ABSTRACT

Convolutional neural networks have been the focus of research aiming to solve image denoising problems, but their performance remains unsatisfactory for most applications. These networks are trained with synthetic noise distributions that do not accurately reflect the noise captured by image sensors. Some datasets of clean-noisy image pairs have been introduced but they are usually meant for benchmarking or specific applications. We introduce the Natural Image Noise Dataset (NIND), a dataset of DSLR-like images with varying levels of ISO noise which is large enough to train models for blind denoising over a wide range of noise. We demonstrate the use of denoising models trained with the NIND and show that they significantly outperform BM3D on ISO noise from unseen images, even when generalizing to images from a different type of camera. We expect that this dataset will prove useful for future image denoising applications. Finally we investigate the use of conditional generative adversarial networks (cGAN) which may provide better denoising performance in some edge cases that a paired dataset may not cover, and we develop a framework which lays the groundwork to train such networks as well as a pair of discriminator-generator architectures that perform well in such systems.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

NIND Natural Image Noise Dataset [5][4]

BCE binary cross-entropy

BM3D block-matching and 3D filtering [9]

BN batch normalization [16]

cGAN conditional generative adversarial network [22]

CNN convolutional neural network [33]

DND Darmstadt Noise Dataset [24]

DCGAN deep convolutional generative adversarial network [25]

DSLR digital single-lens reflex

GAN generative adversarial network [11]

GPU graphics processing unit

LSGAN Least Squares Generative Adversarial Network [21]

MP megapixel

MSE mean squared error

RED-Net very deep Residual Encoder-Decoder Network [20]

ReLU rectified linear unit [23]

PReLU parametric rectified linear unit [12]

SIDD Smartphone Image Denoising Dataset [1]

SID See-in-the-Dark [7]

SSIM Structural Similarity [30]

WGAN	Wasserstein GAN [3]
C500D	Canon EOS 500D DSLR camera
X-T1	Fujifilm X-T1 mirrorless camera
k	kernel size
p	padding size

1 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 [20].

Photographic development usually incorporates a conventional denoising method such as non-local means [6] or wavelets-based methods [27], or a combination thereof, sometimes making use of prior information about a specific sensor's response to different ISO noise values [38].

BM3D [9] is a more powerful denoising method which has been used to benchmark newer techniques [1][2][5][7][19][20][24][32][34][35], though it is seldom used in off-the-shelf software (presumably because of its high computational cost and the need to specify a σ 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 [32], 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 block-matching and 3D filtering [9] (BM3D) [24]. 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 [7][34]. 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-like[2] 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 [32], very deep Residual Encoder-Decoder Network [20] (RED-Net), and U-Net [26], and we analyzed different methods such as DnCNN [32]'s proposed residual learning. These models and methods show state-of-the-art results when presented with synthetic noise, but their performance on real photographic noise removal remained untested until now due to the lack of an appropriate dataset.

In most tests a U-Net model was trained with the Natural Image Noise Dataset [5][4] (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 network [11]s [17] which may provide a solution to specific issues such as unnatural face smoothing and unmatched data.

We expect the Natural Image Noise Dataset [5][4] will be useful for research and applications in image noise removal, because it provides the data needed for neural networks to outperform conventional methods. Our trained models' performance consistently exceed that of BM3D and of models trained with artificial noise. The results we provide with generative neural networks provide a solution that is

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.

usable immediately, and the functional generative adversarial network [11] methods we explored provide groundwork for further work and application-specific use-cases.

2 DATASET

Chapter 2 discusses the existing datasets that are relevant to the denoising problems and the Natural Image Noise Dataset [5][4] provided with this work. The datasets presented in Darmstadt Noise Dataset [24] (DND) and Renoir [2] were created for benchmarking purposes and, as such, they outline the usefulness of a proper image noise dataset that may be used to train a denoising model in place of using synthetic noise. Other works provide paired datasets that are large enough to train a denoising model and the authors successfully demonstrate the performance of such models, but they were designed with specific applications in mind, namely cell-phone [1] and low-light imaging [7]. We introduce the Natural Image Noise Dataset [5][4] for general-purpose image denoising on images from conventional digital single-lens reflex (DSLR) and mirrorless cameras. Section 2.2 describes the content of the NIND, the complete acquisition process, and the steps needed for publication (because the dataset is open to contributions) and retrieval.

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 [30] (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 [24], 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 [9], which was published in 2006, remains one of the best performing methods [24]. RENOIR [2] is a similar dataset that was published prior to the DND; however, Plötz and Roth noted spatial misalignments that reduced its effectivity. 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 [7] (SID) 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 [1] (SIDD) 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

The Natural Image Noise Dataset [5][4] (NIND) fills the gap for an image noise dataset of ground-truth - noisy image sets that targets cameras equipped with larger sensors[2] 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

Table 2.1: Content of 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		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				10	11	74
Total	15	116	11	8	29	13	16	36	16	9	22	9	19	37	17	19	91	133	101	616

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

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.

A base ISO image (ISO200 on the Fujifilm X-T1 mirrorless camera) 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 1/60 s for handheld photography, 1/15 s for devices equipped with optical image stabilization, and 1/1000 s 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 1/13000 s 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, 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 mirrorless camera, which uses a 23.6 x 15.6 mm X-Trans sensor. A Canon EOS 500D DSLR camera (C500D), featuring a 22.3 x 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 [37] (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 [15] 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 [8] 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 directly[1]. 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

We use generative neural networks to generate denoised images from their noisy representation and thus solve the denoising problem. The class of generative network used to complete this task is a convolutional neural network. We introduces the notion of a convolutional neural network [33] and its components in section 3.1, then section 3.2 goes over each specific network architecture we have worked with. All networks presented is this chapter are generative; a generative network takes an image of $W \times H$ pixels with three channels and generates a new image of the same dimensions. It differs from other types of convolutional neural network [33]s which have different output shapes, for example a classification network (such as those used in chapter 4) outputs one to a few probabilities, and segmentation networks usually have the same height and width dimensions as the input image but one channel per segmented feature (segmentation networks can easily be adapted into generative models as is the case for U-Net [26]).

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 multilayer perceptron and a convolutional neural network [33] (CNN) is that a CNN is not fully connected so 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. This process is shown in figure 3.1 where one filter is applied across the previous (in this case first) layer to create one channel of the following layer.

The first and last layer of a generative network have the same shape as the input image, that is its height, width, and 3-channels representing the RGB color-space. Most layers in a deep network are hidden layers between the first and last one. Hidden layers contain many more channels (also called feature maps), they 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-

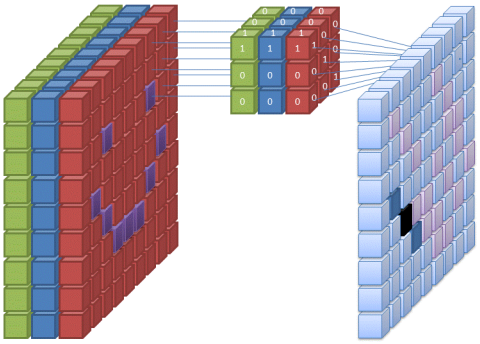

Figure 3.1: A filter is applied to the first 3-channels (RGB) input of an image in a convolutional neural network. The same filter is applied across all channels of the whole image as shown, and many of these filters are applied to form the following layer's feature map (i.e. its channels). Illustration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

propagation algorithm, and the Adam gradient-descent algorithm[18] (the optimizer) adjusts the weights at the end of each batch training iteration. The optimizer's learning rate needs to be fine-tuned in order to optimize fast learning without over-fitting.

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 height[1], taking in every channel and returning one value per filter. Convolutions have a receptive field that is determined by kernel size (k) whose typical values are 3 or 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 $d - k + 2p + 1$. 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, CNNs 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 ($\phi(z) = \frac{1}{1+e^{-z}}$) which constrains its input values between 0 and 1, and (parametric) rectified linear unit [23] (ReLU) ($R(z) = \max(0, z)$) as activation functions.

1 assuming a stride of 1

3.1.2 *Loss function*

The loss function assigns a score to the generated images with respect to the provided ground-truth. A loss function should not simply compare the difference between generated and ground-truth pixels 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. [17]

The MSE (or $\ell 2$ loss) measures the average squared difference between the generated and ground-truth pixels ($MSE(\text{generated}, \text{ground-truth}) = \sum_{p}^{\text{pixels}} \frac{(\text{generated}_p - \text{ground-truth}_p)^2}{|\text{pixels}|}$), it is a simple yet effective loss function. Similarly, the $\ell 1$ 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 an generative adversarial network [11].

Mixing different losses often appears to yield better results [35], 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. [35] We typically use the SSIM index in our standard models and a combination of loss functions in the more challenging generative adversarial network [11] (GAN).

3.2 GENERATIVE NETWORK ARCHITECTURES

3.2.1 *DnCNN*

DnCNN is a simple flat CNN architecture introduced by Zhang et al. [32] to denoise images with residual learning. The network performs end-to-end denoising using only a convolution layer followed by batch normalization [16] 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 [32] 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 because of its simplicity, 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 in order to assess whether it would yield any gain in performance.

3.2.2 *RED-Net*

The RED-Net architecture is a deep encoder-decoder network with residual skip connections [20]. We found this type of network offers significantly better denoising performance and faster convergence than the DnCNN model. 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 in Zhang et al. [32]. A residual network has skip connections which in the case of RED-Net 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 3x3 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 2x2 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:

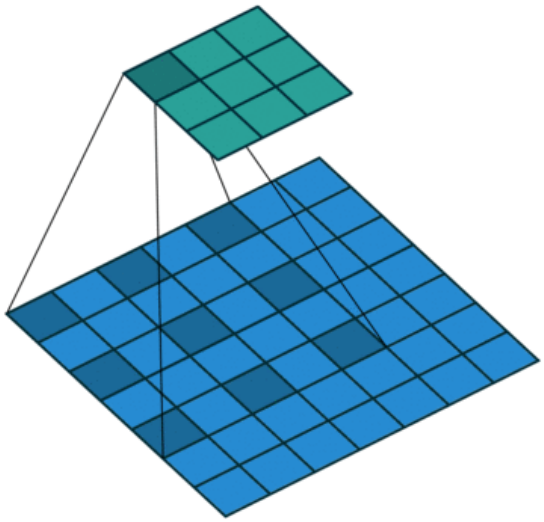

Figure 3.2: Dilated convolution (dilation=2). Illustration by GitHub user vdumoulin (MIT license)

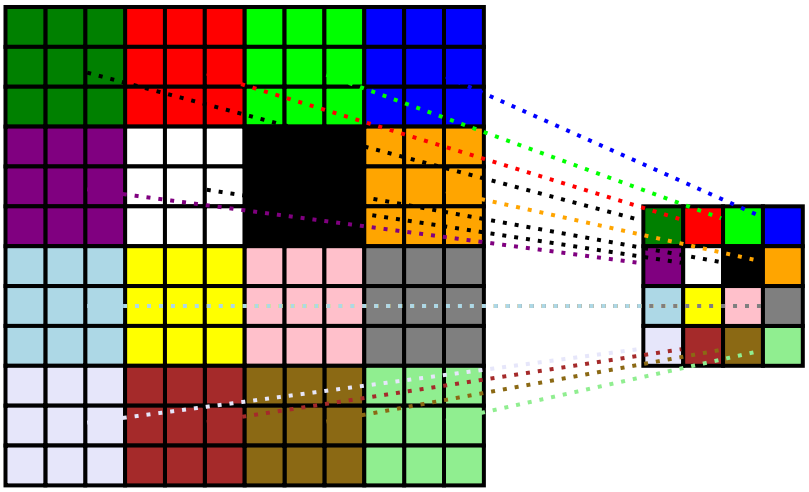

Figure 3.3: 3x3 convolution with stride=3

- **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. Dilated convolutions have been shown to perform well in image segmentation tasks [31].

- **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 3x3 convolution. We apply convolution with a stride of three instead of using a 2x2 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 [14]. Residual connections such as those used in RED-Net [20] and U-Net [26] are element-wise sum of the feature maps, as such the feature maps are combined and the network can no longer choose 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 3x3 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

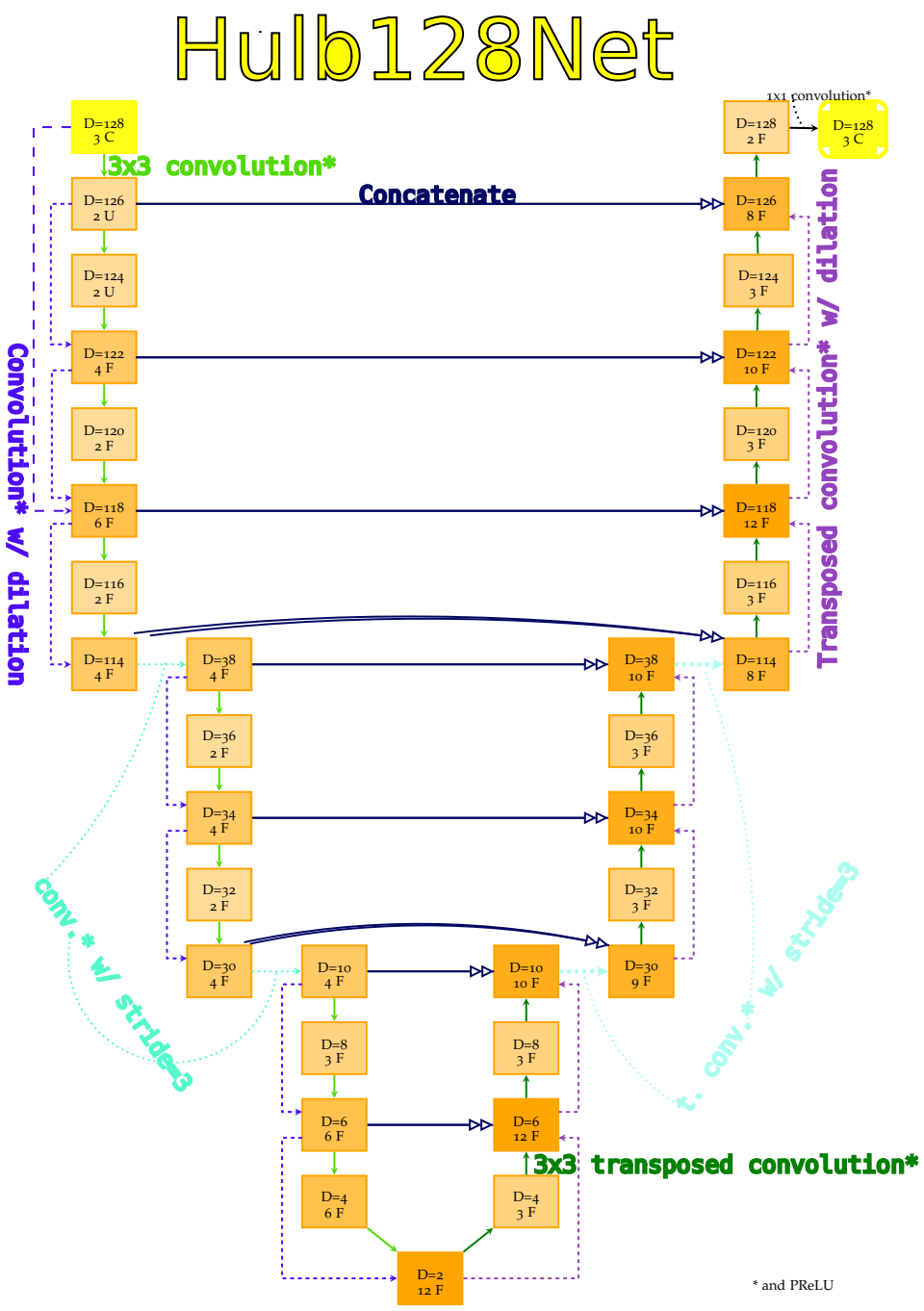

Figure 3.4: The HulbNet architecture is illustrated above. D indicates the height and width dimension when starting with a 128×128 pixel crop. F denotes a "filter unit", which typically consists of 32 filters. This network can work with input crops of dimension $119 + 9 \times n$ where $n \geqslant 0$.

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 1x1 convolution which reduces the number of features down to three channels. Each convolution is followed by a parametric rectified linear unit [12] (PReLU) activation and batch normalization [16] was not applied because experimental results showed worse performance using it in denoised image generation. The HulbNet architecture is illustrated in figure 3.4.

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

4 GENERATIVE ADVERSARIAL NETWORKS

A generative adversarial network [11] (GAN) is a pair 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 GAN is that the loss function of the generator is learned; it does not need to be defined so the trained generator can reach performance exceeding that of a generator trained with a conventional loss. The conventional loss function is a limiting factor because it cannot quantify realism, especially in the case of a one-to-many mapping between a noisy image and the acceptable generated counterparts. The generator can learn to restore high-frequency details which cannot be determined from the noisy observation alone whereas the best score it can obtain with a conventional loss function is by averaging the many possible solutions or overfitting the training data.

Moreover, data may sometimes be used for training without a matching pair of ground-truth - corrupted representations. This can be very useful when paired data is not available, or to further train a network in order to learn specific tasks. It is not always possible to train a GAN without paired data. This is the case with cGANs which always take the noisy observation as the conditional input in addition to either the generated or ground-truth data, although a conditional generative adversarial network [22] (cGAN) may be trained with the noisy portion of the data, or with an adapted training protocol (such as blurring the noisy representation or using a label instead of an image). Regardless, the generator can be trained with a pre-trained discriminator.

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. A GAN is a system that comprises both a discriminative and a generative network.

4.1 TYPES OF GANS

- The GAN is the simplest and initial approach introduced by Goodfellow et al. [11]. A generator is given a noisy 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 realistic features required to trick the discriminator but does not resemble the original 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). GANs 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 [16]. This strategy keeps running statistics and normalizes the activations on the given batch, enabling the GAN to potentially perform better given low-variance input data. [28]

- Conditional generative adversarial networks are GANs which take two inputs: the generated data and typically a label that describes it [22]. 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 [17][36][29]. 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 [36] 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 the commonly used deep convolutional generative adversarial network [25] (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 recent image-to-image translation methods such as pix2pix [17] and CycleGAN [36].

The GAN's log-based BCE loss suffers from the vanishing gradient problem, where the discriminator's gradient reaches zero, thus the

generator no longer receives any information to learn. The Least Squares Generative Adversarial Network [21] (LSGAN) mitigates this issue using the mean squared error loss function which penalizes samples based on their distance from the decision boundary (e.g. a 0.5 uncertainty gives out a loss of $0.5^2 = 0.25$ while a mistaken 0.9 probability has a loss of $0.9^2 = 0.81$) and provides more stable training.

4.2 CHOSEN GAN

Our generator architectures are the U-Net [26] and Hulbnet mentioned in chapter 3. For the discriminator classifier we have experienced with PatchGAN [17] 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 $\ell 1$ loss. The network architecture is made up of five convolutions with 4×4 kernels 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 pixel receptive field and it is up to the chosen MSE loss function to reconcile these losses into a single value.

The HulDisc discriminator uses an architecture similar to the HulbNet generator, but the network is cut in half "vertically" such that it ends with a single feature instead of upsampling back to the original image size with transposed convolutions. Another difference is the use of batch normalization [16]. Our main discriminator focuses on 112×112 pixel crops, this allows for an 8-pixels border not to be discriminated against. 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 by PReLU and batch normalization [16]. This architecture was developed through experimentation with different combinations, namely the use of batch normalization, different (or no) activation functions, the use of a final pooling layer in place of a stride convolution, the effect of mixed losses, the use of a bottleneck 1×1 convolution layer, and different numbers of filters.

4.3 TRAINING METHODS

Our (c)GAN training procedures went through multiple experimental iterations.

Our initial method is summarized in section 4.3.1. Our final method, which appears to provide more stable training requiring fewer parameters, is described in section 4.3.2.

4.3.1 *Initial training method*

Our initial (c)GAN training method is similar to that of Isola et al. [17] 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 $\ell 1$ 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 $\ell 1$ loss (0.05) which provides some smoothness to prevent excessive high-frequency details caused by the discriminator. Switching loss functions multiple times throughout training did not seem to yield viable results.

Balanced initialization is crucial as our 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 (the resulting generated images are uniform). However, starting off with a well-trained generator usually results in a discriminator that never learns to discriminate and 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 by Salimans et al. [28][10] by substracting 0 to 0.05 from the real images' target probabilities 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 close to 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 that 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 adversary's most recent update.

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 by taking its square root for each set of labels, and we 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 were removed but we introduced a variable that 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

```
for all clean_batch, noisy_batch ∈ Dataset do
  generated_batch ← generator(noisy_batch) {Denoised}
  {Train the discriminator}
  if discriminator learns then
    {discriminator learns = discriminator's previous normalized
    loss > random[0,1]}
    ADAM_optimizer_D.clear_gradient()
    if discriminator.is_conditional then
      fake_minibatch ← concatenate(
        remove_borders(noisy_batch),
        remove_borders(generated_batch))
      real_minibatch ← concatenate(
        remove_borders(noisy_batch),
        remove_borders(clean_batch))
    else
      fake_minibatch ← remove_borders(generated_batch)
      real_minibatch ← remove_borders(clean_batch)
    end if
    predictions_on_real_data ← discriminator(real_minibatch)
    loss_real_D ← MSE(
      predictions_on_real_data,
      generate_noisy_probabilities(true))
    loss_real_D.backpropagate()
    predictions_on_fake_data ←
      discriminator(fake_minibatch.detach_from_graph())
    loss_fake_D ← MSE(predictions_on_fake_data, false)
    loss_fake_D.backpropagate()
    ADAM_optimizer_D.update_weights()
  end if
  {Train the generator}
  if generator learns then
    {generator learns = discriminator didn't learn or discrimina-
    tor's last normalized loss < random[0,1]}
    predictions_on_fake_data ← discriminator(fake_minibatch)
    loss_G_GAN ← MSE(predictions_on_fake_data, true)
    loss_G_SSIM ← SSIM(
      remove_borders(generated_batch),
      remove_borders(clean_batch))
    loss_G_ℓ1 ← ℓ1(
      remove_borders(generated_batch),
      remove_borders(clean_batch))
    loss_G_weighted ←
      loss_G_GAN ×(1− weight_loss_SSIM − weight_loss_ℓ1)
      + loss_G_SSIM × weight_loss_SSIM
      + loss_G_ℓ1× weight_loss_ℓ1
    loss_G_weighted.backpropagate()
    ADAM_optimizer_G.update_weights()
    ADAM_optimizer_G.clear_gradient()
  end if
end for
```

5 RESULTS

Chapter 5 describes our use of CNNs to denoise images from the NIND. The first section describes our suggested methods to handle the NIND dataset. Section 5.2 is focused on denoising with a single generator and section 5.3 uses a trained discriminative network as part of the loss function.

5.1 DATASET USAGE

The NIND is cropped in advance to speed up loading times. A crop size of 128×128 pixels was found to work well for training and larger crops did not noticeably affect denoising performance (yet they result in more memory being used up and training time is increased as a result). We found that the border is most often corrupted in a U-Net model. Some network architectures (such as DnCNN [32] and RED-Net) perform better when required to learn to model entire crops down to their border, but it typically takes a lot more training 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 75 % the size of the actual crop size for U-Net models and approximately 87.5 % 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. 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

We initially trained a DnCNN [32] 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 5×5 instead of 3×3. The second architecture tested was a RED-Net [20] with 22 layers and a filter size of 5×5. This model obtained very good performance, albeit with an impractical runtime and memory use (). We settled on a U-Net [26] architecture which provides slightly better performance with significantly lower runtime and memory use. The U-Net model denoises an

image in approximately 0.73 seconds per megapixel (MP) whereas the RED-Net with 5×5 filters takes over 12 seconds per MP on an NVidia GeForce GTX 1070 graphics processing unit (GPU).

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 SIDD (320 provided image pairs):
   Compare the performance obtained using our dataset with 320 images from the SIDD (Smartphone Image Denoising Dataset) [1]. These images were made available for the 2019 NTIRE denoising challenge and they provide some indicators as to the potential for generalization across different types of sensors.
3. BM3D [9] ([13] 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 also included the following experiments:

4. U-Net trained on NIND (dataset composed of the union of Fujifilm X-T1 mirrorless camera (X-T1) and C500D training scenes; 89.5 % and 10.5 %, respectively):
   When tested on a C500D image, this method can be compared with Method 1 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 (generated by much smaller sensors) 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]:
   This experiment compares the performance obtained by a model

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.

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 [32] by training a model that reconstructs the noise and subtracts it from the image.
9. RED-Net [20] trained on NIND (X-T1 subset):
   This uses the same data as Method 1 with a different network architecture

Each network is trained for 48 hours on a GeForce GTX 1080 (11GB) GPU. 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. Figure 5.2 shows a visual denoising comparison on test image "stefantiek". 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.

*In tables 5.1, 5.2, and 5.3, the best scores within two significant digits are marked in **bold**.*

Table 5.1: Average SSIM index on 5 NIND:X-T1 denoised scenes (ursulines-building, stefantiek, CourtineDeVillersDebris, MuseeL-Bobo, ursulines-red).

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
1 NIND:X-T1 (U-Net)	0.949	**0.929**	**0.920**	**0.912**	**0.900**	**0.893**	**0.851**
2 SIDD (U-Net)	0.906	0.907	0.904	0.864	0.882	0.860	0.814
3 BM3D	0.941	**0.925**	0.913	0.875	0.870	0.852	0.785
4 NIND:X-T1+C500D (U-Net)	0.949	**0.929**	**0.920**	**0.912**	**0.899**	**0.893**	**0.851**
5 NIND:X-T1+C500D + SIDD (U-Net)	0.947	**0.928**	**0.919**	**0.910**	**0.898**	**0.892**	**0.850**
6 NIND:X-T1 ISO6400 only (U-Net)	0.919	0.915	0.911	**0.907**	**0.901**	**0.894**	0.821
7 Artificial noise on NIND:X-T1 (U-Net)	0.963	0.920	0.899	0.880	0.810	0.769	0.531
8 Reconstruct noise on NIND:X-T1 (U-Net)	0.950	**0.926**	0.914	0.901	0.876	0.840	0.664
9 NIND:X-T1 (RED-Net)	0.940	0.923	0.915	**0.907**	0.892	**0.886**	0.842

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.

Table 5.2: SSIM index on NIND:C500D denoised set MuseeL-Bobo-C500D

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
1 NIND:X-T1 (U-Net)	0.911	**0.901**	**0.898**	**0.896**	**0.893**	**0.887**	**0.868**
2 SIDD (U-Net)	0.894	0.892	0.890	0.889	0.885	0.876	0.850
3 BM3D	0.921	0.890	0.884	0.877	0.871	0.860	0.813
4 NIND:X-T1+C500D (U-Net)	0.911	**0.901**	**0.899**	**0.896**	**0.894**	**0.888**	**0.872**
5 NIND:X-T1+C500D + SIDD (U-Net)	0.912	**0.901**	**0.899**	**0.896**	**0.893**	**0.888**	**0.871**
6 NIND:X-T1 ISO6400 only (U-Net)	0.899	**0.897**	**0.896**	0.894	**0.892**	**0.886**	**0.865**
7 Artificial noise on NIND:X-T1 (U-Net)	0.946	0.879	0.864	0.836	0.802	0.716	0.430
8 Reconstruct noise on NIND:X-T1 (U-Net)	0.908	0.895	0.887	0.875	0.855	0.807	0.617

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

ISO value	ISO100	ISO200	ISO400	ISO800	ISO1600	ISO3200	High ISO
Number of images	13	9	9	9	8	10	9
Noisy	**0.954**	0.766	0.707	0.619	0.501	0.380	0.220
1 NIND:X-T1 (U-Net)	0.878	**0.856**	**0.854**	**0.848**	**0.845**	**0.834**	**0.805**
2 SIDD (U-Net)	0.851	0.838	0.837	0.834	0.830	0.813	0.774
3 BM3D	0.899	0.836	0.825	0.816	0.811	0.789	0.749
6 NIND:X-T1 ISO6400 only (U-Net)	0.857	0.847	**0.847**	0.845	**0.843**	**0.834**	0.802
8 Reconstruct noise on NIND:X-T1 (U-Net)	0.878	0.845	0.835	0.814	0.785	0.725	0.604

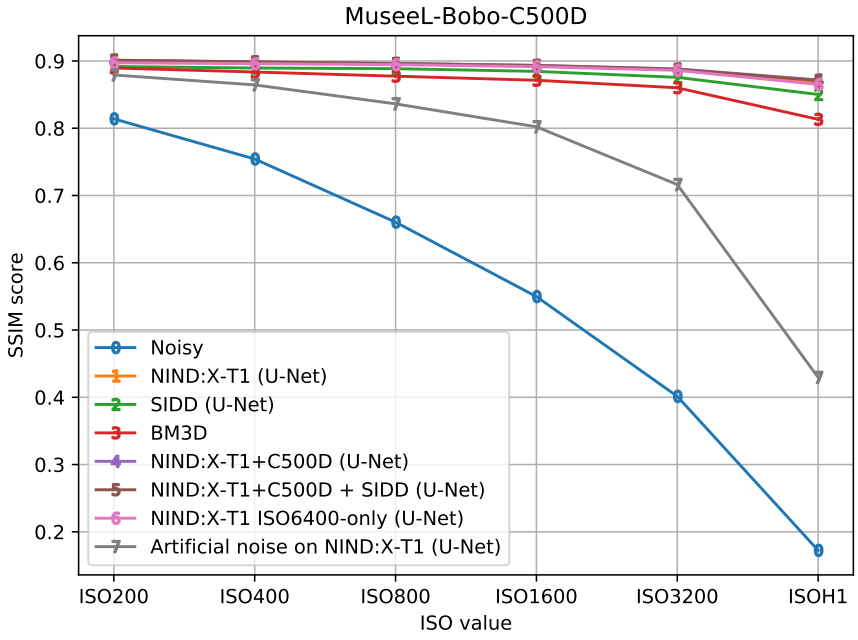

Figure 5.1: Denoising performance of different methods on the MuseeL-Bobo-C500D set over increased ISO value (SSIM values shown in Table 5.2)

Figure 5.2: Denoising stefantiek. 1: ISO200 ground-truth (1/20 s), 2: high ISO (1/1900 s), 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])

Reconstructing the noise (Method 8) as was suggested in [32] 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. Research into this topic [19] 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. These findings suggest that a model can easily tolerate some noise in the ground-truth data, although we found the networks did not learn as well when given only noisy ground-truth data.

Training a model using artificial noise added to ground-truth images (Method 7), as is commonplace in the literature [20][32], yields the worst performance in our tests.

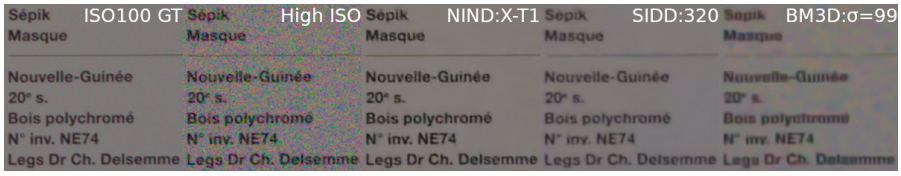

Figure 5.3: Denoising text present on MuseeL-Sepik-C500D. 1: ISO200 ground-truth (2 s), 2: high ISO (1/30 s), 3: 2 denoised using U-Net model trained with NIND:X-T1, 4: 2 denoised using U-Net model trained with SIDD (320-sets), 5: 2 denoised using BM3D ($\sigma = 99$[1])

Figure 5.3 shows satisfying denoising performance from a model trained on X-T1 images with text recovered from a C500D image. 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

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.

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 [8] 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 initially trained denoising networks using the pix2pix approach (the typical use-case is generating photorealistic images from rough sketches) of performing image to image translation using a PatchGAN discriminator and a U-Net generator [17]. The results were not satisfying even after applying the adjustments mentioned in section 4.3.1; the PatchGAN would consistently prefer high frequencies and the denoising performance suffered. We thus designed our own HulDisc discriminator and a similar HulbNet generator and tested their individual building blocks in section 5.3.2.1 before assessing the system as a whole and comparing it to previous approaches in section 5.3.2.2.

5.3.1 *PatchGAN*

We first tried following the pix2pix approach using a PatchGAN discriminator and a U-Net generator [17] with 1:3 training ratio. The discriminator handles high-frequency details and low-frequencies are handled by the $\ell 1$ 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 tended to be exaggerated in low-frequency areas and high frequency details were not properly recovered, often resulting in an overall poor SSIM index. Figure 5.5 illustrates the denoising performance obtained using a 0.25 $\ell 1$ multiplier.

Figure 5.5: Comparison of standard pix2pix [17] applied to image denoising with other methods on a crop from image NIND_MuseeL-Bobo_ISOH2. 1: ISO200 ground-truth crop (SSIM: 1.00), 2: ISOH2 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 $\text{loss}_{\text{G}} = 0.5\text{loss}_{\text{D}} + 0.4\text{loss}_{\text{SSIM}} + 0.1\text{loss}_{\ell 1}$ (SSIM: 0.794),6: Denoised with BM3D using $\sigma = 99$ (SSIM: 0.694)

Some improvements were made through careful fine-tuning of the learning environment. We added the SSIM loss used in single-network training and tested different balances between the $\ell 1$, SSIM, and discriminative losses. We tried training the generator without applying a discriminative loss until a SSIM threshold was 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 an SSIM loss of 0.125 to 0.15. The greatest performance shown occurs when the generator has to obtain $\text{loss}_{\text{SSIM}} < 0.125$ 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 a standard pix2pix cGAN (4) in Figure 5.5.

5.3.2 *Our proposed architecture*

5.3.2.1 *Components*

Our main motivation for designing the "Hul" architectures described in sections 3.2.4 and 4.2 is the poor performance obtained with a PatchGAN discriminator; the discriminative network should not focus

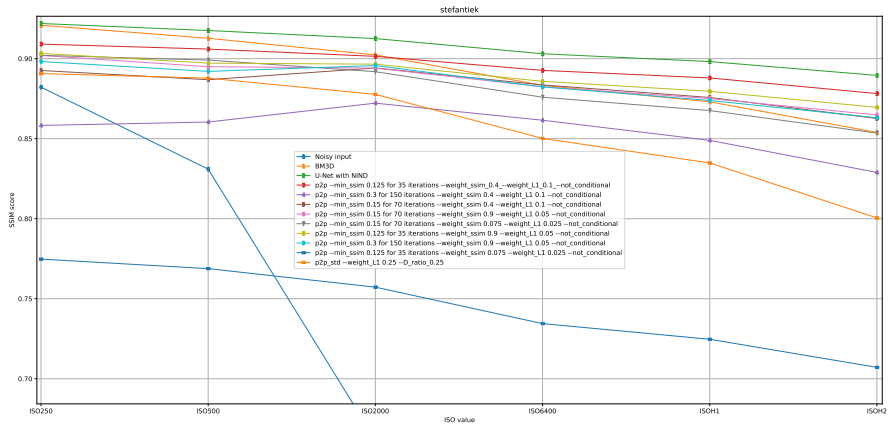

Figure 5.6: Denoising performance of the non conditional GAN with varying weights and schedules on the "stefantiek" set over increased ISO values. The models appear to perform better when they are pre-trained to achieve good performance with the SSIM loss before taking the discriminator's feedback into account and when the discriminator's weight (1 - other weights) does not account for the bulk of the loss function.

only on high-frequencies or its feedback becomes corruptive. We also wanted to replace the U-Net generative network because it loses too much borders, those borders are either easily discriminated against or they do 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 thus 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 in section 5.2, 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.

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 sensitive to color accuracy). We trained a generative network for an epoch with and without batch normalization to finally determine whether batch normalization [16] (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 (we add the letter b such that "HulbNet" denotes a network without BN). We also trained a network for an epoch to determine whether not using a final loss

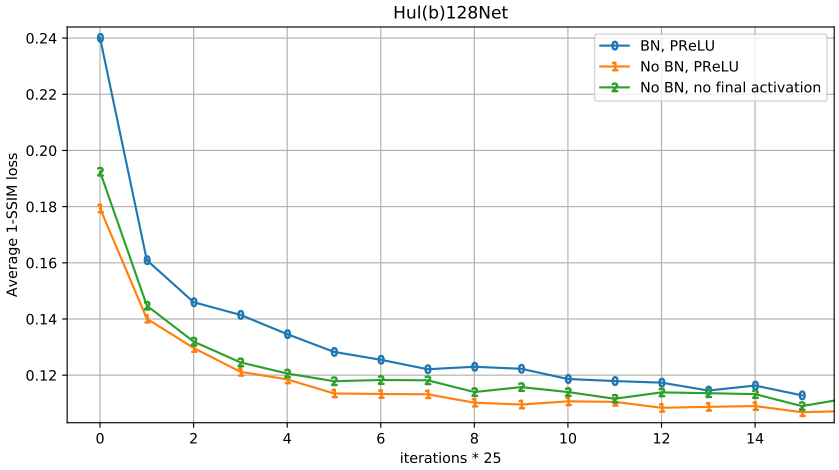

Figure 5.7: Generative network learning curve with and without batch normalization ("HulbNet"), as well as with neither batch normalization nor an activation function. We observe better performance when batch normalization [16] is not applied and with the use of a non-linear final activation function.

function as is sometimes done [32] would yield better performance than PReLU, but we found the network performs better with a non-linear activation function.

Figure 5.8 shows the discriminator's learning curve classifying images as either clean or noisy. We observe that batch normalization [16] is useful in the case of the discriminator, likely because its job does not entail recreating image fidelity from the input but output a single probability, and batch normalization [16] can help when given mini-batches of a single target value (lines 3, 6, 7). We observe that the PReLU activation function sporadically yields very poor performance on lines 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.

Following some incertitude on whether label smoothing should only be applied on positive labels as per [28] 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

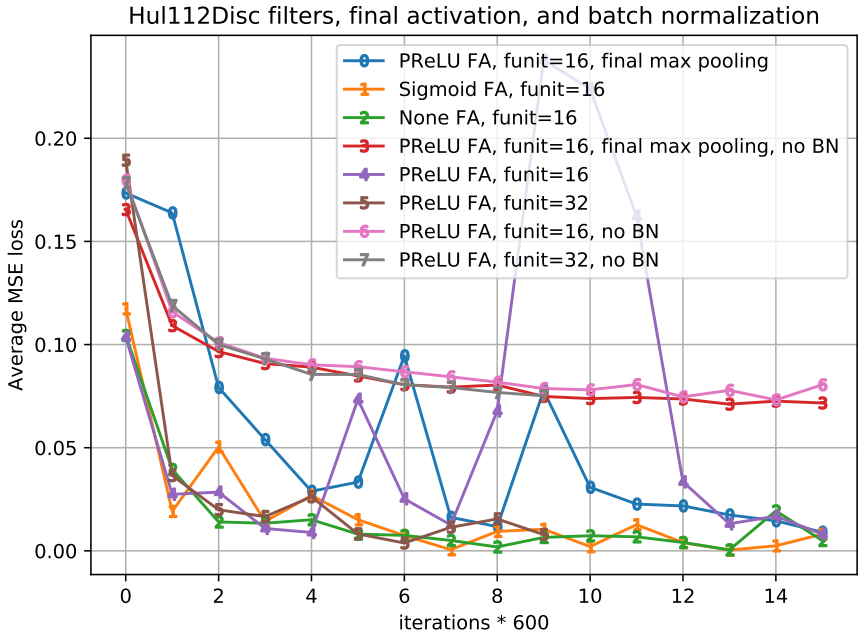

Figure 5.8: Discriminative network learning curve with varying final activation functions (FA) and number of filters. We observe that the discriminative network benefits greatly from BN and that training stability and performance suffer from not having a sufficient number of filters. One funit denotes 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.

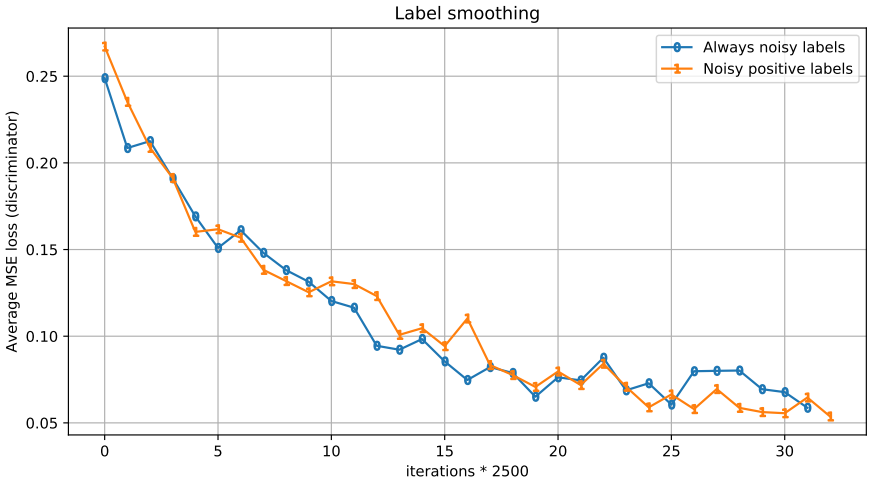

Figure 5.9: Discriminative network performance (MSE loss) with noisy labels (either always noisy or noisy for positive labels only). The network attempts to discriminate images denoised by a generator which has been trained for three epochs and is continuously being updated (alas without any input from the discriminator). We did not observe a notable difference in performance between the two label smoothing methods.

(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.

5.3.2.2 *System as a whole*

In this section we test different configurations of our "Hul" cGAN networks as a whole system trained according to our trained method described in section 4.3.2. The performance is compared to the previously used configurations as well (U-Net alone and the PatchGAN), and we assess the visual quality of denoised images where it was previously lacking (i.e. noisy face images).

The different types of network present in this experiment are as follow:

- Hul is the base name of the network we use
- b: no batch normalization
- f: more channels as the other dimensions decrease
- A number representing the minimum valid input size
- Disc for discriminators or Net for generators

The networks' training performance is shown on Figure 5.10. The baseline is a single generative U-Net network, and we also trained a

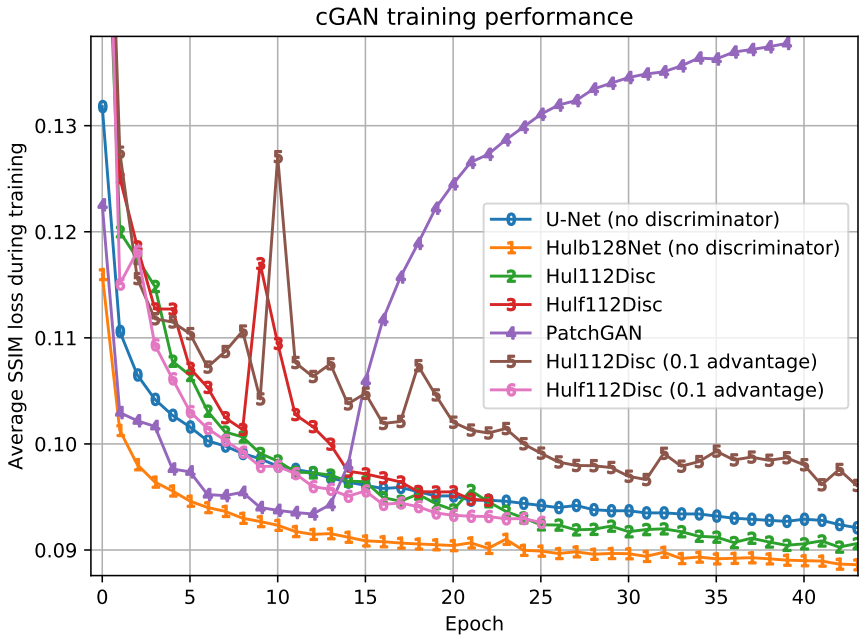

Figure 5.10: cGAN training performance comparison. 0: The U-Net model from experiment 5.2:1, 1: Hulb128Net (without a discriminator). The following networks all use the Hulb128Net generator and a specified discriminator. 2: Hul112Disc, 3: Hulf112Disc (more filters as the network shrinks), 4: PatchGAN [17], 5: Hul112Disc which learns 10 % more often than the generator, 6: Hulf112Disc which learns 10 % more often than the generator. Our HulbNet generator appears to outperform U-Net both as a single network and in a GAN. Although the cGAN system appears to perform better, that is expected because the main loss is no longer based on a correlation with the ground-truth image, and visual analysis of the results later confirm that the cGAN system offers more visually pleasing denoising. Adding more filters to the discriminator appears to only work if the discriminator has a higher training ratio to optimize them. Likewise, a discriminator that is trained more often does not appear to offer a better feedback if its capacity is not high enough.

single Hulb128Net network. The single Hulb128Net network appears to perform better than the U-Net of our previous experiments. We observe that the increased number of filters (Hulf112Disc) destabilizes the discriminator, but performance is restored when the discriminator has more time to learn. On the other hand, giving more time to a discriminator (0.1 advantage) that does not have as much capacity provides worse feedback and the result of the generator suffers. The PatchGAN's performance seems to degenerate quickly, and the Hulb128Net alone offers greater performance than that of a cGAN pair but further analysis shows that the cGAN denoised images are visually superior. The networks that were trained for longer benefit greatly from the additional time and exceed the performance of a single U-Net generator which maxes out at SSIM index = 0.902. The SSIM training loss is shown. Although SSIM is not a meaningful enough factor on its own because the network could recreate a structurally different yet visually pleasing image, it provides an approximate indicator of performance and learning.

All cGAN networks use a $\text{loss}_{\ell 1}$ weight of 0.05 and a $\text{loss}_{\text{SSIM}}$ weight of 0.2. While it may be worth trying different ratios, we found that the networks quickly became unstable as the weight of the discriminative loss increased; a $\text{loss}_{\text{discriminative}}$ weight of 0.95 would typically not complete the first epoch.

Figure 5.11 visually shows the denoising performance of various cGAN and conventional networks. We only displayed one combination of "Hul" generator-discriminator pair because we could not discern an appreciable difference between any of our trained cGAN models. We observe restored high frequencies in the cGAN network, whereas the single generator does not provide as much detail. The PatchGAN offers a very sharp image but much of the generated image looks unnatural (and the bricks in less chaotic parts of the image have an asymmetrical structure). The SSIM index does not correlate with a visually appealing result because it favors blur over details that are different from those present on the ground-truth (one-to-many mapping). This type of image benefits greatly from a discriminative loss because of its many high-frequency details.

Face images are difficult to generate satisfyingly because we are most sensitive to structural imperfections, oversmoothing, and imbalances between sharpness and smoothing. We did observe a difference between the different training methods but none of them would provide satisfying results; the "Hul" networks would result in unnaturally smooth looking skin (the discriminator providing marginal sharpness), while HulbNet combined with PatchGAN could provide sharp but unnatural results. We eventually obtained satisfying results on human skin training a generative model with two different discriminators; the generator's weighted loss is comprised of 0.05 $\ell 1$, 0.25 SSIM, 0.25 PatchGAN discriminator, and 0.5 HulfDisc discriminator (which is

Figure 5.11: Visual comparison of different networks' denoising performance on NIND_CourtineDeVillersDebris_ISOH2. 1: Ground-truth (ISO200 1/4 s, SSIM: 1.000), 2: Noisy (High ISO 1/1100 s, SSIM: 0.232), 3: HulbNet generator with no discriminator (SSIM: 0.784), 4: HulbNet generator and HulDisc discriminator (SSIM: 0.780), 5: HulbNet generator and PatchGAN discriminator (SSIM: 0.687).

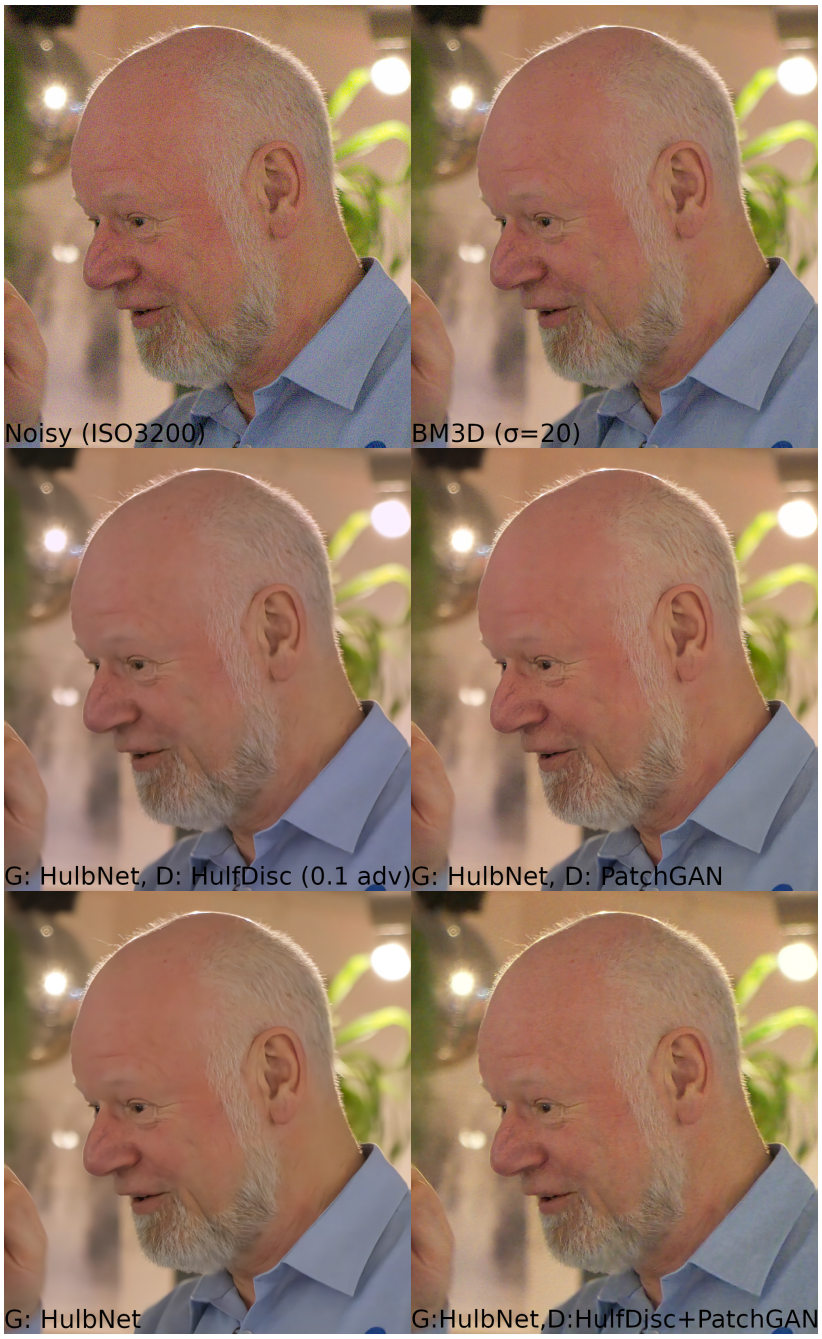

Figure 5.12: Visual comparison of different networks' denoising performance on a human face. top-left: Noisy ISO3200 image, top-right: denoised with BM3D ($\sigma = 20$), middle-left: denoised with a HulbNet generator and HulfDisc discriminator (with a 0.1 advantage), middle-right: denoised with a HulbNet generator and a PatchGAN discriminator, bottom-left: denoised with a HulbNet generator, bottom-right: denoised with a HulbNet generator and both a HulfDisc (0.5 weight) and PatchGAN (0.25 weight) discriminator. The skin appears too smooth on the HulbNet network alone and the HulfDisc discriminator offers marginal improvements in sharpness. The PatchGAN discriminator provides sharper but unnatural looking results (particularly on the nose and the hair). The 3-networks combination generates more natural-looking skin textures without domain-specific training, although the overall image quality is not as great.

given a 0.1 advantage). A sample resulting face is shown on Figure 5.12.

6 CONCLUSION

We investigated the use of deep learning to solve the denoising problems on photographic images and found that convolutional neural networks outperform conventional denoising methods; state-of-the-art denoising performance is achieved with the use of quality training data. Most research involves training and testing models by adding artificial noise to ground-truth data and this method results in poor performance, as Plötz and Roth [24] and our own experiments (5.2:7) have shown. No such complete dataset were available prior to our research; therefore, we created and released the Natural Image Noise Dataset [5][4] (NIND), a dataset of photographic ISO noise with scenes captured using multiple ISO values (and matching settings) which can be used to train a blind denoising model. We found the performance obtained with a single convolutional neural network [33] trained on the NIND to be excellent on a wide variety of scenes, even dynamic ones which could not be captured in the dataset.

A few images remained somewhat unsatisfactory after denoising with a single U-Net model trained on the NIND, primarily those of human faces which could look unnaturally smooth in the absence of noise. The excessive smoothing is normally caused by the conventional loss function which favors a blurry result because there is a one-to-many mapping between a noisy image and its possible denoised representations. We developed a framework to train conditional generative adversarial networks for denoising as a way to alleviate this excessive smoothing. cGAN are expected to produce more realistic results and generate high-frequencies because the discriminator (which is used as a loss function) is trained to recognize generated images, rather than an exact match with the ground-truth. We found that the resulting denoised images would sometimes provide more natural looking features on human subjects, but that was not always the case. Further work showed that using a combination of multiple different discriminative networks results in a generator that creates believable skin textures even without domain-specific training. The (c)GAN training process is much finicky than that of a conventional convolutional neural network and the results are very similar in most cases, as such we do not consider cGANs to be a necessary update over conventional CNNs for general image denoising, but they provide an appreciable improvement in some edge cases. A great benefit of cGANs is that they can be used to train the generator without paired data (given a trained discriminator). In some cases the discriminator can be trained without paired data as well, as is the case of unconditional

GANs and possibly specially designed cGANs. One such design could be by to feed the discriminator with a variable-noise blurred version of the image as its noisy observation. This process could be used to better handle faces or any other problematic type of images that cannot be included in a paired dataset.

Further work includes finding and working with a type of images that is not well handled by the paired dataset, taking advantage of the fact that GANs can work with unpaired data. Different types of GAN loss functions can also be investigated. We have investigated the use of LSGANs (MSE) and DCGANs (BCE) in our work, but other types such as the Wasserstein GAN [3] (WGAN) may provide greater performance and stability. The "Hul" generative and discriminative architecture we introduce perform well but inference remains more computationally intensive than with the U-Net architecture; more assessments should be performed to determine which building blocks can be trimmed without negatively affecting performance. Likewise more tests should be performed to determine the ideal weights for each loss function in a GAN system, and it can be worth training a cGAN model until convergence and examining the result. Finally we would like to implement a light CNN denoising model such as U-Net within an open-source photography development software in order to transition deep learning denoising from a purely scientific research field to widespread application.

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.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein Generative Adversarial Networks." In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, 2017, pp. 214–223. URL: http://proceedings.mlr.press/v70/arjovsky17a.html.

[4] Benoit Brummer. *Natural Image Noise Dataset - Wikimedia Commons*. https://commons.wikimedia.org/wiki/Natural_Image_Noise_Dataset.

[5] Benoit Brummer and Christophe De Vleeschouwer. "Natural Image Noise Dataset." In: *2019 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, arXiv:1906.00270.

[6] 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.

[7] 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.

[8] *Commons:Quality images candidates - Wikimedia Commons*. https://commons.wikimedia.org/wiki/Commons:Quality_images_candidates.

[9] 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.

[10] Ian J. Goodfellow. "NIPS 2016 Tutorial: Generative Adversarial Networks." In: *CoRR* abs/1701.00160 (2017). arXiv: 1701.00160. URL: http://arxiv.org/abs/1701.00160.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] *Hugin - Panorama photo stitcher*. http://hugin.sourceforge.net/.

[16] 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. URL: http://arxiv.org/abs/1502.03167.

[17] 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. URL: https://doi.org/10.1109/CVPR.2017.632.

[18] 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.

[19] 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. Stockholmsmässan, Stockholm Sweden: PMLR, 2018, pp. 2965–2974. URL: http://proceedings.mlr.press/v80/lehtinen18a.html.

[20] 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.

[21] 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).

[22] 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.

[23] 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.

[24] Tobias Plotz and Stefan Roth. "Benchmarking Denoising Algorithms With Real Photographs." In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[25] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks." In: *ICLR*. 2016.

[26] 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.

[27] Dr. Sachin Ruikar and Dharmpal Doye. "Image Denoising Using Wavelet Transform." In: Sept. 2010, pp. 509–515. ISBN: 978-1-4244-8100-2. DOI: 10.1109/ICMET.2010.5598411.

[28] 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.

[29] 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.

[30] 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.

[31] Fisher Yu and Vladlen Koltun. "Multi-Scale Context Aggregation by Dilated Convolutions." In: *ICLR*. 2016.

[32] 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.

[33] 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. URL: http://ao.osa.org/abstract.cfm?URI=ao-29-32-4790.

[34] 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. URL: http://arxiv.org/abs/1812.10366.

[35] 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.

[36] 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.

[37] *darktable*. https://www.darktable.org/.

[38] *profiling sensor and photon noise | darktable*. https://www.darktable.org/2012/12/profiling-sensor-and-photon-noise/. Dec. 2012.