T1-trained models on ten C500D scenes and shows performance which considerably exceeds that of BM3D even though the model was generalizing for a different sensor type.

A model trained with only NIND:X-T1 ISO6400 noisy images yields slightly better performance at and around ISO6400, but this comes with a considerable loss of detail at low ISO and the denoising performance becomes poor as the noise level increases. Moreover the model trained on Fujifilm X-T1 ISO6400 images appears not to generalize as well to different sensors as we found it consistently performs worse on the Canon 500D images. These findings suggest that the cost of generalization is acceptably low and therefore a model mostly benefits from learning with different noise levels and sensors.

Reconstructing the noise (Method 8) as was suggested in [28] typically yields performance below that of BM3D when applied to ISO noise. The difficulty in reconstructing ISO noise was further noticed in an experiment where we mistakenly fed our learning model noisy images as ground-truth 31% of the time and it still exceeded BM3D performance. This went as far as inverting the clean and noisy crops and still learning an appreciable level of denoising. These findings suggest that a model can easily tolerate some noise in the ground-truth data. Research into this topic [17] has been performed to explicitly train models on noisy data and rely on the zero-mean nature of the noise to effectively remove various artificial noise distributions.

Training a model using artificial noise added to ground-truth images (Method 7), as is commonplace in the literature [18][28], yields the worst performance in our tests.

In addition to the aforementioned results based on SSIM, we have subjectively tested our NIND-trained model on single images which are not part of the dataset. The first such image is that of a dynamic outdoor scene in which a human walks towards a group of pigeons, causing them to disperse in multiple directions. This type of fast, moving scene cannot be included in the dataset due to its dynamic nature and it must be captured with settings that result in a poor quality image; a small aperture ($f/11$) to focus everywhere, a fast shutter speed ($1/1500$ s) to capture the flying birds, and a maximum sensor sensitivity (ISO6400) to match the aforementioned settings.

Nonetheless, we found the denoised image to be of high quality; we submitted it to the Wikimedia Commons “Quality Images Candidates” page [6] and it was subsequently promoted to a “Quality Image” by Wikimedia Commons reviewers. A crop of the image is provided on Figure 5.4 with a comparison between the noisy version, one denoised with a U-Net model trained on NIND, and a version that has been denoised using BM3D (with $\sigma = 30$ which, on average, yields the highest SSIM in our ISO6400 test images). The BM3D version shows significant displeasing artifacts, for example on the skirt and the blue uniform panel on the right, while the model trained on NIND smoothed these regions appropriately while retaining a greater level of useful details such as those present on the pigeons’ wings.

5.3 GAN DENOISING

We first tried following the pix2pix approach (whose typical use-case is rough sketches to photorealistic images) of performing image to image translation using a PatchGAN discriminator and a U-Net generator [15] with 1:3 training ratio. The discriminator handles high-frequency details and low-frequencies are handled by the ℓ_1 loss. We were unable to obtain satisfying performance after experimenting with different loss balances, we found that the denoising performance was subpar to that of BM3D and of our previous U-Net network; high-frequencies tend to be exaggerated in low-frequency areas and high frequency details are not properly recovered, often resulting in an overall poor SSIM index. Figure 5.5 illustrates the denoising performance obtained using a 0.25 ℓ_1 multiplier.



Figure 5.5: Comparison of standard pix2pix [15] applied to image denoising with other methods on a crop from image NIND_MuseeL-Bobo_ISOHz2. 1: ISO200 ground-truth crop (SSIM: 1.00), 2: ISOHz2 noisy crop (SSIM: 0.151), 3: Denoised image with a single U-Net network trained on the NIND (SSIM: 0.810), 4: Denoised image with a standard pix2pix network trained on the NIND with $0.25 \times \ell_1$ (SSIM: 0.672), 5: Denoised image with a pix2pix network trained on the NIND with the following adjustments: minimum of 35 iterations with $\text{loss}_{\text{SSIM}} < 0.125$ before using the discriminator with $\text{loss}_G = 0.5\text{loss}_D + 0.4\text{loss}_{\text{SSIM}} + 0.1\text{loss}_{\ell_1}$ (SSIM: 0.794), 6: Denoised with BM3D using $\sigma = 99$ (SSIM: 0.694)

Some improvement were made with careful fine-tuning of the learning environment. We added the **SSIM** loss used in single-network training and tested different balances between the ℓ_1 , **SSIM**, and discriminative losses. We tried training the generator without applying a discriminative loss until a **SSIM** threshold is met for a given number of consecutive iterations, and we tried training a non conditional network. Figure 5.6 shows performance on the "stefantiek" set using different settings on a non conditional network (as well as the baseline network performance without these adjustments). We observe particularly poor performance when the discriminative loss has a high weight (weight_D = 0.9), as well as when we start using the discriminative loss early on before the generator has learned to properly denoise with the conventional loss functions (min_SSIM=0.3). The GAN network obtains reasonable performance when the generator is pretrained to achieve a **SSIM** loss of 0.125 to 0.15. The greatest performance shown occurs when the generator has to obtain loss_{SSIM} < 0.125 for 35 consecutive iterations before using the discriminator as half of the overall loss. A crop denoised with this model (5) is shown beside one trained with standard pix2pix **cGAN** (4) on Figure 5.5.

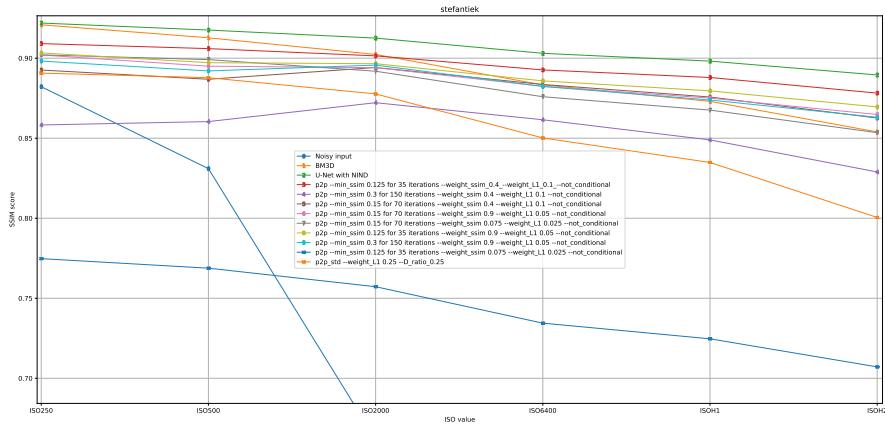


Figure 5.6: Denoising performance of the non conditional GAN with varying weights and schedules on the "stefantiek" set over increased ISO values

Network like [26] which processes multiple scales to capture high-frequencies and smooth textures. Instead of dividing up the images into different resolutions or using multiple networks, we followed an approach similar to that of DenseNet [12] which concatenates multiple channels together from different layers (as opposed to summing the features [resnet]), allowing layers to contain different types of features captured with different convolutions. Some of the convolutions thus capture a wider receptive field using dilation while other focus on local details.

At the same time we wanted to replace the U-Net network which loses too much border that is either easily discriminated against or

does not leave a large enough crop for the discriminator to make up meaningful results. We thus designed both a generator and a discriminator which follow the same design ideas (alas the discriminator is cut in half and loses many of the dense connections), and tested specific components for their respective use-cases.

The two architectures were trained separately to assess and optimize their individual performance before working together. The generator is typically trained using the same framework as that of section ??, while the discriminator main tests involve discriminating between clean and noisy crops from the dataset without receiving the conditional input. This test is not as robust as training with the result of a static generator but it allows us to quickly get a preview of the discriminator's performance whereas it would not always be able to learn from scratch given a well trained generator.

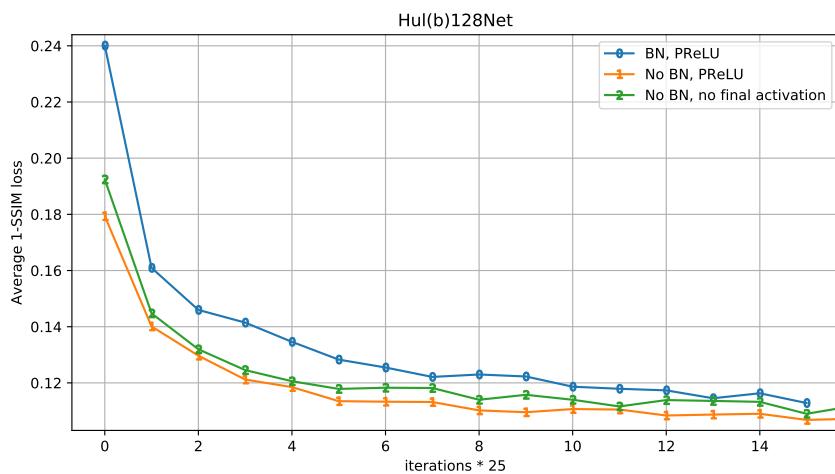


Figure 5.7: Generative network learning curve with and without Batch Normalization [14], as well as with neither BN nor an activation function.

Batch Normalization appeared to hamper learning on the generative networks, often yielding worse color rendition and lowering the SSIM index (which does not appear to be very sensible to color accuracy). We trained a generative network for an epoch with and without Batch Normalization to finally determine whether BN should be removed from the generative network. Figure 5.7 shows the resulting learning curve, which allows us to confidently remove BN from the generative architecture (which is now called "HulbNet"). We also trained a network for an epoch to determine whether not using a final loss function as is commonly done (TODO cite) would yield better performance than PReLU, but we found it better to continue ending with an activation function.

Figure 5.8 shows the discriminator's learning curve classifying images as either clean or noisy. We observe that Batch Normalization [14] is useful in the case of the discriminator, likely because its job

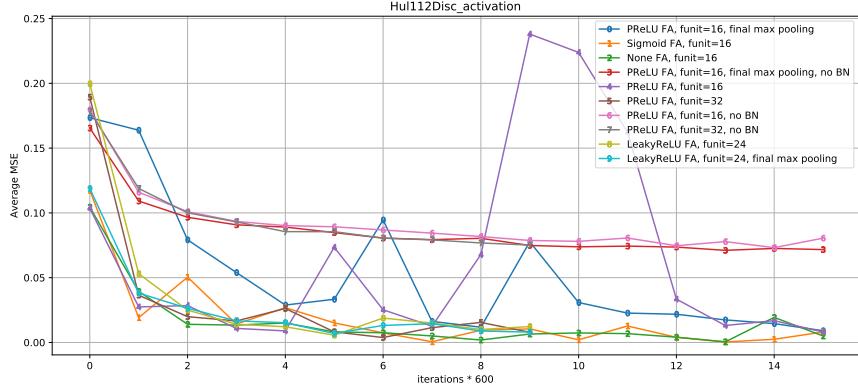


Figure 5.8: Discriminative network learning curve with varying final activation functions (FA) and number of filters. One funit denotes the number of filters multiplier (with the first layer outputting two funits). The network is not conditioned and its task is to discriminate whether an image is clean or noisy. The loss is squared therefore a MSE of 0.25 is equivalent to guessing 0.5.

does not entail recreating image fidelity from the input but output a single probability, and **Batch Normalization!** can help when given mini-batches of a single target value (line 3, 6, 7). We observe that the PReLU activation function yields sporadically very poor performance on line 0 and 4, but performance is restored with a greater number of filters. Ending the discriminative network with a final max-pooling appears worse than a learned convolution. The different activation functions seem roughly equivalent as they overtake one other over time. Although we observe the importance of using a sufficient number of filters, we cannot conclude an actual number of needed filters from this experiment because the task is simpler than discriminating between real (baseline) and fake (denoised) images, but the real task at hand cannot be accomplished because the discriminator never catches up if there is a large discrepancy between an untrained discriminator and a pre-trained generator.

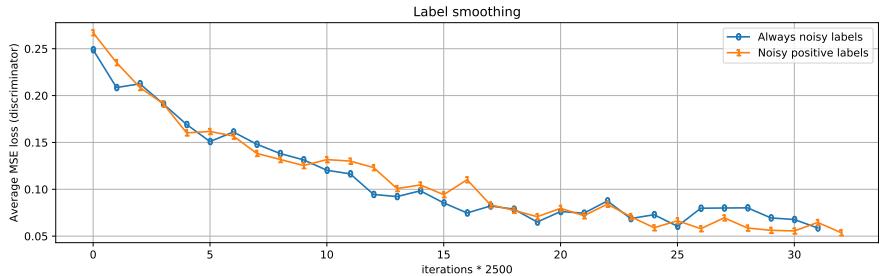


Figure 5.9: Discriminative network performance (MSE loss) with noisy labels (either always noisy or noisy for positive labels only). The network attempts to discriminate images denoised by a generator which has been trained for three epochs and is continuously being updated (alas without any input from the discriminator).

Following some incertitude on whether label smoothing should only be applied on positive labels as per [25] or on all batches used to train the discriminator, we compared the two approaches and show the results on figure 5.9. The test protocol is different from the previous one in that the discriminator attempts to discriminate against images denoised by a generator which has been pre-trained for three epochs (and is continuously being updated, although it receives no feedback from the discriminator). The results we observed did not seem to differ significantly as the two discriminators would continuously overtake one other.

TODO (in-progress): HulbNet w/ HulDisc, HulbNet w/PatchGAN (done), HulbNet w/HulfDisc

(mention that training with weight_D >= 0.95 is not stable enough)

TODO: discriminator_advantage

TODO conditional vs not conditional

TODO invert probabilities

TODO higher - lower weights

6

CONCLUSION

6.1 FUTURE WORK

- train a general network with specialized dataset
 - WGAN
- batch same noise samples or samples from one image/noise at a time to take advantage of BN
 - further fine-tuning of the discriminator: eg BN before/after, lighter generator
- PatchGAN needs carefully balanced loss because high frequencies

BIBLIOGRAPHY

- [1] Abdelrahman Abdelhamed, Stephen Lin, and Michael S. Brown. "A High-Quality Denoising Dataset for Smartphone Cameras." In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.
- [2] Josue Anaya and Adrian Barbu. "RENOIR – A dataset for real low-light image noise reduction." In: *Journal of Visual Communication and Image Representation* 51 (Sept. 2014). doi: [10.1016/j.jvcir.2018.01.012](https://doi.org/10.1016/j.jvcir.2018.01.012).
- [3] Benoit Brummer. *Natural Image Noise Dataset - Wikimedia Commons*. https://commons.wikimedia.org/wiki/Natural_Image_Noise_Dataset.
- [4] A. Buades, B. Coll, and J. . Morel. "A non-local algorithm for image denoising." In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 2. 2005, 60–65 vol. 2. doi: [10.1109/CVPR.2005.38](https://doi.org/10.1109/CVPR.2005.38).
- [5] Chen Chen, Qifeng Chen, Jia Xu, and Vladlen Koltun. "Learning to See in the Dark." In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.
- [6] Commons:Quality images candidates - Wikimedia Commons. https://commons.wikimedia.org/wiki/Commons:Quality_images_candidates.
- [7] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. "Image denoising with block-matching and 3D filtering." In: *Proceedings of SPIE - The International Society for Optical Engineering* 6064 (Feb. 2006), pp. 354–365. doi: [10.1117/12.643267](https://doi.org/10.1117/12.643267).
- [8] Ian J. Goodfellow. "NIPS 2016 Tutorial: Generative Adversarial Networks." In: *CoRR* abs/1701.00160 (2017). arXiv: [1701.00160](https://arxiv.org/abs/1701.00160). URL: <http://arxiv.org/abs/1701.00160>.
- [9] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative Adversarial Nets." In: *Advances in Neural Information Processing Systems* 27. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., 2014, pp. 2672–2680. URL: <http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf>.

- [10] K. He, X. Zhang, S. Ren, and J. Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1026–1034. DOI: [10.1109/ICCV.2015.123](https://doi.org/10.1109/ICCV.2015.123).
- [11] David Honzátko and Martin Kruliš. "Accelerating block-matching and 3D filtering method for image denoising on GPUs." In: *Journal of Real-Time Image Processing* (Nov. 2017), pp. 1–15. DOI: [10.1007/s11554-017-0737-9](https://doi.org/10.1007/s11554-017-0737-9).
- [12] G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger. "Densely Connected Convolutional Networks." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2261–2269. DOI: [10.1109/CVPR.2017.243](https://doi.org/10.1109/CVPR.2017.243).
- [13] *Hugin - Panorama photo stitcher*. <http://hugin.sourceforge.net/>.
- [14] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." In: *CoRR* abs/1502.03167 (2015). arXiv: [1502.03167](https://arxiv.org/abs/1502.03167). URL: <http://arxiv.org/abs/1502.03167>.
- [15] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. "Image-to-Image Translation with Conditional Adversarial Networks." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. 2017, pp. 5967–5976. DOI: [10.1109/CVPR.2017.632](https://doi.org/10.1109/CVPR.2017.632). URL: <https://doi.org/10.1109/CVPR.2017.632>.
- [16] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015. URL: <http://arxiv.org/abs/1412.6980>.
- [17] Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. "Noise2Noise: Learning Image Restoration without Clean Data." In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholm, Sweden: PMLR, 2018, pp. 2965–2974. URL: <http://proceedings.mlr.press/v80/lehtinen18a.html>.
- [18] Xiao-Jiao Mao, Chunhua Shen, and Yu-Bin Yang. "Image Restoration Using Very Deep Convolutional Encoder-Decoder Networks with Symmetric Skip Connections." In: *Proc. Advances in Neural Inf. Process. Syst.* 2016.
- [19] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, and Zhen Wang. "Multi-class Generative Adversarial Networks with the L₂ Loss Function." In: *CoRR* abs/1611.04076 (2016).

- [20] Mehdi Mirza and Simon Osindero. "Conditional Generative Adversarial Nets." In: *CoRR* abs/1411.1784 (2014). arXiv: 1411.1784. URL: <http://arxiv.org/abs/1411.1784>.
- [21] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines." In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, 2010, pp. 807–814. ISBN: 978-1-60558-907-7. URL: <http://dl.acm.org/citation.cfm?id=3104322.3104425>.
- [22] Tobias Plotz and Stefan Roth. "Benchmarking Denoising Algorithms With Real Photographs." In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.
- [23] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks." In: *ICLR*. 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)*. Vol. 9351. LNCS. (available on arXiv:1505.04597 [cs.CV]). Springer, 2015, pp. 234–241. URL: <http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a>.
- [25] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. "Improved Techniques for Training GANs." In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., 2016, pp. 2234–2242. ISBN: 978-1-5108-3881-9. URL: <http://dl.acm.org/citation.cfm?id=3157096.3157346>.
- [26] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. "High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs." In: *CoRR* abs/1711.11585 (2017). arXiv: 1711.11585. URL: <http://arxiv.org/abs/1711.11585>.
- [27] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. "Image Quality Assessment: From Error Visibility to Structural Similarity." In: *IEEE Transactions on Image Processing* 13 (Apr. 2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.
- [28] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. "Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising." In: *IEEE Transactions on Image Processing* 26.7 (2017), pp. 3142–3155.

- [29] Wei Zhang, Kazuyoshi Itoh, Jun Tanida, and Yoshiki Ichioka. “Parallel distributed processing model with local space-invariant interconnections and its optical architecture.” In: *Appl. Opt.* 29.32 (1990), pp. 4790–4797. DOI: [10.1364/AO.29.004790](https://doi.org/10.1364/AO.29.004790). URL: <http://ao.osa.org/abstract.cfm?URI=ao-29-32-4790>.
- [30] Yide Zhang, Yinhao Zhu, Evan Nichols, Qingfei Wang, Siyuan Zhang, Cody Smith, and Scott Howard. “A Poisson-Gaussian Denoising Dataset with Real Fluorescence Microscopy Images.” In: *CoRR* abs/1812.10366 (2018). arXiv: [1812.10366](https://arxiv.org/abs/1812.10366). URL: <http://arxiv.org/abs/1812.10366>.
- [31] H. Zhao, O. Gallo, I. Frosio, and J. Kautz. “Loss Functions for Image Restoration With Neural Networks.” In: *IEEE Transactions on Computational Imaging* 3.1 (2017), pp. 47–57. ISSN: 2333-9403. DOI: [10.1109/TCI.2016.2644865](https://doi.org/10.1109/TCI.2016.2644865).
- [32] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networkss.” In: *Computer Vision (ICCV), 2017 IEEE International Conference on*. 2017.
- [33] *darktable*. <https://www.darktable.org/>.
- [34] *profiling sensor and photon noise | darktable*. <https://www.darktable.org/2012/12/profiling - sensor - and - photon - noise/>. Dec. 2012.